Université du Québec
à Rimouski

Vers une usine de logiciels pour le développement d'applications mobiles pour un tourisme intelligent

Mémoire présenté

dans le cadre du programme de maîtrise en informatique

en vue de l'obtention du grade de maître ès sciences

PAR

© MAMADOU MBOW

Juin 2024

Composition du jury :

Mohamed Tarik Moutacalli, président du jury, Université du Québec à Rimouski

Ismaïl Khriss, directeur de recherche, Université du Québec à Rimouski

Hafedh Mili, examinateur externe, Université du Québec à Montréal

Hamid Mcheick, professeur, Université du Québec à Chicoutimi

Dépôt initial le 29 Février 2024 Dépôt final le 06 Juin 2024

À mes défunts parents.

REMERCIEMENTS

Tout d'abord je remercie Dieu, le Tout-Puissant, pour m'avoir donné la force et le courage d'accomplir ce travail.

Ma profonde reconnaissance va à Monsieur Ismaïl Khriss, mon directeur de recherche, pour sa disponibilité constante, sa collaboration incessante, et ses conseils judicieux qui ont été essentiels à la réalisation de ce travail de recherche. Mes remerciements s'étendent également aux membres du jury qui ont accepté de juger ce travail, pour leur lecture attentive de ce mémoire ainsi que pour les remarques précieuses qu'ils m'ont adressées dans le but d'en améliorer le contenu.

Je tiens à exprimer ma gratitude envers mes collègues qui ont contribué à la validation de ce travail, ainsi qu'à tous les participants qui ont pris part à la collecte de données audio. Je souhaite exprimer ma reconnaissance envers toute ma famille pour leur soutien indéfectible. Vous avez toujours été ma force et ma principale source de motivation dans les moments les plus difficiles. Enfin, j'adresse mes plus sincères remerciements à tous mes proches, mes amis, qui m'ont toujours soutenu au cours de ces années universitaires.

RÉSUMÉ

En appliquant le concept d'intelligence pour répondre aux besoins des voyageurs avant, pendant et après leur voyage, les destinations touristiques pourraient accroître leur compétitivité. Ceci demande le développement d'applications mobiles pour un tourisme intelligent offrant des fonctionnalités telles que les systèmes de recommandation pour suggérer des recommandations personnalisées aux touristes. Le développement de tels systèmes dans les applications mobiles intelligentes de tourisme exige un investissement substantiel en experts en génie logiciel et en intelligence artificielle. Afin de réduire ces coûts, notre proposition est la construction d'une usine de logiciels fournissant des actifs, tels que des langages spécifiques au domaine, pour accélérer le processus de développement. Dans ce mémoire, nous exposons une approche d'ingénierie dirigée par les modèles, utilisant des modèles, des métamodèles et des transformations de modèles comme mécanisme pour soutenir la conception et la mise en œuvre de ce genre de systèmes, tout en offrant une prise en charge de la variabilité des actifs de l'usine de logiciels. Les résultats du processus de validation ont démontré que notre approche a engendré une amélioration significative de la productivité par rapport à une méthode traditionnelle de développement. Notre approche vise également à faciliter la modélisation des systèmes de recommandation en explorant la détection de commandes vocales à partir d'un jeu de données de petite taille. Nous utilisons un modèle de reconnaissance automatique de la parole entraîné sur un vaste ensemble de données linguistiques pour la transcription d'enregistrements vocaux. Nous appliquons diverses approches, dont celles basées sur les règles et l'apprentissage profond, pour détecter l'intention dans le texte obtenu après transcription. Malgré la taille modeste de notre jeu de données, notre modèle réussit à classifier différentes commandes avec un score F1 de 96,7%, confirmant ainsi la validité de notre approche.

Mots-clés : Tourisme intelligent, Système de recommandation, Usine de logiciels, Ingénierie dirigée par les modèles, Langage spécifique au domaine, Transformation de modèle, Reconnaissance automatique de la parole, Classification de commandes vocales.

ABSTRACT

By applying the concept of intelligence to meet the needs of travellers before, during, and after their trip, tourist destinations could increase their competitiveness. Adopting this concept requires the development of mobile applications for intelligent tourism, offering features such as recommendation systems to suggest personalized recommendations to tourists. However, developing these systems in smart mobile tourism applications requires a significant investment in software engineering and artificial intelligence experts. Therefore, to reduce this investment's cost, we propose constructing a software factory that offers a set of assets, such as domain-specific languages, to accelerate the development process. In this thesis, we present a model-driven engineering approach using models, meta-models, and model transformations to support the design and implementation of such systems while providing variability support for the software factory assets. The results showed that our approach significantly improved productivity compared to a classic development method. Our approach also aims to facilitate the modelling of recommendation systems by exploring voice command detection from a small dataset. We use a speech recognition model trained on a vast linguistic dataset for transcribing voice recordings. We apply various approaches, including rule-based and deep learning-based methods, to detect intent in the text obtained after transcription. Despite the modest size of our dataset, our model successfully classifies different commands with an F1 score of 96.7%, confirming the validity of our approach.

Keywords: Smart tourism, Recommender system, Software factory, Model-driven engineering, Domain-specific language, Model transformation, Automatic speech recognition, Speech command classification.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABRÉVIATIONS, DES SIGLES ET DES ACRONYMES

ADL	Analyse Discriminante Linéaire
ALBERT	A Lite BERT
ASR	Automatic Speech Recognition
ATL	Atlas Transformation Language
BERT	Bidirectional Encoder Representations for Transformers
BNF	Backus-Naur Form
CIM	Common Information Model
CNN	Convolutional Neural Networks
CTC	Connectionist Temporal Classification
DM	Decision Modeling
DMN	Decision Model and Notation
DNN	Deep Neural Network
DSL	Domain-Specific Languages
DSM	Domain-Specific Modeling
DTW	Dynamic Time Warping
DTI	Destinations Touristiques Intelligentes
EF	Entity Framework

E2E	End-to-End
EMOF	Essential MOF
FFN	Feed-Forward Network
FODA	Feature Oriented Domain Analysis
FORM	Feature-Oriented Reuse Methodology
FM	Feature Modeling
GELU	Gaussian Error Linear Unit
GMM	Gaussian Mixture Model
GLUE	General Language Understanding Evaluation
GPU	Graphics Processing Unit
GSCD	Google Speech Commands Dataset
GT	Generic Transformations
HLDA	Heteroscedastic Linear Discriminant Analysis
HMM	Hidden Markov Model
IDE	Integrated Development Environment
IDL	Interface Definition Language
IoT	Internet of Things
KNN	K-Nearest Neighbors
LAS	Listen, Attend and Spell
LCDP	Low-Code Development Platforms

LSD	Latent Sequence Decompositions
LSTM	Long Short-Term Memory
MDA	Model-Driven Architecture
MDE	Model-Driven Engineering
MFCC	Mel Frequency Cepstral Coefficients
MLLR	Maximum Likelihood Linear Regression
MLM	Modèle de Langage Masqué
MOF	Meta Object Facility
NLU	Natural Language Understanding
OMG	Object Management Group
OMT	Organisation Mondiale du Tourisme
OCL	Object Constraint Language
ORM	Object-Relational Mapping
PIM	Platform-Independent Model
PSM	Platform-Specific Model
POI	Points of Interest
PPP	Partenariat Public-Privé
QVT	Query View Transformation
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network

RoBERTa	Robustly Optimized BERT Pretraining Approach
RS	Recommender System
SC	Speech Commands
SIT	System-independent transformation
SLU	Spoken Language Understanding
SQuAD	Stanford Question Answer Dataset
SR	Système de Recommandation
SST	System-specific transformation
STD	Smart Tourism Destinations
SVM	Support Vector Machine
SVD	Singular Value Decomposition
T4	Text Template Transformation Toolkit
TIC	Technologies de l'Information et de la Communication
VP	Visual Paradigm
VTLN	Vocal Tract Length Normalization
WLAS	Watch, Listen, Attend and Spell
XLNet	eXtreme Language understanding Network
XML	eXtensible Markup Language
XLSR	Cross-lingual Speech Recognition

XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformations

INTRODUCTION GÉNÉRALE

MOTIVATION

Le tourisme est un domaine majeur pour les applications mobiles qui offrent aujourd'hui un nombre croissant de fonctionnalités pour améliorer l'expérience et la satisfaction du touriste avant, pendant et après le voyage. De plus, l'intelligence artificielle (IA) s'est largement répandue ces dernières années dans l'industrie du tourisme, et son adoption par les destinations touristiques a fait apparaître un nouveau concept : les destinations touristiques intelligentes (DTI ou STD pour Smart Tourism Destinations – Buhalis et Amaranggana (2013)). Ils suggèrent que les destinations réussies peuvent être structurées en « six A » (6A) des destinations touristiques :

1. Attractions, qu'elles soient naturelles comme les montagnes, artificielles comme les parcs d'attractions, ou culturels comme les festivals de film ou de musique.
2. Accessibilité, engloabant l'ensemble du système de transport de la destination, y compris les itinéraires disponibles et les transports publics.
3. Équipements (en anglais, Amenities) comprenant tous les services facilitant un séjour convenable : l'hébergement, la gastronomie et les activités de loisirs.
4. Forfaits disponibles (en anglais, Available Packages), intégrant les services des intermédiaires visant à attirer l'attention des touristes.
5. Activités, motivant les touristes à visiter la destination.
6. Services auxiliaires (en anglais, Ancillary services), regroupant les services d'usage quotidien tels que les banques, la poste, les services médicaux, etc.

C'est pourquoi, à travers le monde, des destinations touristiques dans le monde proposent des applications mobiles offrant des fonctionnalités correspondant aux 6As. Les

touristes devraient pouvoir choisir les points d'intérêt (POI pour Points of Interest en anglais) les plus adaptés à leurs préférences tout en tenant compte de différentes limitations telles que leur budget, le temps et la quantité volumineuse d'informations disponibles sur les POI. Afin de résoudre ce problème, l'un des outils et fonctionnalités les plus précieux que nous pouvons intégrer aux applications mobiles de tourisme intelligent sont les systèmes de recommandation (SR ou RS pour Recommender Systems – Burke et al. (2011)), car ils permettent de fournir des recommandations personnalisées guidant les utilisateurs vers des ressources intéressantes et utiles au sein d'un vaste espace de données (Alaoui et al., 2018). Les SR sont parmi les applications les plus réussies et les plus répandues des technologies d'apprentissage automatique en entreprise (Adomavicius & Tuzhilin, 2005; Falk, 2019). Ils sont généralement classés en différentes catégories, notamment les SR personnalisés, le filtrage collaboratif, le filtrage basé sur le contenu, le filtrage basé sur la connaissance et les SR hybrides. Cependant, le développement de ce type d'applications demande un investissement important en ressources humaines (ingénierie logicielle et expertise en IA), en temps et en efforts.

Généralement, ces applications reposent sur des approches traditionnelles (Abrahamsson et al., 2004; Charland & Leroux, 2011; Amatya, 2013) où la construction de l'interface utilisateur « front-end » et la gestion des données en arrière-plan « back-end » impliquent l'utilisation de différentes technologies spécifiques à chaque plateforme. Cependant, ces approches sont confrontées à des défis en raison de l'évolution rapide des technologies dans le domaine, obligeant les développeurs à constamment mettre à jour leurs compétences pour suivre ces évolutions.

Pour surmonter ces défis et automatiser les processus de développement, diverses approches ont été proposées, telles que l'automatisation des tests, la génération automatique de la documentation, l'utilisation de plateformes "Low-Code/No-Code", et les pratiques DevOps comme l'intégration continue et la livraison continue (CI/CD). C'est pourquoi l'ingénierie dirigée par les modèles (Model-Driven Engineering MDE – Bézivin (2006)) est devenue une alternative pour surmonter ces défis technologiques. Le MDE favorise le développement de modèles à différents niveaux d'abstraction, où les modèles de niveau

supérieur sont (automatiquement ou semi-automatiquement) transformés en modèles de niveau inférieur, puis finalement en code (Rožanc & Mernik, 2021).

Malgré ses avantages, le MDE présente également des désavantages importants. Hutchinson et al. (2011) ont remarqué qu'un effet de "paralysie par la modélisation" ou "sur-modélisation" pourrait avoir une influence négative sur la rentabilité de la modélisation. De plus, l'effort requis pour développer les transformations de modèles et les tester peut également augmenter le temps de développement et de test. Un autre obstacle, non négligeable, est le manque de formation des employés dans le domaine de la MDE, ce qui peut constituer un obstacle à son adoption à grande échelle dans l'industrie (Khriss et al., 2020).

Face à ces défis et aux désavantages du MDE, l'une des solutions possibles, qui permettra également de réduire le coût de l'investissement, est la construction d'une usine de logiciels (software factory en anglais) dédiée au développement de systèmes de recommandation pour les applications mobiles de tourisme intelligent.

Une usine de logiciels est une ligne de produits logiciels qui configure des outils, des processus et du contenu extensible à l'aide d'un modèle de l'usine basé sur un schéma d'usine de logiciels pour automatiser le développement et la maintenance des variantes d'une famille de produits (Greenfield et al., 2004).

L'objectif de cette approche est d'assurer la productivité, la qualité, l'efficacité et la maintenabilité du système logiciel final, tout en réduisant le temps et les efforts nécessaires à sa réalisation à l'aide d'un ensemble d'actifs. Pour atteindre cet objectif, nous devons préparer les actifs de l'usine, tels que les patrons de conception, les cadres d'application (frameworks en anglais), les langages spécifiques au domaine (Domain-Specific Languages DSL – Greenfield et al. (2004)) et les transformations de modèle (model transformations en anglais).

Cook (2004) souligne l'utilité des DSL dans la résolution de problèmes spécifiques dans un domaine donné. En effet, un DSL est un langage de programmation ou un langage de spécification exécutable qui, à travers des notations appropriées d'abstractions, offre un pouvoir expressif focalisé sur un domaine de problème particulier (Fowler, 2010).

La gestion de la variabilité demeure un défi crucial au sein d'une usine de logiciels, surtout dans le cadre de la modélisation et de la mise en œuvre des variations. Une solution à cet enjeu consiste à adopter une approche MDE pour soutenir cette variabilité. En particulier, nous choisirons d'appliquer la terminologie de MDA (Model Driven Architecture), l'une des propositions pratiques de MDE, standardisée par l'OMG (Object Management Group – OMG (2003a)). Ainsi, nous postulons qu'une fusion entre MDE et les concepts des usines de logiciels pourrait conduire à une approche efficace pour automatiser le processus de développement.

OBJECTIFS ET CONTRIBUTIONS

Dans le contexte dynamique du tourisme intelligent en constante évolution, diverses approches ont été explorées pour répondre à la demande croissante d'applications mobiles spécialisées. Notre mémoire s'inscrit dans cette dynamique, ayant pour objectif de proposer une approche pour le développement d'une usine de logiciels dédiée à la famille d'applications mobiles pour le tourisme intelligent. Plus précisément, notre contribution à travers ce projet de recherche consiste à concevoir un DSL en tant qu'actif de cette usine, permettant à un analyste de modéliser un SR pour ces applications et de générer le code source à partir du modèle créé grâce à des transformations.

Dans notre approche, nous cherchons à simplifier le travail de l'analyste en explorant les systèmes ASR (Automatic Speech Recognition) pour lui permettre de modéliser un SR à l'aide de commandes vocales. Notre objectif n'est pas de développer un système ASR complet, compte tenu des défis complexes liés à diverses sources de variabilité telles que le style de discours, l'environnement, les caractéristiques du locuteur et les contraintes spécifiques des tâches (Shreyaa, 2019). Au lieu de cela, nous visons à réutiliser les connaissances acquises par des systèmes ASR éprouvés dans la transformation de la parole en texte, comme wav2vec 2.0 (Baevski et al., 2020). Ce modèle, convertissant les signaux vocaux en texte, offre également des possibilités de personnalisation en fournissant une liste de mots possibles à reconnaître. L'autre contribution de ce travail consiste à réutiliser ces modèles ASR bien établis, entrainés sur des langues riches en ressources, pour transcrire en texte des enregistrements vocaux contenant des phrases en anglais. Cela est suivi de

l'application de différentes approches, y compris celles basées sur les règles et l'apprentissage profond, pour détecter l'intention à partir du texte obtenu après transcription.

Pour évaluer les avantages de notre approche, nous avons réalisé une étude comparative des temps de développement d'un ensemble de SR, en considérant deux scénarios : le développement traditionnel impliquant le codage manuel à partir d'un modèle donné, et la génération automatique du code à partir de ce même modèle. Six développeurs ont participé à cette étude, répartis au hasard entre les deux scénarios. Les résultats obtenus ont révélé une réduction significative des délais de développement grâce à l'utilisation des modèles de transformations. Pour valider notre modèle ASR, nous avons évalué le taux d'erreur de mots (Word Error Rate ou WER) et le taux d'erreur de caractères (Character Error Rate ou CER) sur les transcriptions d'enregistrements audio. En ce qui concerne la détection d'intention avec l'approche basée sur l'apprentissage profond, différents modèles ont été entraînés. Malgré la taille limitée de notre jeu de données, des résultats encourageants ont été obtenus avec un WER de 26,49%, un CER de 6,40%, et un score F1 moyen de 96,7% pour la détection de commandes grâce à l'apprentissage profond.

ORGANISATION DU MÉMOIRE

Le mémoire est structuré en quatre chapitres. Le premier aborde les bases des systèmes de recommandation dans le tourisme intelligent et des usines de logiciels. Le deuxième explore diverses approches d'automatisation du développement d'applications, mettant en avant notre choix d'une usine de logiciels pour les systèmes de recommandation. Ce chapitre inclut également une revue bibliographique sur les modèles d'ASR, les méthodes de classification des commandes vocales et la détection d'intention. Le troisième chapitre présente notre usine de logiciels à travers un exemple concret, détaillant les actifs et notre approche, ainsi qu'un modèle et un jeu de données pour la classification de commandes vocales. Enfin, le quatrième chapitre expose notre protocole de validation, les résultats obtenus et se conclut avec une discussion des conclusions tirées du processus de validation.

CHAPITRE 1
LES NOTIONS DE BASE

Le secteur touristique est reconnu comme l'un des plus importants à l'échelle mondiale, jouant un rôle crucial dans la croissance de l'économie de nombreux pays (Irmanti et al., 2017; Zhang et al., 2020). Dans ce chapitre, nous commencerons par présenter le concept de tourisme intelligent ainsi que ses enjeux. Nous explorerons ensuite le cadre 6A de Buhalis et Amaranggana (2013), qui servira de base à notre étude pour développer un modèle de domaine décrivant les différents aspects du secteur du tourisme. Ensuite, nous aborderons les systèmes de recommandation et les agents conversationnels, composants clés du tourisme intelligent, qui seront intégrés comme actifs dans notre usine de logiciels. L'objectif de notre étude est de développer des applications mobiles de tourisme intelligent, mettant particulièrement l'accent sur l'accélération du développement des systèmes de recommandation. Pour atteindre cet objectif, nous préconisons l'utilisation d'une usine de logiciels pour un développement efficace et rapide de ces applications. Dans ce chapitre, nous présenterons MDE, incluant MDA, ainsi que les langages de modélisation. Ces éléments nous permettront d'explorer en détail les DSL et les concepts liés aux transformations de modèles. Enfin, nous donnerons une vue d'ensemble des usines de logiciels et présenterons le processus recommandé pour leur construction.

1.1 LE TOURISME INTELLIGENT

1.1.1 Définition

Selon l'OMT[1] (Organisation Mondiale du Tourisme – 2015), le tourisme est un

[1] https://www.unwto.org/fr

phénomène social, culturel et économique impliquant le déplacement de personnes vers des pays ou des endroits en dehors de leur environnement habituel à des fins personnelles ou professionnelles. En raison de l'importance des informations dans le tourisme et de la forte dépendance aux technologies de l'information et de la communication (TIC) qui en résulte (Werthner & Klein, 1999; Benckendorff et al., 2014; Law et al., 2014; Koo et al., 2015) , le concept de « tourisme intelligent » est appliqué aux phénomènes qui englobent le tourisme (Gretzel et al., 2015). Le tourisme intelligent peut être considéré comme une évolution logique du tourisme traditionnel et, plus récemment, de l'e-tourisme. Les bases des innovations et de l'orientation technologique de l'industrie et des consommateurs ont été posées tôt avec l'adoption extensive des technologies de TIC dans le tourisme, comme les systèmes de distribution mondiale et de réservation centralisée, et l'intégration des technologies basées sur le Web qui ont conduit à l'émergence de l'e-tourisme (Buhalis, 2003; Werthner & Ricci, 2004). Cette trajectoire de développement s'est poursuivie avec l'adoption généralisée des médias sociaux (Sigala et al., 2012) et l'orientation vers la réalisation du tourisme mobile, reconnaissant la mobilité élevée des informations touristiques et des consommateurs de tourisme (Buhalis & Law, 2008; Wang et al., 2012; Gretzel et al., 2015).

1.1.2 Les Six A (6A) : Un cadre pour l'analyse des destinations touristiques intelligentes

Une destination touristique est formée d'une multitude de caractéristiques contribuant au succès d'un processus dynamique de co-création, renforçant ainsi la compétitivité de la destination dans le secteur touristique (Rajraji, 2022). Buhalis et Amaranggana (2013) ont introduit le cadre des « six A » (6A) pour structurer ces caractéristiques. Depuis lors, ce cadre a été adopté par d'autres travaux dans le domaine (Tran et al., 2017; Huertas et al., 2019). Buhalis et Amaranggana (2013) suggèrent que les destinations réussies peuvent être définies par les 6A des destinations touristiques : Attractions, Accessibilité, Amenities (équipements), Available Packages (forfaits disponibles), Activités et Ancillary services (services auxiliaires). Rajraji (2022) les décrit de la manière suivante :

1. Attractions : Les attractions d'une destination touristique sont des éléments captivants qui motivent les visiteurs, offrant des expériences passionnantes. Ces lieux, largement accessibles au public, servent à des fins de divertissement, d'éducation, ou comme points d'intérêt touristique.

2. Accessibilité : L'accessibilité a été identifiée comme un pilier fondamental des activités touristiques. Les installations touristiques doivent garantir une accessibilité optimale à leurs visiteurs, facilitant ainsi leurs déplacements vers et à l'intérieur de la région, y compris lors de l'utilisation des biens et services proposés, tant à leur arrivée que pendant leur séjour.

3. Les équipements (Amenities) : Les équipements englobent les installations disponibles dans les destinations touristiques. Ces facteurs influent directement sur la qualité des services offerts aux visiteurs par ces établissements. En général, ces installations se trouvent à la fois à l'intérieur et à l'extérieur de l'ensemble de la destination touristique. Elles contribuent à rehausser le niveau de confort des visiteurs de toute destination, englobant des éléments tels que les hôtels et l'hébergement, les restaurants, les installations publiques, les centres commerciaux, et bien d'autres.

4. Forfaits disponibles (Available Packages) : Dans cette composante particulière, les visiteurs peuvent découvrir une variété de forfaits touristiques soigneusement combinés dans le but de les séduire. Chacun de ces forfaits propose des offres uniques susceptibles d'attirer l'attention des visiteurs. Certains forfaits regroupent plusieurs attractions touristiques, proposant ainsi des tarifs spéciaux dans le cadre de voyages organisés. Ils peuvent inclure des services de guide, des visites organisées, ainsi que des excursions d'intérêt particulier.

5. Activités : Les activités touristiques englobent toutes les expériences que les visiteurs peuvent apprécier dans une destination touristique. Il existe une multitude d'activités qui attirent les touristes vers une destination de vacances spécifique. Une destination touristique peut offrir une diversité d'activités à ses visiteurs, comprenant des visites touristiques, la natation, des excursions, des jeux, et la photographie, pour n'en nommer que quelques-unes des options disponibles.

6. Services auxiliaires (Ancillary services) : La composante des services auxiliaires englobe les installations de soutien à l'intérieur et autour des destinations touristiques. Même si certaines de ces installations ne sont pas directement liées au tourisme, elles sont souvent indispensables pour certains visiteurs. L'évaluation des services annexes repose sur un ensemble de variables, comprenant notamment les canaux de communication, les services internet, les guichets automatiques bancaires, les services médicaux et les services postaux.

1.2 LES SYSTÈMES DE RECOMMANDATION

Les systèmes de recommandation (SR) constituent une réponse intuitive au problème du choix excessif des consommateurs. Avec la croissance explosive des informations disponibles sur Internet, les utilisateurs sont souvent confrontés à une multitude de produits et services. Dans ce contexte, la personnalisation devient une stratégie essentielle pour améliorer l'expérience utilisateur. Ces systèmes jouent un rôle vital et indispensable dans divers domaines d'accès à l'information, stimulant et facilitant le processus de prise de décision (Jannach et al., 2010; Ricci et al., 2015). Leur présence est omniprésente dans de nombreux secteurs du Web, tels que les sites de commerce électronique ou les plateformes médiatiques (Zhang et al., 2019).

1.2.1 Définition

En général, les SR aident directement les utilisateurs à découvrir du contenu, des produits ou des services en agrégeant et en analysant les suggestions d'autres utilisateurs, y compris des critiques provenant de différentes autorités et d'utilisateurs (Frias-Martinez et al., 2006; Frias-Martinez et al., 2009; Kim et al., 2010). Ces systèmes utilisent des méthodes d'analyse pour calculer la probabilité qu'un utilisateur achète l'un des produits ou services à chaque endroit, permettant ainsi aux utilisateurs de recevoir des recommandations pour les bons produits ou services à acheter (Park et al., 2012). Il est important de noter que cette définition qualifie les SR comme favorisant la collaboration entre utilisateurs. Certains chercheurs ont élargi la définition pour inclure les systèmes suggérant des éléments d'intérêt, quelle que soit la manière dont ces recommandations sont générées. Ainsi, un système de recommandation peut être défini comme tout système guidant un utilisateur de manière personnalisée vers des objets intéressants ou utiles dans un vaste espace d'options possibles ou produisant de tels objets en sortie (Burke et al., 2011).

D'après les réflexions de Burke et al. (2011), deux principes fondamentaux distinguent la recherche sur les SR :

- Un SR est personnalisé. Les recommandations qu'il génère visent à optimiser l'expérience d'un utilisateur, et non pas à représenter un consensus de groupe pour tous.

- Un SR est conçu pour assister l'utilisateur dans la sélection parmi des options discrètes. En général, les éléments sont déjà connus à l'avance et ne sont pas générés de manière personnalisée.

Burke et al. (2011) notent que l'aspect de la personnalisation distingue fortement la recherche sur les SR de ce qui est généralement compris comme recherche dans les moteurs de recherche et d'autres applications de recherche d'informations. Dans un moteur de recherche ou un autre système de recherche d'informations, on s'attend à ce que l'ensemble

des résultats pertinents pour une requête particulière reste le même, quel que soit l'émetteur de la requête. De nombreux SR atteignent la personnalisation en maintenant des profils d'activités d'utilisateurs (à long terme ou à court terme) ou des préférences déclarées (Schafer et al., 2007). D'autres parviennent à un résultat personnalisé grâce à l'interaction conversationnelle (McGinty & Reilly, 2010).

De plus, les SR nécessitent toujours les données suivantes :

- Profil utilisateur et paramètres contextuels : Le profil de l'utilisateur est une structure de données personnelles contenant les préférences pour un élément dans un SR. Les préférences sont souvent classées dans la catégorie des évaluations si un SR offre une interface pour évaluer les éléments. Les paramètres contextuels dépendent d'informations contextuelles sur l'utilisateur telles que l'emplacement, l'heure de la journée, les moyens de transport, les conditions météorologiques, la raison et le budget du voyage, les sentiments de l'utilisateur, l'environnement social, etc. Ces paramètres contextuels peuvent être recueillis de trois manières générales : explicitement, implicitement et par déduction.

- Données de la communauté : Les préférences des autres utilisateurs concernant les éléments évalués.

- Attributs d'item : Propriétés et caractéristiques décrivant les éléments par leurs métadonnées respectives, incluant un titre, des balises ou des mots-clés pertinents.

- Modèles de connaissances : Cela inclut des structures telles que l'ontologie, la taxonomie etc.

1.2.2 Les catégories de systèmes de recommandation

Un SR est un système de filtrage d'informations qui propose les éléments les plus pertinents à un utilisateur cible. Les SR sont généralement classés en cinq catégories (Owen et al., 2011), comme illustré dans la figure 1.1.

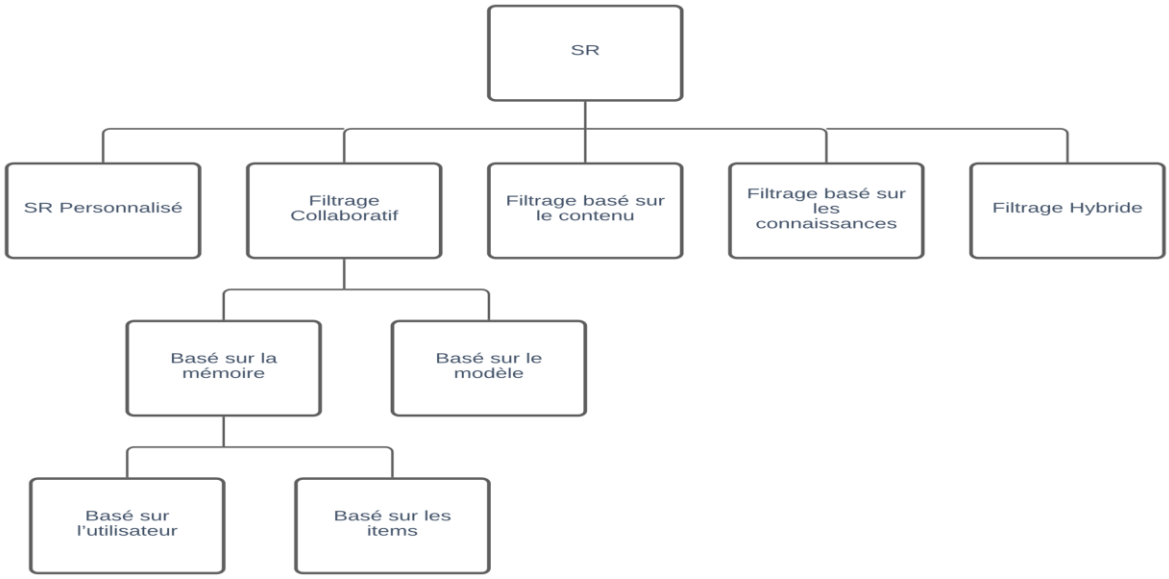


Figure 1.1 – Catégorisation des systèmes de recommandation (adapté d'Owen et al. (2011)).

1.2.2.1 SR personnalisé

Cette catégorie, basée sur le profil de l'utilisateur et des paramètres contextuels, est considérée comme la moins performante et donc la moins utilisée actuellement.

1.2.2.2 Le filtrage collaboratif

Un SR basé sur cette catégorie recommande des éléments similaires à ceux choisis par d'autres utilisateurs ayant des goûts similaires. Ainsi, ils répondent à la question suivante : « Dis-moi ce qui est populaire parmi mes pairs ? ». Les SR de ce type sont regroupés en deux sous-catégories comme suit (Owen et al., 2011) :

- Le filtrage collaboratif basé sur la mémoire : Il s'appuie sur les évaluations de l'utilisateur pour prédire les éléments à recommander. Ainsi, on distingue deux approches principales : l'une est basée sur l'utilisateur (user-based en anglais) et l'autre est basée sur l'item (item-based en anglais). Ces deux algorithmes

nécessitent quatre étapes principales : le calcul de la similarité, la prédiction des notes, la sélection des voisins et la recommandation Top-N.

Étape 1 : Calcul de la similarité

C'est une mesure qui aide à déterminer les utilisateurs ou les items les plus proches via différentes mesures de similarité indiquées ci-dessous :

- Mesure de Cosinus

  La mesure de Cosinus évalue le degré de similarité entre deux vecteurs X et Y en considérant le cosinus de l'angle entre eux dans un espace de produit interne. L'angle cosinus est donné par :

$$cos(X,Y) = \frac{(X \bullet Y)}{\|X\|\|Y\|} \quad (1)$$

  Ici • est le produit scalaire des vecteurs et $\|.\|$ désigne la norme d'un vecteur.

- Coefficient de corrélation Pearson

  Le coefficient de corrélation de Pearson est la mesure du degré de corrélation entre deux variables. La corrélation de Pearson entre X et Y peut être exprimée par la formule ci-dessous :

$$Pearson(\mathrm{X},\mathrm{Y}) = \frac{cov(\mathrm{X},\mathrm{Y})}{\sigma_X \times \sigma_\mathrm{Y}} \quad (2)$$

  Ici $cov()$ est la covariance des points de données et σ est l'écart type.

- Coefficient de corrélation Spearman

  Le coefficient de corrélation de Spearman est le coefficient de corrélation entre les rangs des valeurs prises par deux variables X et Y, et il est exprimé par la formule ci-dessous :

$$\rho(X,Y) = \frac{cov\big(rg(X), rg(Y)\big)}{\sigma_{rg_{(X)}} \times \sigma_{rg_{(Y)}}} \tag{3}$$

- Coefficient Tanimoto

  Le coefficient de Tanimoto ou coefficient de Jaccard, est défini comme le rapport d'intersection de deux ensembles de données différents sur leur union. Ce coefficient est particulièrement utile lorsque le jeu de données est clairsemé. Le coefficient de Tanimoto entre X et Y peut être exprimée par la formule ci-dessous :

$$T(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|} \tag{4}$$

- Distance Euclidienne

  La distance euclidienne est la distance entre les points de l'espace d'Euclide. Par conséquent, la distance euclidienne entre deux points X et Y dans l'espace à n dimensions est définie par la formule suivante :

$$d(X,Y) = \sqrt{\sum_{k=1}^{n} (X_k - Y_k)^2} \tag{5}$$

  Plus la distance est petite, plus la similitude est grande.

$$sim(X,Y) = \frac{1}{1 + d(X,Y)} \tag{6}$$

- Distance de Manhattan

  La distance de Manhattan, également connue sous le nom de distance de valeur absolue, représente la somme des différences absolues entre les points à travers toutes les dimensions. Cette mesure est plus élevée pour les points qui ne sont pas similaires. . Sa formule est définie comme suit :

$$d(X,Y) = \sum_{k=1}^{n} |X_k - Y_k| \tag{7}$$

Étape 2 : Sélection des voisins

Les voisins sont les utilisateurs présentant des similitudes marquées avec l'utilisateur actif. Dans le cadre du filtrage collaboratif basé sur l'utilisateur, les méthodes les plus couramment employées pour déterminer les utilisateurs les plus similaires incluent : le voisinage de taille fixe, qui prend en compte les N premiers utilisateurs (K-Nearest Neighbors KNN – Taunk et al. (2019)), présentant la plus grande similarité en tant que voisins, et le voisinage basé sur le seuil, qui considère tous les utilisateurs dont la similarité dépasse un seuil donné en tant que voisins. En revanche, le filtrage collaboratif basé sur les items applique les mêmes méthodes mentionnées ci-dessus, mais en se concentrant sur les items plutôt que sur les utilisateurs.

Étape 3 : Prédiction des notes

Le SR utilise les notes des voisins pour calculer la moyenne pondérée, une formule visant à estimer ou prédire les éléments inconnus de la matrice utilisateurs-items. Cette formule est définie comme suit :

$$r_{xi} = \frac{\sum_{y \epsilon N} S_{xy} \cdot r_{y_i}}{\sum_{y \epsilon N} S_{xy}} \tag{8}$$

Où $S_{xy} = sim(x, y)$ représente la similarité entre *x* et *y*.

Où r_{xi} est le vecteur d'évaluation de l'utilisateur *x*, et *N* est l'ensemble des *K* utilisateurs les plus similaires à *x* qui ont évalué l'élément *i*.

Étape 4 : Recommandation Top-N

Dans la dernière étape, le SR classe tous les items en fonction de leurs notes prédites et recommande les items finaux à l'utilisateur cible.

- Le filtrage collaboratif basé sur le modèle : Il repose sur les notes de l'utilisateur pour estimer ou apprendre le modèle, qui est ensuite utilisé pour les prédictions. Les méthodes les plus récentes de filtrage collaboratif basé sur le modèle sont développées à l'aide de différents algorithmes d'exploration de données et d'apprentissage automatique pour prédire les évaluations des utilisateurs pour le contenu non évalué. Ces méthodes incluent des approches telles que les réseaux bayésiens (Friedman et al., 1997), les modèles de regroupement (clustering en anglais) (Aggarwal & Reddy, 2014), et les modèles sémantiques latents (Latent Semantic Analysis LSA – Dumais (2004)).

1.2.2.3 Les SR basés sur le contenu

Ces systèmes reposent sur la similarité des attributs de contenu. Ils suggèrent aux utilisateurs des items similaires à ceux qu'ils ont appréciés par le passé, répondant ainsi à la question « Me montrer plus de contenu similaire à ce que j'ai aimé ? ». Pour estimer la probabilité qu'un item plaise à un utilisateur, ils utilisent des techniques telles que des arbres de décision ou des réseaux de neurones (Goodfellow, 2016).

1.2.2.4 Les SR basés sur la connaissance

Ces systèmes recommandent des items en se basant soit sur des inférences sur les préférences des utilisateurs, soit en utilisant des connaissances spécifiques du domaine. De cette manière, ils répondent à la question « Dites-moi ce qui correspond à mes besoins ? ».

1.2.2.5 Les SR hybrides

Ces systèmes combinent les méthodes des catégories mentionnées ci-dessus afin de surmonter certaines de leurs limitations, telles que le problème de démarrage à froid (Cold Start Problem CSP – Falk (2019)), qui se produit en raison du manque d'informations suffisantes.

1.3 LES AGENTS CONVERSATIONNELS

Selon Haristiani (2019), un agent conversationnel (AC) est un programme informatique ou une IA qui mène des conversations via l'audio ou le texte (Shevat, 2017) et interagit avec les utilisateurs dans un domaine ou un sujet particulier en fournissant des réponses intelligentes en langage naturel (Abdul-Kader & Woods, 2015; Azwary et al., 2016). Ces dernières années, l'utilisation des AC s'est avérée précieuse dans divers contextes en automatisant des tâches et en améliorant l'expérience des utilisateurs. Cela inclut notamment les services client automatisés, la réservation de vols et les transactions en ligne. Dans ces circonstances, les AC peuvent être privilégiés par rapport à d'autres formes d'assistance, tels qu'un appel téléphonique ou une recherche en ligne, en raison de leur praticité et de leur rapidité (Brandtzaeg & Følstad, 2017).

1.3.1 La classification des agents conversationnels

Selon Hussain et al. (2019), le domaine des AC a connu une évolution dynamique au cours des dernières années grâce à l'émergence de nouvelles technologies, rendant la classification précise des AC plutôt subjective en fonction de la manière dont ils sont utilisés.

Les AC peuvent être classés dans différentes catégories en fonction de plusieurs critères, tels que le mode d'interaction, le domaine de connaissance, leur utilisation et les techniques de conception (méthode de génération de réponses) généralement utilisées dans la création de ces AC. Ces critères englobent des aspects tels que la philosophie fondamentale de la conception des AC, la manière dont le contexte doit être stocké et pris en compte pour comprendre la conversation, ou encore le type et l'objectif de la conversation pour lesquels l'AC doit être conçu (Ramesh et al., 2017). Cette classification générale (voir figure 1.2) peut être réalisée en utilisant les critères suivants (Hussain et al., 2019) :

1. Mode d'interaction (basé sur le texte ou basé sur la voix/la parole)
2. Application de l'AC (orientée vers les tâches ou non orientée vers les tâches)
3. Basé sur des règles ou sur des techniques avancées en l'IA (apprentissage automatique, apprentissage profond, etc.)
4. Spécifique au domaine ou à domaine ouvert

Selon Saradhi et al. (2022), les AC en mode interactif (interactive mode en anglais) sont ceux que nous utilisons au quotidien, comme Siri d'Apple. Ils peuvent être basés sur du texte (text-based en anglais) ou sur la voix (voice-based en anglais). Leur architecture repose principalement sur une IA complexe. Ils se développent en permanence grâce à des algorithmes d'apprentissage automatique qui observent les interactions des utilisateurs. Les AC basés sur un domaine de connaissance (knowledge domain-based en anglais) se concentrent principalement sur la récupération de réponses appropriées à partir d'une base de données dans un domaine ou une zone particulière où ils possèdent des connaissances, en recherchant et en faisant correspondre les mots-clés dans les questions des utilisateurs. Ils peuvent être à domaine ouvert (open domain en anglais) ou à domaine fermé (closed domain en anglais). Les AC à domaine ouvert n'ont aucune restriction en termes de domaine d'expertise ou de connaissances. Ils ne sont pas limités à un seul type de domaine, mais peuvent en connaître plusieurs. Grâce à cette caractéristique, les conversations avec ces AC sont plus réalistes et naturelles. En revanche, les AC à domaine fermé ont des connaissances

limitées à un certain niveau. Ils ne peuvent répondre qu'aux questions relevant de leur domaine de connaissances.

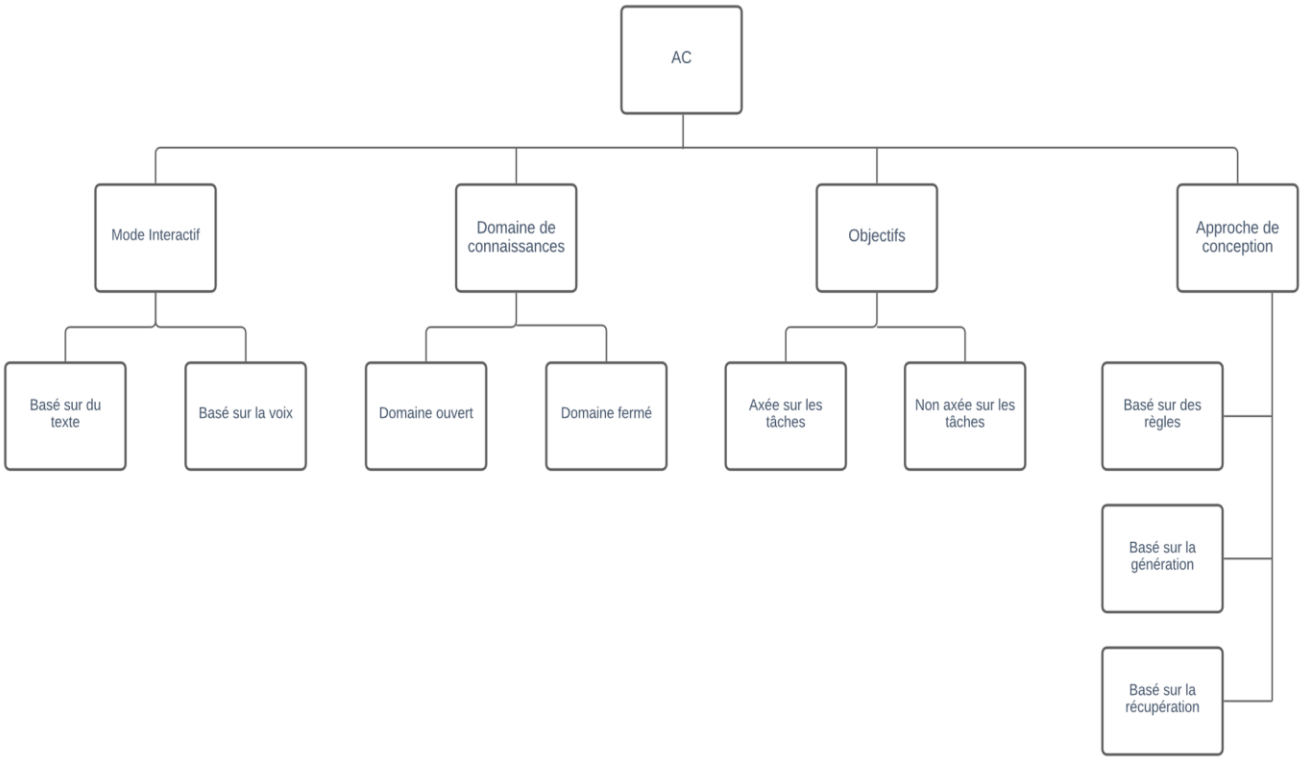


Figure 1.2 – Classification générale des agents conversationnels (adapté d'Hussain et al. (2019)).

Selon Saradhi et al. (2022), les AC basés sur leur approche de conception (design approach en anglais) se déclinent en trois types : les AC basés sur des règles (rule-based en anglais), les AC basés sur la récupération (retrieval-based en anglais) et les AC basés sur la génération (generative-based en anglais). Les réponses des AC basés sur des règles sont toutes prédéfinies, ce qui permet aux développeurs de contrôler les niveaux de conversation du AC (Wayesa, 2022). Ils utilisent une structure arborescente pour répondre aux requêtes des utilisateurs, avec plusieurs questions de suivi pour obtenir la meilleure réponse possible. Les AC basés sur la récupération sont similaires aux AC basés sur des règles, avec des connaissances à domaine fermé et l'utilisation de réseaux neuronaux qui les rendent plus avancés. Ils effectuent principalement trois étapes : classification de l'intention,

reconnaissance des entités et sélection de la réponse. Ils peuvent être mis en œuvre avec des techniques telles que les perceptrons multicouches ou la similarité des phrases, etc. (Ciayandi et al., 2020). Les AC basés sur la génération sont des agents avancés capables de générer des réponses en combinant des éléments linguistiques plutôt que de simplement sélectionner des réponses prédéfinies. Ils sont construits à l'aide de modèles seq2seq (Liu et al., 2018) utilisés pour la traduction automatique. Les techniques d'apprentissage en profondeur peuvent être utilisées pour affiner ces AC (Sheikh et al., 2019).

Cependant, il est important de noter que les AC sont généralement classées en deux catégories principales en fonction de leurs objectifs (Yan et al., 2017; Budulan, 2018) :

1. Les AC orientés vers les tâches
2. Les AC non orientés vers les tâches

Selon Hussain et al. (2019), les AC orientés vers les tâches sont conçus pour une tâche spécifique et sont configurés pour avoir des conversations courtes, généralement dans un domaine restreint. Contrairement aux AC orientés vers les tâches, Les AC non orientés vers les tâches sont des systèmes conçus pour des conversations étendues, mis en place pour imiter les échanges conversationnels non structurés, une caractéristique de l'interaction entre humains, plutôt que de se concentrer sur une tâche particulière telle que la réservation de vols d'avion.

1.3.2 L'architecture typique d'un agent conversationnel

Les AC sont des systèmes complexes qui intègrent une large gamme de technologies de la parole et du langage sous la forme de plusieurs composants, tels que la reconnaissance automatique de la parole (ASR), la compréhension du langage naturel (Natural Language Understanding NLU – Allen (1995)), la gestion des dialogues (Dialogue Management DM – Larsson (2002)), la génération du langage naturel (Natural Language Generation NLG – McDonald (2010)) et la synthèse de parole à partir du texte (Text-To-Speech Synthesis TTS – Taylor (2009)).

Une approche simple et largement reconnue de l'architecture des AC (Khouzaimi, 2022) consiste à la développer sous la forme d'une chaîne de processus (un pipeline), dans laquelle l'agent prend une requête de l'utilisateur en entrée et génère une réponse en sortie. Cette architecture est illustrée à la figure 1.3.

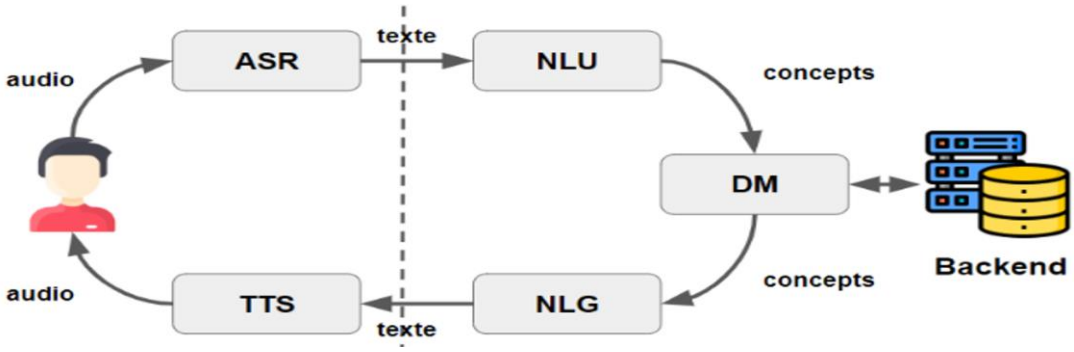


Figure 1.3 – Architecture des agents conversationnels (source : Khouzaimi (2022)).

