ÉCOLE NATIONALE DES CHARTES
UNIVERSITÉ PARIS, SCIENCES & LETTRES

Kelly Christensen
licenciée ès lettres
diplômée de master en musicologie
diplômée de doctorat musicologie

D'ALTO à TEI

Modélisation de transcriptions automatiques pour une pré-éditorialisant des textes

Mémoire pour le diplôme de master
« Technologies numériques appliquées à l'histoire »

2022

Résumé

Quand des modèles OCR et HTR extraient les données d'une ressource textuelle numérisée, les informations relatives à la structure physique de l'image risquent de se perdre. Un schéma XML standardisé qui s'appelle ALTO a été créé afin de conserver et structurer ces données non-textuelles et géometriques en les tenant en relation avec le contenu textuel. La plupart des modèles OCR et HTR compte sur ce schéma. Cependant ALTO ne convient pas bien à l'édition numérique ni aux traitements automatique du langage. Les éditeurs et les chercheurs en lettres attendent un schéma XML plus courant dans le monde des humanités numériques : la TEI. Il faut donc un mapping pour transformer un fichier ALTO en fichier TEI sans perdre aucune donnée lors du processus. Cette transformation automatisée permet à conserver les données particulières au schéma ALTO, telles que celles sur la segmentation et sur la structure physique du document numérisé, ainsi qu'à exploiter le contenu textuel de la ressource textuelle. La flexibilité de la TEI et son usage très répandu rendent le schéma idéal pour mieux valoriser les données produites par les modèles OCR et HTR.

Dans le cadre du stage pour obtenir le diplôme de Master 2 « Technologies numériques appliquées à l'histoire », ce mémoire porte sur la modélisation de la transformation de ALTO en TEI. Cette modélisation a été réalisée dans le cadre du projet *Gallic(orpor)a*, financé par la BnF lors d'un stage qui a eu lieu au sein du laboratoire Automatic Language Modelling and Analysis & Computational Humanities entre avril et juillet 2022.

Mots-clés : HTR, OCR, ALTO, TEI, TAL, édition numérique.

Informations bibliographiques : Kelly Christensen, *D'ALTO à TEI, modélisation de transcriptions automatiques pour une pré-éditorialisant des textes*, mémoire de master « Technologies numériques appliquées à l'histoire », dir. Ségolène Albouy, École nationale des chartes, 2022.

Remerciements

Mes remerciements vont tout d'abord à l'École nationale des chartes et particulièrement le responsable du master Technologies numériques appliquées à l'histoire Thibault Clérice, qui m'a soutenu depuis mon intégration jusqu'à la fin et m'a donné des bases techniques fondamentales au projet du stage et au mon travail après le master.

Je me tiens également à remercier mes tuteurs du stage, Simon Gabay et Ariane Pinche, pour m'avoir fait autant de confiance et m'accompagné dans la publication et la présentation de notre travail. Je remercie aussi ma directrice du mémoire Ségolène Albouy. L'élaboration du mémoire a bénéficié de ses conseils ainsi que les recommandations de Mme. Pinche, M. Gabay, Alix Chagué, et Lucille Delbecchi.

Je pense aussi à Benoît Sagot, responsable de l'équipe Automatic Language Modelling and Analysis & Computational Humanities chez Inria, qui m'a chaleureusement accueillie au sein de l'équipe. Là j'ai eu l'occasion de travailler à côté de compétents chercheurs qui ont partagé leur expertise. Je remercie notamment Rachel Bawden, Mme. Chagué, Hugo Scheithauer, Florianne Chiffoleau, Wisam Antoun, et mon co stagiaire Jules Nuguet.

Je n'oublie pas non plus mes collègues de la promotion TNAH 2022, mes proches, et mon conjoint qui m'ont sans cesse soutenue dans l'élaboration de mon projet et la rédaction de ce mémoire de stage.

Introduction

Plus en plus d'institutions patrimoniales cherchent à numériser leurs ressources textuelles dans le but de démocratiser la recherche [1]. Cet objectif s'est ressenti fortement lors de la pandémie de Covid-19, quand des archives autour du monde ont fermé leurs portes physiques. Les portails des bases de données, tel que Gallica de la Bibliothèque nationale de France, sont fondamentaux pour la recherche. Mais l'extraction du texte des ressources numérisées n'est plus l'enjeu le plus important [2]. Il est tellement facile actuellement de numériser la page d'un document et d'extraire du texte brut et repérable qu'un individu possédant un portable avec un appareil photo et une application OCR (*Optical Character Recognition*) peut le faire. Les interfaces graphiques et libres, telle qu'*eScriptorium* [3], ainsi que les modèles OCR et HTR (*Handwritten Text Recognition*) publiés librement en ligne [4] ont révolutionné la recherche ainsi que l'archivage et la conservation du patrimoine. Le monde dispose désormais d'un nombre croissant de documents numérisés.

Le nouveau défi à relever aujourd'hui est de transformer ces numérisations en des ressources enrichies, qui augmentent le texte extrait et repérable avec de la métadonnée et de l'analyse. Le texte brut et non annoté ne suffit plus pour la recherche en informatique appliquée aux documents historiques. De là vient l'impulsion pour le projet *Gallic(orpor)a*. Le projet envisage la mise en place d'un *pipeline* qui saisit un document numérisé depuis le portail Gallica et renvoie une ressource numérique très enrichie. En plus d'une trans-

1. Depuis 2006, la Bibliothèque nationale de France s'engage à la numérisation et l'océrisation (Optical Character Recognition) en masse de ses documents pour afin qu'ils puissent être recherchés par le texte, au lieu de tout simplement la notice bibliographique, cf. Ahmed Ben Salah, Nicolas Ragot et Thierry Paquet. « Adaptive Detection of Missed Text Areas in OCR Outputs : Application to the Automatic Assessment of OCR Quality in Mass Digitization Projects ». In : Document Recognition and Retrieval XX. T. 8658. 5 fév. 2013, p. 110. DOI : 10.1117/12.2003733. URL : https://hal-bnf.archives-ouvertes.fr/hal-00820564 (visité le 12/09/2022)

2. La reconnaissance du texte à partir des manuscrits et des documents écrits dans un ancien état du français posent toujours plus de difficulté que les imprimés et que les documents en français moderne. Néanmoins, la technologie permettant l'amélioration de la reconnaissance du texte sur les manuscrits est déjà mise en place ; elle s'agit de l'entraînement des modèles supérieurs en s'appuyant sur la création d'encore meilleures données, cf. Simon Gabay, Thibault Clérice et Christian Reul. *OCR17 : Ground Truth and Models for 17th c. French Prints (and Hopefully More)*. Mai 2020. URL : https://hal.archives-ouvertes.fr/hal-02577236 (visité le 12/08/2022).

3. Dassonneville Gautier et al. *Compte-Rendu de La Journée d'étude " Point HTR 2022 "Transkribus / eScriptorium : Transcrire, Annoter et Éditernumériquement Des Documents d'archives*. Research Report. CAPHES - UMS 3610 CNRS/ENS ; AOROC, juin 2022. URL : https://hal.archives-ouvertes.fr/hal-03692413.

4. *Models - Hugging Face*. URL : https://huggingface.co/models (visité le 12/09/2022).

cription du texte repérable, la ressource présentera les données structurelles portant sur la mise en page, ainsi qu'une analyse linguistique du texte extrait et des métadonnées portant sur le document physique et le fac-similé numérique.

Ce mémoire détaille le contexte (partie I), les enjeux (partie II), et la mise en opérationnel du projet *Gallic(orpor)a* (partie III). Notre pipeline a pour but de traiter automatiquement des collections de document aussi bien en ancien-français, moyen français que français classique, issus soit de manuscrits soit d'imprimés produits entre le XV^e^ siècle et le XVIII^e^ siècle. Cependant, le but sous-jacent du projet serait de parvenir à produire un prototype qui pourrait servir d'exemple pour mettre en place des chaines d'acquisition numérique pour des collections de documents issus des institutions patrimoniales. Dans le cadre du stage, je me suis chargée da la réalisation du prototype, dont la démonstration nous avons présenté au colloque du DataLab de la BnF en juin 2022 [5].

Le pipeline du projet *Gallic(orpor)a* se déroule dans cinq étapes. Dans un premier temps, il récupère les fac-similés numériques des documents sources depuis la base de données du portail Gallica. Dans un deuxième temps, il applique des modèles HTR aux fac-similés téléchargés afin de produire une prédiction du texte et une transcription de la mise en page. Ensuite, il crée un document préliminaire qui réunit les données produites par les modèles HTR et les métadonnées récupérées de plusieurs sources en ligne qui portent sur le document source. Le quatrième étape enrichit le document avec une analyse linguistique du texte de la transcription. Et enfin, le pipeline export les données du document enrichit en divers formats pour l'exploitation de divers projets. En plus de la mise en place d'un prototype du pipeline, j'ai créé l'application `alto2tei` qui le complète en allant des fichiers sortis des modèles HTR vers une version préliminaire du document éventuellement produit par le pipeline, celui qui commence à rassembler les aspects différents.

Une ressource numérique ainsi enrichie porte un grand intérêt aux chercheurs et aux lecteurs. Par exemple, si on recherchait le mot « Candide » dans la transcription de la première édition de *Candide* par Voltaire, parmi les résultats serait la première page du premier chapitre puisqu'il répète le titre du livre, *Candide, ou l'optimisme*, à en-tête et qu'il présente le titre du chapitre, « Comment Candide fut élévé dans un beau Château, & comment il fut chaffé d'icelui » [6]. Cependant, un tel résultat ne s'agit pas d'une occurence du nom « Candide » dans le corps du livre. Si on voulait utiliser une méthode computationnelle pour analyser les références au personnage principal, le texte non hiérarchisé ne suffit pas. Notre calcul serait faux puisqu'il compterait les deux occurrences sur la première page du livre, bien que le personnage n'a pas encore été évoqué.

La ressource numérique sortie du pipeline *Gallic(orpor)a* peut bien distinguer entre

5. Kelly CHRISTENSEN, Ariane PINCHE et Simon GABAY. « Gallic(Orpor)a : Traitment Des Sources Textuelles En Diachronie Longue de Gallica ». In : *DataLab de La BnF*. Paris, France, juin 2022. URL : `https://hal.archives-ouvertes.fr/hal-03716534` (visité le 09/08/2022).

6. VOLTAIRE. *Candide, ou L'optimisme , traduit de l'allemand de M. le docteur Ralph*. S.l. : s.n., 1759. 299 p. URL : `http://gallica.bnf.fr/ark:/12148/bpt6k70445g` (visité le 12/09/2022), p. 3.

l'en-tête et le corps du chapitre. Grâce aux modèles de segmentation entraînés par l'équipe de *Gallic(orpor)a* bien qu'au lexique élaboré dans le cadre du projet *SegmOnto*, le pipeline parvient à un texte hiérarchisé dont les différentes parties du documents portent certaines étiquettes, telle que *HeadingLine* pour un titre du chapitre. Ainsi, une ressource portant sur la première édition de *Candide* ne confondrait pas le titre « Comment Candide fut élévé dans un beau Château, & comment il fut chaffé d'icelui » avec une occurrence du nom de personnage dans le corps du premier chapitre. En profitant d'une structure de données imbriquée, spécifiquement *l'eXtensible Markup Language* (XML), la ressource que notre pipeline produite imbrique le texte du document source dans des éléments portant certains noms et attributs. Grâce à l'imbrication, les chercheurs peuvent filtrer leurs recherches et aboutir à des analyses plus raffinées et importantes.

La première partie du mémoire (chap. 1, 2, 3) sert à contextualiser le projet *Gallic(orpor)a*. Le premier chapitre parle des enjeux et des projets précédents qui ont informé notre approche au traitement automatique des documents historique en diachronie longue. Les deux chapitres suivants décrivent deux aspects important au contexte du projet : le lexique pour décrire systématiquement des documents textes et le processus d'extraire du texte à partir des images numériques d'un document source. La deuxième partie du mémoire (chap. 4, 5, 6) porte sur les aspects techniques qui ont conditionné la réalisation du projet. Le quatrième chapitre décrit en détail le pipeline. Le cinquième explique les deux schémas informatiques utilisés pour encoder les données structurelles de la page d'un document texte. Le dernier chapitre de la partie intermédiaire décrit de divers métadonnées ciblées par notre pipeline. Enfin, la dernière partie (chap. 7, 8, 9) porte sur la réalisation du pipeline selon notre modélisation. Le septième chapitre explique comment mon application `alto2tei` a récupérée toutes les métadonnées ciblées depuis de divers sources de données externes. Ensuite, le huitième chapitre expose notre modélisation de la transcription dans la ressource numérique. Enfin, le neuvième chapitre décrit l'enrichissement des données ainsi récupérées dans une version de texte pré-éditorialisée et une version annotée avec l'analyse linguistique.

Première partie

Présentation du projet

Chapitre 1

Le rêve du projet *Gallic(orpor)a*

Le projet *Gallic(orpor)a* s'est développé à partir de plusieurs projets précédents et il tire parti de divers domaines de recherche. Ses créateurs, en mettant en valeur leurs propres connaissances, ont visé à assembler un pipeline qui pourrait traiter les documents issus de la base de données du portail Gallica de la Bibliothèque nationale de France (BnF). Les chercheurs spécialisés en *l'Handwritten Text Recognition* (HTR), en Traitement automatique des langues (TAL), en Histoire, en littérature, mais aussi en lexicographie et en la stylométrie se sont rassemblés pour réaliser ce projet. Le pipeline réalisé dans le cadre du projet *Gallic(orpor)a* a pour but de traiter automatiquement des collections de document aussi bien en ancien-français, moyen français que français classique, issus soit de manuscrits soit d'imprimés produits entre le XV^{e} siècle et le XVIII^{e} siècle. Cependant, le but sous-jacent du projet serait de parvenir à produire un prototype qui pourrait servir d'exemple pour mettre en place des chaines d'acquisition numérique pour des collections de documents issus des institutions patrimoniales.

Les ambitions du *Gallic(orpor)a* ont été rendues possibles grâce aux recherches de plusieurs chercheurs et ingénieurs, tels que Laurent Romary, Philippe Gambette, Thibault Clérice, Pedro Suarez Ortiz, Claire Jahan, Caroline Corbières, et Alexandre Bartz. Mais les acteurs principaux du projet *Gallic(orpor)a* lors de mon stage en 2022 étaient Jean-Baptiste Camps, Simon Gabay, et Ariane Pinche, qui ont développé des modèles HTR pour extraire du texte des documents numériques dans la base de données Gallica. Chez Inria, en tant que stagiaire, j'ai aussi travaillé en collaboration avec Benoît Sagot et Rachel Bawden, qui ont développé des outils d'analyse linguistique. Tous ensemble, ces chercheurs ayant des spécialités diverses ont mis en commun leurs connaissances pour produire un processus du traitement des documents polyvalent.

1.1 Le contexte du projet

1.1.1 Bibliothèque nationale de France et le DataLab

Le DataLab a été mis en place au sein de la Bibliothèque nationale de France (BnF) en 2021 [1]. Lors de sa première année, un premier appel à projets a été lancé pour mettre en valeur les fonds et les ressources de l'institution phare patrimoniale. Le projet *Gallic(orpor)a* faisait partie des premiers projets acceptés en 2021, à côté des projets *AUREJ* (Accès Unifié aux REssources de la Jouabilité), *GALLICAENV*, *BUZZ-F*, et *AGODA* (Analyse sémantique et Graphes relationnels pour l'Ouverture et l'étude des Débats à l'Assemblée nationale) [2]. Ayant sa candidature retenue, *Gallic(orpora)* a profité d'un financement du DataLab de la BnF. La plupart du travail sur le projet a eu lieu pendant la première moitié de l'année 2022, grâce à la mise en place de notre stage et de vacations pour la création de données d'entrainement.

1.2 Inria et l'équipe ALMAnaCH

Inria est l'Institut national de recherche en sciences et technologies du numérique et il compte plusieurs branches dans le monde. La branche parisienne accueille l'équipe ALMAnaCH dont l'acronyme signifie *Automatic Language Modelling and Analysis & Computational Humanities*. Au sein d'ALMAnaCH s'est développé le meilleur modèle TAL pour la langue française, CamemBERT [3]. L'équipe ALMAnaCH encadre des chercheurs, des ingénieurs, des doctorants, et des stagiaires attachés aux projets concernés, soit par le traitement automatique des langues, soit par les humanités numériques. L'acronyme du nom du laboratoire fait référence à ces deux pôles de recherche : *Automatic Language Modelling and Analysis* est le traitement automatique des langues, et *Computational Humanities* pour les humanités numériques. Le projet *Gallic(orpor)a* se situait entre les deux, impliquant l'extraction des données et l'édition des documents historiques, ainsi que l'analyse linguistique du texte extrait.

Benoît Sagot, directeur de recherches d'ALMAnaCH, s'est chargé de l'encadrement du stage du projet *Gallic(orpor)a*. En tant que stagiaire, je faisais partie de l'équipe de début avril et à fin juillet 2022. Pendant le stage, Rachel Bawden a animé un groupe

1. Marie Carlin et Arnaud Laborderie. « Le BnF DataLab, Un Service Aux Chercheurs En Humanités Numériques ». In : *Humanités numériques* 4 (déc. 2021). URL : `https://hal-bnf.archives-ouvertes.fr/hal-03285816` (visité le 11/08/2022).

2. Bibliothèque nationale de France. *Rapport d'activité 2021 de la Bibliothèque nationale de France*. Paris, France, 1er juill. 2022. URL : `https://www.bnf.fr/fr/bnf-rapport-dactivite-2021` (visité le 09/08/2022), p. 123.

3. Louis Martin et al. « CamemBERT : A Tasty French Language Model ». In : *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. ACL 2020. Online : Association for Computational Linguistics, juill. 2020, p. 7203-7219. DOI : `10.18653/v1/2020.acl-main.645`. URL : `https://aclanthology.org/2020.acl-main.645` (visité le 11/08/2022).

de lecture hebdomadaire et des séminaires mensuels sur les nouvelles recherches et les nouveaux enjeux en le Traitement automatique des langues. Elle a aussi développé un modèle TAL pour le projet *Gallic(orpor)a* et m'a guidée dans sa mise en œuvre [4]. J'ai travaillé en présentiel quatre jours par semaine, en passant un jour toutes les semaines à l'École nationale des chartes pour travailler à côté de l'une des chefs du projet, Ariane Pinche. Chez Inria, j'ai profité de l'expertise de mes collègues de l'équipe ALMAnaCH, en particulier d'Alix Chagué et d'Hugo Scheithauer. L'équipe entière de *Gallic(orpor)a* a aussi profité des serveurs d'Inria, qui prenaient en charge une partie de la puissance de calcul et du stockage de données pour l'interface graphique HTR *eScriptorium*.

1.2.1 École nationale des chartes et l'université de Genève

Ariane Pinche, post-doctorante à l'École nationale des chartes, et Simon Gabay, maître-assistant à l'université de Genève, ont géré la mise en place du stage et des vacations que la bourse du Data Lab de la BnF a permis de financer. Deux autres chercheurs qui étaient attachés à l'École nationale des chartes pendant le stage, Jean-Baptiste Camps et Thibault Clérice, ont également contribué au projet *Gallic(orpor)a*. S. Gabay et A. Pinche se sont occupés de l'harmonisation des vérités de terrain produites par l'équipe en relisant toutes les transcriptions que les vacataires ont faites dans l'interface graphique *eScriptorium*. A. Pinche et T. Clérice ont commencé à utiliser les vérités de terrain des documents médiévaux en entraînant de nouveaux modèles HTR et de segmentation [5]. Pour la segmentation, J.-B. Camps, A. Pinche, et S. Gabay ont développé la syntaxe *SegmOnto* qui sert à harmoniser les vérités de terrain produites du corpus d'entraînement [6]. Bien que chaque chercheur ait ses spécialités, ils ont tous collaboré et le projet a pu profiter des expertises de chacun.

1.3 La création des données d'entrainement du projet

Le projet *Gallic(orpor)a* vise à développer et mettre en place des modèles HTR pouvant traiter tout document français dans la base de données Gallica créée entre le XV^e^ siècle et la Révolution française. Un corpus de documents sources numérisés ainsi qu'une

4. Rachel Bawden et al. « Automatic Normalisation of Early Modern French ». In : LREC 2022 - 13th Language Resources and Evaluation Conference. 20 juin 2022. DOI : 10.5281/zenodo.5865428. URL : https://hal.inria.fr/hal-03540226 (visité le 11/08/2022) ; Simon Gabay. *FreEMcorpora/FreEMnorm : FreEM Norm Parallel Corpus*. Zenodo, 17 jan. 2022. DOI : 10.5281/zenodo.5865428. URL : https://zenodo.org/record/5865428 (visité le 11/08/2022).

5. Thibault Clérice. *YALTAi : Segmonto Manuscript and Early Printed Book Dataset*. 10 juill. 2022. DOI : 10.5281/zenodo.6814770. URL : https://zenodo.org/record/6814770 (visité le 12/08/2022).

6. Simon Gabay et al. « SegmOnto : Common Vocabulary and Practices for Analysing the Layout of Manuscripts (and More) ». In : *1st International Workshop on Computational Paleography (IWCP@ICDAR 2021)*. Lausanne, Switzerland, sept. 2021. URL : https://hal.archives-ouvertes.fr/hal-03336528 (visité le 09/08/2022).

sélection de leurs pages à transcrire a permis de générer des données d'entraînement. La sélection a été guidée par la volonté d'avoir une certaine diversité de documents, de genre et d'origine géographique. Comme montre la Figure 1.1, il y avait les manuscrits, les incunables, et les imprimés. De plus, chaque type de document possède des représentants issus de plusieurs genres littéraires : poésie, récit, et le traité. Dans le cas des imprimés, il y avait aussi des pièces de théâtre.

			Ecriture							
Type	Genre	Siècle	Null		antiqua		cursive		gothique	
manuscrit	Poésie	13								2
		15						3		
	Récit	13								7
		14						1		6
		15						6		
		16								1
	Traité	14								1
incunable	Poésie	15								2
	Récit	15								5
		16				1				1
	Traité	15								2
		16				1				2
imprimé	Poésie	16				3				
		17				7				
		18				4				
	Récit	16		1		2		1		2
		17		1		9				
		18				6				
	Théâtre	16				2				
		17				5				
		18				3				
	Traité	16				2				1
		17				3				
		18				12				

FIGURE 1.1 – Diversité documentaire et générique

Pour terminer, un autre souhait quant à la diversité du corpus a porté sur le lieu de publication ou d'apparition du document, comme montre la Figure 1.2. La plupart des documents choisis de la base de données Gallica étaient issus de Paris, à l'image de la collection de la Bibliothèque nationale de France. Toutefois, une partie importante est également originaire de Lyon. Enfin, ont également été inclus des manuscrits, des incunables, et des imprimés écrits en français, venus de villes telles que Londres, Amsterdam, ou encore Rome.

Après avoir établi le corpus, les pages ciblées dans chaque document ont été transcrites par des vacataires en utilisant l'interface graphique *eScriptorium*, qui permet à la fois la transcription du texte et la segmentation de la page. La segmentation de la page se fait en mettant les étiquettes précises sur les lignes et les blocs de texte ; ces étiquettes

FIGURE 1.2 – Diversité géographique

suivent le vocabulaire *SegmOnto*. Ensuite, les vérités de terrain produites avec l'interface graphique *eScriptorium* sont transférées vers les dépôts Github du projet, organisés par siècle et type de document. L'outil `HTRVX` du projet HTR-United a permis d'analyser toutes les transcriptions transférées pour signaler les erreurs potentielles de segmentation, et ainsi faciliter le nettoyage des données.

1.4 Les prédécesseurs du projet

Comme expliqué avant, *Gallic(orpor)a* a rassemblé les recherches de plusieurs projets précédents. Il a profité des glossaires codicologiques, des progrès dans la prédiction et la segmentation des documents, des progrès dans l'analyse linguistique, et des catalogues des données. Par exemple, le vocabulaire *SegmOnto* permet d'harmoniser les vérités de terrain pour les manuscrits et les imprimés. Les modèles HTR sont à la fois l'un des livrables du projet et un support pour la production des vérités de terrain, en opérant une transcription préliminaire qui sert à entraîner des nouveaux modèles. En outre, grâce au catalogue Open Source HTR-United, un portail qui agrège des recherches et permet de retrouver des données, les vérités de terrain du projet *Gallic(orpor)a* seront visibles et aisées à retrouver pour d'autres projets[7]. Enfin, l'analyse linguistique est également

7. Alix CHAGUÉ et Thibault CLÉRICE. « Sharing HTR Datasets with Standardized Metadata : The HTR-United Initiative ». In : Documents Anciens et Reconnaissance Automatique Des Écritures Manuscrites. 23 juin 2022. URL : `https://hal.inria.fr/hal-03703989` (visité le 12/08/2022).

un objectif du projet *Gallic(orpor)a*, et la mise en œuvre de cet aspect s'appuie sur les modèles de la langue française construits par les chercheurs de l'équipe ALMAnaCH. Toutefois, ce point n'a pas encore été ajouté au pipeline du projet.

1.4.1 Le glossaire codicologique

Le projet *Gallic(orpro)a* a produit des données en accord avec le vocabulaire *SegmOnto*, qui est expliqué en détail dans le chapitre 2. Ayant géré les données produites selon cette codicologie, j'ai aussi contribué à l'élaboration et le perfectionnement du vocabulaire. La décision d'utiliser la syntaxe descriptive des lignes et des zones de *SegmOnto* a été prise dès la mise en œuvre du projet *Gallic(orpor)a*. La généricité du vocabulaire permet de traiter des documents très divers dans le cadre du projet. Comme *Gallic(orpor)a* vise à livrer un prototype d'un pipeline qui pourrait traiter tout document source numérisé, la généralisation des étiquettes appliquées aux lignes et aux zones était impérative, et le vocabulaire de *SegmOnto* la meilleure solution.

1.4.2 La segmentation et la prédiction du texte

Les progrès de la reconnaissance du texte sont expliqués en détail dans le chapitre 3. Un logiciel HTR commence par la segmentation de la page, et dès qu'il sait où se trouvent les caractères, les mots, et les lignes du texte, il le prédit à partir de l'écriture. Ces deux tâches se font selon les compétences qu'il a apprises lors de son entraînement. Dans le but de produire les vérités de terrain pouvant entraîner les modèles HTR pour les manuscrits, les incunables, et les imprimés dans la base de données Gallica, le projet *Gallic(orpor)a* a profité de l'expertise de Simon Gabay et d'Ariane Pinche, qui s'occupaient de la relecture des vérités de terrain et la gestion des corpus d'or.

Les progrès dans la prédiction du texte sur les imprimés de l'Ancien Régime ainsi que son analyse ont aidé le projet *Gallic(orpor)a*. Par les imprimés françaises de l'Ancien Régime, S. Gabay a entraîné les modèles sur les vérités de terrain des imprimés du XVI^e^ au XVIII^e^ siècle[8]. En collaboration avec d'autres chercheurs, il a travaillé sur le jeu de données OCR17+ qui a fourni des vérités de terrain des imprimés du XVII^e^ siècle[9]. Pour tester le pipeline, dans l'attente des modèles nouvellement entraînés sur les données produites dans le cadre de *Gallic(orpor)a*, j'ai utilisé un modèle de segmentation et un modèle d'HTR que Gabay a développé dans le cadre de son projet *E-ditiones*[10].

8. Simon GABAY et al. « Standardizing Linguistic Data : Method and Tools for Annotating (Pre-Orthographic) French ». In : *Proceedings of the 2nd International Digital Tools & Uses Congress (DTUC '20)*. Hammamet, Tunisia, oct. 2020. DOI : 10.1145/3423603.3423996. URL : https://hal.archives-ouvertes.fr/hal-03018381 (visité le 12/08/2022).

9. GABAY, CLÉRICE et REUL, *OCR17*.

10. Simon GABAY. *E-Ditiones, 17th c. French Sources*. Nov. 2018. URL : https://hal.archives-ouvertes.fr/hal-02388415 (visité le 10/08/2022).

Pour la reconnaissance du texte sur les documents médiévaux, le projet CREMMA-Lab est la clef. Géré dans le cadre des études postdoctorales d'Ariane Pinche, le Consortium Reconnaissance d'Écriture Manuscrite des Matériaux Anciens, ou CREMMA, est un dépôt des images et leurs transcriptions corrigées à la main, soit des vérités de terrain. Afin d'entraîner un modèle HTR, il faut un jeu des vérités de terrain [11]. Le projet CREMMALab fournit un jeu des vérités de terrain de 13 manuscrits médiévaux qui se composent de 21 656 lignes de texte transcrites [12]. Sur les données du projet, A. Pinche a entraîné un modèle HTR qui est désormais disponible sur Zenodo [13]. Le projet *Gallic(orpor)a* en a également permis la création de vérités de terrain pour les manuscrits médiévaux.

1.4.3 Harmonisation et partage des données

Le projet HTR-United permet de cataloguer des vérités de terrain générées par tout projet *open source* [14]. Sa base de données, librement mise en ligne par GitHub, contient les images et leurs transcriptions faites par plusieurs projets de recherche. Elle comporte sur les documents de plusieurs périodes historiques et écritures. Un modèle HTR peut être entraîné sur ces jeux de données. Par exemple, Alix Chagué a entraîné un modèle HTR sur les vérités de terrain du *LECTAUREP Project*, soutenu par Inria et les Archives Nationales, qui sont recensées sur la base de données HTR-United [15].

Dans l'esprit de la science ouverte, toutes les vérités de terrain produites par le projet *Gallicorpora* apparaissent dans le catalogue HTR-United. À la fin du stage, en juillet 2022, Ariane Pinche et Thibault Clérice ont entraîné un modèle pour les manuscrits médiévaux en utilisant les transcriptions que l'équipe du projet *Gallic(orpor)a* a produites. Ces vérités de terrain sont désormais mises en commun sur HTR-United et A. Pinche et T. Clérice ont lié le premier modèle publié du projet *Gallic(orpor)a* avec le catalogue HTR-United et le dépôt du projet CREMMA (Consortium Reconnaissance d'Écriture Manuscrite des Matériaux Anciens) [16]. Le partage des données du projet est l'un de ses objectifs.

11. Alix Chagué, Thibault Clérice et Laurent Romary. « HTR-United : Mutualisons La Vérité de Terrain ! » In : *DHNord2021 - Publier, Partager, Réutiliser Les Données de La Recherche : Les Data Papers et Leurs Enjeux*. Lille, France : MESHS, nov. 2021. URL : https://hal.archives-ouvertes.fr/hal-03398740 (visité le 10/08/2022).

12. Ariane Pinche et Jean-Baptiste Camps. « CremmaLab Project : Transcription Guidelines and HTR Models for French Medieval Manuscripts ». In : *Colloque "Documents Anciens et Reconnaissance Automatique Des Écritures Manuscrites"*. Paris, France, juin 2022. URL : https://hal.archives-ouvertes.fr/hal-03716526 (visité le 10/08/2022).

13. Ariane Pinche. « HTR model Cremma Medieval ». In : (21 juin 2022). DOI : 10.5281/zenodo.6669508. URL : https://zenodo.org/record/6669508 (visité le 10/08/2022).

14. Alix Chagué et al. *HTR-United/Htr-United : V0.1.28*. Zenodo, 10 août 2022. DOI : 10.5281/zenodo.6979746. URL : https://zenodo.org/record/6979746 (visité le 12/08/2022).

15. Alix Chagué. *LECTAUREP Contemporary French Model (Administration)*. Zenodo, 12 mai 2022. URL : https://zenodo.org/record/6542744 (visité le 12/08/2022).

16. Ariane Pinche et Thibault Clérice. *HTR-United/Cremma-Medieval : Cortado 2.0.0*. Zenodo, 11 juill. 2022. DOI : 10.5281/zenodo.6818057. URL : https://zenodo.org/record/6818057 (visité le 12/08/2022).

1.4.4 L'analyse linguistique

Après l'extraction et le nettoyage des données des documents source de Gallica, le projet *Gallic(orpor)a* a envisagé de lemmatiser les textes. Dans cet objectif, il a profité des derniers progrès dans le domaine de l'analyse linguistique du français classique. L'analyse linguistique du français de l'Ancien Régime, tel que ce qui se voit dans les écrits de Molière, est un domaine de recherche actuellement en plein développement. Depuis une dizaine d'années, les chercheurs en linguistique computationnelle ont élaboré des outils pour analyser des états de langue ancien du français. La recherche dans le domaine du français contemporaine quant à elle est déjà bien ayant bénéficié de jeux de données plus nombreuses et accessibles et de financements privés liés à ces applications commerciales.

L'histoire de l'analyse linguistique et du Traitement automatique des langues (TAL) est trop riche pour être retracée ici. Néanmoins, quelques étapes méritent d'être rappelées. Achim Stein a abordé le sujet de l'analyse linguistique du français du Moyen Âge dans son article de 2013 [17]. Dans le même année, il a collaboré avec Sophie Prévost pour créer un *treebank* pour l'ancien français, le *Syntactic Reference Corpus of Medieval French* (SRCMF) [18]. En 2014, Prévost est des autres chercheurs, Gael Guibon, Isabelle Tellier, Matthieu Constant, et Kim Gerdes, ont soutenu que la variation de lexique de l'ancien français pose un défi à l'analyse linguistique [19]. En 2019, Mathilde Regnault, S. Prévost, et Éric Villemonte de La Clergerie ont utilisé le SRCMF avec le MCVF (Modéliser le changement : les voies du français) pour mieux décrire et structurer notre connaissance de l'évolution du français [20]. Ces études ont été des pionnières qui nous permettent aujourd'hui de mieux élaborer des modèles pour analyser des textes historiques en ancien français. Un exemple d'une telle élaboration récente est le modèle *Deucalion*. Jean-Baptiste Camps, Simon Gabay, Paul Fièvre, Thibault Clérice, et Florian Cafiero l'ont créé pour le français de l'Ancien Régime, en l'entraînant sur les livrets des drames du XVII^{e} siècle [21]. Le développement des nouveaux modèles TAL pour les anciens états du français a rendu possible

17. Achim Stein et Sophie Prévost. *Syntactic Annotation of Medieval Texts : The Syntactic Reference Corpus of Medieval French (SRCMF)*. Narr Verlag, 2013, p. 275. ISBN : 978-3-8233-6760-4. URL : https://halshs.archives-ouvertes.fr/halshs-01122079 (visité le 10/08/2022).

18. Achim Stein et Sophie Prévost. *Syntactic Reference Corpus of Medieval French (SRCMF)*. Stuttgart : ILR University of Stuttgart, 2013. ISBN : 899-492-963-833-3. URL : http://srcmf.org.

19. Gaël Guibon et al. « Parsing Poorly Standardized Language Dependency on Old French ». In : *Thirteenth International Workshop on Treebanks and Linguistic Theories (TLT13)*. Sous la dir. de V. Henrich et al. Proceedings of the Thirteenth International Workshop on Treebanks and Linguistic Theories (TLT13). Tübingen, Germany, déc. 2014, p. 51-61. URL : https://hal.archives-ouvertes.fr/hal-01250959 (visité le 10/08/2022).

20. Mathilde Regnault, Sophie Prévost et Éric Villemonte de La Clergerie. « Challenges of Language Change and Variation : Towards an Extended Treebank of Medieval French ». In : *TLT 2019 - 18th International Workshop on Treebanks and Linguistic Theories*. Paris, France, août 2019. URL : https://hal.inria.fr/hal-02272560 (visité le 10/08/2022).

21. Jean-Baptiste Camps et al. « Corpus and Models for Lemmatisation and POS-tagging of Classical French Theatre ». In : *Journal of Data Mining & Digital Humanities* 2021 (Digital humanities in... 14 fév. 2021), p. 6485. ISSN : 2416-5999. DOI : 10.46298/jdmdh.6485. arXiv : 2005.07505 [cs]. URL : http://arxiv.org/abs/2005.07505 (visité le 09/08/2022).

la mise en œuvre éventuelle d'une telle étape d'analyse au pipeline de *Gallic(orpro)a*.

1.5 Le pipeline

Outre la création des vérités de terrain, qui est expliqué dans une section précédente (Section 1.3), le projet *Gallic(orpor)a* vise à pouvoir générer une ressource numérique qui porte sur le document source en liant une transcription des pages avec deux versions du texte, l'une pré-éditorialisée et l'autre soumise à l'analyse linguistique. En commençant par les pages numérisées du document source, la ressource numérique présente quatre types d'information :

1. les métadonnées à propos du document source
2. la transcription du document, prédite par les modèles HTR
3. le texte pré-éditorialisé du document source
4. le texte analysé par les outils Traitement automatique des langues (TAL)

Chaque type d'information offre un panel de possibilité dans l'usage des données textuelles. Les métadonnées s'informent sur trois types de document concerné par le pipeline : (1) la ressource numérique qu'il créé, (2) le fac-similé numérique du document source d'où vient la transcription par HTR, (3) le document source physique qui se conserve quelque part, sinon qui a été conservé avant sa disparition, et qui est à l'origine du fac-similé numérique. Les données produites par les modèles HTR contiennent la segmentation de la mise en page et la prédiction du texte. Ensuite, le texte pré-éditorialisé est présenté, sans sa segmentation, d'une manière plus propice à l'analyse linguistique. Et enfin, le fichier final devrait proposer une analyse linguistique du texte grâce à l'application de certains modèles TAL.

Voir la représentation du pipeline sur la Figure 1.3. En bleu les saisies des données, en vert les fichiers préliminaires que le pipeline construit, en orange sa première sortie, et en violet les divers moyens d'exploitation de sa sortie. Comme le pipeline permet de générer des métadonnées et la transcription, il a besoin d'au moins deux saisies des données. L'une porte sur les métadonnées des trois types de document concernés (la ressource numérique produite par le pipeline, le fac-similé numérique du document source, le document source physique). Elle se composer de plusieurs sources de données en ligne, afin de fournir à la ressource numérique autant de détail que possible. L'autre saisie des données est l'image numérique elle-même en format JPEG, TIFF, ou PNG.

La Figure 1.3 montre en vert les premiers fichiers préliminaires du pipeline, les fichiers ALTO et la version du fichier TEI sans l'analyse linguistique. Les fichiers ALTO sont d'un format XML qui ont un schéma particulièrement adapté à l'encodage de l'analyse de la mise en page que fait un modèle de segmentation ainsi qu'à la prédiction du texte que fait un modèle HTR. Le chapitre 5 du mémoire en parle en plus de détail. Son

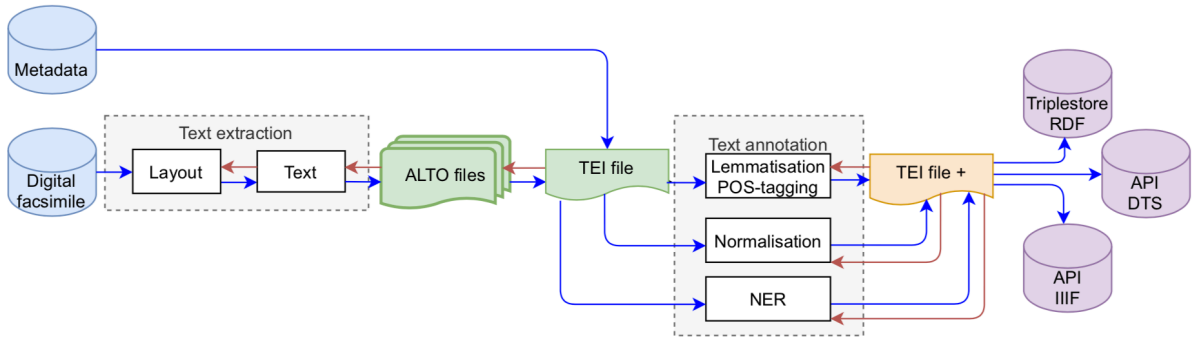


FIGURE 1.3 – Pipeline

acronyme veut dire en anglais *l'Analyzed Layout and Text Object*, ou la mise en page analysée et l'objet de texte. Comme le montre la nature bipartite du nom *Analyzed Layout and Text Object*, les fichiers ALTO alignent la mise en page et le texte prédit. Le pipeline a créé ces fichiers en appliquant les modèles HTR aux données d'entrée visuelles.

Après la création des fichiers préliminaires, le pipeline crée la première version du fichier TEI, qu'il va plus tard enrichir avec l'analyse linguistique du texte prédit. Ce premier fichier TEI occasionne la création des métadonnées. Comme le montre la flèche bleue dans la Figure 1.3, les métadonnées sont récupérées depuis les sources en ligne et ensuite nettoyées et intégrées au fichier TEI. Le fichier TEI préliminaire récupère aussi les données des fichiers ALTO.

À la suite de la récupération, du nettoyage, et enfin de la transformation des données des divers sources, le pipeline continue à traiter le texte qu'il a récupéré des fichiers ALTO. Dans un premier temps, il parse les données récupérées des fichiers ALTO et extrait toute ligne de texte imbriquée dans une zone qui fait partie du texte principal. Cela veut dire qu'il n'extrait pas de numéro de page, d'en-tête, etc. Les lignes de texte ainsi sélectionnées sont présentées comme le texte pré-éditorialisé du document source. Elles représentent une première transcription du texte, en ignorant la mise en page sauf les sauts de ligne.

Ce texte pré-éditorialisé peut aisément servir à l'analyse qu'une utilisatrice ou un utilisateur voudrait faire en s'appuyant sur le texte tel qu'il apparaît dans le document source. Ce texte sert aussi à la génération d'un texte annoté, issue de l'analyse linguistique des outils TAL. Le résultat de ce dernier traitement est visualisé dans la Figure 1.3 en orange sous le nom de fichier TEI enrichi, *TEI file +*. Pour terminer, les données de la sortie peuvent être exploitées dans plusieurs formats, comme le montrent les étapes signalées en violet dans la Figure 1.3.

Chapitre 2

Au commencement, il y avait les *guidelines SegmOnto*

2.1 L'enjeu de l'encodage

Un ordinateur fait ce que nous lui disons à faire. Si nous lui disons de rassembler toute note écrite dans les marges d'un livret autographe du librettiste Eugène Scribe, par exemple, l'ordinateur va chercher ce que nous lui avons dit se constitue une note en marge. Si nous lui avons dit qu'une note se constitue d'une suite de lignes de texte taguée par l'étiquette *MarginalZone*, l'ordinateur va nous renvoie toutes les lignes de texte imbriquées dans une telle zone dans ses données d'entrée. Mais si les données qu'il traite n'appliquaient pas systématiquement l'étiquette *MarginalZone* à toute note en marge, si elles appliquaient de temps en temps l'étiquette *MarginalNoteZone*, par exemple, l'ordinateur ne parviendrait pas à récupérer tout ce que nous attendons. Les données d'entrée sont à faute, ainsi que celui qui les a créé. Si nous aillions produire une édition numérique du livret autographe à partir des données ainsi traitées, en faisant que toute note en marge soit indiquée, notre édition manquerait certaines notes et nous ne saurions pas lesquelles.

Le pipeline du projet *Gallic(orpor)a* vise à produire une ressource numérique sur laquelle les chercheurs et les lecteurs peuvent compter. Il est important que la transcription produite par les modèles HTR et transformée en le document TEI soit aussi juste et complète que possible. Si les modèles qui ont produit la transcription représentée dans la ressource n'avaient pas appliqué systématiquement des étiquettes aux blocs de texte, les processus plus tard dans le pipeline, surtout mon application `alto2tei`, n'arriveraient pas à produire une ressource qui portent justement sur le document source.

L'équipe de *Gallic(orpor)a* a choisi de mettre en pratique le vocabulaire codicologique du projet *SegmOnto*. Son vocabulaire est l'une de plusieurs solutions proposées qui ont pour but de standardiser les étiquettes appliquées aux lignes de texte et aux régions d'une page. Le projet *SegmOnto* reprend un défi qui a été franchi par plusieurs chercheurs

depuis le vingtième siècle, comme j'explique prochainement. Par rapport aux autres projets, le vocabulaire de *SegmOnto* convient le mieux aux enjeux du projet *Gallic(orpor)a* parce qu'il a été développé pour un contexte informatique. L'équipe de *Gallic(orpor)a* a entraîné les modèles du projet sur le vocabulaire afin qu'ils appliquent les étiquettes aux documents traités par son pipeline. En outre, la standardisation du vocabulaire *SegmOnto* permet l'application `alto2tei` de filtrer les prédictions des modèles et leur appliquer les étiquettes du schéma TEI.

2.2 Les solutions proposées auparavant

Plusieurs projets ont proposé des solutions pour la description normalisée des documents sources. Hors de la France, des ontologies ont été élaborées notamment en anglais et pour les manuscrits médiévaux. Un exemple important est la base de données *DigiPal* (Digital Resource and Database of Palaeography, Manuscripts and Diplomatic), qui n'est plus mise à jour, mais qui a été développée au sein du département des humanités numériques à King's College London. [1] L'un de ses auteurs, Peter Stokes, travaille actuellement en France et continue dans la même veine en contribuant au projet *SegmOnto* qui était développé dans un environnement francophone, même si son vocabulaire est rédigé en anglais. [2] Mais les projets *DigiPal* et *SegmOnto* ne sont pas les premiers projets à essayer d'élaborer un lexique pour les documents historiques. Avant leur création, la codicologie en France et en Europe suivait le modèle de Denis Muzerelle et son *Vocabulaire codicologique*. Ce dernier n'était pas développé pour une utilisation informatique, mais il servait comme modèle pour les projets du même esprit codicologique.

2.2.1 Le Vocabulaire international de la codicologie

En 1985, Denis Muzerelle a conçu un vocabulaire codicologique qui avait pour but de fournir aux médiévistes des termes communs afin de décrire les différents aspects d'un manuscrit. [3] Après sa publication, l'ouvrage de D. Muzerelle a été traduit en d'autres langues. Marilena Maniaci a publié une version italienne du *Vocabulaire codicologique* en 1996. [4] Pilar Ostos, Luisa Pardo, et Elena RodrÃguez en ont créé un pour l'espagnol

1. *DigiPal : Digital Resource and Database of Palaeography, Manuscripts and Diplomatic.* London. URL : `http://www.digipal.eu/`.

2. Simon Gabay, Jean-Baptiste Camps et Ariane Pinche. « SegmOnto ». In : *Création de Modèle(s) HTR Pour Les Documents Médiévaux En Ancien Français et Moyen Français Entre Le Xe-XIVe Siècle*. Paris, France : Ecole nationale des chartes | PSL, nov. 2021. URL : `https://hal.archives-ouvertes.fr/hal-03481089` (visité le 25/07/2022).

3. Dennis Muzerelle. *Vocabulaire Codicologique : Répertoire Méthodique Des Termes Français Relatifs Aux Manuscrits*. Rubricae 1. Paris : Éd. Cemi, 1985.

4. Marilena Maniaci. *Terminologia Des Libro Manoscritto*. Addenda 3. Rome : Istituto centrale per la patologia del libro, 1996.

l'année suivante.[5] Parfois appelée le *Vocabulaire international de la codicologie*, l'édition multilingue du *Vocabulaire codicologique* initiée par D. Muzerelle en 1985 n'est plus mis à jour.

2.2.2 La Codicologia

Aujourd'hui, la paléographie et l'étude des manuscrits peuvent s'appuyer sur l'application web *Codicologia* qui réunit le *Vocabulaire codicologique* initiée par D. Muzerelle ainsi que deux autres bases de données similaires : le projet multilingue *Lexicon* et le *Glossaire codicologique arabe*. Bien qu'ils soient spécialisées dans d'écritures différentes, les projets s'appuient tous les trois sur la description des manuscrits. Le *Vocabulaire codicologique* a été développé pour les manuscrits en écriture latine. Piloté par Philippe Bobichon, le projet *Lexicon* présente un vocabulaire en français pour décrire les manuscrits écrits en latin, roman, grec, hébreu, et arabe[6]. Un vocabulaire spécialisé plus profondément pour l'arabe a été élaboré dans le *Glossaire codicologique arabe* d'Anne-Marie Eddé et Marc Geoffroy[7]. Ce dernier a été conçu au sein de l'Institut de recherche et d'histoire des textes (IRHT) d'après les modèles de Muzerelle et le vocabulaire codicologique en arabe d'Adam Gacek[8].

L'application web *Codicologia* rassemble ces projets et présente un vocabulaire bien étendu. Par exemple, *Codicologia* fournit 15 termes pour décrire une faute d'écriture dans un manuscrit. Certains de ses termes possèdent eux-mêmes plusieurs définitions fournies par les diverses bases de données. Le terme *caviarder*, par exemple, a une définition courte dans le vocabulaire français de Muzerelle.

> Supprimer un mot, un passage..., en le recouvrant largement d'encre, de façon à ce qu'il ne puisse être lu[9].

Selon le *Lexicon* de Bibichon, par contre, le terme *caviarder* est défini de manière plus détaillée et vise à expliquer l'étymologie du mot afin de préciser son usage en contexte.

> Le mot [*caviarder*] apparaît en 1907 (noircir à l'encre) : il désigne alors un procédé appliqué par la censure russe, sous Nicolas I^{er}. Dans certains manuscrits

5. Pilar Ostos, M. Luisa Pardo et Elena E. Rodríguez. *Vocabulario de codicologÍa : Versión Española Revisada y Aumentada Del Vocabulaire Codicologique*. Instrumenta Bibliologica. Madrid : Arco/Libros, 1997.

6. Philippe Bobichon. « Le Lexicon : Mise En Page et Mise En Texte Des Manuscrits Hébreux, Grecs, Latins, Romans et Arabes ». In : (2009), p. 81. URL : https://cel.archives-ouvertes.fr/cel-00377671 (visité le 25/07/2022).

7. « Glossaire codicologique français-arabe ». In : *Gazette du livre médiéval* 40.1 (2002), p. 79-80. URL : https://www.persee.fr/doc/galim_0753-5015_2002_num_40_1_1563 (visité le 25/07/2022).

8. Adam Gacek. *The Arabic Manuscript Tradition : A Glossary of Technical Terms and Bibliography*. Handbook of Oriental Studies 1, The Near and Middle East. Leiden : Brill, 2001. 269 p. ISBN : ISBN 90-04-12061-0.

9. Denis Muzerelle. *Caviarder*. In : *Codicologia*. Institut de recherche et d'histoire des textes, 2011. URL : http://codicologia.irht.cnrs.fr/theme/liste_theme/413#tr-868 (visité le 25/07/2022).

> grecs, le détail rempli d'encre est surmonté d'un point et d'un trait court destinés à le neutraliser. Ce procédé est très souvent utilisé parmi d'autres, pour la censure des manuscrits hébreux effectuée sous l'autorité de l'inquisition, en Italie, à la fin du XVI^e^ siècle et au début du XVII^e^. [10]

Étant élaboré à partir d'un corpus très diversifié, le *Lexicon* de P. Bibichon a moins de termes que le *Vocabulaire codicologique*, mais ses termes sont plus généralistes. Le vocabulaire de Muzerelle, par contre, fait plus de distinctions entre les différents aspects d'un manuscrit et donc a plus de termes distincts par rapport aux deux autres vocabulaires de l'application *Codicologia*.