Dans cette architecture, l'ASR prend le message vocal de l'utilisateur et le transforme en un texte en langage naturel. Le NLU analyse le texte et génère une représentation sémantique (extraction de l'intention et des entités). Cette représentation est ensuite gérée par le gestionnaire de dialogue (DM), qui examine le discours et le contexte de la conversation, et génère une réponse à un niveau sémantique. Le NLG génère alors une représentation textuelle, qui se présente sous forme de phrases en langage naturel, et la transmet à un convertisseur texte-parole (TTS) qui génère la sortie audio destinée à l'utilisateur.

1.3.2.1 La compréhension du langage naturel

Selon Kavlakoglu (2020), la compréhension du langage naturel (NLU) est une sous-discipline du traitement du langage naturel, qui utilise l'analyse syntaxique et sémantique du texte et de la parole pour déterminer le sens d'une phrase. La syntaxe fait référence à la structure grammaticale d'une phrase, tandis que la sémantique renvoie à son sens voulu. La compréhension du langage naturel établit également une ontologie pertinente : une structure de données qui spécifie les relations entre les mots et les phrases. Alors que les humains le

font naturellement dans une conversation, la combinaison de ces analyses est nécessaire pour qu'une machine comprenne le sens voulu de différents textes.

1.3.2.2 La génération du langage naturel

La génération du langage naturel (NLG) est un autre sous-ensemble du NLP. Alors que le NLU se concentre sur la compréhension de la lecture par l'ordinateur, le NLG permet aux ordinateurs d'écrire. Le NLG est le processus de production d'une réponse en langage humain basée sur certaines données en entrée. Ce texte peut également être converti en un format vocal grâce aux services de synthèse de parole à partir du texte (TTS). Le NLG englobe également des capacités de résumé de texte qui génèrent des résumés à partir de documents en entrée tout en préservant l'intégrité de l'information. Initialement, les systèmes de NLG utilisaient des modèles pour générer du texte. Cependant, au fil du temps, les systèmes de génération de langage naturel ont évolué avec l'application de chaînes de Markov cachées (Hidden Markov Model HMM – Gales et Young (2008)) et de réseaux de neurones récurrents (Recurrent Neural Network RNN – (Sainath et al., 2020; Li et al., 2020)) permettant une génération de texte plus dynamique en temps réel (Kavlakoglu, 2020).

Tout comme avec le NLU, les applications du NLG doivent prendre en compte les règles linguistiques basées sur la morphologie, les lexiques, la syntaxe et la sémantique pour faire des choix sur la manière de formuler les réponses de manière appropriée. Elles abordent cela en trois étapes (Kavlakoglu, 2020):

1. Planification du texte : pendant cette étape, le contenu général est formulé et organisé de manière logique.

2. Planification des phrases : cette étape prend en compte la ponctuation et le flux du texte, en répartissant le contenu en paragraphes et en phrases, et en incorporant des pronoms ou des conjonctions lorsque cela est approprié.

3. Réalisation : cette étape tient compte de l'exactitude grammaticale, en veillant à ce que les règles concernant la ponctuation et les conjugaisons soient respectées.

1.3.2.3 Le gestionnaire de dialogue

Selon Deng (2022), le composant de gestion de dialogue (DM) est la partie « cérébrale » d'un AC. Il décide de la manière dont le système devrait répondre ou agir. Le DM génère la réponse du système sous forme d'une représentation logique, qui ensuite devient l'entrée du module de NLG. Un rôle crucial du composant DM est de représenter l'état actuel de la conversation en se basant sur les états précédents. Pour cela, il suit l'historique des échanges afin de créer un contexte de conversation (state tracking en anglais). Cet état est ensuite utilisé pour déterminer la prochaine action à entreprendre : quel type de réponse envoyer, s'il faut faire une requête API, et bien d'autres choses encore.

1.3.2.4 La synthèse vocale

La synthèse vocale (TTS) est un processus de modélisation du langage naturel qui consiste à transformer des unités de texte en unités de parole pour une présentation audio. C'est l'inverse de la conversion de la parole en texte, où une technologie prend des paroles et tente de les enregistrer avec précision sous forme de texte. La synthèse texte-parole est désormais courante dans les technologies qui cherchent à convertir des textes numériques en sortie audio pour aider ceux qui ne peuvent pas lire ou à d'autres fins d'utilisation (Rouse, 2012). Selon Pouget (2017), les synthétiseurs de parole peuvent être regroupés en trois classes qui ont émergé au fil du temps avec l'amélioration des capacités de stockage et de calcul des ordinateurs : la synthèse par règles (Klatt, 1987), la synthèse par corpus (Hunt & Black, 1996) et la synthèse paramétrique statistique (Karaali et al., 1996).

1.3.2.5 La reconnaissance automatique de la parole

La reconnaissance automatique de la parole (ASR) se réfère à plusieurs types de systèmes dont la mission est de décoder l'information portée par le signal vocal. L'objectif de l'ASR est de transformer un signal vocal en une séquence de symboles représentant le contenu du signal.

a) *HISTORIQUE*

La première machine à reconnaître la parole était un jouet des années 1920 appelé « Radio Rex », un chien en celluloïd qui bougeait grâce à un ressort libéré par une énergie acoustique de 500 Hz, semblant ainsi venir lorsqu'on l'appelait (Jurafsky & Martin, 2009). Depuis lors, des progrès considérables ont été réalisés, aboutissant à l'utilisation courante des systèmes ASR dans divers appareils, devenant partie intégrante des systèmes domestiques intelligents.

En 2010, Siri Inc. a lancé Siri[2], révolutionnant la reconnaissance vocale. L'application pouvait répondre aux demandes des utilisateurs, fournir des recommandations et interagir individuellement avec chaque utilisateur en analysant ses demandes et son comportement. La même année, Apple a acquis Siri Inc., consolidant ainsi la technologie de reconnaissance vocale et d'assistance virtuelle dans son écosystème. En 2011, Google a introduit Google Voice Search[3], permettant aux utilisateurs de naviguer sur Internet grâce à des commandes vocales. En 2014, Amazon a lancé Alexa[4], un assistant virtuel qui a popularisé l'interaction vocale dans les foyers via des dispositifs comme l'Amazon Echo. Tout comme Siri et Google Voice Search, Alexa peut répondre aux demandes des utilisateurs, proposer des recommandations et contrôler diverses fonctions de la maison intelligente, marquant une nouvelle étape dans l'évolution de la reconnaissance automatique de la parole.

b) *LE SCHEMA D'UN SYSTEME DE RECONNAISSANCE AUTOMATIQUE DE LA PAROLE*

La figure 1.4 illustre un schéma typique d'un ASR de la parole. Il comprend les étapes et modèles suivants (Young, 1996) :

[2] https://www.apple.com/ca/fr/siri/

[3] https://www.google.com/search/about/

[4] https://alexa.amazon.com/

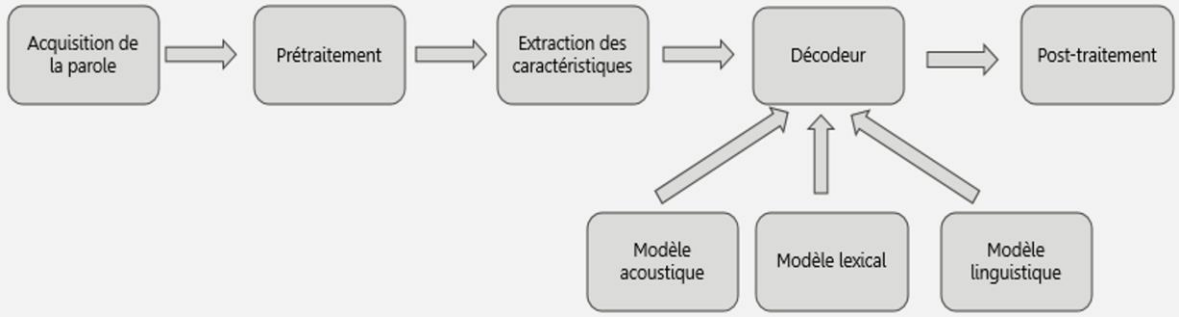


Figure 1.4 – Schéma d'un ASR (adapté de Young (1996)).

- Acquisition de la parole : Cette étape se fait généralement à l'aide d'un microphone ou d'un autre dispositif d'enregistrement audio.

- Prétraitement : Cette étape vise à améliorer la qualité du signal audio en réduisant le rapport signal sur bruit, en atténuant les sources de bruit indésirable et en filtrant le signal. Elle parvient à cela en tenant compte de la nature non stationnaire de la parole, grâce à l'utilisation de techniques de fenêtrage. Cette étape permet d'éliminer les distorsions causées par le canal de transmission et d'identifier les segments de parole active, contribuant ainsi à améliorer la précision globale du système ASR.

- Extraction des caractéristiques : À cette étape, les caractéristiques distinctives des signaux vocaux sont extraites en préservant les informations pertinentes tout en étant résistantes au bruit et aux variations. Les coefficients cepstraux de fréquence de Mel (Mel Frequency Cepstral Coefficients MFCC – Davis et Mermelstein (1980)) sont parmi les caractéristiques les plus fréquemment utilisées dans les systèmes de reconnaissance de la parole (Rabiner & Juang, 1993; Hinton et al., 2012). Le processus d'extraction (voir figure 1.5) implique la segmentation du signal vocal en trames quasi stationnaires de courte durée (20 à 40 ms), chevauchées toutes les 10 ms. Pour chaque trame, une banque de filtres à l'échelle Mel est appliquée à son estimation du spectre de puissance. Les MFCC mesurent l'enveloppe du spectre de

puissance dans chaque trame, corrélée à la forme du tractus vocal, fournissant ainsi une représentation appropriée du son (Li & Principe, 2018).

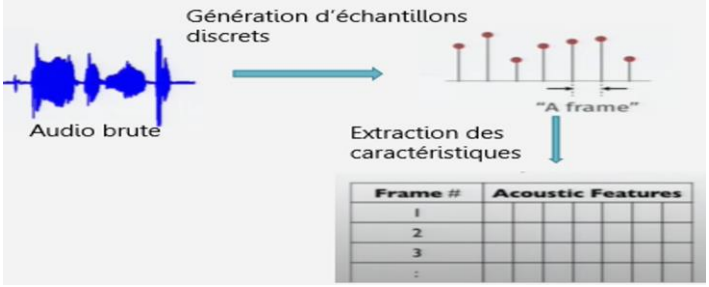


Figure 1.5 – Illustration de l'étape d'extraction des caractéristiques (traduit de Jyothi (2019)).

- Modèle acoustique : Il représente les caractéristiques de la parole en se basant sur le principe que les sons de la parole peuvent être représentés par des phonèmes. Les phonèmes, qui sont des unités discrètes et distinctives d'une langue, sont utilisés pour différencier les mots.

- Modèle lexical : Équivalent au modèle de prononciation, il établit un lien entre les phonèmes et les mots. Il joue le rôle d'intermédiaire entre le modèle acoustique et le modèle linguistique. Contrairement aux modèles acoustiques et linguistiques, le modèle de prononciation n'est pas entraîné, mais plutôt créé par des experts en tant que simple dictionnaire de prononciations.

- Modèle linguistique : C'est un modèle probabiliste qui réorganise les mots dans un ordre dont la séquence est la plus probable. Il existe deux types de modèles de langage : le modèle déterministe, défini par les règles de grammaire, et le modèle statistique, défini par la probabilité d'occurrence de séquences de mots (uni-gramme, bi-gramme, tri-gramme, n-gramme), c'est-à-dire la probabilité que des mots différents apparaissent en tenant compte du contexte du mot récent.

- Décodeur : Cette étape combine les trois composantes (modèle acoustique, modèle lexical et modèle linguistique) pour construire un graphe de recherche à travers un

espace considérable de possibilités. L'objectif est de trouver la meilleure séquence possible parmi toutes les séquences possibles de mots. Ce processus peut être effectué en utilisant des algorithmes tels que l'algorithme de Viterbi (Viterbi, 1967; Forney, 1973) ou la recherche en faisceau (en anglais, Beam Search) (Rubin & Reddy, 1977; Collobert et al., 2019).

- Post-traitement : Cette étape vise à améliorer la transcription du texte généré dans l'étape précédente (décodage) par les humains ou les machines.

1.4 L'INGÉNIERIE DIRIGÉE PAR LES MODÈLES

L'ingénierie dirigée par les modèles (MDE) est une approche systématique utilisant des modèles comme entités centrales tout au long du cycle de vie du génie logiciel. Son objectif est de générer, en tout ou en partie, le code du système à partir de ces modèles, établissant ainsi le développement sur des transformations successives conduisant du modèle aux codes sources (Bézivin, 2006; Chénard et al., 2012). Une implémentation bien connue de la MDE est l'architecture dirigée par les modèles[5] (Model-Driven Architecture MDA – (OMG, 2014)).

[5] Il existe deux visions principales concernant la relation entre MDE et MDA. La première vision considère MDA comme une mise en œuvre spécifique et normalisée des principes de MDE. Dans ce contexte, MDA emploie des modèles et des langages standardisés pour automatiser et simplifier le développement logiciel, en accord avec les objectifs globaux de MDE. La seconde vision perçoit MDE comme une évolution ou une généralisation des concepts introduits par MDA. Bien que MDA ait été formalisé en premier par l'OMG en 2001, MDE a par la suite étendu ces concepts pour englober une variété de pratiques et de méthodologies de modélisation, offrant ainsi une plus grande flexibilité et adaptabilité en ingénierie logicielle. De plus, l'introduction de MDE visait aussi à contourner le fait que MDA est une marque déposée de l'OMG, permettant une adoption plus large des principes de modélisation sans les contraintes associées à une marque spécifique.

1.4.1 L'architecture dirigée par les modèles (MDA)

1.4.1.1 Les principes et concepts du MDA

L'architecture dirigée par les modèles est basée sur les principes suivants (Brown, 2004) :

- Les modèles exprimés dans une notation bien définie sont la pierre angulaire de la compréhension des systèmes pour les solutions à l'échelle de l'entreprise.
- La construction de systèmes peut être structurée autour d'un ensemble de modèles avec des transformations définies entre eux, organisés dans un cadre architectural de couches et de transformations.
- Un fondement formel, basé sur des métamodèles, facilite l'intégration et ls transformation significative entre les modèles, permettant l'automatisation par des outils.
- L'acceptation à grande échelle de cette approche nécessite des normes industrielles pour assurer l'ouverture aux consommateurs et stimuler la concurrence entre les fournisseurs.

Les concepts qui constituent les fondements de l'approche MDA sont les suivants (OMG, 2014) :

- Système : Il est défini comme une collection de parties et de relations entre ces parties, organisées en vue d'accomplir un certain objectif. La modélisation, se distinguant des technologies d'implémentation comme le code source, offre un moyen puissant de représenter, comprendre et spécifier les systèmes.
- Modèle : Il est défini comme une information représentant sélectivement un aspect d'un système, en lien explicite ou implicite avec ce système. Le modèle doit inclure toutes les informations dans la portée du système, les règles d'intégrité qui lui sont

applicables, ainsi que la signification des termes utilisés. Il peut représenter divers aspects d'un système tels que l'activité, le domaine, le logiciel, le matériel, l'environnement, etc., et peut prendre différentes formes, lisibles par l'homme ou existant sous forme électronique.

- Couches architecturales : Elles sont généralement catégorisées en trois niveaux d'abstraction :

    1. Modèles de domaine : Ils représentent des aspects réels du domaine, tels que les personnes, les lieux, les choses et les lois. Ces modèles décrivent des « choses réelles » du domaine, pas simplement des représentations dans un système d'information.

    2. Modèles de systèmes logiques : Ces modèles décrivent comment les composants d'un système interagissent entre eux, avec les personnes et les organisations. Ils sont conçus pour aider une organisation ou une communauté à atteindre ses objectifs.

    3. Modèles de mise en œuvre : Ils décrivent la manière dont un système ou un sous-système particulier est mis en œuvre pour accomplir ses fonctions. Ces modèles sont souvent liés à une technologie ou une plate-forme d'implémentation spécifique.

- Vue et Point de vue : Un point de vue établit des critères réutilisables pour construire, sélectionner et présenter une partie de l'information sur un système. Il répond aux préoccupations spécifiques des parties prenantes. D'autre part, une vue est une représentation d'un système particulier qui est conforme à un point de vue donné. Les vues sont des instantanés spécifiques du système qui reflètent les aspects pertinents du point de vue choisi.

- Abstraction : Dans le contexte de modélisation, elle consiste à comprendre un système de manière plus générale en éliminant certains éléments de la portée

définie. Un modèle est plus abstrait s'il englobe un ensemble plus large de systèmes, tandis qu'il est moins abstrait s'il est spécifique à un seul système ou à un ensemble restreint de systèmes. La modélisation et l'abstraction vont de pair, permettant de comprendre et de spécifier un système tout en abstrayant certains détails pour rendre le modèle plus généralement applicable. Un équilibre doit être trouvé pour que le modèle soit à la fois abstrait et exploitable.

- Transformation : Elle consiste à produire différents modèles, points de vue ou artefacts à partir d'un modèle de base. Elle peut être utilisée pour générer une représentation différente du même contenu conceptuel ou pour passer entre différents niveaux d'abstraction et couches architecturales. Les transformations MDA combinent souvent plusieurs modèles d'entrée pour produire un modèle de sortie, permettant ainsi de passer de modèles abstraits à plus concrets. Les spécifications de transformation MDA fournissent les mécanismes nécessaires pour effectuer ces transformations entre différentes représentations et niveaux d'abstraction.

1.4.1.2 Les catégories de modèles dans MDA

Dans l'approche MDA, selon Joaquin Miller et Jishnu Mukerji (2003) de l'OMG, quatre modèles représentent les niveaux d'abstraction de base :

1. Modèle Indépendant du Calcul (CIM - Computation Independent Model) : Il représente les exigences d'affaires, les objectifs et les règles d'un système sans préciser les détails de sa mise en œuvre technique.

2. Modèle Indépendant de la Plateforme (PIM - Platform-independent Model) : C'est un modèle d'un système indépendant de la plateforme technologique spécifique, fournissant des spécifications formelles de la structure et de la fonction du système tout en abstrayant les détails techniques.

3. Modèle Spécifique à une Plateforme (PSM - Platform-Specific Model) : Il est exprimé en termes du modèle de spécification de la plateforme, détaillant la manière dont le système sera mis en œuvre sur une plateforme technologique spécifique.

4. Modèle de Description d'une Plateforme (PDM - Platform Description Model) : Il offre un ensemble de concepts techniques représentant les composants d'une plateforme, ainsi que les services fournis par cette plateforme. Il propose des concepts utilisés pour spécifier l'utilisation de la plateforme par un système dans un PSM.

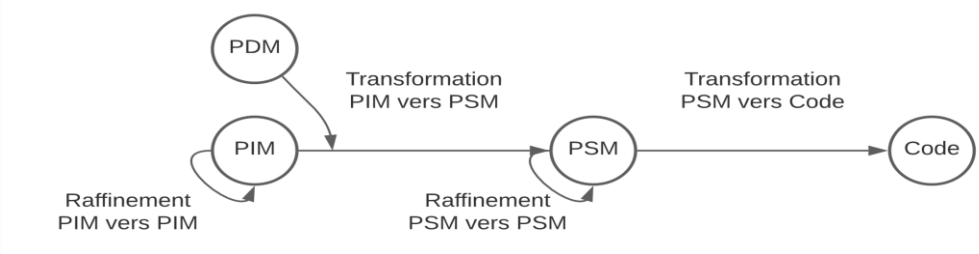


Figure 1.6 – Modèles et transformations dans l'approche MDA (adapté d'Hardebolle (2008)).

Les transformations possibles entre les types de modèles MDA sont représentées dans la figure ci-dessous (Hardebolle, 2008) :

- Transformations PIM → PIM et PSM → PSM : Ces transformations de modèle à modèle (model-to-model transformation) visent à filtrer, spécialiser ou enrichir le modèle. Elles peuvent être automatisées dans certains cas, comme la traduction vers un autre langage.

- Transformation PIM → PSM : C'est une transformation de modèle à modèle qui permet de spécialiser le PIM en fonction de la plateforme cible choisie. Elle est effectuée une fois que le PIM est suffisamment raffiné et s'appuie sur les informations fournies par PDM.

- Transformation PSM → Code : Cette transformation de modèle à texte (model-to-text transformation) convertit le PSM en une forme textuelle, généralement le code source. Le code résultant peut parfois être considéré comme un PSM exécutable, mais des ajustements manuels peuvent être nécessaires pour compléter le code généré.

1.4.1.3 Les langages de transformation MDA

Pour implémenter les types de transformation mentionnés ci-dessus, plusieurs langages de transformation sont utilisés. Parmi les langages bien connus qui prennent un modèle en entrée pour générer un modèle en sortie, on peut citer : ATL (Bézivin et al., 2003), QVT (OMG, 2005) et XSLT (W3C, 2017).

a) *ATL*

ATL (Atlas Transformation Language – Bézivin et al. (2003)) est l'un des langages de transformation de modèle à modèle les plus populaires. Une fois que le métamodèle cible et le métamodèle source sont disponibles, un fichier de transformation ATL peut produire un modèle cible à partir d'un modèle source. Pendant la transformation, un attribut d'une instance de modèle cible reçoit une valeur de retour d'une expression OCL (Object Constraint Language – OMG (2003a)) qui est basée sur une instance de modèle source. L'environnement de développement intégré d'ATL fournit un certain nombre d'outils tels qu'un éditeur de texte avec coloration syntaxique, un débogueur et un interpréteur qui visent à faciliter le développement de transformations ATL. ATL utilise également des actions impératives pour décrire les opérations spécifiques à effectuer pendant la transformation.

b) *QVT*

QVT (Query/View/Transformation) est une norme adoptée en 2005 par l'OMG (Object Management Group) pour la spécification de transformations de modèles. Elle intègre les

concepts de modèles et de méta-modèles définis par la norme MOF (Meta Object Facility – OMG (2008)). Les requêtes sont des interrogations sur un modèle MOF dans le but de trouver des éléments de modèle à transformer. Le standard QVT est principalement représenté par deux langages de transformations de modèles : QVT-R (QVT Relations), un langage déclaratif, et QVT-O (QVT Operational), un langage impératif. Cette norme joue un rôle clé dans l'architecture dirigée par le modèle (MDA) et a influencé d'autres langages de transformation tels qu'ATL (Völter et al., 2013; OMG, 2014).

c) *XSLT*

Une autre bonne alternative pour développer des transformations est XSLT (Extensible Stylesheet Language Transformations – W3C (2017)), qui bénéficie d'un écosystème mature avec un ensemble étendu d'outils et un soutien de la communauté. XSLT, défini au sein de la recommandation XSL (Extensible Stylesheet Language) du W3C[6] (World Wide Web Consortium), est un langage de transformation XML (Extensible Markup Language) déclaratif. XSLT permet d'utiliser une feuille de style pour transformer un document XML initial de deux façons : manipuler et trier le contenu (y compris en réordonnant l'ensemble du document si besoin) d'une part et transformer le contenu dans un format différent d'autre part. Un autre aspect important de XSLT est sa portabilité, ce qui le rend hautement adaptable et facilement intégrable dans divers outils et environnements. XSLT utilise XPath (XML Path Language – W3C (2017)) pour cibler des parties d'un document XML. XPath navigue dans les éléments et attributs des documents XML, guidant XSLT pour identifier des sections correspondant à des modèles prédéfinis pendant la transformation. Une fois la correspondance trouvée, XSLT transforme cette partie du document source en un document résultant (W3C, 2017; Zia, 2017). Comme cela correspond à nos besoins spécifiques, XSLT est devenu notre premier choix en raison de sa facilité de manipulation et de sa capacité à être largement utilisé dans divers domaines, notamment dans le domaine des transformations

[6] https://www.w3.org/

de modèles. Particulièrement, du fait de sa large utilisation dans d'autres contextes, on peut présumer que de nombreux utilisateurs le comprennent déjà, simplifiant ainsi le processus de création et de gestion des transformations. Nous expliquerons plus en détail notre choix de XSLT dans le chapitre 3 où nous présentons notre approche.

1.4.2 Les langages de modélisation

MDA considère UML (Unified Modeling Language – OMG (2003b)) comme un langage universel visant à faciliter le développement logiciel tout en préservant les investissements réalisés dans UML lors de la modélisation du domaine métier d'une application (J. Miller & J. Mukerji, 2003; Khriss et al., 2020). Toutefois, dans son célèbre article publié en 2004 (Cook, 2004), Steve Cook de Microsoft Corporation a émis des critiques à l'égard de MDA et UML, démontrant la pertinence continue des langages spécifiques au domaine. Dans cet article, il a exploré l'utilisation d'UML à travers trois perspectives (Khriss et al., 2020):

1. UMLAsSketch : Il s'agit d'un moyen de créer une documentation informelle du logiciel, efficace pour éliminer les différences entre les langages de modélisation.

2. UMLAsBlueprint : Dans cette perspective, les modèles UML sont considérés comme des artefacts à traduire en code source. Cependant, cela peut poser des problèmes de traduction directe vers différentes technologies. Par exemple, des difficultés ont été rencontrées avec les propriétés C# et les interfaces Java, qui contiennent des champs statiques. Pour pallier ces problèmes, l'utilisation de profils UML a été envisagée pour offrir une plus grande flexibilité. Toutefois, en pratique, cette approche ne garantit pas toujours une correspondance transparente d'UML vers les langages populaires.

3. UMLAsProgrammingLanguage : Cette approche, soutenue par une petite communauté, est similaire à la méthode Executable UML (Mellor et al., 2002).

1.4.2.1 Le langage spécifique au domaine

Selon Hermans et al. (2009), les DSL, ou langages spécifiques à un domaine, sont adaptés à des applications spécifiques. Ils sont décrits dans la littérature depuis plusieurs décennies, prenant différentes formes telles que les langages de programmation à usage spécifique (Wexelblat, 2014), orientés application (Sammet, 1972), spécialisés (Bergin Jr & Gibson Jr, 1996) ou spécifiques à une tâche (Nardi, 1993). Pour être efficace, un DSL doit explicitement définir son objectif, utiliser des concepts compréhensibles par des personnes familières avec le domaine, proposer une notation graphique ou textuelle conviviale, et disposer d'une grammaire définie pour régir la formation des expressions (Greenfield et al., 2004). Une revue des DSL les plus courants peut être consultée dans Iung et al. (2020) . La plupart des experts s'accordent sur les avantages significatifs des DSL, notamment la réduction du temps de développement (Herndon & Berzins, 1988; Sledziewski et al., 2010) et l'amélioration de la maintenabilité (Batory et al., 2000; Deursen et al., 2003; Ewald & Uhrmacher, 2014; Zheng et al., 2022).

La théorie des langages joue un rôle crucial dans la définition des DSL en se basant généralement sur quatre éléments principaux :

1. Syntaxe abstraite : Elle définit les divers concepts du langage et les règles régissant leur combinaison, indépendamment de toute représentation ou encodage spécifique. La syntaxe abstraite est généralement spécifiée par le biais de la métamodélisation et de la grammaire hors contexte (context-free grammar en anglais). Ainsi, dans le contexte de l'utilisation de grammaires, les extensions BNF (Backus-Naur Form – Wimmer et Kramler (2006)) sont couramment utilisées pour décrire la syntaxe des DSL, tandis que la métamodélisation recourt à un métamodèle exprimé à l'aide du standard de l'OMG, le cadre MOF (Meta Object Facility – OMG (2008)). Utilisé dans divers systèmes tels que les outils de modélisation, les entrepôts de données et les référentiels de métadonnées, le MOF est également impliqué dans des technologies normalisées comme UML et ses profils associés (Löwe et al., 2001). En 2006, l'OMG

a défini deux points de conformité pour le MOF : l'EMOF (Essential MOF) et le CMOF (Complete MOF). L'EMOF sert de cadre simple pour mapper les modèles MOF vers des implémentations, tandis que le CMOF est utilisé pour spécifier les métamodèles avancés, notamment pour UML (OMG, 2006). Notre étude se base sur la spécification de la syntaxe abstraite en recourant à la technique de métamodélisation utilisant le cadre MOF.

2. Syntaxe concrète : Elle représente la manière dont la syntaxe abstraite du langage est concrètement exprimée. Cette syntaxe peut être textuelle ou graphique. La forme textuelle est généralement exprimée à l'aide de grammaires, tandis que la forme graphique utilise des formes, des symboles et des annotations textuelles. La syntaxe concrète doit être non ambiguë et compréhensible par les humains.

3. Syntaxe sémantique : Elle représente la signification des éléments définis dans le langage et la signification des différentes façons de les combiner.

4. Syntaxe de sérialisation : Utilisée pour la persistance et comme format d'échange, elle n'a pas besoin d'être dans une forme compréhensible par l'humain. XML est fréquemment utilisé à cet effet.

1.5 LES USINES DE LOGICIELS

Greenfield et Short (2003) définissent une usine de logiciels comme une ligne de produits logiciels configurant des outils, des processus et des contenus extensibles à l'aide d'un modèle d'usine de logiciels basé sur un schéma. Cela automatise le développement et la maintenance de variantes d'une famille de produits en adaptant, assemblant et configurant des composants basés sur un cadre d'application. L'usine de logiciels se caractérise par son schéma et des templates. Le schéma décrit les artefacts nécessaires pour produire un produit logiciel, organisé sous la forme d'un graphe dirigé avec des termes de correspondances (mappings en anglais) définissant les relations entre les points de vue. Un terme de correspondance contient des connaissances sur la mise en œuvre des artefacts d'un point de

vue par rapport à un autre. Les templates sont des ensembles d'actifs mettant en œuvre le schéma pour construire un membre de la famille. La configuration personnalise la description de l'usine de logiciels, tandis que la configuration du template personnalise les outils et l'environnement de développement (Greenfield et al., 2004). Gérer les variabilités est l'un des défis majeurs des usines de logiciels, un sujet que nous explorerons dans le prochain chapitre.

L'usine de logiciels repose sur deux processus clés comme illustré dans la figure ci-dessous.

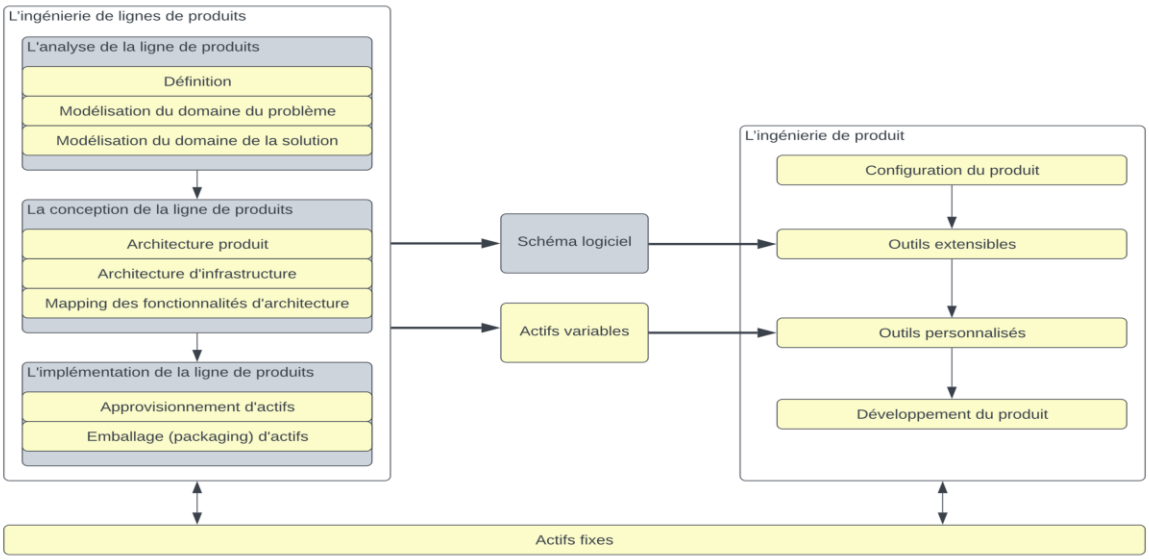


Figure 1.7 – Vue générale de l'usine de logiciels (traduit de Greenfield et Short (2003)).

1.5.1 L'ingénierie de ligne de produits

Le processus de l'ingénierie d'une ligne de produits se compose des étapes suivantes :

- Analyse de la ligne de produits : Cette étape vise à décrire le domaine du problème auquel la ligne de produits s'applique. Elle identifie et modélise les caractéristiques du problème en catégorisant chaque caractéristique candidate en tant que commune ou variable. Le

résultat de cette étape est la définition de la portée du domaine de la solution et sa modélisation. Cette définition exprime les exigences de la ligne de produits, les organisant en points de vue au sein du schéma d'usine de logiciels.

- Conception de la ligne de produits : L'objectif de cette étape est de définir l'architecture de la ligne de produits en distinguant entre les caractéristiques de conception communes et variables. Les caractéristiques de conception communes sont décrites en identifiant les décisions de conception nécessaires pour répondre à ces caractéristiques, tandis que les caractéristiques de conception variables sont exprimées à travers des points de variabilité.

- Implémentation de la ligne de produits : Cette étape est constituée de deux sous-étapes :

    1. Approvisionnement d'actifs : Cette sous-étape implique l'acquisition ou la construction d'actifs réutilisables. Ces actifs englobent des éléments liés aux exigences, tels que les langages de spécification, des ressources de mise en œuvre comprenant des modèles, des cadres d'applications et des composants, ainsi que des ressources de processus et de déploiement, comme les configurations pour les hôtes logiques.

    2. Emballage (packaging) d'actifs : Dans cette sous-étape, les actifs de production sont conditionnés en tant que templates d'usine de logiciels, prêts à être utilisés pour construire des membres spécifiques de la famille de produits.

1.5.2 L'ingénierie de produit

Le processus d'ingénierie de produit implique le développement de produits spécifiques en utilisant les actifs de la ligne de produits, englobant la création de nouveaux produits et la modification de produits existants. Des étapes et outils sont impliqués dans le développement et la modification du produit (Greenfield & Short, 2003) :

- Configuration du produit : Durant cette étape, le produit est adapté selon les besoins spécifiques et les configurations pour répondre aux fonctionnalités et

caractéristiques requises, incluant par exemple le choix de fonctionnalités spécifiques et la personnalisation de l'interface utilisateur.

- Outils extensibles : Un ensemble d'outils adaptables et extensibles facilite les différentes étapes du processus d'ingénierie du produit. Parmi ces outils, on trouve des environnements de développement intégrés (IDE) tels qu'Eclipse IDE[7] et Visual Studio[8], automatisant ainsi le développement et la maintenance des membres de la famille de produits.

- Outils personnalisés : Ces outils sont personnalisés pour correspondre aux exigences spécifiques de la famille de produits en cours de développement.

- Développement du produit : Cette étape englobe le processus de création du produit en respectant les spécifications définies, incluant la conception, la programmation, les tests, ainsi que d'autres activités de développement. Après avoir configuré l'usine de logiciels de manière appropriée, les développeurs de produits l'utilisent pour élaborer le produit, en créant des modèles pour les points de vue définis par le schéma logiciel, produisant ainsi des exécutables.

1.6 SYNTHÈSE

Ce chapitre a jeté les bases du tourisme intelligent et des technologies clés, telles que les systèmes de recommandation, les agents conversationnels et la reconnaissance automatique de la parole, qui constituent des éléments clés de notre usine de logiciels. Nous avons également exploré le cadre d'application 6A, utilisé dans notre étude pour modéliser divers aspects du secteur du tourisme en tant que modèle de domaine dans le développement d'applications mobiles pour le tourisme intelligent. Les usines de logiciels ont été présentées comme la solution choisie pour accélérer le développement des systèmes de recommandation

[7] https://www.eclipse.org/ide/

[8] https://visualstudio.microsoft.com/fr/

dans le domaine du tourisme intelligent. Nous avons examiné les langages de modélisation et les techniques pour les définir. De plus, nous avons abordé les concepts liés aux transformations de modèles pertinents pour notre approche. Dans le prochain chapitre, nous explorerons différentes approches d'automatisation du développement d'applications, en mettant en lumière notre préférence pour l'usine de logiciels dans le développement des systèmes de recommandation. Nous présenterons également une revue bibliographique sur les modèles de reconnaissance automatique de la parole, les méthodes de classification des commandes vocales et la détection d'intention.

CHAPITRE 2
L'ÉTAT DE L'ART

Dans notre étude, l'objectif est d'accélérer le développement des applications mobiles pour le tourisme intelligent en automatisant la création des systèmes de recommandation (SR). Cette démarche, déjà explorée dans d'autres domaines à l'aide de techniques de développement basées sur les modèles, sera abordée dans ce chapitre. Notre choix stratégique consiste à construire une usine de logiciels dédiée au développement des SR. Ce chapitre se concentrera sur la gestion de la variabilité au sein de l'usine de logiciels. Notre approche vise également à minimiser les tâches manuelles en intégrant la détection de commandes vocales dans la modélisation des SR. Ainsi, cette section sera consacrée à présenter des travaux antérieurs qui ont exploré la classification des commandes vocales.

2.1 AUTOMATISATION DES PROCESSUS DE DÉVELOPPEMENT D'APPLICATIONS

Le développement traditionnel se concentre généralement sur le code (Code-first development en anglais), où les développeurs écrivent directement le code sans passer par une étape de modélisation. Cette approche a conduit à la nécessité d'intégrer la modélisation et d'automatiser les tâches de développement. Dans le premier chapitre, nous avons exploré l'importance des langages de modélisation. Notre démarche a impliqué l'utilisation tant de l'UML que des DSL. Toutefois, nous avons privilégié les DSL là où cela s'avérait pertinent afin de rendre la modélisation plus intuitive, en intégrant des concepts étroitement liés au domaine concerné. Pour l'automatisation des tâches de développement, diverses initiatives ont émergé, telles que l'automatisation des tests, la génération automatique de documentation, la génération de code, l'utilisation de cadres d'application, de bibliothèques, de composants prédéfinis, l'adoption de plateformes "Low-Code/No-Code" pour un développement rapide avec un minimum de codage, et les pratiques DevOps, notamment

l'intégration continue et la livraison continue (CI/CD pour Continuous integration and Continuous Delivery). MDE et les usines de logiciels sont des composantes clés de ces efforts d'automatisation, bénéficiant également aux applications mobiles. Cette section explorera différentes approches visant à automatiser et accélérer le développement des SR dans divers domaines, notamment les langages de modélisation, les lieux géographiques, les applications web de commerce électronique et les applications mobiles.

Almonte et al. (2020) présentent un cadre générique visant à simplifier le développement des systèmes de recommandation (SR). Ce cadre facilite la création d'applications logicielles pour des utilisateurs non experts en développement, en utilisant des plates-formes à faible code. Basé sur un métamodèle prédéfini, l'architecture propose un DSL permettant de configurer divers aspects du SR, tels que les algorithmes de recommandation, les méthodes de fractionnement des données et les techniques d'évaluation. Pour démontrer leur approche, les auteurs guident les utilisateurs dans la création d'un SR suggérant des attributs, des méthodes et des superclasses pour de nouvelles classes, en se basant sur la définition d'autres classes similaires dans leurs projets. Le système analyse les caractéristiques des classes existantes, telles que le nom et le type de données pour les attributs, le nom et le type de retour pour les opérations, ainsi que le nom des superclasses, afin d'identifier des similitudes.

Di Sipio et al. (2020) ont adopté une approche similaire à celle d'Almonte et al. (2020), en suivant la même philosophie pour simplifier le développement des SR. Leur paradigme de développement met à la disposition des utilisateurs des interfaces graphiques, des outils de glisser-déposer et des formulaires pour définir, simplifier et démocratiser le processus de création de ces systèmes. Ils préconisent l'utilisation d'une plateforme à faible code construite à partir d'un métamodèle générique, offrant tous les composants de modélisation nécessaires pour mettre en œuvre des fonctionnalités récurrentes dans les SR, telles que le prétraitement des données, la capture du contexte et la production et la présentation des recommandations.

Dans leur article (Di Sipio et al., 2022), les mêmes auteurs présentent un prototype extensible visant à faciliter le processus de développement des SR en fournissant un outil à faible code. Intitulé LEV4REC, ce prototype propose un support complet pour la conception, la configuration et le déploiement des SR, y compris l'ajustement fin des paramètres. Le processus commence par la définition d'un modèle de fonctionnalités permettant de sélectionner les caractéristiques du SR, suivi de la génération automatique d'un modèle de configuration initial. Enfin, un générateur de code produit le code source correspondant du SR modélisé, prêt pour le déploiement réel.

Rojas et al. (2009) ont proposé une approche basée sur le MDE pour faciliter le développement de SR de points d'intérêt (POI) destinés aux applications web. Cette stratégie de modélisation vise à résoudre les défis liés à l'adoption des techniques de recommandation dans ces applications en identifiant les algorithmes de recommandation à partir d'un niveau d'abstraction élevé. Les modèles UML, tels que le diagramme de classes pour la spécification structurelle et le diagramme de communication pour la spécification comportementale des algorithmes de recommandation, ont été utilisés. L'approche a intégré un algorithme de filtrage collaboratif, impliquant la comparaison des préférences des utilisateurs pour différents éléments, avec une définition des aspects clés des algorithmes de recommandation, tels que les préférences des utilisateurs, les métriques de similarité, et la sélection et l'ordre des recommandations. Dans une extension ultérieure (Rojas & Uribe, 2013), les auteurs ont combiné différents types de préférences d'utilisateurs et intégré plusieurs algorithmes de recommandation, incluant le filtrage collaboratif avec du contenu basé sur le contexte et la localisation, pour améliorer la structure du système de recommandation mobile.

Rojas et Garrido (2017) ont introduit un cadre de développement visant à accélérer l'implémentation des SR. Au cœur de ce cadre se trouve un modèle abstrait décrivant les concepts essentiels de la recommandation, en utilisant des techniques de filtrage collaboratif et des éléments de réseaux sociaux. Ce modèle se présente sous la forme d'un diagramme de classes UML, où les concepts spécifiques au domaine sont associés à des concepts indépendants du domaine exprimés dans le modèle abstrait. Le noyau du modèle est complété

par des modèles d'algorithmes de similarité et de prédiction, représentés par des diagrammes de communication UML. Ces diagrammes calculent la similarité entre les éléments ou les utilisateurs et attribuent une valeur de préférence aux éléments recommandés. En utilisant des règles de transformation M2T (model to text), à la fois le diagramme de classes UML résultant et les diagrammes de communication sélectionnés sont automatiquement convertis en code, facilitant ainsi la création rapide de prototypes de SR.

L'expérience menée pour valider l'approche de développement de SR a consisté à comparer les temps de développement de trois scénarios distincts. Dans le premier scénario, l'application complète du cadre de développement, incluant la génération automatique du code à partir du modèle abstrait, a montré un avantage significatif avec une moyenne de 33 minutes. Le deuxième scénario, où cette approche a été intégrée partiellement avec les algorithmes du cadre d'application NReco.Recommender d'Apache Mahout (Owen et al., 2011), a également montré des résultats positifs avec une moyenne de 31,5 minutes. En revanche, le troisième scénario, impliquant la création manuelle du diagramme architectural du SR sans utiliser le modèle abstrait, a nécessité en moyenne 66 minutes de développement. Ces résultats soulignent l'efficacité de l'utilisation du modèle abstrait et de la génération automatique de code pour accélérer le processus de développement des SR.

2.2 GESTION DE LA VARIABILITÉ DANS LES USINES DE LOGICIELS

La variabilité, selon Bachmann et Clements (2005), désigne la capacité d'un système, d'un actif ou d'un environnement de développement à produire un ensemble d'artefacts planifiés présentant des différences entre eux. La gestion efficace de cette variabilité dans les usines de logiciels revêt une importance cruciale pour le développement de produits flexibles et adaptables. La plupart des approches pour modéliser la variabilité reposent soit sur la modélisation des caractéristiques (feature modeling ou FM), soit sur la modélisation des décisions (decision modeling en anglais ou DM) (Czarnecki et al., 2012).

La FM a été initialement proposée dans le cadre de la méthode d'analyse de domaine orientée caractéristiques (Feature oriented domain analysis ou FODA) par Kang et al. (1990).

Dans le cadre de FODA, un domaine est défini comme un ensemble de systèmes actuels et futurs partageant des capacités communes. L'analyse de domaine vise à découvrir et à représenter les similitudes et les variabilités entre eux. Un élément clé de la modélisation des caractéristiques est le diagramme de caractéristiques, qui est utilisé pour représenter graphiquement les caractéristiques du système et leurs relations. Dans un diagramme de caractéristiques, les caractéristiques sont généralement représentées sous forme de nœuds, avec des liens entre les différentes caractéristiques pour indiquer les relations telles que l'inclusion, l'exclusion mutuelle, la dépendance, etc. Cette représentation visuelle permet aux développeurs de visualiser facilement la structure et les dépendances des caractéristiques du système, ce qui facilite la compréhension et la gestion de la variabilité.

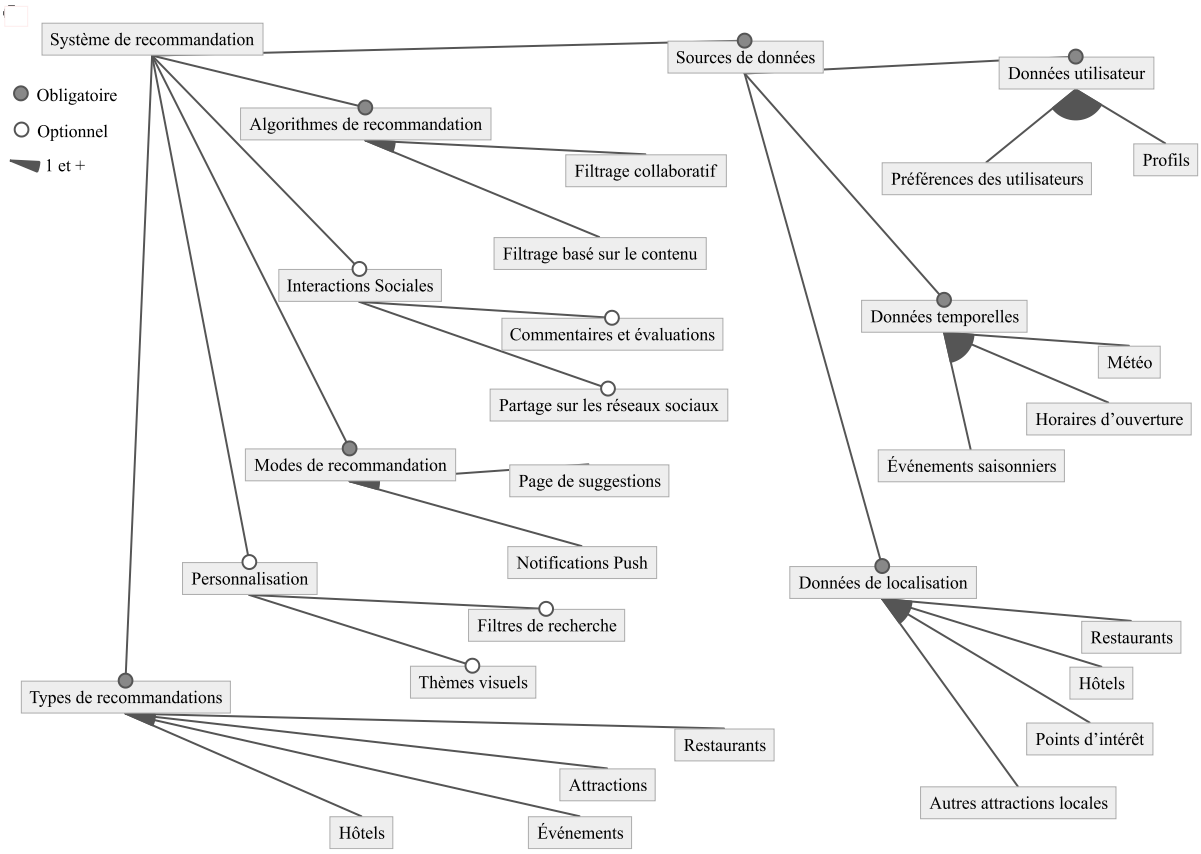


Figure 2.1 – Exemple de modèle de caractéristiques pour un SR.