En réunissant les trois bases de données, sans en privilégier aucune, le *Codicologia* de l'IRHT présente un vocabulaire codicologique vraiment vaste. L'application *Codicologia* cherche à proposer à la communauté scientifique des termes communs pour décrire les manuscrits, malgré son étendue. Ayant plus de deux milles termes en français—certains d'entre eux ont eux-mêmes plusieurs définitions—la solution proposée par *Codicologia* livre un vocabulaire documenté, mais trop étendu pour être appliqué dans une approche informatique.

Sans un corpus d'entraînement gigantesque, qui couterait une somme énorme, l'apprentissage automatique ne peut pas faire de distinction au niveau des termes conçus par Muzerelle et les autres auteurs des bases de données de *Codicologia*. Aujourd'hui, un modèle ne peut pas apprendre des milliers d'étiquettes différents et distinguer, par exemple, 15 types de faute d'écriture. Un humain peut le faire, et pour cette raison les bases de données de *Codicologia* sont utiles. Mais leurs vocabulaires ne conviennent pas bien à une approche informatique.

2.3 Les *guidelines* de *SegmOnto*

Le projet *SegmOnto* propose un vocabulaire plus petit qui peut pourtant décrire une grande diversité de documents historiques, soit des manuscrits, soit des imprimés. Cet objectif est encore plus compliqué à achever que le vocabulaire spécialisé aux manuscrits qu'ont réalisé les projets *Codicologia* et *DigiPals*. Décrire les documents en diachronie longue, et sans préférence d'une écriture en particulier, exige un équilibre délicat entre la généralité et la particularité. Pour y arriver, les *guidelines* du projet *SegmOnto* limitent le nombre de termes de son vocabulaire, sans perdre de distinction importante entre eux.

Les *guidelines* se divisent en deux catégories : les « zones » et les « lines ». La première désigne des régions sur la page qu'elles ont du texte ou non, telle comme la décoration ou l'enluminure. La plupart de temps, l'étiquette de la ligne veut décrire les différents types

10. Philippe BOBICHON. *Caviarder*. In : *Codicologia*. Institut de recherche et d'histoire des textes, 2011. URL : `http://codicologia.irht.cnrs.fr/theme/liste_theme/413#tr-868` (visité le 25/07/2022).

de lignes de texte. Cependant, dans le vocabulaire *SegmOnto*, une ligne peut aussi tracer la ligne d'une partition musicale ou être une ligne réelle sur la page qui n'oriente pas d'autres systèmes d'écriture, telle qu'une ligne qui divise la page en deux. Chacune des deux catégories, les lignes et les zones, se compose d'une liste des étiquettes. Et chacune d'étiquettes cherche à parvenir à l'équilibre entre la généralité et la particularité. Une étiquette devrait pouvoir être appliquée soit à un manuscrit, soit à un imprimé, quels que soient la langue ou le type d'écriture.

2.3.1 Les zones

- **CustomZone** : une zone qui ne convient pas à aucune d'autres catégories de zone.
- **DamageZone** : une zone qui contient des marques de dégâts sur le document source, tel qu'un trou.
- **DecorationZone** : une zone qui contient un élément graphique, y compris de la peinture et les petits dessins dans la marge de la page.
- **DigitizationArtefactZone** : une zone qui contient un item qui n'appartient pas au document source, mais est lié au processus de la numérisation, tel qu'une règle pour montrer la mesure du document.
- **DropCapitalZone** : une zone qui contient une initiale ; l'initiale peut prendre l'espace de plusieurs lignes de texte ou porter une décoration importante, tel que de l'historicisation, l'ornementation, ou des dessins.
- **MainZone** : une zone qui contient le texte principal du document source.
- **MarginTextZone** : une zone qui contient le texte dans la marge du document source.
- **MusicZone** : une zone qui contient une partition musicale.
- **NumberingZone** : une zone qui contient des numéros de page, y compris les numéros rédigés en chiffres romans.
- **QuireMarksZone** : une zone qui contient des notes en bas page destinées à la fabrication du document source pour garder les pages dans le bon ordre.
- **RunningTitleZone** : une zone qui contient une version du titre du document ou d'une section du document qui se trouve en tête de la page.
- **SealZone** : une zone qui contient un sceau sur le document source.
- **StampZone** : une zone qui contient l'empreinte d'un tampon sur le document source.
- **TableZone** : une zone qui contient une table.
- **TitlePageZone** : une zone souvent sur l'une des premières pages du document source qui contient toutes les informations concernant le titre et l'édition du document.

2.3.2 Les lignes

— **CustomLine** : une ligne qui ne convient pas à aucune d'autres catégories de ligne.
— **DefaultLine** : une ligne qui contient du texte attendu dans la zone.
— **DropCapitalLine** : une ligne qui contient l'initiale.
— **HeadingLine** : une ligne qui contient le texte d'un titre, tel que celui d'une section ou d'un chapitre.
— **InterlinearLine** : une ligne qui traverse la page pour marquer une limite.
— **MusicLine** : une ligne de la portée d'une partition.

Chapitre 3

Qu'est-ce que l'HTR ?

Qu'est-ce qu'est l'*Handwritten Text Recognition* ou la reconnaissance automatique d'écriture manuscrite ? L'*Handwritten Text Recognition* (HTR) est l'approche que l'équipe de *Gallic(orpor)a* a choisi pour prédire le texte écrit sur les fac-similés numériques de Gallica, y compris les manuscrits et les imprimés. À partir des pages numérisées, un logiciel HTR peut décomposer l'image en segments et en lignes de texte ainsi que reconnaître l'écriture dedans. Dit simplement, l'HTR parvient à réaliser un texte bien structuré selon la logique de la page à partir des images numériques du texte. Cette technologie était utile au projet *Gallic(orpor)a* dont le pipeline a pour but de saisir les pages numérisées et d'arriver à une transcription du texte ainsi qu'aux versions du texte pré-éditorialisées et annotées.

3.1 Les origines de l'HTR

L'*Handwritten Text Recognition* a évolué à partir des recherches sur l'*Optical Character Recognition*[1]. Cette technologie précédente a pour but la reconnaissance du texte imprimé, y compris la police du texte. N'étant pas développé pour l'écriture manuscrite, l'OCR ne convient pas assez bien à la prédiction du texte des documents manuscrits. Afin de surmonter les polices qui imposent des limites à l'OCR, l'HTR a été développé. La première conférence *Frontiers in Handwriting Recognition* a eu lieu en 1990. Aujourd'hui, le champs de recherche de l'HTR est en plein développement. Bien que l'HTR se soit développé en parallèle avec l'OCR, le dernier l'a précédé et a préparé le terrain pour les nouveautés dans l'HTR à la fin du siècle précédent.

Les origines de l'OCR remontent au dix-neuvième siècle et les progrès les plus importants au début ont eu lieu aux États-Unis dans les années 1960. En effet, le développement

1. Sebastiano Impedovo. « More than Twenty Years of Advancements on Frontiers in Handwriting Recognition ». In : *Pattern Recognition*. Handwriting Recognition and Other PR Applications 47.3 (1er mars 2014), p. 916-928. issn : 0031-3203. doi : 10.1016/j.patcog.2013.05.027. url : https://www.sciencedirect.com/science/article/pii/S0031320313002513 (visité le 29/08/2022).

des logiciels OCR a été réalisé dans le cadre de l'amélioration du traitement en masse des données numérisées. Dans les années soixante, le besoin de gérer et traiter de grandes quantités de données est devenu de plus en plus pressant notamment dans les grandes entreprises, dont les banques [2]. La numérisation des journaux et le traitement en masse des lettres à la poste furent aussi l'un des contextes importants du développement de la reconnaissance automatique du texte sur des images.

Dans les années 1960, de plus en plus de sociétés souhaitaient adopter les méthodes de traitement des données assisté par ordinateur. Ce traitement a exigé l'encodage des données dans un format directement exploitable par ordinateur. Il y avait alors deux supports pour l'encodage : (i) la carte perforée dans laquelle une machine faisait des trous qu'un ordinateur savait lire, et (ii) la bande magnétique sur laquelle une machine imprimait des bytes, la plus petite unité exploitable par ordinateur. Dans les deux cas, le matériel a changé, du carton à la bande magnétique, mais le mécanisme binaire d'analyse restait : plein ou vide pour le carton, positif ou négatif pour la bande.

Les deux supports ont été développées pour l'acquisition manuelle des données de texte, mais l'OCR a révolutionnée la bande magnétique en l'utilisant pour l'acquisition automatique. Un individu (souvent une femme [3]) saisit des données en lisant du texte et en le tapant soit à la perforatrice pour l'enregistrer sur la carte perforée, soit par le clavier du *Magnetic Tape/Selectric Typewriter* (MT/ST) pour l'enregistrer sur la bande magnétique. Ce dernier outil a été développé par l'entreprise américaine IBM dans les années 1960. Comme le MT/ST, l'*Optical Character Recognition* s'appuyait aussi sur la bande magnétique mais il a ajouté la possibilité d'enregistrer automatiquement du texte. Au lieu d'une personne au clavier, un logiciel OCR saisit l'image d'une page qu'un scanner a numérisée.

En 1971, le chercheur Ben R. Schneider a comparé l'OCR et les deux autres moyens de l'encodage du texte qui étaient courants pendant les années 1960 : le MT/ST et la carte perforée [4]. Schneider a déterminé que l'OCR n'avait pas encore surpassé l'appareil MT/ST d'IBM parce que, malgré sa saisie automatique des données, il exigeait toujours beaucoup de correction à la main. Afin de corriger la transcription du logiciel OCR, il fallait relire sa prédiction en entière en lisant au même temps le texte d'origine. L'acquisition manuelle du texte par l'appareil MT/ST avait une exactitude supérieure à l'OCR à l'époque parce que la personne au clavier pouvait corriger au même temps qu'elle enregistre les données,

2. Deepa Berchmans et S S Kumar. « Optical Character Recognition : An Overview and an Insight ». In : *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT). Juill. 2014, p. 1361-1365. DOI : 10.1109/ICCICCT.2014.6993174.

3. Thomas J. Misa. *Gender Codes : Why Women Are Leaving Computing*. John Wiley & Sons, 14 sept. 2011. 326 p. ISBN : 978-1-118-03513-9. Google Books : EjDYh_KHls8C, p. 85-87.

4. Ben R. Schneider. « The Production of Machine-Readable Text : Some of the Variables ». In : *Computers and the Humanities* 6.1 (sept. 1971), p. 39-47. ISSN : 0010-4817, 1572-8412. DOI : 10.1007/BF02402324. URL : http://link.springer.com/10.1007/BF02402324 (visité le 02/08/2022).

en faisant les deux tâches dans une même étape.

En outre, un logiciel OCR était limité par son jeu de données de polices, puisqu'il prédit du texte en faisant un comparaison entre un caractère qu'il a mémorisé et le motif qu'il a reconnu sur l'image. Même aujourd'hui l'OCR se distingue de l'HTR par le fait qu'il compte sur une base de données des polices. Contraire à l'OCR dans les années 1960, l'appareil MT/ST pouvait encoder des documents des diverses polices puisque la technologie comptait sur le discernement d'un être humain lors de la saisie des données.

En 1977, Gian Piero Zarri, en résumant l'état de l'art de l'OCR, a remarqué que la reconnaissance du texte sur les manuscrits n'était pas encore faisable.

> Rappelons que la reconnaissance optique des caractères permet la lecture directe du texte par un « scanner » qui se charge d'effectuer le transfert sur bande magnétique ; le texte doit être composé avec des caractères de type « imprimerie » ou « machine à écrire », car la lecture de caractères « manuels » ne semble pas encore actuellement complètement sortie de la phase expérimentale. [5]

Dans les mêmes années, le chercheur américain Ray Kurzweil a fait le même constat et a développé le *Kurzweil Reading Machine* (KRM), qui pouvait reconnaître plus qu'une police [6]. Bien que l'OCR puisse désormais en reconnaître plusieurs, la technologie reste toujours sous la dépendance des polices et, par conséquent, la remarque de Zarri reste toujours importante pour les objectifs du projet *Gallic(orpor)a* et continue de contextualiser les progrès de l'HTR.

3.2 Le fonctionnement général de l'HTR

Le projet *Gallic(orpor)a* a besoin de l'HTR parce que son pipeline traite à la fois les imprimés et les manuscrits. Bien qu'il mette en œuvre les modèles différents selon le type de document, soit un modèle pour les imprimés, soit un modèle pour les manuscrits, le pipeline profite toujours de l'HTR et ne s'appuie pas sur l'OCR. Son modèle destiné aux imprimés s'appuie sur la reconnaissance d'écriture manuscrite pour deux raisons principales. De l'un côté, les polices des incunables et des imprimés avant la révolution française ne sont pas si standardisées de ne pas pouvoir bien profiter du traitement HTR. De l'autre côté, les documents sources de la base de données Gallica peuvent contenir du texte manuscrit, telle qu'une note en marge. Une juste transcription de la page devrait conserver toute ligne de texte particulière au fac-similé numérique, quel que soit le type, imprimée

5. Gian Piero Zarri. « Quelques aspects techniques de l'exploitation informatique des documents textuels : saisie des données et problèmes de sortie ». In : *Publications de l'École Française de Rome* 31.1 (1977), p. 399-413. URL : https://www.persee.fr/doc/efr_0000-0000_1977_act_31_1_2286 (visité le 01/08/2022).

6. Gregory Goodrich. « Kurzweil Reading Machine : A Partial Evaluation of Its Optical Character Recognition Error Rate ». In : *Journal of Visual Impairment and Blindness* (12 jan. 1979).

ou manuscrite. Puisque les logiciels OCR ne se spécialisent pas au texte manuscrit, c'est mieux d'utiliser un logiciel HTR.

3.2.1 L'image numérique

L'un des objectifs de l'HTR est d'imiter l'œil humain, c'est-a-dire de reconnaître les lignes et les points d'une écriture sur une page numérisée. Un système informatique stock une image numérique dans un format qui la décompose en unités de pixel. Issu de l'anglais, le mot pixel signifie *picture element* et désigne l'élément minimal d'une image numérique [7]. Vus tous regroupés, des pixels peuvent donner l'impression des courbes d'un objet ou d'un caractère. Par exemple, les pixels visualisés dans la Figure 3.1 montrent la diagonale de la lettre « A » écrite en crayon rouge.

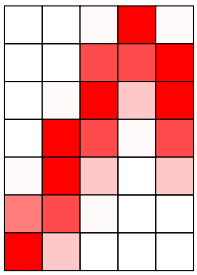


FIGURE 3.1 – Une couleur par pixel

Chaque pixel se compose de la superposition de l'intensité de trois couches de couleur : rouge, vert, et bleu. La Figure 3.2 montre l'exemple de la diagonale de la lettre « A » écrite en rouge avec la valeur de chaque couche de couleur de chaque pixel. Avec un peu de distance, l'œil humain distingue les formes qui composent une image à partir de ce canevas (*canvas*) des petits pixels.

3.2.2 Le *preprocessing*

Avant de faire des prédictions sur une image de texte, tout logiciel HTR ou OCR a besoin des étapes préliminaires qui sont connus ensemble sous le nom anglais *preprocessing*. L'une des étapes est souvent le redressement de l'image dans laquelle l'image s'ajuste jusqu'a les lignes de texte sont perpendiculaires. Un tel traitement est le *dewarping* que le projet *AGODA* a utilisé afin de corriger les courbes sur les pages numérisées des journaux pour que les lignes de texte soient toutes horizontales. [8]

7. Les pixels sont stockés sous le format *bitmap* ou BMP et chaque pixel peut compter 1 bit, 4 bits, 8 bits, ou 24 bits selon l'encodage de l'image

8. Aurélien PELLET et Marie PUREN. « Le Projet AGODA. Océrisation Des Débats Parlementaires Français de La Troisième République : Problèmes, Défis et Perspectives ». In : Séminaire OMNSH-Epitech : Le Numérique Au Service Des Sciences Humaines et Sociales. Le Kremlin-Bicêtre, France, avr. 2022. URL : `https://hal.archives-ouvertes.fr/hal-03651146` (visité le 29/08/2022).

255,255,255	255,255,255	255,250,250	255,0,0	255,250,250
255,255,255	255,255,255	255,75,75	255,75,75	255,0,0
255,255,255	255,250,250	255,0,0	255,200,200	255,0,0
255,255,255	255,0,0	255,75,75	255,250,250	255,75,75
255,250,250	255,0,0	255,200,200	255,255,255	255,200,200
255,125,125	255,75,75	255,250,250	255,255,255	255,255,255
255,0,0	255,200,200	255,255,255	255,255,255	255,255,255

FIGURE 3.2 – Donnée RVB (rouge, vert, bleu) du pixel

Avant le redressement de l'image, une étape qui se fait souvent est la binarisation. Depuis quelques années, cette étape n'est plus nécessaire pour l'HTR dont les logiciels réussissent à redresser et segmenter l'image en traitant directement les pixels RVB. Cependant, la binarisation est toujours courante pour les logiciels OCR contemporains [9]. Comme expliquent Patrick Jentsch et Stephan Porada, « *The idea is to only extract the pixels which actually belong to the characters and discard any other pixel information which, for example, is part of the background* » [10]. La binarisation trie les pixels d'une image en deux classes : l'arrière-plan (*background* en anglais) et le premier plan (*foreground* en anglais).

Le niveau de gris

Normalement pour binariser une image, on veut remplacer la valeur composée d'un pixel, c'est-à-dire les trois degrés du rouge, du vert, et du bleu, avec la valeur simple

9. Patrick JENTSCH et Stephan PORADA. « From Text to Data Digitization, Text Analysis and Corpus Linguistics ». In : *Digital Methods in the Humanities : Challenges, Ideas, Perspectives*. Sous la dir. de Silke SCHWANDT. Bielefeld University Press, 2021.

10. Ibid., p. 107.

d'un décimal. Ce décimal veut représenter l'échelle des gris dans le pixel. En anglais, ce traitement s'appelle le *grayscale* ou le niveau de gris en français. Aujourd'hui les langages de programmation ont souvent des librairies qui fournissent des méthodes pour rendre une image en niveau de gris automatiquement. Mais dans la figure 3.3, nous donnons un calcul simple qui permet le traitement des niveaux de gris. On commence toujours avec la donnée tripartite du pixel, qu'on veut remplacer par une seule valeur. Représentons chaque partie de la donnée du pixel par les variables p :

$$p_1, p_2, p_3$$

Dans un premier temps, on prend la moyenne des degrés de la couleur, c'est-à-dire la moyenne de p ou $\bar{p}$. La Figure 3.5a montre le résultat de ce calcul superposé à l'image originale.

$$\bar{p} = \sum_{i=1}^{3} \frac{1}{3} p_i = \frac{1}{3}(p_1 + p_2 + p_3)$$

Ensuite, on récupère $\bar{p}$ et la divise par la valeur maximale du p, qui est 255 parce que le degré du rouge, vert, ou bleu d'un pixel ne monte qu'à 255.

Donnée tripartite du pixel p_1, p_2, p_3	Moyenne du pixel $\sum_{i=1}^{3} p_i$	Valeur niveau de gris du pixel $\frac{\sum_{i=1}^{3} p_i}{\max\{p_i\}}$
255,000,000	$\bar{p} = \frac{255+0+0}{3} = 85$	$\frac{\bar{p}}{255} = 0.33$
255,075,075	$\bar{p} = \frac{255+75+75}{3} = 135$	$\frac{\bar{p}}{255} = 0.53$
255,125,125	$\bar{p} = \frac{255+125+125}{3} = 168.33$	$\frac{\bar{p}}{255} = 0.66$
255,200,200	$\bar{p} = \frac{255+200+200}{3} = 218.33$	$\frac{\bar{p}}{255} = 0.86$
255,250,250	$\bar{p} = \frac{255+220+220}{3} = 251.67$	$\frac{\bar{p}}{255} = 0.99$
255,255,255	$\bar{p} = \frac{255+255+255}{3} = 255$	$\frac{\bar{p}}{255} = 1.00$

FIGURE 3.3 – Évaluation des pixels

Le seuil

Il y a plusieurs techniques de binarisation mais toute a besoin d'un seuil (*threshold* en anglais). Le seuil nous permet de trier les pixels en les deux classes : le *background* et le *foreground*. Nos yeux font cette étape facilement, mais un ordinateur a besoin d'un algorithme. L'un des algorithmes les plus courant pour définir le seuil a été élaboré en 1979 par le chercheur Nobuyuki Otsu [11].

11. Nobuyuki OTSU. « A Threshold Selection Method from Gray-Level Histograms ». In : *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (jan. 1979), p. 62-66.

La méthode Otsu de seuillage reste toujours courante dans l'HTR [12] et elle fait partie de la librairie Python `numpy` [13]. Elle examine la variance entre les deux classes (*background* et *foreground*) pour déterminer un seuil idéal pour le jeu de données. Visualisées dans un histogramme, comme on voit dans la figure 3.4, les données d'un jeu de pixels devraient se lever dans deux sommets et, idéalement, une baisse profonde devrait les diviser. Cette variance veut dire qu'il y a dans l'image une distinction importante entre les contours d'un caractère et l'arrière-plan de l'image. La méthode de seuillage examine la variance entre ces deux classes, le contour et l'arrière-plan, pour générer un seuil adapté aux données.

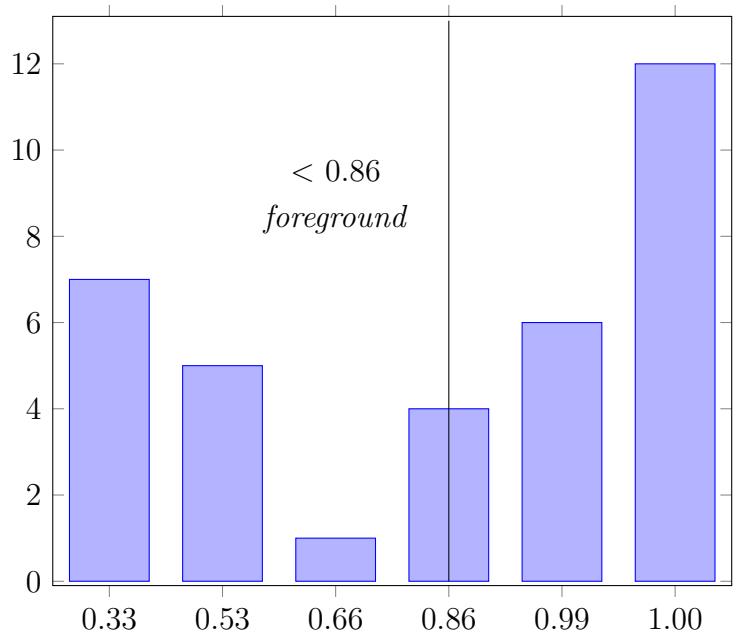


FIGURE 3.4 – Histogramme des valeurs niveau de gris des pixels

Le seuil sert à transformer la valeur numérique de chaque pixel soit en 0, soit en 1. Les pixels dont la valeur tombe au-dessous du seuil prennent la valeur 1 ; les autres, étant déterminés de ne pas faire partie du contour du caractère, prennent la valeur 0. La Figure 3.5c montre un exemple du résultat de ce triage. Ainsi, les logiciels OCR et les logiciels HTR qui utilisent toujours le *preprocessing* de l'OCR peuvent analyser les valeurs identiques et contiguës dans les données de l'image.

3.2.3 Les tâches d'un logiciel HTR

Pour aboutir à une prédiction, le système d'HTR passe par trois étapes principales : la reconnaissance des différentes zones de la page, l'analyse de ces zones, c'est-à-dire

12. Imran-Ahmed SIDDIQI, Florence CLOPPET et Nicole VINCENT. « Writing Property Descriptors : A Proposal for Typological Groupings ». In : *Gazette du livre médiéval* 56.1 (2011), p. 42-57. DOI : 10.3406/galim.2011.1981. URL : https://www.persee.fr/doc/galim_0753-5015_2011_num_56_1_1981 (visité le 02/08/2022).

13. *Numpy : NumPy Is the Fundamental Package for Array Computing with Python*. Version 1.23.1. URL : https://www.numpy.org (visité le 04/08/2022).

255	255	251.67	85	251.67
255	255	135	135	85
255	251.67	85	218.33	85
255	85	135	251.67	135
251.67	85	218.33	255	218.33
168.33	135	251.67	255	255
85	218.33	255	255	255

(a) La moyenne de chaque pixel

1.00	1.00	0.98	0.33	1.98
1.00	1.00	0.53	0.53	0.33
1.00	0.98	0.33	0.86	0.33
1.00	0.33	0.53	0.98	0.53
0.98	0.33	0.86	1.00	0.86
0.66	0.53	0.98	1.00	1.00
0.33	0.86	1.00	1.00	1.00

(b) L'image en niveau de gris

0	0	0	1	0
0	0	1	1	1
0	0	1	0	1
0	1	1	0	1
0	1	0	0	0
1	1	0	0	0
1	0	0	0	0

(c) L'image binarisée

FIGURE 3.5 – La binarisation

l'analyse de la mise en page, et la transcription du texte. L'analyse de la mise en page n'est pas obligatoire, mais les deux autres tâches sont essentielles. Chacune exige son propre modèle entraîné spécifiquement pour sa tâche. (cf. Figure 3.6)

La segmentation des zones de la page

La première tâche de la reconnaissance du texte est de localiser l'emplacement du texte. Cette étape s'appelle souvent la segmentation et elle est indispensable [14]. Selon son entraînement, un modèle de segmentation recherche les espaces entre les contours d'un caractère ou entre les lignes de texte. Si le modèle était entraîné pour reconnaître du texte écrit en japonais, par exemple, il rechercherait l'ensemble de caractères bordés à gauche et à droit de l'espace et les reconnaîtrait comme étant une ligne de texte. Le terme « masque »

14. CHAGUÉ, CLÉRICE et ROMARY, « HTR-United ».

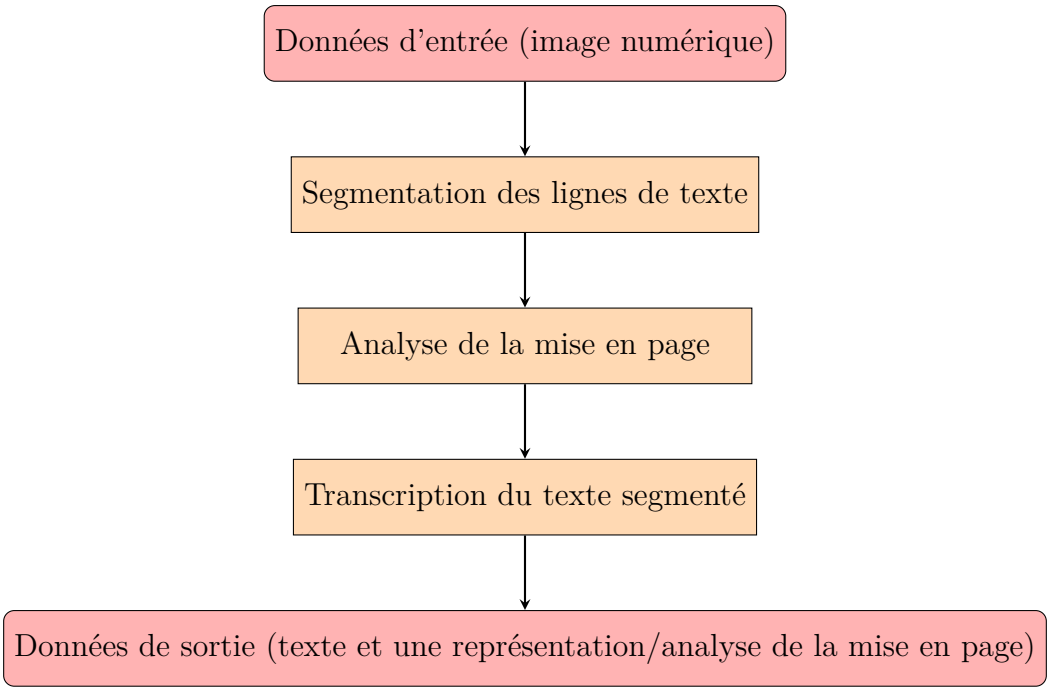


FIGURE 3.6 – Processus d'un logiciel HTR

signifie l'ensemble des pixels qui contient une ligne de texte ou un caractère[15]. Du point de vue du système informatique, un masque est un ensemble de coordonnées signifiant un objet contigu qui encadre une ligne de texte ou un caractère. Cet objet contigu est la base à partir de laquelle le modèle HTR peut reconnaître le texte dedans.

La transcription

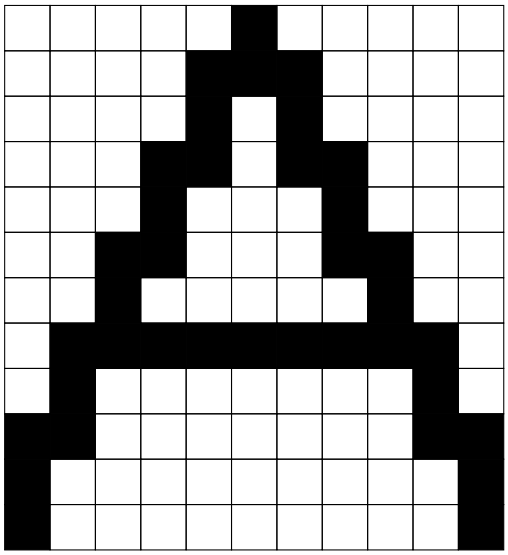
La transcription du texte est l'étape fondamentale de l'HTR. Elle s'effectue à partir des données encadrées dans les masques et en appliquant un modèle de reconnaissance, parfois appelé le modèle HTR. La transcription du texte consiste à faire produire, par le modèle, une suite de caractères encadrée dans un objet « masque ». La sortie d'un logiciel HTR peut donc se constituer tout simplement du texte brut bien qu'en un fichier hiérarchisé où le texte prédit et les données de la mise en page sont tous les deux alignés. En tout cas, la phase de transcription est essentielle dans un protocole de reconnaissance automatique du texte.

3.2.4 L'algèbre linéaire, les matrices, et l'intelligence artificielle

Tout logiciel HTR transforme les pixels d'un caractère en une matrice, telle que celle présentée dans la figure 3.7b. Et pourquoi ? À la base, l'ordinateur est une calculatrice.

15. Denis COQUENET, Clément CHATELAIN et Thierry PAQUET. « Handwritten Text Recognition : From Isolated Text Lines to Whole Documents ». In : *ORASIS 2021*. Saint Ferréol, France : Centre National de la Recherche Scientifique [CNRS], sept. 2021. URL : `https://hal.archives-ouvertes.fr/hal-03339648` (visité le 04/08/2022).

Les logiciels OCR et HTR décomposent une image numérique en représentations mathématiques. Les matrices permettent aux logiciels de faire des calculs probabilistes et de prédire quel caractère correspond à quelle représentation. Mais pour faire des prédictions, un logiciel a besoin d'une intelligence artificielle.



(a) Le masque de la lettre A,
sur l'image binarisée

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

(b) La matrice du masque de
la lettre A

FIGURE 3.7 – La matrice d'un caractère

Template matching

Pendant la dernière moitié du XX^{e} siècle, la reconnaissance du texte et l'intelligence artificielle ont connu des progrès importants. En 1950, il y a plus que soixante-dix ans,

Alan Turing a publié sa théorisation de l'IA [16]. Pendant les années soixante, au début de l'OCR, l'IA a rendu possible la prédiction du texte représenté par les matrices en faisant le lien entre deux entités : (1) la représentation qu'un logiciel a reconnue et (2) la représentation d'un caractère qu'il avait dans sa base de données. Cette technique initiale est le *template matching* ou le filtrage par modèle. Elle n'est pas le moyen le plus pratique. Les chercheurs V. K. Govindan et A. P. Shivaprasad l'ont revu en 1990, après qu'elle a été dépassée par le *feature analysis*.

> *[Template matching] directly compares an input character to a standard set of prototypes stored* [aux modèles stockés]. *The prototype that matches most closely provides recognition. […] This type of technique suffers from sensitivity to noise and is not adaptive to differences in writing style* [17].

Sous la dépendance du *template matching*, un logiciel OCR ne parviendrait pas à prédire un caractère si la totalité de sa représentation en matrice n'est pas suffisamment similaire au modèle stocké lors de la programmation du modèle.

Feature analysis

Les progrès dans le domaine de l'IA pendant les années 1970 et 1980 ont rendu possible le développement d'une nouvelle technique dont profite toujours les logiciels OCR, bien qu'elles aient été améliorées depuis. Au lieu de faire correspondre la représentation d'un caractère à une autre, les logiciels OCR décomposent un caractère en ses *features* ou ses aspects. En 1990, V. K. Govindan et A. P. Shivaprasad ont résumé l'état de cette technique.

> *The features may represent global and local properties of the characters. These include strokes, and bays in various directions, end points, intersections of line segments, loops […], and stroke relations, angular properties, sharp protrusions […]. These features have high tolerances to distortions and style variations, and also tolerate a certain degree of translation and rotation. However, the extraction processes are in general very complex and it is difficult to generate masks for these types of features* [18].

Ainsi, le logiciel OCR compare l'ensemble de certains aspects d'une représentation à l'ensemble des mêmes aspects auxquels le logiciel associe un caractère. Selon V. K. Govindan et A. P. Shivaprasad, un aspect (*feature*) peut être un trait, le croisement des lignes, une boucle, la relation entre plusieurs traits, la caractéristique des angles, ou une saillie.

16. A. M. Turing. « Computing Machinery and Intelligence ». In : *Mind* LIX.236 (1er oct. 1950), p. 433-460. ISSN : 0026-4423. DOI : 10.1093/mind/LIX.236.433. URL : https://doi.org/10.1093/mind/LIX.236.433 (visité le 04/08/2022).

17. V. K. Govindan et A. P. Shivaprasad. « Character Recognition — A Review ». In : *Pattern Recognition* 23.7 (1990), p. 671. ISSN : 0031-3203. URL : https://www.academia.edu/6986960/Character_recognition_A_review (visité le 05/08/2022).

18. Ibid.

Grâce à la technique de *feature analysis*, un logiciel OCR moderne parvient à faire une bonne prédiction même si certains aspects du caractère du document source ne sont pas présents dans la représentation du caractère que le modèle a apprise. Si un ensemble suffisamment robuste des aspects est reconnu dans le caractère, le modèle peut l'associer à un caractère qu'il a appris reconnaître. *L'Handwritten Text Recognition* utilise aussi du *feature analysis*. Mais au lieu de produire les métadonnées sur la police du caractère ainsi que la langue du texte, l'HTR analyse les aspects d'un caractère pour prédire simplement le caractère et le mot. Cependant, cette l'analyse HTR n'est pas plus simple que celle réalisée par un logiciel OCR.

Le logiciel HTR a besoin d'associer beaucoup d'aspects d'une variété immense à un seul caractère. Une même main peut écrire une même lettre de plusieurs façons, qui produit une variation « intra-classe », c'est-à-dire une variation importante dans les constituants de la classe d'une seule lettre [19]. En outre, l'écriture d'une main peut varier d'une page à une autre, et une page peut avoir plusieurs mains. Dit simplement, l'OCR analyse les aspects d'un caractère du point de vue d'une langue et d'une police attendue. L'HTR se focalise uniquement sur les aspects visuels d'un glyphe écrit, sans avoir besoin d'identifier la langue ni d'avoir appris la police du texte.

Deep learning

Aujourd'hui l'HTR profite du *feature analysis* ainsi que d'un développement plus récent dans l'intelligence artificielle qui s'appelle le *deep learning* ou l'apprentissage profond. Pouvant s'améliorer grâce aux réseaux neurones, un logiciel OCR ou HTR moderne peut prendre des décisions de plus en plus pertinentes quand il essaie de prédire du texte à partir de l'analyse des aspects (*feature analysis*). L'une des premières mises en œuvre des réseaux neurones pour *l'Handwritten Text Recognition* était le projet européen *Transkribus* qui a aussi mis au point l'OCR [20]. Un autre projet européen a suivi *Transkribus* et, contrairement au premier, laisse ses données et les architectures de ses modèles ouvertes : *Kraken* [21]. Élaboré dans l'esprit de la science ouverte, le projet *Gallic(orpor)a* a choisi d'utiliser *Kraken* et une interface de transcription, ouverte elle aussi, *eScriptorium*.

19. Une même lettre écrite de la même manière par plusieurs mains produit une variation « inter-classe ».

20. Guenter Muehlberger et al. « Transforming Scholarship in the Archives through Handwritten Text Recognition : Transkribus as a Case Study ». In : *Journal of Documentation* 75.5 (9 sept. 2019), p. 954-976. ISSN : 0022-0418. DOI : 10.1108/JD-07-2018-0114. URL : https://www.emerald.com/insight/content/doi/10.1108/JD-07-2018-0114/full/html (visité le 04/08/2022).

21. Benjamin Kiessling. « Kraken - an Universal Text Recognizer for the Humanities ». In : ADHO 2019 - Utrecht. 2019. URL : https://dh-abstracts.library.cmu.edu/works/9912 (visité le 10/08/2022).

3.3 Le modèle HTR

Puisque *Transkribus* et *Kraken* profitent tous les deux de l'apprentissage profond, les processus mis en œuvre par les interfaces graphiques *Transkribus* et *eScriptorium* ressemblent généralement à la même description. Ils commencent avec l'entraînement d'un modèle HTR. Ensuite, ils appliquent le modèle entraîné aux données d'entrées, c'est-à-dire une page numérisée. Le taux de réussite du processus est calculé en fonction du pourcentage des caractères que le modèle n'a jamais vus mais qui étaient bien prédits.

3.3.1 Les données d'entrée

Dans un premier temps, avant de penser à l'entraînement d'un modèle, il faut bien connaître la saisie des données. Les données d'entrée d'un modèle vont être les images numériques, composées des pixels. En général, leurs contenus textuels devraient être similaires afin que le modèle se spécialise dans une écriture particulière. Il est possible d'entraîner un modèle très généraliste, mais cela nécessite alors un très large corpus d'entraînement.

3.3.2 Les données d'entraînement

Le premier défi de l'entraînement d'un modèle HTR est la création des données d'entraînement. Ces données sont des transcriptions annotées et corrigées à la main à partir des images. L'ensembles des paires formées par une image et sa transcription s'appelle une vérité de terrain ou *ground truth* en anglais, puisque les transcriptions doivent être parfaites ou *vraies*[22]. Afin d'entraîner un modèle HTR, il faut un jeu des vérités de terrain. Alix Chagué décrit les vérités de terrain comme,

> « des ensembles de données annotées et corrigées de manière à fournir au modèle des paires composées d'une part d'une image ou d'une portion d'image (entrée) et d'autre part de l'annotation attendue (sortie), qui peut être des coordonnées dans le cas de la segmentation ou un ensemble de caractères pour la transcription » [23].

Lors de l'apprentissage, les vérités de terrain fournissent au modèle le texte attendu à partir d'un segment d'image pour qu'il puisse savoir comment s'améliorer afin de produire les bonnes prédictions ou les prédictions *vraies*. Les données générées reproduisent au plus près la prédiction idéale des vérités de terrain. Grâce à l'apprentissage profond, un

22. David Lassner et al. « Publishing an OCR Ground Truth Data Set for Reuse in an Unclear Copyright Setting. Two Case Studies with Legal and Technical Solutions to Enable a Collective OCR Ground Truth Data Set Effort ». Version 1.0. In : *Fabrikation von Erkenntnis – Experimente in den Digital Humanities. Hg. von Manuel Burghardt* Lisa Dieckmann (2021). Avec la coll. d'Herzog August Bibliothek, 5). doi : 10.17175/SB005_006. url : https://zfdg.de/sb005_006 (visité le 10/08/2022).

23. Chagué, Clérice et Romary, « HTR-United ».

modèle peut apprendre comment arriver à la prédiction souhaitée à partir de données d'entraînement.

3.3.3 L'entraînement

Lors de l'entraînement, le modèle HTR (comme un modèle TAL) s'évalue périodiquement et une dernière fois quand l'entraînement est terminié. La dernière évaluation s'appelle le *score*. Afin d'obtenir un meilleur *score*, on peut optimiser l'entraînement du modèle en jouant sur plusieurs paramètres.

Par exemple, on peut modifier le nombre de fois que le modèle révise sa manière de faire ses tâches de prédiction. Chaque essaie s'appelle une époque, dérivé de l'anglais *epoch*, et un plus grand nombre d'époque donne au modèle plus de chances de s'améliorer. Cependant, plus d'époque pèse plus sur le budget d'un projet puisqu'il consomme plus de puissance de calcul et prend plus de temps. En outre, on ne veut pas faire passer trop d'époque et risquer le sur-apprentissage d'un modèle, qui s'appelle le *overfitting* en anglais. Cela veut dire que la fonction prédictive du modèle s'est trop bien adaptée à ses données d'entraînement, y compris tout le bruit des données, et elle n'est pas suffisamment généraliste pour réussir sur des données que le modèle n'aurait jamais vues. On peut également régler la taille du pas d'apprentissage après chaque époque, c'est à dire son *learning rate*.

On peut aussi modifier la composition du jeu de données traité lors d'une époque. Ce dernier paramètre s'appelle un *batch size*, et il est aussi un entier, comme l'époque. Disons qu'on veut entraîner notre modèle sur 400 images. Une époque prendrait trop de temps et trop de puissance de calcul s'il essayait de traiter toutes les images de notre jeu de données au même temps. Du coup, on veut le diviser selon notre paramètre *batch size*. Si on déclarait un *batch size* de 100 images, on dirait au modèle qu'il faut itérer sur son jeu de données 4 fois afin de compléter une époque, en rappelant qu'une époque est égale à une passe complète sur le jeu de données d'entraînement, voir la figure 3.8.

Ce qu'on appelle l'erreur, soit la différence entre la prédiction du modèle et la vérité de terrain, se calcule à chaque itération d'un *batch* dans une époque. On veut que cette valeur diminue, c'est à dire que la prédiction du modèle se rapproche de plus en plus à la vérité de terrain. Pour optimiser le modèle, on peut choisir entre plusieurs fonctions pour calculer l'erreur, ce qu'on appelle une *loss function*. Dans l'exemple de la Figure 3.8, on pourrait appliquer une *loss function*, telle que le *mean squared error* (MSE), à toutes les prédictions d'un *batch* afin de connaître l'erreur ou le *loss* du modèle à ce point de l'entraînement.

Dans le cadre du projet *Gallic(orpor)a*, Ariane Pinche a entraîné des modèles HTR sur le corpus *gold* que l'équipe a fait à la main et qu'elle et Simon Gabay ont vérifié. A. Pinche l'a scindé en trois, comme la Figure 3.9 visualise. Une majorité des images

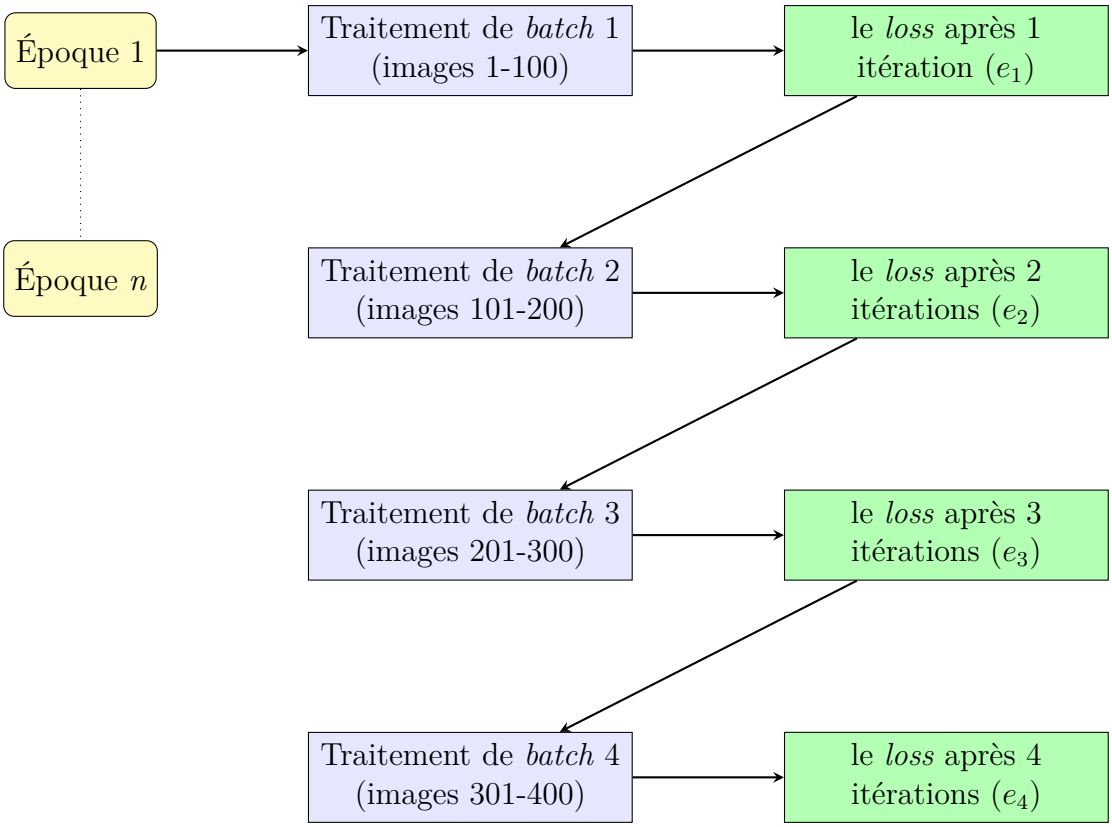


FIGURE 3.8 – La visualisation d'époque 1

(80%) compose le jeu d'entraînement (*training set*). Une petite partie (10%) compose le jeu de validation (*development set*). Et la dernière partie (10%) compose le jeu de test (*testing set*). Les données du *training set* sont ceux qui se divisent en *batches*; chacune est traitée une fois lors d'une époque. En prenant compte du taux de réussite du modèle sur les données du *training set*, le réseau neuronal fait des ajustements à la nature des connexions entre ses neurones, spécifiquement comment chaque connexion pèse sur le réseau. Après quelques ajustements, l'entraînement s'arrête pour évaluer le modèle en cours de développement. Les données du *development set* sont utilisées pour donner au réseau une évaluation impartiale de son succès puisqu'elles sont des données qu'il n'a pas pris en compte lors de l'ajustement des poids des connexions. Enfin, les données du *testing set* sont utilisées à la fin de l'entraînement pour donner un *score* final au modèle sur les données qu'il n'a jamais vues.

FIGURE 3.9 – La division du corpus gold pour l'entraînement d'un modèle

3.3.4 Le résultat de l'entraînement

Le résultat de l'entraînement s'appelle un modèle. L'un des buts du projet *Gallic(orpor)a* était de proposer pour tous les documents du XV^e^ au SIX^e^ siècle deux modèles généralistes. Un se spécialisera dans l'écriture des manuscrits et des incunables et l'autre sur les imprimés.

Deuxième partie

Exposition de la préparation et du travail d'analyse

Chapitre 4

Un pipeline visant à tout rassembler

Le pipeline du projet *Gallic(orpor)a* se déroule dans cinq étapes. Dans un premier temps, il récupère les fac-similés numériques des documents sources sur Gallica. Dans un deuxième temps, il applique des modèles HTR aux fac-similés ainsi téléchargés afin de produire une prédiction du texte et une transcription de la mise en page. Ensuite, il crée un fichier TEI préliminaire qui réunit les données produites par les modèles HTR et les métadonnées récupérées de plusieurs sources en ligne. Le quatrième étape enrichit le fichier TEI avec une analyse linguistique du texte de la transcription. Et enfin, le pipeline export les données du fichier TEI en divers formats, y compris les données conformant aux schémas RDF, DTS, et IIIF.

4.1 La récupération des fac-similés numériques

L'accès aux pages numérisées d'un document source est une condition essentielle à toute étape du pipeline de *Gallic(orpor)a*. Afin de transcrire le document source, il faut d'abord dire au logiciel comment il peut accéder au fac-similé numérique de la source. De plus, les images doivent être d'une qualité aussi bonne, sans être pourries par trop de bruit, que le logiciel HTR peut bien reconnaître le texte et les segments dedans. Pour terminer, bien qu'il ne soit pas essentiel à la transcription des pages numérisées, les images devraient s'associer aux métadonnées qui portent sur le document source. Grâce à une telle relation entre source numérique et source physique, le pipeline enrichit la ressource numérique qu'elle livre avec les informations portant sur le document source.

4.1.1 Archival Resource Key (ARK)

D'abord, avant de penser à récupérer les images, il faut mettre en place un système pour associer des pages, c'est-à-dire une suite d'images numériques, au document source, c'est-à-dire le concept d'une œuvre qui n'existe pas vraiment dans aucun format informatique. Pour associer plusieurs pages distinctes à un document, le pipeline profite

d'un identifiant unique qui s'appelle l'*Archival Resource Key* (ARK). Très utilisé dans le monde archivistique, cette clef indique une ressource, soit numérique soit physique, dont la conservation une institution, telle que la Bibliothèque nationale de France, se charge. Le schéma ARK est actuellement maintenu par la communauté ARK Alliance[1]. L'identifiant facilite la localisation d'une ressource, sans la confondre avec d'autres exemplaires de la même œuvre. La précision qu'accorde l'ARK améliore l'exactitude d'un processus de traitement automatique.

La mise en place d'un système de fichiers

Le pipeline se sert de l'identifiant ARK afin de gérer le lien entre les images numériques et le document source. La Figure 4.1 montre comment le système de fichiers du pipeline organise deux documents sources, où chacun a trois pages numérisées en format JPEG. Le chemin d'accès à chaque image se compose donc de l'identifiant ARK. Ainsi son association au document source est conservée et accessible au logiciel. Par exemple, le chemin de fichier du deuxième folio du premier document montré dans la Figure 4.1 est `data/ARK1/folio2.jpg`. En portant un tel chemin, l'image est toujours associé au bon document.

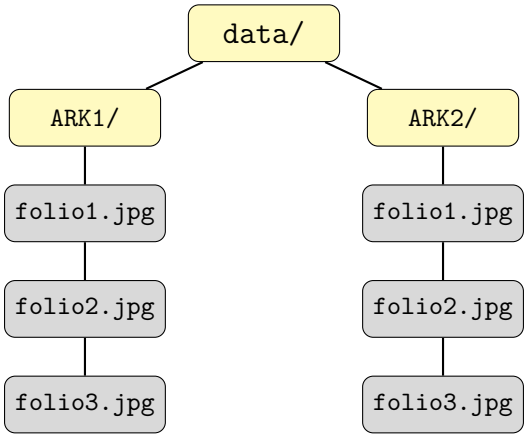


FIGURE 4.1 – Système de fichiers

4.1.2 International Image Interoperability Framework (IIIF)

En plus d'associer les images à leur document source, l'identifiant ARK sert aussi à récupérer les images depuis l'internet. Ce genre de requête se fait par un outil très utilisé qui s'appelle une API (*Application Programming Interface*). Une telle interface facilite la communication entre deux ordinateurs pour qu'ils puissent échanger d'information. Ainsi, un ordinateur peut demander aux serveurs de la Bibliothèque nationale de France les images d'un document source qu'ils conservent.