La figure 2.1 montre un exemple de modèle de caractéristiques pour un système de recommandation. Le modèle est structuré autour de cinq catégories principales, chacune détaillant des caractéristiques spécifiques destinées à améliorer l'expérience utilisateur et à fournir des recommandations personnalisées basées sur des données diverses.

1. Sources de données (Obligatoire) : Elle se subdivise en :

    - Données utilisateur (1 et +), comprenant les profils et préférences des utilisateurs, essentiels pour personnaliser l'expérience.

    - Données de localisation (1 et +), qui rassemblent des informations sur les points d'intérêt, restaurants, hôtels et autres attractions locales.

    - Données temporelles (1 et +), incluant les événements saisonniers, la météo et les horaires d'ouverture, pour offrir des recommandations opportunes.

2. Algorithmes de recommandation (1 et +) : Avec des options telles que le filtrage collaboratif ou basé sur le contenu, activables selon les besoins spécifiques.

3. Types de recommandations (1 et +) : Les utilisateurs peuvent recevoir des recommandations dans divers domaines, incluant des hôtels, attractions, restaurants et événements locaux, permettant une expérience riche et diversifiée.

4. Modes de recommandation (1 et +) : Les recommandations peuvent être reçues via des notifications push ou consultées sur une page de suggestions spécifique dans l'application, offrant flexibilité et commodité.

5. Personnalisation (Optionnel) : Cette fonctionnalité permet aux utilisateurs de modifier les thèmes visuels et les filtres de recherche, renforçant ainsi l'engagement et la satisfaction utilisateur.

6. Interactions Sociales (Optionnel) : Permet aux utilisateurs de partager des recommandations et des expériences sur les réseaux sociaux, ainsi que de commenter et évaluer les lieux visités, favorisant ainsi une communauté active et engagée.

La plupart des approches actuelles de la FM sont plus ou moins directement dérivées du travail de (Czarnecki et al., 2012). Feature-Oriented Reuse Methodology (FORM — Kang et al. (1998)) est une de ces extensions.

La DM vise à capturer les décisions prises à différents niveaux de l'architecture logicielle, des exigences aux spécifications en passant par la conception et l'exécution. Les techniques de modélisation des décisions, telles que les langages formels comme DMN (Decision Model and Notation — OMG (2015)), permettent aux développeurs de décomposer les logiques de décision complexes en composants réutilisables.

Pour comparer la FM et la DM, Czarnecki et al. (2012) proposent un cadre d'évaluation basé sur plusieurs dimensions, parmi lesquelles nous sélectionnons les suivantes : leur application respective, l'unité de variabilité considérée, la prise en compte de la hiérarchie, la correspondance aux artefacts du logiciel, le moment et le mode de liaison des variations, la facilité de modularité dans la gestion de la variabilité, ainsi que l'aspect des outils disponibles pour les supporter.

L'application évalue les domaines dans lesquels chaque approche de modélisation est couramment appliquée. La FM trouve principalement son application dans l'ingénierie des lignes de produits logiciels, où elle est utilisée pour représenter et gérer la variabilité au sein des familles de produits. En revanche, la DM est largement adoptée dans divers domaines, notamment la modélisation des processus métier, l'automatisation des décisions et l'ingénierie des exigences.

L'unité de variabilité désigne la plus petite composante qui encapsule la variabilité dans le modèle. Dans le contexte de la FM, cette unité est représentée par les caractéristiques. En revanche, pour la DM, les décisions ou les règles métier remplissent ce rôle en représentant les choix ou les conditions qui influencent le comportement du système.

La hiérarchie évalue la capacité de l'approche de modélisation à représenter les relations hiérarchiques entre les entités dans le modèle. Dans le cadre de la FM, les caractéristiques soutiennent ces relations hiérarchiques, ce qui permet la représentation de

structures de caractéristiques complexes. En revanche, pour la DM, les décisions, souvent hiérarchisées, sont décomposées en sous-décisions ou conditions, ce qui permet la modélisation des dépendances et des relations entre les décisions.

La correspondance (mapping en anglais) évalue dans quelle mesure les éléments du modèle (caractéristiques ou décisions) correspondent aux artefacts tangibles dans le processus de développement logiciel. Dans le cas de la FM, les caractéristiques sont généralement associées à des artefacts logiciels tels que des modules de code, des composants ou des options de configuration. En revanche, pour la DM, les décisions sont souvent liées à des règles métier, des spécifications d'exigences ou une logique conditionnelle au sein des systèmes logiciels.

Le moment et le mode de liaison examinent quand et comment la variabilité est liée ou résolue dans le processus de développement. Dans le cadre de la FM, la variabilité est souvent liée à la compilation ou à la conception à travers une configuration explicite des caractéristiques, permettant une gestion statique de la variation. En revanche, pour la modélisation des décisions la DM, les décisions peuvent être liées à la conception ou à l'exécution, selon l'approche d'automatisation des décisions utilisée. La liaison peut être explicite via une configuration ou implicite via des moteurs de décision.

La modularité évalue le degré de support de l'approche de modélisation pour la modularisation de la variabilité, permettant une maintenance facile, une réutilisation et une composition des points de variation. Dans le cas de la FM, cette modularité est favorisée en permettant aux caractéristiques d'être organisées en modules réutilisables, facilitant ainsi la réutilisation systématique et la composition des modèles de caractéristiques. Quant à la DM, elle supporte la modularité grâce à la décomposition des décisions en composants réutilisables, ce qui permet la séparation des préoccupations et facilite la réutilisation des décisions.

L'aspect des outils évalue la disponibilité et l'efficacité des outils ainsi que le support des outils pour chaque approche de modélisation, incluant notamment l'existence d'éditeurs

spécialisés, de validateurs et de générateurs. La FM bénéficie du support d'outils spécialisés tels que FeatureIDE (Thüm et al., 2014) et pure::variants[9]. Ces outils offrent des fonctionnalités pour la création de modèles, la validation et la génération de code. De même, la DM est soutenue par divers outils, incluant les plates-formes DMN (Decision Model and Notation) comme Camunda[10] et Trisotech[11], qui fournissent des capacités pour la modélisation, la simulation et l'exécution de décisions.

Nous soutenons que, dans le cadre de la gestion de la variabilité, MDE offre une approche exceptionnelle qui répond efficacement aux dimensions du cadre d'évaluation. En effet, MDE peut être appliquée dans divers domaines, y compris la modélisation des caractéristiques et des décisions, en fournissant des techniques et des outils pour créer, analyser et transformer des modèles à différents niveaux d'abstraction. De plus, MDE prend en charge la représentation hiérarchique des modèles, permettant de définir des structures complexes et des relations entre les entités de modèle.

Les modèles en MDE peuvent être utilisés pour générer automatiquement des artefacts logiciels tels que du code source, de la documentation, des configurations, etc., en fonction des spécifications de modèle. Ainsi, les modèles peuvent être utilisés pour les caractéristiques ou les décisions aux artefacts correspondants. MDE offre la flexibilité de lier la variabilité à différents moments du processus de développement, selon les besoins du projet. Cela peut inclure la liaison à la compilation, à l'exécution ou même à la conception, et peut être réalisé via des mécanismes de configuration explicite ou de génération de code automatique à partir des modèles.

[9] https://www.pure-systems.com/purevariants

[10] https://docs.camunda.org/manual/7.20/

[11] https://www.trisotech.com/bpmn-quickguide/

MDE favorise la modularité en permettant de décomposer les modèles en sous-modèles réutilisables, ce qui facilite la gestion et la réutilisation de la variabilité à travers différents contextes et applications.

Néanmoins, le principal défi auquel est confrontée MDE réside dans le niveau de maturité des outils et des technologies qui lui sont associés[12]. Malgré les avancées significatives dans le développement d'outils de modélisation, de simulation, de génération de code et de transformation de modèles, de nombreuses solutions dans le domaine de la MDE demeurent encore relativement récentes et souffrent de lacunes en termes de fonctionnalités, de performances et de convivialité. Ainsi, il est impératif de poursuivre les efforts de recherche et de développement afin d'accroître la maturité des outils utilisés en MDE, garantissant ainsi une adoption plus large et plus efficace de cette approche.

2.3 LA RECONNAISSANCE AUTOMATIQUE DE LA PAROLE

La modélisation des SR dans notre approche est facilitée par l'intégration de la reconnaissance de commandes vocales. Cette démarche, détaillée dans le chapitre 3, comprend deux étapes essentielles : la transcription automatique de la parole (speech-to-text en anglais) à l'aide d'un ASR, et la détection d'intention pour anticiper la commande. Cette section explore ces deux aspects.

2.3.1 La tâche de transcription automatique de la parole

Dans le domaine de la reconnaissance automatique de la parole, plusieurs modèles et approches jouent un rôle central. Ces techniques visent à transcrire des signaux acoustiques en séquences de symboles, permettant ainsi aux machines de comprendre et d'interpréter le langage humain. Parmi ces modèles clés, la tâche de transcription de l'enregistrement audio est réalisée grâce à divers modèles, méthodes et algorithmes, tels que les modèle de Markov

[12] Ce point est souvent perçu comme l'obstacle principal entravant l'adoption à grande échelle de MDE dans l'industrie (Hutchinson et al., 2011).

caché (HMM – Rabiner et Juang (1986)), la déformation temporelle dynamique (DTW Dynamic Time Warping – Sakoe et Chiba (1978)), les réseaux neuronaux et la reconnaissance automatique de la parole de bout-en-bout (E2E ASR end-to-end – (Hannun et al., 2014; Chorowski et al., 2014; Bahdanau et al., 2016); Li (2022)).

2.3.1.1 Le modèle de Modèle de Markov caché (HMM)

Les modèles de Markov caché (HMM – Rabiner et Juang (1986)) sont des modèles statistiques capables de générer une séquence de symboles ou de quantités. Dans le contexte de la reconnaissance de la parole, ils sont privilégiés en raison de la nature stationnaire de la parole sur de courtes périodes, généralement égales à 10 millisecondes. Les HMM peuvent être automatiquement entraînés et sont relativement simples à utiliser. Ils produisent des vecteurs réels multidimensionnels, appelés coefficients cepstraux (Vergin et al., 1999), qui capturent les caractéristiques de la parole. Chaque mot ou phonème a une distribution de sortie distincte, et les HMM sont concaténés pour l'entraînement des séquences de mots ou de phonèmes.

La reconnaissance de la parole moderne intègre diverses techniques pour améliorer les résultats par rapport à l'approche de base reposant sur l'utilisation de modèles HMM. Cela inclut la dépendance contextuelle pour les phonèmes, la normalisation cepstrale (Vergin et al., 1999) afin d'adapter la parole à différents locuteurs et conditions d'enregistrement, ainsi que des techniques telles que la normalisation de la longueur du tractus vocal (VTLN Vocal Tract Length Normalization – Eide et Gish (1996)) pour égaliser entre les voix féminines et masculines, et la régression linéaire par maximum de vraisemblance (MLLR Maximum Likelihood Linear Regression – Leggetter et Woodland (1995)) pour une adaptation plus générale au locuteur. Les caractéristiques de la parole comprennent des coefficients capturant la dynamique de la parole, ainsi que l'analyse discriminante linéaire hétéroscedastique (HLDA Heteroscedastic Linear Discriminant Analysis – Kumar et Andreou (1998)). Le décodage utilise l'algorithme de Viterbi (Viterbi, 1967; Forney, 1973) pour trouver le meilleur chemin, pouvant inclure la création dynamique d'un modèle HMM combiné

intégrant informations acoustiques et modèle de langage. Une amélioration potentielle du processus du décodage consiste à conserver plusieurs candidats et à utiliser une fonction de notation (rescoring) pour choisir la meilleure transcription, avec la distance de Levenshtein comme fonction de perte courante, permettant ainsi d'obtenir des systèmes de reconnaissance de la parole plus performants (Rabiner, 1989; Satish & Gururaj, 1993; Newberg, 2009).

2.3.1.2 La déformation temporelle dynamique (DTW)

La DTW (Dynamic Time Warping – Sakoe et Chiba (1978)) est un algorithme mesurant la similarité entre deux séquences qui peuvent varier dans le temps ou la vitesse. Elle a trouvé des applications dans l'ASR pour traiter différentes vitesses de parole. En comparant des séquences de longueurs différentes, la DTW permet d'aligner les séquences en considérant la flexibilité temporelle, déformant les séquences de manière non linéaire pour les faire correspondre. Par exemple, elle peut aligner le mot "schedule" prononcé avec un son "sk" en anglais américain et avec un son "sh" en anglais britannique. La DTW est souvent utilisée en conjonction avec les HMM pour résoudre ce problème d'alignement de séquences (Fang, 2009; Bringmann & Künnemann, 2015).

2.3.1.3 Les réseaux neuronaux

À la fin des années 1980, les réseaux neuronaux ont été appliqués dans divers domaines de la reconnaissance vocale, tels que la classification des phonèmes (Graves & Schmidhuber, 2005), la reconnaissance de mots isolés (Lamel et al., 1981), la reconnaissance audiovisuelle des locuteurs et l'adaptation des locuteurs (Maison et al., 1999). Contrairement aux HMM, les réseaux neuronaux reposent sur moins d'hypothèses explicites sur les propriétés statistiques des caractéristiques, ce qui les rend attrayants en tant que modèles d'ASR. Lorsque que les réseaux neuronaux sont utilisés pour l'estimation des probabilités d'un segment de caractéristiques vocales, ils permettent un entraînement discriminatif de manière naturelle et efficiente. Cependant, malgré leur efficacité pour classer des unités de courte durée comme les phonèmes individuels et les mots isolés, les premiers réseaux neuronaux

ont rarement réussi dans les tâches de reconnaissance continue en raison de leur capacité limitée à représenter les dépendances temporelles. Une méthode pour faire face à cette limitation a été l'utilisation des réseaux neuronaux en tant qu'étape de prétraitement, de transformation des caractéristiques ou de réduction de la dimensionnalité avant la reconnaissance basée sur les HMM. Cependant, récemment, les réseaux de neurones récurrents (Recurrent Neural Network RNN – (Sainath et al., 2020; Li et al., 2020)) (avec leurs variantes telles que les LSTM (Long Short-Term Memory – Lee et al. (2019)), les réseaux neuronaux à retard temporel (Time-Delay Neural Networks – Waibel et al. (2013)), ainsi que les transformateurs (Chan et al., 2015), ont montré des performances améliorées dans ce domaine.

2.3.1.4 Les approches de reconnaissance de la parole de bout-en-bout

Les modèles de reconnaissance de la parole de bout-en-bout (end-to-end ASR – (Hannun et al., 2014; Chorowski et al., 2014; Bahdanau et al., 2016) ont suscité un intérêt significatif depuis 2014, offrant une approche plus intégrée en apprenant simultanément tous les composants de l'ASR, tels que la prononciation, l'acoustique et le modèle de langage. Comparé aux approches traditionnelles basées sur la phonétique avec des modèles acoustiques cachés (HMM), cette méthode simplifie l'entraînement et le déploiement des systèmes ASR. En particulier, cela se traduit par des avantages en termes de ressources mémoire, rendant ces modèles plus adaptés au déploiement local sur des appareils mobiles. Dans la suite de cette section, nous examinerons l'évolution de ces modèles.

a) *LES MODÈLES BASÉS SUR LA CLASSIFICATION TEMPORELLE CONNEXIONNISTE*

La première tentative de l'ASR de bout-en-bout (Graves & Jaitly, 2014) a été réalisée avec les systèmes basés sur la classification temporelle connexionniste (CTC – Graves et al. (2006)) en 2014. Le modèle était composé de RNN couplé à une couche CTC. Ainsi, le modèle entraîné (RNN-CTC) apprenait simultanément la prononciation et le modèle acoustique. Toutefois, il était limité car ne pouvant pas apprendre la langue en raison

d'hypothèses d'indépendance conditionnelle semblables à celles des HMM. En conséquence, les modèles CTC pouvaient apprendre directement à associer les caractères anglais à partir de l'acoustique de la parole, mais généraient des erreurs d'orthographe courantes. Ils dépendaient sur un modèle de langage distinct dans le but de corriger les transcriptions. Par la suite, Baidu[13] a étendu cette approche en utilisant des ensembles de données extrêmement volumineux, obtenant ainsi un succès commercial important en chinois mandarin et en anglais (Amodei et al., 2016). En 2016, l'université d'Oxford a présenté à son tour LipNet[14] qui fut le premier modèle de lecture labiale au niveau des phrases de bout-en- bout. Le modèle en question utilise des convolutions spatiotemporelles (Tran et al., 2018) associées à une architecture RNN-CTC, dépassant les performances humaines dans un ensemble de données de grammaire restreinte (Assael et al., 2016). Enfin, en 2018, Google DeepMind[15] a amélioré l'architecture en ajoutant un composant de réseau de neurones convolutif (Convolutional Neural Networks CNN – Agarap (2017)) en dévoilant une architecture CNN-RNN-CTC à grande échelle, qui atteint des performances 6 fois supérieures à celle des experts humains (Shillingford et al., 2018).

b) *LES MODÈLES BASÉS SUR L'ATTENTION*

Les modèles basés sur l'attention (Bahdanau et al., 2014), tels que le LAS (Listen, Attend and Spell – Chan et al. (2015)), présentent une alternative aux modèles basés sur la CTC. En écoutant littéralement le signal acoustique, en accordant une attention particulière à différentes parties du signal et en épellant la transcription caractère par caractère, ces modèles évitent les hypothèses d'indépendance conditionnelle. Contrairement aux modèles CTC, ils peuvent intégrer directement tous les composants d'un système de reconnaissance vocale, y compris la prononciation, le modèle acoustique et le modèle de langage. Cette

[13] https://www.baidu.com/

[14] https://lipnet.ai/

[15] https://www.deepmind.com/

approche offre l'avantage de ne pas nécessiter le transport d'un modèle de langue lors du déploiement, ce qui la rend particulièrement pratique pour les applications avec des contraintes de mémoire. À la fin de 2016, les modèles basés sur l'attention ont rencontré un succès considérable, surpassant même les modèles CTC, qu'ils soient associés ou non à un modèle de langue externe (Chorowski & Jaitly, 2016).

Depuis la création du modèle LAS initial, plusieurs extensions ont été proposées. La décomposition de séquences latentes (Latent Sequence Decompositions LSD – Chan et al. (2016)) a été introduite pour émettre directement des unités sub-lexicales[16], plus naturelles que les caractères individuels. Une autre extension, Conformer (Convolution-augmented Transformer for Speech – (Gulati et al., 2020), intègre une attention autorégressive pour capturer des relations temporelles plus complexes, améliorant ainsi les performances de génération de texte dans des contextes spécifiques. Conformer intégre des blocs de convolution en plus des blocs de transformateurs (Vaswani et al., 2017) , visant à capturer efficacement les caractéristiques à différentes échelles temporelles et spatiales dans les séquences audio, ce qui est crucial pour la reconnaissance de la parole.

De plus, le modèle LAS a été étendu pour créer le modèle WLAS (Watch, Listen, Attend and Spell – Son Chung et al. (2017)), capable de traiter la lecture labiale. Ce modèle combine des informations visuelles (Watch) et auditives (Listen), utilisant un mécanisme d'attention (Attend) pour générer des transcriptions textuelles (Spell).

c) *LES MODÈLES BASÉS SUR L'APPRENTISSAGE PAR TRANSFERT*

L'apprentissage par transfert (transfert learning en anglais) est une technique qui utilise les connaissances d'un modèle pré-entraîné pour apprendre un autre ensemble de données. Son objectif est d'améliorer l'apprentissage dans le domaine cible en exploitant les connaissances du domaine source et de la tâche d'apprentissage. Différents paramètres

[16] Des éléments plus petits que les lexèmes et peuvent inclure des morphèmes, des préfixes, des suffixes et d'autres composants linguistiques qui sont en dessous du niveau des mots complets.

d'apprentissage par transfert sont définis en fonction du type de tâche et de la nature des données disponibles dans les domaines source et cible (Bhoi et al., 2022). En reconnaissance vocale, cette approche offre des avantages significatifs en cas de données d'entraînement étiquetées limitées ou lors de l'adaptation à de nouveaux domaines ou langues. Le processus se divise généralement en deux étapes : le pré-entraînement (pre-training en anglais), où le modèle apprend sur un ensemble de données volumineux et étiqueté, et l'affinage (fine-tuning en anglais), où il se spécialise sur un ensemble de données plus restreint et spécifique à la tâche ou au domaine cible.

Comme exemples de modèles ASR basés sur l'apprentissage par transfert, citons ceux fondés sur l'architecture des transformateurs (Vaswani et al., 2017), tels que BERT, ainsi que des modèles basés sur des RNN ou des CNN, comme DeepSpeech.

Le modèle BERT (Bidirectional Encoder Representations from Transformers – Devlin et al. (2019)), qui repose sur des transformateurs, a révolutionné les tâches de traitement du langage naturel et a trouvé une application remarquable en reconnaissance vocale. Les modèles basés sur BERT tirent avantage de l'apprentissage par transfert, acquérant des représentations contextuelles à partir d'un vaste corpus de données vocales non étiquetées. Pendant la phase de pré-entraînement, ces modèles se perfectionnent en prédisant des segments masqués de la parole, capturant ainsi une riche information contextuelle. L'architecture typique de ces modèles comprend un CNN ou un encodeur basé sur un transformateur, suivi d'un quantificateur et d'un modèle de langage basé sur un transformateur. Les modèles pré-entraînés basés sur BERT peuvent être affinés sur des ensembles de données étiquetés plus petits pour atteindre une grande précision dans diverses tâches de reconnaissance vocale (Cantu, 2023).

Dans la même veine d'utilisation de l'apprentissage par transfert pour améliorer les performances dans le domaine de la reconnaissance vocale, wav2vec 2.0 (Baevski et al., 2020), un modèle basé sur BERT, suit un processus d'entraînement en deux étapes. Pendant

la première phase, un modèle auto-supervisé est entraîné à l'aide d'une fonction contrastive[17] qui prédit les trames masquées à l'intérieur d'un segment vocal d'entrée. Dans la seconde phase, un modèle acoustique basé sur un transformateur est entraîné en utilisant une perte CTC sur un ensemble de données étiqueté. Cette phase d'affinage permet au modèle de s'adapter à la tâche spécifique visée, garantissant des performances de pointe en tant que modèle ASR (Schneider et al., 2019; Baevski et al., 2020; Cantu, 2023).

DeepSpeech (Hannun et al., 2014), l'ASR de bout-en-bout développé par Mozilla[18], a également adopté les techniques d'apprentissage par transfert. Il a été entraîné sur des ensembles de données à grande échelle, tels que LibriSpeech[19] (Panayotov et al., 2015), comprenant des livres audio, et CommonVoice[20] (Ardila et al., 2019), qui est un ensemble de données participatives avec des personnes lisant des phrases. DeepSpeech utilise des techniques d'apprentissage profond, en particulier des CNN et RNN, pour traiter et analyser les données audio.

2.3.2 La tâche de détection d'intention/de commande

Selon Lin et al. (2023), la détection d'intention est un élément clé des systèmes de dialogue orientés vers les tâches, car elle permet d'identifier l'intention de l'utilisateur à partir d'un énoncé. Cette tâche est généralement effectuée dans la composante NLU. Les approches initiales de détection d'intention reposaient principalement sur des règles spécifiques prédéfinies. Ces approches utilisent un ensemble de règles généralement créées manuellement en se basant sur des connaissances spécifiques du domaine ou de modèles prédéfinis pour faire correspondre l'entrée de l'utilisateur à une intention spécifique. Par exemple, un système basé sur des règles permet de rechercher des mots-clés tels que

[17] Une fonction mathématique ou algorithmique qui mesure la dissimilarité ou la différence entre deux éléments.

[18] https://www.mozilla.org/

[19] https://www.openslr.org/12

[20] https://commonvoice.mozilla.org/fr

« commander » ou « acheter » pour déterminer que l'intention de l'utilisateur est d'effectuer une réservation. Tian et al. (2020) soutiennent que la méthode de détection d'intention par des règles permet l'obtention d'une meilleure précision de reconnaissance pour les besoins présentant une forte régularité et l'extraction des informations précises. Cependant, cette méthode nécessite également une plus grande participation humaine pour la formulation des règles.

Avec l'avènement de l'apprentissage automatique, les techniques de classification supervisée ont été employées pour la détection d'intention. En général, ces approches exigent l'utilisation de jeux de données annotés, où chaque requête est associée à une intention spécifique. Différents algorithmes de classification tels que les forêts d'arbres décisionnels (random forest en anglais – Biau et Scornet (2016)), les RNN et les machines à vecteurs de support (SVM Support Vector Machine – Tang (2013)) sont utilisés pour entraîner des modèles capables de prédire l'intention d'une requête donnée. Khan et Meenai (2021) soulignent que les approches de détection d'intention basées sur l'apprentissage automatique nécessitent l'extraction de caractéristiques clés à partir d'un texte volumineux. L'extraction manuelle de ces caractéristiques entraîne un coût très élevé sans fournir des résultats précis en termes de caractéristiques sélectionnées. Ces approches ne parviennent pas à saisir le contexte réel des données textuelles brutes, compte tenu de la nature informelle de ces données. Dans ce contexte, certains chercheurs explorent des techniques avancées telles que les réseaux de neurones profonds avec des architectures comme les CNN et les RNN. L'utilisation de BERT, comme décrit dans notre approche détaillée au cinquième chapitre, met l'accent sur les parties importantes des requêtes, améliorant ainsi significativement la performance de la détection d'intention.

Selon Casanueva et al. (2020), bien que la détection d'intention joue un rôle crucial dans les systèmes conversationnels orientés vers des tâches, les jeux de données publics pour ces tâches restent rares. Les ensembles de données standards précédents, tels que Ask

Ubuntu[21] et SNIPS[22] Coucke et al. (2018), se limitent à un petit nombre de classes (<10), ce simplifiant exagérément la tâche et ne reflètant pas fidèlement l'environnement réel des systèmes commerciaux. Ainsi, des travaux plus récents ont reconnu le besoin de jeux de données améliorés pour la détection d'intention :

1. Le jeu de données de X. Liu et al. (2019), appelé HWU64[23], comprend 25 716 exemples pour 64 intentions réparties dans 21 domaines tels que la santé, les voyages, la finance et l'éducation, etc.
2. Le jeu de données de Larson et al. (2019), appelé CLINC150[24], couvre 150 intentions et 23 700 exemples répartis dans 10 domaines, incluant la banque, la réservation de vols, les loisirs, etc.
3. Le jeu de données de Budzianowski et al. (2018), nommé MultiWOZ[25], est une collection entièrement étiquetée de conversations écrites entre humains couvrant plusieurs domaines tels que la réservation de restaurants, la réservation d'hôtels, la réservation de taxis, etc., totalisant 10 000 dialogues répartis sur 7 domaines.

Cependant, ces jeux de données récents, bien que multi-domaines, pourraient ne pas suffisamment refléter la complexité complète de chaque domaine telle qu'elle est rencontrée dans des scénarios du monde réel. Afin de compléter les efforts récents de collecte de données pour la détection d'intention, un ensemble de jeux de données mono-domaine appelé BANKING77[26] a été proposé par Casanueva et al. (2020). Cet ensemble fournit une série très détaillée d'intentions dans le domaine bancaire, non inclus dans HWU64 et CLINC150.

[21] https://github.com/taolei87/askubuntu

[22] https://paperswithcode.com/dataset/snips

[23] https://github.com/xliuhw/NLU-Evaluation-Data

[24] https://github.com/clinc/oos-eval

[25] https://github.com/budzianowski/multiwoz

[26] https://huggingface.co/datasets/banking77

Il comprend 13 083 requêtes de service client étiquetées avec 77 intentions. Sa focalisation sur la détection d'intention mono-domaine le rend complémentaire aux deux autres ensembles de données.

Dans le contexte de la reconnaissance vocale, l'expression « détection de commandes » est privilégiée, mettant l'accent sur l'identification précise des actions ou demandes spécifiques énoncées verbalement par l'utilisateur. Cela est observé notamment dans les assistants virtuels tels que Siri d'Apple[27], Alexa d'Amazon[28], Google Assistant[29] ou Cortana de Microsoft[30].

La détection de commande avec Siri commence par l'activation de Siri en prononçant la phrase « Hey Siri. » Un détecteur vocal en continu écoute ces deux mots et utilise des réseaux de neurones profonds (DNN) pour convertir le motif acoustique en une distribution de probabilité. En intégrant temporellement ces probabilités, un score de confiance est calculé. Si le score est élevé, Siri s'active pour comprendre et interpréter les commandes de l'utilisateur, utilisant des techniques de NLP comme l'analyse syntaxique. Les informations sont envoyées aux serveurs d'Apple Siri pour une analyse plus approfondie, incluant des modèles linguistiques et des probabilités pour déterminer l'intention de l'utilisateur. La transformation vocale a évolué, initialement basée sur l'analyse discriminante linéaire (ADL) [31] et ensuite améliorée avec des données d'enrôlement explicites et l'adoption de DNN non linéaires (Saini, 2019).

[27] https://www.apple.com/ca/fr/siri/

[28] https://alexa.amazon.com/

[29] https://assistant.google.com/

[30] https://www.microsoft.com/en-us/cortana

[31] Technique sous contrainte (canonique) qui divise une matrice de réponse en groupes en fonction d'un facteur en trouvant la combinaison de variables qui donne la meilleure séparation possible entre les groupes. Le regroupement est effectué en maximisant la dispersion entre les groupes par rapport à la dispersion à l'intérieur des groupes.

La détection de commande avec Alexa s'articule autour d'un processus en plusieurs étapes. Après le traitement du signal pour éliminer les bruits, Alexa réalise la « détection du mot de réveil » en recherchant un mot d'activation (wake word en anglais) en cherchant un mot d'activation, généralement « Alexa » ou « Hey Alexa ». Suiv par l'activation du nom d'invocation spécifique pour lancer une compétence particulière. L'énoncé de l'utilisateur représente ensuite ce qui est demandé, et Alexa identifie l'intention associée pour déterminer l'action à exécuter. L'approche de présélection et de réorganisation neuronale en deux étapes utilisée par Alexa implique un modèle initial, Shortlister, pour un classement rapide des intentions, suivi du « Hypotheses Reranker » (HypRank – Khan et al. (2015)) pour affiner et trouver la meilleure intention pour traiter la requête (Kim et al., 2018).

La détection de commande avec Google Assistant commence par la transcription des requêtes vocales à l'aide de l'API Google Speech-to-Text[32], qui prend en charge environ 120 langues et dialectes. Après la transcription, l'extraction des entités est effectuée pour récupérer des informations telles que la date, l'heure, le lieu, le nom, le numéro de téléphone, etc. Ces entités sont utilisées pour assigner des actions appropriées, comme passer un appel. Ensuite, l'intention de la requête est identifiée à l'aide d'un modèle d'apprentissage automatique tel que BERT, facilitant la génération de dialogues et la détermination de l'action à entreprendre. Enfin, la réponse est générée en fonction de l'entrée et de l'intention de la requête (Gupta, 2020).

Cortana s'appuie sur l'API de traduction vocale de Microsoft, un service infonuagique pour la traduction automatique, utilisant Bing[33] comme moteur de recherche sur le web, développé en ASP.NET[34]. Les capacités de traitement naturel de Cortana, sont couplées à une base de données de recherche sémantique appelée Satori. La reconnaissance de la parole utilise des algorithmes comme DTW (voir section 2.3.1.2) pour mesurer la similarité entre

[32] https://cloud.google.com/speech-to-text?hl=fr

[33] https://www.bing.com/

[34] https://dotnet.microsoft.com/en-us/apps/aspnet

les séquences temporelles, tandis que l'interprétation sémantique inclut des vérifications de la grammaire et des propriétés sémantiques pour déterminer la réponse appropriée à l'utilisateur. Les erreurs sont corrigées, le sens des mots est extrait, et la tâche demandée est exécutée, générant une réponse sous forme de parole ou de texte après accomplissement de la tâche (Bhat et al., 2017).

Les assistants virtuels tels que GPT-3, GPT-3.5 Turbo et GPT-4, développés par OpenAI, représentent une avancée significative dans la génération autonome de texte. Modèles de langage comme ChatGPT[35] (GPT pour Generative Pre-trained Transformer – Han et al. (2021)), qui s'appuient sur ces technologies, utilisent une architecture de transformateur et intègrent l'apprentissage par renforcement dans le processus de réglage fin. Pré-entraîné sur d'énormes ensembles de données, ChatGPT génère un langage cohérent, contextuellement approprié, et souvent indiscernable de ce qui est écrit par des humains. L'apprentissage par renforcement guide le modèle à produire des réponses plus utiles, informatives ou pertinentes dans le contexte de l'interaction utilisateur (Guinness, 2023).

Il est important de souligner l'existence de jeux de données publics dédiés à la classification des commandes vocales, couvrant divers domaines tels que la domotique, les maisons connectées et les véhicules. Ces ensembles de données jouent un rôle fondamental dans le développement et l'évaluation de systèmes de reconnaissance vocale, stimulant ainsi la recherche et l'innovation dans le domaine de l'intelligence artificielle vocale.

Le CI-AVSR[36] (Cantonese Audio-Visual Speech Dataset for In-car Command Recognition – Dai et al. (2022)) représente une avancée significative dans la reconnaissance des commandes en voiture pour la langue cantonaise. En intégrant des données vidéo et audio, ce jeu de données de 4 984 échantillons provenant de 30 locuteurs natifs du cantonais offre une base riche pour le développement de systèmes avancés de compréhension et de

[35] Un modèle d'apprentissage profond pré-entraîné sur d'énormes corpus de données textuelles, basé sur l'architecture de Transformateur, peut être affiné pour des tâches spécifiques comme la génération de langage, l'analyse des sentiments, la modélisation du langage, la traduction automatique et la classification du texte.

[36] https://github.com/HLTCHKUST/CI-AVSR

réponse aux commandes vocales dans des contextes automobiles spécifiques. Les catégories générales couvrent des domaines tels que la navigation, la lecture de musique et les prévisions météorologiques. Un exemple de commande pourrait être « Emmène-moi à [location] » dans la catégorie navigation.

Le jeu de données FSC[37] (Fluent Speech Commands – Lugosch et al. (2019)) offre une ample collection de 30 043 énoncés enregistrés par 97 locuteurs, chacun représentant des commandes destinées à contrôler des appareils domestiques intelligents ou des assistants virtuels. Les enregistrements, au format .wav mono à 16 kHz, couvrent une diversité d'instructions telles que « mettre de la musique » ou « augmenter la chaleur dans la cuisine ». Chaque énoncé est étiqueté avec des valeurs « action », « objet » et « emplacement », permettant ainsi d'entraîner des modèles à prédire ces informations. Cette richesse d'annotations et la flexibilité du jeu de données en font une ressource précieuse pour diverses expérimentations dans le domaine de la reconnaissance des commandes vocales.

Les jeux de données Grabo (Renkens et al., 2014), Domotica (Tessema et al., 2013; Ons & Gemmeke, 2014) et Patcor (Tessema et al., 2013) sont trois ensembles de données liés contenant des commandes vocales pour le contrôle de robots et des jeux de cartes. Ils ont été développés par l'Université de Louvain[38] et ont été utilisés dans Renkens (2018).

Les jeux de données Grano (Renkens et al., 2014), Domotica (Tessema et al., 2013; Ons & Gemmeke, 2014) et Patcor (Tessema et al., 2013), développés par l'Université de Louvain[39] et utilisés dans Renkens (2018), constituent une collection complète d'ensembles dédiés à l'étude des commandes vocales pour le contrôle de robots et des jeux de cartes.

1. Le jeu de données Grabo :
    - Commandes en anglais et en néerlandais destinées à un robot.

[37] https://fluent.ai/fluent-speech-commands-a-dataset-for-spoken-language-understanding-research/

[38] https://www.esat.kuleuven.be/psi/spraak/downloads/

[39] https://www.esat.kuleuven.be/psi/spraak/downloads/

- Exemples de commandes : « déplace-toi à la position x » ou « attrape l'objet y ».
- 30 libellés de sortie incluant des positions dans le monde, des actions du robot, etc.
- Enregistrements de 11 locuteurs émettant 36 commandes différentes avec 15 répétitions.

2. Jeux de données Pactor :
   - Énoncés en néerlandais issus d'un jeu de cartes guidé par la voix (« Patience »).
   - Commandes typiques : « placez la carte x sur la carte y » ou « nouvelles cartes ».
   - 38 libellés de sortie comprenant la valeur et la couleur de la carte déplacée, la position cible, etc.
   - Enregistrements de 8 locuteurs.
3. Jeux de donnée Domotica :
   - Énoncés de locuteurs néerlandais utilisant des commandes vocales pour l'automatisation domestique.
   - Exemples de commandes : « ouvrir la porte x » ou « allumer la lumière y ».
   - 25 libellés de sortie couvrant l'ensemble des lumières, des portes et des actions du système.
   - Enregistrements de 17 locuteurs présentant différents niveaux de dysarthrie, avec une variabilité de données par locuteur.

Warden (2018) a introduit une version améliorée du jeu de données Google Speech Commands Dataset (GSCD) sous le nom de Speech Commands v2 (SC) Google (2018). Cette version actualisée comprend 105 829 extraits audio d'une seconde au format « wav », échantillonnés à 16 kHz. Elle englobe 35 mots distincts répartis en catégories avec des fréquences variées, introduisant également davantage de variations dans le bruit de fond. Les mots couvrent des commandes pertinentes pour les applications d'Internet des objets ou robotiques, tels que "Yes", "No", "Up", "Down", "Left", "Right", "On", "Off", "Stop", les

chiffres zéro à neuf, ainsi que des noms de personnes et de lieux. Cette version se concentre sur la langue anglaise, mais les auteurs recommandent l'utilisation de techniques telles que l'apprentissage par transfert pour élargir l'applicabilité du jeu de données à d'autres langues.

Ces dernières années, l'avènement des DNN a considérablement amélioré la précision des modèles de reconnaissance vocale, permettant des transcriptions précises même dans des contextes difficiles. Bien que la conversion de la parole en texte ait progressé, la compréhension du langage parlé (SLU) reste un défi. Alors que certaines approches se concentrent sur la SLU de bout en bout, notre méthode préconise la conversion préalable de la parole en texte avant de prédire l'intention associée. Des recherches telles que celles de Qian et al. (2017), Serdyuk et al. (2018), et Chen et al. (2018) explorent différentes approches, tandis que Lugosch et al. (2019) utilisent l'ensemble de données FSC et proposent une stratégie de pré-entraînement pour améliorer les performances de la SLU de bout en bout, démontrant des améliorations sur des ensembles d'entraînement de tailles variées. Des pistes futures incluent l'exploration des limites de la SLU de bout en bout, notamment face à de nouvelles formulations et synonymes non observés dans les ensembles de données, pour évaluer la capacité à surmonter ces défis (Lugosch et al., 2019; Fluent.ai, 2020; Avila et al., 2023).

2.4 SYNTHÈSE

L'accélération du développement des SR n'est pas nouvelle, comme en témoignent les travaux mentionnés dans ce chapitre qui explorent le MDE comme base de l'approche proposée. Des études telles qu'Almonte et al. (2020) et de Di Sipio et al. (2020) ont exploré l'utilisation de plates-formes à faible code, intégrant des métamodèles et des DSL pour configurer divers aspects du développement des SR. Cependant, le MDE présente des inconvénients, comme la complexité dans la gestion des modèles et le besoin de compétences techniques avancées. Notre approche fusionne les usines de logiciels et le MDE pour automatiser efficacement les tâches dans les processus de développement. Nous explorons également l'utilisation des ASR pour modéliser les SR à l'aide de commandes vocales.

L’apprentissage par transfert, en exploitant des modèles pré-entraînés tels que DeepSpeech et wav2vec 2.0, améliore les performances des ASR, ouvrant la voie à des améliorations continues. Notre perspective inclut l’utilisation des transcriptions générées avec wav2vec 2.0 comme entrée pour BERT afin d’effectuer la détection d’intention, profitant de représentations linguistiques plus riches et contextualisées par rapport à l’approche de bout en bout qui montre ses limites dans ce domaine.

CHAPITRE 3
L'USINE DE LOGICIELS PROPOSÉE

Dans cette étude, notre objectif est de mettre en place une usine de logiciels spécialisée dans le développement d'applications mobiles pour le tourisme intelligent. Pour ce faire, nous préparons les actifs de l'usine, telles que les patrons de conception, les cadres d'application, les transformations de modèle et les DSL. La gestion de la variabilité reste un défi que nous abordons en adoptant MDE. Cette intégration de MDE dans nos usines de logiciels nous permet de développer efficacement des applications mobiles personnalisées pour le tourisme intelligent. Nous illustrons notre approche à travers le développement d'une petite application mobile, mettant en lumière les actifs impliqués dans le développement des SR. Les sections suivantes détaillent les composants de notre usine, allant du métamodèle des SR à l'utilisation d'outils pour la mise en œuvre. Nous présentons également des modèles de transformations basés sur des architectures d'application telles que NReco.Recommender (NReco, 2023) et la Clean Architecture (Martin, 2017), choisie comme architecture de référence côté serveur. Enfin, la dernière section expose notre approche pour la classification des commandes vocales, intégrant la reconnaissance automatique de la parole parmi les actifs de notre usine de logiciels.

Remarquons que l'architecture côté client (front end) adoptée, suivant le patron MVVM (Anderson, 2012), ne fera pas l'objet de discussion dans ce mémoire.

3.1 PRÉSENTATION GÉNÉRALE

Rappelons que notre étude s'inscrit dans le cadre d'un vaste projet de recherche visant à développer une usine de logiciels pour le développement d'applications dédiées au tourisme intelligent. La figure 3.1 offre une vue simplifiée des actifs de cette usine de logiciels et de leur utilisation pour développer une instance de cette famille de produits. Elle met en

évidence les actifs développés dans cette étude, ainsi que ceux développés ou sont actuellement en développement par d'autres membres de l'équipe de recherche.

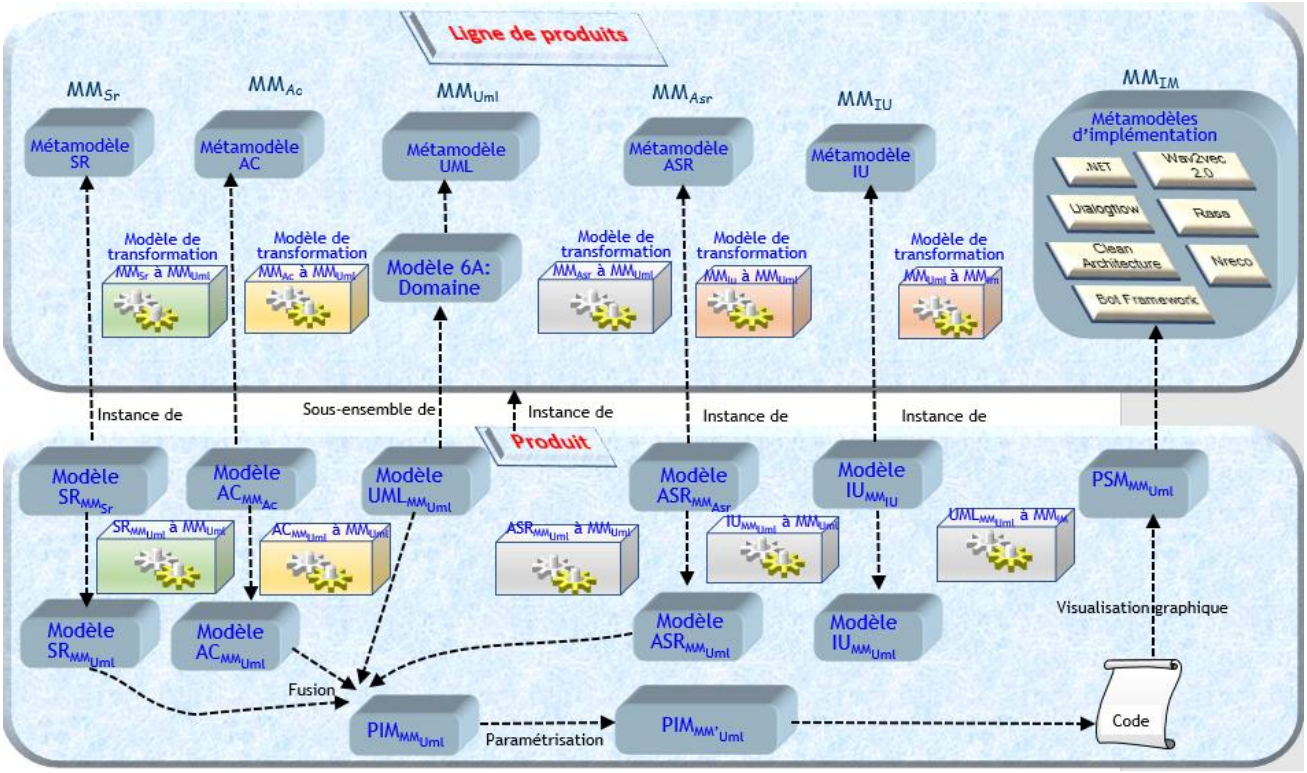


Figure 3.1 – Vue d'ensemble de notre approche.

Les actifs constitutifs de la base de notre usine de logiciels sont les suivants :

1. Le DSL **MM_{Sr}**, conçu pour le développement de systèmes de recommandation (SR), décrit les fonctionnalités de recommandation à un niveau d'abstraction supérieur. Il permet de faciliter l'intégration de mécanismes de recommandation au sein des applications mobiles, offrant une personnalisation adaptée aux préférences de l'utilisateur.

2. Le DSL **MM_{Ac}**, conçu pour le développement d'agents conversationnels (AC), décrit les agents à un niveau d'abstraction supérieur. Il permet de concevoir des agents interactifs fournissant des informations aux utilisateurs sur les destinations.

3. Le modèle de domaine **6A**, représenté par un diagramme de cas d'utilisation et un diagramme de classes d'UML, rassemble les différents concepts du domaine du tourisme intelligent pour fournir une description la plus complète possible du domaine. Ce modèle est décrit en ANNEXE I.

4. Le modèle $\mathbf{MM_{Asr}}$, basé sur un modèle de reconnaissance automatique de la parole (ASR), permet de modéliser les caractéristiques des SR à l'aide de commandes vocales.

5. Le DSL $\mathbf{MM_{IU}}$, conçu pour la description de l'interface utilisateur, facilite la conception de l'interface utilisateur indépendamment de la technologie à utiliser lors de l'implémentation.

6. Un ensemble prêt à l'emploi de cadres d'application $\mathbf{MM_{IM}}$ utiles pour l'implémentation des applications mobiles.

Dans la suite, nous démontrons l'application concrète des actifs développés ou utilisés dans cette étude. En utilisant l'actif $\mathbf{MM_{Sr}}$, un analyste marketing peut décrire les fonctions de recommandation à intégrer dans l'application mobile. À titre d'exemple, la figure 3.2 la nécessité de fonctions de recommandation pour les attractions (*GetSimilarPreferredAttractions* et *GetAttractionsWithSameUserProfile*), les hôtels (*GetHotelsWithSameUserProfile*) et les événements (*GetSimilarPreferredEvents*).

Le type de fonction de recommandation, configuré par l'analyste, peut être basé soit sur l'utilisateur (user-based en anglais), soit sur l'élément (item-based en anglais). Suite à cette phase de configuration, un modèle nommé $\mathbf{SR_{MM_{Sr}}}$ est généré. Ce modèle est ensuite transformé en un modèle $\mathbf{SR_{MM_{Uml}}}$, fusionné avec le modèle $\mathbf{UML_{MM_{Uml}}}$, un sous ensemble du modèle de domaine $\mathbf{6A}$[40], pour obtenir le PIM $\mathbf{PIM_{MM_{Uml}}}$.

[40] Ce sous-modèle est le résultat de la sélection des fonctionnalités à implanter.