1. The ARK ALLIANCE. *Community*. ARK Alliance. URL : `https://arks.org/community/` (visité le 23/08/2022).

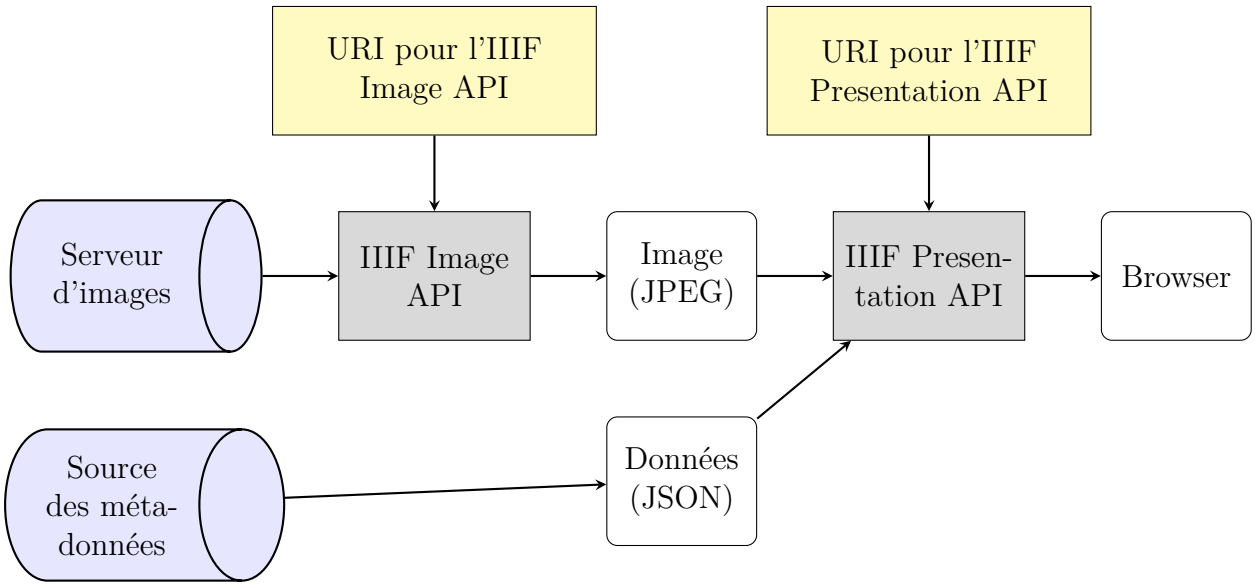
Il y a beaucoup de types d'API, chacune programmée pour parler certains langages de requête et pour communiquer certains types d'information. Une API destinée à la communication des données visuelles, surtout les fac-similés numériques du patrimoine, a été développée par *l'International Image Interoperability Framework* (IIIF). Le IIIF s'engage à standardiser la gestion des ressources visuelles mises en ligne avec le but d'améliorer l'exploitation et l'échange des ressources et de leurs métadonnées. Suite à une requête HTTP(S) standardisée, c'est-à-dire un URI (*Uniform Resource Identifier*) qui commence par « `https://` », l'IIIF Image API transfère les données visuelles requêtées vers l'ordinateur de l'expéditrice ou de l'expéditeur. L'URI se compose de quatre éléments génériques suivis par cinq éléments qui précisent (1) la région, (2) la taille, (3) la rotation, (4) la qualité, et (5) le format de l'image demandée.

scheme	`https://`
server	`gallica.bnf.fr`
prefix	`/iiif`
identifier	`/ark:12148/bpt6k1057726c`
folio	`/f1`
region	`/full`
size	`/full`
rotation	`/0`
quality	`/native`
format	`.jpg`

(a) URI pour l'IIIF Image API

scheme	`https://`
server	`gallica.bnf.fr`
prefix	`/iiif`
identifier	`/ark:12148/bpt6k1057726c`
manifest	`/manifest`
format	`.json`

(b) URI pour l'IIIF Presentation API



(c) IIIF Image API et IIIF Presentation API

FIGURE 4.2 – IIIF APIs

Puisque la BnF s'engage à l'initiative de l'IIIF, toute page numérisée qui se conserve

sur sa base de données Gallica peut être requêtée par l'URI que l'IIIF Image API attend. La Figure 4.2 montre comment construire les URIs afin d'acquérir les données visuelles du portail Gallica. Prenons le document source de l'identifiant ARK bpt6k1057726c, par exemple. Pour récupérer ses pages numérisées depuis la base de données Gallica, on envoie une requête à l'IIIF Image API ; l'exemple pour récupérer la totalité de la première page du document en format JPEG, sans rotation ni d'autre modification, se voit dans la Figure 4.2a. Comme la Figure 4.2c visualise, une telle requête HTTPS envoyée à l'IIIF Image API renverra un objet numérique de l'image. Afin d'accès aux métadonnées de l'image, on enverrait une requête d'une autre structure à la Presentation API, montrée dans la Figure 4.2b, qui renverrait des données en format JSON.

4.1.3 La mise en pratique

Dans le cadre du projet Artl@s en 2020, Caroline Corbières a développé un script pour importer à partir de l'IIIF Image API les pages d'un fac-similé numérique stocké sur les serveurs de la BnF [2]. Le projet *Gallic(orpor)a* a profité du travail de Corbières et l'a utilisé pour récupérer des fac-similés numériques ciblés par l'utilisatrice ou l'utilisateur du pipeline. En ajoutant une option au script (`-l`) je l'ai modifié afin de permettre qu'une utilisatrice ou utilisateur puisse limiter les nombres de pages récupérées. La limite évite le téléchargement de trop d'images ainsi qu'économise l'énergie dépensée. Ainsi, l'utilisatrice ou l'utilisateur peut tester la mise en œuvre des modèles HTR ou TAL sur un panel restreint des données.

4.2 L'application des modèles HTR

Le pipeline du projet *Gallic(orpor)a* vise à transcrire les ressources textuelles avec un modèle HTR qui convient bien au type du document, en rappelant qu'un modèle se spécialise dans une écriture particulière grâce à son entraînement. Cela exige donc que le pipeline se dispose des modèles entraînés et qu'il prend sa décision automatiquement sur quel modèle convient auquel document traité. Lors de l'installation du pipeline, l'utilisatrice ou l'utilisateur fournit des modèles à mettre en œuvre [3]. En outre, l'utilisatrice ou l'utilisateur désigne un modèle de segmentation et celui de HTR par défaut, que le pipeline utiliserait s'il ne pensait pas que les autres modèles installés conviennent bien au

2. Simon Gabay et al. « Automating Artl@s – Extracting Data from Exhibition Catalogues ». In : *EADH 2021 - Second International Conference of the European Association for Digital Humanities*. Krasnoyarsk, Russia, sept. 2021. URL : `https://hal.archives-ouvertes.fr/hal-03331838` (visité le 23/08/2022).

3. Le pipeline peut s'adapter aux divers modèles et mettre en œuvre tout modèle HTR qu'une utilisatrice ou un utilisateur lui donne lors de l'installation du pipeline s'il est basé sur l'engin *Kraken*. En s'adaptant aux modèles publiés ainsi qu'aux modèles toujours en cours qu'une utilisatrice ou un utilisateur lui donne directement, selon sa configuration pendant l'installation, le pipeline facilite les expériences scientifiques sur plusieurs corpus.

document qu'il est en train de traiter. En plus de mettre en pratique des modèles HTR, le projet *Gallic(orpor)a* vise à en construire des nouveaux.

4.2.1 L'entraînement des modèles

Dans le cadre du projet *Gallic(orpor)a*, l'équipe a entraîné des nouveaux modèles généralistes, l'un pour les manuscrits et les incunables en français, l'autre pour les imprimés françaises créées avant la révolution. Ces modèles ont besoin des vérités de terrain produites lors du projet *Gallic(orpor)a*. Bien que le pipeline ne refasse pas la création de ces vérités de terrain, ni de l'entraînement des modèles, il profite de cet objectif parallèle du projet puisque le pipeline a besoin de modèles très performants.

La création des vérités de terrain

Lors de la première moitié de 2022, l'équipe du projet *Gallic(orpor)a* se composait des vacataires qui se chargeaient de la création des vérités de terrain pour les nouveaux modèles HTR. Les chefs du projet Simon Gabay et Ariane Pinche ont sélectionné un corpus de documents à transcrire. Les membres de l'équipe se spécialisaient dans certains types de document présents sur la liste, tel qu'un incunable du Moyen Âge. Un vacataire prend l'un des documents du corpus et saisit son identifiant ARK dans l'interface de transcription *eScriptorium*, dont je parle dans le chapitre 3. L'interface imite ce que le pipeline *Gallic(orpor)a* fait, ainsi que le script de Corbières, en téléchargeant les images du document depuis l'IIIF Image API de la BnF.

L'interface *eScriptorium* facilite la segmentation et la transcription des pages. Les vacataires l'utilisent pour segmenter les zones de la page et y mettre les étiquettes conformant au vocabulaire *SegmOnto*. Portant les étiquettes de *SegmOnto*, les transcriptions aident à entraîner les modèles avec un vocabulaire cohérent pour tout type de document, y compris les manuscrits et les imprimés. L'interface *eScriptorium* export les vérités de terrain dans un fichier XML ALTO avec des images téléchargés en format JPEG ou PNG.

4.2.2 La sélection des modèles

Dans l'intérêt de maximiser l'automatisation du pipeline, la mise en œuvre d'un modèle s'effectue automatiquement, selon une analyse des métadonnées du document à traiter. J'ai réalisé l'attribution automatique des bons modèles aux documents grâce à deux idées : le modèle par défaut et le modèle idéal. Dans un premier temps, le pipeline récupère les métadonnées IIIF du fac-similé numérique qui portent sur sa langue et sa date de création. Ensuite, j'ai cherché dans les modèles installés avec le pipeline s'il y en a un qui a été entraîné sur les données de la même langue et du même siècle. S'il n'y en a pas, le pipeline utilise son modèle par défaut. Soit l'utilisatrice ou l'utilisateur du pipeline

déclare le modèle par défaut selon ses besoins, soit le modèle par défaut programmé en dur dans le pipeline est utilisé, au cas où l'utilisatrice ou l'utilisateur n'en précise aucun. Disposé de l'attribution automatique du modèle, le pipeline peut traiter beaucoup de documents d'une diachronie longue, sans besoin d'intervention d'une personne à chaque application du modèle HTR.

4.2.3 La sortie des modèles segmentation et HTR

Les modèles HTR basés sur *Kraken* produisent des fichiers XML qui conforment soit au schéma ALTO, soit au schéma PAGE. Les deux schémas sont bien adaptés à l'encodage des données portant sur la mise en page et sur la transcription d'un texte. L'équipe de *Gallic(orpor)a* a privilégié le schéma ALTO puisqu'il accorde plus de détail sur la mise en page. La Bibliothèque nationale de France le décrit comme comme, « un schéma XML standardisé, qui permet de stocker les informations relatives à la structure physique et au texte extrait par OCR » [4]. Par rapport au schéma PAGES, le schéma ALTO conserve plus de données produites par le modèle de segmentation, tout en gardant leur relation aux données textuelles produites par le modèle HTR.

4.2.4 Le schéma ALTO

Le schéma ALTO a été développé dans le cadre du projet METS (*Metadata Encoding and Transmission Schema*). Le dernier date de 2001 [5]. Comme résume la Bibliothèque nationale de France,

> METS renseigne alors sur la structure logique de la page (nature sémantique des blocs de texte, par exemple titre, partie d'article, légende d'illustration, etc.), tandis qu'ALTO localise des contenants (blocs, lignes, etc.) sur la matrice de l'image de la page [6].

Le schéma ALTO vise donc à renseigner sur la mise en page ainsi que sur le texte d'une page. Certains de ses éléments, tel que l'élément `<polygon>`, précisent les coordonnées d'une région ou d'une zone sur la page qu'avait prédite un modèle de segmentation. Cependant, certains attributs, tel que l'attribut `@CONTENT`, sont associés à la prédiction du texte. Prenons l'exemple montré dans la Figure 4.3 où le schéma ALTO encode une phrase incomplète : *oyseaux lesquelz ie esperoye pren.* Les données imbriquées dans l'élément `<TextLine>` parviennent à porter sur la mise en page de la ligne ainsi que sur son texte. La puissance du schéma ALTO est sa manière de combiner ses deux types d'information.

4. Bertrand Caron. *Formats de Données Pour La Préservation à Long Terme : La Politique de La BnF*. Technical Report. Bibliothèque Nationale de France (Paris), oct. 2021. URL : `https://hal-bnf.archives-ouvertes.fr/hal-03374030` (visité le 23/08/2022), p. 75.

5. *METS : Metadata Encoding and Transmission Standard*. BnF - Site institutionnel. URL : `https://www.bnf.fr/fr/mets-metadata-encoding-and-transmission-standard` (visité le 23/08/2022).

6. Caron, *Formats de Données Pour La Préservation à Long Terme*, p. 75.

Dans la Figure 4.3, les quatre points du rectangle qui encadre la ligne de texte sont données par les attributs de l'élément `<TextLine>` : `@HPOS`, `@VPOS`, `@WIDTH`, `@HEIGHT`. Puisque le modèle de segmentation a aussi encadré la ligne de texte dans un polygone, tous les points de l'objet sont données par l'attribut `@POINTS` de l'élément `<Polygon>`. Le schéma ALTO renseigne aussi sur le *baseline* de la ligne de texte en donnant les coordonnées x,y du début et de la fin dans l'attribut `@BASELINE`. En plus de ces données structurelles, le schéma permet d'encoder les données textuelles dans l'attribut `@CONTENT` de l'élément `<String>`, où se trouve la ligne de texte *oyseaux lesquelz ie esperoye pren.*

```
<TextLine ID="eSc_line_1ed06324"
    TAGREFS="LT825"
    BASELINE="1193 982 2263 969"
    HPOS="1184"
    VPOS="877"
    WIDTH="1079"
    HEIGHT="127">
    <Shape>
        <Polygon POINTS="1193 982 1184 903 1303 877 1307 877 2255 890 2263
   969 2263 995 1193 1004"/>
    </Shape>
    <String
        CONTENT="oyseaux lesquelz ie esperoye pren"
        HPOS="1184"
        VPOS="877"
        WIDTH="1079"
        HEIGHT="127">
    </String>
</TextLine>
```

FIGURE 4.3 – L'encodage d'une ligne de texte en ALTO

L'encodage vu dans la Figure 4.3 ne représente pas tout manière d'encoder une ligne de texte possible dans le schéma ALTO. Le schéma peut se réaliser avec plusieurs niveaux de détail. Par exemple, le contenu de la ligne de texte peut être désagrégé au point que chaque segment du texte se trouve dans son propre élément `<String>` descendant du `<TextLine>`. Comme cela, les coordonnées de chaque mot peut être précisées. Ce format plus détaillé est celui produit depuis la ligne de commande (CLI, ou *Command Line Interface*) de *Kraken*[7]. Le format montré par la Figure 4.3, le moins détaillé, est celui utilisé par l'interface *eScriptorium*. L'un des défis du pipeline est de s'adapter aux divers formats d'ALTO qu'un logiciel HTR pourrait produire.

7. En traitant les fichiers XML ALTO sortis de la ligne de commande, il doit reconstruire le contenu de la ligne de texte à partir de plusieurs éléments `<String>` qui représentent les mots dans une ligne de texte, au lieu de la ligne de texte elle-même.

4.3 Réunir la transcription et les métadonnées

Jusqu'ici, le pipeline a récupéré les images numériques en format JPEG ou PNG depuis l'IIIF Image API et les a traité avec les modèles HTR en produisant des fichiers XML ALTO. Ensuite, le pipeline recherchera les métadonnées en plusieurs formats, y compris JSON, XML, et HTML, depuis l'internet. Avec une telle diversité de structures de données, un format robuste est exigé pour tout rassembler et le mettre en ordre. L'équipe de *Gallic(orpor)a* a choisi de s'appuyer sur un fichier XML, où les données sont imbriquées dans une façon hiérarchisée, mais du schéma TEI. Contrairement au schéma ALTO, le schéma TEI se spécialise dans l'édition numérique du texte. La communauté du TEI décrit l'objectif du projet ainsi :

> *The Text Encoding Initiative (TEI) is a consortium which collectively develops and maintains a standard for the representation of texts in digital form. Its chief deliverable is a set of Guidelines which specify encoding methods for machine-readable texts, chiefly in the humanities, social sciences and linguistics*[8].

En plus de la représentation numérique du texte, la TEI dispose des éléments XML qui conviennent bien à la représentation de la mise en page. La TEI est donc un format idéal pour fusionner les transcriptions sorties des modèles HTR, les métadonnées récupérées des sources en ligne, et le texte extrait des transcriptions. En plus, la TEI est un format très utilisé et beaucoup d'outils numériques s'y adaptent déjà.

4.3.1 L'extrait des données des fichiers ALTO

Les données sorties du traitement HTR sont du schéma XML ALTO mais le pipeline *Gallic(orpor)a* veut se terminer par un document TEI. Il faut donc transformer la sortie des modèles HTR en le format souhaité en ajoutant des métadonnées récupérées depuis plusieurs sources de données externes. En plus de mettre en œuvre un prototype du pipeline entier, je me suis chargée de la modélisation d'ALTO à TEI. Le livrable principal de mon stage est l'application `alto2tei`[9].

Nous de l'équipe de *Gallic(orpor)a* avons conclu que l'élément XML TEI `<sourceDoc>` convient bien à l'encodage de toute donnée des modèles HTR portant sur le texte prédit et de la mise en page. Des autres chercheurs, tel que Hugo Scheithauer, Alix Chagué, et Laurent Romary, ont arrivés à la même conclusion[10]. Les *guidelines* de la Text Encoding Initiative définissent l'élément `<sourceDoc>` ainsi :

8. *TEI : Text Encoding Initiative*. URL : `https://tei-c.org/` (visité le 24/08/2022).

9. Kelly Christensen. *Gallicorpora/Application*. URL : `https://github.com/Gallicorpora/application` (visité le 21/08/2022).

10. Hugo Scheithauer, Alix Chagué et Laurent Romary. « From eScriptorium to TEI Publisher ». In : *Brace Your Digital Scholarly Edition !* Berlin, France, nov. 2021. URL : `https://hal.inria.fr/hal-03538115` (visité le 23/08/2022).

> `<sourceDoc>` *contains a transcription or other representation of a single source document potentially forming part of a dossier génétique or collection of sources*[11].

Les données structurelles et textuelles de la page numérisée sont balisés dans les éléments XML descendant du `<sourceDoc>`. Ces choix sont décrits et justifiés dans le chapitre 5.

4.3.2 Le récupération des métadonnées

En plus de transformer les données des fichiers ALTO, le pipeline récupère les métadonnées portant sur le fac-similé numérique, le document source physique qui a occasionné la création du fac-similé, ainsi que la ressource numérique sortie du pipeline lui-même. Toutes ses métadonnées sont encodées dans l'élément XML TEI `<teiHeader>`. Comment réussit le pipeline à récupérer les métadonnées qui ne sont pas données par les fichiers ALTO ? Avec l'identifiant ARK, qu'il prend du système de fichiers, le pipeline récupère dans un premier temps les métadonnées du fac-similé numérique, dont les images les modèles HTR ont traité. Même pour les fac-similés hébergés par divers institutions, le pipeline récupère les métadonnées diffusées directement par l'API IIIF..

Créée dans le cadre du projet *Gallic(orpor)a*, l'application `alto2tei` récupère les métadonnées qui portent sur le document source physique grâce à une relation qui se donne entre le fac-similé numérique sur Gallica et le document source physique conservé par la BnF. L'application demande cette relation de l'API IIIF que la BnF met à disposition. Si elle est bonne, l'application passe ensuite par l'API SRU de la BnF. Ce genre d'API renvoie des données hiérarchisées dans un schéma XML. En anglais, son nom veut dire *Search/Retrieve via URL* (SRU). Enfin, le pipeline recherche encore des métadonnées dans le catalogue du Système Universitaire de Documentation(SUDOC) qui portent sur les institutions patrimoniales en France qui hébergent des documents sources.

4.3.3 La construction du document préliminaire TEI

La première étape de la construction du document TEI est la récupération des données, à la fois depuis les fichiers XML TEI et les sources de données externes. Les données de la transcription sont représentées dans l'élément `<sourceDoc>` du document TEI, les métadonnées dans son élément `<teiHeader>`. Ensuite, la deuxième étape est le traitement des données ainsi extraites et nettoyées des fichiers ALTO. D'abord, le pipeline extrait toute ligne de texte et la balise dans le bon élément TEI intégré dans le `<body>`.

11. *TEI element sourceDoc.* URL : `https://tei-c.org/release/doc/tei-p5-doc/en/html/ref-sourceDoc.html` (visité le 23/08/2022).

4.4 L'analyse linguistique et le fichier final

L'avant dernier étape du pipeline est d'analyser le texte récupéré et intégré dans le `<body>`. D'abord, le pipeline colle des lignes de texte en des phrases complètes. Ensuite, il met en œuvre des modèles TAL pour tokéniser et analyser le texte. La tokénisation divise la phrase en composants (*tokens* ou lexèmes). Une autre analyse linguistique, la lemmatisation traite les *tokens* ainsi distingués. Tout *token* pourrait ensuite être normalisé par un modèle TAL qui transforme le mot prédit par le modèle HTR en sa version normalisée. Par exemple, un modèle TAL de normalisation peut transformer le mot prédit *nostre* en le mot normalisé *nôtre*, comme montré par les expériences de Rachel Bawden et son modèle FREEMnorm [12]. Enfin, un modèle TAL NER (*Named-Entity Recognition*) peut encore traiter le texte pour reconnaître les entités nommées, tel qu'une personne ou un lieu. Le pipeline met en œuvre plusieurs modèles TAL afin d'analyser ses divers aspects linguistiques du texte.

4.4.1 L'ODD (One Document Does it all

Un fichier TEI se soumettent aux règles qui assurent l'uniformité. Afin de mettre en pratique les traitements à l'échelle pour tout document produit par le pipeline, il faut que tout élément TEI soit utilisé de la même manière. Pour parvenir à une telle régularité, la TEI dispose d'un document qui applique des règles personnalisées au produit souhaité du pipeline. Ce document se connaît par son acronyme ODD, qui veut dire *One Document Does it all* ou un fichier qui tout fait. Dans le cadre du stage, j'ai aidé à la rédaction d'un ODD qui explique en détail tout aspect du document TEI sorti du pipeline.

4.5 L'exploitation des données

Enfin, les données encodées dans le fichier TEI enrichi et final peuvent être exploitées en tant qu'un fichier XML TEI ainsi que dans divers formats secondaires. Le logiciel TEI Publisher, par exemple, peut aisément publier un fichier XML TEI et permettre les utilisateurs de naviguer les données dans un visionneur de l'édition en ligne [13]. Le document TEI peut aussi être exploité par les fichiers de transformation XSL. Il y a plusieurs formats secondaires qui pourraient servir à l'exploitation des données encodées dans le fichier XML TEI.

L'équipe envisage trois formats secondaires par lesquels les données produites par le pipeline peuvent être exploitées. L'un de ses formats est l'IIIF, le même qui a permis la construction du document TEI dans un premier temps. Tout attribut `@source` dans

12. Bawden et al., « Automatic Normalisation of Early Modern French ».

13. *Eeditiones/Tei-Publisher-App*. e-editiones.org. URL : `https://github.com/eeditiones/tei-publisher-app` (visité le 11/09/2022).

le `<sourceDoc>` du document TEI contient un URI qui permet de visionner la partie de l'image concernée. Par exemple, si l'attribut `@source` descend d'un élément `<zone>` qui porte sur une ligne de texte, sa valeur donnerait dans un browser ou dans un visionneur uniquement la partie de l'image source qui contient cette ligne de texte. Ainsi, les données du fichier XML TEI final peuvent être exploitées afin de visionner les bloques de textes, lignes de textes, les mots, et les caractères transcrits. Les deux autres formats secondaires profitent plutôt des métadonnées du fichier TEI ; ils sont le DTS (Distributed Text Services) et le RDF (Resource Description Framework).

Chapitre 5

L'analyse des structures des données XML

Afin d'extraire et transformer en TEI les données des fichiers XML ALTO, produits par les modèles HTR, il faut d'abord bien connaître le format des données d'entrée. En outre, afin de modéliser et justifier la transformation d'ALTO à TEI, il faut bien connaître les possibilités du dernier schéma. L'objectif de ce chapitre est d'expliquer les normes de ces deux structures de données fondamentales au projet *Gallic(orpor)a*.

5.1 ALTO : *Analyzed Layout and Text Object*

Qu'est-ce qu'est l'ALTO ? Le format XML ALTO a originellement été développé par le projet européen METAe en 2003. Le projet s'occupait du développement de logiciels destinés aux institutions patrimoniales. Le but de tels logiciels était de faciliter l'extrait à partir des pages numérisées des informations portant sur la mise en page et sur la logique d'un document texte numérisé. En 2003, les logiciels OCR étaient déjà répandus : l'enjeu était alors d'élaborer un schéma de données qui reproduirait le contenu textuel, ainsi que sa mise en forme structurelle. Là où un logiciel OCR ne faisait que reconnaitre le texte d'un titre et de son sous-titre, le nouveau format visait à distinguer les deux lignes de texte pour ensuite les hiérarchiser, en signalant que le sous-titre est subordonné au titre.

Le projet METAe a donc développé le format METS (*Metadata Encoding and Transmission Standard*) qui avait pour but de prendre le texte et la mise en page prédits par un logiciel OCR et augmenter ces données avec la logique du document. Par exemple, les lignes de texte d'une page et celles d'une autre page seraient imbriquées dans de différents éléments XML, afin de distinguer entre les deux suites de lignes de texte. Le schéma METS a réussi à combiner les métadonnées de la ressource numérique, ainsi que de le texte transcrit avec ses données structurelles et textuelles grâce au format hiérarchisé de l'XML.

Mais le format primordial METS n'avait pas répondu à la question de comment bien structurer les données dans une page, telle que la ligne d'un titre et la ligne d'une paragraphe sur la même page. Les créateurs du format ALTO ont décrit son prédécesseur comme un « emballage » (*wrapper*) pour la structure de données ALTO [1]. Tandis que le format METS organise les métadonnées et la logique du document, telles que l'occurence et l'ordre des pages, le format ALTO décrit la logique et la transcription produite de chaque page numérisée.

Normalement, un fichier XML ALTO décrit une page (ou une image) d'un document. Mais, depuis sa première version publiée en 2003, l'élément `<Layout>` peut en fait contenir plusieurs éléments `<Page>` [2]. Un exemple de la structure de la première version est visualisée dans la Figure 5.1. Malgré la possibilité d'en contenir plusieurs dans le `<Layout>`, la plupart de logiciels HTR qui utilisent le format ALTO crée un fichier par page numérisée et donc ne met qu'un `<Page>` dans le `<Layout>`.

```
<Layout>
  <Page ID="XXX" PHYSICAL_IMG_NR="000" HEIGHT="000" WIDTH="000" STYLEREFS="
    XXX">
    <PrintSpace ID="XXX" HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000">
      <TextBlock ID="XXX" HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000">
        <TextLine ID="XXX" HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000">
          <String ID="XXX" HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000"
    CONTENT="XXX"/>
          <Sp ID="XXX" HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000"/>
          <String ID="XXX" HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000"
    CONTENT="XXX"/>
        </TextLine>
        <TextLine ID="XXX" HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000">
          <String ID="XXX" HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000"
    CONTENT="XXX"/>
          <Sp ID="XXX" HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000"/>
          <String ID="XXX" HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000"
    CONTENT="XXX"/>
        </TextLine>
      </TextBlock>
    </PrintSpace>
  </Page>
  <Page ID="XXX" PHYSICAL_IMG_NR="000" HEIGHT="000" WIDTH="000" STYLEREFS="
    XXX">
    <PrintSpace ID="XXX" HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000">
<!-- ... -->
</Layout>
```

FIGURE 5.1 – La structure ALTO version 1, circa 2003

Le schéma ALTO est actuellement dans sa quatrième version, mais la structure

1. Birgit STEHNO, Alexander EGGER et Gregor RETTI. « METAe–Automated Encoding of Digitized Texts ». In : *Literary and Linguistic Computing* 18.1 (1er avr. 2003), p. 77-88. ISSN : 0268-1145, 1477-4615. DOI : 10.1093/llc/18.1.77. URL : https://academic.oup.com/dsh/article-lookup/doi/10.1093/llc/18.1.77 (visité le 24/08/2022), p. 81.
2. Ibid.

actuelle ressemble toujours au modèle qu'ont présenté les auteurs Birgit Stephno, Alexader Egger, et Gregor Retti en 2003. Dans sa première version, montrée dans la Figure 5.1, les éléments les plus petits étaient les segments de texte (`<String>`) et les espaces entre mots (`<Sp>`), balisés dans une ligne de texte (`<TextLine>`) qui appartient à un bloque de texte (`<TextBlock>`). Tous ces éléments XML portent un identifiant unique (`@ID`) et quatre coordonnées portant sur le rectangle dans lequel est contenu l'élément. Le contenu textuel est représenté par l'attribut `@CONTENT` de l'élément `<String>`.

5.1.1 La structure actuelle des fichiers XML ALTO

Aujourd'hui, le plus petit élément d'une structure de données ALTO est le glyphe (`<Glyph>`), au lieu d'une chaîne de caractères (`<String>`). Par conséquent, dans le nouveau format, le contenu textuel est représenté deux fois, une fois comme l'attribut `@CONTENT` de l'élément classique `<String>` et une deuxième comme le même attribut de l'élément `<Glyph>`. Une comparaison entre les deux arborescences est représentée dans la Figure 5.3. Comme montre la sous-figure 5.3b, la nouvelle architecture permet d'aller en plus de détail. Certains logiciels, tel que l'interface *eScriptorium*, produisent toujours les fichiers ALTO avec une variation de l'ancienne structure où l'élément `<String>` n'est pas répétable et représente le contenu textuel de la ligne.

En général, toute donnée portant sur la mise en page se compose de quatre coordonnées qui, ensemble, tracent un rectangle. Les valeurs des attributs `@HPOS` et `@VPOS` font les coordonnées x,y du point le plus haut à gauche du rectangle, comme se voit dans la Figure 5.2. La valeur de l'attribut `@HEIGHT` compte la différence entre la coordonnée y du point le plus haut et la coordonnée y du point le plus bas. La valeur de l'attribut `@WIDTH` calcule aussi la différence entre le côté gauche du carré et son côté droit. Ces quatre attributs sont attribués aux éléments `<PrintSpace>`, `<TextBlock>`, `<TextLine>`, `<String>`, `<Sp>`, et `<Glyph>`.

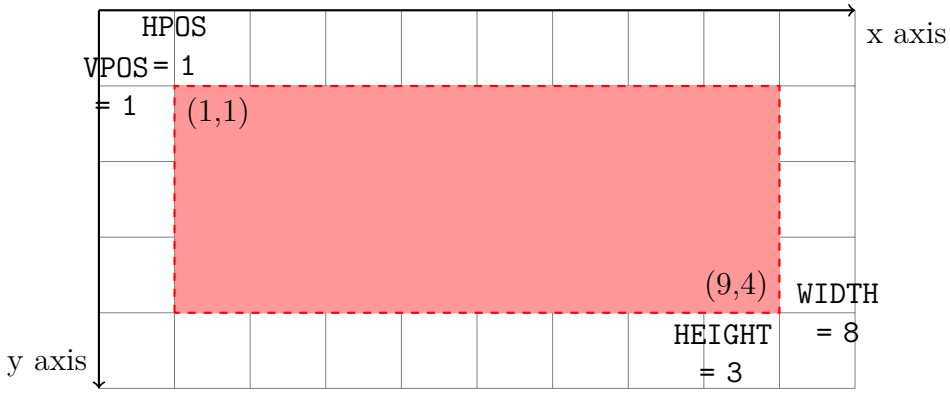
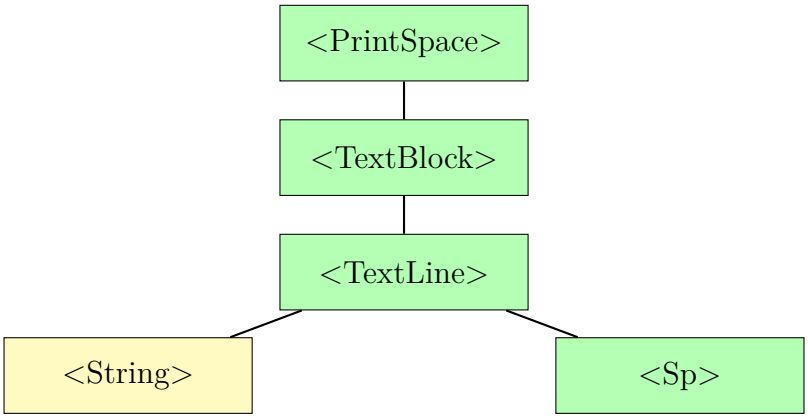


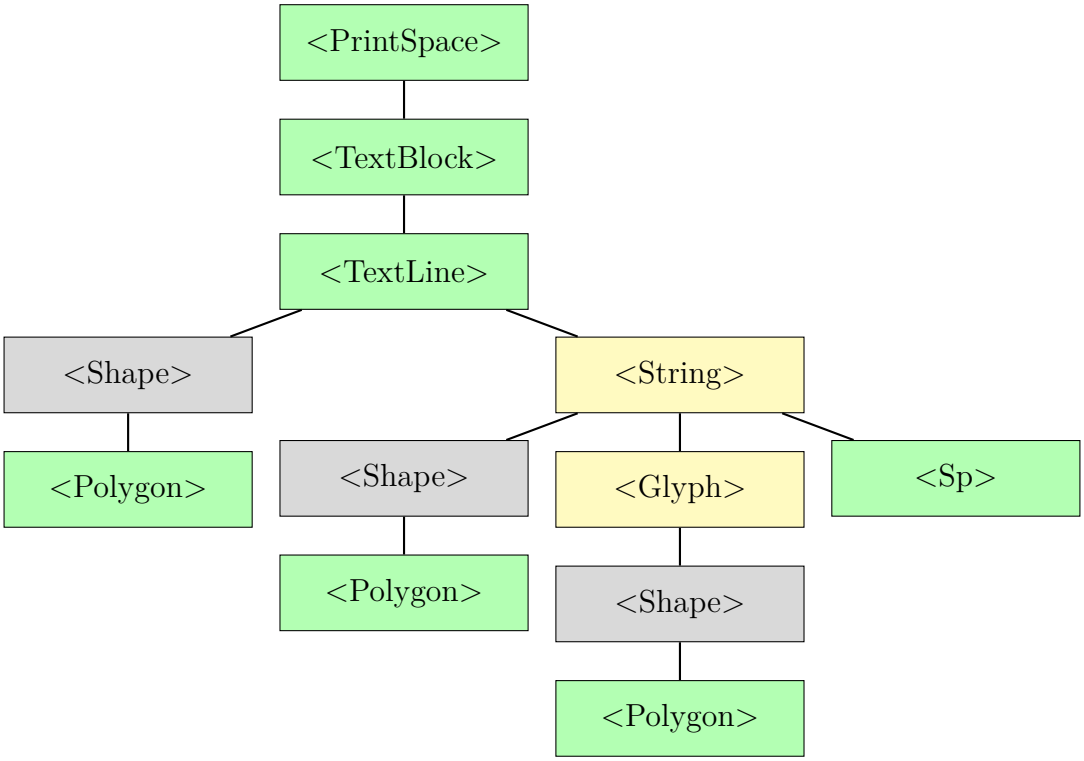
FIGURE 5.2 – Les coordonnées d'un masque en rectangle

L'arborescence actuelle du format ALTO diffère de l'original car, aujourd'hui, elle peut définir les coordonnées d'un polygone en plus d'un rectangle. Cela est un dévelop-

pement dans la technologie des logiciels OCR et HTR. La visualisation dans la sous-figure 5.3b montre le nouvel élément `<Polygon>`. Le `<Polygon>` descend directement de l'élément `<Shape>`[3]. Parmi les deux structures, le contenu textuel est toujours balisé dans l'élément `<String>`. Mais en allant en plus détail, la nouvelle arborescence montre le contenu textuel dans l'élément `<Glyph>`, qui représente tout caractère ou signe de ponctuation composant un segment (`<String>`) de la ligne de texte. La Figure 5.3 indique tout élément qui contient du texte en jaune.



(a) Ancienne structure



(b) Nouvelle structure

FIGURE 5.3 – Modélisation des formats ALTO

3. L'élément intermédiaire `<Shape>` ne porte pas d'attribut ni d'autre information. Sa raison d'être est de baliser l'élément `<Polygon>` qui pourrait, théoriquement, peut importe quel forme dans laquelle encadre l'objet de texte. Cet élément est indiqué en gris dans la sous-figure 5.3b.

Certains attributs de l'arborescence moderne, tel que l'attribut `@BASELINE` de l'élément `<TextLine>` et l'attribut `@POINTS` de tout élément `<Polygon>`, prennent comme valeur une chaîne d'entiers. Les entiers sont séparés par une espace, bien qu'ils soient des pairs de x,y indiquant une coordonnée. Voir la Figure 5.4. Le polygone peut avoir plusieurs points (`@POINTS`) tout le long de son périmètre (ligne 7, Fig. 5.4). Par contre, la *baseline* d'une ligne de texte (`@BASELINE`) compte toujours quatre entiers puisqu'il n'a qu'un début et une fin, donc deux paires x,y (ligne 5, Fig. 5.4).

```
1  <Layout>
2    <Page ID="XXX" PHYSICAL_IMG_NR="000" WIDTH="000" HEIGHT="000">
3      <PrintSpace HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000">
4        <TextBlock ID="XXX">
5          <TextLine ID="XXX" HPOS="000" VPOS="000" HEIGHT="000" WIDTH="000"
     BASELINE="1 2 3 4">
6            <Shape>
7              <Polygon POINTS="1 2 3 4 5 6 7 8"/>
8            </Shape>
9            <String ID="XXX" CONTENT="AB" HPOS="000" VPOS="000" WIDTH="000"
     HEIGHT="000" WC="1.0">
10             <Shape>
11               <Polygon POINTS="1 2 3 4 5 6 7 8 9 10"/>
12             </Shape>
13             <Glyph ID="XXX" CONTENT="A" HPOS="000" VPOS="000" WIDTH="000"
     HEIGHT="000" GC="1.0">
14               <Shape>
15                 <Polygon POINTS="1 2 3 4 5 6 7 8"/>
16               </Shape>
17             </Glyph>
18             <Glyph ID="XXX" CONTENT="B" HPOS="000" VPOS="000" WIDTH="000"
     HEIGHT="000" GC="1.0">
19               <Shape>
20                 <Polygon POINTS="1 2 3 4 5 6 7 8"/>
21               </Shape>
22             </Glyph>
23           </String>
24         </TextLine>
25 <!-- ... -->
26       </TextBlock>
27     </PrintSpace>
28   </Page>
29 </Layout>
```

FIGURE 5.4 – La structure ALTO version 4, circa 2022

5.2 TEI : *Text Encoding Initiative*

La raison pour laquelle la TEI a été choisie pour fusionner toute donnée du pipeline *Gallic(orpor)a* est parce qu'elle est un format XML souple, qui peut s'adapter facilement à plusieurs types de document et d'édition numérique. Cela veut dire qu'une exposition

détaillée du schéma TEI n'est pas possible : la manière d'encoder une ligne de texte n'est notamment pas aussi fixée que celle du schéma ALTO.

Tandis qu'une ligne de texte dans un fichier XML ALTO est balisée dans l'élément `<TextLine>`, une ligne de texte dans un document TEI peut être encodée de plusieurs façons. Elle peut suivre l'élément vide `<lb/>` ou elle peut apparaître à côté d'autres lignes de texte, toutes balisées ensemble à l'intérieur d'un élément tel que `<p>` ou `<div>`. De plus, parce que la TEI permet de classer les composants d'une ligne de texte selon la logique ou le genre du document, certains mots ou phrases peuvent être balisés dans d'autres éléments, tel que l'élément `<date>` pour encoder une année. Dans le même ordre des idées, une ligne de texte peut aussi être balisée par des éléments qui expliquent sa fonction dans le document. Par exemple, une ligne de texte peut être un item dans une liste (`<item>`) ou la salutation à la fin d'une lettre (`<salute>`).

Prenant l'exemple d'une lettre, la Figure 5.5 présente l'encodage d'une salutation dans les deux schémas, ALTO et TEI. On voit que l'ALTO excelle à préciser l'emplacement des mots (et des caractères) sur la page d'un document. Mais mise à part la reconnaissance de la lettre, dont la certitude du modèle se représente par l'attribut `@WC` (*word certainty*), le schéma ALTO ne donne pas d'autre information. L'encodage dans le format TEI, par contre, enrichit la ligne de texte avec beaucoup d'information. Grâce à l'élément TEI `<salute>` on sait que la ligne de texte est la salutation d'une lettre ou quelque autre forme de communication. De plus, l'encodage s'appuie sur la date en-tête pour attribuer au mot *demain* une date précise qui est encodée dans l'attribut `@when`. Enfin, le schéma TEI dispose d'un système pour réunir les occurrences du même concept dans un texte, tel qu'une personne. L'encodage dans la Figure 5.5c utilise l'attribut `@ref` pour dire que l'occurence du mot *chérie* fait référence à une personne à laquelle a été donnée, dans les métadonnées du document TEI, l'identifiant `"Kelly"` [4].

5.2.1 Qu'est-ce qu'est la TEI ?

Comme montrent les exemples de la Figure 5.5, le schéma TEI est spécialisé dans la représentation d'un texte et en vue de son édition numérique. Il facilite l'enrichissement du texte avec les métadonnées, telles que les références internes dans le document, ainsi que la classification de la nature d'un mot ou d'une phrase. Les normes de la TEI sont intentionnellement souples, afin de permettre les encodages personnalisés qui se focalisent sur de divers aspects d'un texte. Le même texte peut donc être encodé en TEI de plusieurs manières, selon les besoins et les objectifs des personnes qui se chargent de l'encodage.

Les normes de la TEI sont maintenues par une communauté internationale et leur usage est très répandu dans le monde. Naomi Truan et Laurent Romary ont dit en 2021 que la TEI « *has become, since its inception in 1987, the reference technical standard for*

4. Dans le TEI, les identifiants n'ont pas de mot-dièse, mais quand ils sont référencés dans le document la référence en porte un.

12 août 2022

Coucou ! J'ai fait une réservation
pour ton anniversaire.

À demain ma chérie,

(a) L'exemple d'une lettre

```
<TextLine ID="line1" HPOS="0" VPOS="0" HEIGHT="40" WIDTH="200" BASELINE="0
    40 200 40">
  <Shape>
    <Polygon POINTS="...."/>
  </Shape>
  <String ID="seg1" CONTENT="À" HPOS="..." VPOS="..." WIDTH="..." HEIGHT="
    ..." WC="1.0">
  <!-- ... -->
  <String ID="seg2" CONTENT="demain" HPOS="..." VPOS="..." WIDTH="..."
    HEIGHT="..." WC="0.888">
  <!-- ... -->
  <String ID="seg3" CONTENT="ma" HPOS="..." VPOS="..." WIDTH="..." HEIGHT="
    ..." WC="0.9">
  <!-- ... -->
  <String ID="seg3" CONTENT="chérie," HPOS="..." VPOS="..." WIDTH="..."
    HEIGHT="..." WC="0.91">
  <!-- ... -->
</TextLine>
```

(b) La dernière ligne de la lettre encodée dans le schéma ALTO

```
<salute>À <date when="2022-08-13">demain</date> ma <name ref="#Kelly" type=
    "person">chérie</name>,</salute>
```

(c) La dernière ligne de la lettre encodée dans le schéma TEI

FIGURE 5.5 – Le comparaison de l'encodage d'une ligne de texte en ALTO et TEI

the representation of textual content in the humanities »[5]. Aujourd'hui l'association est soutenue par le financement des institutions patrimoniales qui comptent sur ses *guidelines* et contribuent des cas d'utilisation. Sur son site web, l'association explique qu'elle continue à modifier ses normes selon les besoins des utilisateurs.

> *The scope of the TEI is constantly expanding and the Guidelines are in steady ongoing development to keep pace with the emerging needs of the TEI community*[6].

La croissance de la TEI rend le schéma très approprié à l'édition et à l'échange puisque beaucoup d'institutions ont développé des outils numériques qui l'utilisent.

La souplesse de la TEI est à la fois un avantage et un défi à surmonter. Puisque le schéma permet de plusieurs encodages du même document, il est donc possible de réaliser plusieurs transformations d'un encodage en ALTO vers un encodage en TEI. Mais pour mettre en œuvre une transformation automatique à l'échelle, il faut une seule modélisation qui s'adapte à tout type de document dont la transcription est encodée en ALTO. En outre, l'enrichissement du texte possible dans la TEI est compliqué à réaliser par ordinateur. Tandis qu'un humain pourrait voir la date en-tête sur la lettre dans la Figure 5.5a et puis savoir que la date référencée dans la salutation est le jour suivant, le 13 août, un logiciel ne pourrait pas faire le liaison entre les deux données si facilement. Donc, bien qu'il puisse savoir, grâce au TAL, que le mot *demain* veut parler d'une date, il ne saurait pas de quelle date parle la lettre ; par contre, une lectrice ou un lecteur humain la saurait avec facilité. Voici quelques défis d'une transformation d'ALTO à TEI.

5.2.2 Les éléments de base de la TEI

La TEI peut s'adapter à plusieurs types de documents mais elle exige toujours certains éléments de la racine qui donnent au schéma son arborescence générale. Depuis la racine `<TEI>` d'un document TEI, il faut au moins ces deux descendants : le `<teiHeader>` et le `<body>`. Comme le schéma ALTO, le schéma TEI a besoin des métadonnées à propos du document encodé et de l'encodage lui-même. Le document TEI imbrique les métadonnées dans l'élément `<teiHeader>`. L'élément `<body>` porte sur les données qui constituent la transcription ou la représentation du document ou des documents ; le dernier sera le cas où le document TEI réalise une édition critique qui ressemblent plusieurs exemplaires d'une œuvre, par exemple. Pour résumer, la TEI a besoin d'au moins les métadonnées, encodées dans le `<teiHeader>`, et les données qui représentent le texte, encodées dans le `<body>`.

5. Naomi TRUAN et Laurent ROMARY. « Building, Encoding, and Annotating a Corpus of Parliamentary Debates in TEI XML : A Cross-Linguistic Account ». In : *Journal of the Text Encoding Initiative* Issue 14 (Issue 14 17 mars 2021). ISSN : 2162-5603. DOI : 10.4000/jtei.4164. URL : https://journals.openedition.org/jtei/4164#tocto2n4 (visité le 26/08/2022), p. 21.

6. *About – TEI : Text Encoding Initiative*. URL : https://tei-c.org/about/ (visité le 25/08/2022).

Après ces deux éléments obligatoires, le schéma TEI autorise d'autres éléments facultatifs de descendre directement de la racine `<TEI>`. L'un d'eux est l'élément `<sourceDoc>` dont nous parlons dans la section 4.3.1. La TEI définit le « `<sourceDoc>` comme un élément qui peut contenir *une transcription ou une représentation d'un seul document source, qui se réserve le pouvoir à faire partie d'un dossier génétique ou d'une collection d'autres sources* » (traduction par l'autrice)[7] . Comme se justifie dans la section 4.3.1, le projet *Gallic(orpor)a* a choisi d'encoder toute donnée du fichier ALTO dans l'élément TEI `<sourceDoc>`. Le schéma TEI destine l'élément `<sourceDoc>` à la transcription d'un document source. Un fichier ALTO contient une telle transcription, produite par un logiciel OCR ou HTR. Le `<sourceDoc>` convient bien aux données d'un fichier ALTO car les éléments qui descendent du `<sourceDoc>` portent sur la mise en page ainsi que sur la transcription des images de texte.

Un autre élément facultatif qui descend de la racine `<TEI>` est l'élément `<standOff>`. Le projet *Gallic(orpor)a* s'appuyait aussi sur cet élément parce qu'il est destiné à la représentation des annotations au texte. L'avant dernière étape du *pipeline* du projet est l'analyse linguistique du texte prédit par le logiciel HTR. Le résultat de cette analyse est une version du texte annotée qui pourrait se différer sensiblement da la transcription. Selon les *guidelines* de la TEI :

> *The* `standOff` *element is intended to hold content that does not fit well in the* `text` *(e.g. because it is not transcribed from the source), nor in the* `teiHeader` *(e.g. because it is not metadata about the source or transcription). Examples include [...] annotations indicating the morphosyntactic features of a text, and annotations commenting on or associating parts of a text with additional information*[8].

Comme s'est révélé par les *guidelines*, l'élément `<standOff>` convient bien au texte annoté et normalisé. Ainsi, la transcription du texte tel qu'il apparaît dans le document source, avec toute saute de ligne et faute d'orthographe, se trouvera dans les éléments `<sourceDoc>` et `<body>`, qui descend du `<text>`. Par contre, la version du texte qui n'existe pas dans le document source mais qui sert bien à l'analyse du document se trouvera dans l'élément facultatif `<standOff>`. Pour résumer, les quatre éléments pertinents qui descendent de la racine `<TEI>` sont visualisés dans la Figure 5.6.

7. *TEI element sourceDoc.*

8. *16 Linking, Segmentation, and Alignment - The TEI Guidelines.* URL : `https://tei-c.org/release/doc/tei-p5-doc/en/html/SA.html#SASOstdf` (visité le 25/08/2022).

```xml
<TEI>
  <teiHeader><!-- métadonnées --></teiHeader>
  <sourceDoc><!-- transcription --></sourceDoc>
  <text>
    <body><!-- texte pré-éditorialisé --></text>
  </text>
  <standOff><!-- texte annoté --></standOff>
</TEI>
```

FIGURE 5.6 – Les éléments de base du schéma TEI

Chapitre 6

À la recherche des métadonnées

Tout document TEI doit comporter des métadonnées qui renseignent sur le document TEI lui-même ainsi que sur le texte qu'il représente. Dans le cadre du projet *Gallic(orpor)a*, le texte produit est toujours la transcription d'un document source numérisé. Les métadonnées du fichier doivent en plus de la ressource numérisée prendre en compte le document source physique qui a été numérisé. Pour les besoins du projet *Gallic(orpor)a*, les métadonnées du document TEI portent sur les trois objets de texte suivants :

1. la ressource numérique, c'est-à-dire les informations sur le document TEI produit par le pipeline *Gallic(orpor)a*
2. la numérisation du document source, c'est-à-dire le fac-similé numérique stocké dans la base de données Gallica
3. le document source physique, qui a été numérisé

Malgré la diversité documentaire du projet *Gallic(orpor)a*, le `<teiHeader>` de tout document TEI que le pipeline produit est toujours constitué d'informations sur ces trois objets. Chacun porte sur un fac-similé numérique, dérivé d'un document source physique et transformé en objet numérique par le pipeline. Les métadonnées doivent être les mêmes pour tous les documents issus du pipeline. Un manuscrit écrit par plusieurs mains et sans éditeur, issu d'un scriptorium à une date approximative, doit disposer des mêmes types de métadonnées qu'un imprimé écrit par une autrice et publié par un éditeur. Une solution pour surmonter ce défi est de réduire le nombre de métadonnées à l'essentiel de sorte à toujours respecter la même structure au sein de la collection.