Par la suite, $\mathbf{PIM}_{\mathbf{MM_{Uml}}}$ est paramétré pour donner $\mathbf{PIM'}_{\mathbf{MM_{Uml}}}$, comme illustré dans la figure 3.3. Cette étape, dirigée par un développeur, consiste à spécifier les algorithmes de recommandation nécessaires pour la configuration des fonctions de recommandation, incluant les mesures de similarité et de voisinage le plus proche, ainsi que les différentes décisions de conception liées à la spécification de l'architecture (dans notre cas, nous utilisons la clean architecture (voir section 3.3)).

$\mathbf{PIM'}_{\mathbf{MM_{Uml}}}$ est basé sur $\mathbf{UML}_{\mathbf{MM_{Uml}}}$, qui est une instance du modèle de domaine 6A. Outre le regroupement des classes nécessaires à la configuration des fonctions de recommandation (*Event*, *Hotel*, *Attraction*), ainsi que les classes dont elles héritent (*Amenity*, *Location*, *Activity*) qui, à leur tour, héritent de la classe *A_Object*, ce modèle intègre également des classes essentielles pour l'utilisation de l'application mobile développée à titre d'exemple (voir figure 3.3).

Parmi celles-ci, on trouve notamment des classes dédiées à des fonctionnalités telles que la création d'un compte utilisateur (*User*, *Account*), qui fait appel aux classes *UserAddress* pour la gestion de l'adresse de l'utilisateur, *UserAbility* pour ses compétences, *UserInterest* pour ses centres d'intérêt, *UserPaymentOption* pour ses options de paiement, et *Language* pour sa langue. Les énumérations comprennent *AdressType*, *PaymentOptions*, *Interests*, *Gender* et *UserRole*.

Pour conclure, la dernière étape consiste à appliquer un modèle de transformation basé sur XSLT (voir sous-section c) de la section 1.4.1.3), en incorporant des cadres d'application spécifiques de l'actif $\mathbf{MM_{IM}}$, afin d'obtenir le code de l'application mobile.

Notre approche facilite également la création d'un modèle de spécification de la plateforme (PSM) par une simple représentation graphique du code. Le PSM est structuré en divers packages conformes à l'architecture d'implémentation. Dans le contexte de notre exemple d'illustration, axé spécifiquement sur les fonctionnalités des SR, nous identifions les packages suivants au niveau du PSM (voir figure 3.4) :

- Le paquetage *Core*, en tant que pivot central de l'architecture, englobe divers éléments tels que des entités, des interfaces, des services de domaine, etc.

- Le paquetage *SharedKernel* renferme des éléments partagés entre différentes applications qui mettent en œuvre la même architecture.

- Le paquetage Infrastructure concrétise les interfaces définies dans le paquetage *Core*, englobant généralement le code pour accéder aux sources de données ou aux applications tierces.

- Le paquetage *NReco* abrite les classes du cadre d'application NReco.Recommender.

- Le paquetage *API* offre une couche d'interaction avec le système, facilitant la communication avec d'autres composantes de l'architecture.

La figure 3.5 offre un extrait de $\mathbf{PSM_{MM_{Uml}}}$, mettant en évidence les méthodes de recommandation de la classe *Attraction* pour une compréhension claire des dépendances entre les différents paquetages mentionnés précédemment. Cette représentation illustre également la mise en œuvre du patron *Repository* (Ampatzoglou et al., 2013) dans notre architecture, simplifiant le développement de référentiels. Ces référentiels, qu'il s'agisse de classes ou de composants, encapsulent la logique nécessaire pour accéder aux sources de données, consolidant ainsi les fonctionnalités d'accès aux données communes. Cette consolidation améliore la maintenabilité et déconnecte l'infrastructure ou la technologie utilisée pour accéder aux bases de données de la couche *Core*. La classe *AttractionService*, qui implémente l'interface I*AttractionService*, dépend de deux classes : *GenericUserBasedRecommender* (recommandation basée sur l'utilisateur) et *GenericItemBasedRecommender* (recommandation basée sur l'élément). La méthode *GetSimilarPreferredAttractions* utilise la recommandation basée sur l'élément via *GenericItemBasedRecommender*, qui dépend à son tour de la classe

EuclideanDistanceSimilarity et de la classe *ThresholdUserNeighborhood* pour les algorithmes de similarité et de voisinage. D'autre part, *GetAttractionsWithSameUserProfile* exploite la recommandation basée sur l'utilisateur via *GenericUserBasedRecommender*, qui dépend également de la classe *UncenteredCosineSimilarity* pour l'algorithme de similarité. Ainsi, *AttractionService* s'appuie sur ces dépendances pour fournir des recommandations spécifiques basées sur l'utilisateur et l'élément, chacune avec des approches algorithmiques distinctes

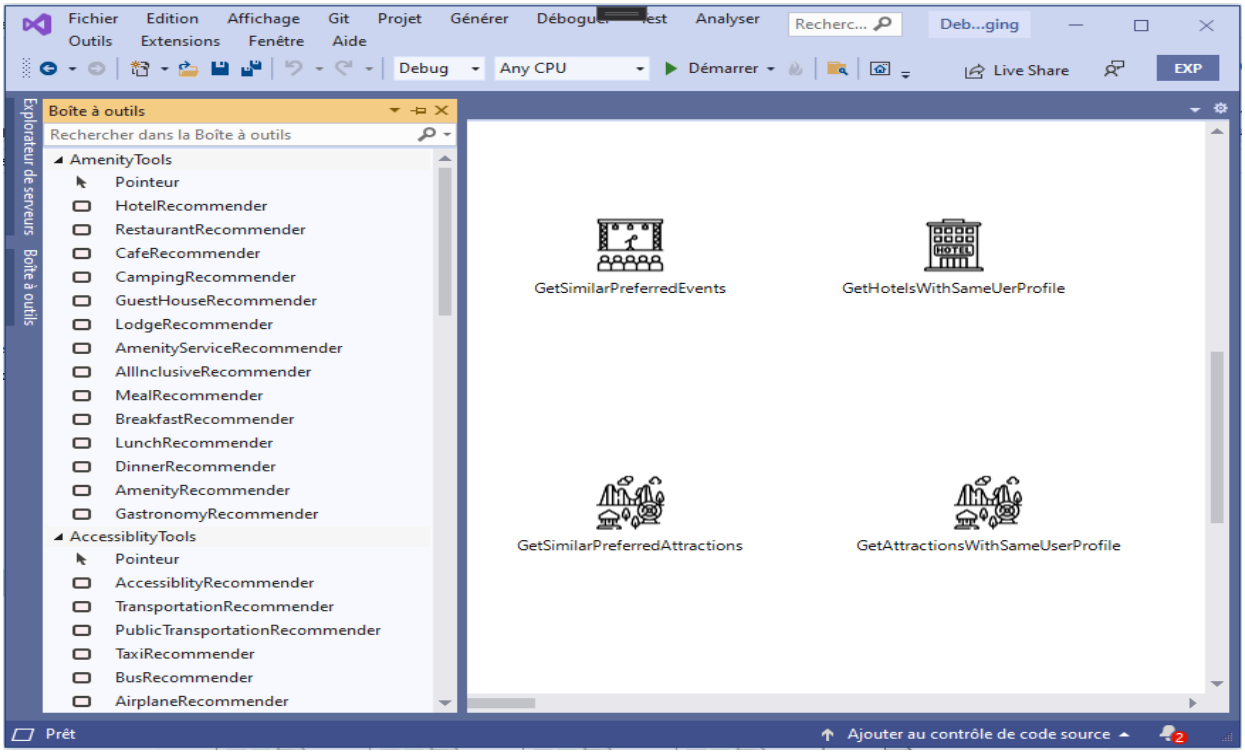


Figure 3.2 – À l'aide de cette interface utilisateur, un analyste marketing peut spécifier les fonctions de recommandation pour l'application mobile.

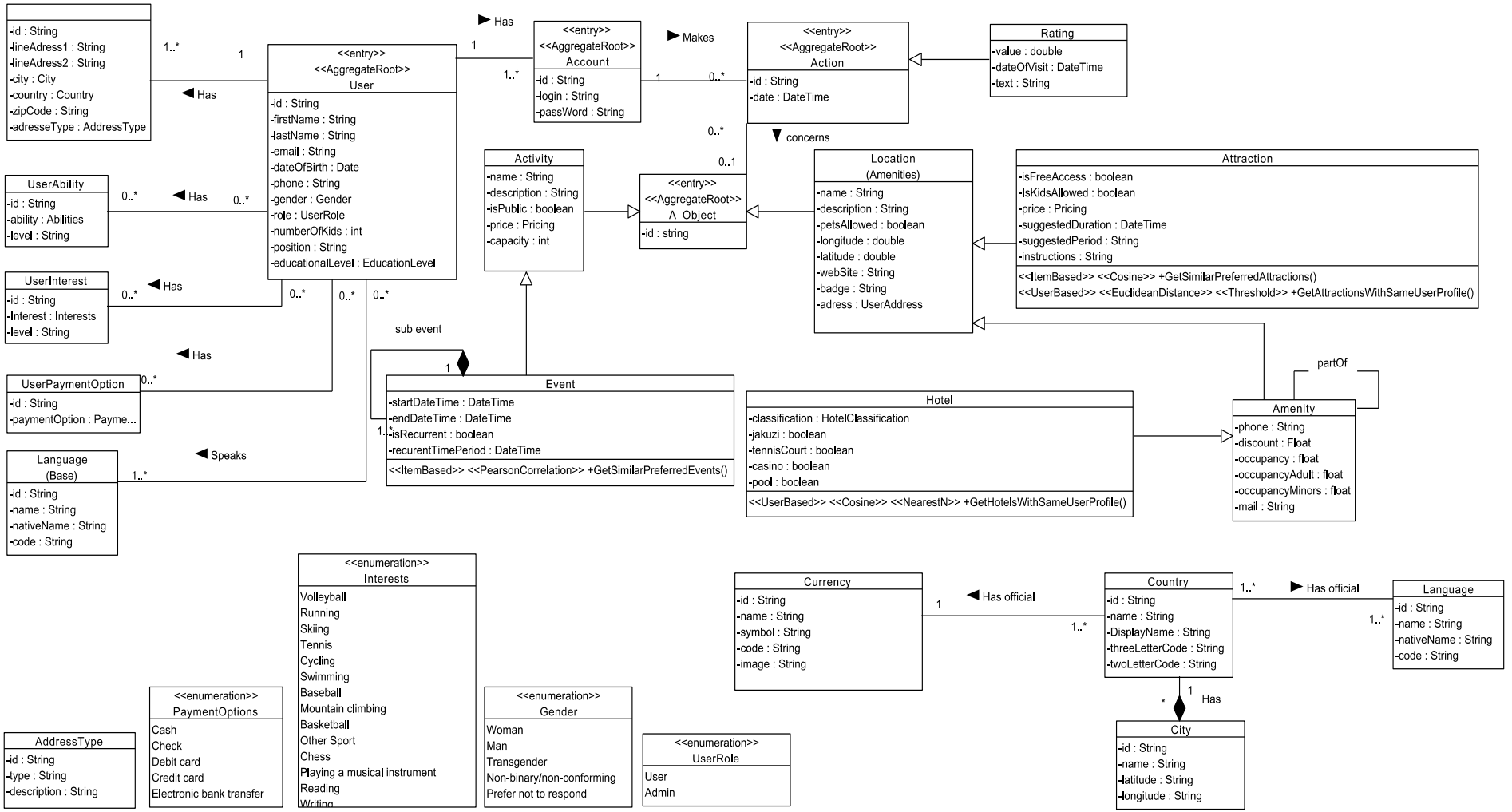


Figure 3.3 – PIM paramétré.

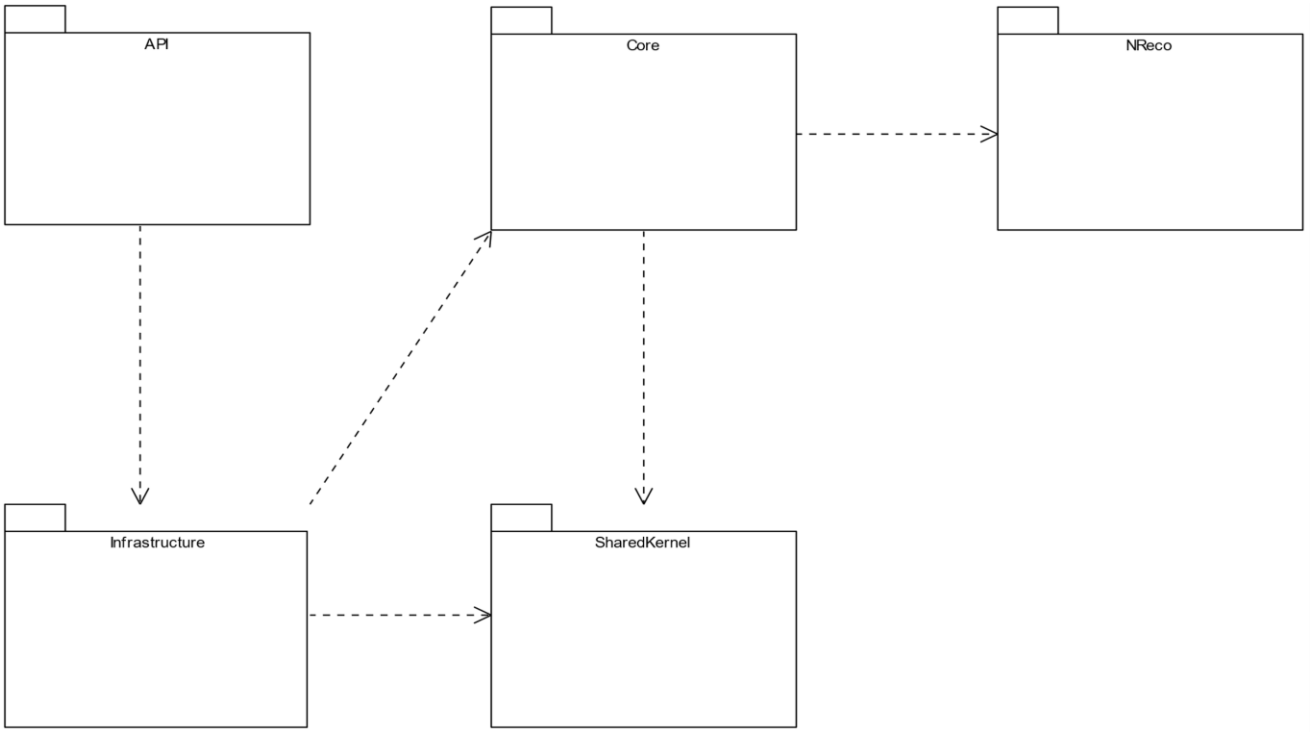


Figure 3.4 – Les paquetages du PSM.

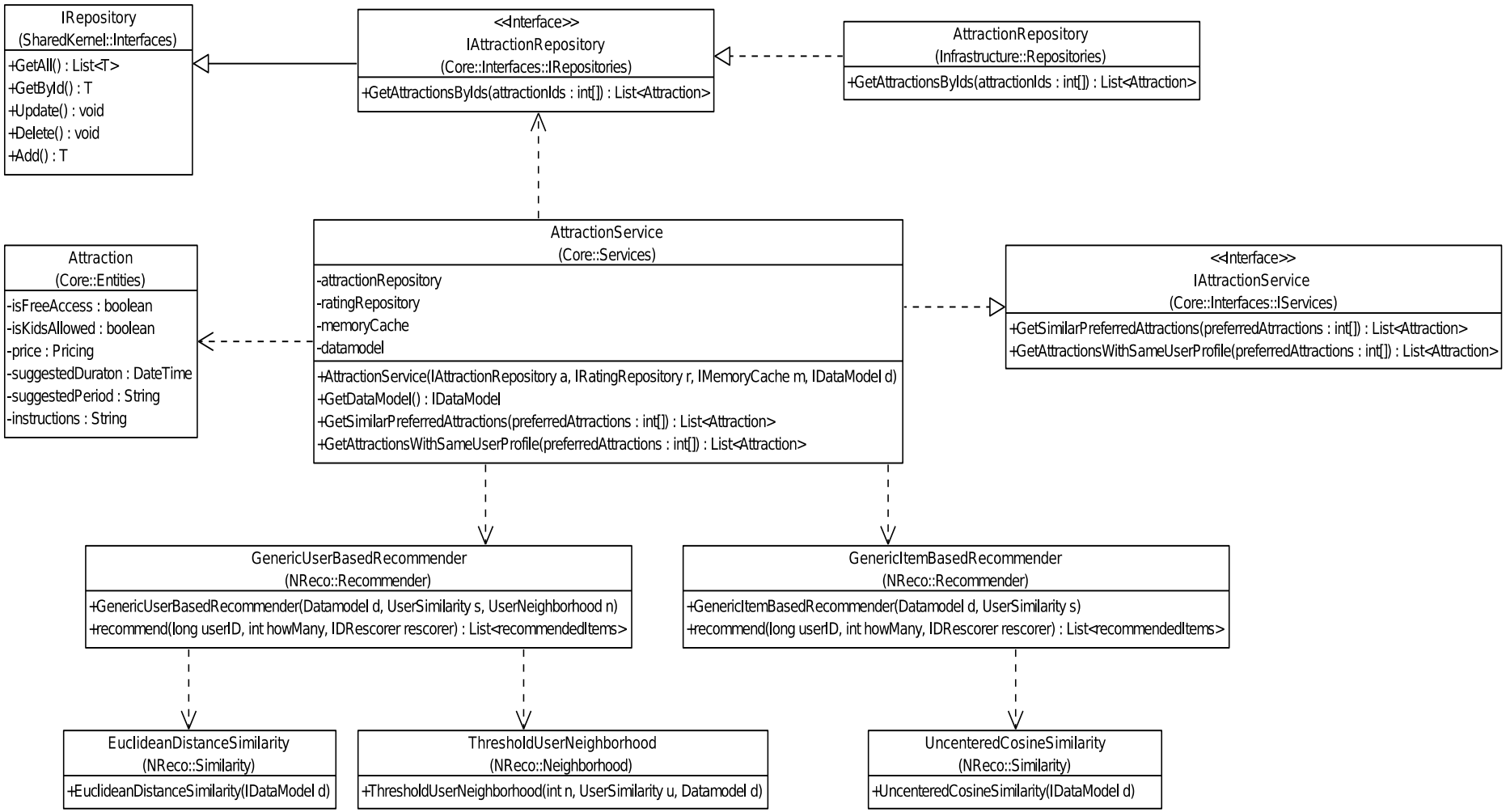


Figure 3.5 – Un extrait du PSM.

3.2 LE LANGAGE SPÉCIFIQUE AU DOMAINE DU SYSTÈME DE RECOMMANDATION

3.2.1 Le métamodèle

Dans la première phase de notre démarche, axée sur l'analyse et la modélisation de notre système applicatif, nous faisons usage de diagrammes UML et de DSL. Pour définir ces DSL, nous structurons notre métamodèle en suivant les couches du MOF. Le modèle MOF se compose de deux paquetages, EMOF (Essential MOF) et CMOF (Complete MOF). Dans notre étude de cas, nous érigeons notre métamodèle sur la base du CMOF. Cette décision découle de la richesse des capacités de modélisation qu'offre le CMOF, telles que la fusion et l'importation de paquetages, ainsi que l'instanciation d'associations et de classes. À l'inverse, l'EMOF ne permet d'instancier que des classes. À l'intérieur de la hiérarchie de modélisation de notre métamodèle, nous distinguons :

- La couche M3 (voir figure 3.6) constitue une couche de spécification de langage et se compose d'un unique métamodèle, à savoir le MOF, qui autorise la spécification d'autres métamodèles.

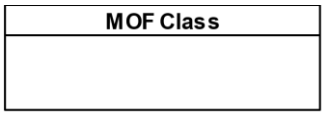


Figure 3.6 – Illustration de la couche M3 du métamodèle.

- La couche M2 (voir figure 3.7) représente la couche de métamodèle avec son autoreprésentation basée sur CMOF, établissant une relation d'instance avec le métamodèle MOF (couche M3). *ModelElement* est la métaclasse racine abstraite UML, dépourvue de superclasse dans la hiérarchie des éléments UML. *NamedElement* est un élément abstrait qui peut avoir un nom, utilisé pour l'identification de l'élément nommé dans les espaces de noms dans lesquels il est défini ou accessible. *Type* est une métaclasse UML abstraite qui représente un ensemble de valeurs et sert de contrainte sur la plage de valeurs représentées par

l'élément typé associé. *Classifier* est une métaclasse abstraite qui décrit un ensemble d'instances partageant des caractéristiques communes. *Class* est un classificateur qui décrit un ensemble d'objets partageant les mêmes caractéristiques, contraintes et sémantiques. En tant que caractéristique structurelle, *Property* représente une partie nommée de la structure d'un classificateur.

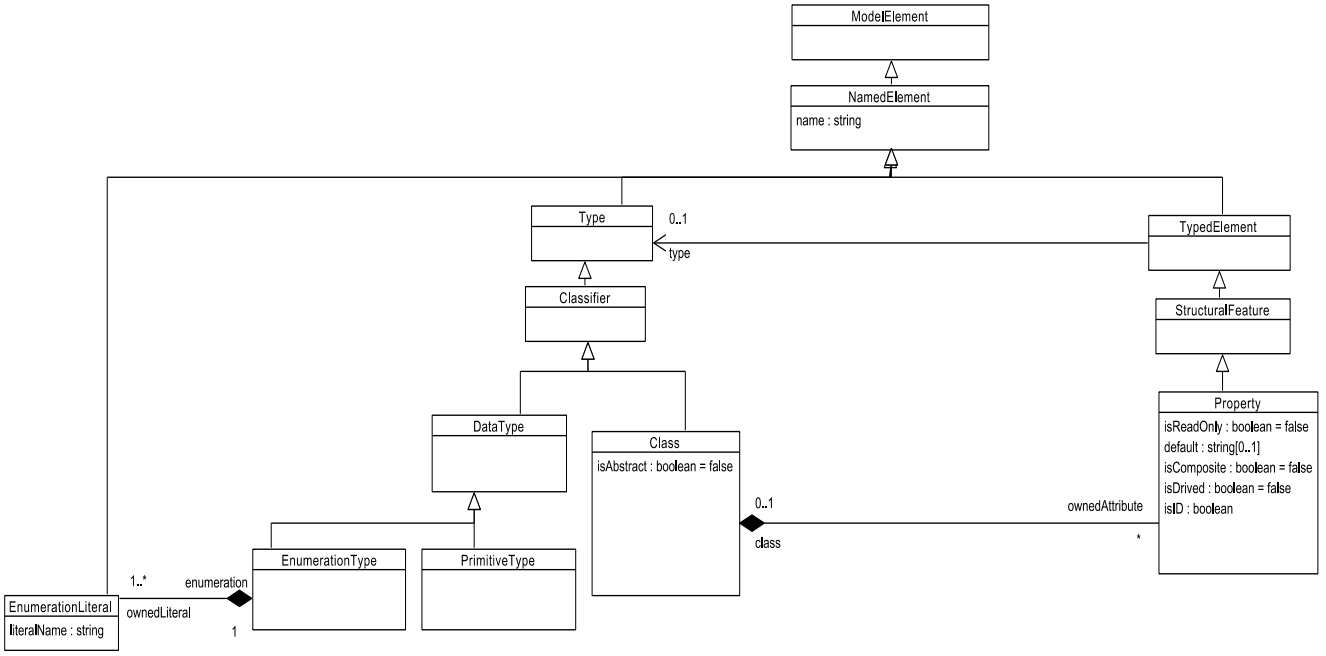


Figure 3.7 – Illustration de la couche M2 du métamodèle.

- La couche M1 (voir figure 3.8) représente la couche de modèle de domaine regroupant les fonctionnalités de recommandation conformes aux exigences du cadre 6A. Toutes les classes héritent de la classe abstraite *Recommender*. Cette dernière contient deux attributs nécessaires pour instancier un modèle de SR : *title*, référençant le titre à renseigner pour le SR choisi depuis l'outil supportant le DSL, et *recommenderType*, qui doit également être renseigné, c'est-à-dire le type de système de recommandation, pouvant être soit basé sur l'utilisateur (user-based en anglais) ou basé sur l'item (item-based en anglais).

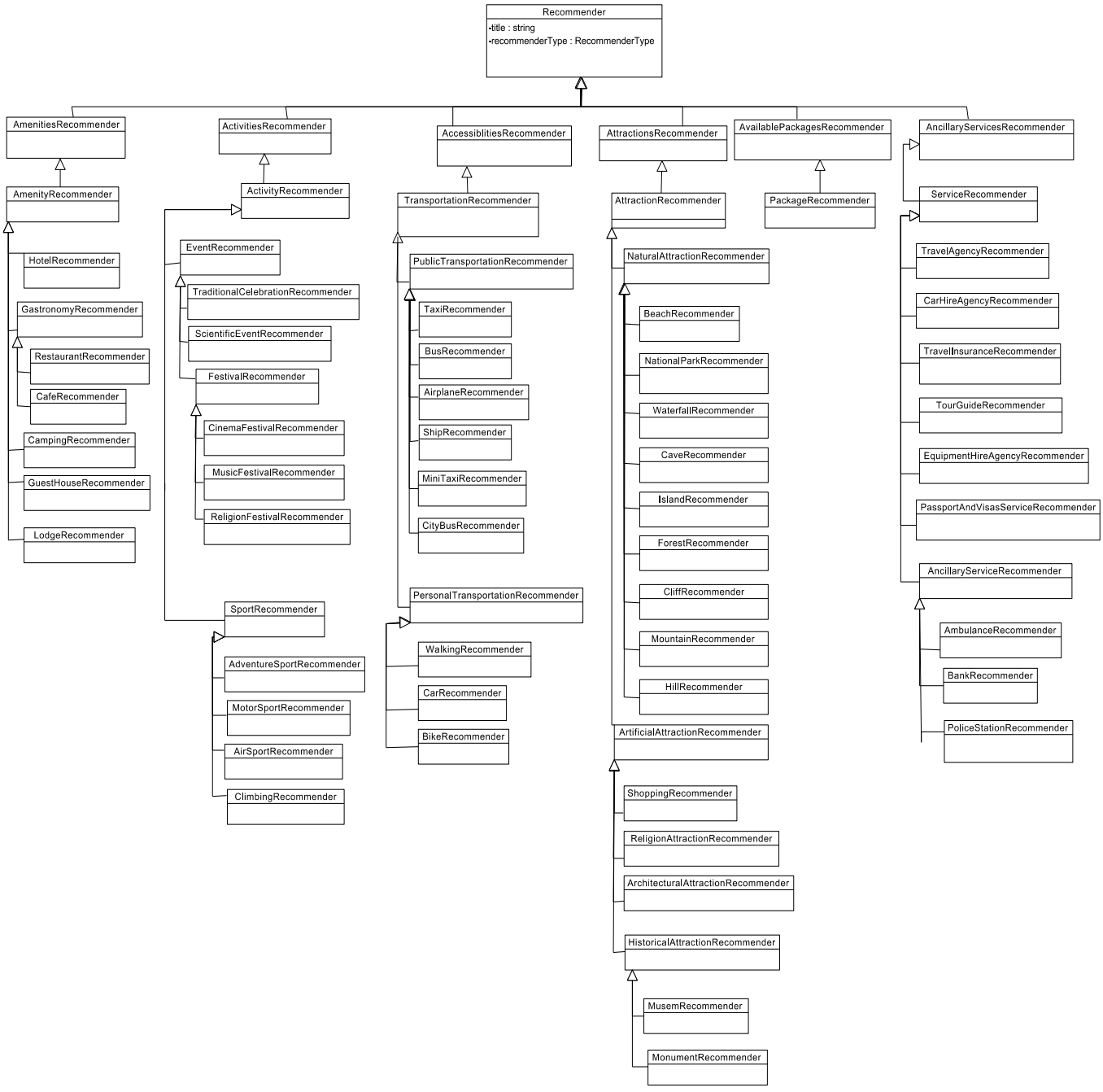


Figure 3.8 – Illustration de la couche M1 du métamodèle.

La figure 3.9 illustre la relation entre les couches de modélisation précédemment évoquées. Pour la représentation de la couche M1, nous nous sommes limités à mentionner la superclasse *Recommender*, puisque toutes les autres classes de notre modèle héritent de

cette classe, ainsi que de ses propriétés *title* et *recommenderType*, qui ont comme *type* une instanciation indirecte de *DataType*.

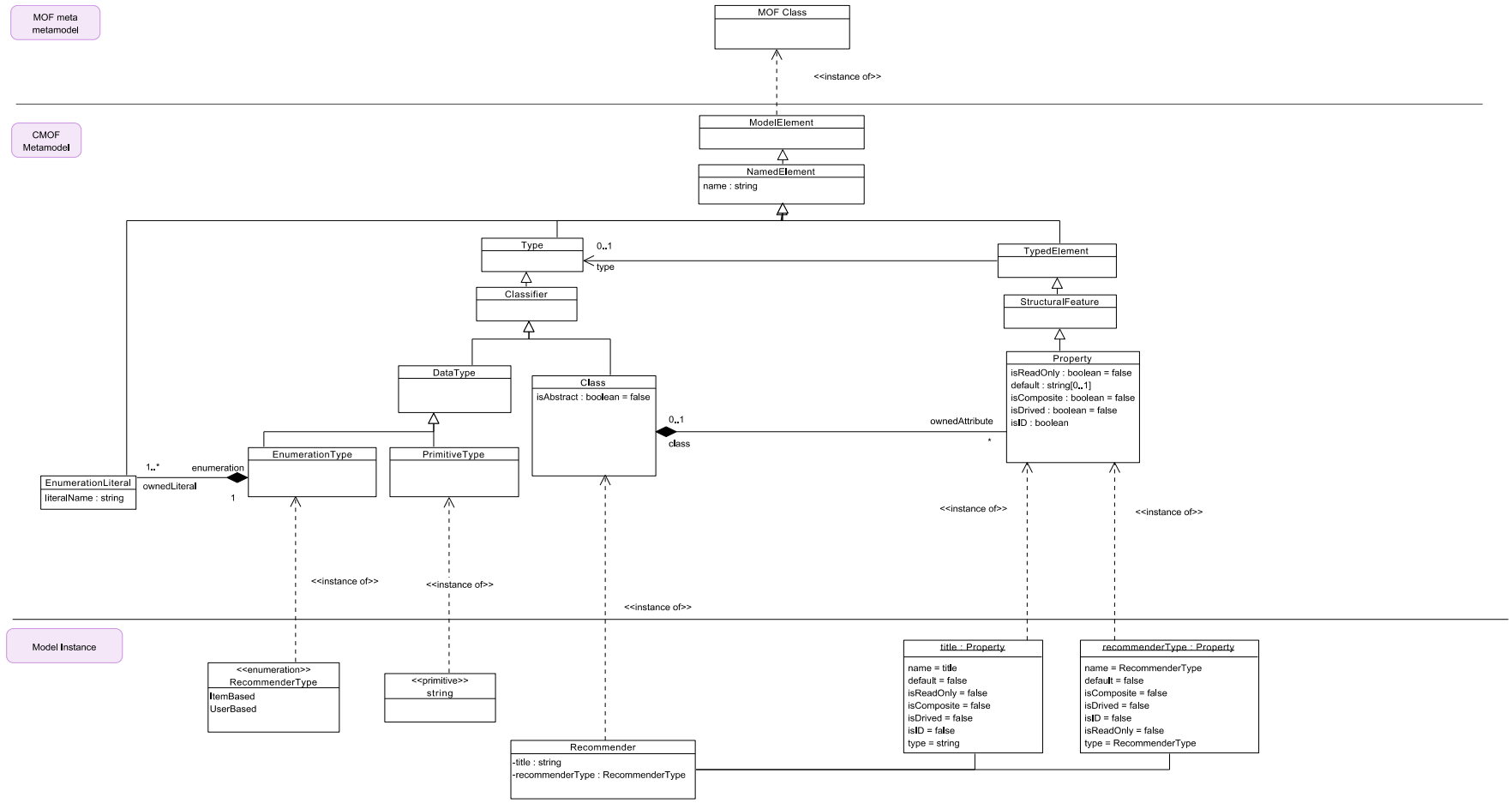


Figure 3.9 – Relations entre les couches du métamodèle.

3.2.2 L'implémentation

Il existe de nombreux outils dans le domaine de la modélisation spécifique au domaine. Pour notre étude, nous avons sélectionné le DSL Modeling SDK Tool[41] comme environnement pour concevoir un DSL basé sur notre métamodèle, que nous pouvons utiliser comme actif dans notre usine de logiciels. Cet outil se compose des éléments suivants (voir figure 3.10) (Microsoft, 2023) :

- Un assistant de projet utilisant différents modèles de solution pour le développement du DSL.
- Un concepteur graphique permettant la création et l'édition de la définition du DSL.
- Un moteur de validation garantissant la conformité de la définition du DSL et signalant les erreurs ou avertissements en cas de problèmes.
- Un générateur de code transformant la définition du DSL en code source en sortie.

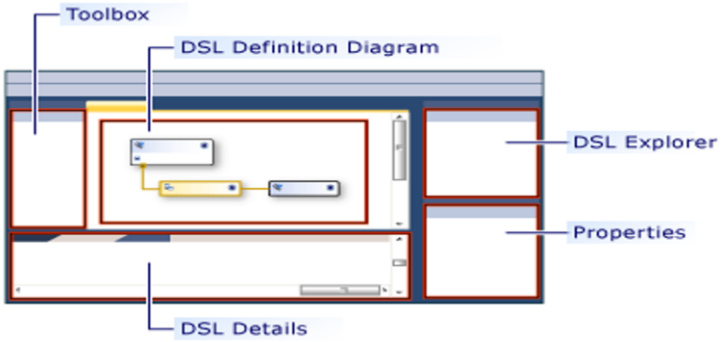


Figure 3.10 – Description de l'interface de DSL Modeling SDK Tool (source : Microsoft (2023)).

Le diagramme de définition du DSL présent au sein du DSL Modeling SDK Tool (voir figure 3.11) rassemble les informations relatives à la définition du DSL. L'explorateur DSL

[41] https://learn.microsoft.com/en-us/visualstudio/modeling/modeling-sdk-for-visual-studio-domain-specific-languages?view=vs-2022

intègre des données supplémentaires du diagramme de définition du DSL et facilite la personnalisation avancée de notre DSL. Lorsqu'un élément est sélectionné au sein du diagramme de définition du DSL ou dans l'explorateur DSL, ses informations associées sont affichées dans la fenêtre *Propriétés* ainsi que dans la fenêtre *Détails* du DSL.

Le diagramme de définition du DSL est utilisé pour détailler les éléments du modèle, en particulier les classes de notre modèle de domaine, ainsi que les relations qui établissent les liens entre ces éléments. En appliquant ce concept à notre DSL, nous pouvons visualiser divers éléments du modèle (voir figure 3.12) en spécifiant les classes du modèle de domaine dans le diagramme de définition du DSL, conformément au métamodèle défini précédemment.

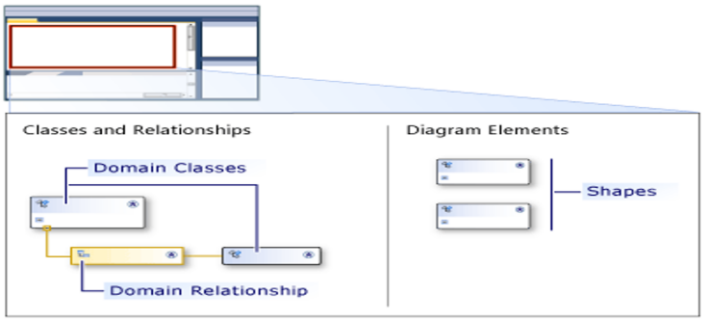


Figure 3.11 – Les éléments du diagramme de définition du DSL (source : Microsoft (2023)).

L'outil de conception du DSL est doté d'un cadre de génération de texte basé sur des modèles, notamment le Text Template Transformation Toolkit, communément appelé T4. Ainsi, un modèle de texte T4 se compose de blocs de texte et d'instructions logiques rédigés sous forme de fragments de code en C# ou Visual Basic, permettant de générer un fichier texte tel qu'un fichier de ressources, une page web, ou du code source de programme dans n'importe quel langage de programmation (Microsoft, 2023). Dans notre cas, le fichier généré par le fichier source T4 (voir ANNEXE II) lors de l'instanciation de notre modèle DSL est un fichier XML. Ce fichier se présente sous forme sérialisée du diagramme de classes, contenant les fonctions de recommandations choisies par l'utilisateur intégrées en

tant qu'opérations dans des classes appropriées du modèle du domaine. Par exemple, les recommandations sur les attractions sont intégrées dans la classe *Attraction*.

Dans la prochaine section, nous examinerons les modèles de transformations pour générer le code source des SR, basés sur le cadre d'application NReco.Recommender[42]. Cela nous permettra de mettre en œuvre les différents algorithmes issus du modèle du DSL mentionné précédemment. Nous discuterons également de l'outil développé pour appliquer ces modèles de transformations.

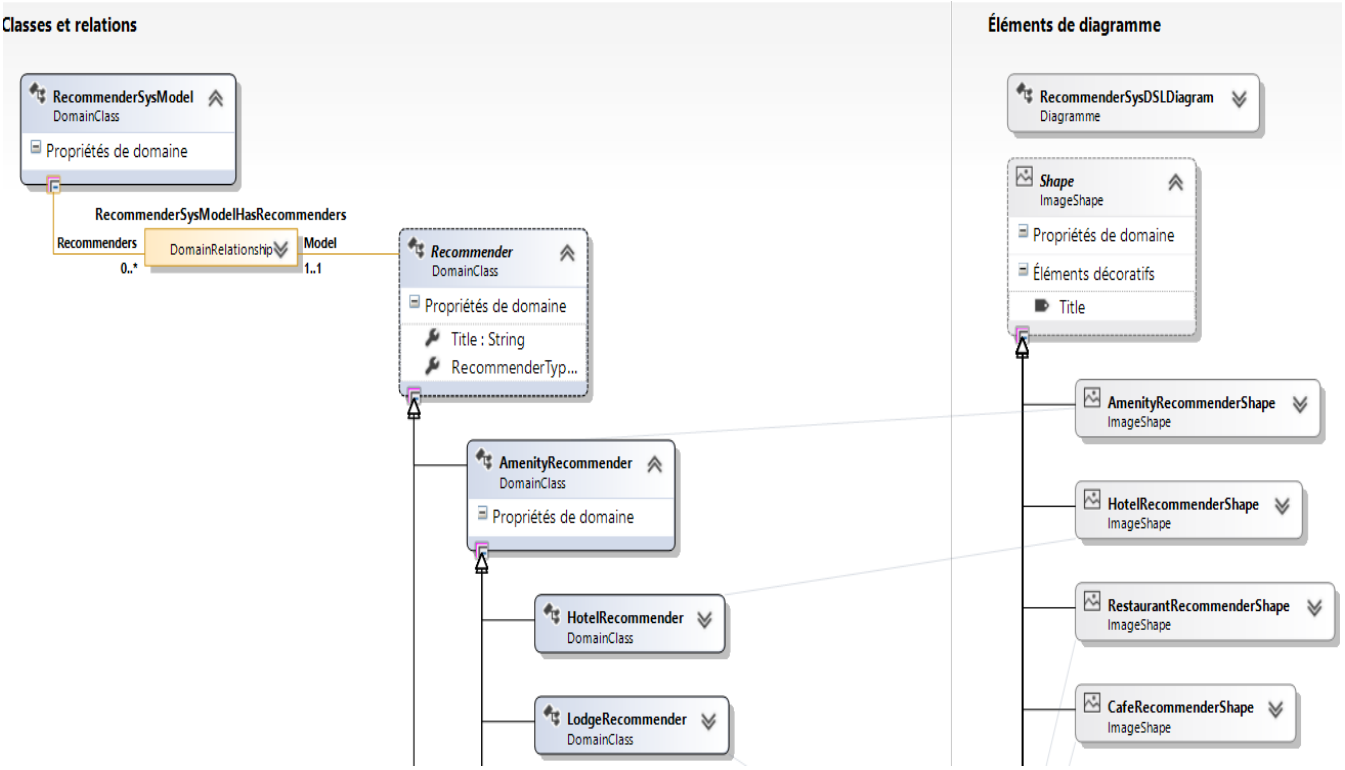


Figure 3.12 – Le diagramme de définition du DSL.

[42] https://www.nrecosite.com/recommender_net.aspx

3.3 MODÈLES DE TRANSFORMATION

Les informations présentées dans les sous-sections 3.3.1 et 3.3.3 sont basées sur les travaux de Abdelmalek et Khriss (2023).

3.3.1 Le métamodèle

Le processus de développement logiciel en MDA implique l'application d'une plateforme décrite dans un PDM au PIM pour générer le PSM, qui est ensuite utilisé pour générer le code source du système. Notre approche MDE utilise deux vues pour décrire le PDM (voir figure 3.13) : un profil UML et un ensemble de transformations. Le profil UML est utilisé pour paramétrer le PIM avec les décisions de conception, tandis que les transformations sont appliquées au PIM paramétré pour produire le code source du système ou le PSM.

Le profil UML élargit le métamodèle d'UML pour prendre en charge un domaine ou une plateforme spécifique avec des stéréotypes et des valeurs étiquetées. Il utilise des concepts et des contraintes, où les concepts sont les éléments clés d'une plateforme d'implémentation, définis par leur nom, type, description, et élément de conception (design concern en anglais). Les éléments de conception (design concern en anglais), définies par leur nom, type, éléments UML associés (package, classificateur, attribut, opération, paramètre, généralisation, association, extrémité d'association, et dépendance), et description, permettent de formuler des problèmes de conception. Les contraintes, préservant l'intégrité de la plateforme, sont définies par leur nom, concepts concernés, type (dépendance, compatibilité, incompatibilité, raffinement), et description.

Un PDM peut être indépendant du système ou spécifique au système. Un PDM spécifique au système peut être développé à partir de zéro ou composé en réutilisant des PDM indépendants du système.

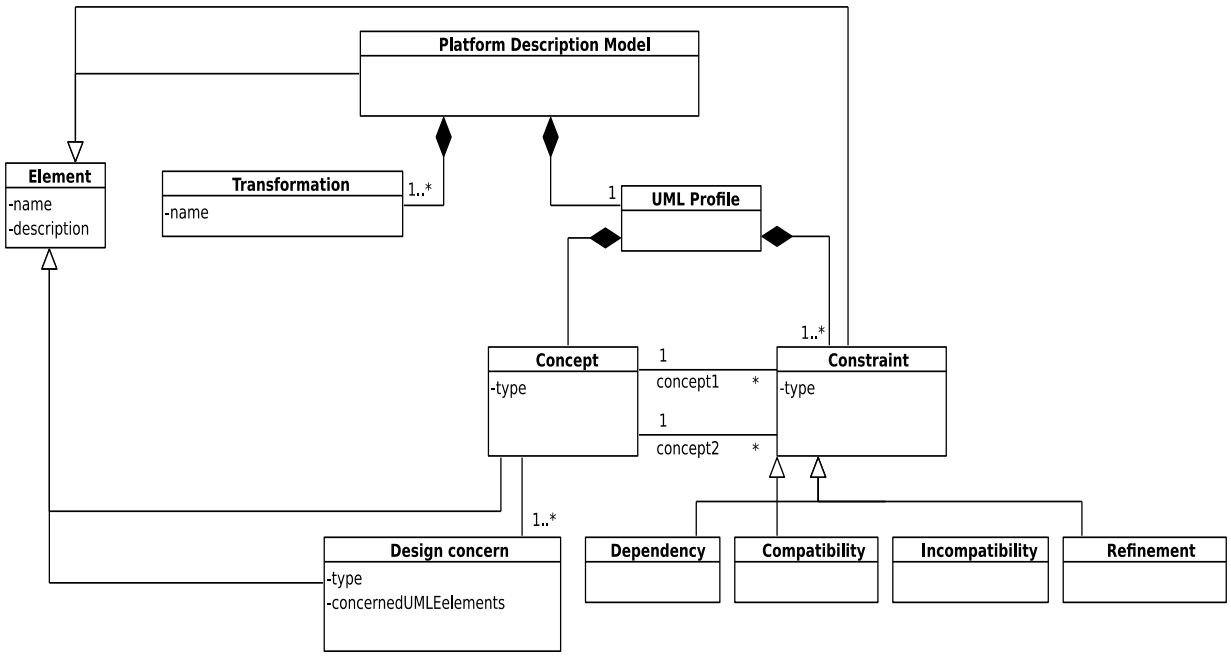


Figure 3.13 – Le métamodèle des transformations (1) (source : Abdelmalek et Khriss (2023)).

Une transformation s'applique à un type d'élément de modèle spécifique (voir figure 3.14), lié au métamodèle d'UML, tel que paquetage (package en anglais), classificateur, attribut, opération, paramètre, généralisation, association, extrémité d'association ou dépendance. Le contexte, basé sur les propriétés d'un type d'élément de modèle, peut être déterminé par des conditions comme l'existence d'une propriété de stéréotype nommée *Repository* dans un élément de modèle de classe. Cette transformation génère un élément de modèle qui implémente un concept d'architecture ou de plateforme, pouvant être un classificateur (classe ou interface), opération, attribut, paramètre ou artefact. Identifiée par son nom, une transformation peut avoir des paramètres, dont l'existence dépend du type de transformation, qui peut être générique ou de conception.

Les transformations de conception, qu'elles soient indépendantes ou spécifiques au système, peuvent être élaborées en utilisant des transformations génériques (Generic Transformations ou GT). De plus, une GT peut servir de conteneur, d'élément de base, ou même les deux simultanément.

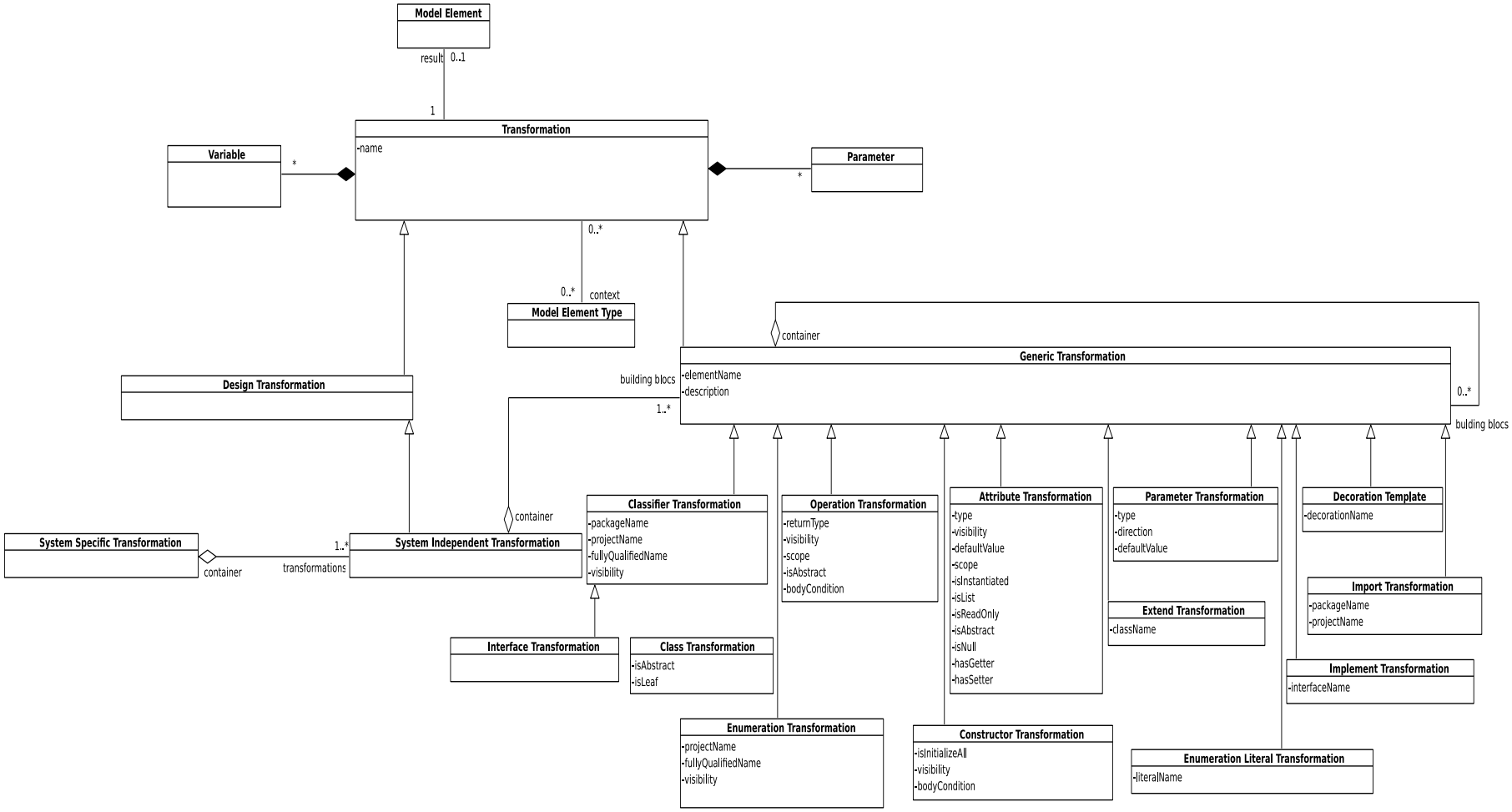


Figure 3.14 – Le métamodèle des transformations (2) (source : Abdelmalek et Khriss (2023)).