Selon les normes de la TEI, trois parties peuvent constituer le `<teiHeader>` : une description bibliographique de l'encodage (`<fileDesc>`), une description des aspects non bibliographiques tels qu'un classement du contenu textuel qui appartiennent au texte représenté (`<profileDesc>`), et une description technique de l'encodage dans `<encodingDesc>`. L'objectif de ce chapitre est d'expliquer la structure du `<teiHeader>` que nous avons modélisée dans le cadre du projet *Gallic(orpor)a*. Le chapitre 7 explique comment réaliser cette structure en récupérant toutes les métadonnées déterminées essentielles, selon la

modélisation exposée dans ce chapitre.

6.1 La description bibliographique (<fileDesc>)

Dans un premier temps, la ressource numérique elle-même doit être décrite. Ces informations sont organisées dans l'élément <fileDesc>. Traduit littéralement en français comme *la description du fichier*, le <fileDesc> décrit le document TEI lui-même. Cette description doit porter sur les trois aspects suivants :

1. le titre et la responsabilité de la ressource numérique (<titleStmt>)
2. la distribution de la ressource, y compris les droits d'utilisation (<publicationStmt>)
3. le document source (<sourceDesc>)

D'autres éléments peuvent être ajoutés dans le <teiHeader> afin d'apporter encore plus de détails bibliographiques. Par exemple, le <editionStmt> précise l'édition de l'œuvre. Les documents TEI produits par le pipeline *Gallic(orpor)a* ne profitent pas de cet élément parce que certains documents du corpus traité, tels que les manuscrits, n'ont pas d'édition et on veut que chaque document TEI ait les mêmes types de métadonnées dans le <teiHeader>. Au contraire, d'autres éléments facultatifs, tel que l'élément <extent>, sont produits par le pipeline. L'élément <extent> est utile parce qu'il porte sur la taille de la ressource (nombre de pages transcrites) ce qui permet d'indiquer le volume de total de la ressource.

6.1.1 Le titre et la responsabilité (<titleStmt>)

Après le titre (<title>), il est recommandé d'indiquer l'individu ou les individus responsables de la création du texte représenté et/ou de la ressource numérique. Dans le cadre du projet *Gallic(orpor)a*, nous avons conçu un schéma <titleStmt> simple qui comporte : titre, auteur(s), et responsables de la production de la ressource numérique.

La responsabilité

Entre les lignes 4 et 21 de la Figure 6.1, l'élément <respStmt> contient des informations sur l'équipe du projet *Gallic(orpor)a*. Tout document encodé par le pipeline comporte un titre (<title>), le nom de la personne ou des personnes auxquels est attribuée la responsabilité du texte (<author>), et la mention de l'équipe qui a conçu le pipeline et l'application `alto2tei` (<respStmt>). La déclaration de responsabilité peut être personnalisée selon le projet ou selon l'équipe qui utilise le pipeline *Gallic(orpor)a* ou l'application `alto2tei` pour créer la ressource numérique. En général, elle devrait contenir une phrase qui résume la nature de la création de la ressource, telle que la phrase « Transformation from ALTO4 to TEI by », balisée dans l'élément <resp> (ligne 5, Fig. 6.1).

Ensuite, elle devrait contenir des éléments <persName> pour renseigner le nom des personnes en charge de la production du document.

```
<titleStmt>
  <title>Titre du document source traité</title>
  <author>Auteur</author>
  <respStmt>
    <resp>Transformation from ALTO4 to TEI by</resp>
    <persName>
      <forename>Kelly</forename>
      <surname>Christensen</surname>
      <ptr type="orcid" target="000000027236874X"/>
    </persName>
    <persName>
      <forename>Simon</forename>
      <surname>Gabay</surname>
      <ptr type="orcid" target="0000000190944475"/>
    </persName>
    <persName>
      <forename>Ariane</forename>
      <surname>Pinche</surname>
      <ptr type="orcid" target="0000000278435050"/>
    </persName>
  </respStmt>
</titleStmt>
```

FIGURE 6.1 – Les informations sur le titre de la ressource

Le titre

La ressource produite par le pipeline *Gallic(orpor)a* a besoin de son propre titre, et l'application `alto2tei` que j'ai créée lui attribue le nom du document source. Le schéma TEI permet de construire un nouveau titre lors de l'encodage ou d'associer plusieurs titres au document TEI. Cependant, nous avons décidé d'utiliser le titre du document source fourni par les métadonnées produites par la BnF, au lieu de construire un nouveau titre.

Dans le schéma TEI, plusieurs titres peuvent être indiqués dans le <titleStmt>. Par exemple, le projet *The Bodelian First Folio* a encodé les premières éditions des drames de Shakespeare en TEI et chacun porte plusieurs types de titre [1]. L'encodage de la comédie *Twelfth Night* possède un titre du type "*statement*" qui représente le titre tel qu'il se trouve sur l'imprimé historique (ligne 2, Fig. 6.2). Il donne aussi une variante du titre qui est le nom du recueil dans lequel se trouve la pièce (ligne 3, Fig. 6.2). Enfin, l'encodage présente le titre qui sert à identifier la source aux archives, « Bodelian First Folio, Arch. G c.7 » (ligne 4, Fig. 6.2)

1. *The Bodelian First Folio : Digital Facsimile of the First Folio of Shakespeare's Plays.* URL : `http://firstfolio.bodleian.ox.ac.uk/` (visité le 27/08/2022).
2. ibid.

```
<titleStmt>
  <title type="statement">Twelfe Night, or What You Will from Mr. William
   Shakespeares comedies, histories, &amp; tragedies. Published according
   to the true originall copies.</title>
  <title type="variant">Mr. VVilliam Shakespeares comedies, histories, &amp
   ; tragedies</title>
  <title type="distinctive">Bodleian First Folio, Arch. G c.7</title>
<!-- ... -->
</titleStmt>
```

FIGURE 6.2 – Les informations sur le titre d'un imprimé [2]

Pour un manuscrit, l'attribution d'un titre pourrait obliger la création d'un nom qui ne se trouve pas sur le document source. Par exemple, le projet *CatCor* qui a encodé des lettres écrites par Catherine II de la Russie a choisi d'attribuer un titre qui s'appuie sur l'identifiant du document source. Dans l'encodage d'une lettre destinée à Fréderick II le 21 juillet 1744, le `<title>` dans le `<titleStmt>` est un titre qui n'apparaît nulle parte sur la source (ligne 2, Fig. 6.3) [3]. Contrairement à l'encodage de l'imprimé de Shakespeare, l'encodage de la lettre manuscrite ne donne pas de type au titre attribué à la ressource TEI. La classification `@type` du `<title>` n'est pas exigée par le schéma TEI, mais elle est recommandée s'il y a plusieurs titres. Puisque la lettre manuscrite n'a pas de titre donné par l'auteur, l'équipe de *CatCor* l'en a créé un et uniquement un pour l'encodage de la lettre manuscrite.

```
<titleStmt>
  <title>CatCor Project: letter-02633</title>
<!-- ... -->
</titleStmt>
```

FIGURE 6.3 – Les informations sur le titre d'un manuscrit [4]

Afin d'encoder les éléments `<title>` à l'échelle, il faut qu'un logiciel (1) ait d'accès aux métadonnées et (2) sache la nature des titres associés au document source. Certains corpus auront d'accès à des métadonnées déjà classifiées. Le catalogue général de la BnF, par exemple, organise ses métadonnées dans une structure de données XML UNIMARC. Chaque titre associé au document est balisé dans des éléments XML qui portent des noms ou des attributs qui précisent la nature du titre. Par exemple, l'UNIMARC présente le type « titre uniforme » dans l'élément `<mxc:datafield tag="500">` et le type « titre de forme » dans l'élément `<mxc:datafield tag="503">`. En récupérant les métadonnées depuis une source ainsi organisée, un logiciel pourrait attribuer un type à l'élément `<title>`.

3. CatCor PROJECT. *Letter 02633 : To Frederick II (the Great), 21 July 1744*. Sous la dir. d'Andrew KAHN et Kelsey RUBIN-DETLEV. 2021. URL : https://catcor.seh.ox.ac.uk/id/letter-02633.

4. ibid.

L'auteur

Le <titleStmt> renseigne sur l'individu ou les individus aux lesquels la propriété intellectuelle du document source est attribuée. Cette donnée est encodée dans l'élément <author>. L'encodage de la responsabilité peut être minimal et ne contenir qu'une chaine de caractères avec le nom des auteurs ou bien un simple l'élément <name>. Voir Figure 6.4. Sinon, la description peut être enrichie par d'autres éléments afin d'apporter plus de détail sur les composants du nom de l'auteur. Voir Figure 6.5.

```
<author>
  <name>Donatien Alphonse François de Sade</name>
</author>
```

FIGURE 6.4 – L'auteur simple

```
<author xmlid="Sa1">
  <persName>
    <forename>Donatien Alphonse François</forename>
    <nameLink>de</nameLink>
    <surname>Sade</surname>
    <ptr type="isni" target="0000000084961458"/>
  </persName>
</author>
```

FIGURE 6.5 – L'auteur enrichi

L'élément <ptr> qui veut dire *pointer* en anglais indique une ressource ou une donnée externe, tel que l'identifiant ISNI, afin d'enrichir les informations de l'objet auquel il est attaché. On voit un exemple du *pointer* sur les lignes 9, 14, et 19 de la Figure 6.1 où l'élément indique l'ORCID unique de l'individu responsable de la création de la ressource numérique.

L'exemple d'un document TEI produit par le pipeline *Gallic(orpor)a* et donc de notre modélisation du <author> se voit dans la Figure 6.5. Le nom de l'auteur est divisé en trois composants, selon les données UNIMARC fournies par le catalogue général de la BnF. Le catalogue désigne *Donatien Alphonse François de* comme la « partie du nom autre que l'élément d'entrée ». L'UNIMARC balise cette partie secondaire dans l'élément <mxc:subfield code="b"> de l'élément <mxc:datafield tag="700"> ou <mxc:datafield tag="701">[5]. Mon application alto2tei, développée pour le pipeline *Gallic(orpor)a*, ensuite tire la partie *de* du nom puisqu'elle est un lien entre ses composants et peut donc être encodée dans l'élément <nameLink>. Le catalogue général catégorise le nom *Sade* comme le nom d'entrée de l'auteur, que l'UNIMARC balise dans l'élément

5. *Manuel UNIMARC : format bibliographique*. Transition bibliographique - Programme national. URL : https://www.transition-bibliographique.fr/unimarc/manuel-unimarc-format-bibliographique/ (visité le 27/08/2022).

`<mxc:subfield code="a">`. Notre modélisation balise donc cette donnée dans l'élément `<surname>`, c'est-à-dire le nom de famille. Dans le schéma TEI, le `<titleStmt>` peut renseigner plusieurs auteurs en répétant l'élément `<author>` comme dans la Figure 6.6.

Le schéma TEI permet d'indiquer le rôle de chacun des auteurs listés dans un `<titleStmt>`. Toutefois, par manque de métadonnées disponibles dans les sources de données que nous avons ciblées, notre modélisation actuelle ne s'appuie pas sur cette donnée. Dans l'exemple de la Figure 6.6, les données UNIMARC récupérées du catalogue général de la BnF n'indiquent pas que Giacomo Meyerbeer est le compositeur de l'opéra *Les Huguenots* ni que lui et le librettiste Eugène Scribe partagent en parts égales la responsabilité pour l'œuvre.

```
<titleStmt>
  <title>Les Huguenots</title>
  <author xml:id="Me1">
    <persName>
      <forename>Giacomo</forename>
      <surname>Meyerbeer</surname>
      <ptr type="isni" target="0000000122817116"/>
    </persName>
  </author>
  <author xml:id="Sc1">
    <persName>
      <forename>Eugène</forename>
      <surname>Scribe</surname>
      <ptr type="isni" target="000000012122970X"/>
    </persName>
  </author>
  <author xml:id="De1">
    <persName>
      <forename>Émile</forename>
      <surname>Deschamps</surname>
      <ptr type="isni" target="0000000122807567"/>
    </persName>
  </author>
  <author xml:id="Ro1">
    <persName>
      <forename>Gaetano</forename>
      <surname>Rossi</surname>
      <ptr type="isni" target="0000000121219499"/>
    </persName>
  </author>
<!-- ... -->
</titleStmt>
```

FIGURE 6.6 – Plusieurs auteurs dans un `<titleStmt>`

6.1.2 La taille de la ressource numérique (`<extent>`)

Lors de la phase de reconnaissance du texte, les modèles HTR génèrent un certain nombre de fichiers XML ALTO. Grâce à l'unité indiquée comme « images » dans l'élément

<measure>, la taille est facilement calculé à partir du compte des fichiers XML ALTO, les mêmes fichiers produits et ensuite traités pour constituer l'élément <sourceDoc> de la ressource numérique.

```
<extent>
  <measure unit="images" n="20"/>
</extent>
```

FIGURE 6.7 – La taille de la ressource

6.1.3 La distribution de la ressource numérique (<publicationStmt>)

L'élément <publicationStmt> contient des données importantes qui renseignent sur la distribution et les droits d'utilisation de la ressource numérique en format XML TEI. Toutes les métadonnées présentes dans cet élément portent sur le contexte de la création de la ressource, aucune sur le document source représenté. Pour toutes les ressources produites par le pipeline, les données du <publicationStmt> ne doivent pas changer, à l'exception de la date de création de la ressource, que l'application alto2tei génère automatiquement.

Comme montre la Figure 6.8, trois données du <publicationStmt> peuvent être personnalisées. L'entité reconnue comme l'éditeur (*publisher*) de la ressource peut être changé selon le projet. Dans l'exemple de la Figure 6.8, le *publisher* est « Gallic(orpor)a ». L'autorité qui l'a financé et qui est civilement responsable pour la ressource est le Data-Lab de la BnF indiqué dans l'élément <authority>. Enfin, les droits d'utilisation de la ressource sont indiqués par les éléments <availability> et <licence>.

```
<publicationStmt>
  <publisher>Gallic(orpor)a</publisher>
  <authority>BnF DATALab</authority>
  <availability status="restricted" n="cc-by">
    <licence target="https://creativecommons.org/licenses/by/4.0/"/>
  </availability>
  <date when="2022-07-29"/>
</publicationStmt>
```

FIGURE 6.8 – Plusieurs auteurs dans un <publicationStmt>

6.1.4 Le document source (<sourceDesc>)

Le dernier aspect de la description bibliographique du <teiHeader> porte sur le document source physique dont le texte est représenté. Cet aspect est balisé dans l'élément <sourceDesc> qui est le dernier élément du <fileDesc> dans notre modélisation TEI. Cet élément s'appuie sur un grand nombre de sources externes afin de renseigner cette

partie de la manière la plus diversifiée et précise possible. La récupération de ces données est expliquée dans le chapitre 7.

La citation bibliographique (<bibl>)

Dans un premier temps, la description de la source représentée dans le <sourceDesc> s'appuie sur les informations bibliographiques. En répétant certaines informations du <titleStmt>, l'élément <bibl> présente (1) les individus auxquels est attribuée la propriété intellectuelle du document, (2) le titre du document source–qui est le même titre attribué à la ressource numérique produite, selon notre modélisation, (3) son lieu de publication, (4) l'éditeur, et (5) la date de publication. Voir l'arborescence du <bibl> dans la Figure 6.9.

Les dates de publication ou d'apparition du document source physique sont représentées dans l'élément <date>. Le schéma TEI permet d'ajouter plusieurs détails à la date. Par exemple, notre modélisation TEI attribue un degré de certitude à la date attribué à la source quand la ressource s'appuie sur le catalogue général de la BnF. Les données UNIMARC du catalogue nous permet de déterminer un degré de certitude quant à la date parmi les trois valeurs suivantes : « low », « medium », ou « high ». De plus, la TEI recommande fortement l'encodage de la date dans un format standardisé. Notre modélisation TEI conseille donc l'encodage d'une date selon le format visualisé dans la ligne 21 de la Figure 6.9, où l'attribut @resp représente l'autorité qui a accordé le degré de certitude.

L'exemple de la Figure 6.9 est issu d'un imprimé du XVIII^{e} siècle dont la date de publication est connue mas pas l'éditeur. L'éditeur n'est indiqué dans les données UNIMARC de la BnF comme « s.n. » pour signaler que la bibliothèque n'en est pas certaine. Notre modélisation TEI envisage à répondre aux données incertaines ou manquantes des sources de données en laissant vides ou incertaines les données intégrées dans l'arborescence du <teiHeader>. Par conséquent, l'élément <publisher> dans la Figure 6.9 contient la donnée du catalogue même si elle indique une manque d'information.

Le <bibl> d'un manuscrit contient les mêmes éléments que celui d'un imprimé. Même si un manuscrit n'a pas d'éditeur ni n'est pas publié de la même manière qu'un imprimé, la structuration TEI reste la même et l'élément <bibl> contiendra tout de même les sous-éléments <pubPlace> et <publisher>. Normalement, le catalogue général de la BnF n'indiquera pas de donnée pour ces aspects. Quand la donnée n'est pas disponible, notre modélisation exige que l'arborescence générique est toujours gardée et qu'elle indique dans l'élément standard que la donnée n'existe pas.

```
<sourceDesc>
  <bibl>
    <ptr target="http://catalogue.bnf.fr/ark:/12148/cb30369299r"/>
    <author ref="#Re1">
      <persName>
        <forename>Jean-François</forename>
        <surname>Regnard</surname>
        <ptr type="isni" target="000000012118509X"/>
      </persName>
    </author>
    <author ref="#Du2">
      <persName>
        <forename>Charles</forename>
        <surname>Du Fresny</surname>
        <ptr type="isni" target="0000000140935001"/>
      </persName>
    </author>
    <title>Scènes françoises de la comédie italienne intitulée "la Foire S
    .-Germain" , comme elles ont paru dans les premières représentations</
    title>
    <pubPlace key="FR">Grenoble</pubPlace>
    <publisher>[s.n.]</publisher>
    <date when="1696" cert="high" resp="BNF">1696</date>
  </bibl>
  <msDesc>
<!-- ... -->
  </msDesc>
</sourceDesc>
```

FIGURE 6.9 – La citation bibliographique (<bibl>)

La description de la source physique (<msDesc>)

Le document source physique et son fac-similé numérique sont tous les deux décrits dans l'élément <msDesc>. Tandis que la citation bibliographique (<bibl>) porte sur le document en tant qu'œuvre, la *description du manuscrit* (<msDesc>) porte sur le document en tant qu'un objet matériel dans le monde réel. Le <msDesc> a deux enfants principaux, voir Figure 6.10. Premièrement, l'élément <msIdentifier> sert à identifier le document physique et le fac-similé numérique dans un catalogue. Deuxièmement, l'élément <physDesc> sert à décrire le document manuscrit soit imprimé.

L'identification du document physique est très importante. Par exemple, un imprimé peut avoir plusieurs exemplaires ; chacun pourrait avoir des différences et une transcription distincte. Afin de bien indiquer quel objet a été traité par le pipeline *Gallic(orpor)a*, il faut identifier le document source physique qui a servi de base aux prédictions des modèles HTR. Les éléments <idno> dans le <msDesc> indiquent l'identifiant d'un document. L'identifiant du document source physique est l'élément principal descendant du <msIdentifier>. Le <idno> alternatif est l'ARK du fac-similé numérique sur Gallica. Le <idno> principal du <msDesc> est le cote du document physique selon le catalogue de la Bibliothèque nationale de France. Après cette description des sources numériques et physiques (<msDesc>), sur lesquels la ressource numérique TEI est basée, la description du fichier <fileDesc> est complète selon notre modélisation.

```
<sourceDesc>
  <bibl>
<!-- ... -->
  </bibl>
  <msDesc>
    <msIdentifier>
      <country key="FR"/>
      <settlement>Paris</settlement>
      <repository>Bibliothèque nationale de France</repository>
      <idno>YF-5877</idno>
      <altIdentifier>
        <idno type="ark">bpt6k1281160s</idno>
      </altIdentifier>
    </msIdentifier>
    <physDesc>
      <objectDesc>
        <p>Texte imprimé</p>
      </objectDesc>
    </physDesc>
  </msDesc>
</sourceDesc>
```

FIGURE 6.10 – La description de la source (<msDesc>)

6.2 La description non bibliographique (<profileDesc>)

La description du fichier (<fileDesc>) porte sur les détails bibliographiques de la ressource numérique, tel que le document source. Ensuite, notre modélisation du <teiHeader> renseignent sur une description non bibliographique de la ressource numérique en format TEI. Normalement la description non bibliographique informe sur les langues utilisées dans le texte. Elle peut aussi donner les noms de lieux ou de personnages référencés dans le texte. Notre modélisation est moins détaillé. Notre <profileDesc> renseigne simplement sur la langue du texte et sur l'application alto2tei qui a construit le document TEI. Actuellement, cette application ne peut porter que sur une seule langue. C'est un point de modélisation des données qui pourrait évoluer pour, si besoins à l'avenir proposer un encodage capable d'accepter plusieurs langues du texte ou encore des index.

La Figure 6.12 présente un <profileDesc> réalisé par l'application alto2tei et donc montre l'exemple de notre modélisation TEI. La langue identifiée est français. Cette information a été récupérée depuis le *manifest* IIIF du fac-similé numérique de Gallica. Mais quand l'application alto2tei peut aussi accéder aux données du catalogue général de la BnF, le script s'appuie en priorité sur cette source. L'identifiant de la langue est souvent disponible dans les données UNIMARC. L'identifiant « fre » indique le français moderne. Même si la langue identifiée dans le *manifest* IIIF est le français, les données UNIMARC du catalogue de la BnF pourrait en préciser la période, par exemple le moyen français (identifiant « frm »). Même si notre modélisation du <profileDesc> est simple, il s'appuie sur deux sources de données et présente l'identifiant standardisé de la langue utilisé dans le document. Cela permet de filtrer les documents traités par notre pipeline selon la langue.

```
<profileDesc>
  <langUsage>
    <language ident="fre">français</language>
  </langUsage>
</profileDesc>
```

FIGURE 6.11 – La description non bibliographique de la source (<profileDesc>)

6.3 La description technique (<encodingDesc>)

Le dernier composant du <teiHeader> est une description technique de l'encodage. Balisée dans l'élément <encodingDesc>, la description technique informe sur la manière dont l'encodage a été produit. Tout projet scientifique devrait pouvoir reproduire ses résultats. La ressource doit donc renseigner l'ensemble des étapes et des méthodes mises en place pour aboutir à la production du texte présent dans le fichier. L'élément TEI

<appInfo> renseigne sur le logiciel HTR qui a permis de prédire l'analyse de la mise en page et le texte depuis les images numériques.

```
<encodingDesc>
  <appInfo>
    <application ident="Kraken" version="3.0.13">
      <label>Kraken</label>
      <ptr target="https://github.com/mittagessen/kraken"/>
    </application>
  </appInfo>
  <classDecl>
    <taxonomy xml:id="SegmOnto">
      <bibl>
        <title>SegmOnto</title>
        <ptr target="https://github.com/segmonto"/>
      </bibl>
      <category xml:id="SegmOntoZones">
        <catDesc xml:id="MainZone">
          <title>MainZone</title>
          <ptr target="https://segmonto.github.io/gd/gdZ/MainZone"/>
        </catDesc>
        <catDesc xml:id="TitlePageZone">
          <title>TitlePageZone</title>
          <ptr target="https://segmonto.github.io/gd/gdZ/TitlePageZone"/>
        </catDesc>
<!-- more SegmOnto Zones -->
      </category>
      <category xml:id="SegmOntoLines">
        <catDesc xml:id="DefaultLine">
          <title>DefaultLine</title>
          <ptr target="https://segmonto.github.io/gd/gdL/DefaultLine"/>
        </catDesc>
        <catDesc xml:id="HeadingLine">
          <title>HeadingLine</title>
          <ptr target="https://segmonto.github.io/gd/gdL/HeadingLine"/>
        </catDesc>
<!-- more SegmOnto Lines -->
      </category>
    </taxonomy>
  </classDecl>
</encodingDesc>
```

FIGURE 6.12 – La description non bibliographique de la source (<profileDesc>)

Le <encodingDesc> informe également sur la syntaxe qui a permis de structurer le texte encodé en utilisant l'élément <classDecl>. La *déclaration des classes* (<classDecl>) porte sur les classes qui ont permis d'organiser le texte prédit. Dans notre modélsiation, les classes du vocabulaire *SegmOnto* sont attribuées aux lignes de texte et aux zones qui contenaient le texte. Puisque les lignes de texte et les zones portent les noms du vocabulaire *SegmOnto*, le <classDecl> fournit une description de chaque classe. La Figure 6.12 montre comment la description de classe est balisée dans l'élément <catDesc> qui lui-même se trouve balisé dans la catégorie (<category>) de la classe, soit la ligne soit la zone.

Troisième partie

Mise en opérationnelle du projet

Chapitre 7

La génération du `<teiHeader>`

J'ai créé l'application `alto2tei` afin que le pipeline *Gallic(orpor)a* puisse aller des fichiers sortis des modèles HTR vers une version préliminaire de la ressource numérique qu'il produit [1]. L'objectif est donc de transformer des données du format XML ALTO en un format XML TEI. Cependant, la création d'un document TEI préliminaire n'est pas si simple que la transposition des données d'une structure dans une autre. Le schéma TEI exige plus que cela. Dans le chapitre 6, l'exposition de notre modélisation des métadonnées explique que la transcription d'un document n'est pas tout ce qu'il faut dans un document TEI. Il faut aussi des métadonnées, mais elles ne se trouvent pas dans la transcription prédite par des modèles HTR. L'application `alto2tei` a donc besoin de récupérer les métadonnées à partir des sources externes, qui contiennent des données autre que celles créées par le pipeline.

Quelles métadonnées faut-il chercher ? Et où ? Dans le chapitre 6, je parle de trois objets de texte conceptuels pertinents dans notre modélisation des métadonnées : (1) la ressource numérique elle-même en format XML TEI, (2) le fac-similé numérique disponible depuis l'API de la BnF dont les images sont traitées par les modèles HTR, et (3) le document physique qui est conservé par la BnF et qui a été numérisé. Chacun est distinct et ses métadonnées se trouvent depuis des sources différentes. Afin d'informer les métadonnées de la ressource numérique, il faut donc s'appuyer sur les métadonnées de ces dits objets de texte.

Le pipeline du projet *Gallic(orpor)a* a besoin de diversifier sa manière de récupérer des métadonnées. L'application `alto2tei` profite de quatre sources externes de données afin de renseigner sur les trois documents conceptuels concernés.

1. Pour faire un petit résumé de ce que nous avons déjà abordé, la première étape du pipeline est la récupération des pages numérisées du document source. Ensuite le pipeline prédit le texte et la mise en page du fac-similé numérique en traitant chaque page avec des modèles HTR. Les données produites par les modèles sont en format XML ALTO. Mais le pipeline veut présenter les données enrichies par les métadonnées et par l'analyse linguistique en format XML TEI puisque le schéma TEI est plus utilisé et il convient mieux à l'édition du texte que le schéma ALTO. Mon application `alto2tei` a complété le pipeline en construisant un document TEI à partir des données des modèles HTR.

1. les métadonnées sur la ressource lexicographique numérique
   — source : fichier de configuration personnalisable
   — format : YAML (*Yet Another Markup Language*)
   — portée : informations administratives portant sur la création de la ressource, y compris les droits d'utilisation et les autorités qui s'en chargent
2. les métadonnées sur le fac-similé numérique
   — source : manifest IIIF renvoyé de l'IIIF API
   — format : JSON (*JavaScript Object Notation*)
   — portée : données bibliographiques qui servent à identifier les exemplaires numériques traités par des modèles HTR
3. les métadonnées sur le document source physique
   — source : réponse à la requête envoyée à l'API SRU (*Search/Retrieve via URL*) de la BnF, qui interroge le catalogue général de la BnF
   — format : UNIMARC XML
   — portée : données bibliographiques, y compris la responsabilité du document source, sa création, et sa conservation actuelle

Les métadonnées sur le document source physique s'appuient encore sur une autre source de données afin de récupérer où se trouve l'institution qui héberge le document. Dans le cadre du projet *Gallic(orpor)a*, la réponse à cette question est toujours Paris, puisque la BnF se situe au capital. Mais, afin d'éviter l'encodage dur, l'application `alto2tei` recherche cette donnée dans la base de données du Système Universitaire de Documentation (SUDOC) puisque la localisation de l'institution hôte du document physique n'est pas encodée dans les données UNIMARC selon l'usage de la BnF.

7.1 La récupération des métadonnées

Comme est expliqué dans l'exposition du pipeline entier (chap. 4), l'application `alto2tei` s'appuie sur un système de fichier fixé. Cela lui permet d'accéder à l'identifiant ARK (Archival Resource Key) du document dont les pages numérisées ont été traitées par les modèles HTR. Cet identifiant donne la clef aux données sur le fac-similé numérique encodées dans le *manifest* IIIF (International Image Interoperability Framework). L'une des données dans le *manifest* IIIF, au moins pour les documents de la base de données Gallica, porte sur la relation entre le fac-similé numérique est la source physique à partir duquel le fac-similé numérique a été produit, voir la Figure 7.1. Cette relation permet de rechercher les métadonnées du document physique dans le catalogue général de la BnF.

Si la relation entre le document numérique sur Gallica et le document physique dans l'un des magasin de la BnF n'est pas établie, l'application `alto2tei` n'abandonne pas tout essai de produire un document TEI. Au lieu de s'arrêter, elle compte uniquement

```
"metadata": [
  {
  "label": "Relation",
  "value": "Notice du catalogue : http://catalogue.bnf.fr/ark:/12148/
    cb30369299r"
  },
]
```

FIGURE 7.1 – Une partie du *manifest* IIIF pour le fac-similé numérique d'ARK `bpt6k1281160s`, un imprimé numérisé sur Gallica

sur les données du *manifest* IIIF. Ces données sont vérifiées puisqu'elles portent sur le fac-similé numérique dont les images les modèles HTR du pipeline ont traitées. Le fac-similé numérique est toujours l'un des objets de texte pertinents aux métadonnées de la ressource. L'application `alto2tei` peut donc créer deux genres de `<teiHeader>`, l'un avec un maximum d'information et l'autre, au cas où la relation entre le fac-similé numérique et le document physique n'était pas établie, avec un peu moins d'information.

7.2 Les sources des métadonnées

Six source de données servent à informer des éléments du `<teiHeader>`, mais l'une d'eux est un calcul simple de données internes au pipeline et l'autre est une donnée donnée au démarrage de l'application ou du pipeline. Elles sont (1) le *manifest* IIIF, (2) les données UNIMARC du catalogue général de la BnF, (3) la base de données du site web du Système Universitaire de Documentation (SUDOC), (4) le fichier de configuration, (5) les fichiers ALTO produits par des modèles HTR, et (6) la version de *Kraken* utilisée par les modèles HTR. Les trois premières sources exigent une requête à une API. Le *manifest* IIIF est renvoyé par une requête à l'API IIIF du fac-similé numérique sur Gallica, selon les normes imposées par la BnF. Les données UNIMARC du catalogue général de la BnF sont accessibles à partir d'une requête à l'API SRU (*Search/Retrieve via URL*) de la BnF. Enfin, les données du Système Universitaire de Documentation sont renvoyées à partir d'une requête à l'*endpoint* API du site SUDOC. L'application `alto2tei` dispose directement du fichier de configuration ainsi que des fichiers ALTO qu'elle traite.

7.2.1 Le format du *manifest* IIIF

Le *manifest* IIIF est un format de données standardisé qui est envoyé par tout API IIIF selon la syntaxe de requête demandée par l'organisation qui gère l'API. La Bibliothèque nationale de France exige la syntaxe suivante pour toute requête envoyée à l'API IIIF des fac-similés numériques mis sur Gallica :

`https://gallica.bnf.fr/iiif/ark:/12148/<ARK>/manifest.json`

Pour utiliser cette requête, il faut simplement remplacer la partie `<ARK>` par l'ARK du document cherché. Grâce au système de fichier du pipeline *Gallic(orpor)a*, l'ARK du document est facilement disponible. Il est le nom du dossier qui contient les fichiers XML ALTO traités.

L'API IIIF renvoie les données dans un fichier en format JSON. La structure des données dedans peut varier un peu, mais certains composants sont toujours attendus. Le Tableau 7.2 montre toute clef principale du *manfiest* pour un fac-similé numérique mis en ligne sur Gallica par la BnF. L'exemple prend le livret imprimé du ballet *Les divers entretiens de la fontaine de Vaucluse* de 1649, dont le fac-similé a l'identifiant `bpt6k1513919s`. Les données les plus importantes sont celle imbriquées dans la clef *metadata* et celles de la clef *sequences*. La première contient une suite des pairs clef-valeur, chacun porte sur une métadonnée du fac-similé numérique. La dernière s'appuie sur les images numériques.

clef	**valeur**
@id	https ://gallica.bnf.fr/iiif/ark :/12148/bpt6k1513919s/manifest.json
label	BnF, département Réserve des livres rares, RES-YF-1990
attribution	Bibliothèque nationale de France
license	https ://gallica.bnf.fr/html/und/conditions-dutilisation-des-contenus-de-gallica
logo	https ://gallica.bnf.fr/mbImage/logos/logo-bnf.png
related	https ://gallica.bnf.fr/ark :/12148/bpt6k1513919s
description	Les divers entretiens de la fontaine de Vaucluse : ballet dansé à la grande salle du Roure l'année 1649...
metadata	*suite des pairs qui portent sur les métadonnées du fac-similé numérique*
sequences	*imbrication complexe pour contenir des données sur les images, dit canvases, du fac-similé numérique*
thumbnail	*pair qui contient un lien pour l'aperçu du document*
context	http ://iiif.io/api/presentation/2/context.json

FIGURE 7.2 – Les clefs principales du *manifest* IIIF d'un document sur Gallica (ARK `bpt6k1513919s`)

Les données imbriquées dans la clef *metadata* peuvent varier de document à document, même quand elles sont encodées par la même institution telle que la Bibliothèque nationale de France. Comme est expliqué dans le chapitre 6, il faut déterminer l'essentiel quant aux métadonnées à mettre dans le schéma TEI. Il faut chercher les métadonnées qui s'appliquent tous les deux aux manuscrits et aux imprimés. En plus de cela, il faut aussi chercher les métadonnées qui, selon l'usage de la BnF, sont normalement encodées dans le *manifest* IIIF d'un document sur Gallica. J'ai déterminé que l'essentiel du *manifest* IIIF sont les métadonnées listées dans le Tableau 7.3.

Certaines métadonnées du *manifest* posent des avantages et des défis. L'un des avantages est la manière par laquelle la BnF encode la donnée *Relation* pour les documents mis sur Gallica. La donnée *Relation* peut se répéter parce que le fac-similé peut être lié

label	***value***
Relation	Notice du catalogue : http ://catalogue.bnf.fr/ark :/12148/cb33352789b
Repository	Bibliothèque nationale de France
Shelfmark	Bibliothèque nationale de France, département Réserve des livres rares, RES-YF-1990
Title	Les divers entretiens de la fontaine de Vaucluse : ballet dansé à la grande salle du Roure l'année 1649
Language	français
Creator	Nouguier, de. Auteur du texte
Date	1649

FIGURE 7.3 – Les métadonnées essentielles du *manifest* IIIF (ARK `bpt6k1513919s`)

aux plusieurs notices du catalogue. Cependant, elle termine toujours par l'ARK unique du document physique auquel le fac-similé est lié. Ainsi, l'application `alto2tei` dispose de la donnée *Relation* et, en traitant la fin de la chaîne, extrait l'ARK du document physique dans le catalogue général de la BnF. Un défi à surmonter est quand un type (*label*) de métadonnée n'est pas répété mais il possède plusieurs valeurs. Cela peut arriver quand un document a plusieurs auteurs. Pour cette raison, l'application `alto2tei` traite l'encodage de toute donnée portant sur l'auteur d'une manière différente que celle pour les autres métadonnées.

7.2.2 Le format UNIMARC

Étant communiquées en XML, les données du catalogue général de la BnF se trouvent sur l'arbre de la notice en format UNIMARC ; ce format leur donne le préfix `mxc` qui fait référence au *MarcXchange*. Un exemple des données UNIMARC renvoyées par l'API SRU de la BnF est montré dans la Figure 7.4b. L'exemple prend toujours le livret imprimé du ballet *Les divers entretiens de la fontaine de Vaucluse* de 1649. Le fac-similé numérique sur Gallica a l'identifiant `bpt6k1513919s`. Mais, comme expliqué ci-dessus, l'identifiant du document physique est récupéré en examinant le *manifest* IIIF du fac-similé numérique. L'ARK du document physique (`cb33352789b` dans l'exemple) permet d'interroger les données du catalogue général de la BnF grâce à son API *Search/-Retrieve via URL* (SRU).

Les données UNIMARC sont emballées dans les éléments du préfix `srw` qui fait référence au protocole SRU [2]. L'élément `<srw:recordData>` (lignes 11-15, Fig 7.4b) présente les données da la notice du catalogue trouvée. L'application `alto2tei` détermine qu'elle a bien localisé le bon document physique dans le catalogue général de la BnF si la réponse de l'API SRU ne contient qu'une seule notice. Le nombre de notices trouvées est sur la ligne 6 de la Figure 7.4b. Si ce nombre est plus que 1 cela indique que l'ARK récupérée

2. *SRU : Search/Retrieval via URL – SRU, CQL and ZeeRex (Standards, Library of Congress).* URL : `https://www.loc.gov/standards/sru/` (visité le 02/09/2022).

```python
class SRU:
  def __init__(self, ark):
    """Args:
      ark (string): document ARK in BnF catalogue"""
    self.ark = ark

  def request(self):
    r = requests.get(f'http://catalogue.bnf.fr/api/SRU?version=1.2&
    operation=searchRetrieve&query=(bib.persistentid all "ark:/12148/{self.
    ark}")')
```

(a) La requête envoyée à l'API SRU par l'application python `alto2tei`

```xml
<srw:searchRetrieveResponse xmlns:srw="http://www.loc.gov/zing/srw/" xmlns=
    "http://catalogue.bnf.fr/namespaces/InterXMarc">
<!-- ... -->
  <srw:echoedSearchRetrieveRequest>
    <srw:query>(bib.persistentid all "ark:/12148/cb33352789b")</srw:query>
  </srw:echoedSearchRetrieveRequest>
  <srw:numberOfRecords>1</srw:numberOfRecords>
  <srw:records>
    <srw:record>
      <srw:recordSchema>marcxchange</srw:recordSchema>
      <srw:recordPacking>xml</srw:recordPacking>
      <srw:recordData>
        <mxc:record format="Unimarc" id="ark:/12148/cb33352789b" type="
    Bibliographic" xmlns:mxc="info:lc/xmlns/marcxchange-v2">
<!-- data in Unimarc formatting, with prefix "mxc" -->
        </mxc:record>
      </srw:record>
    </srw:recordData>
<!-- ... -->
```

(b) La réponse de l'API SRU pour le document d'ARK `bpt6k1513919s`

FIGURE 7.4 – La requête et la réponse à l'API SRU [version 1.2]

depuis la donnée *Relation* du *manifest* IIIF est liée aux notices de plusieurs documents. Ne pouvant pas déterminer quel document est le bon, l'application `alto2tei` abandonne les données UNIMARC et compte uniquement sur les métadonnées du *manifest* IIIF.

7.2.3 Le format des données du site SUDOC

Puisque les données UNIMARC ne renseignent pas sur le lieu de l'institution qui conserve le document physique, il faut d'autre source de données pour remplir l'élément TEI `<settlement>` du `<sourceDesc>`. Pour trouver une solution, j'en ai discuté avec une spécialiste à la Bibliothèque nationale de France, Florence Tfibel. Dans un courriel, Tfibel a dit que « la localisation de l'institution n'y est effectivement nulle part renseignée *en dur* » dans les données UNIMARC du catalogue[3]. Alors elle a eu l'idée de la chercher

3. Florence TFIBEL. *RE : Tr : [Gallic(Orpor)a] Question Sur l'Unimarc Du Catalogue Généra.* E-mail. 13 juill. 2022.

sur le site web du Système Universitaire de Documentation qui gère un répertoire des centres de ressources (RCR), auquel appartient la BnF. La BnF dispose d'un numéro RCR. Son emplacement sur l'arbre des données UNIMARC du catalogue est montré dans la Figure 7.5. Le numéro RCR de la BnF est 759999999.

```
<mxc:datafield tag="930" ind1=" " ind2=" ">
<!-- ... -->
  <mxc:subfield code="b">759999999</mxc:subfield>
<!-- ... -->
</mxc:datafield>
```

FIGURE 7.5 – L'emplacement du RCR sur l'arbre des données UNIMARC

Le site web du Système Universitaire de Documentation ne dispose pas d'un genre d'API SRU aussi efficace que celui de la BnF, mais les données de son site peuvent être interrogées en traitant la page du centre de ressource qui porte le numéro cherché. Il est possible de lire la page destinée à l'exposition d'un centre de recherche, telle que la BnF, en construisant l'URL que le site attend à la suite d'une recherche. Si on recherche le numéro de la BnF sur le site de SUDOC, l'URL sera la suivante, où le dernier chiffre après la signe d'égalité est le numéro RCR de la BnF :

```
http://www.sudoc.abes.fr/cbs/xslt//DB=2.2/
    CMD?ACT=SRCHA&IKT=8888&SRT=RLV&TRM=759999999
```

L'application `alto2tei` demande à l'API du site SUDOC la page de cette URL ; ensuite, elle traite les données HTML renvoyées, voir Fig 7.6. La page HTML possède un tableau avec une une cellule (`<td>`) qui contient la phrase « Adresse postale » dans le `<span>` d'un `<div>`. Après cette cellule il y a une autre cellule avec les données portant sur l'adresse du centre. Ayant l'a localisée avec l'aide de la première cellule, l'application `alto2tei` traite les données de cette dernière cellule et extrait la ville et le nom uniforme du centre. Même si la ville et le nom de la BnF seront toujours Paris et la Bibliothèque nationale de France, respectivement, il est meilleur de s'appuyer sur les sources de données pour que l'application puisse s'adapter plus facilement aux autres utilisations.

7.2.4 Le format du fichier de configuration

Une partie importante des métadonnées s'appui aussi sur le fichier de configuration que les utilisateurs de l'application `alto2tei` peuvent facilement personnaliser. Son format est le YAML (*Yet Another Markup Language*) qui n'exige pas de grande expertise dans la programmation. Comme *l'eXtensible Markup Language* (XML) le YAML est ce qu'on appelle un langage de « Mark Up » et donc il imbrique des données. Mais au lieu d'utiliser les éléments XML (`< >`), le YAML utilise simplement les deux points et de la tabulation.

```
<tr>
  <td class="rec_lable">
    <div>
      <span>Adresse postale : </span>
    </div>
  </td>
  <td clas="rec_title">
    <div>
      <span>Bibliothèque nationale de France</span>
    </div>
    <div>
      <span>quai Francois Mauriac </span>
    </div>
    <div>
      <span>75706 Paris CEDEX 13</span>
    </div>
    <div>
      <span>France</span>
    </div>
    <div>
      <span> </span>
    </div>
  </td>
</tr>
```

FIGURE 7.6 – Les données cherchées depuis le site SUDOC

Ainsi les utilisateurs de divers spécialités peuvent adapter l'application `alto2tei` et le pipeline *Gallic(orpor)a* aux données de leur propre projet.

Le fichier de configuration informe trois aspects de la création du document TEI. Dans un premier temps, il informe l'application `alto2tei` où il peut trouver les fichiers ALTO qui a été traités par les modèles HTR. Cette donnée est personnalisable pour qu'une utilisatrice ou un utilisateur puisse transformer les transcriptions ALTO stockés sous divers dossiers. Dans un deuxième temps, le fichier de configuration informe la requête que l'application `alto2tei` envoie à l'API IIIF, la même API qui réserve d'accès aux images et aux métadonnées du fac-similé numérique et qui est discutée dans la section 7.2.1. L'application `alto2tei` permet de personnaliser la requête envoyée à l'API IIIF selon les normes de l'institution qui la gère. Le fichier configuré pour l'API IIIF gérée par la BnF est montré dans la Figure 7.7. Les paramètres `manifest_prefix` et `image_prefix` sont pareils pour l'API IIIF de la BnF, mais le fichier de configuration les garde distinct afin qu'ils puissent s'adapter aux APIs qui distinguent entre l'URL de l'IIIF Image API et de la Presentation API.

Enfin, le fichier de configuration informe les éléments descendant du `<publicationStmt>` dans le `<fileDesc>` du `<teiHeader>`. Les métadonnées récupérées par l'application `alto2tei` ainsi que leur format en YAML sont montrés dans la Figure 7.8. Le nom du projet (ligne 18, Fig. 7.8) et les noms des membres de l'équipe peuvent être personnalisés. Le prénom ou les prénoms de chaque membre sont renseignés par la chaîne entre guillemets qui

```
iiifURI:
  scheme: "https"
  server: "gallica.bnf.fr"
  manifest_prefix: "/iiif/ark:/12148/"
  image_prefix: "/iiif/ark:/12148/"
  manifest_suffix: "/manifest.json"
```

FIGURE 7.7 – La partie du fichier de configuration portant sur la requête envoyée à l'API IIIF

suit les deux points après le terme *forename*. Le terme *surname* indique le même chose pour le nom de famille. L'identifiant du membre peut être personnalisé soit en le type d'identifiant, tel que ORCID, soit en la valeur qui suit les deux points après le terme *target*.

```
responsibility:
  text: "Transformation from ALTO4 to TEI by"
  resp: [{"forename":"Kelly",
    "surname":"Christensen",
    "ptr":{"type":"orcid","target":"000000027236874X"}},

    {"forename":"Simon",
    "surname":"Gabay",
    "ptr":{"type":"orcid","target":"0000000190944475"}},

    {"forename":"Ariane",
    "surname":"Pinche",
    "ptr":{"type":"orcid","target":"0000000278435050"}}
    ]
  # name the publisher of this digital edition (project name)
  publisher: "Gallic(orpor)a"
  # name which authority finances or supports the project
  authority: "BnF DATALab"
  # specify the edition's availability to the public/licencing
  availability: {"status": "restricted", "n":"cc-by"}
  licence: {"target":"https://creativecommons.org/licenses/by/4.0/"}
```

FIGURE 7.8 – La partie du fichier de configuration portant sur la requête envoyée à l'API IIIF

7.3 La mise en place des données récupérées

Ayant découvert les structures de données des sources de données externes, il faut parler d'où ses données ainsi récupérées vont dans le `<teiHeader>`. Les métadonnées intégrées au document TEI s'appuient sur (1) le *manifest* IIIF envoyé par l'API IIIF en format JSON, (2) les données UNIMARC du catalogue général de la BnF envoyées par l'API SRU en format XML, (3) les données HTML récupérées à partir de la page de recherche du site SUDOC, et (4) le fichier de configuration en format YAML. L'appli-

cation recherche des données dans deux autres sources : (5) les fichiers XML ALTO et (6) la commande saisie pour démarrer l'application. Dans un premier temps, l'application `alto2tei` compte les fichiers ALTO du fac-similé numérique et utilise ce nombre pour l'élément `<extent>`. Dans un deuxième temps, la commande saisie pour la démarrer exige un paramètre qui déclare la version de l'engin *Kraken* qui a été utilisée pour mettre en œuvre les modèles HTR et réaliser les transcriptions.

7.3.1 Les sources des données du `<titleStmt>`

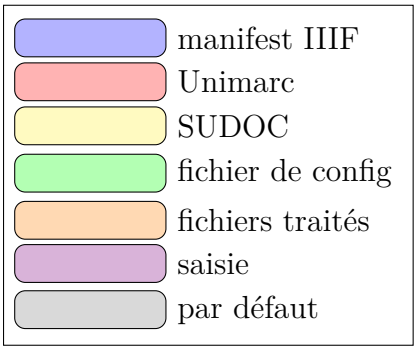
La Figure 7.9 visualise les sources de données du `<titleStmt>`. Les éléments portant sur le titre de la ressource (`<title>`) et sur les personnes auxquels est attribuée la propriété intellectuelle du texte représenté (`<author>`) sont informés par l'un de deux sources de données possibles. Soit les données du catalogue général donnent beaucoup d'information sur les auteurs ainsi qu'un titre propre, soit le *manifest* IIIF donne simplement le titre du texte et les noms non annotés des auteurs. Le choix entre les deux sources de données se fait à partir de la disponibilité des données du catalogue de la BnF. Normalement, l'application `alto2tei` met en priorité les données du catalogue.

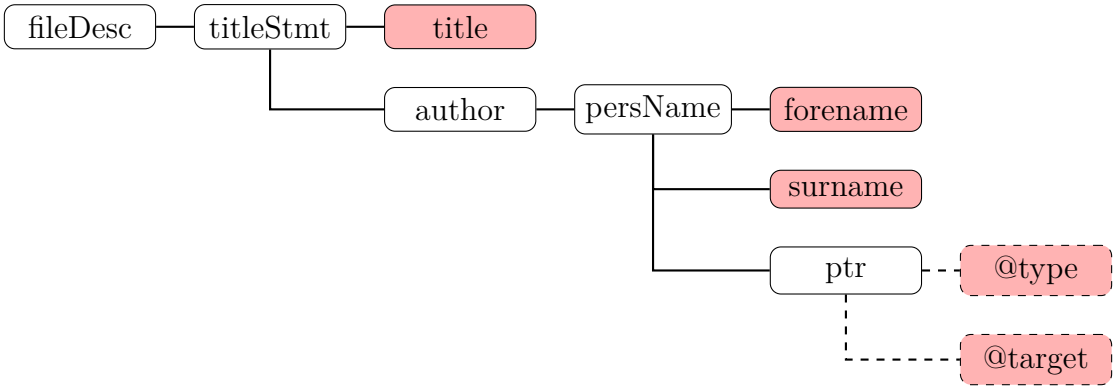
La Figure 7.9a visualise les éléments du `<titleStmt>` dans le cas où la relation entre le fac-similé numérique sur Gallica et le document physique appartenant à la BnF est établie. Cela veut dire que le document dont parle la notice du catalogue est le même document à partir duquel le fac-similé numérique sur Gallica a été fait. Quand la notice du bon document est trouvée, les données UNIMARC cherchées depuis le catalogue sont (1) le titre uniforme du document et (2) le nom et l'identifiant de ses auteurs. Les éléments descendants du `<respStmt>` s'appuient toujours sur le fichier de configuration afin qu'ils puissent être personnalisées par les personnes qui utilisent l'application `alto2tei` pour générer leurs propres documents TEI. Toutes ces sources ainsi que la destination de leurs données sont visualisées dans la Figure 7.9. Les Tableaux 7.10 et 7.11 décrivent d'où viennent chaque donnée des sources de données en JSON et XML, soit le *manifest* IIIF et le catalogue général, respectivement [4]

7.3.2 Les sources des données du `<extent>`

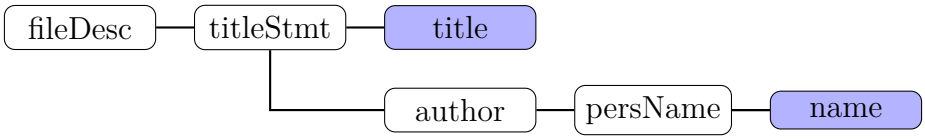
L'élément `<extent>` du `<teiHeader>`, qui indique le nombre de fichiers traités par l'application `alto2tei` lors de la création de la ressource numérique, s'appuie simplement sur un compte des fichiers. L'application a besoin de traiter tout fichier XML ALTO dans le dossier du document, comme expliqué dans la section 4.1.1, et mettre ses données dans le `<sourceDoc>`, selon la modélisation de l'application `alto2tei`, voir la Figure 7.12. Elle

4. Pour les attributs d'un élément XML, la table utilise le syntaxe *XPath* par lequel l'attribut est déclaré avec l'arobase. Par exemple, l'attribut `@att` d'un élément `<element>`, qui descende de l'élément `<parent>`, serait représenté dans le format suivant : `parent/element[@att]`.