Une transformation indépendante du système (System-independent transformation ou SIT) est élaborée en définissant son contexte et en réutilisant des GT comme éléments de base. En revanche, une transformation spécifique au système (System-specific transformation ou SST) constitue un autre type de transformation de conception, construite en réutilisant les SIT pour mettre en œuvre les décisions de conception propres à un système donné.

3.3.2 Les transformations développées

Le développement d'une application de tourisme intelligent requiert de prendre plusieurs décisions cruciales en matière de conception, notamment le choix de l'architecture de référence et des technologies à utiliser. Dans notre cas, nous avons opté pour la Clean Architecture[43] (Martin, 2017) comme architecture principale pour la partie côté serveur de l'application, ainsi que NReco.Recommender comme cadre d'application pour le support de bas niveau des fonctionnalités du SR.

NReco.Recommender est une bibliothèque .NET[44] basée sur le moteur de filtrage collaboratif d'Apache Mahout (Owen et al., 2011). Elle analyse le comportement des utilisateurs (telles que les statistiques d'utilisation, les préférences et les évaluations) pour identifier les utilisateurs similaires (NReco, 2023). Parmi les algorithmes de recommandation proposés par Apache Mahout, on trouve principalement deux catégories : la catégorie basée sur la mémoire, qui inclut les algorithmes basés sur l'utilisateur et les algorithmes basés sur l'item, et la catégorie basée sur le modèle, qui comprend l'algorithme de décomposition en valeurs singulières (Singular Value Decomposition ou SVD - Klema et Laub (1980)). Pour notre étude, nous avons opté pour la première catégorie.

[43] Ensemble de principes, de règles et de pratiques architecturales développés par Robert C. Martin, également connu sous le nom de "Uncle Bob". Cependant, il est également considéré comme une architecture à part entière, car il propose une structure et une organisation spécifiques pour les applications logicielles.

[44] https://dotnet.microsoft.com/en-us/

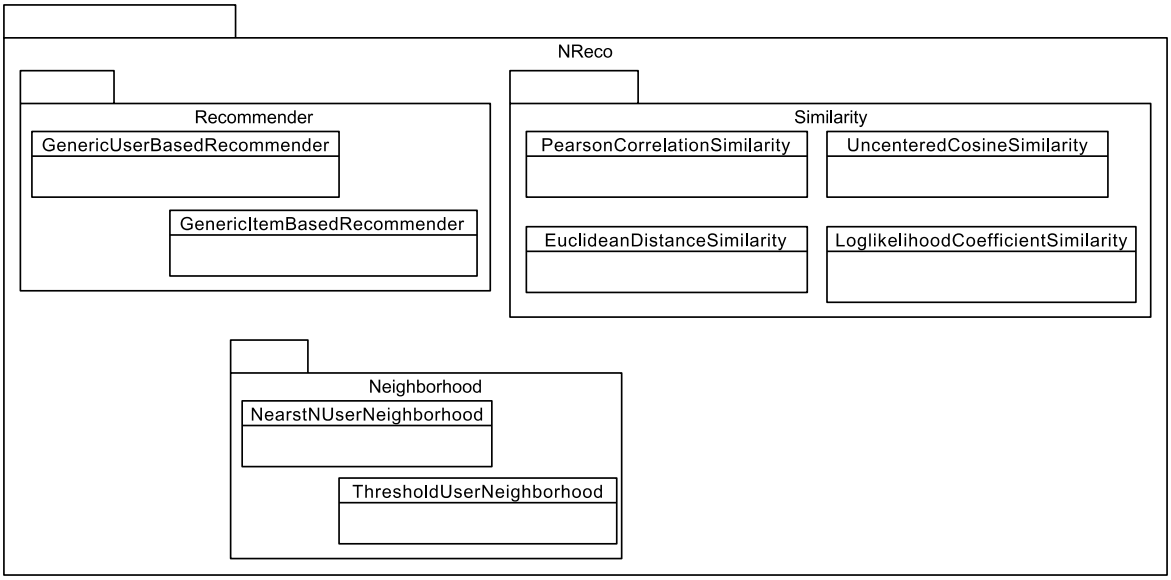


Figure 3.15 – Les paquetages du cadre d'application NReco.

NReco.Recommender comprend principalement trois abstractions de haut niveau (Owen et al., 2011) (voir figure 3.15) :

- La similarité : Cette abstraction intègre différentes métriques pour calculer les mesures de similarité sur un jeu de données. Les implémentations existantes sont utilisées pour évaluer le degré de corrélation entre les utilisateurs ou les articles en fonction de leurs préférences respectives. Les différentes mesures de similarité prises en charge comprennent la distance euclidienne (*EuclideanDistanceSimilarity*), la corrélation de Pearson (*PearsonCorrelationSimilarity*), le cosinus non centré (*UncenteredCosineSimilarity*) et le coefficient de vraisemblance logarithmique (*LoglikelihoodCoefficientSimilarity*).

- Le voisinage : Après le calcul de similarité, ce composant fournit le groupe d'utilisateurs qui sont les plus similaires à un utilisateur donné. Les approches de voisinage prises en charge incluent les plus proches (*NearstNUserNeighborhood*) et celles basées sur un seuil (*ThresholdUserNeighborhood*).

- Le recommandeur : Ce moteur utilise tous les composants ci-dessus ensemble pour recommander des éléments à un utilisateur cible (recommendation « top-N ») ainsi que pour estimer la préférence d'un utilisateur envers un élément inconnu (prédiction de notation) en utilisant une moyenne pondérée. Les approches de recommandation prises en charge comprennent celles basées sur l'utilisateur (*GenericUserBasedRecommender*) et sur l'item (*GenericItemBasedRecommender*).

Pour implémenter les algorithmes de filtrage collaboratif (basés sur l'utilisateur ou sur l'item), les étapes suivantes sont nécessaires :

1. Collecte de toutes les préférences, qui sont des données d'utilisateur et d'item.
2. Calcul de la similarité entre les utilisateurs/items en utilisant des implémentations existantes pour mesurer le degré de corrélation entre eux en fonction de leurs préférences.
3. Identification des utilisateurs les plus similaires (voisinage) à l'utilisateur sélectionné dans le cadre d'une recommandation basée sur l'utilisateur.
4. Recommandation des items à l'utilisateur sélectionné en fonction de la méthode de recommandation choisie, que ce soit une recommandation basée sur l'utilisateur ou sur l'itemPour générer le code source des SR de l'application mobile, le système utilise les transformations de la Clean Architecture et du cadre d'application NReco.Recommender.

La première étape consiste à définir le PDM de chacun d'entre eux. Le profil UML de la Clean Architecture contient, des éléments tels qu'*Entity*, *AggregateRoot* et *Service*. Le profil de NReco contient, des éléments tels que *UserBased* et *ItemBased* pour spécifier le type de recommandation, *NearstN* et *Threshold* pour spécifier l'algorithme de voisinage, et enfin *Cosine*, *EuclideanDistance*, *PearsonCorrelation* et *Loglikelihood* pour spécifier l'algorithme de similarité.

Après la spécification des PDM, nous paramétrons le PIM avec différentes décisions de conception en utilisant le profil UML des PDM (voir section 3.1). La paramétrisation de la classe *Event* du PIM est présentée dans la figure 3.16. Par exemple, l'opération *GetSimilarPreferredEvents* de la classe *Event* est paramétrée avec les stéréotypes suivants : *ItemBased* et *PearsonCorrelation* de NReco, signifiant respectivement le type de recommandation et l'algorithme de similarité de l'opération *GetSimilarPreferredEvents*.

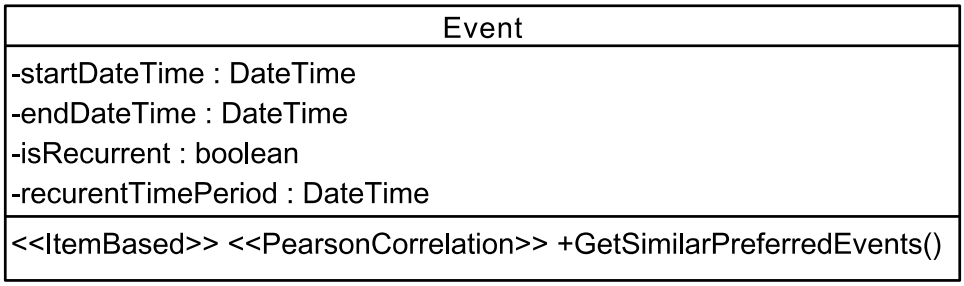


Figure 3.16 – Un extrait de la paramétrisation du PIM.

La dernière étape consiste à appliquer une transformation M2T au PIM paramétré pour générer le code source du système. Un extrait de la transformation M2T, rédigé avec le langage XSLT (voir ANNEXE III), pour le développement des SR est présenté dans la figure 3.17. Cette portion de code XSLT examine la présence du stéréotype *ItemBased* au niveau des opérations du PIM. Si le stéréotype *ItemBased* est détecté, une vérification supplémentaire est effectuée avec un deuxième stéréotype pour déterminer l'algorithme de similarité associé. En fonction de cette double vérification, elle génère du code C# spécifique à l'algorithme de similarité choisi, créant ainsi une variable utilisée dans la méthode de recommandation, instanciée par l'utilisateur depuis l'outil prenant en charge notre DSL.

Le choix d'un langage de transformation est un facteur important lors du développement de transformations. Dans notre approche, nous avons opté pour XSLT en raison de sa portabilité, car plusieurs environnements de développement le prennent en charge. Ces environnements offrent des fonctionnalités de débogage et de test pour les transformations XSLT.

```
<xsl:for-each select="./Stereotypes/Stereotype">
    <xsl:if test="@Name = 'ItemBased'">
    <xsl:for-each select="./../Stereotype">
<xsl:if test="@Name='Cosine'">
var similarity = new UncenteredCosineSimilarity(dataModel);
    </xsl:if>
    <xsl:if test="@Name = 'EuclideanDistance'">
var similarity = new EuclideanDistanceSimilarity(dataModel);
    </xsl:if>
    <xsl:if test="@Name = 'LogLikelihood'">
var similarity = new LogLikelihoodSimilarity(dataModel);
    </xsl:if>
    <xsl:if test="@Name = 'PearsonCorrelation'">
var similarity = new PearsonCorrelationSimilarity(dataModel);
    </xsl:if>
    </xsl:for-each>
  </xsl:if>
</xsl:for-each>
```

Figure 3.17 – Un extrait d'une transformation M2T.

3.3.3 L'outil support

Un plugin pour Visual Paradigm[45] (VP) a été développé en utilisant son API ouverte en Java, étendant ainsi ses fonctionnalités pour permettre la spécification de PDM, la paramétrisation de PIM et la génération de code (Abdelmalek & Khriss, 2023). La spécification de PDM est simplifiée grâce à des interfaces utilisateur permettant de créer le profil UML et de définir les transformations. L'éditeur de modélisation VP facilite la paramétrisation de PIM en permettant la sélection des éléments UML et des PDM, suivie de l'application de diverses décisions de conception. Enfin, le plugin génère du code en utilisant le PIM paramétré et les transformations définies. L'outil utilise le format XML pour l'import/export des PDM et PIM.

L'architecture du plugin est illustrée dans la figure 3.18, organisée en quatre paquetages : *Controllers*, *UserInterface*, *Structures* et *Utilities*, avec le fichier XML *plugin.xml* et la classe *MainMDE*.

[45] https://www.visual-paradigm.com/

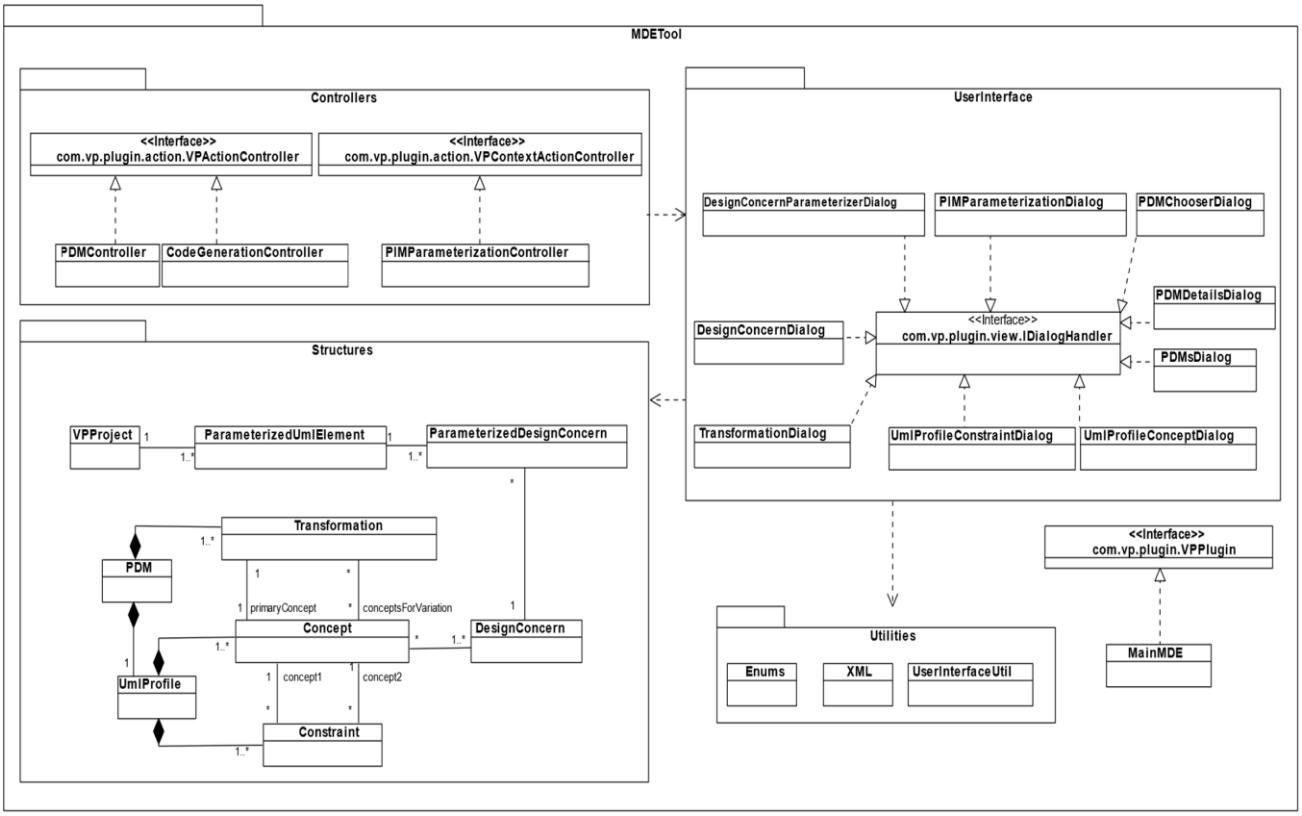


Figure 3.18 – L'architecture du plugin (source : Abdelmalek et Khriss (2023)).

Le fichier XML plugin.xml revêt une importance capitale dans la définition du plugin. Il spécifie un identifiant, une description, le fournisseur, la classe principale, les ensembles d'actions et d'actions contextuelles. La classe principale *MainMDE* implémente *com.vp.plugin.VPPlugin* et est la première classe exécutée lors du chargement du plugin. Les ensembles d'actions permettent la personnalisation des barres d'outils et des menus dans VP. On distingue deux types d'actions : celles agissant sur les barres d'outils principales et de diagramme via des ensembles d'actions, et celles dans le menu contextuel via des ensembles d'actions contextuelles. Le plugin définit deux actions sur la barre d'outils principale, l'une pour la définition de PDM et l'autre pour la génération de code. De plus, une action dans le menu contextuel permet la paramétrisation de PIM, impliquant la sélection des éléments UML du diagramme de classe. Chaque action est associée à un contrôleur d'action dans le

paquetage *Controllers*, où se trouvent les classes *PDMController*, *CodeGenerationController* et *PIMParameterizationController*.

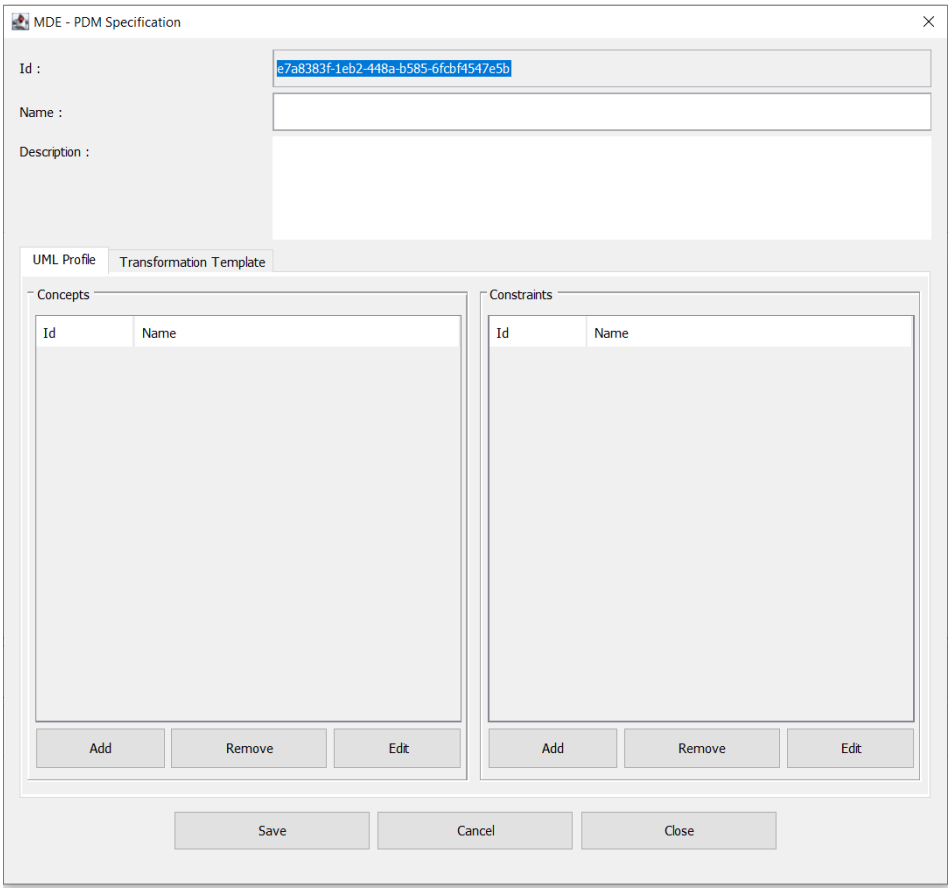


Figure 3.19 – L'interface utilisateur pour spécifier une spécification de PDM (source : Abdelmalek et Khriss (2023)).

Le paquetage *UserInterface* héberge des boîtes de dialogue conçues pour faciliter les interactions avec l'outil. Chaque boîte de dialogue doit implémenter l'interface *com.vp.plugin.view.IDialogHandler* de l'API. On distingue deux types de boîtes de dialogue : les boîtes principales, déclenchées par un contrôleur d'action, et les sous-boîtes, invoquées depuis d'autres boîtes. Parmi les boîtes principales, on trouve *PDMsDialog* et *PIMParameterizationDialog*. *PDMsDialog*, associé à *PDMController*, est responsable de la gestion des opérations telles que l'ajout, la modification, la suppression et la sauvegarde des PDM. Pour ajouter ou modifier un PDM, *PDMDetailsDialog* (voir figure 3.19) est utilisé, en

collaboration avec *UmlProfileConceptDialog*, *DesignConcernDialog*, *UmlProfileConstraintDialog* et *TransformationDialog*, qui permettent respectivement de gérer les concepts, contraintes et transformations. *PIMParameterizationDialog* simplifie le processus de paramétrage de PIM en permettant la sélection des PDM via *PDMChooserDialog* et en affichant les éléments UML sélectionnés sous forme d'arborescence. Les utilisateurs ont la possibilité de choisir plusieurs PDM, ce qui crée des arbres avec le nom du PDM comme racine et les éléments UML sélectionnés comme enfants.

Dans la figure 3.20, la boîte de dialogue *PIMParameterizationDialog* affiche deux arbres correspondant aux PDM de la Clean Architecture et de NReco. Une fois que les arbres sont affichés, les utilisateurs peuvent procéder au paramétrage de chaque élément UML. Cette opération de paramétrage est effectuée en appliquant les stéréotypes et les valeurs étiquetées nécessaires, avec l'aide de *DesignConcernParameterizerDialog*.

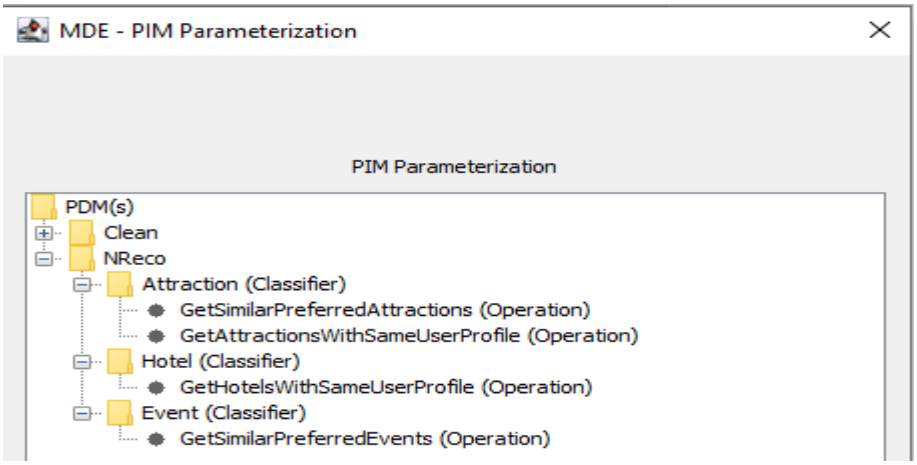


Figure 3.20 – L'interface utilisateur pour paramétrer un PIM.

Le paquetage Structures contient les classes du métamodèle telles que *PDM*, *UMLProfile*, *Concept*, *DesignConcern*, *Constraint*, ains que *Transformation*. Il comprend également *ParameterizedDesignConcern*, *ParameterizedUmlElement*, et *VPProject* qui sont associés au paramétrage. *ParameterizedDesignConcern* capture l'élément de conception (design concern en anglais) paramétré, tandis que *ParameterizedUmlElement* encapsule

l'élément UML avec ses paramètres. *VPProject* enregistre les éléments UML paramétrés dans un diagramme de classe.

Dans le paquetage *Utilities*, on trouve des classes utilitaires comme *XML*, *UserInterfaceUtil*, et *Enums*. *XML* possède des méthodes pour importer/exporter le PDM et le PIM paramétré. *UserInterfaceUtil* simplifie la disposition des contrôles et la gestion des fichiers/dossiers. *Enums* contient des énumérations utilisées dans le plugin, notamment *TransformationType*, *DesignConcernType*, *UMLElementType*, *UMLProfileConceptType*, et *UMLProfileConstraintType*.

3.4 RECONNAISSANCE AUTOMATIQUE DE LA PAROLE

3.4.1 L'approche suivie

Dans notre usine de logiciels, nous envisageons également d'intégrer la reconnaissance vocale pour détecter des commandes vocales prédéfinies. Pour ce faire, cette fonctionnalité suivra un processus composé de en deux tâches principales (voir figure 3.21) : la transcription automatique de la parole et la détection d'intention pour prédire la commande.

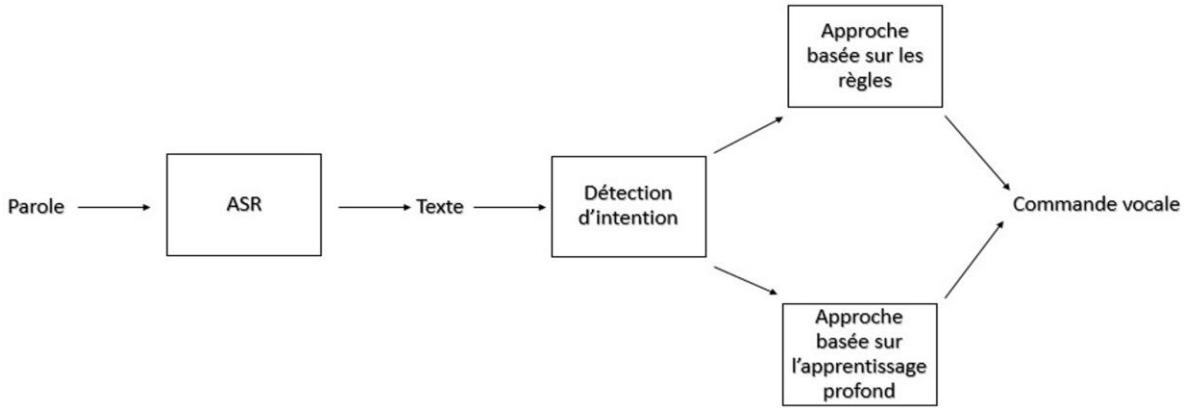


Figure 3.21 – Approche proposée pour la classification de commandes.

3.4.1.1 La tâche de transcription automatique de la parole

La première tâche consiste à transcrire un enregistrement audio en texte. À cette fin, nous utilisons un modèle préentraîné basé sur wav2vec 2.0, plus précisément le modèle jonatasgrosman/wav2vec2-large-xlsr-53-english[46]. Ce modèle repose sur l'architecture XLSR (Cross-lingual Speech Recognition - Conneau et al. (2020)), développée par Facebook AI Research (FAIR) pour permettre la reconnaissance automatique de la parole dans plusieurs langues. Le XLSR apprend des représentations multilingues de la parole en préentraînant un modèle unique à partir des formes d'onde brutes de la parole dans plusieurs langues. Ils se basent sur wav2vec 2.0, comme discuté dans la sous-section c) de la section 2.3.1.4, qui est entraîné en résolvant une tâche contrastive sur des représentations de parole latentes masquées. De plus, le modèle apprend une quantification des latents partagée entre les langues. Les résultats de l'affinement du modèle sur des données étiquetées montrent que l'apprentissage préalable multilingue est nettement supérieur à l'apprentissage préalable monolingue.

Dans notre étude, nous cherchons à classifier des commandes vocales prononcées en anglais à travers des enregistrements audio. Il est important de souligner que XLSR offre des performances supérieures par rapport à wav2vec 2.0, même pour une seule langue justifiant ainsi notre choix de modèle.

Selon Conneau et al. (2020), le modèle XLSR-English, préentraîné uniquement sur des données en anglais, améliore de manière significative les performances par rapport aux autres modèles monolingues. En moyenne, XLSR-English présente une réduction relative du taux d'erreur phonémique de 24% par rapport à un modèle wav2vec 2.0 entraîné uniquement en anglais. Ces résultats montrent que l'apprentissage préalable multilingue avec XLSR peut également être bénéfique pour la reconnaissance de la parole en anglais.

[46] https://huggingface.co/jonatasgrosman/wav2vec2-large-xlsr-53-english

3.4.1.2 La tâche de détection d'intention

La détection d'intention vise à associer la commande adéquate à la transcription audio. Pour réaliser cette prédiction, nous utilisons deux approches distinctes : l'une basée sur des règles, et l'autre sur l'apprentissage profond.

a) *L'APPROCHE BASÉE SUR LES RÈGLES*

Dans cette méthode, nous examinons les mots-clés spécifiques présents dans la transcription audio pour effectuer la détection d'intention. Chaque classe est associée à un ensemble prédéfini de mots-clés. Par exemple, la catégorie « hotel », des termes synonymes tels que « hostel » et « lodge » lui sont associés. Si l'un de ces mots-clés est identifié dans la transcription audio, la commande correspondante est attribuée. En cas de détection de plusieurs mots-clés correspondant à différentes commandes, toutes ces commandes sont considérées comme des prédictions potentielles. Cette approche offre une classification rapide basée sur des mots-clés spécifiques, utilisant uniquement le modèle de transcription d'audio en texte.

Il est important de souligner que les plongements (embeddings en anglais) des mots-clés peuvent être utilisés comme base de comparaison plutôt que le texte brut. Les plongements sont des représentations vectorielles de mots ou d'éléments dans un espace de dimension réduite, conçus pour capturer les relations sémantiques et la structure des données d'origine. Ces représentations permettent une analyse et un traitement plus efficaces des données de haute dimension, telles que les mots dans un vocabulaire ou les éléments d'un espace vectoriel. Les travaux de Mikolov et al. (2013) ont grandement contribué au développement de cette technique, notamment avec la publication de leur modèle word2vec.

b) *L'APPROCHE BASÉE SUR L'APPRENTISSAGE PROFOND*

Si aucun mot-clé n'est détecté dans la transcription audio, nous recourons à un modèle de traitement du langage naturel basé sur BERT. Ce modèle est entraîné sur un jeu de données issu de la transcription d'enregistrements vocaux afin d'effectuer une détection d'intention plus approfondie. Dans cette démarche, nous distinguons trois types de phrases qui émanent de la transcription d'enregistrements vocaux pour l'instanciation d'un SR.

Le premier type de phrase contient l'objet de la recommandation (« hotel », « event », « attraction » etc.). Le deuxième type de phrase contient le type de recommandation (« item-based » ou « user-based »). Enfin, le troisième type de phrase contient à la fois l'objet et le type de recommandation (par exemple, « hotel » et « user-based » dans « set up a user-based system to recommend hotels »). Compte tenu de la possibilité qu'une phrase puisse englober à la fois l'objet et le type de recommandation, nous avons choisi de procéder à un double étiquetage de notre jeu de données, comme décrit dans la section 3.4.2.2. Ce double étiquetage en fonction de l'objet et du type de recommandation ce qui nous permet de classifier une phrase donnée en fonction de ces deux aspects, établissant ainsi deux tâches distinctes de classification.

Le modèle proposé (voir figure 3.22 et ANNEXE V pour le code) comprend une couche d'encodage partagée pour résoudre deux tâches de classification distinctes. Chaque classificateur est entraîné sur le même ensemble de données, mais avec des annotations différentes. Bien que la couche d'encodage soit commune aux deux, des couches de décodage distinctes sont utilisées pour chacune d'entre elles. L'encodage est réalisé à l'aide d'un réseau BERT. Chaque jeton est encodé par un analyseur lexical, puis traité par le modèle BERT « base-uncased[47] », qui produit un vecteur encodé de 768 dimensions. L'objectif de la phase d'encodage est de créer une représentation vectorielle pour chaque jeton (mot) d'une phrase, en tenant compte du contexte dans lequel il apparaît, ce qui inclut les jetons environnants

[47] https://huggingface.co/transformers

dans la phrase. Le vecteur résultant de cette phase d'encodage sert de base d'analyse pour les deux couches de décodage.

Plus en détail, chaque phrase est introduite dans la structure d'encodage sous forme d'une séquence de jetons (x), produisant ainsi une séquence de vecteurs d'encodage, un pour chaque jeton (h). Ces vecteurs d'encodage générés subissent ensuite un ensemble de décodages spécifiques en fonction de la tâche. Ensuite, la phrase est évaluée par un $score_{RecommenderTag}$ dans le cadre de la classification de l'objet de la recommandation, attribuant ce score indépendamment de la présence ou de l'absence d'un objet (classe « hotel », « event » ou « none »). De plus, elle est évaluée par $score_{RecommenderType}$ dans le cadre de la classification du type de recommandation, attribuant ce score si un type est présent (classes « userbased » ou « itembased ») ou si le type est absent (classe « none »).

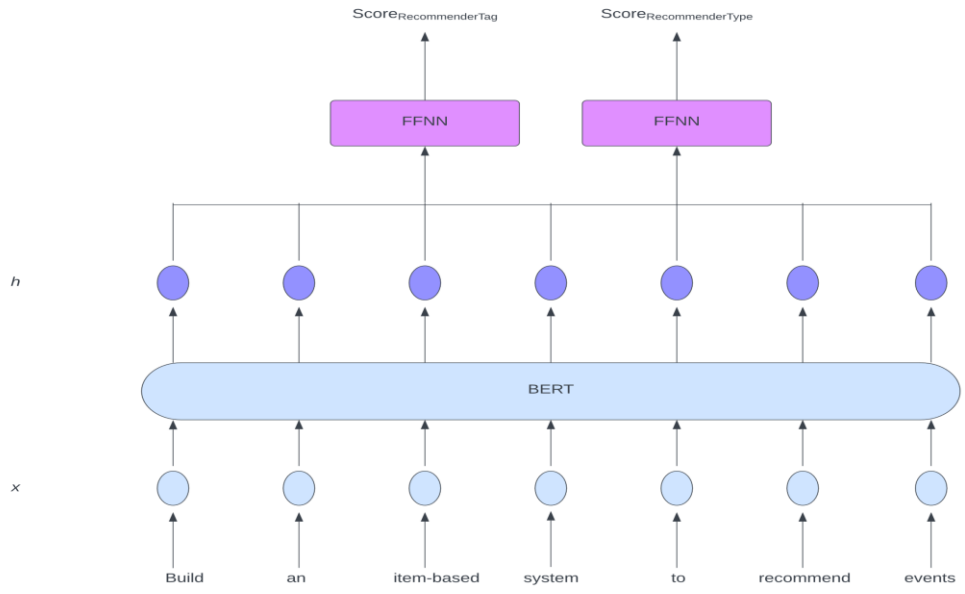


Figure 3.22 – Le modèle proposé de l'approche basée sur l'apprentissage profond.

3.4.2 Le jeu de données

3.4.2.1 La composition du jeu de données

Pour cette étude, nous avons initialement travaillé avec un jeu de données contenant 270 enregistrements audio. Après la transcription, nous avons identifié et supprimé les enregistrements audio en double, ce qui nous a laissés avec 185 enregistrements audio uniques. Ensuite, afin d'enrichir notre ensemble de données, nous avons appliqué diverses techniques d'augmentation audio, telles que la perturbation temporelle, l'ajout de bruit, la variation de volume et la perturbation de fréquence, ce qui nous a permis de générer un total de 300 enregistrements. Nous avons veillé à ce que chaque enregistrement audio augmenté conserve une transcription unique.

Étant donné que notre modèle de détection d'intention basé sur l'apprentissage profond doit effectuer deux tâches de classification distinctes, chaque enregistrement audio de notre ensemble de données est associé à une intention spécifique. Cette intention représente soit le type, soit l'objet de la recommandation, en fonction de la tâche de classification correspondante. Ces informations sont essentielles pour l'instanciation efficace d'un SR.

Nous avons divisé le jeu de données utilisé pour les deux tâches de classification en trois ensembles distincts : un ensemble d'entraînement, comprenant 210 enregistrements audio, ce qui représente 70 % de l'ensemble total des enregistrements ; un ensemble de validation, composé de 45 enregistrements audio, soit 15 % du total ; et enfin, un ensemble de test également constitué de 45 enregistrements audio, soit également 15 % du total. Cette répartition nous permet d'entraîner notre modèle, d'évaluer sa généralisation et d'évaluer son efficacité de manière robuste.

Nous avons fait appel à la plateforme Phonic AI[48] pour collecter les enregistrements audio au format « wav ». À l'aide d'une interface graphique conviviale, nous avons spécifié

[48] https://www.phonic.ai/

les différentes phrases que chaque participant devait prononcer. Cette démarche a impliqué la participation d'environ quarante participants. L'avantage de cette approche réside dans la diversité vocale obtenue dans notre jeu de données, comprenant une variété d'accents, de prononciations, et autres particularités linguistiques.

3.4.2.2 L'annotation du jeu de données

Dans notre jeu de données, chaque intention est associée à différentes phrases en anglais prononcées par les participants dans les enregistrements audio. Selon la tâche de classification, une phrase donnée peut représenter l'objet ou le type de système de la recommandation. Cette association entre les phrases et les intentions permet d'entraîner le modèle à prédire les intentions spécifiques à partir des enregistrements audio transcrits en texte.

Nous avons annoté les phrases avec les intentions correspondantes en fonction des tâches de classification. La tâche 1 concerne la classification en fonction de l'objet de la recommandation, tandis que la tâche 2 porte sur la classification en fonction du type de recommandation. Cette annotation est détaillée dans le tableau 1.

Les enregistrements audios sont renommés selon une convention spécifique après leur collecte. Chaque fichier audio est identifié par le nom de l'intention suivi d'un numéro de séquence. Par exemple, le premier enregistrement audio de l'intention « hotel » est nommé « hotel.00001.wav », le deuxième est nommé « hotel.00002.wav », et ainsi de suite. Cette méthode d'annotations des intentions et de nommage des fichiers facilite l'organisation et l'analyse des données. Un fichier Excel est utilisé pour stocker les informations suivantes pour chaque enregistrement audio :

- Nom du fichier audio
- Intention associée
- Transcription textuelle

Ce fichier Excel nous permet de conserver toutes les données nécessaires pour l'entraînement de notre modèle de classification.

Tableau 1. Tableau d'annotation des phrases et intentions pour la détection des classes à recommander, avec attribution de l'intention « none » aux phrases sans classe.

Phrase	**Intention**	
	Tâche 1	**Tâche 2**
Set up a recommendation for hotels. Generate hotel recommendation. Create a system to provide hotels.	hotel	none
Set up a user-based system to recommend hotels. Generate user-based system to recommend hotels. Create a user-based system to recommend hotels.	hotel	userbased
Build a system to provide event recommendations. Give an event recommendation. Design an event recommendation.	event	none
Build an item-based system to recommend events. Give an item-based system to recommend events. Design an item-based system to recommend events.	event	itembased

Set up a user-based recommendation system. Generate user-based recommendation system. Create a user-based recommendation system.	none	userbased
Build an item-based recommendation system. Give an item-based recommendation system. Design an item-based recommendation system.	none	itembased

3.5 SYNTHÈSE

Dans ce chapitre, nous avons exposé les objectifs de notre approche visant à accélérer le développement de la famille d'applications mobiles pour le tourisme intelligent. Nous avons décrit son fonctionnement ainsi que les différentes étapes qui la composent. De plus, nous avons mis en lumière les actifs de notre usine de logiciels nécessaires pour le développement d'applications mobiles de tourisme intelligent. Dans le prochain chapitre, nous détaillerons notre protocole de validation, les résultats obtenus, puis nous discuterons des conclusions tirées du processus de validation.

CHAPITRE 4
VALIDATION DES ACTIFS DE L'USINE DE LOGICIELS

Dans le chapitre précédent, nous avons introduit les actifs de notre usine de logiciels, conçus pour accélérer le développement d'applications mobiles dans le domaine du tourisme intelligent. Nous avons examiné en détail trois de ces actifs, centrés sur le développement des SR. Les deux premiers actifs, un DSL et des modèles de transformation, offrent un soutien à une approche de développement basée sur MDE pour les SR. Le troisième actif consiste en un modèle ASR permettant de modéliser les caractéristiques des SR à l'aide de commandes vocales. Dans ce chapitre, nous présenterons deux protocoles de validation que nous avons suivis pour évaluer ces trois actifs. Le premier protocole vise à évaluer les deux premiers actifs en examinant l'efficacité de l'approche MDE pour accélérer le processus de développement des SR. Le deuxième protocole se concentre sur le troisième actif, à savoir le modèle ASR.

4.1 L'ACCÉLÉRATION DU DÉVELOPPEMENT DES SYSTÈMES DE RECOMMANDATION

4.1.1 Le protocole

Pour évaluer l'efficacité de notre approche, nous avons réalisé une expérimentation. Celle-ci avait pour objectif de comparer les temps de développement du côté serveur d'une petite application mobile. Cette application offre des recommandations pour des attractions, des hôtels et des événements. Nous avons examiné deux scénarios : le développement traditionnel, impliquant une programmation manuelle à partir du PIM paramétré (voir figure 4.1), et la génération automatique du code à partir de ce même modèle.

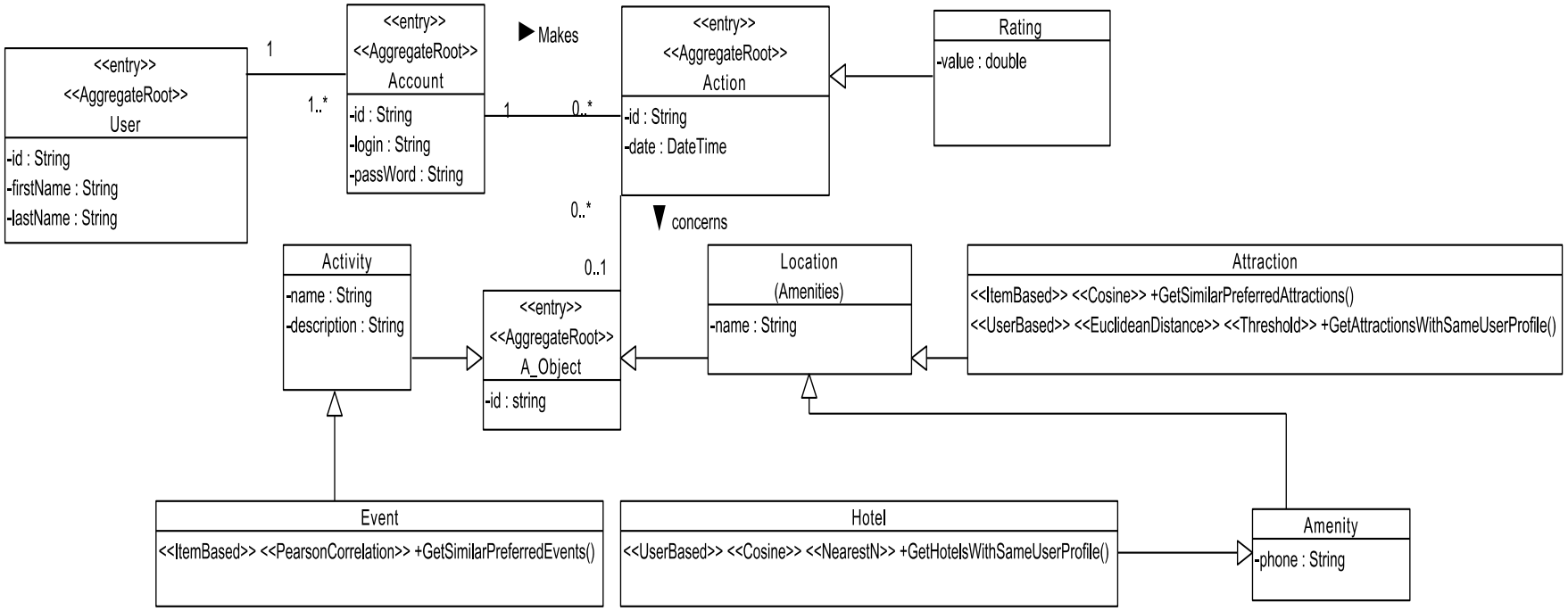


Figure 4.1 – Le PIM paramétré de l'exemple de validation.

Le tableau 2 résume les caractéristiques de chaque recommandation.

Tableau 2 : Fonctions de recommandation de l'exemple de validation.

Classe	Titre	Type	Voisinage	Similarité
Attraction	GetSimilarPreferredAttractions	Item Based	-	Cosine
Attraction	GetAttractionsWithSameUserProfile	User Based	Threshold	Euclidean Distance
Event	GetSimilarPreferredEvents	Item Based	-	Pearson Correlation
Hotel	GetHotelsWithSameUerProfile	User Based	NearestN	Cosine

Pour cette expérience, le développement de cette petite application a été confié à une équipe composée de cinq développeurs : un étudiant en dernière année d'études d'ingénieurs en informatique et quatre ingénieurs informaticiens en études supérieures. Les développeurs ont été assignés aléatoirement à l'un des deux scénarios[49]. Chacun d'entre eux a reçu des instructions précises et dispose d'un poste de travail équipé l'environnement de développement Visual Studio[50]. Les outils spécifiques incluent Visual Paradigm[51] pour la visualisation dans le scénario 1 et l'édition du modèle UML dans le scénario 2, ainsi

[49] Il est important de noter que dans les deux cas, le point de départ est un PIM. De plus, dans les deux scénarios, les équipes ont bénéficié d'une formation. Par exemple dans le cas du processus manuel, l'équipe doit également se former sur les différentes architectures utilisées, entre autres aspects. Le temps de formation et de familiarisation avec les deux techniques n'est pas pris en compte dans cette étude.

[50] https://visualstudio.microsoft.com/

[51] https://www.visual-paradigm.com/

qu'Entity Framework[52] et NReco d'Apache Mahout pour le scénario 1. Pour le scénario 2, nous utilisons aussi notre outil MDE, permettant l'application des transformations nécessaires.

4.1.2 Les résultats

Le tableau 3 présente une comparaison détaillée du nombre de lignes de code dans les deux scénarios. Dans le scénario 1, 270 lignes de code ont été écrites à la main, tandis que 237 lignes de code supplémentaires ont été réutilisées de diverses sources telles que des tutoriels et la documentation de NReco. Cela porte le total à 507 lignes de code pour le scénario 1. En revanche, dans le scénario 2, ces 507 lignes de code ont été entièrement générées par le biais de transformations de modèles, marquant une nette progression en ternes d'efficacité du développement. Cette automatisation a permis de réduire significativement le temps de développement, passant, en moyenne, de 160 minutes dans le scénario 1 à seulement 6 minutes dans le scénario 2.

Quant à la distribution du temps de développement moyen dans le scénario 2, l'analyse révèle que les développeurs ont alloué en moyenne 35,6% de leur temps à la sélection de fonctionnalités de recommandation via l'interface du DSL, 48,6% aux processus de modélisation et de transformations (y compris la paramétrisation du modèle etl'application des transformations), et les 16,6% restants à la configuration du code généré, incluant l'exécution des commandes de migration de la base de données. Cette répartition met en évidence une optimisation notable de l'allocation du temps, soulignant l'efficacité des transformations des modèles dans l'accélération du développement de logiciels.

[52] https://learn.microsoft.com/en-us/aspnet/entity-framework

Tableau 3. Répartition des lignes de code dans les scénarios de développement.

Type de ligne de code	**Scénario 1 (Développement manuel)**	**Scénario 2 (Généré à partir d'un modèle)**
Lignes écrites manuellement	270	--
Lignes réutilisées	237	--
Lignes générées par les transformations	--	507
Total[53]	**507**	**507**

4.1.3 Discussion

En évaluant la validité de l'expérience, certaines limites méritent d'être soulignées. Tout d'abord, l'expérience limitée des développeurs avec les outils de génération automatique de code basés sur des modèles UML constitue une préoccupation. Ainsi, une formation spécialisée serait bénéfique pour optimiser l'efficacité de cette approche. De plus, la taille de notre échantillon et le nombre de participants pourraient influencer les résultats,

[53] Le nombre de lignes de code généré automatiquement est identique à celui produit manuellement car il s'agit d'un code typique pour implanter ce genre système. Un programmeur bien familiarisé avec les technologies utilisées écrirait le même code que celui généré. De plus, la formation reçue par les développeurs incluait un exemple de code implémentant l'architecture, y compris le code supportant les fonctionnalités de recommandation. Généralement, ce code est répétitif ou semi-répétitif, c'est-à-dire qu'il comporte des sections standardisées qui sont fréquemment réutilisées avec de légères modifications basées sur les méta-données. Par exemple, le code des méthodes *Get* et *Set* des propriétés des classes est semi-répétitif, le seul changement étant lié au nom de la propriété.

soulignant l'importance d'une plus grande participation de développeurs professionnels pour une comparaison plus robuste.