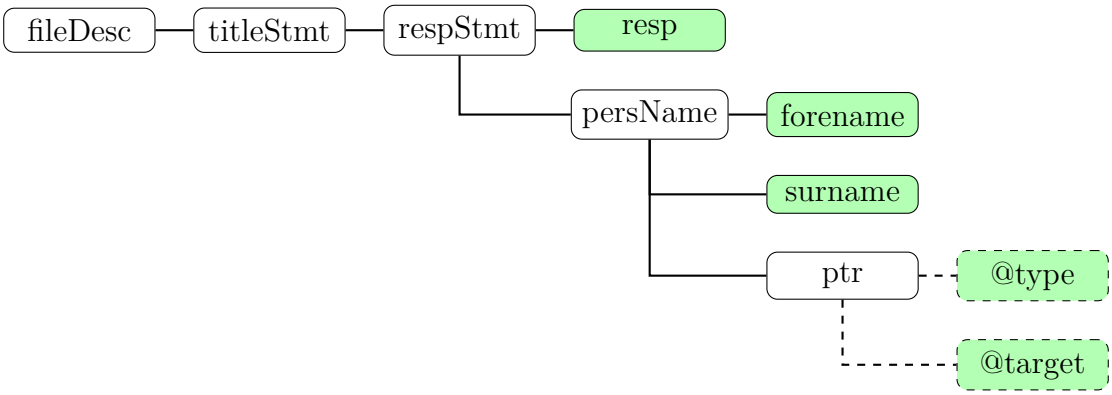




(a) Les sources de données du `<title>` et du `<author>` quand la relation entre le fac-similé numérique et le document physique est établie



(b) Les sources de données du `<title>` et du `<author>` quand la relation entre le fac-similé numérique et le document physique *n'est pas* établie



(c) Les sources de données du `<respStmt>` dans tout cas

FIGURE 7.9 – Les sources de données des éléments du `<titleStmt>`

élément ou attribut TEI	*manifest* IIIF en JSON	données XML UNIMARC
author/ name	"Creator"	–
author/[@xml:id]	2 premières caractères de la valeur "Creator" + chiffre	2 premières caractères de la valeur datafield[@tag="700"]/ subfield[@code="a"] ou datafield[@tag="703"]/ subfield[@code="a"] + chiffre
persName/ forename	–	datafield[@tag="700"]/ subfield[@code="b"] ou datafield[@tag="703"]/ subfield[@code="b"] sans les éléments suivants : *van der, de la, de, du, von, van*
persName/ namelink	–	l'un des éléments *van der, de la, de, du, von,* ou *van* extrait du datafield[@tag="700"]/ subfield[@code="b"] ou datafield[@tag="703"]/ subfield[@code="b"]
persName/ surname	–	datafield[@tag="700"]/ subfield[@code="a"] ou datafield[@tag="703"]/ subfield[@code="a"]
persName/ ptr[@target]	–	datafield[@tag="700"]/ subfield[@code="o"] ou datafield[@tag="703"]/ subfield[@code="o"]

FIGURE 7.10 – *Mapping* des données du <author> utilisé dans le <titleStmt> et le <sourceDesc>

élément ou attribut TEI	*manifest* IIIF	données UNIMARC
titleStmt/ title	"Title"	datafield[@tag="200"]/ subfield[@code="a"]

FIGURE 7.11 – *Mapping* des données du <titleStmt>

peut également compter le nombre des fichiers XML ALTO qui appartiennent au document. Ce compte elle met comme la « taille » de la ressource numérique, dans l'élément `<measure>`.

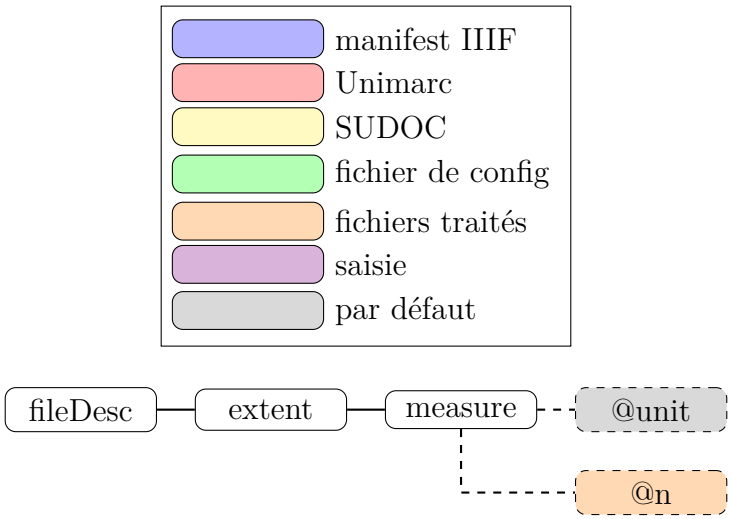


FIGURE 7.12 – La source des données du `<extent>`

7.3.3 Les sources des données du `<publicationStmt>`

La distribution de la ressource numérique est décrite par le `<publicationStmt>`. La Figure 7.18 montre comment toute métadonnée du `<publicationStmt>` vient du fichier de configuration, où les utilisateurs du pipeline et/ou de l'application peuvent personnaliser les détails de leur projet et les droits d'utilisation.

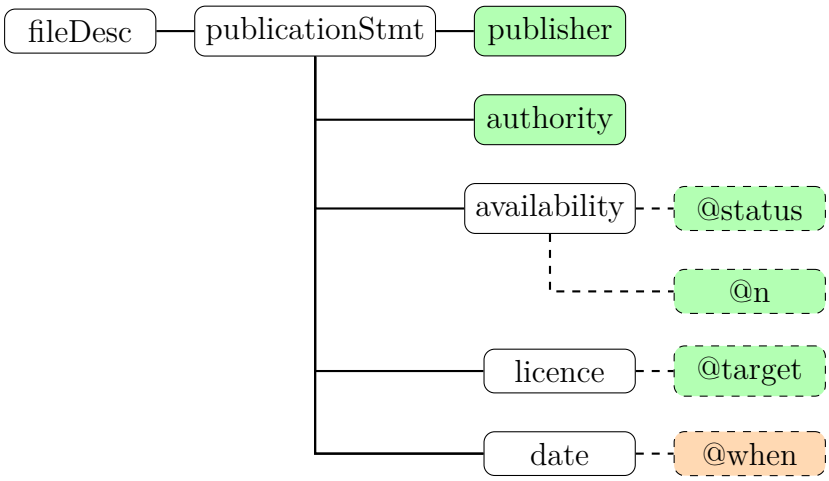


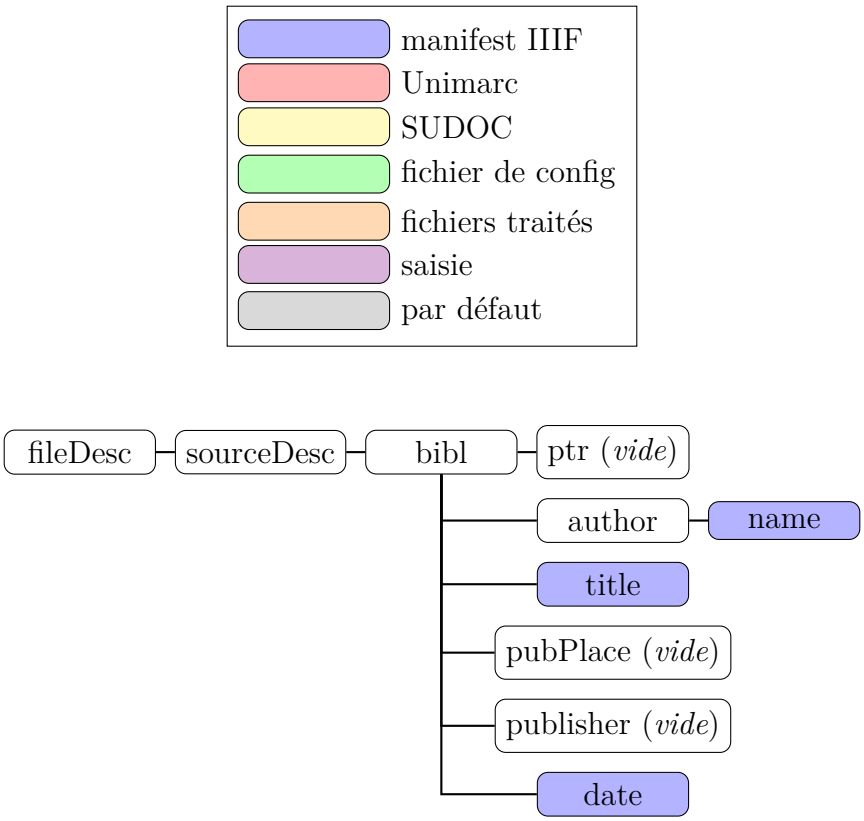
FIGURE 7.13 – La source des données du `<publicationStmt>`

7.3.4 Les sources des données du <sourceDesc>

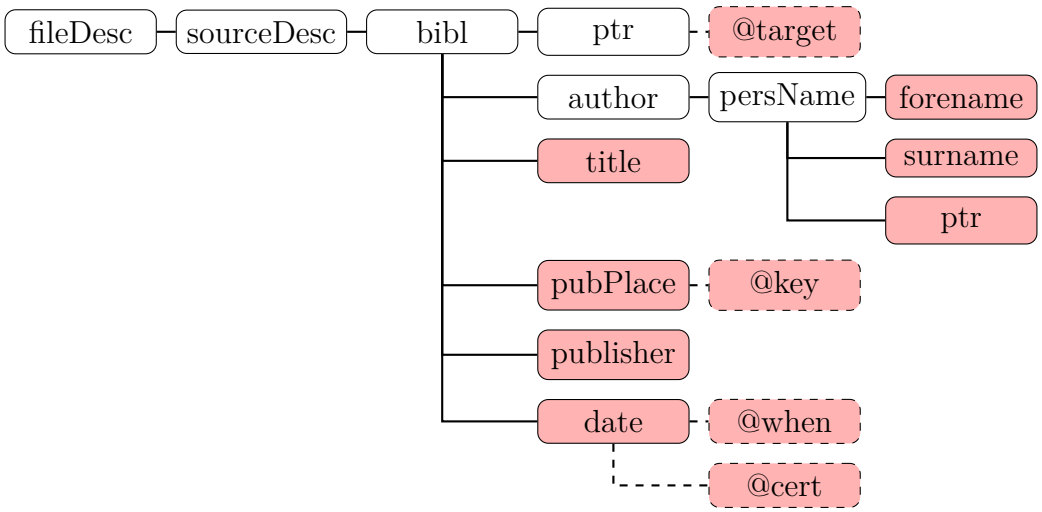
Le <sourceDesc> décrit le document source dont la transcription et le texte sont représentés par la ressource numérique. Il se compose de deux éléments principales. Dans un premier temps, le <sourceDesc> contient le <bibl> qui est une description de la source, portant sur les auteurs, le titre, l'éditeur, la date et le lieu de création. Dans un deuxième temps, le <sourceDesc> contient le <msDesc> qui est une citation bibliographique du document source. Si besoin, l'application alto2tei peut livrer un <sourceDesc> en utilisant uniquement le *manifest* IIIF. Mais il n'est pas une source sûre sur l'éditeur ni sur le lieu de publication du document source. Les données UNIMARC du catalogue général de la BnF, par contre, portent toujours sur ces détails. Même si le catalogue général dit que la donnée n'est pas connue, la notice renseigne toujours sur l'éditeur et le lieu de publication du document et l'application alto2tei répète l'information incertaine du catalogue. Les Tableaux 7.14 et 7.17 précisent quelles données sont intégrées dans le <sourceDesc> et lesquelles sont laissées vides si la relation entre le fac-similé numérique et le document physique n'est pas établie. Les Figures 7.15 et 7.16 visualisent les source de données pour ces deux éléments selon leur disponibilité.

élément ou attribut TEI	*manifest* IIIF	données UNIMARC
sourceDesc/ bibl/ ptr[@target]	–	controlfield[@tag="003"]
sourceDesc/ bibl/ title	"Title"	datafield[@tag="200"]/ subfield[@code="a"]
sourceDesc/ bibl/ pubPlace	–	datafield[@tag="210"]/ subfield[@code="a"]
sourceDesc/ bibl/ pubPlace[@key]	–	datafield[@tag="102"]/ subfield[@code="a"]
sourceDesc/ bibl/ publisher	–	datafield[@tag="210"]/ subfield[@code="c"]
sourceDesc/ bibl/ date	"Date"	Caractères 9-13 de datafield[@tag="100"]/ subfield[@code="a"]
sourceDesc/ bibl/ date[@when	–	Caractères 9-13 de datafield[@tag="100"]/ subfield[@code="a"]
sourceDesc/ bibl/ date[@cert	–	8e caractère de datafield[@tag="100"]/ subfield[@code="a"]

FIGURE 7.14 – *Mapping* des données du <bibl> du <sourceDesc>

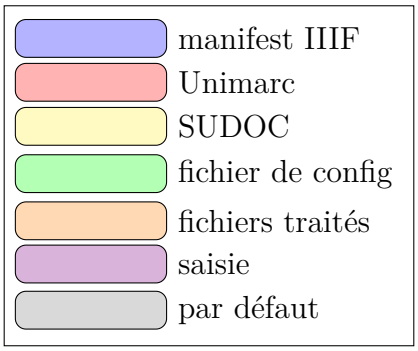


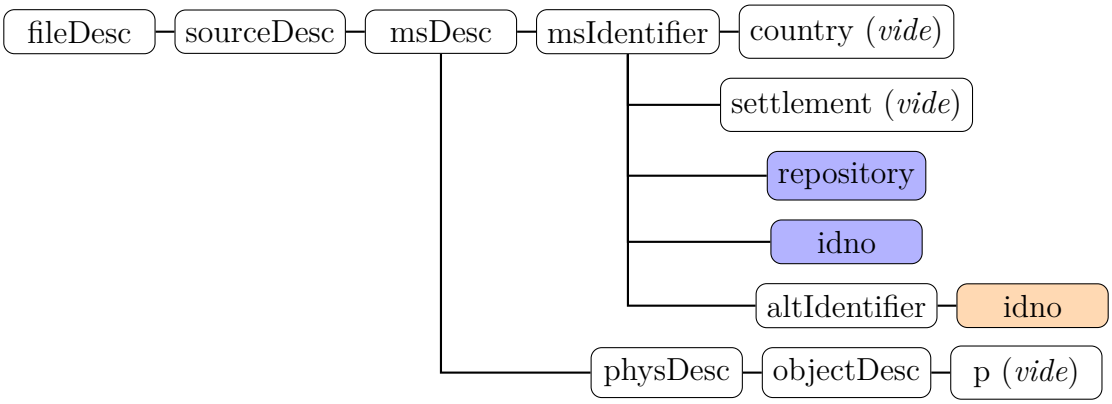
(a) Quand la relation entre le fac-similé numérique (renseigné par le *manifest* IIIF et le document physique (renseigné par catalogue général de la BnF) n'est pas établie



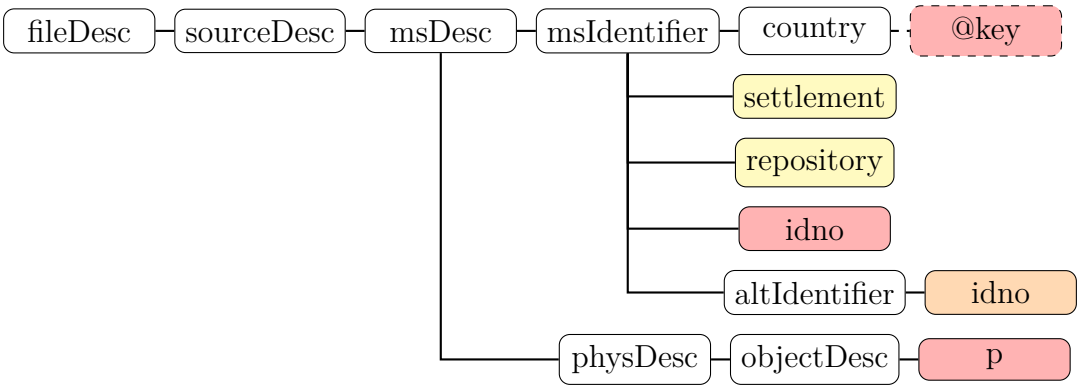
(b) Quand la relation entre le fac-similé numérique et le document physique est établie

FIGURE 7.15 – Les sources des données du `<bibl>` du `<sourceDesc>`





(a) Quand la relation entre le fac-similé numérique (renseigné par le *manifest* IIIF et le document physique (renseigné par catalogue général de la BnF) n'est pas établie



(b) Quand la relation entre le fac-similé numérique et le document physique est établie

FIGURE 7.16 – Les sources des données du `<msDesc>` du `<sourceDesc>`

élément ou attribut TEI	*manifest* IIIF	données UNIMARC
`sourceDesc/ msDesc/ msIdentifier[@key]`	–	`datafield[@tag="801"]/ subfield[@code="a"]`
`sourceDesc/ msDesc/ msIdentifie repository`	`"Repository"`	donnée du site SUDOC en utilisant le numéro RCR du `datafield[@tag="930"]/ subfield[@code="b"]`
`sourceDesc/ msDesc/ msIdentifier idno`	`"Shelfmark"`	`datafield[@tag="930"]/ subfield[@code="a"]`
`sourceDesc/ msDesc/ physDesc/ objectDesc/ p`	–	`datafield[@tag="200"]/ subfield[@code="b"]`

FIGURE 7.17 – *Mapping* des données du `<msDesc>` du `<sourceDesc>`

7.3.5 Les sources des données du `<profileDesc>`

Selon notre modélisation, qui est justifiée dans le chapitre 6, le `<profileDesc>` est très minimal. Il ne présente que de la donnée sur la langue utilisée dans le document source. Dans sa version actuelle, l'application `alto2tei` ne supporte qu'une seule langue. Néanmoins, malgré la simplicité conceptuelle d'un tel `<profileDesc>`, son élément `<language>` compte sur deux sources de données. Le mot balisé dans l'élément `<language>` vient du *manifest* IIIF puisque la langue est l'une des données déterminée essentielle et recherchée lors du traitement du *manifest*. La valeur de l'attribut `@ident`, par contre, est récupérée depuis le catalogue général de la BnF parce qu'elle est une clef standardisée pour la langue utilisée. La Figure 7.18 visualise les deux sources de données pour le `<profileDesc>`. Le Tableau 7.19 précise la localisation de ces deux données.

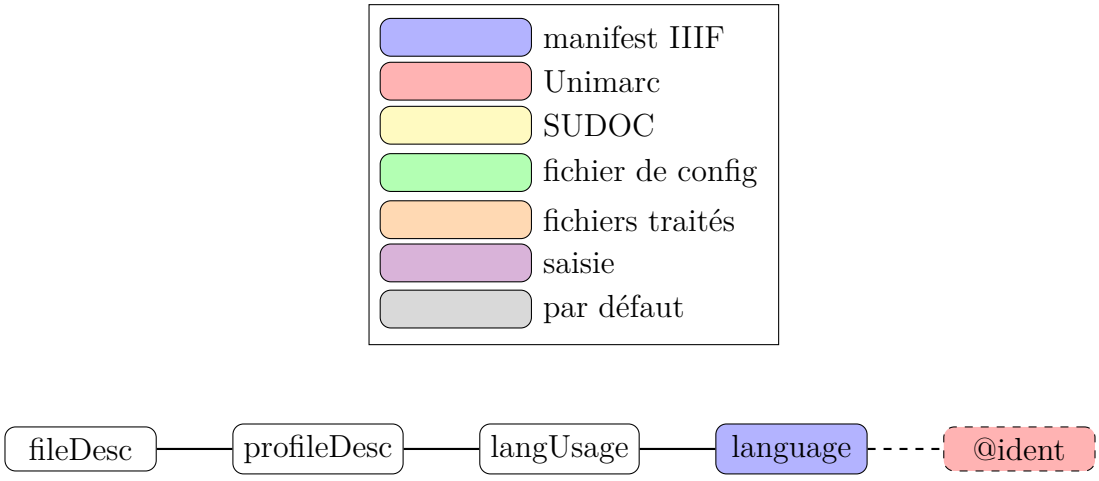


FIGURE 7.18 – Les sources des données du `<profileDesc>`

élément ou attribut TEI	*manifest* IIIF	données UNIMARC
`profileDesc/ langUsage/ language`	`"Language"`	–
`profileDesc/ langUsage/ language[@ident]`	–	`datafield[@tag="101"/ subfield[@code="a"]`

FIGURE 7.19 – *Mapping* des données du `<profileDesc>`

7.3.6 Les sources des données du `<encodingDesc>`

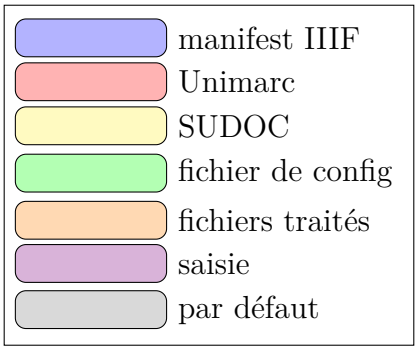
La description de l'encodage et de l'appareil qui a fait la transcription sont essentiels. Sans eux, la transcription présentée dans la ressource ne pourra pas être reproduite. Et sans pouvoir reproduire la transcription, les scientifiques ne peuvent pas la vérifier. Pouvoir reproduire des résultats scientifiques est fondamental à la recherche.

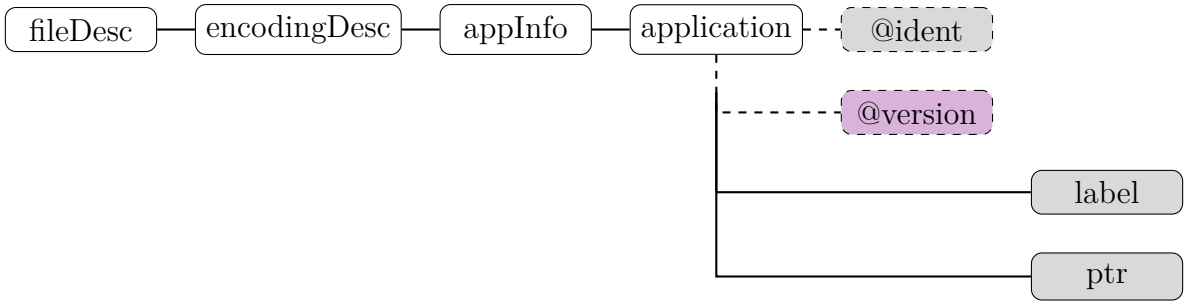
Le `<encodingDesc>` contient deux descendants directs. Dans un premier temps, il contient de l'information sur l'engin qui a mis en œuvre les modèles HTR (`<application>`). Le pipeline *Gallic(orpor)a* attend toujours l'engin *Kraken* et la version actuelle de l'application `alto2tei` ne renseigne que sur les versions de cet engin ; la version est donnée dans l'élément `<appInfo>`. Quand l'application `alto2tei` est démarré depuis la ligne de commande, un paramètre obligatoire déclare quelle version de *Kraken* a été utilisée pour créer les fichiers XML ALTO qu'elle va traiter. L'application `alto2tei` tient ce paramètre et l'utilise pour renseigner sur la version de l'application *Kraken* dans l'élément `<application>`. Si l'application est intégrée dans le pipeline *Gallic(orpor)a*, la version de *Kraken* est récupérée depuis la liste de librairies installées pour appliquer les modèles HTR.

Dans un deuxième temps, le `<encodingDesc>` s'appuie sur les fichiers XML ALTO que l'application `alto2tei` traite. L'application analyse les étiquettes de ligne et de zone que les modèles HTR ont appliquées. Ensuite, elle compare les étiquettes trouvées avec une liste des étiquettes du vocabulaire *SegmOnto*. Pour chaque type d'étiquette utilisé dans les fichiers XML ALTO qui appartient au vocabulaire *SegmOnto*, l'application `alto2tei` la déclare dans la bonne catégorie de l'élément `<taxonomy>` du `<classDecl>`, voir la sous-figure 7.20b.

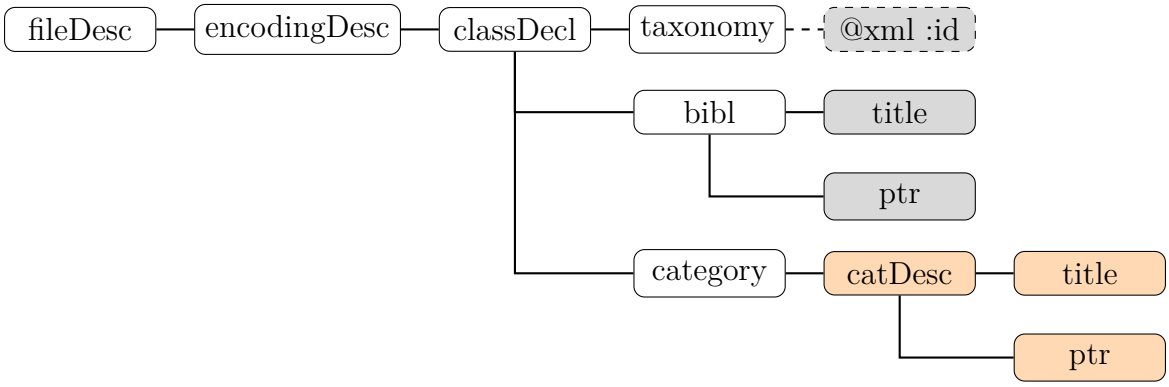
En outre, puisque l'équipe du projet *Gallic(orpor)a* a choisi de s'appuyer sur le vocabulaire *SegmOnto* (cf. Chap. 2), certains éléments du `<classDecl>` sont encodés en dur. La taxonomie déclarée dans l'identifiant de l'élément `<taxonomy>` ainsi que le titre attribué à la taxonomie (`<title>`) est toujours « SegmOnto ». De plus, le *pointer* (`<ptr>`) qui indique l'URL où se trouvent des informations mises à jour sur le vocabulaire utilisé est toujours celui du projet *SegmOnto*. Ces données ne sont pas personnalisables. En plus, bien que les descendants des classes (`<category>`) des lignes et des zones soient informés par les fichiers ALTO, les données encodées dans les éléments viennent plutôt d'une liste que l'application `alto2tei` maintient. Cette liste encodée en dur porte l'URL de toute

zone et toute ligne sur le site web du projet *SegmOnto*.





(a) La source de données du `<appInfo>`



(b) La source de données du `<classDecl>`

FIGURE 7.20 – Les sources de données du `<encodingDesc>`

Chapitre 8

La génération du <sourceDoc>

À partir d'une image numérique, les modèles HTR disposant de l'engin *Kraken* produisent une transcription de la page en format XML ALTO (*Analyzed Layout and Text Object*). Comme expliqué et justifié dans le chapitre 4, ce format est transformé en le format *pivot* du pipeline *Gallic(orpor)a*, la TEI (*Text Encoding Initiative*). Le schéma TEI est flexible et il permet d'encoder la transcription d'un document texte en plusieurs façons. La manière préférée par l'équipe du projet *Gallic(orpor)a* s'appuie sur l'élément TEI `<sourceDoc>`. Selon les *guidelines* de la *Text Encoding Initiative*, l'élément `<sourceDoc>` contient « *a transcription or other representation of a single source document potentially forming part of a dossier génétique or collection of sources* » [1]. L'autre option est l'élément `<facsimile>`. Il contient « *a representation of some written source in the form of a set of images rather than as transcribed or encoded text* » [2].

Entre les deux options, l'élément `<souceDoc>` convient mieux à la transcription encodée dans un fichier XML ALTO puisqu'il est destiné à traiter la représentation d'une source et un fichier XML ALTO contient la représentation d'une image de texte. L'élément `<facsimile>` pourrait bien servir à l'encodage de l'image elle-même, mais la prédiction et/ou transcription de son contenu est mieux encodée avec l'élément `<sourceDoc>`. Après tout, il faut se souvenir que le pipeline *Gallic(orpor)a* vise à conserver dans le document TEI toute donnée de la transcription du fichier XML ALTO. L'un des objectifs du projet est que la ressource numérique produite par le pipeline permet de recréer des fichiers XML ALTO à partir du document TEI, afin que des utilisateurs puissent utiliser les fichiers XML ALTO reconstruits comme des vérités de terrain et entraîner des nouveaux modèles HTR. Il faut donc un élément TEI destiné à la transcription d'une image, au lieu de l'image elle-même.

Par rapport au *mapping* des données du `<teiHeader>`, qui est expliqué dans le chapitre 7, le *mapping* des données du `<sourceDoc>` est plus direct et fixé. Tandis que

1. *TEI element sourceDoc.*
2. *TEI element facsimile.* URL : `https://tei-c.org/release/doc/tei-p5-doc/en/html/ref-facsimile.html` (visité le 03/09/2022).

le contenu du <teiHeader> compte sur la disponibilité variable des données depuis les divers sources de données, le contenu du <sourceDoc> est très prévisible et compte sur un schéma ALTO qui est très systématique. La manière de transcrire les pages numérisées ne change pas et s'effectue par les mêmes modèles HTR bien que les documents transcrits soient différents de l'un à l'autre.

Cependant, il y a une variation possible dans la granularité de la transcription. D'une part, le fichier XML ALTO peut porter des prédictions sur les caractères d'un mot, y compris leur contenu et leur emplacement sur la page. De l'autre part, le fichier XML ALTO peut porter des prédictions sur une ligne de texte, où son contenu est une chaîne des mots et des espaces entre mots. Dans ce dernier cas plus simple, le fichier XML ALTO présente moins de détail par rapport à l'autre forme. L'application alto2tei s'adapte aux deux puisqu'elles sont toutes les deux valables et produites par l'engin *Kraken*. Depuis la ligne de commande, l'engin *Kraken* met en pratique des modèles HTR et produit les fichiers XML ALTO qui contiennent des prédictions sur les caractères ou les « glyphes » des mots de la ligne de texte. Depuis l'interface *eScriptorium*, la même version de *Kraken* et les mêmes modèles HTR produisent des fichiers XML ALTO qui ne contiennent que des prédictions sur la ligne de texte, n'allant pas en détail sur les caractères et les mots ou les « segments ». Les formes différentes sont expliquées dans la section 5.1.1 du chapitre 5.

8.1 Le modèle du <sourceDoc>

L'application alto2tei récupère toute donnée significative des fichiers XML ALTO et les met sur un arbre TEI qu'elle construit, spécifiquement sur la branche de l'élément <sourceDoc>. L'arborescence du <sourceDoc> prendre deux formes, selon le détail du fichier XML ALTO que des modèles HTR ont produit. Si les modèles avaient enregistré des prédictions sur les glyphes et les segments dans le fichier XML ALTO, le <sourceDoc> aurait quatre niveaux d'élément <zone> pour porter sur les masques d'un bloc de texte, d'une ligne dans le bloc, d'un mot dans la ligne, et d'un caractère dans le mot. Le contenu textuel serait donc représenté deux fois ; il serait balisé dans l'élément qui porte sur la prédiction du caractère et aussi dans un élément qui représente tous caractères prédits dans une ligne de texte. L'exemple des quatre niveaux est montré dans la Figure 8.1. Si les modèles avaient enregistré uniquement des prédictions sur les blocs et les lignes de texte, le <sourceDoc> aurait deux niveaux d'élément <zone>, un pour représenter le bloc prédit et l'autre pour la ligne prédite. L'exemple de cette arborescence plus simple est montré dans la Figure 8.2.

```
<sourceDoc>
  <surface><!-- Région d'une page -->
<!-- ... -->
    <zone type="SegmOntoZone"><!-- Région d'un bloc de texte -->
<!-- ... -->
      <zone type="SegmOntoLine"><!-- Région d'une ligne de texte (ex. "
    Texte ici.") -->
<!-- ... -->
        <zone type="String"><!-- Région d'un segment dans la ligne (ex. "
    Texte") -->
<!-- ... -->
          <zone type="Glyph"><!-- Région d'un caractère dans le segment (ex
    . "T") -->
<!-- ... -->
            <c>T</c>
          </zone>
<!-- ... -->
        <zone type="Space"/><!-- Région d'une espace entre mots dans la
    ligne -->
<!-- ... -->
        <line>Texte ici.</line>
      </zone>
    </zone>
  </surface>
</sourceDoc>
```

FIGURE 8.1 – Le `<sourceDoc>` de quatre niveaux de masques imbriqués

8.1.1 La page

Normalement, un document XML ALTO représente la transcription d'une seule page du document source. Dans le schéma ALTO, des données portant sur une page sont imbriquées dans l'élément `<Page>` dont les attributs décrivent le longueur (`@WIDTH`) et l'hauteur (`@HEIGHT`), ainsi que le compte d'image (`@PHYSICAL_IMAGE_NR`) dans la suite des images traitées. Contrairement aux données susdites, il ne faut pas conserver l'identifiant (`@ID`) donné à la page. En reconstituant un fichier XML ALTO à partir du `<sourceDoc>`, peu important quel identifiant peut être donné à la nouvelle `<Page>` pourvu qu'il soit unique. L'application `alto2tei` génère un nouveau identifiant pour la `<Page>` et pour tout élément TEI qu'elle construit.

Des règles générales sur l'identifiant de l'élément TEI

L'identifiant de tout élément descendant du `<sourceDoc>` se construit de certains composants qui s'accumulent. Le parent, c'est-à-dire la page, porte tout simplement l'identifiant de la page, soit le numéro du folio du fac-similé numérique. Tout élément descendant et imbriqué dans la page retient cette donnée dans son identifiant, en y ajoutant à la suite encore plus de données. Par exemple, pour l'onzième folio d'un fac-similé numérique, l'identifiant de la page serait « f11 ». L'identifiant du premier bloc du texte sur l'onzième

```
<sourceDoc>
  <surface><!-- Région d'une page -->
<!-- ... -->
    <zone type="SegmOntoZone"><!-- Région d'un bloc de texte -->
<!-- ... -->
      <zone type="SegmOntoLine"><!-- Région d'une ligne de texte (ex. "
    Texte ici.") -->
        <line>Texte ici</line>
      </zone>
    </zone>
  </surface>
</sourceDoc>
```

FIGURE 8.2 – Le `<sourceDoc>` de deux niveaux de masques imbriqués

folio porterait donc l'identifiant « f11-textblock_0-blockCount1 ». L'identifiant du bloc se compose de l'identifiant de la page, l'identifiant donné au premier élément `<TextBlock>` par les modèles HTR et enfin une traduction de ce dernier composant en « blockCount1 » qui commence compter les blocs à partir du numéro 1 au lieu d'à partir de zéro, comme font souvent les logiciels HTR. Pour donner encore un exemple, l'identifiant de la première ligne de texte du premier bloc sur l'onzième folio porterait l'identifiant « f11-textblock_0-textline_0-lineCount1 ». Encore, l'identifiant se compose des étiquettes des éléments parents (f11, textblock_0) ainsi qu'une traduction du dernier composant (textline_0) en une chaîne plus logique (lineCount1).

La page en particulier

Selon notre modélisation, l'élément TEI `<surface>` représente une page et contient donc toute donnée encodée dans l'élément ALTO `<Page>` et son enfant `<PrintSpace>`. Souvent le fichier XML ALTO présente le longueur et l'hauteur de la page dans les éléments `<Page>` et `<PrintSpace>`. Cet redondance se produit au niveau de la page parce que l'entièreté de la page traitée est aussi ce qui est transcrit. Quand on construit un fichier XML ALTO à partir du document TEI il faut recréer cette redondance en répétant le longueur et l'hauteur dans les deux éléments ALTO. La transformation en TEI représente ces deux données dans un seul élément, le `<surface>`. Imbriqué dans l'élément `<surface>`, l'élément `<graphic>` combine l'ARK du fac-similé numérique et l'URI IIIF pour l'IIIF Image API de la BnF. Cet URI renvoie l'image entière de la page numérisée. L'exemple du `<Page>` et l'exemple de sa modélisation en TEI sont donnés dans la Figure 8.3. Une visualisation de la transformation d'ALTO à TEI est donnée à la fin de ce chapitre, dans la Figure 8.8.

```
<Layout>
  <Page WIDTH="2568" HEIGHT="3631" PHYSICAL_IMG_NR="2" ID="page_2">
    <PrintSpace HPOS="0" VPOS="0" WIDTH="2568" HEIGHT="3631">
<!-- ... -->
    </PrintSpace>
  </Page>
</Layout>
```

(a) Le <Page> en ALTO

```
<surface xml:id="f11" n="2" ulx="0" uly="0" lrx="2568" lry="3631">
  <graphic url="https://gallica.bnf.fr/iiif/ark:/12148/btv1b8610802d/f11/
    full/full/0/native.jpg"/>
<!-- ... -->
</surface>
```

(b) Le <surface> en TEI

FIGURE 8.3 – La modélisation du <Page>

8.1.2 Le bloc

La *SegmOntoZone* indique un bloc sur la page. Elle peut contenir du texte, comme dans le cas d'une *MainZone*, ou elle peut n'en avoir pas, comme dans le cas d'une *GraphicZone* qui décrit la région d'une page dans laquelle se trouve un dessin. Un fichier XML ALTO encode tout type de bloc dans l'élément <TextBlock> même s'il ne contient pas du texte. Étant modélisées en TEI, les données du <TextBlock> sont transformées en l'élément <zone>.

Des règles générales pour l'élément <zone> dans le modèle TEI

Avant d'aller en plus de détail particulier à la transformation du <TextBlock> en TEI, il faut parler de certaines transformations généralisées pour plusieurs éléments du fichier XML ALTO. Dans notre modélisation, l'élément TEI <zone> représente plus que le <TextBlock> du schéma ALTO. En fait, il représente tout masque prédit par des modèles HTR. La région prédite d'un bloc (*SegmOntoZone*) et celle d'une ligne de texte (*SegmOntoLine*) ainsi que celle d'un mot et celle d'un glyphe sont toutes représentées par l'élément TEI <zone>. Il est bien adapté à représenter les données géométriques d'un masque.

Le <zone> doit porter certains attributs d'usage, peu importe quel type de masque il représente. Ces attributs décrivent l'étiquette (@type) appliquée à la région décrite et les quatre coordonnées (@HPOS, @VPOS, @WIDTH, @HEIGHT) du rectangle qui l'encadre. Comme explique la section 5.1.1, les valeurs des attributs @HPOS et @VPOS font les coordonnées x et y, respectivement, du point le plus haut à gauche du rectangle, comme se voit dans la Figure 5.2. La valeur de l'attribut @HEIGHT compte la différence entre le point le plus

haut et le point le plus bas du rectangle. La valeur de l'attribut `@WIDTH` calcule aussi la différence entre le côté gauche du carré et son côté droit. En outre, les quatre coordonnées du rectangle se sont transformés afin de construire l'attribut `@source` pour tout `<zone>`. Le `@source` fournit l'URL pour visionner la région de l'image dans un API IIIF. Selon les normes de l'IIIF, l'URL se compose des parties suivantes :

titre	exemple
scheme	https ://
server	gallica.bnf.fr
prefix	/iiif/ark :/12148
identifier (/ARK/folio)	/btv1b8610802d/f11
nombre de pixels entre la position 0 et la position la plus à gauche de la région sur l'axe des x (`@HPOS` en ALTO)	323
nombre de pixels entre la position 0 et la position la plus en haute de la région sur l'axe des y (`@VPOS` en ALTO)	336
nombre de pixels entre la position la plus à gauche et celle la plus à droite sur l'axe des x (`@WIDTH` en ALTO)	2056
nombre de pixels entre la position la plus en haute et celle la plus en bas sur l'axe des y (`@HEIGHT` en ALTO)	2812
size	full
rotation	0
quality	native
.format	.jpg

Les composants de la table ci-dessus constituent l'URL suivant :

```
https://gallica.bnf.fr/iiif/ark:/12148/btv1b8610802d/f11
    /323,336,2056,2812/full/0/native.jpg
```

Cette URL se donne comme la valeur de l'attribut `@source` des éléments `<zone>` dans notre modélisation TEI. Elle permet de visionner la partie du fac-similé numérique concernée depuis un éditeur, tel que *TEIPublisher*, qui requête l'image de l'API IIIF.

Normalement, les modèles HTR d'aujourd'hui prédisent le rectangle qui encadre la région sur la page et aussi le polygone qui fait un masque plus précis. Si le modèle produit les deux formes de masque, il les font uniquement pour les régions sur la page qui contiennent soit du texte, soit une image. Dit autrement, tout type de région, y compris l'espace entre mots, s'encadre dans un rectangle, mais les types qui contiennent quelque chose autre qu'une espace vide, donc toute région sauf l'espace entre mots, s'encadrent dans un polygone. Le polygone porte plus de coordonnées que le rectangle. Dans le schéma ALTO, les valeurs des coordonnées du polygone sont données dans l'attribut `@POINTS` de l'élément `<Polygon>` qui descend indirectement de l'élément sur lequel il porte. Notre

modélisation en TEI représent les coordonnées du polygone dans l'attribut `@points` du même élément `<zone>` qui est concerné.

Le bloc (<TextBlock>) en particulier

En plus des pratiques généralisés pour toute zone, la modélisation TEI du `<TextBlock>` exige la composition d'une URL pour visionner le masque du bloc (`@source`) et la décomposition de l'étiquette appliquée au bloc. Entraîné sur le vocabulaire *SegmOnto*, le modèle HTR devrait donner au bloc une référence à une étiquette qui peut se composer de trois parties : le type, le sous-type, et le numéro dans la suite. Sur la page d'un manuscrits, par exemple, la deuxième colonne porterait l'étiquette `MainZone:column:2`. Les étiquettes ainsi composées sont données aux blocs et aux lignes de texte. Les parties du document encore plus petites, tel que le mot ou le glyphe, ne portent pas d'étiquette composée.

Par conséquent, uniquement les étiquettes attribuées aux éléments du `<TextBlock>` et `<TextLine>` sont décomposées lors de leur transformation en TEI puisqu'elles peuvent se diviser en trois. Dans notre modélisation, l'attribut `@type` prend la première partie de l'étiquette, le `@subtype` prend le sous-type qui pourrait suivre les deux points, et le `@n` prend le numéro s'il y en a un qui suit les deux points à la fin. Si le modèle HTR n'a pas mis en pratique des étiquettes aussi détaillées, les attributs `@subtype` et `@n` prennent la valeur *none* pour le bloc. Mais pour la ligne de texte (`<TextLine>`), la valeur du numéro se constitue du compte que fait l'application `alto2tei` des lignes de texte traitées sur la page. Un tel compte n'est pas si logique pour les blocs et donc l'attribut `@n` ne porte pas de valeur significative si l'étiquette n'en a pas donnée aucune. L'exemple du `<TextBlock>` et l'exemple de sa modélisation en TEI sont donnés dans la Figure 8.4. Une visualisation de la transformation d'ALTO à TEI est donnée à la fin du chapitre dans la Figure 8.9.

8.1.3 La ligne de texte

Dans le vocabulaire *SegmOnto*, l'étiquette *line* s'applique à la région de l'image dans laquelle s'encadre une ligne de texte. L'élément ALTO qui prend cette donnée est l'élément `<TextLine>`. Contrairement au `<TextBlock>` qui ne contient pas forcement du texte, l'élément `<TextLine>` doit avoir des prédictions du texte encodées dedans et doit donc avoir d'enfants qui descendent de lui. Comme l'étiquette *SegmOnto* du `<TextBlock>`, celle du `<TextLine>` se divise en trois parties. Si la valeur du `@type`, la première partie de l'étiquette, est identique à l'une des étiquettes listée dans le `<taxonomy>` du `<teiHeader>`, l'élément portera aussi l'attribut `@corresp` qui prendra comme valeur une référence à la classe.

La modélisation des données du `<TextLine>` en TEI s'appuient comme d'habitude sur l'élément `<zone>` parce qu'il porte sur la représentation d'une région de la page et une partie des données du `<TextLine>` portent sur le masque de la ligne. Les attributs

```
<TextBlock HPOS="323" VPOS="336" WIDTH="2056" HEIGHT="2812" ID="textblock_0
    " TAGREFS="BT2062">
  <Shape>
    <Polygon POINTS="2379 336 2379 3148 323 3148 323 336"/>
  </SHAPE>
<!-- ... -->
</TextBlock>
```

(a) Le <TextBlock> en ALTO

```
<zone xml:id="f11-textblock_0-blockCount1" type="MainZone" corresp="#
    MainZone" subtype="none" n="none" ulx="323" uly="336" lrx="2379" lry="
    3148" points="379,336 2379,3148 323,3148 323,336" source="https://
    gallica.bnf.fr/iiif/ark:/12148/btv1b8610802d/f11/323,336,2056,2812/full
    /0/native.jpg">
<!-- ... -->
</zone>
```

(b) Le <zone> du <TextBlock> en TEI

FIGURE 8.4 – La modélisation du <TextBlock>

du <zone> pour la ligne de texte sont identiques à ceux du bloc de texte. Il y a les quatre coordonnées du rectangle @ulx, @uly, @lrx, @lry récupérées respectivement depuis les attributs suivants du <TextLine> : @HPOS, @VPOS, @WIDTH, @HEIGHT. Ensuite l'attribut @source se compose en part de ces quatre coordonnées. Enfin, le <zone> contient les points du <Polygon>, l'élément qui descend du <TextLine> et qui décrit le périmètre du polygone qui encadre la ligne de texte.

Le <zone> du <TextLine> contient deux enfants directs particulier à la ligne de texte : le <line> et le <path>. L'élément <line> contient le texte de la ligne. Comme attribut, il porte simplement un identifiant et le nombre de la ligne de texte lors du traitement du fichiers XML ALTO. Le <path> représente le *baseline* de la ligne de texte, c'est-à-dire le début et la fin de la ligne linéaire. Il se compose donc de quatre nombres, un pair x,y indiquant le point du début et un deuxième pair x,y indiquant le point de la fin. Les quatre nombres sont encodés directement dans le fichier XML ALTO comme la valeur de l'attribut @BASELINE du <TextLine>. L'élément TEI qui convient le mieux à la donnée du *baseline* est l'élément <path>. L'exemple du <TextLine> et l'exemple de sa modélisation en TEI sont donnés dans la Figure 8.5. Une visualisation de la transformation d'ALTO à TEI est donnée dans la Figure 8.10.

La ligne de texte quand il y a des prédictions sur les mots et les glyphes dedans

Pour certains fichiers XML ALTO, tel que ceux qui sortent de l'interface *eScriptorium*, la modélisation en TEI s'arrête là. Le fichier XML ALTO ne porte pas de plus de détail après la ligne de texte. Mais pour certains d'autres fichiers, tel que ceux qui sont

```
<TextLine ID="textline_0" TAGREFS="LT722" BASELINE="605 944 2010 925" HPOS=
    "596" VPOS="777" WIDTH="1414" HEIGHT="182">
  <Shape>
    <Polygon POINTS="605 944 596 816 666 795 669 795 672 795 814 810 838
    792 838 789 841 789 844 789 847 789 932 804 953 789 956 789 959 789 962
    789 1050 801 1323 783 1326 783 1704 798 1768 777 1771 777 1774 777 2004
    798 2010 925 2004 953 1798 941 1750 956 1747 956 1744 956 605 959"/>
  </Shape>
  <String CONTENT="A MONSIEVR" HPOS="596" VPOS="777" WIDTH="1414" HEIGHT="
    182"/>
</TextLine>
```

(a) Le <TextLine> en ALTO où le texte est contenu dans l'attribut @CONTENT de l'élément descendant <String>

```
<zone xml:id="f11-textblock_0-textline_0-lineCount1" type="HeadingLine"
    corresp="#HeadingLine" subtype="none" n="none" ulx="596" uly="777" lrx="
    2010" lry="959" points="605,944 596,816 666,795 669,795 672,795 814,810
    838,792 838,789 841,789 844,789 847,789 932,804 953,789 956,789 959,789
    962,789 1050,801 1323,783 1326,783 1704,798 1768,777 1771,777 1774,777
    2004,798 2010,925 2004,953 1798,941 1750,956 1747,956 1744,956 605,959"
    source="https://gallica.bnf.fr/iiif/ark:/12148/btv1b8610802d/f11
    /596,777,1414,182/full/0/native.jpg">
  <path xml:id="f11-textblock_0-textline_0-lineCount1-baseline" points="
    605,944 2010,925"/>
  <line xml:id="f11-textblock_0-textline_0-lineCount1-text" n="1">A
    MONSIEVR</line>
</zone>
```

(b) Le <zone> du <TextLine> en TEI

FIGURE 8.5 – La modélisation du <TextLine>

produits par l'engin *Kraken* depuis la ligne de commande, ils attestent aux prédictions sur des mots et sur des glyphes. Dans ce cas, la ligne de texte a plus de descendants, mais afin de garder une arborescence générique qui facilite des comparaisons entre des fichiers de divers formats, le <zone> de la ligne de texte garde toujours les mêmes deux enfants : le <path> et le <line>. En plus de ces deux, le <zone> contient une suite d'éléments <zone> pour tout segment prédit sur la ligne, soit un mot, soit une espace entre mots.