Selon Rojas et Garrido (2017), l'adoption de bonnes pratiques de génie logiciel, couplée à une utilisation étendue d'UML et de techniques de génération automatique de code, peut accélérer significativement la création de prototypes de SR. Nos résultats expérimentaux confirment cette tendance, révélant des économies substantielles de temps grâce à l'utilisation de modèles abstraits pour automatiser le processus de développement.

En envisageant l'avenir, cette proposition vise à poser les bases pour des améliorations continues dans le développement de SR pour le tourisme intelligent. Des études ultérieures devraient explorer des approches complémentaires telles que le filtrage collaboratif et d'autres cadres d'application. En intégrant ces éléments, nous pourrons offrir des solutions encore plus adaptées et personnalisées, répondant efficacement aux besoins évolutifs des utilisateurs.

4.2 LA CLASSIFICATION DE COMMANDES VOCALES

4.2.1 La tâche de transcription automatique de la parole

4.2.1.1 Le protocole

Dans notre démarche de détection de commandes vocales, la première étape consiste à convertir un enregistrement audio en texte. Pour ce faire, nous utilisons le modèle jonatasgrosman/wav2vec2-large-xlsr-53-english, détaillé dans la sous-section b) de la section 3.4.1.2. Ce modèle, préentraîné sur Wav2Vec 2.0, chargé à l'aide des bibliothèques spécifiées dans le tableau 4 pour exécuter la transcription. Pour mener à bien cette expérimentation, nous nous appuyons sur l'architecture des machines virtuelles de l'environnement Google Colab utilisées, caractérisées par un système Linux 5.15.109+, un processeur Intel(R) Xeon(R) à une vitesse de 2,20 GHz et 2 cœurs de processeur.

Tableau 4. Bibliothèques requises pour la tâche de transcription.

Bibliothèque Python	**Version**
torch	1.12.1
pandas	1.5.3
transformers	4.31.0
simpletransformers	0.63.11
openpyxl	3.0.10
huggingsound	0.1.6

Pour évaluer quantitativement notre approche, nous nous sommes appuyées sur deux indicateurs : le taux d'erreur de mots (WER) et le taux d'erreur de caractères (CER). Ces métriques permettent d'évaluer respectivement le nombre de mots et de caractères mal identifiés par le modèle de transcription, comme décrit par (Panayotov et al., 2015).

Le WER est défini par l'équation suivante :

$$WER = (I + D + S) / N \quad (9)$$

, où :

- I représente le nombre de mots insérés dans une phrase,
- D représente le nombre de mots supprimés,
- S représente le nombre de mots substitués dans une phrase,
- N le nombre total de mots dans la phrase.

De manière similaire, le CER est calculé en remplaçant les mots par les caractères, toujours selon la même formule. Ainsi, un CER de 10 % indique que chaque dixième caractère (incluant lettres, ponctuation, espaces, etc.) n'a pas été correctement reconnu. Cela équivaut à une précision de 90 %.

4.2.1.2 Les résultats

Le tableau 5 illustre les résultats de transcriptions des enregistrements audio en texte en termes de WER et de CER. Ces résultats confirment que le modèle de transcription utilisé est efficace pour convertir avec un certain succès les enregistrements audio de notre jeu de données.

Tableau 5. Résultats de WER et de CER.

WER	**CER**
26.49%	6.40%

4.2.1.3 Discussion

Notre étude ambitionne d'accélérer le développement des SR en introduisant une approche de modélisation basée sur la détection de commandes vocales. Cette méthode recueille des informations essentielles sur le type de SR et l'objet de la recommandation. Nous avons adopté un modèle préentraîné basé sur wav2vec 2.0 avec l'architecture XLSR pour transcrire les enregistrements audio en anglais collectés lors de notre étude en texte. Grâce à l'apprentissage préalable multilingue, le modèle a affiché des performances supérieures sur différentes langues, en particulier en anglais.

Les performances de notre modèle d'ASR se traduisent par un WER de 26,49%, indiquant une transcription correcte d'environ 73,51% des mots. Le CER s'établit à 6,40%, témoignant d'une transcription précise pour environ 93,60% des caractères. Néanmoins, ces performances peuvent être influencées par des facteurs tels que la qualité audio ou la clarté

de la parole. Sur le jeu de données de Common Voice anglais, notre modèle a atteint un CER de 7,69%.

Intégrer un modèle de langage a boosté la précision de la transcription, réduisant le CER à 6,84%. Bien que cette amélioration puisse sembler marginale, elle est notable compte tenu de la taille conséquente du jeu de données. Pour des recherches futures, l'exploration de modèles de langage spécifiques à notre domaine pourrait s'avérer fructueuse. Par exemple, en utilisant des modèles de langage n-grammes, nous pourrions élaborer des modèles probabilistes optimisés pour anticiper le mot suivant dans une phrase, ce qui pourrait renforcer la précision, comme l'ont démontré (Zvornicanin, 2023).

En perspective, l'expansion de notre jeu de données et l'expérimentation avec diverses architectures de modèles d'ASR nous permettraient d'approfondir nos analyses, en alignement avec les travaux antérieurs tels que ceux de (de Morais & Saha, 2021).

4.2.2 La tâche de détection d'intention/de commande basée sur l'apprentissage profond

4.2.2.1 Le protocole

Dans notre approche, nous utilisons deux tâches de classification distinctes pour détecter l'intention à travers l'apprentissage profond. La première tâche associe chaque enregistrement audio à une intention qui identifie l'objet de la recommandation, tandis que la seconde associe l'enregistrement au type de recommandation. Ces associations sont nécessaires à l'instanciation d'un SR. Grâce à cette démarche, notre modèle peut prédire avec une grande précision les intentions spécifiques cachées derrière les enregistrements audio. Lors de la phase de prétraitement, les données sont traitées par lots, permettant l'extraction des phrases et des étiquettes du jeu de données. Les phrases sont ensuite normalisées pour atteindre une longueur uniforme, tandis que les étiquettes sont préparées pour l'entraînement. Ces étapes préparatoires sont vitales pour assurer l'efficacité de la détection d'intention en utilisant BERT.

Le modèle a été optimisé par une exploration méthodique de diverses configurations d'hyperparamètres. Pour les tâches de classification, nous avons testé différentes valeurs de taux d'apprentissage, à savoir 2e-5, 3e-5, et 4e-5, tout en maintenant un taux de dropout constant de 0.2. Les performances les plus élevées ont été atteintes avec un taux d'apprentissage de 4e-5 pour la première tâche, qui consiste à identifier l'objet de la recommandation, et de 2e-5 pour la seconde tâche, qui se focalise sur la détection de son type. Les paramètres incluent également une période d'entraînement fixée à 10 époques.

Un classificateur linéaire, activé par une fonction ReLU, a été employé pour formuler les prédictions de notre modèle. Durant le processus, nous avons recours à la technique de l'arrêt précoce pour identifier le moment idéal de conclusion de l'entraînement, ce qui nous permet d'éviter le surapprentissage et de maximiser la performance globale du modèle. L'arrêt précoce agit comme une méthode de régularisation en interrompant l'entraînement du réseau de neurones avant d'atteindre le nombre fixé d'époques ou d'itérations. Cette stratégie implique de suivre attentivement les performances du modèle sur un jeu de données de validation et d'arrêter l'entraînement dès que l'erreur de validation augmente ou cesse de diminuer. Ainsi, elle prévient le risque que le réseau apprenne de manière excessive à partir des données d'entraînement, compromettant sa capacité à généraliser à de nouvelles données, comme souligné par (Malinowski, 2023).

La fonction de perte d'entropie croisée a été sélectionnée pour évaluer à la fois l'entraînement et la validation du modèle dans les deux tâches de classification, comme détaillé dans les figures 4.2 et 4.3. Essentielle à l'optimisation du modèle, l'entropie croisée mesure la différence entre les probabilités prédites par le modèle et les probabilités réelles observées dans les données d'entraînement. La valeur de la perte varie de 0 à 1, où 0 indique une prédiction parfaitement en accord avec les données réelles. L'objectif est de réduire autant que possible cette valeur de perte, visant idéalement vers zéro, pour accroître l'efficacité du modèle.

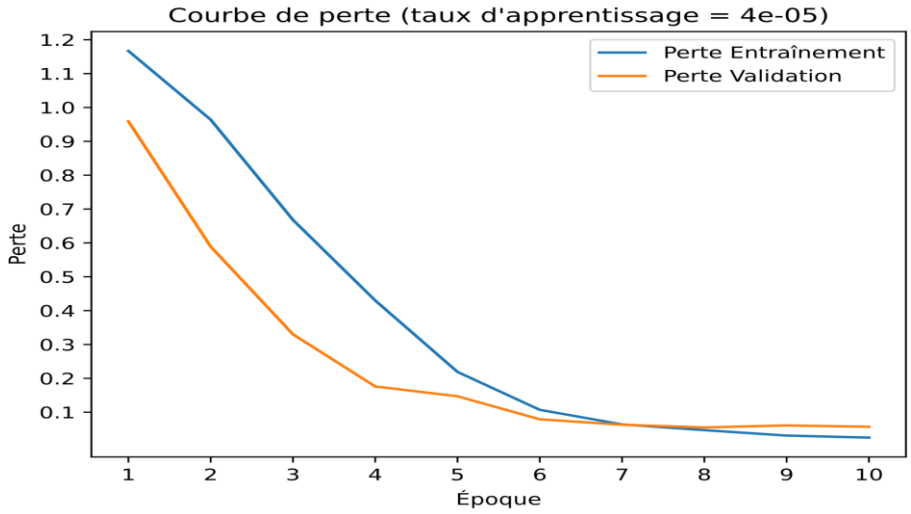


Figure 4.2 – Illustration de l'évolution de la fonction de perte calculée pour le modèle sur les ensembles d'entraînement et de validation pour la tâche de classification de l'objet de la recommandation. Le modèle optimal ayant été obtenu à l'époque 9.

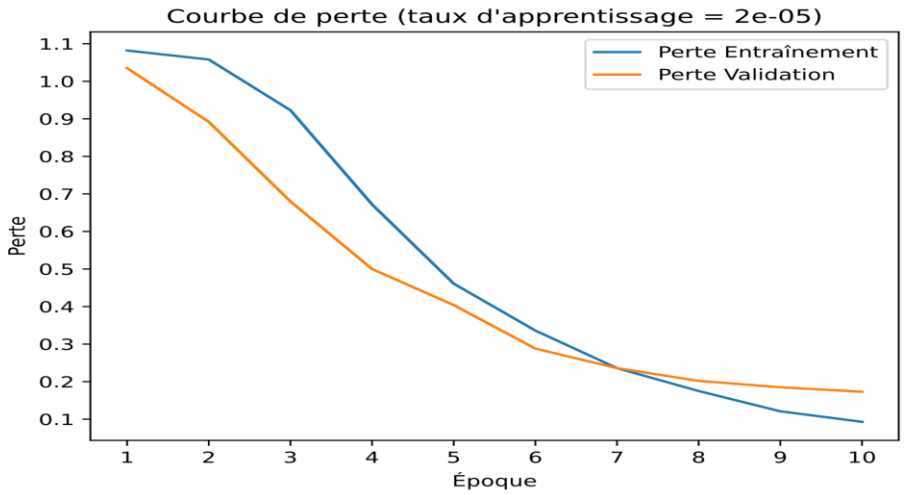


Figure 4.3 – Illustration de l'évolution de la fonction de perte calculée pour le modèle sur les ensembles d'entraînement et de validation pour la tâche de classification du type de recommandation. Le modèle optimal ayant été obtenu à l'époque 10.

Étant donné que la tâche de détection d'intention peut être considérée comme une tâche de classification, nous avons utilisé les métriques suivantes pour évaluer notre approche :

- Précision

  La précision est le pourcentage de prédictions correctes parmi toutes les prédictions positives. En termes d'équation mathématique, nous pouvons la définir comme suit :

$$Précision = VP / (VP + FP) \quad (10)$$

- Rappel

  Le rappel est le pourcentage de prédictions correctes parmi toutes les prédictions positives réelles. En termes d'équation mathématique, nous pouvons le définir comme suit :

$$Rappel = VP / (VP + FN) \quad (11)$$

- Score F1 :

  Le score F1 mesure l'équilibre entre la précision et le rappel. En termes d'équation mathématique, nous pouvons le définir comme suit :

$$Score\ F1 = 2 * (Précision * Rappel) / (Précision + Rappel) \quad (10)$$

Où :

VP = Vrai Positif

FP = Faux Positif

FN = Faux Négatif

4.2.2.2 Les résultats

Les résultats optimaux obtenus pour chaque tâche de classification, après avoir sélectionné le taux d'apprentissage le plus efficace, sont synthétisés dans le tableau 6 pour la première tâche et le tableau 7 pour la seconde.

Pour la tâche de classification de l'objet de la recommandation, illustrée par la figure 4.4, les courbes F1 affichent un score F1 maximal de 1.0 pour l'ensemble d'entraînement et de 96,4 pour l'ensemble de validation. De manière similaire, dans le cas de la classification du type de recommandation, présentée dans la figure 4.5, les scores F1 maximaux atteints sont de 1.0 pour l'ensemble d'entraînement et de 96,7 pour l'ensemble de validation. Ces scores illustrent l'excellence de la performance du modèle.

Tableau 6. Résultats obtenus par le modèle pour la tâche de détection de l'objet de la recommandation.

Taux d'apprentissage	Précision	Rappel	F1
4e-5	96,9	97	96,4

Tableau 7. Résultats obtenus par le modèle pour la tâche de détection du type de recommandation.

Taux d'apprentissage	Précision	Rappel	F1
2e-5	96,9	96,3	96,7

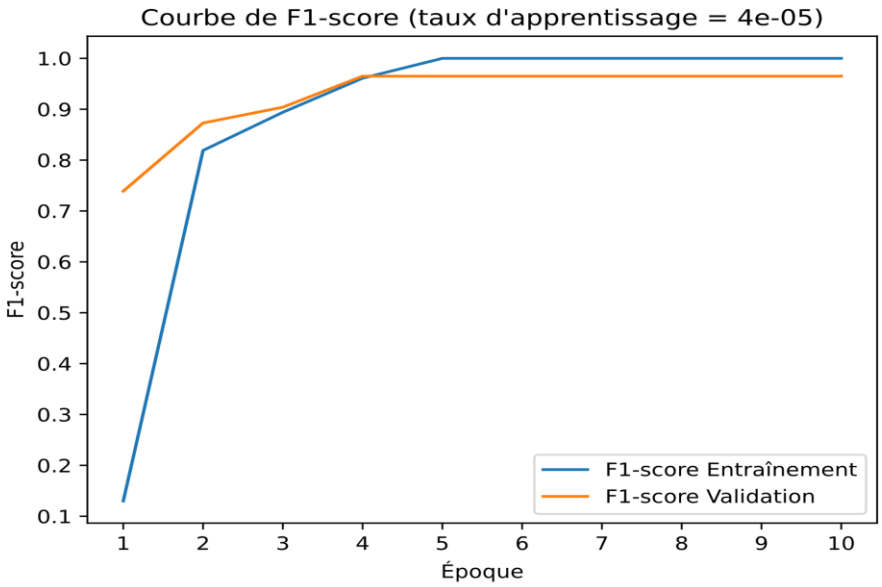


Figure 4.4 – Évolution du score F1 du modèle sur les ensembles d'entraînement et de validation pour la classification de l'objet de la recommandation.

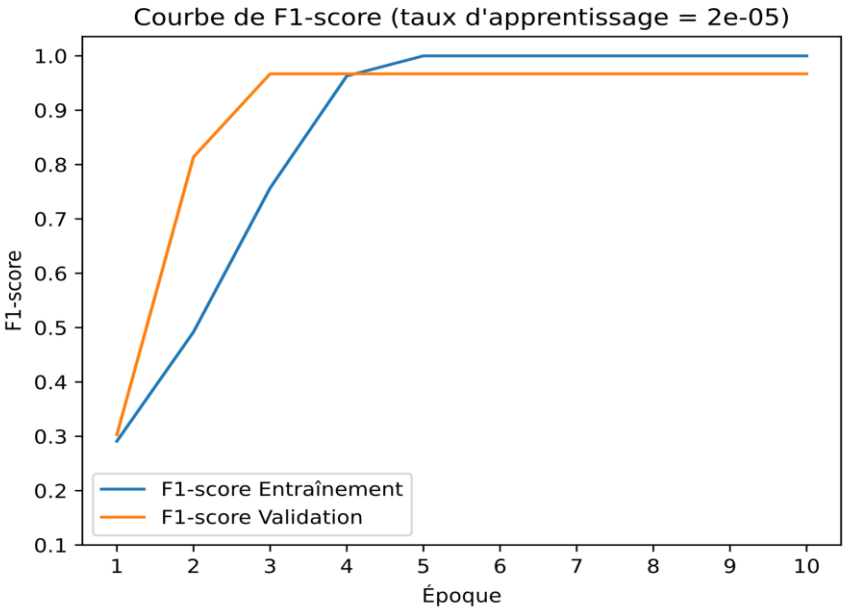


Figure 4.5 – Évolution du score F1 du modèle sur les ensembles d'entraînement et de validation pour la classification du type de recommandation.

4.2.2.3 Discussion

Dans notre approche, la détection d'intention basée sur l'apprentissage profond est effectuée grâce à un modèle basé sur BERT, qui est un modèle de traitement du langage naturel utilisant une architecture de transformateur préentraînée sur de vastes ensembles de données non étiquetées lui permettant de capturer des informations syntaxiques et sémantiques complexes dans le langage naturel pour la détection d'intention. La tâche de détection d'intention consistant à assigner la commande appropriée à la transcription audio effectuée préalablement grâce à un modèle ASR préentraîné. Chaque transcription est associée à une intention représentant soit l'objet de la recommandation, soit le type de recommandation dans le contexte de l'instanciation d'un SR, permettant ainsi au modèle de prédire avec précision les intentions spécifiques des enregistrements audio. Malgré le jeu de données de faible taille utilisé avec le modèle BERT, des résultats positifs ont été obtenus. En combinant les résultats des deux tâches de classification du modèle, nous obtenons une moyenne de précision de 96,9% et une moyenne de score F1 de 96,5%.

Nous avons utilisé différentes techniques pour améliorer la performance du modèle, telles que l'arrêt précoce pour éviter l'apprentissage excessif et la perte de croisement dans le but d'améliorer le processus d'entraînement pour une meilleure détection des intentions. Un des avantages de BERT est que l'étape d'extraction des caractéristiques n'est pas nécessaire. Alors que différentes techniques d'extraction de caractéristiques sont requises dans les modèles de base, l'extraction de caractéristiques est automatique avec le BERT basé sur l'apprentissage profond. De plus, les étapes de prétraitement telles que le découpage, la division de n-gramme ne sont pas nécessaires pour l'architecture BERT. Les expériences peuvent être réalisées en quelques étapes, telles que l'extraction de caractères redondants, lors du prétraitement. Cela montre que, contrairement aux algorithmes d'apprentissage automatique traditionnels, BERT peut être utilisé dans le cadre de la détection d'intention sans avoir besoin de méthodes de prétraitement et d'extraction de caractéristiques supplémentaires et des résultats positifs peuvent être obtenus (Çelıkten & Bulut, 2021).

Comme travaux futurs, nous prévoyons d'étendre le jeu de données en y ajoutant de nouvelles intentions. Dans ce cas, des comparaisons peuvent être faites avec d'autres méthodes basées sur les transformateurs tels que RoBERTa (Robustly Optimized BERT Pretraining Approach – Y. Liu et al. (2019)), ALBERT (A Lite BERT – Lan et al. (2019)), DistilBERT (Distilled version of BERT – Sanh et al. (2019)), ou encore XLNet (eXtreme Language understanding Network – Yang et al. (2019)), en plus de BERT. Des études peuvent être menées pour comparer les performances des réalisations de détection d'intention en utilisant également d'autres modèles basés sur des réseaux de neurones artificiels (Grossi & Buscema, 2007).

4.3 SYNTHÈSE

L'étude que nous avons menée vise à valider la possibilité d'accélérer le développement de la famille d'applications mobiles pour le tourisme intelligent grâce approche basée sur l'utilisation de modèles pour décrire et automatiser le processus de développement. Pour cela, nous avons comparé les temps de développement d'un ensemble de SR basé sur le cadre d'application NReco d'Apache Mahout, en considérant deux scénarios : l'un où le code du modèle est entièrement implémenté manuellement, et l'autre où le code du modèle est généré. Les résultats obtenus ont démontré que l'utilisation de modèles pour décrire et automatiser le processus de développement permet une économie de temps considérable. Dans le cadre de cette étude, nous proposons une approche de modélisation des SR basée sur la détection de commandes vocales. Nous avons utilisé un modèle pré-entraîné de reconnaissance automatique de la parole basé sur wav2vec 2.0 pour transcrire des enregistrements audio en texte, puis un modèle basé sur BERT pour détecter l'intention dans le texte contenant les informations nécessaires pour instancier un SR. Malgré la taille modeste de notre jeu de données, nous avons obtenu d'excellents résultats dans la détection d'intention, en partie grâce à une bonne qualité de transcription des enregistrements audio.

CONCLUSION GÉNÉRALE

Afin de stimuler le succès et la croissance de l'industrie touristique, de nombreux acteurs se tournent vers les applications mobiles de tourisme intelligent. Ces applications visent à enrichir l'expérience des voyageurs à chaque étape de leur parcours : avant, pendant et après le voyage. Dans cette optique, plusieurs destinations touristiques adoptent le cadre 6A dans leurs applications pour renforcer la gestion des destinations touristiques intelligentes.

Parmi les fonctionnalités essentielles d'une application de tourisme intelligent, les systèmes de recommandation jouent un rôle prépondérant. Ils offrent des recommandations sur mesure aux utilisateurs en se basant sur leurs préférences, intérêts et historique de voyages. Toutefois, le développement de telles applications requiert des ressources considérables en termes de main-d'œuvre, de temps et d'investissement. Dans cette perspective, la création d'une usine de logiciels dédiée au développement d'applications spécialisées pourrait offrir une solution efficace. Cette approche permettrait de rationaliser les coûts et d'accélérer le processus de mise en œuvre des systèmes logiciels.

Dans ce mémoire, nous avons introduit une approche pour la création d'une usine de logiciels dédiée au développement d'applications mobiles pour le tourisme intelligent. Plus spécifiquement, notre démarche vise à concevoir un DSL, un actif essentiel de cette usine, permettant à un analyste de créer une représentation abstraite de haut niveau des systèmes de recommandation pour une application mobile. Ce DSL, une fois combiné avec d'autres modèles issus d'actifs divers, facilite la génération automatisée du code source de l'application mobile grâce à des transformations spécifiques.

Nous avons adopté MDE pour supporter la variabilité dans notre usine de logiciels. Cette intégration nous permet de développer efficacement des applications mobiles

personnalisées pour le tourisme intelligent. En effet, nos résultats ont démontré une réduction significative du temps nécessaire, passant de 160 minutes dans un cadre de développement traditionnel à seulement 6 minutes grâce à l'utilisation de notre usine.

À l'heure actuelle, notre approche se concentre exclusivement sur le développement de systèmes de recommandation basés sur le filtrage collaboratif. Pour les prochaines étapes, il sera essentiel d'étendre notre usine de logiciels afin de prendre en charge d'autres méthodes de collaboration et d'intégrer différents cadres d'application pour l'implémentation de ces systèmes.

Dans notre démarche, nous avons exploré la reconnaissance vocale pour modéliser les systèmes de recommandation des applications mobiles de tourisme intelligent. Nous avons utilisé un modèle ASR basé sur wav2vec 2.0 pour convertir des enregistrements audio en texte. Notre jeu de données comprenait 300 enregistrements vocaux , 46 par commande. Nous avons adopté diverses méthodes, notamment des approches basées sur des règles et l'apprentissage profond, pour détecter ces commandes. Malgré la taille limitée du jeu de données, nos résultats étaient prometteurs : un WER de 26,49%, un CER de 6,40%, indiquant une précision de transcription d'environ 93,60% des caractères, et un score F1 moyen de 96,7% pour la détection de commandes avec l'apprentissage profond.

La reconnaissance vocale des commandes est un élément clé de notre initiative visant à dynamiser le processus de développement au sein de notre usine de logiciels. Bien que cette technologie soit actuellement limitée à certaines fonctionnalités, elle peut susciter des interrogations sur son utilité face à des méthodes interactives simples, telles que la sélection de commandes via une interface utilisateur. Néanmoins, l'intégration de technologies innovantes facilite l'accès à des automatisations plus avancées. À l'avenir, nous prévoyons de développer notre support de reconnaissance vocale pour permettre un processus de développement continu, en interaction directe avec une usine de logiciels.

RÉFÉRENCES BIBLIOGRAPHIQUES

Abdelmalek, H., & Khriss, I. (2023). *Towards an Effective Approach for Composition of Model Transformations.* [Technical report].

Abdul-Kader, S. A., & Woods, J. C. (2015). Survey on chatbot design techniques in speech conversation systems. *International Journal of Advanced Computer Science and Applications*, *6*(7).

Abrahamsson, P., Hanhineva, A., Hulkko, H., Ihme, T., Jäälinoja, J., Korkala, M., Koskela, J., Kyllönen, P., & Salo, O. (2004). Mobile-D: an agile approach for mobile application development. Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications,

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, *17*(6), 734-749.

Agarap, A. F. (2017). An Architecture Combining Convolutional Neural Network (CNN) and Support Vector Machine (SVM) for Image Classification. arXiv:1712.03541. Retrieved December 01, 2017, from https://ui.adsabs.harvard.edu/abs/2017arXiv171203541A

Aggarwal, C. C., & Reddy, C. K. (2014). Data clustering. *Algorithms and Applications*, 54.

Alaoui, L. L., ASBAI, M., & BENAMMI, M. H. (2018). L'impact du marketing digital sur le tourisme marocain. *Public & Nonprofit Management Review*, *3*(1).

Allen, J. (1995). *Natural language understanding*. Benjamin-Cummings Publishing Co., Inc.

Almonte, L., Cantador, I., Guerra, E., & de Lara, J. (2020). Towards automating the construction of recommender systems for low-code development platforms. Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings,

Amatya, S. (2013). Cross-platform mobile development: An alternative to native mobile development. In.

Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., & Chen, G. (2016). Deep speech 2: End-to-end speech recognition in english and mandarin. International conference on machine learning,

Ampatzoglou, A., Michou, O., & Stamelos, I. (2013). Building and mining a repository of design pattern instances: Practical and research benefits. *Entertainment Computing*, *4*(2), 131-142.

Anderson, C. (2012). The model-view-viewmodel (mvvm) design pattern. In *Pro Business Applications with Silverlight 5* (pp. 461-499). Springer.

Ardila, R., Branson, M., Davis, K., Henretty, M., Kohler, M., Meyer, J., Morais, R., Saunders, L., Tyers, F. M., & Weber, G. (2019). Common voice: A massively-multilingual speech corpus. *arXiv preprint arXiv:1912.06670*.

Assael, Y. M., Shillingford, B., Whiteson, S., & De Freitas, N. (2016). Lipnet: End-to-end sentence-level lipreading. *arXiv preprint arXiv:1611.01599*.

Avila, A. R., Rezagholizadeh, M., & Xing, C. (2023). Multimodal Audio-textual Architecture for Robust Spoken Language Understanding. *arXiv preprint arXiv:2306.06819*.

Azwary, F., Indriani, F., & Nugrahadi, D. T. (2016). Question answering system berbasis artificial intelligence markup language sebagai media informasi. *Klik-Kumpulan Jurnal Ilmu Komputer*, *3*(1), 48-60.

Bachmann, F., & Clements, P. (2005). *Variability in software product lines*. Carnegie Mellon University, Software Engineering Institute.

Baevski, A., Zhou, Y., Mohamed, A., & Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in Neural Information Processing Systems*, *33*, 12449-12460.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. *ArXiv*, *1409*.

Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., & Bengio, Y. (2016). End-to-end attention-based large vocabulary speech recognition. 2016 IEEE international conference on acoustics, speech and signal processing (ICASSP),

Batory, D., Johnson, C., MacDonald, B., & von Heeder, D. (2000). Achieving extensibility through product-lines and domain-specific languages: A case study. Software Reuse: Advances in Software Reusability: 6th International Conference, ICSR-6, Vienna, Austria, June 27-29, 2000. Proceedings 6,

Benckendorff, P., Sheldon, P., & Fesenmaier, D. (2014). Tourism information technology. CAB International. In: Oxford, UK.

Bergin Jr, T. J., & Gibson Jr, R. G. (1996). *History of programming languages---II*. ACM.

Bézivin, J. (2006). Model Driven Engineering: An Emerging Technical Space. In *Generative and Transformational Techniques in Software Engineering* (pp. 36-64). http://dx.doi.org/10.1007/11877028_2

Bézivin, J., Dupé, G., Jouault, F., Pitette, G., & Rougui, J. E. (2003). First experiments with the ATL model transformation language: Transforming XSLT into XQuery. 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture,

Bhat, H. R., Lone, T. A., & Paul, Z. M. (2017). Cortana-intelligent personal digital assistant: A review. *International Journal of Advanced Research in Computer Science*, *8*(7), 55-57.

Bhoi, A. K., de Albuquerque, V. H. C., Srinivasu, P. N., & Marques, G. (2022). *Cognitive and Soft Computing Techniques for the Analysis of Healthcare Data*. Academic Press.

Biau, G., & Scornet, E. (2016). A random forest guided tour. *Test*, *25*, 197-227.

Brandtzaeg, P. B., & Følstad, A. (2017). Why people use chatbots. Internet Science: 4th International Conference, INSCI 2017, Thessaloniki, Greece, November 22-24, 2017, Proceedings 4,

Bringmann, K., & Künnemann, M. (2015). Quadratic conditional lower bounds for string problems and dynamic time warping. 2015 IEEE 56th Annual Symposium on Foundations of Computer Science,

Brown, A. W. (2004). Model driven architecture: Principles and practice. *Software and Systems Modeling*, *3*, 314-327.

Budulan, S. (2018). Chatbot Categories and Their Limitations. https://dzone.com/articles/chatbots-categories-and-their-limitations-1

Budzianowski, P., Wen, T.-H., Tseng, B.-H., Casanueva, I., Ultes, S., Ramadan, O., & Gašić, M. (2018). Multiwoz--a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling. *arXiv preprint arXiv:1810.00278*.

Buhalis, D. (2003). *eTourism: Information technology for strategic tourism management*. Pearson education.

Buhalis, D., & Amaranggana, A. (2013). Smart tourism destinations. Information and Communication Technologies in Tourism 2014: Proceedings of the International Conference in Dublin, Ireland, January 21-24, 2014,

Buhalis, D., & Law, R. (2008). Progress in information technology and tourism management: 20 years on and 10 years after the Internet—The state of eTourism research. *Tourism management*, *29*(4), 609-623.

Burke, R., Felfernig, A., & Göker, M. H. (2011). Recommender systems: An overview. *Ai Magazine*, *32*(3), 13-18.

Cantu, J. (2023). Advancing Speech Recognition with Transfer Learning Techniques. https://medium.com/@jesus.cantu217/advancing-speech-recognition-with-transfer-learning-techniques-949bc65f655

Casanueva, I., Temčinas, T., Gerz, D., Henderson, M., & Vulić, I. (2020). Efficient intent detection with dual sentence encoders. *arXiv preprint arXiv:2003.04807*.

Çelıkten, A., & Bulut, H. (2021). Turkish medical text classification using bert. 2021 29th signal processing and communications applications conference (SIU),

Chan, W., Jaitly, N., Le, Q. V., & Vinyals, O. (2015). Listen, attend and spell. *arXiv preprint arXiv:1508.01211*.

Chan, W., Zhang, Y., Le, Q., & Jaitly, N. (2016). Latent sequence decompositions. *arXiv preprint arXiv:1610.03035*.

Charland, A., & Leroux, B. (2011). Mobile application development: web vs. native. *Communications of the ACM*, *54*(5), 49-53.

Chen, Y.-P., Price, R., & Bangalore, S. (2018). Spoken language understanding without speech recognition. 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),

Chénard, G., Khriss, I., & Salah, A. (2012). Towards the automatic discovery of platform transformation templates of legacy object-oriented systems. Proceedings of the 6th International Workshop on Models and Evolution,

Chorowski, J., Bahdanau, D., Cho, K., & Bengio, Y. (2014). End-to-end continuous speech recognition using attention-based recurrent nn: First results. *arXiv preprint arXiv:1412.1602*.

Chorowski, J., & Jaitly, N. (2016). Towards better decoding and language model integration in sequence to sequence models. *arXiv preprint arXiv:1612.02695*.

Ciayandi, A., Mawardi, V. C., & Hendryli, J. (2020). Retrieval based chatbot on tarumanagara university with Multilayer Perceptron. IOP Conference Series: Materials Science and Engineering,

Collobert, R., Hannun, A., & Synnaeve, G. (2019). A fully differentiable beam search decoder. International Conference on Machine Learning,

Conneau, A., Baevski, A., Collobert, R., Mohamed, A., & Auli, M. (2020). Unsupervised cross-lingual representation learning for speech recognition. *arXiv preprint arXiv:2006.13979*.

Cook, S. (2004). Domain-Specific Modeling and Model-Driven Architecture. *MDA Journal*.

Coucke, A., Saade, A., Ball, A., Bluche, T., Caulier, A., Leroy, D., Doumouro, C., Gisselbrecht, T., Caltagirone, F., & Lavril, T. (2018). Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*.

Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., & Wąsowski, A. (2012). Cool features and tough decisions: a comparison of variability modeling approaches. Proceedings of the 6th International workshop on variability modeling of software-intensive systems,

Dai, W., Cahyawijaya, S., Yu, T., Barezi, E. J., Xu, P., Yiu, C. T. S., Frieske, R., Lovenia, H., Winata, G. I., & Chen, Q. (2022). Ci-avsr: A cantonese audio-visual speech dataset for in-car command recognition. *arXiv preprint arXiv:2201.03804*.

Davis, S., & Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, *28*(4), 357-366.

de Morais, R. A., & Saha, B. N. (2021). End-to-End Speech Recognition Using Recurrent Neural Network (RNN). *AIJR Proceedings*, 152-158.

Deng, A. (2022). NLG In Conversational Ai: The Challenges Of Generating Language. https://www.soapboxlabs.com/blog/the-role-of-nlg-in-conversational-ai/

Deursen, A. v., Marin, M., & Moonen, L. (2003). *Aspect Mining and Refactoring* Proceeding of First Intl. Workshop on REFactoring: Achievements, Challenges, Effects (REFACE),

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* https://doi.org/10.18653/v1/n19-1423

https://aclanthology.org/N19-1423/

Di Sipio, C., Di Ruscio, D., & Nguyen, P. T. (2020). Democratizing the development of recommender systems by means of low-code platforms. Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems: companion proceedings,

Di Sipio, C., Rocco, J., Ruscio, D., & Nguyen, P. T. (2022). LEV4REC: A Low-Code Environment to Support the Development of Recommender Systems.

Dumais, S. T. (2004). Latent semantic analysis. *Annu. Rev. Inf. Sci. Technol.*, *38*(1), 188-230.

Eide, E., & Gish, H. (1996). A parametric approach to vocal tract length normalization. 1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings,

Ewald, R., & Uhrmacher, A. M. (2014). SESSL: A domain-specific language for simulation experiments. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, *24*(2), 1-25.

Falk, K. (2019). *Practical recommender systems*. Simon and Schuster.

Fang, C. (2009). From dynamic time warping (DTW) to hidden markov model (HMM). *University of Cincinnati*, *3*, 19.

Fluent.ai. (2020). Fluent Speech Commands: A dataset for spoken language understanding research. https://fluent.ai/fluent-speech-commands-a-dataset-for-spoken-language-understanding-research/

Forney, G. D. (1973). The viterbi algorithm. *Proceedings of the IEEE*, *61*(3), 268-278.

Fowler, M. (2010). *Domain-specific languages*. Pearson Education.

Frias-Martinez, E., Chen, S. Y., & Liu, X. (2009). Evaluation of a personalized digital library based on cognitive styles: Adaptivity vs. adaptability. *International Journal of Information Management*, *29*(1), 48-56.

Frias-Martinez, E., Magoulas, G., Chen, S., & Macredie, R. (2006). Automated user modeling for personalized digital libraries. *International Journal of Information Management*, *26*(3), 234-248.

Friedman, N., Geiger, D., & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine learning*, *29*, 131-163.

Gales, M., & Young, S. (2008). The application of hidden Markov models in speech recognition. *Foundations and Trends® in Signal Processing*, *1*(3), 195-304.

Goodfellow, I. B., Y.; and Courville, A. (2016). *Deep Learning*. MIT Press.

Google. (2018). *Speech commands dataset version 2*. http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz

Graves, A., Fernández, S., Gomez, F., & Schmidhuber, J. (2006). Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. Proceedings of the 23rd international conference on Machine learning,

Graves, A., & Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. International conference on machine learning,

Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM networks. Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.,

Greenfield, J., & Short, K. (2003). Software factories: assembling applications with patterns, models, frameworks and tools. Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications,

Greenfield, J., Short, K., Cook, S., & Kent, S. (2004). *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. John Wiley & Sons, Inc.

Gretzel, U., Sigala, M., Xiang, Z., & Koo, C. (2015). Smart tourism: foundations and developments. *Electronic Markets*, *25*, 179-188.

Grossi, E., & Buscema, M. (2007). Introduction to artificial neural networks. *European journal of gastroenterology & hepatology*, *19*(12), 1046-1054.

Guinness, H. (2023). How does ChatGPT work? https://zapier.com/blog/how-does-chatgpt-work/#

Gulati, A., Qin, J., Chiu, C.-C., Parmar, N., Zhang, Y., Yu, J., Han, W., Wang, S., Zhang, Z., & Wu, Y. (2020). Conformer: Convolution-augmented transformer for speech recognition. *arXiv preprint arXiv:2005.08100*.

Gupta, S. (2020). Natural Language Processing Use Case – How Do Personal Assistant Apps Work? https://www.springboard.com/blog/data-science/nlp-use-cases/

Han, X., Zhang, Z., Ding, N., Gu, Y., Liu, X., Huo, Y., Qiu, J., Yao, Y., Zhang, A., & Zhang, L. (2021). Pre-trained models: Past, present and future. *AI Open*, *2*, 225-250.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., & Coates, A. (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

Hardebolle, C. (2008). *Composition de modèles pour la modélisation multi-paradigme du comportement des systèmes* Université Paris Sud-Paris XI].

Haristiani, N. (2019). Artificial Intelligence (AI) chatbot as language learning medium: An inquiry. Journal of Physics: Conference Series,

Hermans, F., Pinzger, M., & Van Deursen, A. (2009). Domain-specific languages in practice: A user study on the success factors. Model Driven Engineering Languages and Systems: 12th International Conference, MODELS 2009, Denver, CO, USA, October 4-9, 2009. Proceedings 12,

Herndon, R. M., & Berzins, V. A. (1988). The realizable benefits of a language prototyping language. *IEEE Transactions on Software Engineering*, *14*(6), 803-809.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., & Sainath, T. N. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, *29*(6), 82-97.

Huertas, A., Moreno, A., & My, T. H. (2019). Which destination is smarter? Application of the (SA) 6 framework to establish a ranking of smart tourist destinations. *International Journal of Information Systems and Tourism (IJIST)*, *4*(1), 19-28.

Hunt, A. J., & Black, A. W. (1996). Unit selection in a concatenative speech synthesis system using a large speech database. 1996 IEEE international conference on acoustics, speech, and signal processing conference proceedings,

Hussain, S., Ameri Sianaki, O., & Ababneh, N. (2019). A survey on conversational agents/chatbots classification and design techniques. Web, Artificial Intelligence and Network Applications: Proceedings of the Workshops of the 33rd International Conference on Advanced Information Networking and Applications (WAINA-2019) 33,

Hutchinson, J., Whittle, J., Rouncefield, M., & Kristoffersen, S. (2011). Empirical assessment of MDE in industry. Proceedings of the 33rd international conference on software engineering,

Irmanti, D., Hidayat, M. R., Amalina, N. V., & Suryani, D. (2017). Mobile smart travelling application for indonesia tourism. *Procedia Computer Science*, *116*, 556-563.

Iung, A., Carbonell, J., Marchezan, L., Rodrigues, E., Bernardino, M., Basso, F. P., & Medeiros, B. (2020). Systematic mapping study on domain-specific language development tools. *Empirical Software Engineering*, *25*, 4205-4249.

Jannach, D., Zanker, M., Felfernig, A., & Friedrich, G. (2010). *Recommender systems: an introduction*. Cambridge University Press.

Jurafsky, D., & Martin, J. H. (2009). *Speech and Language Processing (2nd Edition)*. Prentice-Hall, Inc.

Jyothi, P. (2019). Automatic Speech Recognition. https://www.cse.iitb.ac.in/~pjyothi/cs753/

Kang, K. C., Hess, J. A., William, E., & Novak, A. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study.

Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E., & Huh, M. (1998). FORM: A feature-; oriented reuse method with domain-; specific reference architectures. *Annals of Software Engineering*, *5*(1), 143-168.

Karaali, O., Corrigan, G., & Gerson, I. (1996). Speech synthesis with neural networks. World Congress on Neural Networks, San Diego,

Kavlakoglu, E. (2020). NLP vs. NLU vs. NLG: the differences between three natural language processing concepts. https://www.ibm.com/blog/nlp-vs-nlu-vs-nlg-the-differences-between-three-natural-language-processing-concepts/

Khan, O. Z., Robichaud, J.-P., Crook, P. A., & Sarikaya, R. (2015). Hypotheses ranking and state tracking for a multi-domain dialog system using multiple ASR alternates. Sixteenth Annual Conference of the International Speech Communication Association,

Khan, V., & Meenai, T. A. (2021). Pretrained Natural Language Processing Model for Intent Recognition (BERT-IR). *Human-Centric Intelligent Systems*, *1*(3-4), 66-74.

Khouzaimi, H. (2022). Les agents conversationnels avec python (chatbot). https://www.stat4decision.com/fr/les-agents-conversationnels-avec-python-chatbot/

Khriss, I., Jakimi, A., & Abdelmalek, H. (2020). Towards an Effective Implementation of a Model-Driven Engineering Approach for Software Development. 2020 1st International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET),

Kim, J. K., Kim, H. K., Oh, H. Y., & Ryu, Y. U. (2010). A group recommendation system for online communities. *International Journal of Information Management*, *30*(3), 212-219.

Kim, Y.-B., Kim, D., Kumar, A., & Sarikaya, R. (2018). Efficient large-scale neural domain classification with personalized attention. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers),

Klatt, D. H. (1987). Review of text-to-speech conversion for English. *The Journal of the Acoustical Society of America*, *82*(3), 737-793.

Klema, V., & Laub, A. (1980). The singular value decomposition: Its computation and some applications. *IEEE Transactions on automatic control*, *25*(2), 164-176.

Koo, C., Gretzel, U., Hunter, W. C., & Chung, N. (2015). The role of IT in tourism. *Asia Pacific Journal of Information Systems*, *25*(1), 99-102.

Kumar, N., & Andreou, A. G. (1998). Heteroscedastic discriminant analysis and reduced rank HMMs for improved speech recognition. *Speech communication*, *26*(4), 283-297.

Lamel, L., Rabiner, L., Rosenberg, A., & Wilpon, J. (1981). An improved endpoint detector for isolated word recognition. *IEEE transactions on acoustics, speech, and signal processing*, *29*(4), 777-785.

Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Larson, S., Mahendran, A., Peper, J., Clarke, C., Lee, A., Hill, P., Kummerfeld, J., Leach, K., Laurenzano, M., Tang, L., & Mars, J. (2019). *An Evaluation Dataset for Intent Classification and Out-of-Scope Prediction*. https://doi.org/10.18653/v1/D19-1131

Larsson, S. (2002). *Issue-based dialogue management*. Citeseer.

Law, R., Buhalis, D., & Cobanoglu, C. (2014). Progress on information and communication technologies in hospitality and tourism. *International Journal of Contemporary Hospitality Management*.

Lee, J., Seo, S., & Choi, Y. S. (2019). Semantic Relation Classification via Bidirectional LSTM Networks with Entity-aware Attention using Latent Entity Typing. *Symmetry*, *11*, 785.

Leggetter, C. J., & Woodland, P. C. (1995). Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. *Computer Speech & Language*, *9*(2), 171-185.

Li, J. (2022). Recent advances in end-to-end automatic speech recognition. *APSIPA Transactions on Signal and Information Processing*, *11*(1).

Li, J., Zhao, R., Meng, Z., Liu, Y., Wei, W., Parthasarathy, S., Mazalov, V., Wang, Z., He, L., & Zhao, S. (2020). Developing RNN-T models surpassing high-performance hybrid models with customization capability. *arXiv preprint arXiv:2007.15188*.

Li, K., & Principe, J. C. (2018). Biologically-inspired spike-based automatic speech recognition of isolated digits over a reproducing kernel Hilbert space. *Frontiers in neuroscience*, *12*, 194.

Lin, Y.-T., Papangelis, A., Kim, S., Lee, S., Hazarika, D., Namazifar, M., Jin, D., Liu, Y., & Hakkani-Tur, D. (2023). Selective in-context data augmentation for intent detection using pointwise v-information. *arXiv preprint arXiv:2302.05096*.

Liu, T., Wang, K., Sha, L., Chang, B., & Sui, Z. (2018). Table-to-text generation by structure-aware seq2seq learning. Proceedings of the AAAI conference on artificial intelligence,

Liu, X., Eshghi, A., Swietojanski, P., & Rieser, V. (2019). Benchmarking natural language understanding services for building conversational agents. *arXiv preprint arXiv:1903.05566*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Löwe, W., Ludwig, A., & Schwind, A. (2001). *Understanding software – static and dynamic aspects* 17th International Conference on Advanced Science and Technology,

Lugosch, L., Ravanelli, M., Ignoto, P., Tomar, V. S., & Bengio, Y. (2019). Speech model pre-training for end-to-end spoken language understanding. *arXiv preprint arXiv:1904.03670*.

Maison, B., Neti, C., & Senior, A. (1999). Audio-visual speaker recognition for video broadcast news: some fusion techniques. 1999 IEEE Third Workshop on Multimedia Signal Processing (Cat. No. 99TH8451),

Malinowski, A. (2023). What are the benefits and drawbacks of early stopping in neural networks? https://www.linkedin.com/advice/1/what-benefits-drawbacks-early-stopping

Martin, R. C. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design. 2017. In (pp. 978-0134494166): ISBN.

McDonald, D. D. (2010). Natural language generation. *Handbook of natural language processing*, *2*, 121-144.

McGinty, L., & Reilly, J. (2010). On the evolution of critiquing recommenders. In *Recommender systems handbook* (pp. 419-453). Springer.

Mellor, S. J., Balcer, M., & Jacoboson, I. (2002). *Executable UML: A Foundation for Model-Driven Architectures*. Addison-Wesley Longman Publishing Co., Inc.

Microsoft. (2023). *Modeling SDK for Visual Studio - Domain-Specific Languages*. https://learn.microsoft.com/en-us/visualstudio/modeling/modeling-sdk-for-visual-studio-domain-specific-languages?view=vs-2022

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). *Distributed representations of words and phrases and their compositionality* Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2, Lake Tahoe, Nevada.

Miller, J., & Mukerji, J. (2003). *MDA Guide Version 1.0.1* (omg/2003-06-01).

Miller, J., & Mukerji, J. (2003). MDA Guide Version 1.0. 1. In: OMG Framingham.

Naimi, L., & Khriss, I. (2023). *Vers un modèle de tourisme intelligent: une approche basée sur le cadre 6A*.

Nardi, B. A. (1993). *A small matter of programming: perspectives on end user computing*. MIT press.

Newberg, L. A. (2009). Error statistics of hidden Markov model and hidden Boltzmann model results. *BMC bioinformatics*, *10*(1), 1-10.

NReco. (2023). Recommendation Engine. https://www.nrecosite.com/recommender_net.aspx

OMG. (2003a). *UML 2.0 OCL - 2nd Revised Submission*. http://www.omg.org/cgi-bin/doc?ad/2003-01-07

OMG. (2003b). *Unifed Modeling Lanugage (UML) Specification : Infrastructure, Version 2.0*.