8.1.4 Le segment

Dans notre modélisation, les données des fichiers XML ALTO qui contiennent plus de détail sont donnés encore des éléments <zone> pour des segments (<String>) et des glyphes (<Glyph>). Les segments qui contiennent des glyphes peuvent représenter la prédiction d'un mot, la ponctuation, bien qu'un mot avec de la ponctuation à côté. L'important est que le segment contient soit la prédiction d'au moins un glyphe ou la prédiction d'une espace entre mots. Uniquement les segments (<String>) qui contiennent des pré-

dictions sur des glyphes portent aussi un polygone dans lequel s'encadre la chaîne des glyphes. Si le <String> représente une espace entre mots, les modèles HTR ne prédisent qu'un rectangle. Les modèles HTR évaluent leur taux de réussite à partir de l'ensemble de glyphes bien prédits dans un mot. L'évaluation de sa prédiction est représentée dans l'attribut @WC de l'élément <String>. L'acronyme *WC* signifie en anglais *word confidence*. L'exemple du <String> et l'exemple de sa modélisation en TEI sont donnés dans la Figure 8.6. Une visualisation de la transformation d'ALTO à TEI est donnée dans la Figure 8.11.

```
<String ID="segment_1" CONTENT="MONSIEVR" HPOS="837" VPOS="777" WIDTH="1172
    " HEIGHT="182" WC="0.9.64">
  <Shape>
    <Polygon POINT="..."/>
  </Shape>
<!-- ... -->
</String>
```

(a) Le <String> en ALTO où le texte n'est pas contenu dans l'attribut @CONTENT de l'élément descendant <String>

```
<zone xml:id="f11-textblock_0-textline_0-segment_2-segCount3" type="String"
    ulx="837" uly="777" lrx="2009" lry="959" points="..." source="https://
    gallica.bnf.fr/iiif/ark:/12148/btv1b8610802d/f11/837,777,1172,182/full
    /0/native.jpg">
  <certainty xml:id="f11-textblock_0-textline_0-segment_2-segCount3-cert"
    target="#f11-textblock_0-textline_0-segment_2-segCount3-text" locus="
    value" degree="0.9064"/>
<!-- ... -->
</zone>
```

(b) Le <zone> du <String> en TEI

FIGURE 8.6 – La modélisation du <String>

8.1.5 Le glyphe

Les caractères et de la ponctuation prédits par des modèles HTR sont tous encodés dans l'élément <Glyph> selon le schéma ALTO. Contrairement au <String> qui sert à contenir une chaîne de glyphes, le <Glyph> du fichiers XML ALTO contient à la fois le masque et le texte. Pour cette raison, la modélisation en TEI représente, comme d'habitude, le masque du <Glyph> dans l'élément <zone> et le texte prédit dans l'élément <c>. Ce dernier est un élément du schéma TEI destiné à la représentation d'un caractère, soit une lettre, soit de la ponctuation. Il convient bien donc à la représentation de toute prédiction dans l'élément <Glyph> du fichier XML ALTO. Le modèle HTR évalue son taux de réussite de sa prédiction du glyphe. L'évaluation de sa prédiction est représentée

dans l'attribut `@GC` de l'élément `<String>`. L'acronyme *GC* signifie en anglais *glyph confidence*. L'exemple du `<Glyph>` et l'exemple de sa modélisation en TEI sont donnés dans la Figure 8.7. Une visualisation de la transformation d'ALTO à TEI est donnée dans la Figure 8.12.

```
<Glyph ID="char_1" CONTENT="M" HPOS="837" VPOS="777" WIDTH="159" HEIGHT="
    162" WC="0.8127">
  <Shape>
    <Polygon POINT="..."/>
  </Shape>
<!-- ... -->
</String>
```

(a) Le `<Glyph>` en ALTO

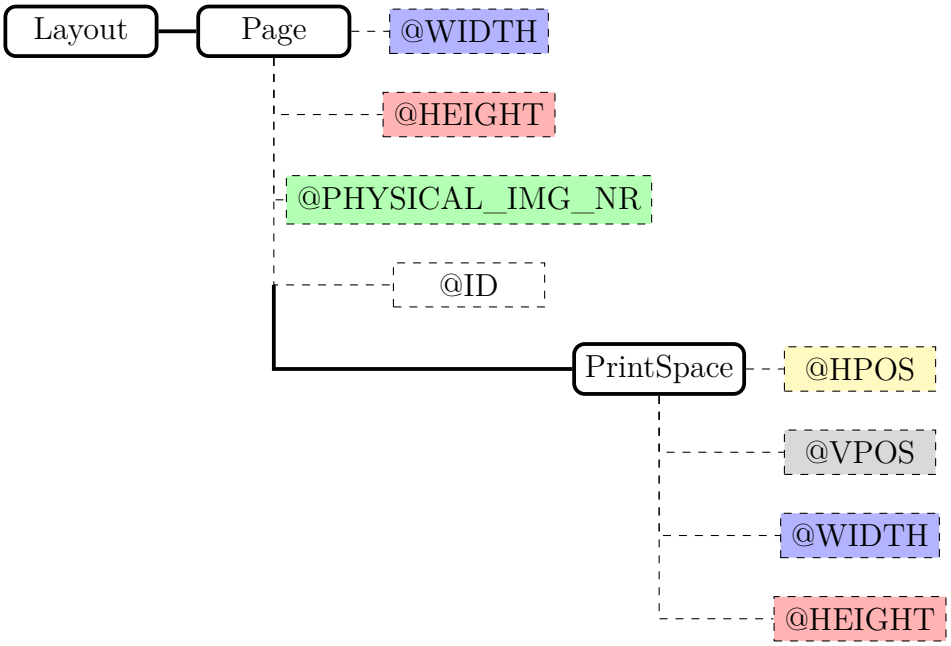
```
<zone xml:id="f11-textblock_0-textline_0-segment_2-char_1-glyphCount2" type
    ="String" ulx="837" uly="777" lrx="996" lry="939" points="..." source="
    https://gallica.bnf.fr/iiif/ark:/12148/btv1b8610802d/f11
    /837,777,154,120/full/0/native.jpg">
  <certainty xml:id="f11-textblock_0-textline_0-segment_2-char_1-
    glyphCount2-cert" target="#f11-textblock_0-textline_0-segment_2-char_1-
    glyphCount2-text" locus="value" degree="0.8127"/>
  <c xml:id="f11-textblock_0-textline_0-segment_2-char_1-glyphCount2-text">
    M</c>
</zone>
```

(b) Le `<zone>` du `<Glyph>` en TEI

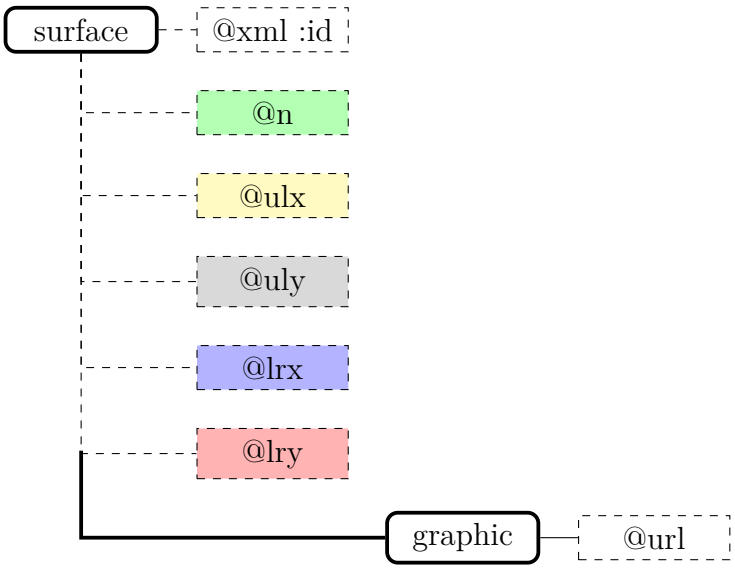
FIGURE 8.7 – La modélisation du `<Glphy>`

8.2 Les visualisations de la transformation

Des visualisations de la modélisation de chaque élément du fichier XML ALTO sont montrées dans les figures qui suivent. Les attributs sont visualisés par les carrés en ligne tirée et les éléments XML sont visualisés par les carrés en ligne solide. La couleur de l'élément ou de l'attribut du schéma ALTO est répétée dans la visualisation du schéma TEI quand sa valeur est utilisée.

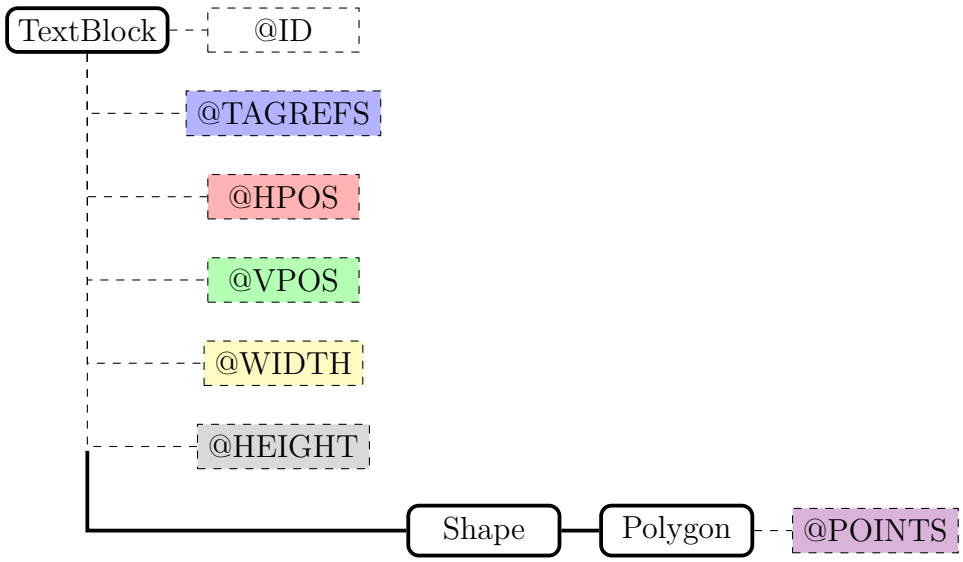


(a) Le modèle du <Page> en ALTO

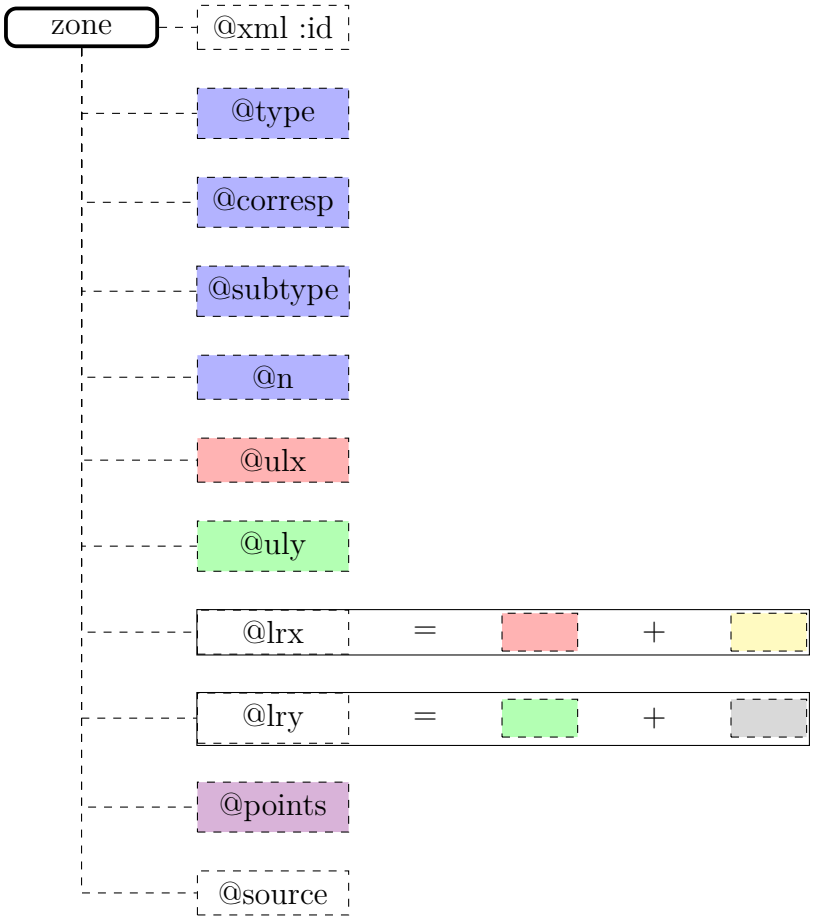


(b) La modélisation du <Page> en TEI

FIGURE 8.8 – La transformation du <Page> d'ALTO à TEI

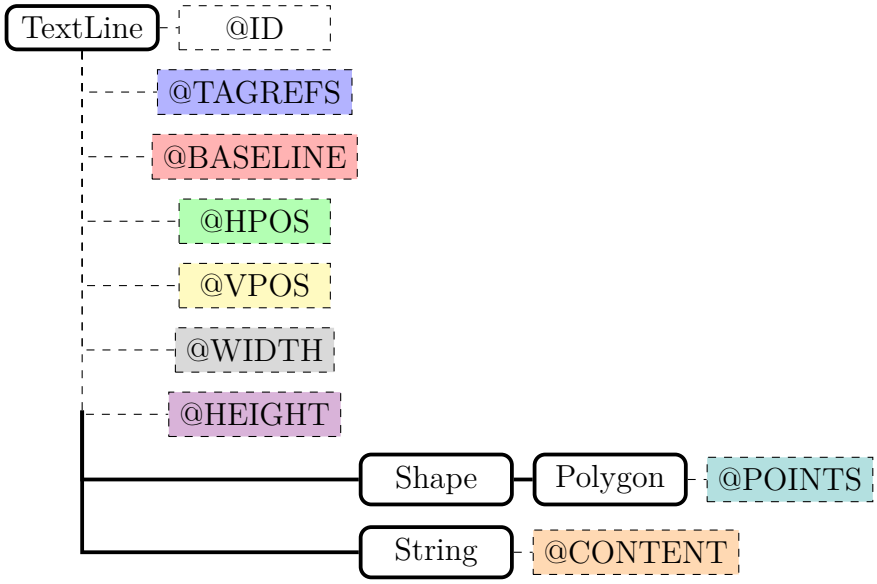


(a) Le `<TextBlock>` en ALTO

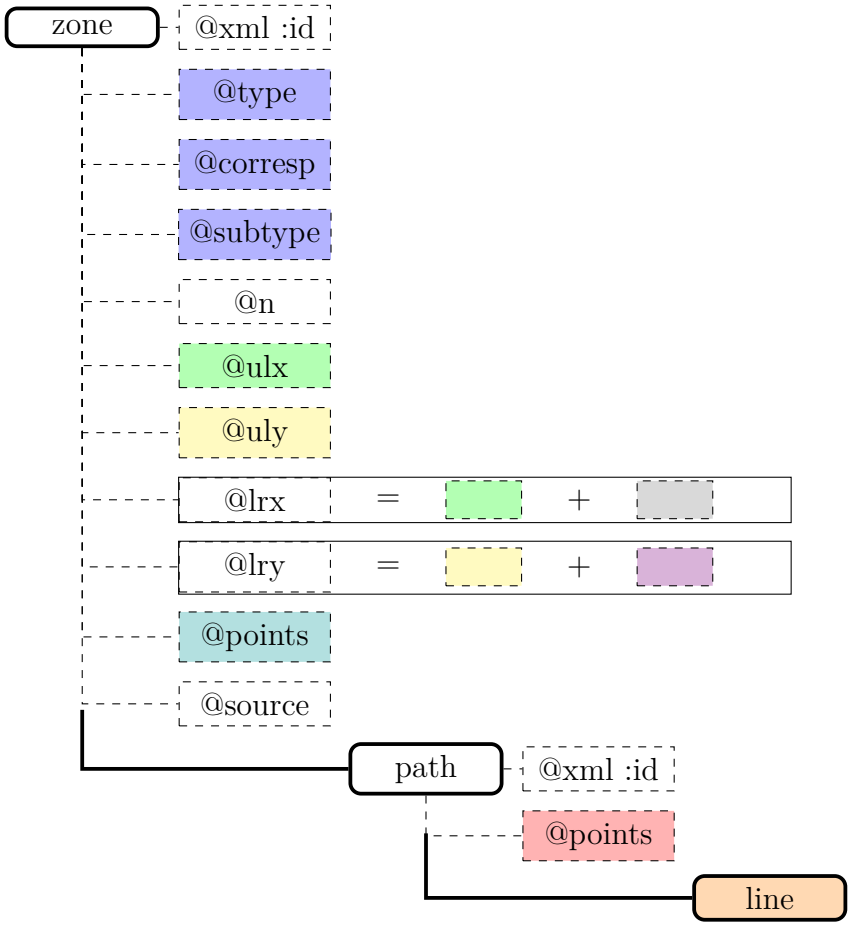


(b) La modélisation du `<TextBlock>` en TEI

FIGURE 8.9 – La transformation du `<TextBlock>` en TEI

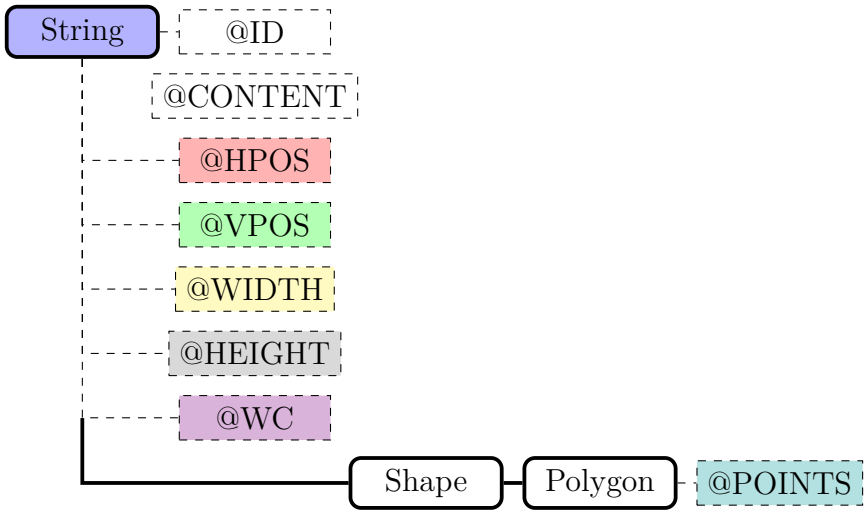


(a) Le <TextLine> en ALTO où tout le contenu textuel prédit de la ligne est présenté dans l'attribut @CONTENT

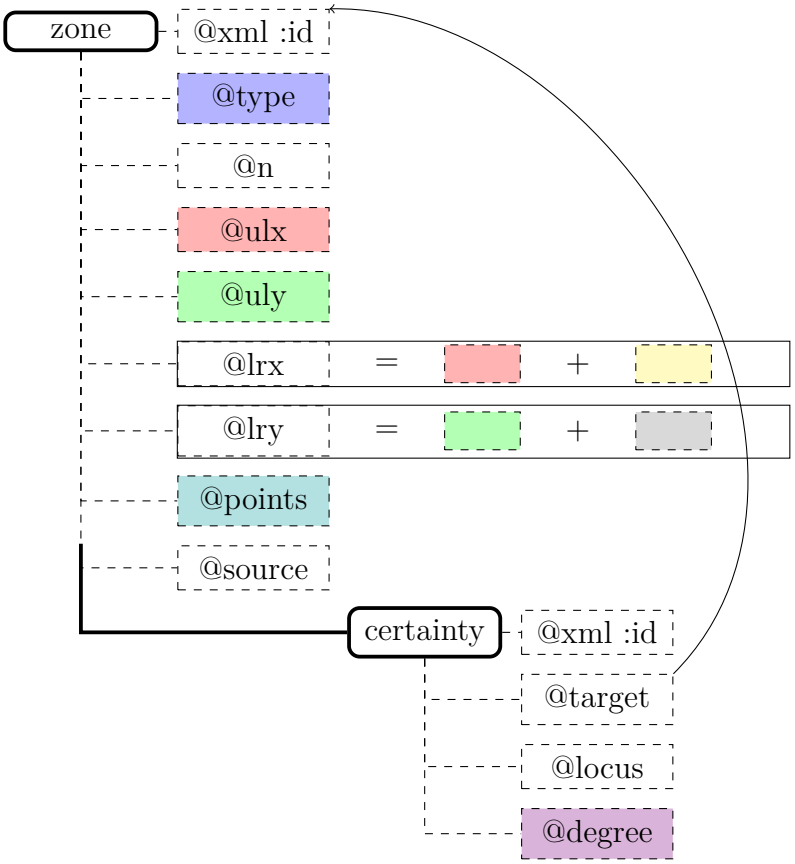


(b) La modélisation du <TextLine> en TEI

FIGURE 8.10 – La transformation du <TextLine> en TEI

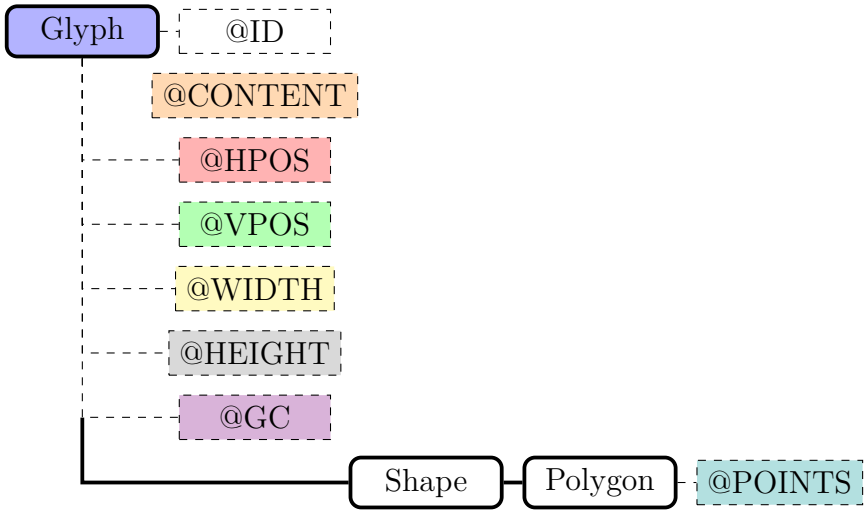


(a) Le `<String>` en ALTO

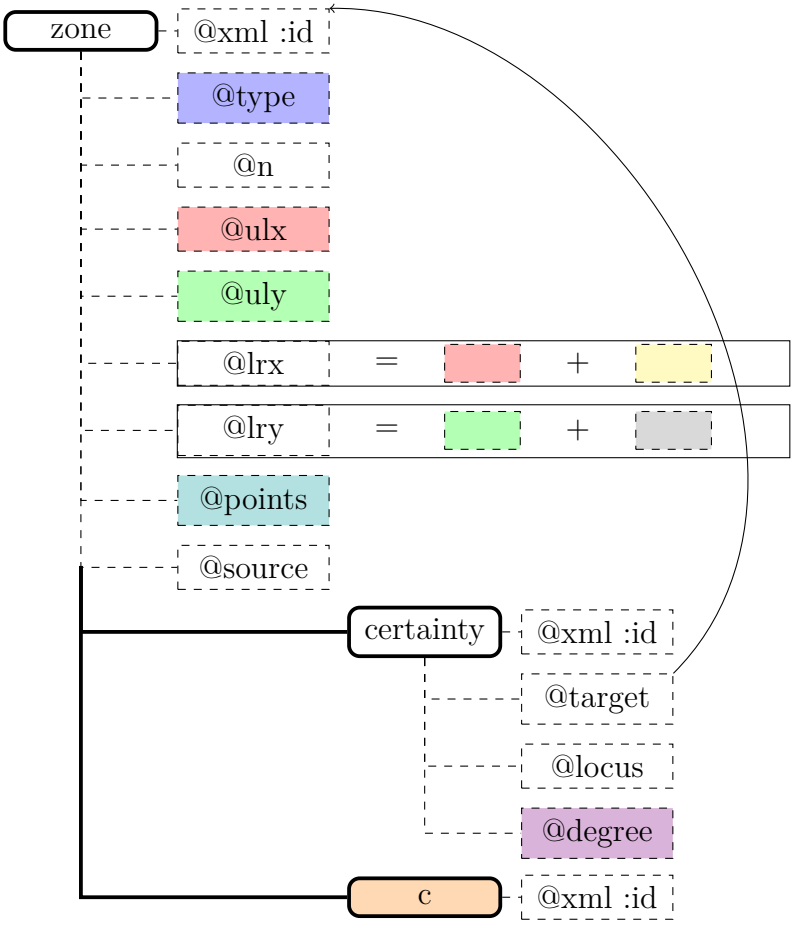


(b) La modélisation du `<String>` en TEI

FIGURE 8.11 – La transformation du `<String>` en TEI



(a) Le <Glyph> en ALTO



(b) La modélisation du <Glyph> en TEI

FIGURE 8.12 – La transformation du <Glyph> en TEI

Chapitre 9

Le traitement des données produites

Jusqu'à présent dans l'exposition sur la modélisation des fichiers ALTO en TEI, nous avons parlé de deux éléments descendant directement de la racine TEI. Le `<teiHeader>` renseigne sur les trois objets de texte concernés par la ressource numérique : la ressource elle-même, le fac-similé numérique dont les images des modèles HTR ont traité, et le document source physique à partir duquel le fac-similé a été fait. Le document TEI contient aussi des métadonnées utiles à l'exploitation de la ressource numérique. Dans un deuxième temps, le `<sourceDoc>` a récupéré toute donnée significative des fichiers ALTO et les met dans un arbre TEI, spécifiquement dans les éléments descendant du `<sourceDoc>` et des attributs. Ayant récupéré toutes les données des sources externes, l'application `alto2tei` traite ensuite les données déjà présentes dans la ressource en cours de construction.

La ressource numérique présente deux versions du texte prédit que l'application `alto2tei` produit à partir des données qu'elle a traités et mises dans le `<sourceDoc>`. Dans un premier temps, elle présente dans l'élément `<body>` une version du texte pré-éditorialisée, c'est-à-dire dans la manière par laquelle le texte s'apparaît sur la page. Les fautes d'orthographe, les sauts de ligne, les coupures de mot sont tous conservés dans la représentation du texte de l'élément `<body>`. La version pré-éditorialisée sert à l'analyse du texte transcrit ainsi qu'à l'analyse linguistique que peuvent vouloir faire des chercheurs.

Dans un deuxième temps, la ressource numérique du pipeline *Gallic(orpor)a* présente son propre analyse linguistique du texte transcrit grâce aux modèles TAL qu'elle met en pratique. La représentation du texte ainsi analysé, avec des entités nommés et des mots normalisés, est contenu dans l'élément TEI `<standOff>` pour qu'elle ne soit pas traité comme la transcription du texte. La transcription du texte est représentée dans l'élément `<body>` qui peut être facilement exploité par les éditeurs et les visionneurs de texte en format TEI. L'état actuel de l'application `alto2tei` n'effectue pas d'analyse linguistique, mais elle prépare la représentation du texte pré-éditorialisé du `<body>` duquel un *feature* ajouté plus tard pourrait disposer pour mettre en œuvre l'analyse linguistique produite par des modèles TAL.

9.1 La génération du `<body>` grâce au vocabulaire *SegmOnto*

L'objectif du `<body>` est de permettre aux logiciels d'exploiter la transcription du texte que des modèles HTR ont prédit. Cependant, des modèles prédisent plusieurs aspects de la page numérisée, plus que celui qui concerne le texte du document. Selon le vocabulaire *SegmOnto*, par exemple, les parties de l'image dans lesquelles s'encadre un dessin (*GraphicZone*), ou un numéro de page (*NumberingZone*), ou un titre en tête (*RunningTitleZone*) ne devraient pas être incluses dans la représentation du texte pré-éditorialisé. La plupart des chercheurs qui désireront analyser la transcription du fac-similé auront besoin du texte appartenant à l'œuvre telle qu'elle se conçoit. Les en-têtes, les numéros de pages, le texte des tampons, et les notes ajoutées après la publication de l'imprimé ou l'apparition du manuscrit sont tous intéressants à la recherche, mais ils dérangent l'analyse computationnelle de l'œuvre qui compte sur une représentation du texte continu, où les mots coupés au saut de ligne sont recollés et le texte qui continue sur la page suivante fait partie d'un encodage sans une telle interruption.

L'application `alto2tei` met en œuvre deux étapes pour extraire et nettoyer le texte transcrit. Dans un premier temps, elle recherche tous les éléments `<line>` du `<sourceDoc>` afin de recueillir les lignes de texte prédit dans le document source. Ces lignes deviennent directement de la prédiction des modèles HTR. Il faut donc les nettoyer. L'application garde toute saut de ligne à la fin de chaque chaîne qu'elle extrait du `<sourceDoc>` sauf si elle coupe un mot, ce que l'application sait si la ligne se termine par un tiret. Dans un tel cas, elle recolle le début du mot à la fin de ligne et le reste du mot au début de la ligne suivante.

Dans un deuxième temps, l'application `alto2tei` analyse les étiquettes de toutes ses lignes de texte ainsi nettoyées. L'équipe *Gallic(orpor)a* a déterminé une traduction entre les étiquettes du vocabulaire *SegmOnto* et l'arborescence de la TEI. La Table 9.1 montre quelles lignes font partie du texte principal et comment notre modélisation les balise dans des éléments du `<body>`. Toute ligne se précède par un `<lb/>`, l'élément du schème TEI destiné à indiquer le début d'une ligne de texte. Balisant le `<lb/>` est (au moins) l'un des éléments TEI de la Table 9.1.

SegmOntoZone	*SegmOntoLine*	représentation en TEI
NumberingZone	DefaultLine	`<fw type="NumberingZone">`
QuireMarksZone	DefaultLine	`<fw type="QuireMarksZone">`
RunningTitleZone	DefaultLine	`<fw type="RunningTitleZone">`
MarginTextZone	DefaultLine	`<note type="MarginTextZone">`
MainZone	DefaultLine	`<ab type="MainZone">`
MainZone	DropCapitalLine	`<hi rend="DefaultLine">`

FIGURE 9.1 – Les étiquettes du texte principal

Les lignes d'affilé dans une *MainZone* s'encadrent toutes dans l'élément `<ab>`, mais la ligne d'un *drop capital* se balise dans l'élément `<hi>` qui lui-même se balise dans le `<ab>` de la zone à laquelle la ligne appartient, comme se voit dans la Figure 9.2.

Ligne de texte
Deuxième ligne de texte

```
<ab corresp="#sourceDocBlockID" type="MainZone">
  <lb corresp="#sourceDocLine1ID"/><hi rend="DropCapitalLine">L</hi>igne de
    texte.
  <lb corresp="#sourceDocLine2ID"/>Deuxième ligne de texte.
</ab>
```

FIGURE 9.2 – Les lignes de la *MainZone*

Ainsi encodée, où les lignes de texte se suivent l'une après l'autre dans l'élément `<body>`, en ignorant les zones détectées qui ne portent pas de texte, la ressource numérique peut être exploitée par un logiciel configuré pour le TEI. Normalement l'élément `<lb/>` produira une saut de ligne pour que le texte qui traîne derrière l'élément vide s'apparaisse au début d'une nouvelle ligne. En plus, en recollant les mots coupés à la fin de ligne dans le document source, les mots ainsi nettoyés et reconstitués peuvent être cherchés grâce aux outils de recherche du logiciel. Par exemple, une utilisatrice ou un utilisateur ne trouverait pas le mot *bâtiment* s'il avait été écrit dans le document source comme *bâti-ment*, avec une saut de ligne après le tiret, et qu'il n'était pas recollé avant la recherche de l'utilisatrice ou l'utilisateur.

Enfin, le texte hiérarchisé du `<body>` peut supporter des annotations et de l'analyse encore plus profonde et particularisée au genre du document source. Par exemple, une équipe de chercheurs pourraient développer un système de filtrage, tel qu'une feuille de transformation XSL, pour traiter les répliques d'un drame que notre modélisation balise dans le `<ab>` d'une *MainZone*. Un tel exemple d'élaboration de l'encodage du `<body>` est visualisé dans la Figure 9.3. Le texte pré-éditorialisé du `<body>` permet d'élaborer un texte éditorialisé grâce à la base produite par l'application `alto2tei` et conceptualisée par l'équipe *Gallic(orpor)a*.

9.2 L'analyse linguistique

Le dernière aspect de la ressource numérique qu'avait conceptualisé l'équipe du projet *Gallic(orpor)a* est une analyse linguistique du texte prédit. Mon application `alto2tei` ne réalise pas encore l'application de cet objectif. Cependant, sa structure supporte l'ajout plus tard d'un *feature* qui soit reprendra le texte nettoyé du `<body>`, soit reprendra le texte que mon application a structuré dans les éléments du `<sourceDoc>`. Les deux options sont

```
<ab corresp="#sourceDocBlockID" type="MainZone">
  <lb corresp="#sourceDocLine1ID"/>Angélique.
  <lb corresp="#sourceDocLine2ID"/>Regarde-moi un peu.
  <lb corresp="#sourceDocLine3ID"/>Toinette.
  <lb corresp="#sourceDocLine4ID"/>Hé bien! je vous regarde.
</ab>
```

(a) Le texte pré-éditorialisé du `<body>`

```
<div2 type="scene" n="4">
  <sp>
    <speaker corresp="#sourceDocLine1ID"/>Angélique.</speaker>
    <p>
      <lb corresp="#sourceDocLine2ID"/>Regarde-moi un peu.
    </p>
  </sp>
  <sp>
    <speaker corresp="#sourceDocLine3ID"/>Toinette.</speaker>
    <p>
      <lb corresp="#sourceDocLine4ID"/>Hé bien! je vous regarde.
    </p>
  </sp>
</div2>
```

(b) Le texte éditorialisé

FIGURE 9.3 – La transformation des répliques prédits dans une *MainZone*

intéressants parce qu'elles permettent des approches différentes à la mise en œuvre des modèles TAL (Traitement automatique des langues). L'un des enjeux de l'analyse linguistique et l'alignement entre le texte saisie et le texte produit. Idéalement, un mot normalisé par des modèles TAL se lierait toujours à sa version transcrite depuis le document source. Ainsi, la donnée que des modèles TAL produite ferrait référence à l'identifiant du élément `<zone>` dans le `<sourceDoc>`.

Le défi à surmonter est que les modèles TAL ont besoin des mots complets, c'est-à-dire recollés s'ils était coupés à la fin de ligne. La transcription représentée dans le `<sourceDoc>` donne à chaque segment de texte son propre identifiant ; par conséquent, un mot peut porter deux identifiants s'il est coupé en deux. Il est possible de lier les deux identifiants du mot dans la transcription avec le seul identifiant du même mot nettoyé et donné aux modèles TAL. Mais la complexité qui se produit pour certains mots et pas pour des autres est un défi qu'une application doit pouvoir surmonter.

Un autre défi qui s'applique tous le temps concerne les lignes de texte de la transcription. Normalement les modèles TAL saisissent les segments du texte qui ont une cohérence lexicale, afin qu'ils puissent en chercher la signification linguistique de la phrase. Il faut donc leur donner des lignes de texte recollées. Comme l'application `alto2tei` cherche les tirets à la fin de ligne, afin de recoller les mots coupés, une application qui met en œuvre

l'analyse linguistique devrait chercher les signes de ponctuation qui indique le début ou la fin d'une phrase cohérente. Pour le français moderne, qui a déjà adopté des règles quant à la ponctuation et à l'emploi de lettres majuscules, un tel filtrage du texte est facile et déjà mis en pratique par plusieurs applications. Mais pour les documents historiques du moyen français le défi devient plus compliqué. En outre, il se complique encore quand on veut élaborer un système pour recoller les lignes de texte performant pour plusieurs états du français.

Dans le cadre du stage, j'ai expérimenté avec la mise en œuvre de l'analyse linguistique en saisissant le texte que mon application a préparé pour le `<body>` de la ressource numérique. Puisque les mots du `<body>` sont déjà recollés, la première étape est de composer des phrases qui ont d'une cohérence lexicale sur laquelle des modèles TAL peuvent s'appuyer. J'avais déjà écrit un code pour faire cela dans le cadre de la création des vérités de terrain. Il y avait une équipe qui s'occupait des vérités de terrain pour les modèles HTR et une autre qui s'occupait de la lemmatisation du texte transcrit par la première équipe. Ce dernier jeu des vérités de terrain est ciblé à entraîner des modèles TAL pour l'application de l'analyse linguistique au corpus. Mon code a aidé l'équipe qui s'en chargeait car il traite les lignes de texte et sort un fichier TXT. Le texte se donne soit dans des phrases complets, soit dans une phrase partielle qui ne termine pas par des signes de ponctuation en fin de phrase, tel comme le point, le point d'exclamation, et le point d'interrogation.

Normalement, des modèles TAL saisissent des listes itératives des phrases et des mots [1]. Ayant préparé la liste itérative des phrases d'une cohérence lexicale, grâce à mon script, l'expérience a profité des scripts développés dans le cadre du projet *e-Ditiones* par Alexandre Bartz [2]. L'une des premières étapes de l'analyse linguistique, et quelque chose que fait le script de Bartz, est la tokenisation, par laquelle la phrase est désagrégée en *tokens*, c'est-à-dire des petits unités lexicale. Le texte ainsi atomisé peut être ensuite traité par des modèles TAL entraînés pour faire certaines tâches, telle comme la lemmatisation et la normalisation.

Le script de Bartz apport la lemmatisation au texte grâce à la librairie pytho *pie-extended*, développée par Thibault Clérice [3]. Un modèle entraîné pour faire la lemmatisation trait tout *token*, que le script de Bartz a préparé, et renvoie deux genres de données : la forme canonique du *token* (le *lemme*) et une suite d'analyses en fonction de la nature du

1. Pedro Javier Ortiz Suárez, Benoît Sagot et Laurent Romary. « Asynchronous Pipeline for Processing Huge Corpora on Medium to Low Resource Infrastructures ». In : *7th Workshop on the Challenges in the Management of Large Corpora (CMLC-7)*. Sous la dir. de Piotr Bański et al. Cardiff, United Kingdom : Leibniz-Institut für Deutsche Sprache, juill. 2019. doi : 10.14618/IDS-PUB-9021. url : https://hal.inria.fr/hal-02148693 (visité le 07/09/2022).

2. Alexandre Bartz, Juliette Janes et Simon Gabay. *Annotator*. E-ditiones. url : https://github.com/e-ditiones/Annotator (visité le 07/09/2022).

3. Thibault Clérice. *Pie Extended, an Extension for Pie with Pre-Processing and Post-Processing*. Zenodo, juin 2020. doi : 10.5281/zenodo.6534764. url : https://zenodo.org/record/6534764 (visité le 07/09/2022).

Humble salut et recognoissance de sa liberalité enuers luy.

(a) Le texte de saisie en phrase cohérente

token	*lemma*	*pos*
Humble	Humble	ADJqua
salut	salut	NOMcom
et	et	CONcoo
recognoissance	recognoissance	ADJqua
de	de	PRE
sa	son	DETpos
liberalité	liberalité	NOMcom
enuers	enuer	VERcjg
luy	luy	PROind
.	.	PONfrt

(b) La tokenisation et lemmatisation en format CSV

token	*lemma*	normalisation
Humble	Humble	Humble
salut	salut	salut
et	et	et
recognoissance	recognoissance	reconnaissance
de	de	de
sa	son	sa
liberalité	liberalité	liberalité
enuers	enuer	envers
luy	luy	lui
.	.	.

(c) La normalisation en format CSV

FIGURE 9.4 – L'expérience du Traitement automatique des langues sur le document d'ARK `bpt6k724151`

lemme[4]. Après la lemmatisation, le script de Bartz applique un modèle de normalisation qui transforme le français du document source en une version moderne. Dans l'expérience que j'ai faite avec un imprimé du XVIe siècle (ARK bpt6k724151), j'ai utilisé un modèle de normalisation entraîné par Rachel Bawden[5]. Un exemple des résultats des deux tâches de TAL est montré dans la Figure 9.4.

Le dernier défi pour une application éventuelle de l'analyse linguistique dans le pipeline *Gallic(orpor)a* est la modélisation des données produites. La sortie des modèles TAL est typiquement du format CSV (Comma Separated Values) ou TSV (Tab Separated Values), mais la ressource numérique exige des données en XML et formatées selon les

4. Dans l'analyse du script de Bartz, la nature (*part of speech* ou POS) du *lemme* est accompagnée par des analyses supplémentaires et dépendents sur la langue. Par exemple, il donne au *lemme* identifié comme un verbe son temps, et celui identifié comme un nom son genre.

5. Rachel BAWDEN. *ModFr-Normalisation*. URL : `https://github.com/rbawden/ModFr-Norm` (visité le 07/09/2022).

normes de la TEI. Une telle modélisation a été présentée par Bartz, Juliette Janes, Laurent Romary, Philippe Gambette, Rachel Bawden, Pedro Ortiz Suaréz, Benoît Sagot, et Simon Gabay en 2021 à la conférence TEI[6]. L'équipe *Gallic(orpor)a* envisage à élaborer une modélisation du `<standOff>` similaire. Ainsi disposé du script de l'application `alto2tei` qui prépare une version pré-éditorialisée du texte, le pipeline *Gallic(orpor)a* est prêt à avoir intégrée de l'analyse linguistique et pouvoir construire le `<standOff>`.

6. Alexandre Bartz et al. « Expanding the Content Model of annotationBlock ». In : *Next Gen TEI, 2021 - TEI Conference and Members' Meeting*. Virtual, United States, oct. 2021. URL : `https://hal.archives-ouvertes.fr/hal-03380805` (visité le 07/09/2022).

Conclusion

Ce qu'il faut retenir de cette expérience de la transformation d'ALTO à TEI est que la génération du `<teiHeader>` et celle du `<sourceDoc>` sont des étapes très différentes. La transformation des données du schéma ALTO vers la TEI est un défi qui intéresse de plus en plus d'équipes et les solutions sont assez généralisables. Une solution proposée pour les transcriptions en ALTO d'un corpus des documents d'une bibliothèque à Vienne peut bien marcher pour un corpus des documents d'une bibliothèque à Rennes. Mais la génération du `<teiHeader>` compte beaucoup plus sur l'origine des documents transcrits. Un script qui sait rechercher des métadonnées depuis l'API SRU pour le catalogue général de la Bibliothèque nationale de France n'arriverait pas à récupérer des métadonnées pour un document provenant du fonds d'une bibliothèque à Vienne, dont la notice est encodée et repérable par de différents moyens. Pour cette raison, je conseille que les futurs projets qui reprennent la génération automatique d'un document TEI, y compris son `<teiHeader>`, bien divise ces deux étapes. Une équipe peut s'occuper de l'interrogation des métadonnées depuis des sources externes et une autre peut se charger de la transformation des données du schéma ALTO en TEI. Les deux tâches n'ont rien à voir l'un à l'autre, et peuvent se faire indépendamment.

En outre, puisque la génération automatique du `<teiHeader>` et celle du `<sourceDoc>` sont si distinctes, elles peuvent être conceptualisées et résolus par de divers scripts. Une application (ou bien un fichier XSLT) peut transformer les données du schéma ALTO, et une autre application peut spécialiser dans la récupération et gestion des métadonnées depuis une suite de sources de données. Mon application `alto2tei` est conçu pour gérer les métadonnées des sources du portail Gallica et donc elle envoye certaines requêtes aux certaines APIs. Mais une autre application spécialisée pour la génération automatique du `<teiHeader>` peut être conçue pour s'adapter à de divers bibliothèques et sources de fac-similés numériques. Sinon, une architecture plus générique que celle que nous avons construite pour le pipeline *Gallic(orpor)a* pourrait être conçue qui saisirait la syntaxe des requêtes ainsi que les *endpoints* de divers APIs. Disposant de telles données d'entrée, une telle application générique pourrait se spécialiser aux plusieurs structures de données et aux plusieurs schémas, tel que Dublin Core ainsi que UNIMARC.

Comme se voit dans l'appendice A.1.2, mon application `alto2tei` rassemble trois tâches distinctes. Dans un premier temps, elle génère le `<teiHeader>` qui s'appuient sur

les sources de données à la fois internes et externes. Dans un deuxième temps, elle génère le `<sourceDoc>` qui traite les fichiers XML ALTO produits par des modèles HTR. Enfin, l'application parse le texte représenté dans l'élément `<sourceDoc>` ainsi construit à partir des fichiers XML ALTO. Les modules sur lesquelles l'application compte pourrait facilement conçus pour de divers défis. Si une équipe tentait à réaliser une application plus générique qui produit un `<teiHeader>` à partir des données XML UNIMARC de peu importe laquelle bibliothèque, par exemple, peu importe quel pipeline qui cherche à traiter des documents dont la notice détaillée est encodée en UNIMARC pourrait s'en servir.

L'une des autres conclusions à retenir de l'expérience du projet *Gallic(orpor)a* est qu'un tel pipeline, l'un qui commence par des fac-similés numériques et termine par une ressource encodée en TEI, est bien possible. De plus, l'entraînement des modèles à reconnaître la mise en page par le syntaxe du projet *SegmOnto* rendre possible la génération automatique d'un `<body>` très détaillé et qui conforme bien aux normes de la TEI. Le défi qui reste à surmonter et l'entraînement des modèles de segmentation. Le concept a été prouvé qu'un tel syntaxe faciliterait la création d'une version du texte pré-éditorialisée. Mais la réalisation des modèles qui parviennent à bien taguer les zones et les lignes sur la page reste à être améliorer. En conclusion, l'expérience de la transformation d'ALTO à TEI selon notre modélisation a eu une bonne résultat, et un pipeline automatisé qui saisit des images et renvoie une ressource en TEI est bien faisable. L'amélioration de la segmentation et l'ajout de l'analyse linguistique sont des enjeux qui restent pour de futurs projets qui reprennent les ambitions du projet *Gallic(orpor)a*.

Annexe A

ALTO2TEI

Le livrable principal du stage était une application en python qui construit un document XML TEI pour le pipeline *Gallic(orpor)a*. Ce document préliminaire doit saisir la transcription produite par des modèles HTR, qui est en format XML ALTO, bien que chercher des métadonnées depuis divers sources de données externes et internes au pipeline. La librairie principale pour manipuler des données XML en python est `lxml`, dont j'ai disposé toute la durée du stage [1]. La librairie `lxml` aide à traiter les fichiers XML ALTO ainsi que la création et gestion de l'arbre XML TEI. Pour traiter les métadonnées, j'ai disposé de la libraire python requests pour envoyer des requêtes d'API [2]. Le code pour l'application `alto2tei` est disponible sur mon GitHub [3].

A.1 Setup et démarrage

L'application `alto2tei` se démarre depuis la ligne de commande. Le pipeline *Gallic(orpor)a* l'appel via son script en bash. L'exemple de la commande est la suivante :

```
alto2tei --config config.yml --version "3.0.13" --header --sourcedoc --body
```

Le premier argument optionnel (`--config`) localise le fichier de configuration. La deuxième (`--version`) dit à l'application quelle version de l'engin HTR *Kraken* a été utilisée pour créer les fichiers XML ALTO. Les trois arguments en dernier permettent de créer certaines parties du document TEI et pas d'autres, au cas où une utilisatrice ou un utilisateur veut simplement créer le `<teiHeader>` à partir des fichiers XML ALTO.

1. lxml dev TEAM. *Lxml : Powerful and Pythonic XML Processing Library Combining Libxml2/Libxslt with the ElementTree API.* Version 4.9.1. URL : `https://lxml.de/` (visité le 14/09/2022).

2. Kenneth REITZ. *Requests : Python HTTP for Humans.* Version 2.28.1. URL : `https://requests.readthedocs.io` (visité le 14/09/2022).

3. Kelly CHRISTENSEN, Simon GABAY et Ariane PINCHE. *Alto 2 Tei.* URL : `https://github.com/kat-kel/alto2tei` (visité le 14/09/2022).

A.1.1 ./setup.py

Afin de faciliter l'utilisation de l'application par de divers utilisateurs qui n'ont pas forcement de l'expérience en programmation, j'ai packagé l'application. Après l'avoir installé (dans un environnement virtuel), la commande `alto2tei` appel l'application, comme se voit dans l'exemple de commande dessus et dans la ligne 18 du fichier `setup.py`.

```
1  from setuptools import find_packages, setup
2
3  setup(
4      name='alto2tei',
5      version='0.0.1',
6      packages=find_packages(),
7      install_requires=[
8          'certifi==2022.6.15',
9          'charset-normalizer==2.1.0',
10         'idna==3.3',
11         'lxml==4.9.1',
12         'PyYAML==6.0',
13         'requests==2.28.1',
14         'urllib3==1.26.11'
15     ],
16     entry_points={
17         'console_scripts': [
18             'alto2tei=src.__main__:main'
19         ]
20     }
21
22 )
```

A.1.2 ./src/__main__.py

Le module python `__main__.py` traite les arguments passés à l'application depuis la ligne de commande et puis mettre en pratique la construction des parties de l'arbre TEI demandées.

```
1  import argparse, os
2  import yaml
3  from collections import namedtuple
4  from pathlib import Path
5  from time import perf_counter
6
7  from src.build import TEI
8  from src.write_output import Write
9
10 def file_path(string):
```

```
    """Verify if the string passed as the argument --config is a valid file path.
    Args:
        string (str): file path to YAML configuration file.
    Raises:
        FileNotFoundError: informs user that the file path is invalid.
    Returns:
        (str): validated path to configuration file
    """
    if os.path.isfile(string):
        return string
    else:
        raise FileNotFoundError(string)


def get_args():
    """Parse command-line arguments and verify (1) the config file exist, (2) the TEI elements demanded can be constructed.
    """
    parser = argparse.ArgumentParser()
    parser.add_argument("--config", nargs=1, type=file_path, required=True,
                        help="path to the YAML configuration file")
    parser.add_argument("--version", nargs=1, type=str, required=True,
                        help="version of Kraken used to create the ALTO-XML files")
    parser.add_argument("--header", default=False, action='store_true',
                        help="produce TEI-XML with <teiHeader>")
    parser.add_argument("--sourcedoc", default=False, action='store_true',
                        help="produce TEI-XML with <sourceDoc>")
    parser.add_argument("--body", default=False, action='store_true',
                        help="produce TEI-XML with <body>")
    args = parser.parse_args()
    return args.config, args.version, args.header, args.sourcedoc, args.body

def main():
    config, version, header, sourcedoc, body = get_args()

    if body and not sourcedoc:
        print("")
        warning = '\n    Cannot produce <body> without <sourceDoc>.\n    To call the program with the --body option, include also the --sourcedoc option.'
        raise Exception(warning)

    with open(config[0]) as cf_file:
        config = yaml.safe_load(cf_file.read())
```

```
52
53      # for every directory in the path indicated in the configuration file,
54      # get the directory's name (str) and the paths of its ALTO files (os.
        path)
55      Docs = namedtuple("Docs", ["doc_name", "filepaths"])
56      docs = [Docs    (d.name,                                    # name of
        document folder
57                      [f for f in d.iterdir() if f.suffix==".xml"]) #
        relative filepath for file
58              for d in Path(config.get(("data"))["path"]).iterdir() if d.
        is_dir()]
59
60      for d in docs:
61          # instantiate the class TEI for the current document in the loop
62          tree = TEI(d.doc_name, d.filepaths)
63          tree.build_tree()
64          print("\n====================================")
65          print(f"\33[32m~ now processing document {d.doc_name} ~\x1b[0m")
66
67          if header:
68              print(f"\33[33mbuilding <teiHeader>\x1b[0m")
69              t0 = perf_counter()
70              tree.build_header(config, version[0])
71              print("|________finished in {:.4f} seconds".format(perf_counter
        () - t0))
72
73          if sourcedoc:
74              print(f"\33[33mbuilding <sourceDoc>\x1b[0m")
75              t0 = perf_counter()
76              tree.build_sourcedoc(config)
77              print("|________finished in {:.4f} seconds".format(perf_counter
        () - t0))
78
79          if body:
80              print(f"\33[33mbuilding <body>\x1b[0m")
81              t0 = perf_counter()
82              tree.build_body()
83              print("|________finished in {:.4f} seconds".format(perf_counter
        () - t0))
84
85          # -- output XML-TEI file --
86          if not os.path.exists('./data/'):
87              os.mkdir('./data/')
88          Write(d.doc_name, tree.root).write()
89
90  if __name__ == "__main__":
91      main()
```

La première étape de la construction du document TEI dans le module `./src/__main__.py` est la création d'une racine XML TEI, voir lignes 62-63. Dans un premier temps, l'arborescence de notre modélisation est réalisé avec des valeurs par défaut. La classe pour l'arbre d'un document a besoin de deux données : l'ARK du fac-similé numérique (le nom du dossier en cours de traitement) et une liste de fichiers XML ALTO ordonnée par numéro de folio.

A.1.3 `./src/order_files.py`

Pour créer une arborescence par défaut, la classe `TEI` du module `./src/build/` est instantanée dans la ligne 63 du `./src/__main__.py`. L'un de ses deux arguments est une liste de chemins des fichiers XML ALTO. Cette donnée est gérée par la classe `Files` du module `./src/order_files.py`, spécifiquement la méthode de la classe `order_files()` qui met à jour l'attribut de la classe `self.lf` avec une liste ordonnée de chemins de fichier.

```
1  # ------------------------------------------------------------
2  # Code by: Kelly Christensen
3  # Python class to organize the file paths in the data directory.
4  # ------------------------------------------------------------
5
6  import re
7  from collections import namedtuple
8
9  class Files:
10     def __init__(self, document, filepaths):
11         self.d = document
12         self.fl = filepaths  # list
13
14     def order_files(self):
15         File = namedtuple("File", ["num", "filepath"])
16         ordered_files = sorted([File(int(re.search(r"(\d+).xml$", f.name).
       group(1)), f)for f in self.fl])
17         return ordered_files
```

A.2 Construction de l'arborescence TEI générique

Suite à la ligne 63 du module `./__main__.py`, la première étape de la construction du document TEI est la méthode `build_tree.py` de la classe `TEI` dans le module `./src/build.py`. Cette méthode est la première de la classe à être appelée dans le script parce qu'elle créé la racine XML TEI du document en cours de développement. Après la création de l'arborescence générique, la même classe `TEI` du module `./src/build.py` est utilisée pour ses méthodes `build_header()`, `build_sourcedoc()`, et `build_body()`.

A.2.1 ./src/build.py

La classe `TEI` organise des données du fac-similé en cours de traitement ainsi qu'applique les méthodes qui construisent les branches principales de l'arbre TEI. Toute branche peut disposer des données suivantes, qui sont instantanées avec la classe

- l'ARK (`self.d`)
- le chemin de tout fichier XML ALTO (`self.fp`)
- les métadonnées récupérées du *manifest* IIIF, du catalogue général de la BnF, et du site SUDOC (`self.metadata`)
- une liste de toute étiquette utilisée dans les fichiers (`self.tags`)
- la racine de l'arbre XML TEI en cours de construction (`self.root`)
- les zones de *SegmOnto* utilisées dans les fichiers (`self.segmonto_zones`)
- les lignes de *SegmOnto* utilisées dans les fichiers (`self.segmonto_lines`)

Les métadonnées sont récupérées lors de la construction du `<teiHeader>`. Par conséquent, la méthode pour construire le `<teiHeader>` a besoin de la saisie, afin de savoir la version de *Kraken*, ainsi que le fichier de configuration, pour informer du `<respStmt>`, voir ligne 32. Le `<sourceDoc>` a besoin d'une autre partie du fichier de configuration, celle qui informe des composants de la requête HTTPS envoyée à la IIIF Image API qui est représentée dans l'attribut `@source` des éléments `<zone>`, voir ligne 37.

```
1  from lxml import etree
2  from src.teiheader_metadata.clean_data import Metadata
3  from src.teiheader_build import teiheader
4  from src.sourcedoc_build import sourcedoc
5  from src.text_data import Text
6  from src.body_build import body
7
8  class TEI:
9      metadata = {"sru":None, "iiif":None}
10     tags = {}
11     root = None
12     segmonto_zones = None
13     segmonto_lines = None
14     def __init__(self, document, filepaths):
15         self.d = document  # (str) this document's name / name of directory
       contiaining the ALTO files
16         self.fp = filepaths  # (list) paths of ALTO files
17         self.metadata  # (dict) dict with two keys ("iiif", "sru"), each of
       which is equal to its own dictionary of metadata
18         self.tags  # (dict) a label-ref pair for each tag used in this
      document's ALTO files
19         self.root  # (etree_Element) root for this document's XML-TEI tree
20         self.segmonto_zones
21         self.segmonto_lines
22
```

```
23
24      def build_tree(self):
25          """Parse and map data from ALTO files to an XML-TEI tree's <
        teiHeader> and <sourceDoc>.
26          """
27
28          # instantiate the XML-TEI root for this document and assign the
        root basic attributes
29          tei_root_att = {"xmlns":"http://www.tei-c.org/ns/1.0", "{http://www
        .w3.org/XML/1998/namespace}id":f"ark_12148_{self.d}"}
30          self.root = etree.Element("TEI", tei_root_att)
31
32      def build_header(self, config, version):
33          # confirm that the metadata is being récupéré
34          self.metadata = Metadata(self.d, config["iiifURI"]).prepare()
35          self.root, self.segmonto_zones, self.segmonto_lines = teiheader(
        self.metadata, self.d, self.root, len(self.fp), config, version, self.fp
        , self.segmonto_zones, self.segmonto_lines)
36
37      def build_sourcedoc(self, config):
38          sourcedoc(self.d, self.root, self.fp, self.tags, self.
        segmonto_zones, self.segmonto_lines, config["iiifURI"])
39
40      def build_body(self):
41          text = Text(self.root)
42          body(self.root, text.data)
```

A.3 Récupération des métadonnées

En revenant au module `./src/__main__.py` qui dirige les étapes de l'application, voir lignes 67-70, le `<teiHeader>` est créé si la commande pour démarrer l'application avait l'argument `--header`. Pour créer le `<teiHeader>`, l'application appelle la méthode `build_header()` de la classe `TEI` du module `./src/build.py`. La première étape de cette méthode est la récupération es métadonnées. Pour créer les métadonnées, la méthode créé une instance de la classe `Metadata` du module `./src/teiheader_metadata/clean_data.py`. L'instance de cette classe a besoin d'une donnée de la classe `TEI`, l'ARK du document traité (`self.d`, voir ligne 34), ainsi que les parties de la requête à envoyer à l'IIIF Image API, qui sont données par les utilisateurs dans le fichier de configuration. Parce que les données du fichier de configuration ne font pas partie de la classe `TEI`, la méthode le saisit directement par l'argument `config`, voir ligne 32 du module `./src/build.py`.

A.3.1 ./src/teiheader_metadata/clean_data.py

Depuis la ligne 32 du module `./src/build.py`, l'instance de la classe `Metadata` est créée avec deux arguments : (1) l'ARK du document venant de la classe `TEI` et (2) les données portant sur la requête pour l'API IIIF. La seule méthode de la classe `Metadata` est `prepare()`, qui créé des instances de deux classes. La classe `IIIF` du module `./src/teiheader_metadata/iiif_data.py` sert à récupérer les métadonnées du fac-similé numérique. Et la classe `SRU` du module `./src/teiheader_metadata/sru_data.py` sert à récupérer les métadonnées du document source depuis l'API SRU. La méthode `prepare()` les appelle dans un certain ordre afin que la relation entre le fac-similé numérique et le document source dont la notice est dans le catalogue général de la BnF soit saisie depuis le *manifest* IIIF.

```
1  from src.teiheader_metadata.iiif_data import IIIF
2  from src.teiheader_metadata.sru_data import SRU
3
4  class Metadata:
5      metadata = {"sru":None, "iiif":None}
6      def __init__(self, document, config):
7          self.d = document
8          self.metadata
9          self.iiifURI = config
10
11     def prepare(self):
12         # -- Parse data from document's IIIF manifest --
13         # instantiate the class IIIF for this document
14         iiif = IIIF(self.d, self.iiifURI)
15         # request the digital document's IIIF manifest data and clean it
       with methods from IIIF class
16         iiif_data = iiif.clean(iiif.request())
17         # add the cleaned IIIF data to this document's metadata
18         self.metadata.update({"iiif":iiif_data})
19
20         # -- Parse data from BnF's SRU API --
21         # isntantiate the class SRU for this document
22         sru = SRU(iiif_data["Catalogue ARK"])
23         # request the physical document's catalogue data from the BnF (
       response)
24         # and/or a boolean confirming if the physical document was found (
       perfect_match)
25         response, perfect_match = sru.request()
26         # clean the MARCXML response and prepare a dictinoary of relevant
       metadata
27         sru_data = sru.clean(response, perfect_match)
28         # add the cleaned catalogue data to this document's metadata
29         self.metadata.update({"sru":sru_data})
```

```
30
31          return self.metadata
```

A.3.2 ./src/tei_metadata/iiif_data.py

Dans le module `./src/teiheader_metadata/clean_data.py`, la première classe utilisée dans sa méthode `prepare()` est celle-là qui récupère des métadonnées du *manifest* IIIF qui est renvoyée suite à une requête à l'API, voir la méthode `request()` sur les lignes 19-27. Uniquement les données du *manifest* déterminées essentielles, selon la justification exposée dans le chapitre 6, sont renvoyée par la méthode `clean()`, voir les lignes 29-50.

```
1  # ------------------------------------------------------------
2  # Code by: Kelly Christensen
3  # Python class to parse and store data from a document's IIIF manifest.
4  # ------------------------------------------------------------
5
6  import os
7  import requests
8  import re
9
10 class IIIF:
11     def __init__(self, document, iiifURI):
12         self.document = document
13         self.scheme = iiifURI["scheme"]
14         self.server = iiifURI["server"]
15         self.manifest_prefix = iiifURI["manifest_prefix"]
16         self.manifest_suffix= iiifURI["manifest_suffix"]
17
18     def request(self):
19         # Request manifest from the IIIF Presentation API
20         r = requests.get(f"{self.scheme}://{self.server}{self.
    manifest_prefix}{os.path.basename(self.document)}{self.manifest_suffix}"
    )
21         try:
22             response = {d["label"]:d["value"] for d in r.json()["metadata"
    ]}
23         except:
24             print("The IIIF manifest was not read correctly.")
25             response = {}
26         return response
27
28     def clean(self, response):
29         """Clean metadata received from Gallica API.
30         Returns:
31             clean_data (dict): cleaned data from IIIF manifest with values
    == None if not present in API request
```

```
32          """
33
34          # Make defaultdict for cleaned metadata
35          fields = ["Relation", "Catalogue ARK", "Repository", "Shelfmark", "
    Title", "Language", "Creator", "Date"]
36          clean_data= {}
37          {clean_data.setdefault(f, None) for f in fields}
38          for k,v in response.items():
39              if type(v) is list and list(v[0].keys())[0]=="@value":
40                  clean_data[k]=v[0]["@value"]
41              else:
42                  clean_data[k]=v
43          # Derive catalogue ARK from "Relation" field; this will be used to
    access the BnF's SRU API
44          if clean_data["Relation"] and re.search(r"\/((?:ark:)\/\w+\/\w+)",
    clean_data["Relation"]):
45              clean_data["Catalogue ARK"]=(re.search(r"\/((?:ark:)\/\w+\/\w+)
    ", clean_data["Relation"]).group(1))
46          # Clean author name, getting rid of ". Auteur du texte" at the end
    of the string
47          if clean_data["Creator"]:
48              clean_data["Creator"]=re.sub(r"(\s\(|\.).+", '', clean_data["
    Creator"])
49          return clean_data
```

A.3.3 ./src/teiheader_metadata/sru_data.py

Après avoir stocké un dictionnaire des données du *manifest* IIIF comme l'attribut `self.metadata` dans la classe `Metadata`, la méthode `preapre()` de cette dernière classe créé une instance de la classe `SRU` avec l'ARK du document physique dans le catalogue général de la BnF. Dans un premier temps, la classe `SRU` dispose d'une méthode `request()`, voir lignes 19-34, pour demander les données XML UNIMARC du catalogue général. Cette méthode renvoie le boolean pour dire si la requête a trouvé la bonne notice dans la catalogue (`perfect_match`) ainsi que la racine XML des données UNIMARC (`root`). Dans un deuxième temps, plusieurs méthodes sont utilisées pour extraire les données ciblées, selon notre modélisation expliquée dans les chapitres 6 et 7.

```
1 # ------------------------------------------------------------
2 # Code by: Kelly Christensen
3 # Python class to parse and store data from the BNF's general catalogue.
4 # ------------------------------------------------------------
5
6 from lxml import etree
7 import requests
8 import re
```

```
9
10 NS = {"s":"http://www.loc.gov/zing/srw/", "m":"info:lc/xmlns/marcxchange-v2
   "}
11
12
13 class SRU:
14     def __init__(self, ark):
15         """Args:
16             ark (string): document ARK in BnF catalogue"""
17         self.ark = ark
18
19     def request(self):
20             """Request metadata from the BnF's SRU API.
21             Returns:
22                 root (etree_Element): parsed XML tree of requested Unimarc
   data
23                 perfect_match (boolean): True if request was completed with
    Gallica ark / directory basename
24             """
25             print("|        requesting data from BnF's SRU API")
26             r = requests.get(f'http://catalogue.bnf.fr/api/SRU?version=1.2&
   operation=searchRetrieve&query=(bib.persistentid all "ark:/12148/{self.
   ark}")')
27             root = etree.fromstring(r.content)
28             if root.find('.//s:numberOfRecords', namespaces=NS).text=="0":
29                 perfect_match = False
30                 print(f"|        \33[31mdid not find digitised document in
   BnF catalogue\x1b[0m")
31             else:
32                 perfect_match = True
33                 print(f"|        \33[32mfound digitised document in BnF
   catalogue\x1b[0m")
34             return root, perfect_match
35
36     def author_data(self, author_element, count):
37         """Create and fill datafields for relevant author data.
38         Args:
39             author_element (etree_Element): <mxc: datafield> being parsed
40             count (int): author's count in processing
41         Returns:
42             data (dict) : relevant authorship data (isni, surname, forename
   , xml:id)
43         """
44         # create and set defaults for author data
45         fields = ["isni", "primary_name", "secondary_name", "namelink" , "
   xmlid"]
46         data = {}
```

```
47          {data.setdefault(f, None) for f in fields}
48
49          # -- identifier (700s subfield "o") --
50          has_isni = author_element.find('m:subfield[@code="o"]', namespaces=
    NS)
51          if has_isni is not None and has_isni.text[0:4]=="ISNI":
52              data["isni"] = has_isni.text[4:]
53
54          # -- primary name (700s subfield "a") --
55          has_primaryname = author_element.find('m:subfield[@code="a"]',
    namespaces=NS)
56          if has_primaryname is not None:
57              data["primary_name"] = has_primaryname.text
58
59          # -- secondary name (700s subfield "b") --
60          has_secondaryname = author_element.find('m:subfield[@code="b"]',
    namespaces=NS)
61          if has_secondaryname is not None:
62              x = re.search(r"(?:van der)|(?:de la)|(?:de)|(?:du)|(?:von)|(?:
    van)", has_secondaryname.text)
63              if x:
64                  data["namelink"] = x.group(0)
65              y = re.sub(r"(?:van der)|(?:de la)|(?:de)|(?:du)|(?:von)|(?:van
    )","", has_secondaryname.text)
66              if y != "":
67                  data["secondary_name"] = y
68
69          # -- unique xml:id for the author --
70          if data["primary_name"]:
71              name = data["primary_name"]
72              data["xmlid"] = f"{name[:2]}{count}"
73          elif data["secondary_name"]:
74              data["xmlid"] = f"{name[:2]}{count}"
75          else:
76              data["xmlid"] = f"au{count}"
77
78          return data
79
80      def clean(self, root, perfect_match):
81          """Parse and clean data from SRU API response.
82          Returns:
83              data (dict): all relevant metadata from BnF catalogue
84          """
85          # create and set defaults for data
86          fields = ["authors", "title", "ptr", "pubplace", "pubplace_key", "
    publisher", "date", "when", "date_cert", "date_resp", "country", "idno",
     "objectdesc", "lang", "repo", "settlement"]
```

```
87          data = {}
88          {data.setdefault(f, None) for f in fields}
89
90          if not perfect_match:
91              data["found"]=False
92
93          else:
94              data["found"]=True
95              # enter author data into data dictionary
96              data["authors"] = self.clean_authors(root)
97
98              # enter link to the work in the institution's catalogue
99              has_ptr = root.find('.//m:controlfield[@tag="003"]', namespaces
    =NS)
100             if has_ptr is not None:
101                 data["ptr"] = has_ptr.text
102
103             # enter date of publication
104             has_date_100 = root.find('.//m:datafield[@tag="100"]/m:subfield
    [@code="a"]', namespaces=NS)
105             if has_date_100 is not None and has_date_100.text[8]!="u":
106                 data["date"] = has_date_100.text[9:13]
107                 data["when"] = has_date_100.text[9:13]
108                 data["date_cert"] = self.date_cert(has_date_100.text[8])
109                 data["date_resp"] = "BNF"
110             else:
111                 has_date_210 = root.find('.//m:datafield[@tag="210"]/m:
    subfield[@code="d"]', namespaces=NS)
112                 if has_date_210 is not None:
113                     data["date"] = has_date_210.text
114
115
116             # enter language of document
117             has_lang = root.find('.//m:datafield[@tag="101"]/m:subfield[
    @code="a"]', namespaces=NS)
118             if has_lang is not None:
119                 data["lang"] = has_lang.text
120
121             # enter country code of publication place
122             has_place_key = root.find('.//m:datafield[@tag="102"]/m:
    subfield[@code="a"]', namespaces=NS)
123             if has_place_key is not None:
124                 data["pubplace_key"] = has_place_key.text
125
126             # enter cleaned title
127             has_title = root.find('.//m:datafield[@tag="200"]/m:subfield[
    @code="a"]', namespaces=NS)
```

```
128             if has_title is not None:
129                 data["title"] = has_title.text
130 
131             # enter type of document (manuscript or print)
132             has_objectdesc = root.find('.//m:datafield[@tag="200"]/m:
    subfield[@code="b"]', namespaces=NS)
133             if has_objectdesc is not None:
134                 data["objectdesc"] = has_objectdesc.text
135 
136             # enter publication place
137             has_place = root.find('.//m:datafield[@tag="210"]/m:subfield[
    @code="a"]', namespaces=NS)
138             if has_place is not None:
139                 data["pubplace"] = has_place.text
140 
141             # enter publisher
142             has_publisher = root.find('.//m:datafield[@tag="210"]/m:
    subfield[@code="c"]', namespaces=NS)
143             if has_publisher is not None:
144                 data["publisher"] = has_publisher.text
145 
146             # enter country where the document is conserved
147             has_country = root.find('.//m:datafield[@tag="801"]/m:subfield[
    @code="a"]', namespaces=NS)
148             if has_country is not None:
149                 data["country"] = has_country.text
150 
151             # enter catalogue number of the document in the insitution
152             has_isno = root.find('.//m:datafield[@tag="930"]/m:subfield[
    @code="a"]', namespaces=NS)
153             if has_isno is not None:
154                 data["idno"] = has_isno.text
155 
156             has_repo = root.find('.//m:datafield[@tag="930"]/m:subfield[
    @code="b"]', namespaces=NS)
157             if has_repo is not None:
158                 data["settlement"], data["repo"] = self.request_sudoc_data(
    has_repo.text)
159 
160         return data
161 
162     def request_sudoc_data(self, num_rcr):
163         """Request and parse a search results page from SUDOC for the
    repository whose RCR number was found in the Unimarc 930B data.
164         Example of the relevant HTML from the SUDOC results page:
165             <tr>
```

```
166                 <td class="rec_lable"><div><span>Adresse postale : </span
    ></div></td>
167                 <td class="rec_title">
168                     <div><span>NAME OF LIBRARY</span></div>
169                     <div><span>STREET ADDRESS</span></div>
170                     <div><span>CITY AND POSTAL CODE</span></div>
171                     <div><span>COUNTRY</span></div>
172                     <div><span> </span></div>
173                 </td>
174             </tr>
175         """
176         # Get the HTML from the search result for the institution's RCR
    number from Sudoc
177         print("|        requesting data from Sudoc")
178         r = requests.get(f"http://www.sudoc.abes.fr/cbs/xslt//DB=2.2/CMD?
    ACT=SRCHA&IKT=8888&SRT=RLV&TRM={num_rcr}")
179         if r.status_code == 200:
180             # Parse the HTML into an etree document
181             doc = etree.HTML(r.content)
182             # Grab the <span> that has the label "Adresse postale"
183             address_label = doc.xpath('.//td[@class="rec_lable"]//span[
    contains(text(),"Adresse postale")]')[0]
184             # Grab the <td> element directly after the <td> that contains
    the label "Adresse postale"
185             address_title = address_label.getparent().getparent().getnext()
186             # Find the div/span that starts with the country "France"
187             country = address_title.xpath('.//span[starts-with(text(), "
    France")]')[0]
188             # Grab the text in the first div/span of the address, which
    should be the name of the institution
189             repository = address_title.xpath('.//span')[0].xpath('string()'
    )
190             # Grab the div/span immediately preceding that which declare
    the country, which should contain the City
191             city_and_postal_code = country.getparent().getprevious().xpath(
    'string()')
192             # Remove any postal code, other numbers, and/or "CEDEX" from
    the repository and clean up spaces around remaining text
193             city_and_postal_code_without_cedex = re.sub(r'CEDEX', '',
    city_and_postal_code)
194             city_with_spaces = re.search(r'(\D+)',
    city_and_postal_code_without_cedex).group(0)
195             city_without_first_space = re.sub(r'^\s*', '', city_with_spaces
    )
196             city_without_first_or_last_space = re.sub(r'\s*$', '',
    city_without_first_space)
197             if city_without_first_or_last_space and repository:
```

```
198                 print(f"|        \33[32mfound Sudoc data about city and
    repository\x1b[0m")
199             else:
200                 print(f"|        \33[31mdid not find city and repository in
     Sudoc data\x1b[0m")
201         else:
202             city_without_first_or_last_space = None
203             repository = None
204         return city_without_first_or_last_space, repository
205 
206     def date_cert(self, key):
207         """Assigns a degree of certainty to the document's publication date
    .
208         """
209         # UNIMARC Norms (ca. 2012)
210         # a = currently published continuing resource
211         # b = continuing resource no longer being published
212         # c = continuing resource of unknown status
213         # d = monograph complete when issued, or issued within one calendar
     year
214         # e = reproduction of a document
215         # f = monograh, date of publication uncertain
216         # g = mongraph whose publication continues for more than a year
217         # h = monograph with both actual and copyright/privilege date
218         # i = monograph with both release/issue date and production date
219         # j = document with detailed date of production
220         # k = monograph published in a certain year and printed in a
    different year
221         # u = dates of publication unkonwn
222         if key == "a" or key == "b" or key == "d" or key == "e" or key == "
    h" or key == "i" or key == "j":
223             degree = "high"
224         if key == "g" or key == "k":
225             degree = "medium"
226         if key == "f":
227             degree = "low"
228         return degree
229 
230     def clean_authors(self, root):
231         """Parses and cleans author data from Unimarc fields 700 and/or
    701.
232         Returns:
233             authors (dict): relevant authorship data (isni, surname,
    forename, xml:id)
234         """
235         authors = []
236         count = 0
```

```
237         if root.find('.//m:datafield[@tag="700"]', namespaces=NS) is not
    None:
238             # datafield 700 is not repeatable
239             author_element = root.find('.//m:datafield[@tag="700"]',
    namespaces=NS)
240             count+=1
241             authors.append(self.author_data(author_element, count))
242         if root.find('.//m:datafield[@tag="701"]', namespaces=NS) is not
    None:
243             # datafield 701 is repeatable
244             author_elements = root.findall('.//m:datafield[@tag="701"]',
    namespaces=NS)
245             for element in author_elements:
246                 count+=1
247                 authors.append(self.author_data(element, count))
248         return authors
```

A.4 Construction du <teiHeader>

Avec les métadonnées récupérées et disponibles comme l'attribut `self.metadata` de la classe `TEI` du module `./src/build.py` (voir ligne 34), le `<teiHeader>` peut être construit. Dans notre modélisation, l'arborescence du `<teiHeader>` peut varier un peu. Le `<teiHeader>` par défaut aura un certain nombre d'auteurs imbriqués dans le `<titleStmt>` et `<sourceDesc>`, selon le nombre trouvé lors de la récupération des métadonnées. Pour cette raison, la classe destinée à la construction de l'arborescence générique (`DefaultTree`) a déjà besoin des métadonnées récupérées, même si elle ne les met pas en place.

A.4.1 ./src/teiheader_build.py

La fonction `teiheader()` du module `./src/teiheader_build.py` créé les instances de deux classes : (1) `DefaultTree`, qui met en place les valeurs par défaut au cas où une donnée n'était pas récupérée, et (2) `FullTree` qui rempli la branche du `<teiHeader>`.

```
1  # -------------------------------------------------------------
2  # Code by: Kelly Christensen
3  # Python script to assemble the <teiHeader> of a TEI file.
4  # -------------------------------------------------------------
5
6  from src.teiheader_default import DefaultTree
7  from src.teiheader_full import FullTree
8
9  NS = {"s":"http://www.loc.gov/zing/srw/", "m":"info:lc/xmlns/marcxchange-v2
       "}
10
```

```
11
12 def teiheader(metadata, document, root, count_pages, config, version,
   filepaths, segmonto_zones, segmonto_lines):
13     """Create all elements of the <teiHeader>.
14     Args:
15         document (str): name of directory containing ALTO-encoded
   transcriptions of the document's pages
16         root (etree): XML-TEI tree
17         count_pages (string): number of files in directory
18     Returns:
19         root (etree): XML-TEI tree
20     """
21
22     # step 1 -- generate default <teiHeader>
23     elements = DefaultTree(config, document, root, metadata, count_pages,
   version)  # deafult_teiheader.py
24     elements.build()
25
26     # step 2 -- enter available metadata into relevant element in <
   teiHeader>
27     htree = FullTree(elements.children, metadata)  # full_teiheader.py
28     htree.author_data()
29     htree.bib_data()
30     segmonto_zones, segmonto_lines = htree.segmonto_taxonomy(filepaths)
31     return root, segmonto_zones, segmonto_lines
```

A.4.2 ./src/teiheader_default.py

La classe `DefaultTree` du module `./src/teiheader_default.py` réalise notre modélisation du `<teiHeader>` qui est expliquée dans le chapitre 6.

```
1 # -------------------------------------------------------------
2 # Code by: Kelly Christensen
3 # Python class to build the architecture of a default <teiHeader>.
4 # -------------------------------------------------------------
5
6 from email.mime import application
7 from lxml import etree
8 from datetime import datetime
9 from collections import defaultdict
10
11 class DefaultTree:
12     children = defaultdict(list)
13     def __init__(self, config, document, root, metadata, count_pages,
   version):
14         self.config = config
15         self.document = document
```

```
16          self.root = root
17          self.sru = metadata["sru"]
18          self.iiif = metadata["iiif"]
19          self.count = str(count_pages)
20          self.version = version
21
22      def build(self):
23          if self.sru["found"]:
24              default_text = "Information not available."
25              num_authors = len(self.sru["authors"])
26          else:
27              default_text = "Digitised resource not found in BnF catalogue."
28              num_authors = 1  # method of extracting IIIF manifest data will
    only return 1 author
29
30          teiHeader = etree.SubElement(self.root, "teiHeader")
31          # Three children of the root <teiHeader>
32          fileDesc = etree.SubElement(teiHeader, "fileDesc")
33          profileDesc = etree.SubElement(teiHeader, "profileDesc")
34          encodingDesc = etree.SubElement(teiHeader, "encodingDesc")
35
36          # <fileDesc>
37          titleStmt = etree.SubElement(fileDesc, "titleStmt")
38          self.children["titleStmt"] = titleStmt
39          self.children["ts_title"] = etree.SubElement(titleStmt, "title")  #
    pass to other methods
40          self.children["ts_title"].text = default_text
41          for i in range(num_authors):
42              etree.SubElement(titleStmt, "author")
43          if num_authors == 0:
44              ts_author = etree.SubElement(titleStmt, "author")
45              ts_author.text = default_text
46          respStmt = etree.SubElement(titleStmt, "respStmt")
47          resp = etree.SubElement(respStmt, "resp")
48          resp.text = self.config["responsibility"]["text"]
49          for i in range(len(self.config["responsibility"]["resp"])):
50              persName = etree.SubElement(respStmt, "persName")
51              forename = etree.SubElement(persName, "forename")
52              forename.text = self.config["responsibility"]["resp"][i]["
    forename"]
53              surname = etree.SubElement(persName, "surname")
54              surname.text = self.config["responsibility"]["resp"][i]["
    surname"]
55              etree.SubElement(persName, "ptr", self.config["responsibility"
    ]["resp"][i]["ptr"])
56          extent = etree.SubElement(fileDesc, "extent")
57          etree.SubElement(extent, "measure", unit="images", n=self.count)
```

```
58          publicationStmt = etree.SubElement(fileDesc, "publicationStmt")
59          publisher = etree.SubElement(publicationStmt, "publisher")
60          publisher.text = self.config["responsibility"]["publisher"]
61          authority = etree.SubElement(publicationStmt, "authority")
62          authority.text = self.config["responsibility"]["authority"]
63          availability = etree.SubElement(publicationStmt, "availability",
    self.config["responsibility"]["availability"])
64          etree.SubElement(availability, "licence", self.config["
    responsibility"]["licence"])
65          today = datetime.today().strftime('%Y-%m-%d')
66          etree.SubElement(publicationStmt, "date", when=today)
67          sourceDesc = etree.SubElement(fileDesc, "sourceDesc")
68          bibl = etree.SubElement(sourceDesc, "bibl")
69          self.children["bibl"] = bibl
70          self.children["ptr"] = etree.SubElement(bibl, "ptr")  # pass to
    other methods
71          for i in range(num_authors):
72              etree.SubElement(bibl, "author")
73          if num_authors == 0:
74              bib_author = etree.SubElement(bibl, "author")
75              bib_author.text = default_text
76          self.children["bib_title"] = etree.SubElement(bibl, "title")  #
    pass to other methods
77          self.children["bib_title"].text = default_text
78          self.children["pubPlace"] = etree.SubElement(bibl, "pubPlace")  #
    pass to other methods
79          self.children["pubPlace"].text = default_text
80          self.children["publisher"] = etree.SubElement(bibl, "publisher")  #
     pass to other methods
81          self.children["publisher"].text = default_text
82          self.children["date"] = etree.SubElement(bibl, "date")  # pass to
    other methods
83          self.children["date"].text = default_text
84          msDesc = etree.SubElement(sourceDesc, "msDesc")
85          msIdentifier = etree.SubElement(msDesc, "msIdentifier")
86          self.children["country"] = etree.SubElement(msIdentifier, "country"
    )  # pass to other methods
87          self.children["settlement"] = etree.SubElement(msIdentifier, "
    settlement")  # pass to other methods
88          self.children["settlement"].text = default_text
89          self.children["repository"] = etree.SubElement(msIdentifier, "
    repository")  # pass to other methods
90          self.children["repository"].text = default_text
91          self.children["idno"] = etree.SubElement(msIdentifier, "idno")  #
    pass to other methods
92          self.children["idno"].text = default_text
93          altIdentifer = etree.SubElement(msIdentifier, "altIdentifier")
```

```
94          alt_idno = etree.SubElement(altIdentifer, "idno", type="ark")  #
    pass to other methods
95          alt_idno.text = self.document
96          physDesc = etree.SubElement(msDesc, "physDesc")
97          objectDesc = etree.SubElement(physDesc, "objectDesc")
98          self.children["p"] = etree.SubElement(objectDesc, "p")  # pass to
    other methods
99          self.children["p"].text = default_text
100
101         # <profileDesc>
102         langUsage = etree.SubElement(profileDesc, "langUsage")
103         self.children["language"] = etree.SubElement(langUsage, "language")
     # pass to other methods
104         self.children["language"].attrib["ident"] = ""
105
106         # <encodingDesc>
107         appInfo = etree.SubElement(encodingDesc, "appInfo")
108         application = etree.SubElement(appInfo, "application")
109         application.attrib["ident"] = 'Kraken'
110         application.attrib["version"] = self.version
111         app_label = etree.SubElement(application, "label")
112         app_label.text = "Kraken"
113         app_ptr = etree.SubElement(application, "ptr")
114         app_ptr.attrib["target"] = "https://github.com/mittagessen/kraken"
115         classDecl = etree.SubElement(encodingDesc, "classDecl")
116         taxonomy_id = {"{http://www.w3.org/XML/1998/namespace}id":"SegmOnto
    "}
117         self.children["taxonomy"] = etree.SubElement(classDecl, "taxonomy",
     taxonomy_id)
118         tax_bibl = etree.SubElement(self.children["taxonomy"], "bibl")
119         tax_title = etree.SubElement(tax_bibl, "title")
120         tax_title.text = "SegmOnto"
121         tax_ptr = etree.SubElement(tax_bibl, "ptr")
122         tax_ptr.attrib["target"]="https://github.com/segmonto"
```

A.4.3 ./src/teiheader_full.py

La classe `FullTree` prend toutes les métadonnées ainsi que les éléments créés pour le `<teiHeader>` générique afin qu'elle puisse les remplir avec les bonnes données.

```
1 # ------------------------------------------------------------
2 # Code by: Kelly Christensen
3 # Python class to map the document's and project's metadata to the default
     <teiHeader>.
4 # ------------------------------------------------------------
5
6 from lxml import etree
```

```
7  import re
8  from collections import namedtuple
9  from .sourcedoc_build import labels
10
11 class FullTree:
12     def __init__(self, children, metadata):
13         self.children = children
14         self.sru = metadata["sru"]
15         self.iiif = metadata["iiif"]
16
17     def author_data(self):
18         """Enter document's title and author into <titleStmt>.
19         """
20         if self.sru["found"]:  # if the document's IIIF manifest had a
    valid catalogue ARK
21             self.authors(self.children["titleStmt"], True, True)
22             self.authors(self.children["bibl"], True, False)
23         else:  # if the document's IIIF manifest didn't have a valid
    catalogue ARK
24             self.authors(self.children["titleStmt"], False, True)
25             self.authors(self.children["bibl"], False, False)
26
27     def authors(self, parent, is_catologue_match, is_first_id):
28         """Create elements about authorship in either fileDesc/titleStmt or
     fileDesc/sourceDesc/bibl.
29         Args:
30             parent (etree_Element): the parent element for the author data
    (<titleStmt> or <bibl>)
31             is_catologue_match (boolean): True if the document's metadata
    was found in the BnF catalogue
32             is_first_id (boolean): True if the author id is presented for
    the first time, aka "xml:id"
33                                    if it's not the first time, the id will
     be "ref"
34         """
35
36         if not parent.find('./author').text:  # if the default tree was not
     built for 0 authors and doesn't have default text
37             xml_id = "{http://www.w3.org/XML/1998/namespace}id"
38             if is_catologue_match:
39                 for count, author_root in enumerate(parent.findall('./
    author')):
40                     if is_first_id:
41                         author_root.attrib[xml_id] = self.sru["authors"][
    count]["xmlid"]
42                     else:
43                         ref = self.sru["authors"][count]["xmlid"]
```

```
                        author_root.attrib["ref"] = f"#{ref}"
                    persname = etree.SubElement(author_root, "persName")
                    if self.sru["authors"][count]["secondary_name"]:
                        forename = etree.SubElement(persname, "forename")
                        forename.text = self.sru["authors"][count]["
secondary_name"]
                    if self.sru["authors"][count]["namelink"]:
                        namelink = etree.SubElement(persname,"nameLink")
                        namelink.text = self.sru["authors"][count]["
namelink"]
                    if self.sru["authors"][count]["primary_name"]:
                        surname = etree.SubElement(persname, "surname")
                        surname.text = self.sru["authors"][count]["
primary_name"]
                    if self.sru["authors"][count]["isni"]:
                        ptr = etree.SubElement(persname, "ptr")
                        ptr.attrib["type"] = "isni"
                        ptr.attrib["target"] = self.sru["authors"][count]["
isni"]
            else:
                author_root = parent.find('./author')
                if self.iiif["Creator"] is not None:
                    a = self.iiif["Creator"]
                    if is_first_id:
                        author_root.attrib[xml_id] = f"{a[:2]}"
                    else:
                        author_root.attrib["ref"] = f"#{a[:2]}"
                    name = etree.SubElement(author_root, "name")
                    name.text = a

    def bib_data(self):
        """In the <bibl>, enter the document's catalogue pointer (ptr),
author, title, publication place, publisher, date.
            In the <msDesc>, enter the institution's country code,
settlement, repository name, shelfmark for the doc, and doc type.
        """

        Entry = namedtuple("Entry", ["tei_element","attribute","iiif_data",
"unimarc_data"])
        entries =   [
                    # <title> in <titleStmt>, data from IIIF or SRU
                    Entry("ts_title",None,"Title","title"),
                    # @target for <ptr> element in <bibl>, only data from
SRU
                    Entry("ptr","target",None,"ptr"),
                    # <title> in <bibl>, data from IIIF or SRU
                    Entry("bib_title",None,"Title","title"),
```

```
                    # <pubPlace> in <sourceDesc>, only data from SRU
                    Entry("pubPlace",None,None,"pubplace"),
                    # @key for <pubPlace> in <sourceDesc>, only data from
SRU
                    Entry("pubPlace","key",None,"pubplace_key"),
                    # <publisher> in <sourceDesc>, only data from SRU
                    Entry("publisher",None,None,"publisher"),
                    # <date> in <sourceDesc>, data from IIIF or SRU
                    Entry("date",None,"Date","date"),
                    # @when for <date> in <sourceDesc>, only data from SRU
                    Entry("date","when",None,"when"),
                    # @cert for <date> in <sourceDesc>, only data from SRU
                    Entry("date","cert",None,"date_cert"),
                    # @resp for <date> in <sourceDesc>, only data from SRU
                    Entry("date","resp",None,"date_resp"),
                    # @key for <country> in <msDesc>, onyl data from SRU
                    Entry("country","key",None,"country"),
                    # <settlement> in <msDesc>, only data from SRU
                    Entry("settlement", None, None, "settlement"),
                    # <respository> in <msDesc>, only data from IIIF
                    Entry("repository",None,"Repository","repo"),
                    # <idno> in <msDesc>, data from IIIF or SRU
                    Entry("idno",None,"Shelfmark","idno"),
                    # <p> in <objectDesc>, only data from SRU
                    Entry("p",None,None,"objectdesc"),
                    # <language> in <profileDesc>, only data from IIIF
                    Entry("language",None,"Language",None),
                    # @ident for <language> in <profileDesc>, only data
from SRU
                    Entry("language","ident",None,"lang")]
        for e in entries:
            if self.sru and e.unimarc_data and self.sru[e.unimarc_data]:
                self.entry(self.sru[e.unimarc_data], self.children[e.
tei_element], e.attribute)
            elif e.iiif_data and self.iiif[e.iiif_data]:
                self.entry(self.iiif[e.iiif_data], self.children[e.
tei_element], e.attribute)

    def entry(self, data, tei_element, attribute):
        if attribute:
            tei_element.attrib[attribute] = data
        else:
            tei_element.text = data

    def segmonto_taxonomy(self, filepaths):
        # List all the SegmOnto tags and a URL pointing to their
description.
```

```
125         SegmOntoZones = {
126                 "CustomZone":"https://segmonto.github.io/gd/gdZ/CustomZone/
    ",
127                 "DamageZone":"https://segmonto.github.io/gd/gdZ/DamageZone"
    ,
128                 "DecorationZone":"https://segmonto.github.io/gd/gdZ/
    DecorationZone",
129                 "DigitizationArtefactzone":"https://segmonto.github.io/gd/
    gdZ/DigitizationArtefactzone",
130                 "DropCapitalZone":"https://segmonto.github.io/gd/gdZ/
    DropCapitalZone",
131                 "MainZone":"https://segmonto.github.io/gd/gdZ/MainZone",
132                 "MusicZone":"https://segmonto.github.io/gd/gdZ/MusicZone",
133                 "NumberingZone":"https://segmonto.github.io/gd/gdZ/
    NumberingZone",
134                 "QuireMarksZone":"https://segmonto.github.io/gd/gdZ/
    QuireMarksZone",
135                 "RunningTitleZone":"https://segmonto.github.io/gd/gdZ/
    RunningTitleZone",
136                 "SealZone":"https://segmonto.github.io/gd/gdZ/SealZone",
137                 "StampZone":"https://segmonto.github.io/gd/gdZ/StampZone",
138                 "TableZone":"https://segmonto.github.io/gd/gdZ/TableZone",
139                 "TitlePageZone":"https://segmonto.github.io/gd/gdZ/
    TitlePageZone"
140             }
141         SegmOntoLines = {
142                 "CustomLine":"https://segmonto.github.io/gd/gdL/CustomLine/
    ",
143                 "DefaultLine":"https://segmonto.github.io/gd/gdL/
    DefaultLine",
144                 "DropCapitalLine":"https://segmonto.github.io/gd/gdL/
    DropCapitalLine",
145                 "HeadingLine":"https://segmonto.github.io/gd/gdL/
    HeadingLine",
146                 "InterlinearLine":"https://segmonto.github.io/gd/gdL/
    InterlinearLine",
147                 "MusicLine":"https://segmonto.github.io/gd/gdL/MusicLine"
148             }
149 
150         # Get all the tags used on the pages of this document.
151         all_tag_dicts = [labels(f) for f in filepaths]
152 
153         # With regex, extract the main part (string before a colon, if
    present) of a label in the tag dictionary.
154         # And use dictionary comprehension to parse all the labels in the
    document's tags dictionaries.
```

```
155         unique_labels = list(set(re.match(r"(\w+):?(\w+)?#?(\d?)?", value).
    group(1)\
156                                  for dic in all_tag_dicts\
157                                  for value in dic.values()))
158 
159         # Create a list of zone tags used in this document.
160         document_zones = [label for label in unique_labels if "Zone" in
    label]
161         # Create a list of line tags used in this document.
162         document_lines = [label for label in unique_labels if "Line" in
    label]
163 
164         # Descending directly from <taxonomy>, create the TEI element <
    category> for SegmOnto zones.
165         cat_id = {"{http://www.w3.org/XML/1998/namespace}id":"SegmOntoZones
    "}
166         category = etree.SubElement(self.children["taxonomy"], "category",
    cat_id)
167         # Enter into the <category> every zone in the document that is also
     named in the SemOnto guidelines.
168         for z in set(SegmOntoZones).intersection(set(document_zones)):
169             self.enter_taxonomy_category(category, z, SegmOntoZones[z])
170 
171         # Descending directly from <taxonomy>, create the TEI element <
    category> for SegmOnto lines.
172         cat_id = {"{http://www.w3.org/XML/1998/namespace}id":"SegmOntoLines
    "}
173         category = etree.SubElement(self.children["taxonomy"], "category",
    cat_id)
174         # Enter into the <category> every line in the document that is also
     named in the SemOnto guidelines.
175         for l in set(SegmOntoLines).intersection(set(document_lines)):
176             self.enter_taxonomy_category(category, l, SegmOntoLines[l])
177         return document_zones, document_lines
178 
179     def enter_taxonomy_category(self, category, tag, url):
180         """Enter into the TEI-XML tree a <catDesc> for a specific SegmOnto
    line or zone.
181 
182         Args:
183             category (etree_Element): root for the element <category> in
    the TEI-XML document
184             tag (string): name of the tag identified in the ALTO file
185             url (string): URL pointing to a description fo the line or zone
     in the SegmOnto guidelines
186         """
187         catDesc_id = {"{http://www.w3.org/XML/1998/namespace}id":f"{tag}"}
```

```
188         catDesc = etree.SubElement(category, "catDesc", catDesc_id)
189         title = etree.SubElement(catDesc, "title")
190         title.text = tag
191         ptr = etree.SubElement(catDesc, "ptr")
192         ptr.attrib["target"] = url
```

A.5 Construction du <sourceDoc>

La méthode `build_sourcedoc()` dans la classe `TEI` (voir lignes 37-38 du module `./src/build.py`) prend beaucoup d'attributs de la classe `TEI`. Elle a besoin de l'ARK du fac-similé numérique (`self.d`), la racine XML du document TEI en cours de construction, une liste des chemins vers les fichiers ALTO, une liste de toute ligne et toute zone `SegmOnto` utilisée dans les transcriptions du document. En plus, la classe a besoin d'une donnée qui n'est pas gérée par la classe `TEI` mais qui vient du fichier de configuration traitée par le module `__main__.py` est envoyée quand ce module appelle la création du `<sourceDoc>`, voir ligne 76 du module `__main__.py`.