OMG. (2005). *MOF QVT Final Adopted Specification*. http://www.omg.org/docs/ptc/05-11-01.pdf
OMG. (2008). *MOF 2.0 Query/View/transformation Specification. Version 1.0* (formal/2008-04-03). www.omg.org
OMG. (2014). Model Driven Architecture (MDA): MDA Guide rev. 2.0. *URL: http://www. omg. org/cgi-bin/doc*.
OMG. (2015). *Decision Model and Notation*. https://www.omg.org/spec/DMN/1.0/PDF
Ons, B., & Gemmeke, J. F. (2014). The self-taught vocal interface. *EURASIP Journal on Audio, Speech, and Music Processing, 2014*(1), 1-16.
Owen, S., Friedman, B. E., Anil, R., & Dunning, T. (2011). *Mahout in action*. Simon and Schuster.
Panayotov, V., Chen, G., Povey, D., & Khudanpur, S. (2015). Librispeech: an asr corpus based on public domain audio books. 2015 IEEE international conference on acoustics, speech and signal processing (ICASSP),
Park, D. H., Kim, H. K., Choi, I. Y., & Kim, J. K. (2012). A literature review and classification of recommender systems research. *Expert Systems with Applications*, *39*(11), 10059-10072.
Pouget, M. l. (2017). *Synthèse incrémentale de la parole à partir du texte* Université Grenoble Alpes].
Qian, Y., Ubale, R., Ramanaryanan, V., Lange, P., Suendermann-Oeft, D., Evanini, K., & Tsuprun, E. (2017). Exploring ASR-free end-to-end modeling to improve spoken language understanding in a cloud-based dialog system. 2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU),
Rabiner, L., & Juang, B. (1986). An introduction to hidden Markov models. *ieee assp magazine*, *3*(1), 4-16.
Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257-286.
Rabiner, L. R., & Juang, B.-H. (1993). Fundamentals of speech recognition. Prentice Hall signal processing series,
Rajraji, H. (2022). What is the importance of 6 A's in tourism? https://www.linkedin.com/pulse/what-importance-6-tourism-hicham-rajraji
Ramesh, K., Ravishankaran, S., Joshi, A., & Chandrasekaran, K. (2017). A survey of design techniques for conversational agents. International conference on information, communication and computing technology,
Renkens, V. (2018). Capsule networks for low resource spoken language understanding. *arXiv preprint arXiv:1805.02922*.
Renkens, V., Janssens, S., Ons, B., & Gemmeke, J. F. (2014). Acquisition of ordinal words using weakly supervised NMF. 2014 IEEE Spoken Language Technology Workshop (SLT),
Ricci, F., Rokach, L., & Shapira, B. (2015). Recommender systems: introduction and challenges. *Recommender systems handbook*, 1-34.
Rojas, G., Dominguez, F., & Salvatori, S. (2009). Recommender systems on the Web: A model-driven approach. E-Commerce and Web Technologies: 10th International Conference, EC-Web 2009, Linz, Austria, September 1-4, 2009. Proceedings 10,

Rojas, G., & Garrido, I. (2017). Toward a rapid development of social network-based recommender systems. *IEEE Latin America Transactions*, *15*(4), 753-759.
Rojas, G., & Uribe, C. (2013). A conceptual framework to develop mobile recommender systems of points of interest. 2013 32nd International Conference of the Chilean Computer Science Society (SCCC),
Rouse, M. (2012). Text to Speech. https://www.techopedia.com/definition/23843/text-to-speech-tts
Rožanc, I., & Mernik, M. (2021). The screening phase in systematic reviews: Can we speed up the process? *Advances in Computers*, *123*, 115-191.
Rubin, S. M., & Reddy, R. (1977). The locus model of search and its use in image interpretation. *Cambridge, Massachusetts*, 590-595.
Sainath, T. N., He, Y., Li, B., Narayanan, A., Pang, R., Bruguier, A., Chang, S.-y., Li, W., Alvarez, R., & Chen, Z. (2020). A streaming on-device end-to-end model surpassing server-side conventional model quality and latency. ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),
Saini, K. (2019). Natural Language Processing ft. Siri. https://medium.com/mytake/natural-language-processing-ft-siri-2bc7b854a2a3
Sakoe, H., & Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE transactions on acoustics, speech, and signal processing*, *26*(1), 43-49.
Sammet, J. E. (1972). Programming languages: History and future. *Communications of the ACM*, *15*(7), 601-610.
Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.
Saradhi, V., Reddy, S., & Reddy, C. A. (2022). Survey on Chatbot Classification and Technologies. *International Research Journal of Engineering and Technology (IRJET)*, *09*(11). https://www.academia.edu/90270037/Survey_on_Chatbot_Classification_and_Technologies
Satish, L., & Gururaj, B. (1993). Use of hidden Markov models for partial discharge pattern classification. *IEEE transactions on electrical insulation*, *28*(2), 172-182.
Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). Collaborative filtering recommender systems. *The adaptive web: methods and strategies of web personalization*, 291-324.
Schneider, S., Baevski, A., Collobert, R., & Auli, M. (2019). wav2vec: Unsupervised pre-training for speech recognition. *arXiv preprint arXiv:1904.05862*.
Serdyuk, D., Wang, Y., Fuegen, C., Kumar, A., Liu, B., & Bengio, Y. (2018). Towards end-to-end spoken language understanding. 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),
Sheikh, S. A., Tiwari, V., & Singhal, S. (2019). Generative model chatbot for human resource using deep learning. 2019 International Conference on Data Science and Engineering (ICDSE),
Shevat, A. (2017). *Designing bots: Creating conversational experiences*. " O'Reilly Media, Inc.".

Shillingford, B., Assael, Y., Hoffman, M. W., Paine, T., Hughes, C., Prabhu, U., Liao, H., Sak, H., Rao, K., & Bennett, L. (2018). Large-scale visual speech recognition. *arXiv preprint arXiv:1807.05162*.

Shreyaa, A. (2019). Speech recognition is hard — Part 1. https://towardsdatascience.com/speech-recognition-is-hard-part-1-258e813b6eb7

Sigala, M., Christou, E., & Gretzel, U. (2012). *Social media in travel, tourism and hospitality: Theory, practice and cases*. Ashgate Publishing, Ltd.

Sledziewski, K., Bordbar, B., & Anane, R. (2010). A DSL-based approach to software development and deployment on cloud. 2010 24th IEEE International Conference on Advanced Information Networking and Applications,

Son Chung, J., Senior, A., Vinyals, O., & Zisserman, A. (2017). Lip reading sentences in the wild. Proceedings of the IEEE conference on computer vision and pattern recognition,

Tang, Y. (2013). Deep Learning using Linear Support Vector Machines. arXiv:1306.0239. Retrieved June 01, 2013, from https://ui.adsabs.harvard.edu/abs/2013arXiv1306.0239T

Taunk, K., De, S., Verma, S., & Swetapadma, A. (2019). A brief review of nearest neighbor algorithm for learning and classification. 2019 International Conference on Intelligent Computing and Control Systems (ICCS),

Taylor, P. (2009). *Text-to-speech synthesis*. Cambridge university press.

Tessema, N., Ons, B., van de Loo, J., Gemmeke, J., De Pauw, G., & Daelemans, W. (2013). Metadata for corpora patcor and domotica-2. *KU Leuven, ESAT, Leuven, Belgium, Tech. Rep. KUL/ESAT/PSI/1303*.

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., & Leich, T. (2014). FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming*, *79*, 70-85.

Tian, J., Tu, Z., Wang, Z., Xu, X., & Liu, M. (2020). User intention recognition and requirement elicitation method for conversational ai services. 2020 IEEE International Conference on Web Services (ICWS),

Tran, D., Wang, H., Torresani, L., Ray, J., LeCun, Y., & Paluri, M. (2018). A closer look at spatiotemporal convolutions for action recognition. Proceedings of the IEEE conference on Computer Vision and Pattern Recognition,

Tran, H. M., Huertas, A., & Moreno, A. (2017). 6: A new framework for the analysis of smart tourism destinations. A comparative case study of two Spanish destinations.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is all you need* Proceedings of the 31st International Conference on Neural Information Processing Systems, Long Beach, California, USA.

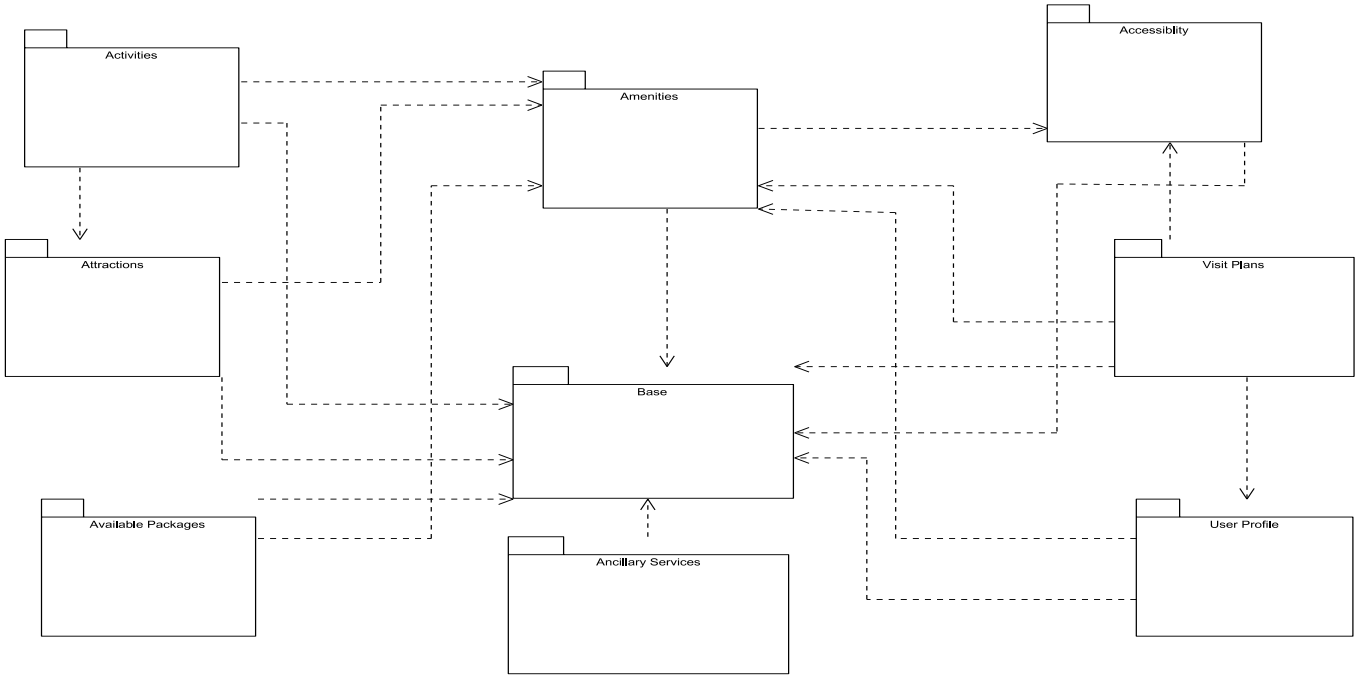
Vergin, R., O'Shaughnessy, D., & Farhat, A. (1999). Generalized mel frequency cepstral coefficients for large-vocabulary speaker-independent continuous-speech recognition. *IEEE Transactions on speech and audio processing*, *7*(5), 525-532.

Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, *13*(2), 260-269.

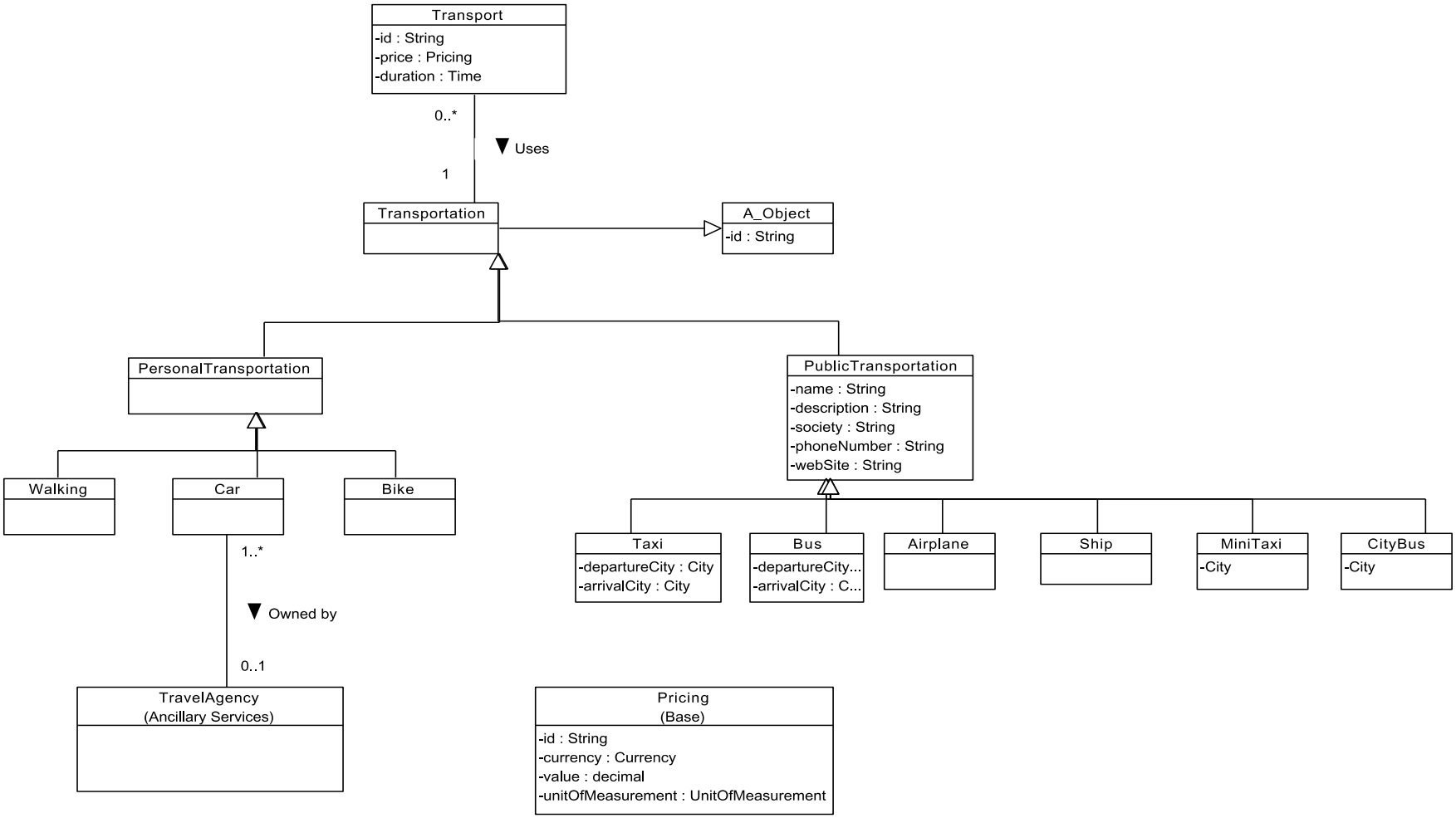
Völter, M., Stahl, T., Bettin, J., Haase, A., & Helsen, S. (2013). *Model-driven software development: technology, engineering, management*. John Wiley & Sons.
W3C. (2017). XSL Transformations (XSLT) Version 3.0. https://www.w3.org/TR/xslt-30/
Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. J. (2013). Phoneme recognition using time-delay neural networks. In *Backpropagation* (pp. 35-61). Psychology Press.
Wang, D., Park, S., & Fesenmaier, D. R. (2012). The role of smartphones in mediating the touristic experience. *Journal of Travel Research*, *51*(4), 371-387.
Wayesa, F. (2022). Design and Implementation of a Rule Based Afaan Oromoo Conversational Chatbots. *Asian Journal of Current Research*, 44-51.
Werthner, H., & Klein, S. (1999). *Information technology and tourism: a challenging ralationship*. Springer-Verlag Wien.
Werthner, H., & Ricci, F. (2004). E-commerce and tourism. *Communications of the ACM*, *47*(12), 101-105.
Wexelblat, R. L. (2014). *History of programming languages*. Academic Press.
Wimmer, M., & Kramler, G. (2006). Bridging Grammarware and Modelware. In *Satellite Events at the MoDELS 2005 Conference* (pp. 159-168). http://dx.doi.org/10.1007/11663430_17
Yan, Z., Duan, N., Chen, P., Zhou, M., Zhou, J., & Li, Z. (2017). Building task-oriented dialogue systems for online shopping. Proceedings of the AAAI Conference on Artificial Intelligence,
Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in Neural Information Processing Systems*, *32*.
Young, S. (1996). A review of large-vocabulary continuous-speech. *IEEE Signal Processing Magazine*, *13*(5), 45.
Zhang, C., Wang, S., Sun, S., & Wei, Y. (2020). Knowledge mapping of tourism demand forecasting research. *Tourism Management Perspectives*, *35*, 100715.
Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM computing surveys (CSUR)*, *52*(1), 1-38.
Zheng, Z., Lu, X.-Z., Chen, K.-Y., Zhou, Y.-C., & Lin, J.-R. (2022). Pretrained domain-specific language model for natural language processing tasks in the AEC domain. *Computers in Industry*, *142*, 103733.
Zia, M. M. (2017). XPath XSLT Tutorial. https://examples.javacodegeeks.com/java-development/core-java/xml/xpath/xpath-xslt-tutorial/
Zvornicanin, E. (2023). What Exactly Is an N-Gram? https://www.baeldung.com/cs/n-gram

ANNEXE I MODÈLE DE DOMAINE 6A

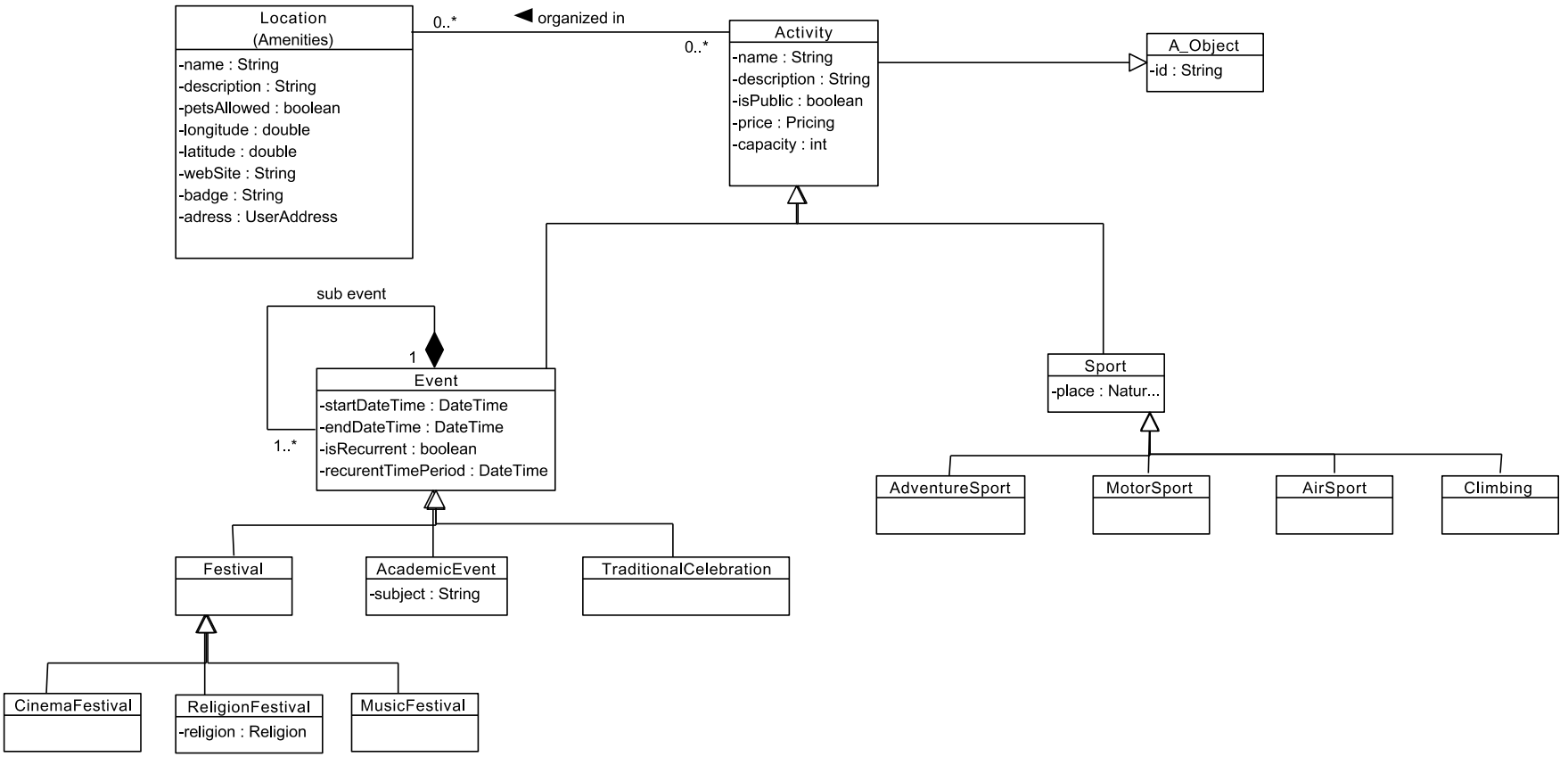
Les figures dans cette annexe sont basées sur la référence de (Naimi & Khriss, 2023).



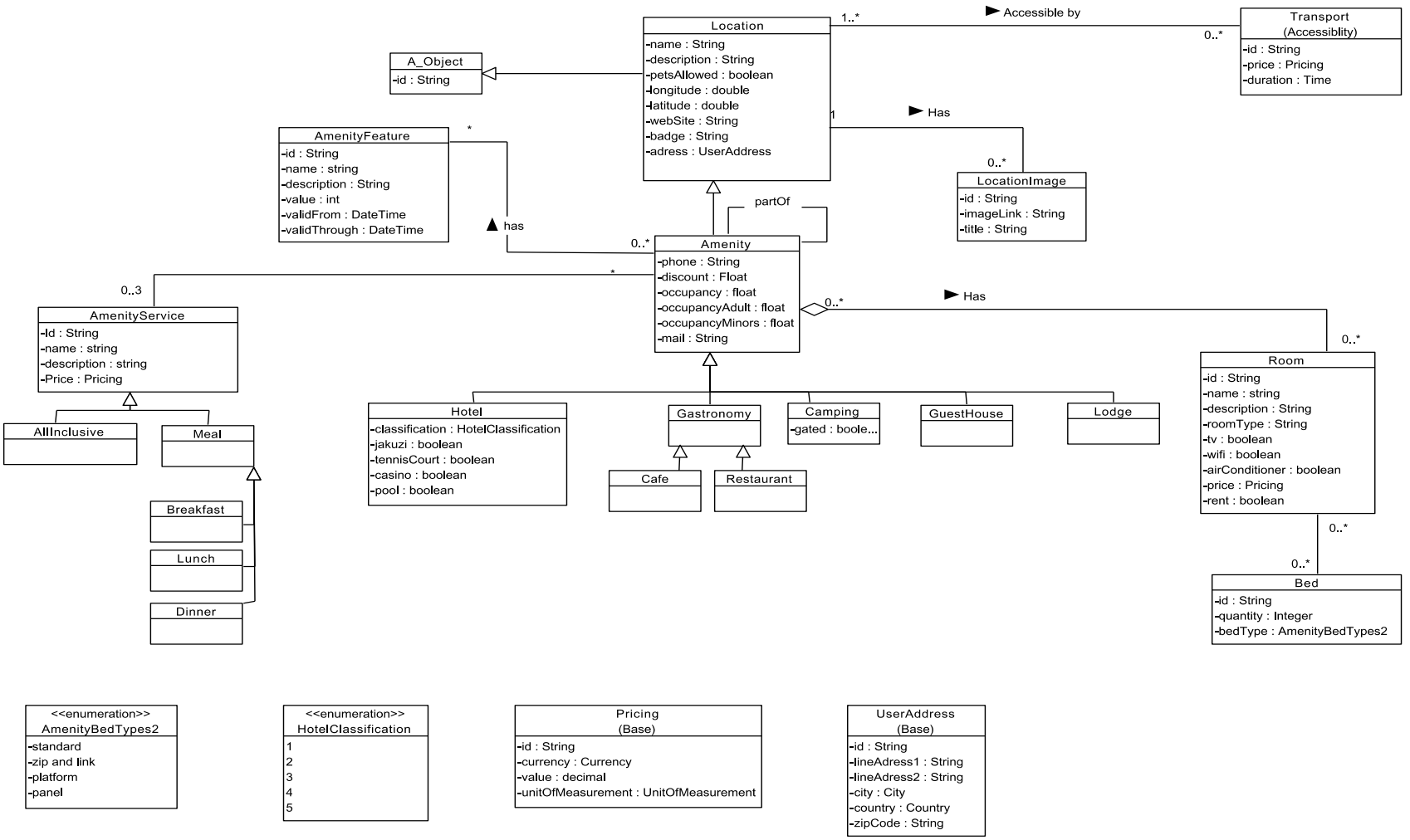
Annexe I.1 – Les packetages du modèle de domaine 6A.



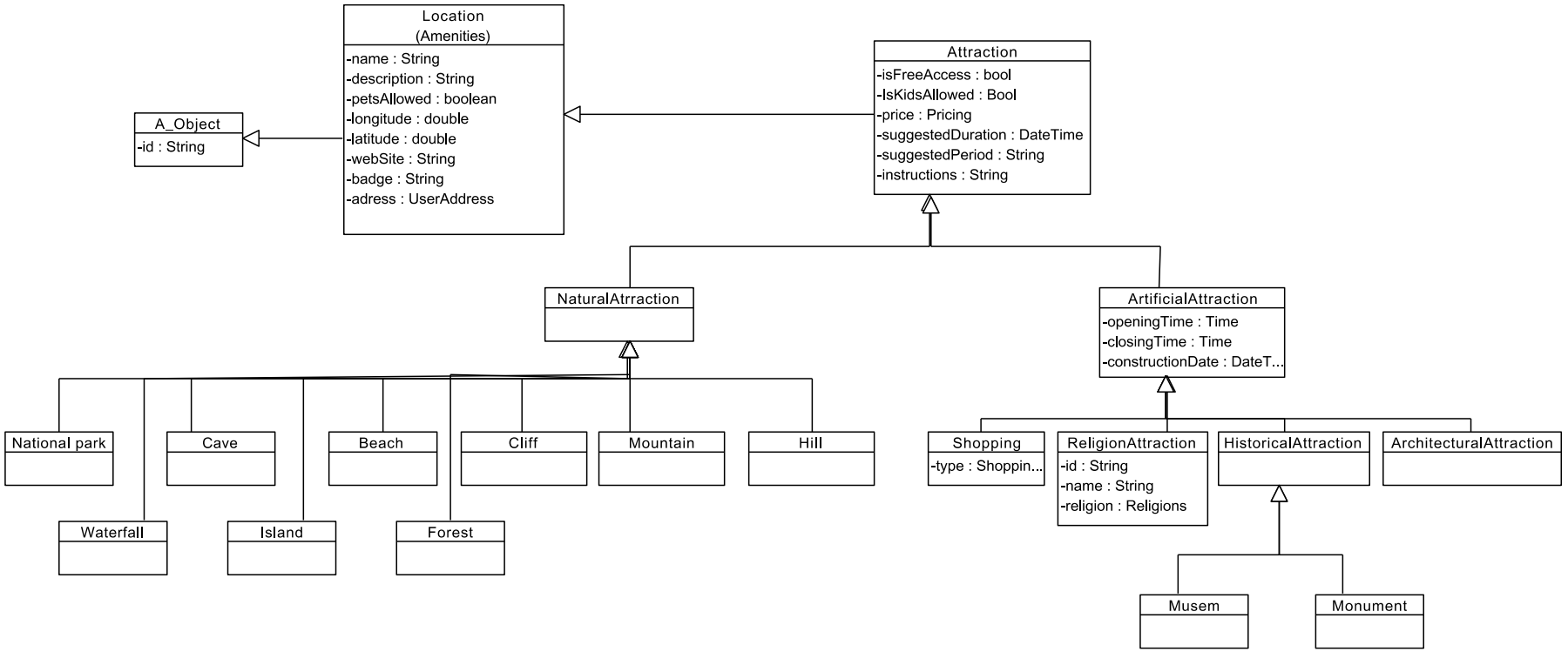
Annexe I.2 – Le packetage Accessibilities du modèle de domaine 6A.



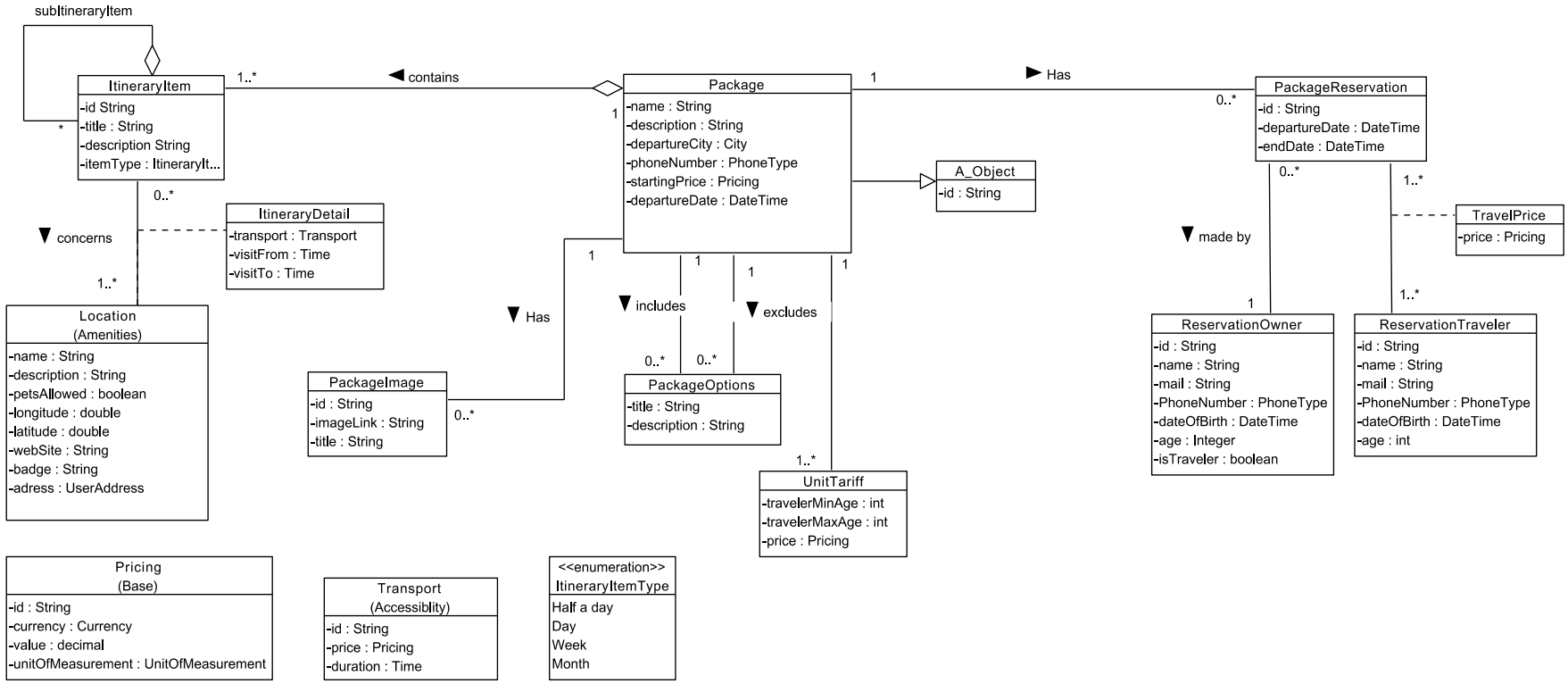
Annexe I.3 – Le packetage Activities du modèle de domaine 6A.



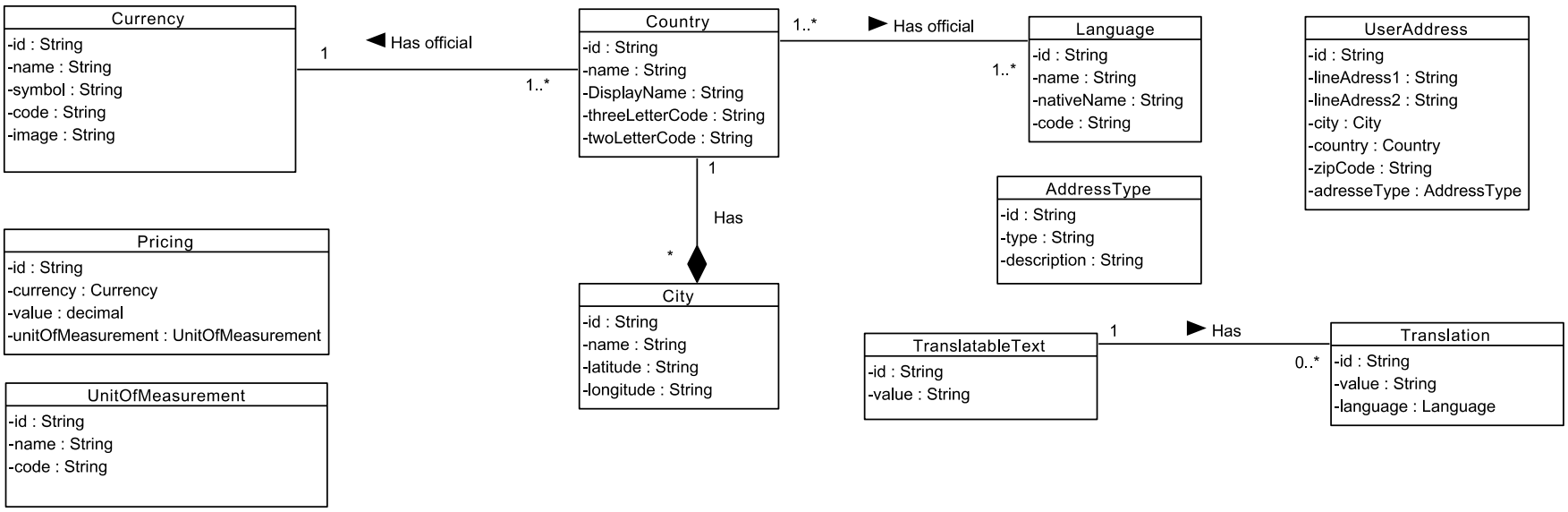
Annexe I.4 – Le packetage Amenities du modèle de domaine 6A.



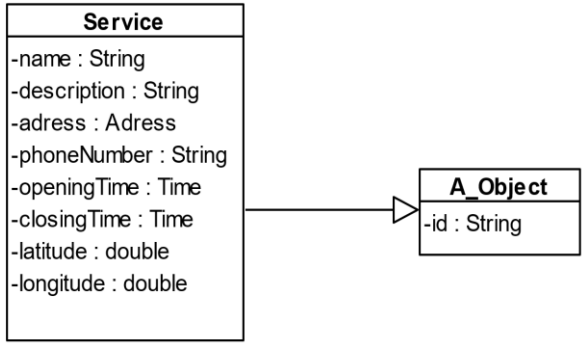
Annexe I.5 – Le packetage Attractions du modèle de domaine 6A.



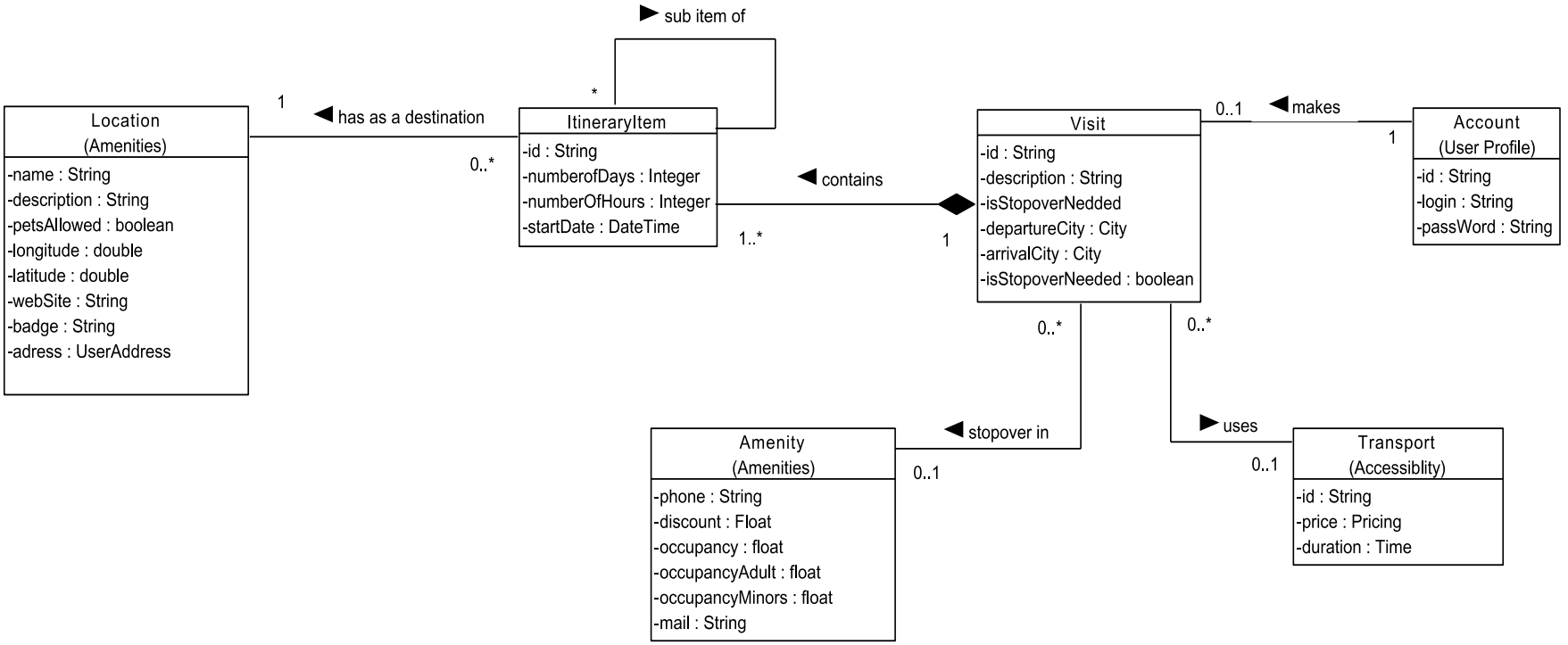
Annexe I.6 – Le packetage Available Packages du modèle de domaine 6A.



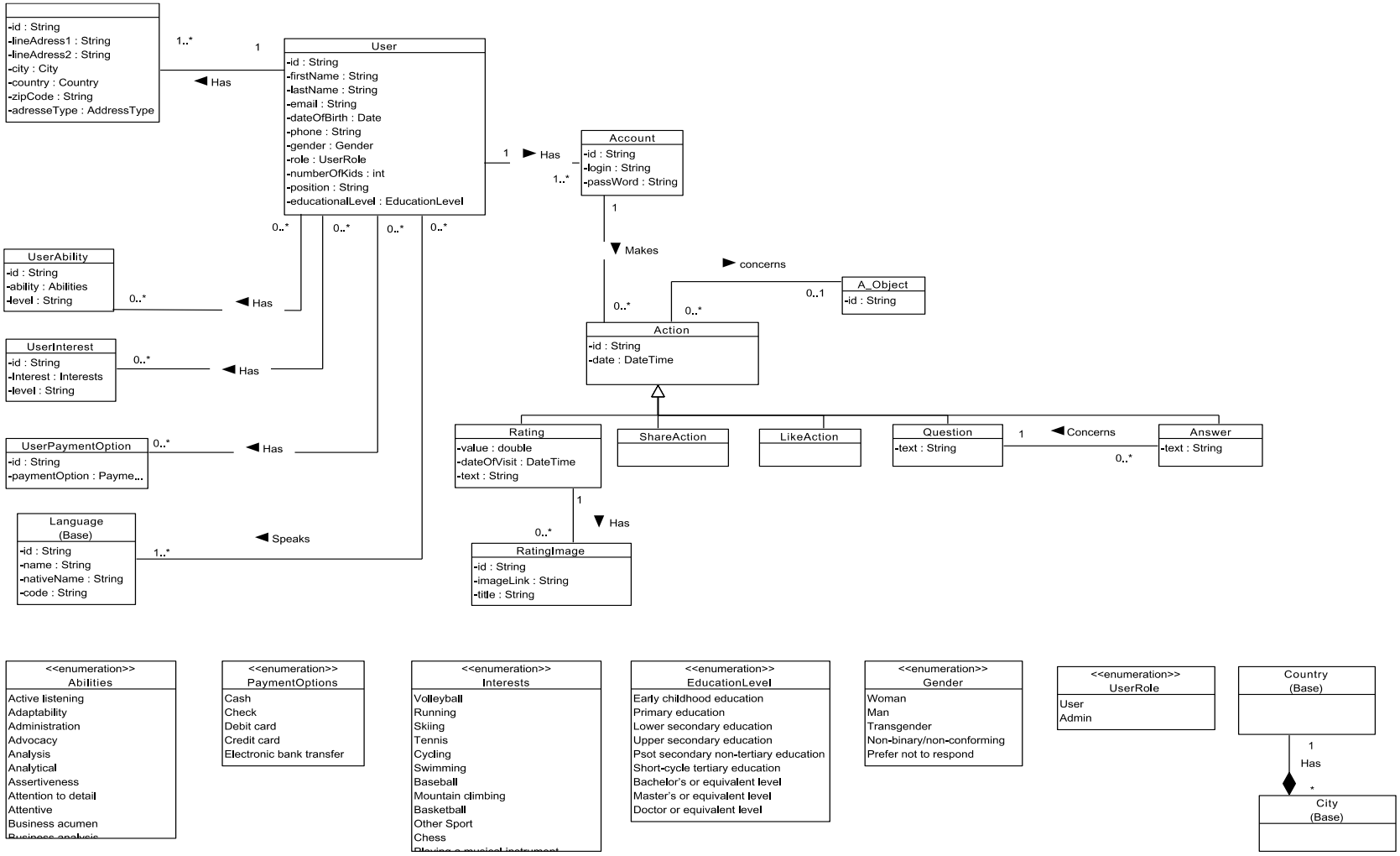
Annexe I.7 – Le packetage Base du modèle de domaine 6A.



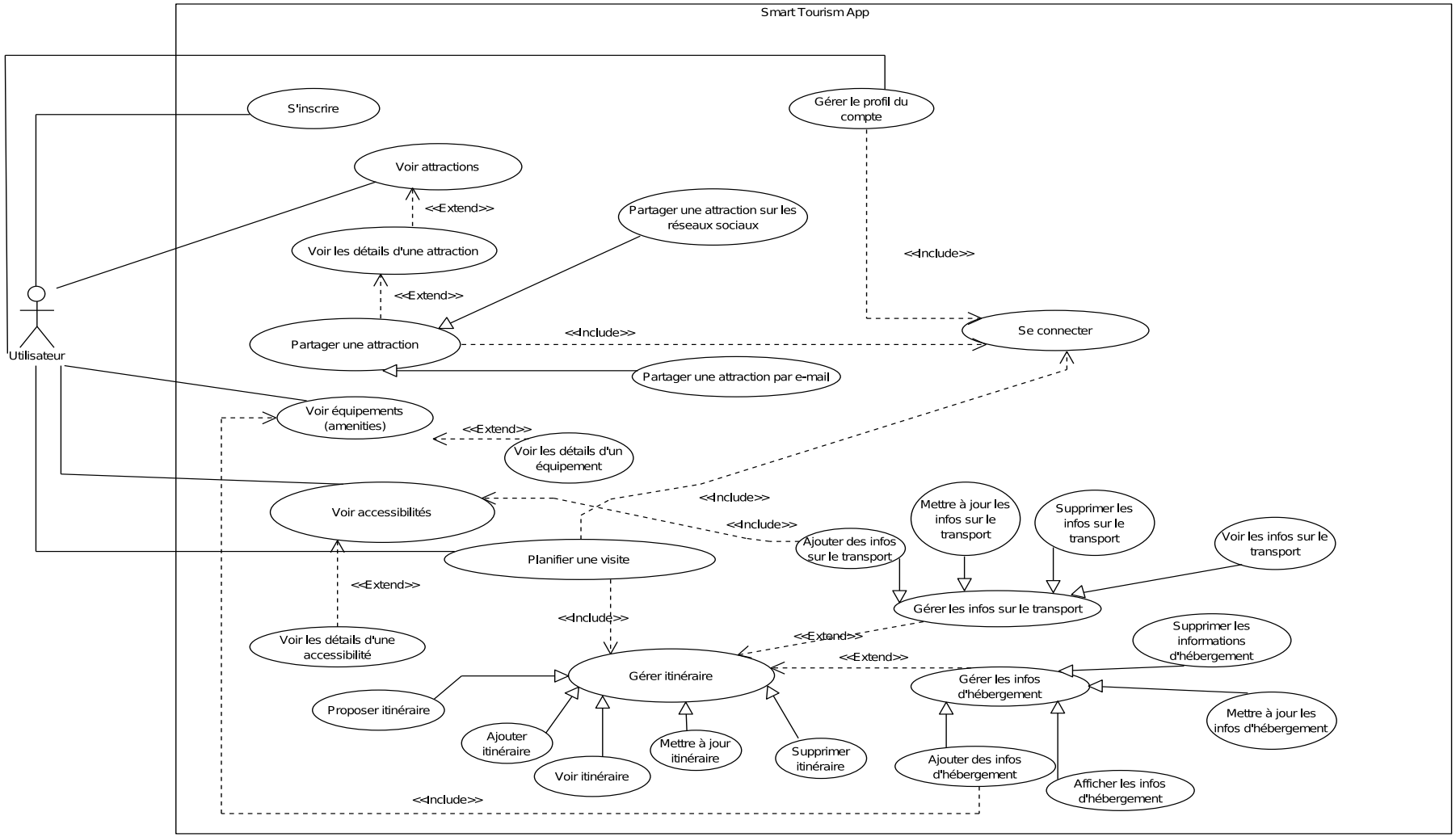
Annexe I.8 – Le packetage Ancillary Services du modèle de domaine 6A.



Annexe I.9 – Le packetage Visit Plans du modèle de domaine 6A.



Annexe I.10 – Le packetage User Profile du modèle de domaine 6A.



Annexe I.11 – Le diagramme de cas d'utilisation du modèle de domaine 6A.