A.5.1 ./src/sourcedoc_build.py

La module `./src/sourcedoc_build.py` contient la fonction `sourcedoc()` qui est appelée par la méthode `build_sourcedoc.py` de la classe `TEI`. La fonction `sourcedoc()` profite de la fonction `labels()` (voir lignes 15-22) pour gérer un dictionnaire qui associe l'identifiant chiffré d'une étiquette *SegmOnto*, que des modèles HTR y ont appliquée, au nom de l'étiquette. En plus d'un dictionnaire qui déchiffre les étiquettes, la fonction `sourcedoc()` dispose des classes gèrent certaines données pour le document en cours de traitement. La classe `Files` fournit une liste des fichiers XML ALTO ordonnée par le numéro de folio, pour que l'application les traite et mette leurs transcriptions dans le bon ordre. La classe `Attributes` gère la création des attributs XML à donnée aux éléments TEI créés, voir ligne 50. Enfin, la classe `SurfaceTree` gère la création des éléments XML TEI pour la racine TEI à partir des éléments descendant de la racine ALTO du fichier XML ALTO en cours de traitement dans la boucle, voir ligne 51.

```
1 # ------------------------------------------------------------
2 # Code by: Kelly Christensen
3 # Python script to map all the data of an ALTO file to the <sourceDoc> of a
      TEI file.
4 # ------------------------------------------------------------
5
6 from collections import defaultdict
7 from src.order_files import Files
8 from src.sourcedoc_attributes import Attributes
9 from src.sourcedoc_elements import SurfaceTree
```

```
from lxml import etree

NS = {'a':"http://www.loc.gov/standards/alto/ns-v4#"}  # namespace for the Alto xml


def labels(filepath):
    root = etree.parse(filepath).getroot()
    elements = [t.attrib for t in root.findall('.//a:OtherTag', namespaces=NS)]
    collect = defaultdict(dict)
    for d in elements:
        collect[d["ID"]] = d["LABEL"]
    tags = dict(collect)
    return tags


def sourcedoc(document_name, output_tei_root, filepath_list, tags, segmonto_zones, segmonto_lines, config):
    """Creates the <sourceDoc> for an XML-TEI file using data parsed from a series of ALTO files.
        The <sourceDoc> collates each ALTO file, which represents one page of a document, into a wholistic
        description of the document.
    """


    ordered_files = Files(document_name, filepath_list).order_files()

    # Create <sourceDoc> and its child <surfaceGrp>.
    sourceDoc = etree.SubElement(output_tei_root, "sourceDoc")

    for file in ordered_files:

        tags = labels(file.filepath)

        # Start count at 0 for number of entities on a page.
        blocks_on_page = 0
        lines_on_page = 0
        strings_on_page = 0
        glyphs_on_page = 0

        # Parse the XML tree for the ALTO file
        input_alto_root = etree.parse(file.filepath).getroot()
        # Instantiate the classes Attributes and SurfaceTree for the ALTO file
```

```
50          attributes = Attributes(document_name, file.num, input_alto_root,
    tags, config)
51          surface_tree = SurfaceTree(document_name, file.num, input_alto_root
    )
52
53          # -- SURFACE --
54          # For every page in the ALTO file, create a <surface> and assign
    its attributes.
55          surface = surface_tree.surface(sourceDoc, attributes.surface())
56
57          # -- TEXTBLOCK --
58          # For every <TextBlock> in a <PrintSpace>, create a <zone> and
    assign it attributes.
59          textblocks = attributes.zones("PrintSpace", "TextBlock",
    segmonto_zones)
60          for tb in textblocks:
61              # Only map the <TextBlock> to the XML-TEI tree if its @ID was
    found.
62              if tb.id:
63                  blocks_on_page+=1
64                  textblock = surface_tree.zone1(surface, tb.attributes, tb.
    id, blocks_on_page)
65
66              textlines = attributes.zones(f'TextBlock[@ID="{tb.id}"]', "
    TextLine", segmonto_lines)
67              # "tl" concerns <TextLine> and its descendant <Polygon>
68              for tl in textlines:
69                  # Only map the <TextLine> to the XML-TEI tree if its @ID
    was found.
70                  if tl.id:
71                      lines_on_page+=1
72                      textline = surface_tree.zone2(textblock, tb.id, tl.
    attributes, tl.id, lines_on_page)
73                      words = ""
74
75                      # If <TextLine> has child <String> that has all the
    line's textual content, map that to the TEI element <line>.
76                      if input_alto_root.find(f'.//a:TextLine[@ID="{tl.id}"]/
    a:String', namespaces=NS).get("CONTENT") is not None\
77                          and len(input_alto_root.find(f'.//a:TextLine[@ID="{
    tl.id}"]/a:String', namespaces=NS).getchildren()) == 0:
78                          # Map the textual data to the TEI element <line>.
79                          surface_tree.line(textline, tb.id, tl.id,
    lines_on_page, None)
80
81                      # If the line's textual content is expressed at the
    level of glyphs, map that textual data to TEI element <c>.
```

```
                elif input_alto_root.find(f'.//a:TextLine[@ID="{tl.id}"]/a:String', namespaces=NS).get("CONTENT") is not None\
                    and input_alto_root.find(f'.//a:TextLine[@ID="{tl.id}"]/a:String', namespaces=NS).get("CONTENT") != ""\
                    and len(input_alto_root.find(f'.//a:TextLine[@ID="{tl.id}"]/a:String', namespaces=NS).getchildren()) > 0:

                    # Loop through all the <String> or <SP> children of a <TextLine>
                    textline_children = input_alto_root.find(f'.//a:TextLine[@ID="{tl.id}"]', namespaces=NS).getchildren()
                    for textline_child in textline_children:

                        # If child of <TextLine> is a space <SP>
                        if etree.QName(textline_child).localname == "SP":
                            textline_child_id = textline_child.attrib["ID"]
                            space_data = attributes.zones(f'TextLine[@ID="{tl.id}"]', f'SP[@ID="{textline_child_id}"]', None)[0]
                            strings_on_page+=1
                            surface_tree.zone3(textline, tb.id, tl.id, space_data.attributes, space_data.id, strings_on_page)

                        # If a child of <TextLine> is a segment of text <String>
                        elif etree.QName(textline_child).localname == "String":
                            textline_child_id = textline_child.attrib["ID"]
                            string_data = attributes.zones(f'TextLine[@ID="{tl.id}"]', f'String[@ID="{textline_child_id}"]', None)[0]
                            strings_on_page+=1
                            string = surface_tree.zone3(textline, tb.id, tl.id, string_data.attributes, string_data.id, strings_on_page)

                            # Loop through all the <Glyph> children of a <String>
                            string_children = input_alto_root.findall(f'.//a:String[@ID="{textline_child_id}"]/a:Glyph', namespaces=NS)
                            if words == "":
                                words = words + "".join([g.get("CONTENT") for g in string_children])
                            else:
                                words = words + " " + "".join([g.get("CONTENT") for g in string_children])

```

```
                                for glyph_child in string_children:
                                    glyph_id = glyph_child.attrib["ID"]
                                    glyph_data = attributes.zones(f'String[@ID="{textline_child_id}"]', f'Glyph[@ID="{glyph_id}"]', None)[0]
                                    glyphs_on_page+=1
                                    glyph = surface_tree.zone4(string, tb.id, tl.id, textline_child_id, glyph_data.attributes, glyph_id, glyphs_on_page)
                                    surface_tree.car(glyph, glyph_child, tb.id, tl.id, textline_child_id, glyph_id, glyphs_on_page)

                        surface_tree.line(textline, tb.id, tl.id, lines_on_page, words)

    return output_tei_root
```

A.5.2 ./src/sourcedoc_elements.py

La classe `SurfaceTree` traite les éléments ciblés dans le fichier XML ALTO en cours de traitement dans la boucle et fournit une architecture pour tous les éléments utilisés dans notre modélisation du `<sourceDoc>`. Chaque fonction pour créer les éléments prend comme argument les attributs de l'élément, qui sont donnés depuis le module parent `./src/sourcedoc_build.py` qui les récupère depuis une instance de la classe `Atrributes`.

```
# ------------------------------------------------------------
# Code by: Kelly Christensen
# Python class to build elements inside the <sourceDoc> and map data to them.
# ------------------------------------------------------------

from lxml import etree
import re

NS = {'a':"http://www.loc.gov/standards/alto/ns-v4#"}  # namespace for the Alto xml


class SurfaceTree:
    """Creates a <surface> element and its children for one page (ALTO file) of a document.
    """

    def __init__(self, doc, folio, alto_root):
        self.doc = doc
        self.folio = folio
        self.root = alto_root
```

```
20
21      def surface(self, surface_group, page_attributes):
22          """Make the TEI <surface> element that will organize all of an ALTO
        file's data.
23
24          Args:
25              surface_group (_type_): _description_
26              page_attributes (_type_): _description_
27
28          Returns:
29              _type_: _description_
30          """
31          surface = etree.SubElement(surface_group, "surface",
        page_attributes)
32          # create <graphic> and assign its attributes
33          etree.SubElement(surface, "graphic", url=f"https://gallica.bnf.fr/
        iiif/ark:/12148/{self.doc}/f{self.folio}/full/full/0/native.jpg")
34          return surface
35
36      def zone1(self, surface, attributes, block_id, blocks_on_page):
37          """Make the xml:id and TEI <zone> element for the ALTO file's <
        TextBlock>.
38
39          Args:
40              surface (etree_Element): _description_
41              textblock_atts (dict): _description_
42              block_id (str): _description_
43              blocks_on_page (int): _description_
44
45          Returns:
46              _type_: _description_
47          """
48          xml_id = {"{http://www.w3.org/XML/1998/namespace}id":f"f{self.folio
        }-{block_id}-blockCount{blocks_on_page}"}
49          zone = etree.SubElement(surface, "zone", xml_id)
50          for k,v in attributes.items():
51              zone.attrib[k]=v
52          return zone
53
54      def zone2(self, textblock, block_parent, attributes, line_id,
        lines_on_page):
55          """Make the xml:id and TEI <zone> element for the second-level <
        zone> for the ALTO file's <TextLine>
56              and make the xml:id for the second-level <zone>'s <path>.
57
58          Args:
59              textblock (_type_): _description_
```

```
60            block_parent (_type_): _description_
61            textline_atts (_type_): _description_
62            line_id (_type_): _description_
63            lines_on_page (_type_): _description_
64
65        Returns:
66            _type_: _description_
67        """
68        # ----------------------------------------
69        zone_id = {"{http://www.w3.org/XML/1998/namespace}id":f"f{self.
    folio}-{block_parent}-{line_id}-lineCount{lines_on_page}"}
70        # Insert the <zone> with this xml:id into the TEI-XML tree.
71        zone = etree.SubElement(textblock, "zone", zone_id)
72        # Insert the
73        for k,v in attributes.items():
74            zone.attrib[k]=v
75        # ----------------------------------------
76        path_id = {"{http://www.w3.org/XML/1998/namespace}id":f"f{self.
    folio}-{block_parent}-{line_id}-lineCount{lines_on_page}-baseline"}
77        #
78        baseline = etree.SubElement(zone, "path", path_id)
79        #
80        b = self.root.find(f'.//a:TextLine[@ID="{line_id}"]', namespaces=NS
    ).get("BASELINE")
81        #
82        baseline.attrib["points"] = " ".join([re.sub(r"\s", ",", x) for x
    in re.findall(r"(\d+ \d+)", b)])
83        return zone
84
85    def line(self, textline, block_parent, line_parent, lines_on_page,
    extracted_words):
86        """If the ALTO file stores all of a line's textual data in the <
    TextLine> attribute @CONTENT,
87            make the xml:id for <line>.
88
89        Args:
90            textline (_type_): _description_
91            block_parent (_type_): _description_
92            line_parent (_type_): _description_
93            lines_on_page (_type_): _description_
94
95        Returns:
96            _type_: _description_
97        """
98        xml_id = {"{http://www.w3.org/XML/1998/namespace}id":f"f{self.folio
    }-{block_parent}-{line_parent}-lineCount{lines_on_page}-text"}
99        # If the
```

```
100         line = etree.SubElement(textline, "line", xml_id)
101         line.attrib["n"] = str(lines_on_page)
102         if extracted_words:
103             line.text = extracted_words
104         else:
105             line.text = self.root.find(f'.//a:TextLine[@ID="{line_parent
    }"]/a:String', namespaces=NS).get("CONTENT")
106         return line
107 
108     def zone3(self, textline, block_parent, line_parent, attributes, seg_id
    , strings_on_page):
109         """Make the xml:id and TEI <zone> element for the ALTO file's <
    String> (segment/word).
110 
111         Args:
112             textline (_type_): _description_
113             block_parent (_type_): _description_
114             line_parent (_type_): _description_
115             attributes (_type_): _description_
116             seg_id (_type_): _description_
117             segments_on_page (_type_): _description_
118 
119         Returns:
120             _type_: _description_
121         """
122         xml_id = {"{http://www.w3.org/XML/1998/namespace}id":f"f{self.folio
    }-{block_parent}-{line_parent}-{seg_id}-segCount{strings_on_page}"}
123         zone = etree.SubElement(textline, "zone", xml_id)
124         for k,v in attributes.items():
125             zone.attrib[k]=v
126 
127         if self.root.find(f'.//a:String[@ID="{seg_id}"]', namespaces=NS) is
     not None \
128             and self.root.find(f'.//a:String[@ID="{seg_id}"]', namespaces=
    NS).get("WC") is not None:
129             word_certainty = self.root.find(f'.//a:String[@ID="{seg_id}"]',
     namespaces=NS).get("WC")
130             cert_attribs = {
131                 "{http://www.w3.org/XML/1998/namespace}id":f"f{self.folio
    }-{block_parent}-{line_parent}-{seg_id}-segCount{strings_on_page}-cert",
132                 "target":f"#f{self.folio}-{block_parent}-{line_parent}-{
    seg_id}-segCount{strings_on_page}-text",
133                 "locus":"value",
134                 "degree":word_certainty
135             }
136             etree.SubElement(zone, "certainty", cert_attribs)
137         return zone
```

```
138
139     def zone4(self, string, block_parent, line_parent, seg_parent,
    attributes, glyph_id, glyphs_on_page):
140         """Make the xml:id and TEI <zone> element for the ALTO file's <
    String> (segment/word).
141
142         Args:
143             textline (_type_): _description_
144             block_parent (_type_): _description_
145             line_parent (_type_): _description_
146             attributes (_type_): _description_
147             seg_id (_type_): _description_
148             segments_on_page (_type_): _description_
149
150         Returns:
151             _type_: _description_
152         """
153         xml_id = {"{http://www.w3.org/XML/1998/namespace}id":f"f{self.folio
    }-{block_parent}-{line_parent}-{seg_parent}-{glyph_id}-glyphCount{
    glyphs_on_page}"}
154         zone = etree.SubElement(string, "zone", xml_id)
155         for k,v in attributes.items():
156             zone.attrib[k]=v
157
158         if self.root.find(f'.//a:Glyph[@ID="{glyph_id}"]', namespaces=NS).
    get("GC") is not None:
159             glyph_certainty = self.root.find(f'.//a:Glyph[@ID="{glyph_id}"]
    ', namespaces=NS).get("GC")
160             cert_attribs = {
161                 "{http://www.w3.org/XML/1998/namespace}id":f"f{self.folio
    }-{block_parent}-{line_parent}-{seg_parent}-{glyph_id}-glyphCount{
    glyphs_on_page}-cert",
162                 "target":f"#f{self.folio}-{block_parent}-{line_parent}-{
    seg_parent}-{glyph_id}-glyphCount{glyphs_on_page}-text",
163                 "locus":"value",
164                 "degree":glyph_certainty
165             }
166             etree.SubElement(zone, "certainty", cert_attribs)
167         return zone
168
169     def car(self, zone,  glyph, block_parent, line_parent, seg_parent,
    glyph_id, glyphs_on_page):
170         xml_id = {"{http://www.w3.org/XML/1998/namespace}id":f"f{self.folio
    }-{block_parent}-{line_parent}-{seg_parent}-{glyph_id}-glyphCount{
    glyphs_on_page}-text"}
171         car = etree.SubElement(zone, "c", xml_id)
```

```
172         if self.root.find(f'.//a:Glyph[@ID="{glyph_id}"]', namespaces=NS).
    get("WC") is not None:
173             word_certainty = self.root.find(f'.//a:Glyph[@ID="{glyph_id}"]'
    , namespaces=NS).get("WC")
174             cert_attribs = {
175                 "{http://www.w3.org/XML/1998/namespace}id":f"f{self.folio
    }-{block_parent}-{line_parent}-{seg_parent}-{glyph_id}-glyphCount{
    glyphs_on_page}-cert",
176                 "locus":"value",
177                 "degree":word_certainty
178             }
179             etree.SubElement(zone, "certainty", cert_attribs)
180         car.text = glyph.attrib["CONTENT"]
181         return car
```

A.5.3 ./src/sourcedoc_attributes.py

La classe `Attributes` parvient à récupérer les attributs des éléments XML ALTO et les transformer dans des éléments TEI selon notre modélisation, voir le chapitre 6. Comme expliqué dans le chapitre 7, qui décrit le *mapping* des données entre les deux schémas, les éléments `<zone>` du schéma TEI se ressemble l'un à l'autre. Pour cette raison, et pour économiser, la fonction `zones()` est complexe et possède beaucoup de conditions *if* pour s'adapter à l'élément XML ALTO en cours de traitement dans la boucle de la fonction `sourcedoc()` du module `./src/sourcedoc_build.py`.

```
1  # ------------------------------------------------------------
2  # Code by: Kelly Christensen
3  # Python class to parse the attributes of the <sourceDoc>'s elements.
4  # ------------------------------------------------------------
5
6  from lxml import etree
7  import re
8  from collections import namedtuple
9
10 NS = {'a':"http://www.loc.gov/standards/alto/ns-v4#"}  # namespace for the
    Alto xml
11
12
13 class Attributes:
14     def __init__(self, doc, folio, alto_root, tags, config):
15         self.doc = doc
16         self.folio = folio
17         self.root = alto_root
18         self.tags = tags
19         self.scheme = config["scheme"]
20         self.server = config["server"]
```

```
21          self.prefix = config["image_prefix"]
22
23      def surface(self):
24          """Create attributes for the TEI <surface> element using data
    parsed from the ALTO file's <Page> element.
25              The TEI attributes for <surface> are: @n (page number),
26                                                    @ulx (upper left x-axis
    pixel position, always 0),
27                                                    @uly (upper left y-axis
    pixel position, always 0),
28                                                    @lrx (lower right x-axis
    pixel position = width of page),
29                                                    @lry (lower right y-axis
    pixel position = length of page)
30          Returns:
31              attributes (dict): dictionary of attribute names and their
    values
32          """
33
34          # create a dictionary of attributes names and their values for the
    ALTO file's <Page> element
35          att_list = self.root.find('.//a:Page', namespaces=NS).attrib
36          # assign the ALTO file's extracted <Page> attribute values to TEI
    attribute names
37          attributes = {"{http://www.w3.org/XML/1998/namespace}id":f"f{self.
    folio}",
38                        "n":att_list["PHYSICAL_IMG_NR"],
39                        "ulx":"0",
40                        "uly":"0",
41                        "lrx":att_list["WIDTH"],
42                        "lry":att_list["HEIGHT"]}
43          return attributes
44
45      def zones(self, parent, target, segmonto_labels):
46          """Create attributes for one of the two types of TEI <zone>
    elements: (a) TextBlock and (b) TextLine.
47
48          Args:
49              parent (str): parent's entity name and @ID in the ALTO file for
     the entity being transformed into a <zone>, followed by a '/'
50                            eg. 'TextBlock[@ID="eSc_textblock_20c2f4d8"]'
51              target (str): entity name in the ALTO file for the entity being
     transformed into a <zone>
52                            eg. 'TextLine'
53          Returns:
54              attributes (list): list of dictionaries {attribute name (str):
    value (str)}
```

```
55          processed_blocks (list): IDs of the elements whose data were extracted
56      """
57
58      # Empty variables in which the zone's data will be stored
59      ZoneData = namedtuple("ZoneData", ["attributes", "id"])
60      output = []
61
62      # List all the XML elements that are children of the given parent
63      element_list = [z for z in self.root.findall(f'.//a:{parent}/a:{target}', namespaces=NS)]
64      #print(element_list)
65
66      for element in element_list:
67          # Only parse data from elements that have an ID / are valid
68          if "ID" in element.attrib:
69              attributes={}
70              id=element.attrib["ID"]
71              # Instantiate the named tuple ZoneData with an empty dictionary and the element's ID if it was found
72              data = ZoneData(attributes, id)
73              if "TAGREFS" in element.attrib and element.attrib["TAGREFS"] in self.tags:
74                  tag = str(self.tags[element.attrib["TAGREFS"]])
75
76                  # parse the three (possible) components of the targeted ALTO element's @TAGREFS, according to SegmOnto guidelines;
77                  # the 3 groups of this regex parse the following expected tag syntax: MainZone:column#1 --> (MainZone)(column)(1)
78                  tag_parts = re.match(r"(\w+):?(\w+)?#?(\d?)?", tag)
79                  data.attributes["type"]=tag_parts.group(1) or "none"
80                  main_type =  data.attributes["type"]
81                  if segmonto_labels is not None and main_type in segmonto_labels:
82                      data.attributes["corresp"]=f"#{main_type}"
83                  data.attributes["subtype"]=tag_parts.group(2) or "none"
84                  data.attributes["n"]=tag_parts.group(3) or "none"
85
86              # If XML element does not have attribute @TAGREFS (aka, is a segment/space/glyph), assign it a type
87              else:
88                  main_type = etree.QName(element).localname
89                  if main_type=="SP":
90                      main_type="Space"
91                  data.attributes["type"]=main_type
92                  #print(main_type)
93
```

```
94                # Only parse coordinate data if it is present
95                if "HPOS" in element.attrib:
96                    x = element.attrib["HPOS"]
97                    y = element.attrib["VPOS"]
98                    w = element.attrib["WIDTH"]
99                    h = element.attrib["HEIGHT"]
100
101                   data.attributes["ulx"]=x
102                   data.attributes["uly"]=y
103                   data.attributes["lrx"]=str(int(w)+int(x))
104                   data.attributes["lry"]=str(int(h)+int(y))
105
106               # Extract the attributes for the child <Polygon> of each
    targeted ALTO element and put that dictionary into a list
107               if element.find('.//a:Polygon', namespaces=NS) is not None
    and element.find('.//a:Polygon', namespaces=NS).attrib["POINTS"] is not
    None:
108                   points = element.find('.//a:Polygon', namespaces=NS).
    attrib["POINTS"]
109                   # Reformat the string of numbers from Polygon[@POINTS]
    so that every 2nd value is joined to the previous value by a comma;
110                   # eg. "2204 4621 2190 4528" --> "2204,4621 2190,4528"
111                   data.attributes["points"]=" ".join([re.sub(r"\s", ",",
    x) for x in re.findall(r"(\d+ \d+)", points)])
112
113               # Only parse coordinate data if it is present
114               if "HPOS" in element.attrib:
115                   data.attributes["source"]=f"{self.scheme}://{self.
    server}{self.prefix}/{self.doc}/f{self.folio}/{x},{y},{w},{h}/full/0/
    native.jpg"
116
117               output.append(data)
118
119       return output
```

A.6 Construction du <body>

La classe `TEI` du module `./src/build.py` construit le `<body>` grâce à la fonction `body()` qu'elle importe depuis le module `./src/body_build.py`, voir ligne 42. Mais en plus de la racine TEI, à partir de laquelle la branche `<body>` est construite, la fonction `body()` a besoin des données nettoyées du texte. Le texte nettoyé de tous les fichiers XML ALTO est géré par la classe `Text` du module `./src/text_data.py`. Le `<body>` a pour but de présenter une version du texte de toute page, toute rassemblée.

A.6.1 ./src/text_data.py

La classe `Text` possède l'attribut `self.data` qui est dérivé par la méthode de la classe `line_data()`. Cette méthode établit un *namedtuple* pour organiser toutes les données ciblées, voir ligne 19. Ensuite, elle attribut à la variable `data` une liste de lignes de texte extrait de l'élément `<line>` du `<sourceDoc>`. Afin que l'extrait du texte pour le `<body>` peut se faire pour tout type de fichier XML ALTO, même eux qui ne mettent pas du contenu de la ligne de texte dans l'attribut `@CONTENT` dans le `<TextLine>`, il faut que le `<sourceDoc>` constitue toujours le texte de la ligne dans l'élément `<line>`.

```
1  # ------------------------------------------------------------
2  # Code by: Kelly Christensen
3  # Python class to parse and store data from text in the <sourceDoc>.
4  # ------------------------------------------------------------
5
6  from collections import namedtuple
7
8
9  class Text:
10     def __init__(self, root):
11         self.root = root
12         self.data = self.line_data()
13
14     def line_data(self):
15         """Parse contextual and attribute data for each text line and store
       it in a named tuple.
16         Returns:
17             data (list of named tuples): list of data for each text line
18         """
19         Line = namedtuple("Line", ["id", "n", "text", "line_type", "
       zone_type", "zone_id", "page_id"])
20         data = [Line(
21             ln.getparent().get("{http://www.w3.org/XML/1998/namespace}id"),
         # @xml:id of the line's zone
22             ln.get("n"),  # line number
23             ln.text,  # text content of line
24             ln.getparent().get("type"),  # @type of line
25             ln.getparent().getparent().get("type"),  # @type of text block
       zone
26             ln.getparent().getparent().get("{http://www.w3.org/XML/1998/
       namespace}id"),  # @xml:id of text block zone
27             ln.getparent().getparent().getparent().get("{http://www.w3.org/
       XML/1998/namespace}id"),  # @xml:id of page
28         ) for ln in self.root.findall('.//line')]
29         return data
```

A.6.2 ./src/build_body.py

La fonction `body()` prend le texte nettoyé par la classe `Text` du module `./src/text_data.py` et boucle sur toute ligne donnée dedans. Les attributs de chaque ligne, y compris l'étiquette *SegmOnto* ainsi que la référence à la ligne dans le `<sourceDoc>`, accompagne le texte de la ligne, grâce au *namedtuple* que la classe `Text` utilise. La fonction `body()` du module `./src/build_body.py` créé le `<body>` grâce à une suite de conditions qui cherchent l'étiquette de la ligne afin de savoir dans lequel élément XML TEI la ligne devrait être balisée, selon notre modélisation expliquée dans le chapitre 9.

```
# ------------------------------------------------------------
# Code by: Kelly Christensen
# Python script to build the <body> of a TEI file with an ALTO File's
    MainZone text.
# ------------------------------------------------------------

from lxml import etree


def body(root, data):
    text = etree.SubElement(root, "text")
    body = etree.SubElement(text, "body")
    div = etree.SubElement(body, "div")
    for line in data:
        # prepare attributes for the text block's zone
        zone_atts = {"corresp":f"#{line.zone_id}", "type":line.zone_type}
        # prepare <lb/> with this line's xml:id as @corresp
        lb = etree.Element("lb", corresp=f"#{line.id}")
        lb.tail = f"{line.text}"

        # if this is the page's first line, create a <pb> with the page's
    xml:id
        if int(line.n) == 1:
            pb = etree.Element("pb", corresp=f"#{line.page_id}")
            div.append(pb)

        # find the last element added to the div
        last_element = div[-1]

        # NumberingZone, QuireMarksZone, and RunningTitleZone line
        if line.zone_type == "NumberingZone" or line.zone_type == "
    QuireMarksZone" or line.zone_type == "RunningTitleZone":
            # enclose any page number, quire marks, or running title inside
     a <fw>
            fw = etree.Element("fw", zone_atts)
            last_element.addnext(fw)
            fw.append(lb)
```

```
34
35          # MarginTextZone line
36          elif line.zone_type == "MarginTextZone":
37              # create a <note> if one is not already the preceding sibling
38              if last_element.tag != "note":
39                  note = etree.Element("note", zone_atts)
40                  last_element.addnext(note)
41                  note.append(lb)
42              else:
43                  last_element.append(lb)
44
45          # MainZone line
46          elif line.zone_type[:4] == "Main":
47              # create an <ab> if one is not already the preceding sibling
48              if last_element.tag != "ab":
49                  ab = etree.Element("ab", zone_atts)
50                  last_element.addnext(ab)
51                  # update the last element in div
52                  last_element = div[-1]
53
54              # if the line is emphasized for being
55              if line.line_type == "DropCapitalLine" or line.line_type == "
    HeadingLine":
56                  # check if there is already an emphasized line in this
    MainZone
57                  ab_children = last_element.getchildren()
58                  if len(ab_children) == 0 or ab_children[-1].tag != "hi" or
    ab_children[-1].get("rend") != line.line_type:
59                      hi = etree.Element("hi", rend=line.line_type)
60                      last_element.append(hi)
61                      hi.append(lb)
62                  elif ab_children[-1].tag == "hi":
63                      ab_children[-1].append(lb)
64
65              # if the line is not emphasized, append it to the last element
    in the <ab>
66              elif line.line_type[:7] == "Default":
67                  last_element.append(lb)
```

A.7 Exporter la ressource numérique

Dans la fonction `main()` du module `./src/__main__.py`, qui dirige la création de la ressource TEI, la dernière étape est l'export du document, voir lignes 86-88. La fonction appelle la classe `Write` et sa méthode `write()` sur la ligne 88 du module. L'instance de la classe `Write` pour le document source en cours de traitement a besoin de deux arguments :

l'ARK du document (`d.doc_name`) et la racine TEI de la ressource construite. En les prenant comme ses deux attributs, la classe `Write` du module `./src/write_output.py` écrit un fichier TEI avec l'arborescence ainsi créée.

A.7.1 ./src/write_output.py

```
# ------------------------------------------------------------
# Code by: Kelly Christensen
# Python class to generate the output XML-TEI file.
# ------------------------------------------------------------

from lxml import etree

class Write:
    def __init__(self, document, root):
        self.d = document
        self.r = root

    def write(self):
        with open(f'./data/{self.d}.xml', 'wb') as f:
                etree.ElementTree(self.r).write(f, encoding="utf-8",
   xml_declaration=True, pretty_print=True)
```

Bibliographie

16 Linking, Segmentation, and Alignment - The TEI Guidelines. URL : https://tei-c.org/release/doc/tei-p5-doc/en/html/SA.html#SASOstdf (visité le 25/08/2022).

About – TEI : Text Encoding Initiative. URL : https://tei-c.org/about/ (visité le 25/08/2022).

Alliance, The ARK. *Community*. ARK Alliance. URL : https://arks.org/community/ (visité le 23/08/2022).

Bartz, Alexandre, Juliette Janes et Simon Gabay. *Annotator*. E-ditiones. URL : https://github.com/e-ditiones/Annotator (visité le 07/09/2022).

Bartz, Alexandre et al. « Expanding the Content Model of annotationBlock ». In : *Next Gen TEI, 2021 - TEI Conference and Members' Meeting*. Virtual, United States, oct. 2021. URL : https://hal.archives-ouvertes.fr/hal-03380805 (visité le 07/09/2022).

Bawden, Rachel. *ModFr-Normalisation*. URL : https://github.com/rbawden/ModFr-Norm (visité le 07/09/2022).

Bawden, Rachel et al. « Automatic Normalisation of Early Modern French ». In : LREC 2022 - 13th Language Resources and Evaluation Conference. 20 juin 2022. DOI : 10.5281/zenodo.5865428. URL : https://hal.inria.fr/hal-03540226 (visité le 11/08/2022).

Berchmans, Deepa et S S Kumar. « Optical Character Recognition : An Overview and an Insight ». In : *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. 2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT). Juill. 2014, p. 1361-1365. DOI : 10.1109/ICCICCT.2014.6993174.

Bibliothèque nationale de France. *Rapport d'activité 2021 de la Bibliothèque nationale de France*. Paris, France, 1er juill. 2022. URL : https://www.bnf.fr/fr/bnf-rapport-dactivite-2021 (visité le 09/08/2022).

Bobichon, Philippe. « Le Lexicon : Mise En Page et Mise En Texte Des Manuscrits Hébreux, Grecs, Latins, Romans et Arabes ». In : (2009), p. 81. URL : https://cel.archives-ouvertes.fr/cel-00377671 (visité le 25/07/2022).

Bobichon, Philippe. *Caviarder*. In : *Codicologia*. Institut de recherche et d'histoire des textes, 2011. URL : http://codicologia.irht.cnrs.fr/theme/liste_theme/413#tr-868 (visité le 25/07/2022).

Camps, Jean-Baptiste et al. « Corpus and Models for Lemmatisation and POS-tagging of Classical French Theatre ». In : *Journal of Data Mining & Digital Humanities* 2021 (Digital humanities in... 14 fév. 2021), p. 6485. ISSN : 2416-5999. DOI : 10.46298/jdmdh.6485. arXiv : 2005.07505 [cs]. URL : http://arxiv.org/abs/2005.07505 (visité le 09/08/2022).

Carlin, Marie et Arnaud Laborderie. « Le BnF DataLab, Un Service Aux Chercheurs En Humanités Numériques ». In : *Humanités numériques* 4 (déc. 2021). URL : https://hal-bnf.archives-ouvertes.fr/hal-03285816 (visité le 11/08/2022).

Caron, Bertrand. *Formats de Données Pour La Préservation à Long Terme : La Politique de La BnF*. Technical Report. Bibliothèque Nationale de France (Paris), oct. 2021. URL : https://hal-bnf.archives-ouvertes.fr/hal-03374030 (visité le 23/08/2022).

Chagué, Alix. *LECTAUREP Contemporary French Model (Administration)*. Zenodo, 12 mai 2022. URL : https://zenodo.org/record/6542744 (visité le 12/08/2022).

Chagué, Alix et Thibault Clérice. « Sharing HTR Datasets with Standardized Metadata : The HTR-United Initiative ». In : Documents Anciens et Reconnaissance Automatique Des Écritures Manuscrites. 23 juin 2022. URL : https://hal.inria.fr/hal-03703989 (visité le 12/08/2022).

Chagué, Alix, Thibault Clérice et Laurent Romary. « HTR-United : Mutualisons La Vérité de Terrain ! » In : *DHNord2021 - Publier, Partager, Réutiliser Les Données de La Recherche : Les Data Papers et Leurs Enjeux*. Lille, France : MESHS, nov. 2021. URL : https://hal.archives-ouvertes.fr/hal-03398740 (visité le 10/08/2022).

Chagué, Alix et al. *HTR-United/Htr-United : V0.1.28*. Zenodo, 10 août 2022. DOI : 10.5281/zenodo.6979746. URL : https://zenodo.org/record/6979746 (visité le 12/08/2022).

Christensen, Kelly. *Gallicorpora/Application*. URL : https://github.com/Gallicorpora/application (visité le 21/08/2022).

Christensen, Kelly, Simon Gabay et Ariane Pinche. *Alto 2 Tei*. URL : https://github.com/kat-kel/alto2tei (visité le 14/09/2022).

Christensen, Kelly, Ariane Pinche et Simon Gabay. « Gallic(Orpor)a : Traitment Des Sources Textuelles En Diachronie Longue de Gallica ». In : *DataLab de La BnF*. Paris, France, juin 2022. URL : https://hal.archives-ouvertes.fr/hal-03716534 (visité le 09/08/2022).

Clérice, Thibault. *Pie Extended, an Extension for Pie with Pre-Processing and Post-Processing*. Zenodo, juin 2020. DOI : 10.5281/zenodo.6534764. URL : https://zenodo.org/record/6534764 (visité le 07/09/2022).

— *YALTAi : Segmonto Manuscript and Early Printed Book Dataset*. 10 juill. 2022. DOI : 10.5281/zenodo.6814770. URL : https://zenodo.org/record/6814770 (visité le 12/08/2022).

COQUENET, Denis, Clément CHATELAIN et Thierry PAQUET. « Handwritten Text Recognition : From Isolated Text Lines to Whole Documents ». In : *ORASIS 2021*. Saint Ferréol, France : Centre National de la Recherche Scientifique [CNRS], sept. 2021. URL : https://hal.archives-ouvertes.fr/hal-03339648 (visité le 04/08/2022).

DigiPal : Digital Resource and Database of Palaeography, Manuscripts and Diplomatic. London. URL : http://www.digipal.eu/.

Eeditiones/Tei-Publisher-App. e-editiones.org. URL : https://github.com/eeditiones/tei-publisher-app (visité le 11/09/2022).

GABAY, Simon. *E-Ditiones, 17th c. French Sources*. Nov. 2018. URL : https://hal.archives-ouvertes.fr/hal-02388415 (visité le 10/08/2022).

— *FreEM-corpora/FreEMnorm : FreEM Norm Parallel Corpus*. Zenodo, 17 jan. 2022. DOI : 10.5281/zenodo.5865428. URL : https://zenodo.org/record/5865428 (visité le 11/08/2022).

GABAY, Simon, Jean-Baptiste CAMPS et Ariane PINCHE. « SegmOnto ». In : *Création de Modèle(s) HTR Pour Les Documents Médiévaux En Ancien Français et Moyen Français Entre Le Xe-XIVe Siècle*. Paris, France : Ecole nationale des chartes | PSL, nov. 2021. URL : https://hal.archives-ouvertes.fr/hal-03481089 (visité le 25/07/2022).

GABAY, Simon, Thibault CLÉRICE et Christian REUL. *OCR17 : Ground Truth and Models for 17th c. French Prints (and Hopefully More)*. Mai 2020. URL : https://hal.archives-ouvertes.fr/hal-02577236 (visité le 12/08/2022).

GABAY, Simon et al. « Standardizing Linguistic Data : Method and Tools for Annotating (Pre-Orthographic) French ». In : *Proceedings of the 2nd International Digital Tools & Uses Congress (DTUC '20)*. Hammamet, Tunisia, oct. 2020. DOI : 10.1145/3423603.3423996. URL : https://hal.archives-ouvertes.fr/hal-03018381 (visité le 12/08/2022).

GABAY, Simon et al. « Automating Artl@s – Extracting Data from Exhibition Catalogues ». In : *EADH 2021 - Second International Conference of the European Association for Digital Humanities*. Krasnoyarsk, Russia, sept. 2021. URL : https://hal.archives-ouvertes.fr/hal-03331838 (visité le 23/08/2022).

GABAY, Simon et al. « SegmOnto : Common Vocabulary and Practices for Analysing the Layout of Manuscripts (and More) ». In : *1st International Workshop on Computational Paleography (IWCP@ICDAR 2021)*. Lausanne, Switzerland, sept. 2021. URL : https://hal.archives-ouvertes.fr/hal-03336528 (visité le 09/08/2022).

GACEK, Adam. *The Arabic Manuscript Tradition : A Glossary of Technical Terms and Bibliography*. Handbook of Oriental Studies 1, The Near and Middle East. Leiden : Brill, 2001. 269 p. ISBN : ISBN 90-04-12061-0.

GAUTIER, Dassonneville et al. *Compte-Rendu de La Journée d'étude " Point HTR 2022 "Transkribus / eScriptorium : Transcrire, Annoter et Éditernumériquement Des Documents d'archives*. Research Report. CAPHES - UMS 3610 CNRS/ENS ; AOROC, juin 2022. URL : https://hal.archives-ouvertes.fr/hal-03692413.

« Glossaire codicologique français-arabe ». In : *Gazette du livre médiéval* 40.1 (2002), p. 79-80. URL : https://www.persee.fr/doc/galim_0753-5015_2002_num_40_1_1563 (visité le 25/07/2022).

GOODRICH, Gregory. « Kurzweil Reading Machine : A Partial Evaluation of Its Optical Character Recognition Error Rate ». In : *Journal of Visual Impairment and Blindness* (12 jan. 1979).

GOVINDAN, V. K. et A. P. SHIVAPRASAD. « Character Recognition — A Review ». In : *Pattern Recognition* 23.7 (1990), p. 671. ISSN : 0031-3203. URL : https://www.academia.edu/6986960/Character_recognition_A_review (visité le 05/08/2022).

GUIBON, Gaël et al. « Parsing Poorly Standardized Language Dependency on Old French ». In : *Thirteenth International Workshop on Treebanks and Linguistic Theories (TLT13)*. Sous la dir. de V. HENRICH et al. Proceedings of the Thirteenth International Workshop on Treebanks and Linguistic Theories (TLT13). Tübingen, Germany, déc. 2014, p. 51-61. URL : https://hal.archives-ouvertes.fr/hal-01250959 (visité le 10/08/2022).

IMPEDOVO, Sebastiano. « More than Twenty Years of Advancements on Frontiers in Handwriting Recognition ». In : *Pattern Recognition*. Handwriting Recognition and Other PR Applications 47.3 (1er mars 2014), p. 916-928. ISSN : 0031-3203. DOI : 10.1016/j.patcog.2013.05.027. URL : https://www.sciencedirect.com/science/article/pii/S0031320313002513 (visité le 29/08/2022).

JENTSCH, Patrick et Stephan PORADA. « From Text to Data Digitization, Text Analysis and Corpus Linguistics ». In : *Digital Methods in the Humanities : Challenges, Ideas, Perspectives*. Sous la dir. de Silke SCHWANDT. Bielefeld University Press, 2021.

KIESSLING, Benjamin. « Kraken - an Universal Text Recognizer for the Humanities ». In : ADHO 2019 - Utrecht. 2019. URL : https://dh-abstracts.library.cmu.edu/works/9912 (visité le 10/08/2022).

LASSNER, David et al. « Publishing an OCR Ground Truth Data Set for Reuse in an Unclear Copyright Setting. Two Case Studies with Legal and Technical Solutions to Enable a Collective OCR Ground Truth Data Set Effort ». Version 1.0. In : *Fabrikation von Erkenntnis – Experimente in den Digital Humanities. Hg. von Manuel Burghardt* Lisa Dieckmann (2021). Avec la coll. d'Herzog August BIBLIOTHEK, 5). DOI : 10.17175/SB005_006. URL : https://zfdg.de/sb005_006 (visité le 10/08/2022).

MANIACI, Marilena. *Terminologia Des Libro Manoscritto*. Addenda 3. Rome : Istituto centrale per la patologia del libro, 1996.

Manuel UNIMARC : format bibliographique. Transition bibliographique - Programme national. URL : https://www.transition-bibliographique.fr/unimarc/manuel-unimarc-format-bibliographique/ (visité le 27/08/2022).

MARTIN, Louis et al. « CamemBERT : A Tasty French Language Model ». In : *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. ACL 2020. Online : Association for Computational Linguistics, juill. 2020, p. 7203-7219. DOI : 10.18653/v1/2020.acl-main.645. URL : https://aclanthology.org/2020.acl-main.645 (visité le 11/08/2022).

METS : Metadata Encoding and Transmission Standard. BnF - Site institutionnel. URL : https://www.bnf.fr/fr/mets-metadata-encoding-and-transmission-standard (visité le 23/08/2022).

MISA, Thomas J. *Gender Codes : Why Women Are Leaving Computing*. John Wiley & Sons, 14 sept. 2011. 326 p. ISBN : 978-1-118-03513-9. Google Books : EjDYh_KHls8C.

Models - Hugging Face. URL : https://huggingface.co/models (visité le 12/09/2022).

MUEHLBERGER, Guenter et al. « Transforming Scholarship in the Archives through Handwritten Text Recognition : Transkribus as a Case Study ». In : *Journal of Documentation* 75.5 (9 sept. 2019), p. 954-976. ISSN : 0022-0418. DOI : 10.1108/JD-07-2018-0114. URL : https://www.emerald.com/insight/content/doi/10.1108/JD-07-2018-0114/full/html (visité le 04/08/2022).

MUZERELLE, Denis. *Caviarder*. In : *Codicologia*. Institut de recherche et d'histoire des textes, 2011. URL : http://codicologia.irht.cnrs.fr/theme/liste_theme/413#tr-868 (visité le 25/07/2022).

MUZERELLE, Dennis. *Vocabulaire Codicologique : Répertoire Méthodique Des Termes Français Relatifs Aux Manuscrits*. Rubricae 1. Paris : Éd. Cemi, 1985.

Numpy : NumPy Is the Fundamental Package for Array Computing with Python. Version 1.23.1. URL : https://www.numpy.org (visité le 04/08/2022).

ORTIZ SUÁREZ, Pedro Javier, Benoît SAGOT et Laurent ROMARY. « Asynchronous Pipeline for Processing Huge Corpora on Medium to Low Resource Infrastructures ». In : *7th Workshop on the Challenges in the Management of Large Corpora (CMLC-7)*. Sous la dir. de Piotr BAŃSKI et al. Cardiff, United Kingdom : Leibniz-Institut für Deutsche Sprache, juill. 2019. DOI : 10.14618/IDS-PUB-9021. URL : https://hal.inria.fr/hal-02148693 (visité le 07/09/2022).

OSTOS, Pilar, M. Luisa PARDO et Elena E. RODRÍGUEZ. *Vocabulario de codicologÍa : Versión Española Revisada y Aumentada Del Vocabulaire Codicologique*. Instrumenta Bibliologica. Madrid : Arco/Libros, 1997.

OTSU, Nobuyuki. « A Threshold Selection Method from Gray-Level Histograms ». In : *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (jan. 1979), p. 62-66.

Pellet, Aurélien et Marie Puren. « Le Projet AGODA. Océrisation Des Débats Parlementaires Français de La Troisième République : Problèmes, Défis et Perspectives ». In : Séminaire OMNSH-Epitech : Le Numérique Au Service Des Sciences Humaines et Sociales. Le Kremlin-Bicêtre, France, avr. 2022. URL : https://hal.archives-ouvertes.fr/hal-03651146 (visité le 29/08/2022).

Pinche, Ariane. « HTR model Cremma Medieval ». In : (21 juin 2022). DOI : 10.5281/zenodo.6669508. URL : https://zenodo.org/record/6669508 (visité le 10/08/2022).

Pinche, Ariane et Jean-Baptiste Camps. « CremmaLab Project : Transcription Guidelines and HTR Models for French Medieval Manuscripts ». In : *Colloque "Documents Anciens et Reconnaissance Automatique Des Écritures Manuscrites"*. Paris, France, juin 2022. URL : https://hal.archives-ouvertes.fr/hal-03716526 (visité le 10/08/2022).

Pinche, Ariane et Thibault Clérice. *HTR-United/Cremma-Medieval : Cortado 2.0.0*. Zenodo, 11 juill. 2022. DOI : 10.5281/zenodo.6818057. URL : https://zenodo.org/record/6818057 (visité le 12/08/2022).

Project, CatCor. *Letter 02633 : To Frederick II (the Great), 21 July 1744*. Sous la dir. d'Andrew Kahn et Kelsey Rubin-Detlev. 2021. URL : https://catcor.seh.ox.ac.uk/id/letter-02633.

Regnault, Mathilde, Sophie Prévost et Éric Villemonte de La Clergerie. « Challenges of Language Change and Variation : Towards an Extended Treebank of Medieval French ». In : *TLT 2019 - 18th International Workshop on Treebanks and Linguistic Theories*. Paris, France, août 2019. URL : https://hal.inria.fr/hal-02272560 (visité le 10/08/2022).

Reitz, Kenneth. *Requests : Python HTTP for Humans.* Version 2.28.1. URL : https://requests.readthedocs.io (visité le 14/09/2022).

Salah, Ahmed Ben, Nicolas Ragot et Thierry Paquet. « Adaptive Detection of Missed Text Areas in OCR Outputs : Application to the Automatic Assessment of OCR Quality in Mass Digitization Projects ». In : Document Recognition and Retrieval XX. T. 8658. 5 fév. 2013, p. 110. DOI : 10.1117/12.2003733. URL : https://hal-bnf.archives-ouvertes.fr/hal-00820564 (visité le 12/09/2022).

Scheithauer, Hugo, Alix Chagué et Laurent Romary. « From eScriptorium to TEI Publisher ». In : *Brace Your Digital Scholarly Edition !* Berlin, France, nov. 2021. URL : https://hal.inria.fr/hal-03538115 (visité le 23/08/2022).

Schneider, Ben R. « The Production of Machine-Readable Text : Some of the Variables ». In : *Computers and the Humanities* 6.1 (sept. 1971), p. 39-47. ISSN : 0010-4817, 1572-8412. DOI : 10.1007/BF02402324. URL : http://link.springer.com/10.1007/BF02402324 (visité le 02/08/2022).

Siddiqi, Imran-Ahmed, Florence Cloppet et Nicole Vincent. « Writing Property Descriptors : A Proposal for Typological Groupings ». In : *Gazette du livre médiéval* 56.1 (2011), p. 42-57. doi : 10.3406/galim.2011.1981. url : https://www.persee.fr/doc/galim_0753-5015_2011_num_56_1_1981 (visité le 02/08/2022).

SRU : Search/Retrieval via URL – SRU, CQL and ZeeRex (Standards, Library of Congress). url : https://www.loc.gov/standards/sru/ (visité le 02/09/2022).

Stehno, Birgit, Alexander Egger et Gregor Retti. « METAe–Automated Encoding of Digitized Texts ». In : *Literary and Linguistic Computing* 18.1 (1er avr. 2003), p. 77-88. issn : 0268-1145, 1477-4615. doi : 10.1093/llc/18.1.77. url : https://academic.oup.com/dsh/article-lookup/doi/10.1093/llc/18.1.77 (visité le 24/08/2022).

Stein, Achim et Sophie Prévost. *Syntactic Annotation of Medieval Texts : The Syntactic Reference Corpus of Medieval French (SRCMF)*. Narr Verlag, 2013, p. 275. isbn : 978-3-8233-6760-4. url : https://halshs.archives-ouvertes.fr/halshs-01122079 (visité le 10/08/2022).

— *Syntactic Reference Corpus of Medieval French (SRCMF)*. Stuttgart : ILR University of Stuttgart, 2013. isbn : 899-492-963-833-3. url : http://srcmf.org.

team, lxml dev. *Lxml : Powerful and Pythonic XML Processing Library Combining Libxml2/Libxslt with the ElementTree API.* Version 4.9.1. url : https://lxml.de/ (visité le 14/09/2022).

TEI element facsimile. url : https://tei-c.org/release/doc/tei-p5-doc/en/html/ref-facsimile.html (visité le 03/09/2022).

TEI element sourceDoc. url : https://tei-c.org/release/doc/tei-p5-doc/en/html/ref-sourceDoc.html (visité le 23/08/2022).

TEI : Text Encoding Initiative. url : https://tei-c.org/ (visité le 24/08/2022).

Tfibel, Florence. *RE : Tr : [Gallic(Orpor)a] Question Sur l'Unimarc Du Catalogue Généra*. E-mail. 13 juill. 2022.

The Bodelian First Folio : Digital Facsimile of the First Folio of Shakespeare's Plays. url : http://firstfolio.bodleian.ox.ac.uk/ (visité le 27/08/2022).

Truan, Naomi et Laurent Romary. « Building, Encoding, and Annotating a Corpus of Parliamentary Debates in TEI XML : A Cross-Linguistic Account ». In : *Journal of the Text Encoding Initiative* Issue 14 (Issue 14 17 mars 2021). issn : 2162-5603. doi : 10.4000/jtei.4164. url : https://journals.openedition.org/jtei/4164#tocto2n4 (visité le 26/08/2022).

Turing, A. M. « Computing Machinery and Intelligence ». In : *Mind* LIX.236 (1er oct. 1950), p. 433-460. issn : 0026-4423. doi : 10.1093/mind/LIX.236.433. url : https://doi.org/10.1093/mind/LIX.236.433 (visité le 04/08/2022).

Voltaire. *Candide, ou L'optimisme , traduit de l'allemand de M. le docteur Ralph.* S.l. : s.n., 1759. 299 p. URL : http://gallica.bnf.fr/ark:/12148/bpt6k70445g (visité le 12/09/2022).

Zarri, Gian Piero. « Quelques aspects techniques de l'exploitation informatique des documents textuels : saisie des données et problèmes de sortie ». In : *Publications de l'École Française de Rome* 31.1 (1977), p. 399-413. URL : https://www.persee.fr/doc/efr_0000-0000_1977_act_31_1_2286 (visité le 01/08/2022).

ref

Glossaire

CREMMALab Consortium pour la reconnaissance d'écriture manuscrite des matériaux anciens. 9

Handwritten Text Recognition La reconnaissance d'écriture manuscrite sur une image numérique. 19

HTR-United Un catalogue et un écosystème pour l'échange et la recherche des vérités de terrain pour la reconnaissance du texte. 7, 9

IIIF Image API Un service de web qui renvoie une image suite à une requête standardisée HTTP(S). La requête peut préciser la région, la taille, la rotation, la qualité, les caractéristiques, et le format de l'image demandée.. 39–41, 44, 80, 96, 124, 125

Inria Institut national de recherche en sciences et technologies du numérique. iii, 3–5, 9

International Image Interoperability Framework Normes internationales de l'exploitation des images numériques et de leurs métadonnées par API. 39

lxml Librairie python pour manipuler des données XML.. 119

One Document Does it all Un fichier XML TEI qui précise les règles d'un schéma TEI personnalisé.. 46, 182

Optical Character Recognition La reconnaissance du texte sur une image numérique. v, 19

requests Librairie python pour requêtes des APIs.. 119

UNIMARC Une référence pour l'échange de données en format XML. 62–64, 66, 69, 74, 75, 77–79, 81, 82, 84, 86, 89, 90, 117, 118, 128, 176, 183

Acronymes

ALMAnaCH Automatic Language Modelling and Analysis & Computational Humanities. iii, 4, 5, 8

ALTO Analyzed Layout and Text Object. i, 11, 12, 41–45, 49–51, 54, 56, 57, 64, 65, 73, 75, 76, 80, 82, 85, 90, 93–109, 117–119, 123, 124, 145, 149, 154, 157, 158, 176, 182

API Application Programming Interface. 38–40, 45, 73–81, 98, 117, 119, 126, 127, 176

ARK Archival Resource Key. 37, 38, 40, 41, 45, 68, 74–78, 96, 98, 114, 123–126, 128, 145, 161, 176, 177, 182

BnF Bibliothèque nationale de France. i, v, vi, 3–5, 38–42, 45, 62–66, 68, 69, 73–82, 86–89, 96, 117, 124, 126, 128

CREMMA Consortium Reconnaissance d'Écriture Manuscrite des Matériaux Anciens. 9

CSV Comma Separated Values. 114

DTS Distributed Text Services. 37, 47

ENC École nationale des chartes. iii, 5

HTML HyperText Markup Language. 44, 79, 81

HTR Handwritten Text Recognition. i, v, vi, 3, 5, 7–9, 11–13, 19, 21–23, 25, 27, 28, 30–32, 37, 40–46, 49, 50, 52, 57, 64, 68, 70, 73–75, 80, 82, 90, 93, 94, 96–99, 102, 109, 110, 113, 118, 119, 145

IIIF International Image Interoperability Framework. 37–39, 41, 45, 46, 69, 74–78, 80–82, 84, 86–90, 96, 98, 124, 126–128, 176, 182, 183

Inria Institut national de recherche en sciences et technologies du numérique. 4

JSON JavaScript Object Notation. 40, 44, 74, 81, 82, 84

METS Metadata Encoding and Transmission Standard. 49, 50

OCR Optical Character Recognition. i, v, 19–23, 25, 28–30, 42, 49, 52, 57

ODD One Document Does it all. 46, 182

RDF Resource Description Framework. 37, 47

SRU Search/Retrieve via URL. 45, 74, 75, 77–79, 81, 117, 126

SUDOC Système Universitaire de Documentation. 45, 74, 75, 78–81, 89, 124, 176, 183

TAL Traitement automatique des langues. 3–5, 10–12, 32, 40, 46, 56, 109, 112–114, 177

TEI Text Encoding Initiative. i, 11–14, 37, 44–47, 49, 53, 54, 56, 57, 59–66, 68, 69, 73, 74, 76, 78, 80–82, 84, 86, 89, 90, 93–111, 115, 117–120, 123, 124, 145, 154, 157, 159–161, 176, 177, 184

TSV Tab Separated Values. 114

XML eXtensible Markup Language. vii, 11, 41–47, 49–51, 53, 54, 62, 64, 65, 73, 74, 76, 77, 79, 81, 82, 84, 85, 90, 93–97, 100–103, 114, 118, 119, 123, 124, 128, 145, 149, 154, 157–159

YAML Yet Another Markup Language. 74, 79–81

Table des figures

Liste des tableaux

Table des matières