ANNEXE II MODÈLES DE TRANSFORMATION T4

```
<#@ template hostspecific="true" inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingTextTransformation" #>
<#@ assembly name="System.Xml" #>
<#@ assembly name="System" #>
<#@ assembly name="System.Collections" #>
<#@ assembly name="System.Linq" #>
<#@ assembly name="System.IO" #>
<#@ assembly name="$(ProjectDir)\packages\Fare.2.2.1\lib\net35\Fare.dll" #>
<#@ output extension=".xml" #>
<#@ RecommenderSysDSL processor="RecommenderSysDSLDirectiveProcessor" requires="fileName='Sample.pl'" #>
<#@ import namespace="System.Xml" #>
<#@ import namespace="Fare" #>
<#@ import namespace="System" #>
<#@ import namespace="System.Collections" #>
<#@ import namespace="System.Linq" #>
<#@ import namespace="System.IO" #>
<#
    XmlDocument xmlDoc = new XmlDocument();
    xmlDoc.PreserveWhitespace = true;
    xmlDoc.Load("project.xml");
    XmlDocumentFragment xfrag = xmlDoc.CreateDocumentFragment();

    XmlNode node = null;
    string className = "";
    string recommenderType = "";
    ArrayList arlist = new ArrayList();

    string pattern = "^[a-zA-Z][a-z]\\do[a-zA-Z][a-zA-Z][A-Z][A-Z][a-zA-Z][a-z][A-Z][A-Z][A-Z][A-Z][a-z]0$";
    var xeger = new Xeger(pattern);
    string generatedString = "";


   foreach (Recommender recommender in this.RecommenderSysModel.Recommenders ) {

       generatedString = xeger.Generate();
       className = (recommender.GetType().Name).Replace("Recommender", "");
       arlist.Add(className);

       recommenderType = (recommender.RecommenderType).ToString();

       if(xmlDoc.SelectSingleNode("/Project/Models/Package/ModelChildren/Class[@Name='" + className + "']/ModelChildren") == null ){

           node = xmlDoc.SelectSingleNode("/Project/Models/Package/ModelChildren/Class[@Name='" + className + "']");
           xfrag.InnerXml = @"    <ModelChildren></ModelChildren>";
           node.AppendChild(xfrag);
       }

       node = xmlDoc.SelectSingleNode("/Project/Models/Package/ModelChildren/Class[@Name='" + className + "']/ModelChildren");
       xfrag.InnerXml = @"<Operation Abstract='false' BacklogActivityId='0' BodyCondition_IsNull='true' ConnectToCodeModel='1'
       Documentation_plain='' Id='"+ generatedString +"' Leaf='false' Lower_IsNull='true' Name='"+ recommender.Title +"' Ordered='fals
       PmAuthor='modou' PmCreateDateTime='2022-11-26T16:50:44.429' PmLastModified='2022-11-26T16:51:54.463'
       QualityReason_IsNull='true' QualityScore='-1' Query='false' ReturnTypeDocumentation_plain=''
       Scope='instance' Static='false' TypeModifier='' Unique='true' Upper_IsNull='true' UserIDLastNumericValue='0'
       UserID_IsNull='true' Visibility='public' Visible='true'></Operation>";
       node.AppendChild(xfrag);

       node = xmlDoc.SelectSingleNode("/Project/Models/Package/ModelChildren/Class[@Name='" + className + "']/ModelChildren/
       Operation[@Id='" + generatedString + "']");

       if(recommenderType.Equals("UserBased")){
           xfrag.InnerXml = @"<Stereotypes><Stereotype Idref='pSK7rSGGAqBAFBc9' Name='UserBased'/></Stereotypes>";
       }else{
           xfrag.InnerXml = @"<Stereotypes><Stereotype Idref='lMM3rSGGAqBAFBdx' Name='ItemBased'/></Stereotypes>";
```

```
63          }
64
65          node.AppendChild(xfrag);
66
67      }
68
69  #>
70  <#= xmlDoc.OuterXml #>
```

ANNEXE III MODÈLES DE TRANSFORMATION XSLT

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  xmlns:uFN="uFN" version="2.0">
  <xsl:import href="Util.xsl"/>
  <xsl:variable name="lt">
    <xsl:text>&lt;</xsl:text>
  </xsl:variable>
  <xsl:variable name="gt">
    <xsl:text>&gt;</xsl:text>
  </xsl:variable>
  <xsl:output method="text" encoding="iso-8859-1" indent="yes"/>
  <xsl:template name="entity">
  <xsl:param name = "contextName" />
  <xsl:result-document href="{$projectName}/Core/Interfaces/IRepositories//I{$contextName}RepositoryRecommender.cs">
using <xsl:value-of select="$projectName"/>.Core.Entities;
using <xsl:value-of select="$projectName"/>.SharedKernel.Interfaces;
using System.Collections.Generic;


namespace <xsl:value-of select="$projectName"/>.Core.Interfaces.IRepositories
{
    public partial interface I<xsl:value-of select="$contextName"/>Repository : IAsyncRepository
    <xsl:value-of disable-output-escaping="yes" select="$lt"/>
    <xsl:value-of select="$contextName"/><xsl:value-of disable-output-escaping="yes" select="$gt"/>
    , IRepository<xsl:value-of disable-output-escaping="yes" select="$lt"/>
    <xsl:value-of select="$contextName"/><xsl:value-of disable-output-escaping="yes" select="$gt"/>
    {

        public List<xsl:value-of disable-output-escaping="yes" select="$lt"/>
        <xsl:value-of select="$contextName"/><xsl:value-of disable-output-escaping="yes" select="$gt"/>
        Get<xsl:value-of select="$contextName"/>sByIds(int[] <xsl:value-of select="uFN:toLowerCase($contextName)"/>Ids);


    }
}
  </xsl:result-document>
 </xsl:template>

 <xsl:template match="/">
    <xsl:for-each select="/Project/Models/Package/ModelChildren/Class/ModelChildren/Operation[1]">
            <xsl:call-template name="entity">
                <xsl:with-param name="contextName" select = "../../@Name" />
            </xsl:call-template><xsl:text>&#xd;</xsl:text>
    </xsl:for-each>
 </xsl:template>

</xsl:stylesheet>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  xmlns:uFN="uFN" version="2.0">
  <xsl:import href="Util.xsl"/>
  <xsl:variable name="lt">
    <xsl:text>&lt;</xsl:text>
  </xsl:variable>
  <xsl:variable name="gt">
    <xsl:text>&gt;</xsl:text>
  </xsl:variable>
  <xsl:output method="text" encoding="iso-8859-1" indent="yes"/>
  <xsl:template name="entity">
  <xsl:param name = "contextName" />
  <xsl:result-document href="{$projectName}/Infrastructure/Interfaces/Repositories//{$contextName}RepositoryRecommender.cs">
using <xsl:value-of select="$projectName"/>.Core.Entities;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using <xsl:value-of select="$projectName"/>.Core.Interfaces.IRepositories;
```

```
namespace <xsl:value-of select="$projectName"/>.Infrastructure.Repositories
{
    public partial class <xsl:value-of select="$contextName"/>Repository : EfRepository
    <xsl:value-of disable-output-escaping="yes" select="$lt"/><xsl:value-of select="$contextName"/>
    <xsl:value-of disable-output-escaping="yes" select="$gt"/>, I<xsl:value-of select="$contextName"/>Repository
    {
```

```
        public List<xsl:value-of disable-output-escaping="yes" select="$lt"/><xsl:value-of select="$contextName"/>
        <xsl:value-of disable-output-escaping="yes" select="$gt"/>
        Get<xsl:value-of select="$contextName"/>sByIds(int[] <xsl:value-of select="uFN:toLowerCase($contextName)"/>Ids) {

            return
                (List<xsl:value-of disable-output-escaping="yes" select="$lt"/><xsl:value-of select="$contextName"/>
                <xsl:value-of disable-output-escaping="yes" select="$gt"/>)_dbSet
                    .Where(m => <xsl:value-of select="uFN:toLowerCase($contextName)"/>Ids.Contains(m.Id))
                    .Include(m => m.Actions)
                    .ToList();

        }


    }
}
  </xsl:result-document>
 </xsl:template>

  <xsl:template match="/">
    <xsl:for-each select="/Project/Models/Package/ModelChildren/Class/ModelChildren/Operation[1]">
            <xsl:call-template name="entity">
                <xsl:with-param name="contextName" select = "../../@Name" />
            </xsl:call-template><xsl:text>&#xd;</xsl:text>
    </xsl:for-each>
 </xsl:template>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  xmlns:uFN="uFN" version="2.0">
  <xsl:import href="Util.xsl"/>
  <xsl:variable name="lt">
    <xsl:text>&lt;</xsl:text>
  </xsl:variable>
  <xsl:variable name="gt">
    <xsl:text>&gt;</xsl:text>
  </xsl:variable>
  <xsl:output method="text" encoding="iso-8859-1" indent="yes"/>
  <xsl:template match="/Project/Models/Package/ModelChildren/Class/ModelChildren/Operation[1]">
  <xsl:variable name="contextName" select="../../@Name"/>
  <xsl:result-document href="{$projectName}/Core/Interfaces/IServices//I{$contextName}ServiceRecommender.cs">
using <xsl:value-of select="$projectName"/>.Core.Entities;
using System;
using System.Collections.Generic;

namespace <xsl:value-of select="$projectName"/>.Core.Interfaces.Services
{
    public partial interface I<xsl:value-of select="$contextName"/>Service
    {
<xsl:for-each select="/Project/Models/Package/ModelChildren/Class[@Name=$contextName]/ModelChildren/Operation">

        List<xsl:value-of disable-output-escaping="yes" select="$lt"/><xsl:value-of select="$contextName"/>
        <xsl:value-of disable-output-escaping="yes" select="$gt"/><xsl:value-of select="./@Name"/>
        (long[] preferred<xsl:value-of select="$contextName"/>Ids);

</xsl:for-each>
    }
}
  </xsl:result-document>
 </xsl:template>
</xsl:stylesheet>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  xmlns:uFN="uFN" version="2.0">
  <xsl:import href="Util.xsl"/>
  <xsl:variable name="lt">
    <xsl:text>&lt;</xsl:text>
  </xsl:variable>
  <xsl:variable name="gt">
    <xsl:text>&gt;</xsl:text>
  </xsl:variable>
  <xsl:output method="text" encoding="iso-8859-1" indent="yes"/>
  <xsl:template match="/Project/Models/Package/ModelChildren/Class/ModelChildren/Operation[1]">
  <xsl:variable name="contextName" select="../../@Name"/>
  <xsl:result-document href="{$projectName}/Core/Services//{$contextName}ServiceRecommender.cs">
using Microsoft.Extensions.Caching.Memory;
using <xsl:value-of select="$projectName"/>.Core.Entities;
using <xsl:value-of select="$projectName"/>.Core.Interfaces;
using <xsl:value-of select="$projectName"/>.Core.Interfaces.Services;
using NReco.CF.Taste.Impl.Common;
using NReco.CF.Taste.Impl.Model;
using NReco.CF.Taste.Impl.Neighborhood;
using NReco.CF.Taste.Impl.Recommender;
using NReco.CF.Taste.Impl.Similarity;
using NReco.CF.Taste.Model;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Threading.Tasks;
using <xsl:value-of select="$projectName"/>.Core.Interfaces.IRepositories;

namespace <xsl:value-of select="$projectName"/>.Core.Services
{
    public partial class <xsl:value-of select="$contextName"/>Service : I<xsl:value-of select="$contextName"/>Service
    {

        private readonly I<xsl:value-of select="$contextName"/>Repository _<xsl:value-of select="uFN:toLowerCase($contextName)"/>Repository;
        private readonly IRatingRepository _ratingRepository;
        private readonly IMemoryCache _memoryCache;
        private readonly IDataModel _dataModel;


        public <xsl:value-of select="$contextName"/>Service(I<xsl:value-of select="$contextName"/>Repository
        <xsl:value-of select="uFN:toLowerCase($contextName)"/>Repository, IRatingRepository ratingRepository, IMemoryCache memoryCache)
        {
            _<xsl:value-of select="uFN:toLowerCase($contextName)"/>Repository = <xsl:value-of select="uFN:toLowerCase($contextName)"/>Repository;
            _ratingRepository = ratingRepository;
            _memoryCache = memoryCache;
            _dataModel = GetDataModel();

        }

        <xsl:for-each select="/Project/Models/Package/ModelChildren/Class[@Name=$contextName]/ModelChildren/Operation">

        public List<xsl:value-of disable-output-escaping="yes" select="$lt"/><xsl:value-of select="$contextName"/>
        <xsl:value-of disable-output-escaping="yes" select="$gt"/><xsl:value-of select="./@Name"/>
        (long[] preferred<xsl:value-of select="$contextName"/>Ids)
        {
            List<xsl:value-of disable-output-escaping="yes" select="$lt"/><xsl:value-of select="$contextName"/>
            <xsl:value-of disable-output-escaping="yes" select="$gt"/>
            recommended<xsl:value-of select="$contextName"/>s = new List<xsl:value-of disable-output-escaping="yes" select="$lt"/>
            <xsl:value-of select="$contextName"/><xsl:value-of disable-output-escaping="yes" select="$gt"/>();

            var dataModel = GetDataModel();
```

```
            <xsl:for-each select="./Stereotypes/Stereotype">
                <xsl:if test="@Name = 'UserBased'">
            var plusAnonymModel = new PlusAnonymousUserDataModel(dataModel);
            var prefArr = new GenericUserPreferenceArray(preferred<xsl:value-of select="$contextName"/>Ids.Length);
            prefArr.SetUserID(0, PlusAnonymousUserDataModel.TEMP_USER_ID);
            for (int i = 0; i <xsl:value-of disable-output-escaping="yes" select="$lt"/>
            preferred<xsl:value-of select="$contextName"/>Ids.Length; i++)
            {
                prefArr.SetItemID(i, preferred<xsl:value-of select="$contextName"/>Ids[i]);

                // in this example we have no ratings of <xsl:value-of select="$contextName"/>s preferred by the user
                prefArr.SetValue(i, 5); // lets assume max rating
            }
            plusAnonymModel.SetTempPrefs(prefArr);

              </xsl:if>
            </xsl:for-each>

            <xsl:for-each select="./Stereotypes/Stereotype">
                <xsl:if test="@Name = 'UserBased'">
                <xsl:for-each select="./../../Stereotype">
                <xsl:if test="@Name='Cosine'">
            var similarity = new UncenteredCosineSimilarity(plusAnonymModel);
                </xsl:if>
                <xsl:if test="@Name = 'EuclideanDistance'">
            var similarity = new EuclideanDistanceSimilarity(plusAnonymModel);
                </xsl:if>
                <xsl:if test="@Name = 'LogLikelihood'">
            var similarity = new LogLikelihoodSimilarity(plusAnonymModel);
                </xsl:if>
                <xsl:if test="@Name = 'PearsonCorrelation'">
            var similarity = new PearsonCorrelationSimilarity(plusAnonymModel);
                </xsl:if>
                <xsl:if test="@Name = 'NearestN'">
            var neighborhood = new NearestNUserNeighborhood(2, similarity, plusAnonymModel);
                </xsl:if>
                <xsl:if test="@Name = 'Threshold'">
```

```
            var neighborhood = new ThresholdUserNeighborhood(2, similarity, plusAnonymModel);
                </xsl:if>
                </xsl:for-each>
              </xsl:if>
            </xsl:for-each>

            <xsl:for-each select="./Stereotypes/Stereotype">
                <xsl:if test="@Name = 'ItemBased'">
                <xsl:for-each select="./../../Stereotype">
            <xsl:if test="@Name='Cosine'">
            var similarity = new UncenteredCosineSimilarity(dataModel);
                </xsl:if>
```

```
            </xsl:if>
            <xsl:if test="@Name = 'EuclideanDistance'">
        var similarity = new EuclideanDistanceSimilarity(dataModel);
            </xsl:if>
            <xsl:if test="@Name = 'LogLikelihood'">
        var similarity = new LogLikelihoodSimilarity(dataModel);
            </xsl:if>
            <xsl:if test="@Name = 'PearsonCorrelation'">
        var similarity = new PearsonCorrelationSimilarity(dataModel);
            </xsl:if>
            </xsl:for-each>
          </xsl:if>
        </xsl:for-each>


        <xsl:for-each select="./Stereotypes/Stereotype">
            <xsl:if test="@Name = 'UserBased'">

        var recommender = new GenericUserBasedRecommender (plusAnonymModel, neighborhood, similarity);
        var recommendedItems = recommender.Recommend(PlusAnonymousUserDataModel.TEMP_USER_ID, 5, null);
            </xsl:if>
            <xsl:if test="@Name = 'ItemBased'">
        var recommender = new GenericItemBasedRecommender(dataModel, similarity);
        var recommendedItems = recommender.MostSimilarItems(preferred<xsl:value-of select="$contextName"/>Ids, 5);
            </xsl:if>
        </xsl:for-each>


        var <xsl:value-of select="$contextName"/>Ids = recommendedItems.Select(ri =><xsl:value-of disable-output-escaping="yes" select="$gt"/>
        (int)ri.GetItemID()).ToArray();

        recommended<xsl:value-of select="$contextName"/>s =  _<xsl:value-of select="uFN:toLowerCase($contextName)"/>
        Repository.Get<xsl:value-of select="$contextName"/>sByIds(<xsl:value-of select="$contextName"/>Ids);


        return recommended<xsl:value-of select="$contextName"/>s;
    }


</xsl:for-each>
```

```
private IDataModel GetDataModel()
{
    var cacheKey = "RecommenderDataModel";
    IDataModel dataModel = _memoryCache.Get<xsl:value-of disable-output-escaping="yes" select="$lt"/>
    IDataModel<xsl:value-of disable-output-escaping="yes" select="$gt"/>(cacheKey);
    if (dataModel != null)
    {
        return dataModel;
    }

    var <xsl:value-of select="uFN:toLowerCase($contextName)"/>Ratings = _ratingRepository.GetAll();

    FastByIDMap<xsl:value-of disable-output-escaping="yes" select="$lt"/>
    IList<xsl:value-of disable-output-escaping="yes" select="$lt"/>IPreference<xsl:value-of disable-output-escaping="yes" select="$gt"/>
    <xsl:value-of disable-output-escaping="yes" select="$gt"/> data = new FastByIDMap<xsl:value-of disable-output-escaping="yes" select="$lt"/>
    IList<xsl:value-of disable-output-escaping="yes" select="$lt"/>IPreference<xsl:value-of disable-output-escaping="yes" select="$gt"/>
    <xsl:value-of disable-output-escaping="yes" select="$gt"/>();
    foreach (var <xsl:value-of select="uFN:toLowerCase($contextName)"/>Rating in <xsl:value-of select="uFN:toLowerCase($contextName)"/>Ratings)
    {
        var userPreferences = data.Get(<xsl:value-of select="uFN:toLowerCase($contextName)"/>Rating.AccountId);
        if (userPreferences == null)
        {
            userPreferences = new List<xsl:value-of disable-output-escaping="yes" select="$lt"/>
            IPreference<xsl:value-of disable-output-escaping="yes" select="$gt"/>(5);
            data.Put(<xsl:value-of select="uFN:toLowerCase($contextName)"/>Rating.AccountId, userPreferences);
        }

        userPreferences.Add(new GenericPreference(<xsl:value-of select="uFN:toLowerCase($contextName)"/>
        Rating.AccountId, <xsl:value-of select="uFN:toLowerCase($contextName)"/>Rating.A_ObjectId, (float)
        <xsl:value-of select="uFN:toLowerCase($contextName)"/>Rating.Value));
    }

    var newData = new FastByIDMap<xsl:value-of disable-output-escaping="yes" select="$lt"/>IPreferenceArray
    <xsl:value-of disable-output-escaping="yes" select="$gt"/>(data.Count());
    foreach (var entry in data.EntrySet())
    {
        var prefList = (List<xsl:value-of disable-output-escaping="yes" select="$lt"/>IPreference
        <xsl:value-of disable-output-escaping="yes" select="$gt"/>)entry.Value;
        newData.Put(entry.Key, (IPreferenceArray)new GenericUserPreferenceArray(prefList));
    }
```

```
    dataModel = new GenericDataModel(newData);

    _memoryCache.Set(cacheKey, dataModel);

    return new GenericDataModel(newData);
}



sl:result-document>
.:template>
stylesheet>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  xmlns:uFN="uFN" version="2.0">
  <xsl:import href="Util.xsl"/>
  <xsl:variable name="lt">
    <xsl:text>&lt;</xsl:text>
  </xsl:variable>
  <xsl:variable name="gt">
    <xsl:text>&gt;</xsl:text>
  </xsl:variable>
  <xsl:output method="text" encoding="iso-8859-1" indent="yes"/>
  <xsl:template match="/Project/Models/Package/ModelChildren/Class/ModelChildren/Operation[1]">
  <xsl:variable name="contextName" select="../../@Name"/>
  <xsl:result-document href="{$projectName}/API/Controllers//{$contextName}ControllerRecommender.cs">
using <xsl:value-of select="$projectName"/>.Core.Entities;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using Microsoft.AspNetCore.Cors;
using <xsl:value-of select="$projectName"/>.Core.Interfaces.Services;

namespace <xsl:value-of select="$projectName"/>.API.Controllers
{
    public partial class <xsl:value-of select="$contextName"/>Controller : ControllerBase
    {

        <xsl:for-each select="/Project/Models/Package/ModelChildren/Class[@Name=$contextName]/ModelChildren/Operation">
        [HttpPost("<xsl:value-of select="./@Name"/>")]
        public List<xsl:value-of disable-output-escaping="yes" select="$lt"/><xsl:value-of select="$contextName"/>
        <xsl:value-of disable-output-escaping="yes" select="$gt"/> <xsl:value-of select="./@Name"/>
        (long[] <xsl:value-of select="uFN:toLowerCase($contextName)"/>sId)
        {
            return _<xsl:value-of select="$contextName"/>Service.<xsl:value-of select="./@Name"/>
            (<xsl:value-of select="uFN:toLowerCase($contextName)"/>sId);
        }
        </xsl:for-each>

    }

}
  </xsl:result-document>
 </xsl:template>
</xsl:stylesheet>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  xmlns:uFN="uFN" version="2.0" >
  <xsl:import href="Util.xsl"/>
  <xsl:variable name="lt">
    <xsl:text>&lt;</xsl:text>
  </xsl:variable>
  <xsl:variable name="gt">
    <xsl:text>&gt;</xsl:text>
  </xsl:variable>
  <xsl:output method="text" encoding="iso-8859-1" indent="yes" />
  <xsl:template match="/">
  <xsl:result-document href="{$projectName}/ConsoleTestApp/{$projectName}.ConsoleTestApp.csproj">
  <xsl:value-of disable-output-escaping="yes" select="$lt"/>Project Sdk="Microsoft.NET.Sdk"
  <xsl:value-of disable-output-escaping="yes" select="$gt"/>

  <xsl:value-of disable-output-escaping="yes" select="$lt"/>PropertyGroup<xsl:value-of disable-output-escaping="yes" select="$gt"/>
    <xsl:value-of disable-output-escaping="yes" select="$lt"/>OutputType<xsl:value-of disable-output-escaping="yes" select="$gt"/>
    Exe<xsl:value-of disable-output-escaping="yes" select="$lt"/>/OutputType<xsl:value-of disable-output-escaping="yes" select="$gt"/>
    <xsl:value-of disable-output-escaping="yes" select="$lt"/>TargetFramework<xsl:value-of disable-output-escaping="yes" select="$gt"/>
    netcoreapp3.1<xsl:value-of disable-output-escaping="yes" select="$lt"/>/TargetFramework<xsl:value-of disable-output-escaping="yes" select="$gt"/>
  <xsl:value-of disable-output-escaping="yes" select="$lt"/>/PropertyGroup<xsl:value-of disable-output-escaping="yes" select="$gt"/>


  <xsl:value-of disable-output-escaping="yes" select="$lt"/>ItemGroup<xsl:value-of disable-output-escaping="yes" select="$gt"/>
    <xsl:value-of disable-output-escaping="yes" select="$lt"/>ProjectReference Include="..\Infrastructure\<xsl:value-of select="$projectName"/>
    .Infrastructure.csproj" /<xsl:value-of disable-output-escaping="yes" select="$gt"/>
  <xsl:value-of disable-output-escaping="yes" select="$lt"/>/ItemGroup<xsl:value-of disable-output-escaping="yes" select="$gt"/>

<xsl:value-of disable-output-escaping="yes" select="$lt"/>/Project<xsl:value-of disable-output-escaping="yes" select="$gt"/>



 </xsl:result-document>
 </xsl:template>
</xsl:stylesheet>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  xmlns:uFN="uFN" version="2.0">
  <xsl:import href="Util.xsl"/>
  <xsl:variable name="lt">
    <xsl:text>&lt;</xsl:text>
  </xsl:variable>
  <xsl:variable name="gt">
    <xsl:text>&gt;</xsl:text>
  </xsl:variable>
  <xsl:output method="text" encoding="iso-8859-1" indent="yes"/>
  <xsl:template match="/">
  <xsl:result-document href="{$projectName}/ConsoleTestApp/Program.cs">using Microsoft.Extensions.Caching.Memory;
```

```
using System;
using System.Collections.Generic;
using <xsl:value-of select="$projectName"/>.Core.Entities;
using <xsl:value-of select="$projectName"/>.Core.Interfaces.IRepositories;
using <xsl:value-of select="$projectName"/>.Core.Interfaces.Services;
using <xsl:value-of select="$projectName"/>.Core.Services;
using <xsl:value-of select="$projectName"/>.Infrastructure;
using <xsl:value-of select="$projectName"/>.Infrastructure.Repositories;


namespace <xsl:value-of select="$projectName"/>.ConsoleTestApp
{
    public partial class Program
    {
        public static void Main(string[] args)
        {
            <xsl:for-each select="/Project/Models/Package/ModelChildren/Class/ModelChildren/Operation">

                <xsl:variable name="i" select="position()" />

            //Test<xsl:value-of select="$i"/>();

            </xsl:for-each>

        }


        <xsl:for-each select="/Project/Models/Package/ModelChildren/Class/ModelChildren/Operation">

        <xsl:variable name="contextName" select="../../@Name"/>

        <xsl:variable name="i" select="position()" />

        static void Test<xsl:value-of select="$i"/>()
        {
            using (<xsl:value-of select="$projectName"/>Context context = new <xsl:value-of select="$projectName"/>Context())
            {
                List<xsl:value-of disable-output-escaping="yes" select="$lt"/><xsl:value-of select="$contextName"/>
                <xsl:value-of disable-output-escaping="yes" select="$gt"/> preferred<xsl:value-of select="$contextName"/>s = null;
                List<xsl:value-of disable-output-escaping="yes" select="$lt"/><xsl:value-of select="$contextName"/>
                <xsl:value-of disable-output-escaping="yes" select="$gt"/> recommended<xsl:value-of select="$contextName"/>s = null;
                I<xsl:value-of select="$contextName"/>Repository <xsl:value-of select="uFN:toLowerCase($contextName)"/>
                Repository = new <xsl:value-of select="$contextName"/>Repository(context);
                IRatingRepository ratingRepository = new RatingRepository(context);
                IMemoryCache memoryCache  = new MemoryCache(new MemoryCacheOptions());
                I<xsl:value-of select="$contextName"/>Service <xsl:value-of select="uFN:toLowerCase($contextName)"/>
                Service = new <xsl:value-of select="$contextName"/>Service(<xsl:value-of select="uFN:toLowerCase($contextName)"/>
                Repository, ratingRepository, memoryCache);
                preferred<xsl:value-of select="$contextName"/>s = <xsl:value-of select="uFN:toLowerCase($contextName)"/>Repository.GetAll();
                if (preferred<xsl:value-of select="$contextName"/>s != null)
                {
                    recommended<xsl:value-of select="$contextName"/>s = <xsl:value-of select="uFN:toLowerCase($contextName)"/>
                    Service.<xsl:value-of select="./@Name"/>(new long[] { preferred<xsl:value-of select="$contextName"/>s[0].Id });
                    foreach (<xsl:value-of select="$contextName"/> item in recommended<xsl:value-of select="$contextName"/>s)
                    {
                        Console.Write("{0}\t", item.Id);
                    }
                    Console.WriteLine();
                }
                else
                {
                    Console.Write("Données inexistantes pour cette table !");
                }

            }
        }

        </xsl:for-each>


    }
}
  </xsl:result-document>
 </xsl:template>
</xsl:stylesheet>
```

ANNEXE IV CODE DE L'APPLICATION MOBILE

```
namespace TourismApp.SharedKernel
{
    7 références
    public abstract partial class BaseEntity
    {
        11 références
        public int Id { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
namespace TourismApp.SharedKernel.Interfaces
{
    8 références
    public partial interface IRepository<T> : IDisposable where T : BaseEntity
    {
        8 références
        List<T> GetAll();
    }

}
```

```
using TourismApp.SharedKernel;
using TourismApp.SharedKernel.Interfaces;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace TourismApp.Infrastructure.Repositories
{
    12 références
    public partial class EfRepository<T> : IAsyncRepository<T>, IRepository<T> where T : BaseEntity, IAggregateRoot
    {
        private bool disposed = false;

        protected readonly TourismAppContext _dbContext;
        protected readonly DbSet<T> _dbSet;

        4 références
        public EfRepository(TourismAppContext dbContext)
        {
            _dbContext = dbContext;
            _dbSet = _dbContext.Set<T>();
        }


        8 références
        public virtual List<T> GetAll()
        {
            return _dbSet.ToList();
        }

```

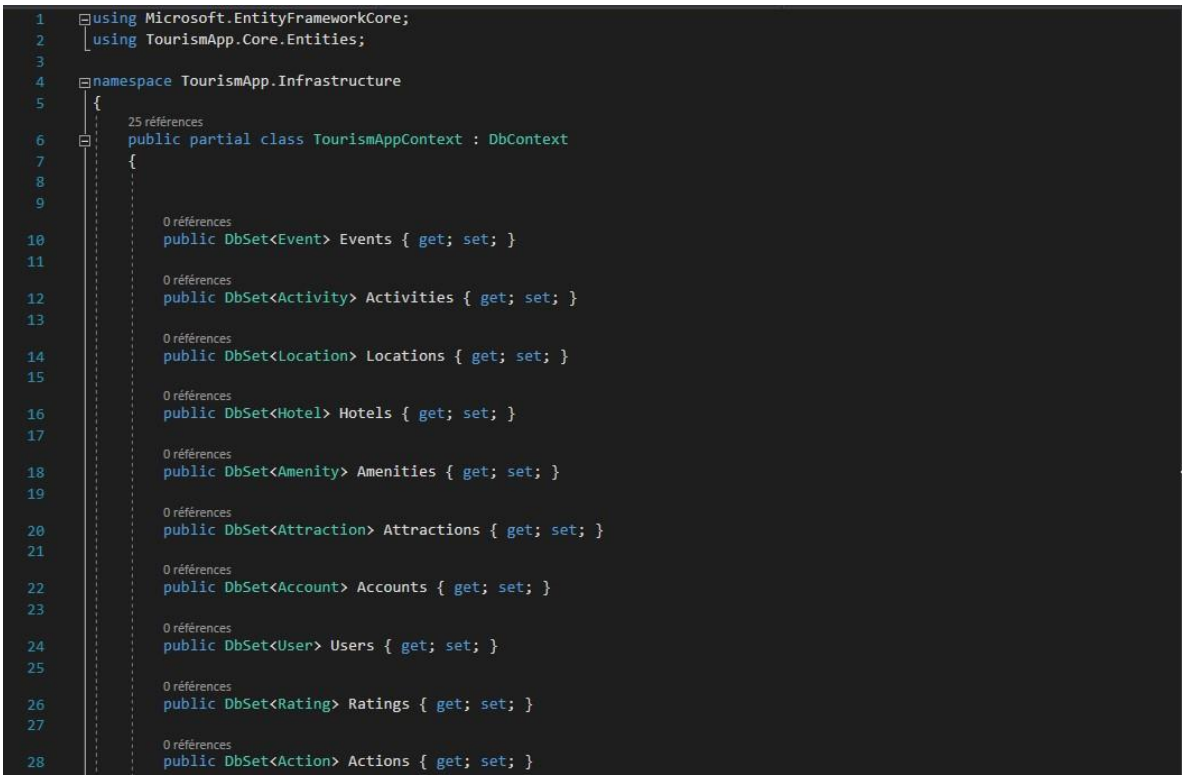
```
        protected virtual void Dispose(bool disposing)
        {
            if (disposed)
            {
                return;
            }

            if (disposing && _dbContext != null)
            {
                _dbContext.Dispose();
            }

            disposed = true;
        }



        0 références
        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }
    }
}
```

```
using System.Collections.Generic;
using System.Threading.Tasks;
namespace TourismApp.SharedKernel.Interfaces
{
    8 références
    public partial interface IAsyncRepository<T> where T : BaseEntity, IAggregateRoot
    {
        0 références
        Task<T> GetByIdAsync(int id);
        0 références
        T GetById(int id);
        0 références
        Task<IReadOnlyList<T>> ListAllAsync();
        0 références
        IReadOnlyList<T> ListAll();
        0 références
        Task<T> AddAsync(T entity);
        0 références
        Task UpdateAsync(T entity);
        0 références
        bool EntityExists(int id);
        0 références
        Task DeleteAsync(T entity);
    }
}
```

```
namespace TourismApp.SharedKernel.Interfaces
{
    6 références
    public partial interface IAggregateRoot
    {

    }
}
```

```
namespace TourismApp.Infrastructure
{
    2 références
    public static partial class Connexion
    {
        public static string local = "Server=(localdb)\\mssqllocaldb;Database=TourismAppDB;Trusted_Connection=True;";

        public static string connexionString = local;
    }
}
```

```
using Microsoft.EntityFrameworkCore.Design;
using Microsoft.EntityFrameworkCore;

namespace TourismApp.Infrastructure
{
    0 références
    public partial class TourismAppContextFactory : IDesignTimeDbContextFactory<TourismAppContext>
    {
        0 références
        public TourismAppContext CreateDbContext(string[] args)
        {
            var optionsBuilder = new DbContextOptionsBuilder<TourismAppContext>();
            optionsBuilder.UseSqlServer(@"" + Connexion.connexionString);
            return new TourismAppContext(optionsBuilder.Options);
        }
    }
}
```



```
using Microsoft.EntityFrameworkCore;
using TourismApp.Core.Entities;

namespace TourismApp.Infrastructure
{
    25 références
    public partial class TourismAppContext : DbContext
    {


        0 références
        public DbSet<Event> Events { get; set; }

        0 références
        public DbSet<Activity> Activities { get; set; }

        0 références
        public DbSet<Location> Locations { get; set; }

        0 références
        public DbSet<Hotel> Hotels { get; set; }

        0 références
        public DbSet<Amenity> Amenities { get; set; }

        0 références
        public DbSet<Attraction> Attractions { get; set; }

        0 références
        public DbSet<Account> Accounts { get; set; }

        0 références
        public DbSet<User> Users { get; set; }

        0 références
        public DbSet<Rating> Ratings { get; set; }

        0 références
        public DbSet<Action> Actions { get; set; }
```

```
        0 références
        public DbSet<A_Object> A_Objects { get; set; }



        1 référence
        public TourismAppContext(DbContextOptions<TourismAppContext> options)
            : base(options)
        {
        }

        4 références
        public TourismAppContext() : base(new DbContextOptionsBuilder<TourismAppContext>()
                    .UseSqlServer(@"" + Connexion.connexionString)
                    .Options)
        { }

        0 références
        protected override void OnModelCreating(ModelBuilder builder)
        {
            builder.Entity<A_Object>().HasMany(c => c.Actions).WithOne(e => e.A_Object);

            builder.Entity<Action>().HasKey(mr => new { mr.AccountId, mr.A_ObjectId });
            builder.Entity<Action>().HasOne(mr => mr.A_Object).WithMany(b => b.Actions).HasForeignKey(mr => mr.A_ObjectId).IsRequired();
        }



    }
}
```

```cs
using TourismApp.Core.Entities;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;


namespace TourismApp.Core.Interfaces.IRepositories
{
    12 références
    public partial interface IRatingRepository : IAsyncRepository<Rating>, IRepository<Rating>
    {

    }
}
```

```cs
using TourismApp.Core.Entities;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using TourismApp.Core.Interfaces.IRepositories;

namespace TourismApp.Infrastructure.Repositories
{
    6 références
    public partial class RatingRepository : EfRepository<Rating>, IRatingRepository
    {
        4 références
        public RatingRepository(TourismAppContext dbContext) : base(dbContext)
        {
        }
    }
}
```

```cs
using TourismApp.Core.Entities;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;

namespace TourismApp.Core.Interfaces.IRepositories
{
    7 références
    public partial interface IHotelRepository : IAsyncRepository<Hotel>, IRepository<Hotel>
    {
        2 références
        public List<Hotel> GetHotelsByIds(int[] hotelIds);
    }
}
```

```cs
using TourismApp.Core.Entities;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using TourismApp.Core.Interfaces.IRepositories;

namespace TourismApp.Infrastructure.Repositories
{
    4 références
    public partial class HotelRepository : EfRepository<Hotel>, IHotelRepository
    {
        1 référence
        public HotelRepository(TourismAppContext dbContext) : base(dbContext)
        {
        }
    }
}
```

```cs
using TourismApp.Core.Entities;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using TourismApp.Core.Interfaces.IRepositories;

namespace TourismApp.Infrastructure.Repositories
{
    4 références
    public partial class HotelRepository : EfRepository<Hotel>, IHotelRepository
    {
        2 références
        public List<Hotel> GetHotelsByIds(int[] hotelIds) {

            return
               (List<Hotel>)_dbSet
                   .Where(m => hotelIds.Contains(m.Id))
                   .Include(m => m.Actions)
                   .ToList();
        }
    }
}
```

```
using TourismApp.Core.Entities;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;


namespace TourismApp.Core.Interfaces.IRepositories
{
    7 références
    public partial interface IEventRepository : IAsyncRepository<Event>, IRepository<Event>
    {
        2 références
        public List<Event> GetEventsByIds(int[] eventIds);
    }
}
```

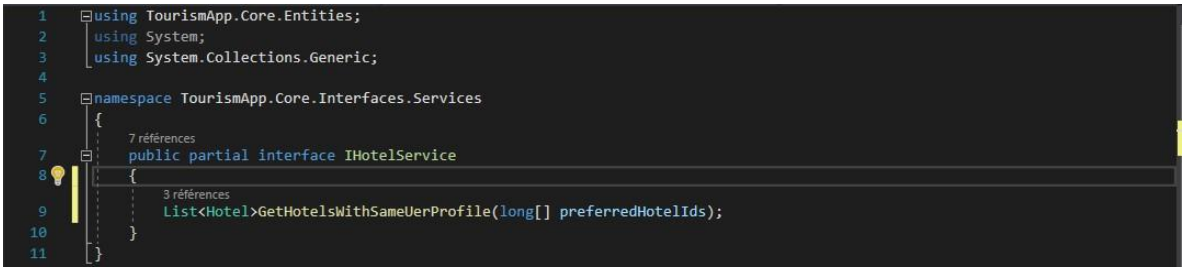
```
using TourismApp.Core.Entities;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using TourismApp.Core.Interfaces.IRepositories;

namespace TourismApp.Infrastructure.Repositories
{
    4 références
    public partial class EventRepository : EfRepository<Event>, IEventRepository
    {
        1 référence
        public EventRepository(TourismAppContext dbContext) : base(dbContext)
        {
        }
    }
}
```

```
using TourismApp.Core.Entities;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using TourismApp.Core.Interfaces.IRepositories;

namespace TourismApp.Infrastructure.Repositories
{
    4 références
    public partial class EventRepository : EfRepository<Event>, IEventRepository
    {
        2 références
        public List<Event> GetEventsByIds(int[] eventIds) {

            return
               (List<Event>)_dbSet
                    .Where(m => eventIds.Contains(m.Id))
                    .Include(m => m.Actions)
                    .ToList();

        }
    }
}
```

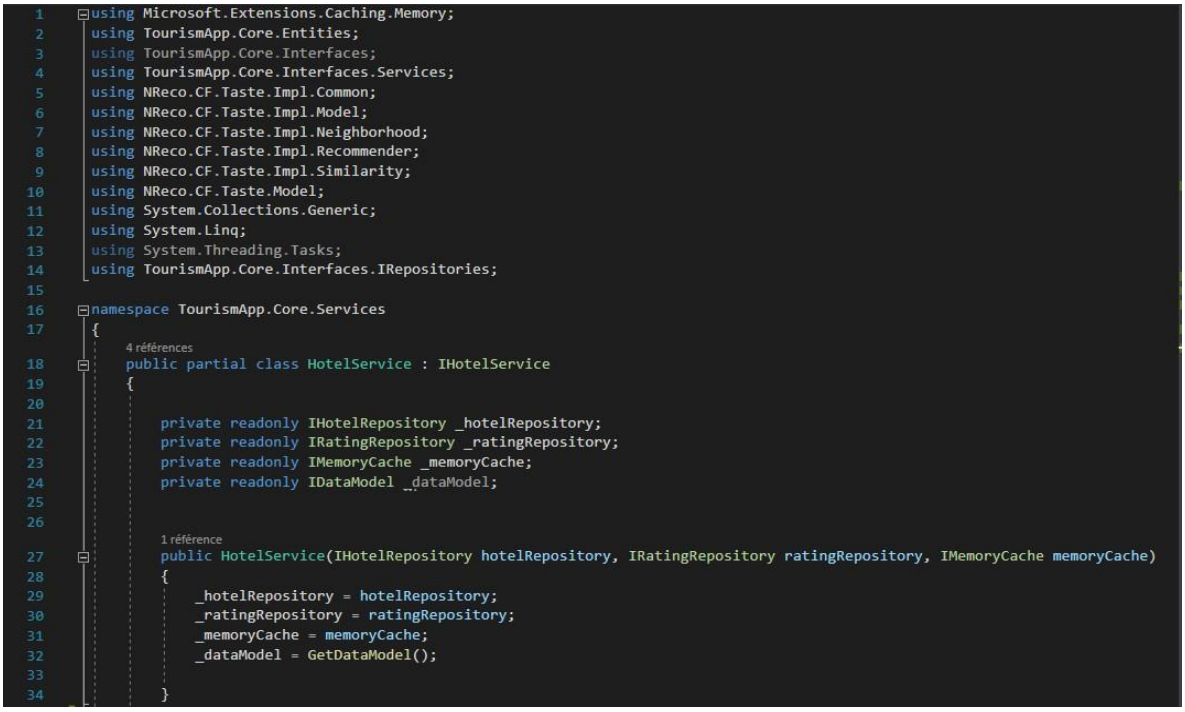
```
using TourismApp.Core.Entities;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;


namespace TourismApp.Core.Interfaces.IRepositories
{
    9 références
    public partial interface IAttractionRepository : IAsyncRepository<Attraction>, IRepository<Attraction>
    {
    }
}
```

```
using TourismApp.Core.Entities;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using TourismApp.Core.Interfaces.IRepositories;

namespace TourismApp.Infrastructure.Repositories
{
    6 références
    public partial class AttractionRepository : EfRepository<Attraction>, IAttractionRepository
    {
        2 références
        public AttractionRepository(TourismAppContext dbContext) : base(dbContext)
        {
        }
    }
}
```

```
using TourismApp.Core.Entities;
using System.Collections.Generic;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using TourismApp.Core.Interfaces.IRepositories;

namespace TourismApp.Infrastructure.Repositories
{
    6 références
    public partial class AttractionRepository : EfRepository<Attraction>, IAttractionRepository
    {
        3 références
        public List<Attraction> GetAttractionsByIds(int[] attractionIds) {

            return
               (List<Attraction>)_dbSet
                   .Where(m => attractionIds.Contains(m.Id))
                   .Include(m => m.Actions)
                   .ToList();
        }
    }
}
```



```
using TourismApp.Core.Entities;
using System;
using System.Collections.Generic;

namespace TourismApp.Core.Interfaces.Services
{
    7 références
    public partial interface IHotelService
    {
        3 références
        List<Hotel>GetHotelsWithSameUerProfile(long[] preferredHotelIds);
    }
}
```



```
using Microsoft.Extensions.Caching.Memory;
using TourismApp.Core.Entities;
using TourismApp.Core.Interfaces;
using TourismApp.Core.Interfaces.Services;
using NReco.CF.Taste.Impl.Common;
using NReco.CF.Taste.Impl.Model;
using NReco.CF.Taste.Impl.Neighborhood;
using NReco.CF.Taste.Impl.Recommender;
using NReco.CF.Taste.Impl.Similarity;
using NReco.CF.Taste.Model;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using TourismApp.Core.Interfaces.IRepositories;

namespace TourismApp.Core.Services
{
    4 références
    public partial class HotelService : IHotelService
    {

        private readonly IHotelRepository _hotelRepository;
        private readonly IRatingRepository _ratingRepository;
        private readonly IMemoryCache _memoryCache;
        private readonly IDataModel _dataModel;


        1 référence
        public HotelService(IHotelRepository hotelRepository, IRatingRepository ratingRepository, IMemoryCache memoryCache)
        {
            _hotelRepository = hotelRepository;
            _ratingRepository = ratingRepository;
            _memoryCache = memoryCache;
            _dataModel = GetDataModel();

        }
```

```
36          public List<Hotel>GetHotelsWithSameUerProfile(long[] preferredHotelIds)
37          {
38              List<Hotel> recommendedHotels = new List<Hotel>();
39
40              var dataModel = GetDataModel();
41
42
43              var plusAnonymModel = new PlusAnonymousUserDataModel(dataModel);
44              var prefArr = new GenericUserPreferenceArray(preferredHotelIds.Length);
45              prefArr.SetUserID(0, PlusAnonymousUserDataModel.TEMP_USER_ID);
46              for (int i = 0; i < preferredHotelIds.Length; i++)
47              {
48                  prefArr.SetItemID(i, preferredHotelIds[i]);
49
50                  // in this example we have no ratings of Hotels preferred by the user
51                  prefArr.SetValue(i, 5); // lets assume max rating
52              }
53              plusAnonymModel.SetTempPrefs(prefArr);
54
55              var similarity = new UncenteredCosineSimilarity(plusAnonymModel);
```

```
56
57              var neighborhood = new NearestNUserNeighborhood(2, similarity, plusAnonymModel);
58
59              var recommender = new GenericUserBasedRecommender (plusAnonymModel, neighborhood, similarity);
60              var recommendedItems = recommender.Recommend(PlusAnonymousUserDataModel.TEMP_USER_ID, 5, null);
61
62              var HotelIds = recommendedItems.Select(ri => (int)ri.GetItemID()).ToArray();
63
64              recommendedHotels =  _hotelRepository.GetHotelsByIds(HotelIds);
65
66              return recommendedHotels;
67          }
68
                2 références
69          private IDataModel GetDataModel()
70          {
71              var cacheKey = "RecommenderDataModel";
72              IDataModel dataModel = _memoryCache.Get<IDataModel>(cacheKey);
73              if (dataModel != null)
74              {
75                  return dataModel;
76              }
77
78              var hotelRatings = _ratingRepository.GetAll();
79
80              FastByIDMap<IList<IPreference>> data = new FastByIDMap<IList<IPreference>>();
81              foreach (var hotelRating in hotelRatings)
82              {
83                  var userPreferences = data.Get(hotelRating.AccountId);
84                  if (userPreferences == null)
85                  {
86                      userPreferences = new List<IPreference>(5);
87                      data.Put(hotelRating.AccountId, userPreferences);
88                  }
89
```

```
90                  userPreferences.Add(new GenericPreference(hotelRating.AccountId, hotelRating.A_ObjectId, (float)hotelRating.Value));
91              }
92
93              var newData = new FastByIDMap<IPreferenceArray>(data.Count());
94              foreach (var entry in data.EntrySet())
95              {
96                  var prefList = (List<IPreference>)entry.Value;
97                  newData.Put(entry.Key, (IPreferenceArray)new GenericUserPreferenceArray(prefList));
98              }
99
100             dataModel = new GenericDataModel(newData);
101
102             _memoryCache.Set(cacheKey, dataModel);
103
104             return new GenericDataModel(newData);
105         }
106
107     }
108 }
```

```
using TourismApp.Core.Entities;
using System;
using System.Collections.Generic;

namespace TourismApp.Core.Interfaces.Services
{
    7 références
    public partial interface IEventService
    {
        3 références
        List<Event>GetSimilarPrefferedEvents(long[] preferredEventIds);
    }
}
```

```
using Microsoft.Extensions.Caching.Memory;
using TourismApp.Core.Entities;
using TourismApp.Core.Interfaces;
using TourismApp.Core.Interfaces.Services;
using NReco.CF.Taste.Impl.Common;
using NReco.CF.Taste.Impl.Model;
using NReco.CF.Taste.Impl.Neighborhood;
using NReco.CF.Taste.Impl.Recommender;
using NReco.CF.Taste.Impl.Similarity;
using NReco.CF.Taste.Model;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using TourismApp.Core.Interfaces.IRepositories;

namespace TourismApp.Core.Services
{
    4 références
    public partial class EventService : IEventService
    {
        private readonly IEventRepository _eventRepository;
        private readonly IRatingRepository _ratingRepository;
        private readonly IMemoryCache _memoryCache;
        private readonly IDataModel _dataModel;

        1 référence
        public EventService(IEventRepository eventRepository, IRatingRepository ratingRepository, IMemoryCache memoryCache)
        {
            _eventRepository = eventRepository;
            _ratingRepository = ratingRepository;
            _memoryCache = memoryCache;
            _dataModel = GetDataModel();

        }
        3 références
        public List<Event>GetSimilarPrefferedEvents(long[] preferredEventIds)
        {
```

```
        public List<Event>GetSimilarPrefferedEvents(long[] preferredEventIds)
        {
            List<Event> recommendedEvents = new List<Event>();

            var dataModel = GetDataModel();
            var similarity = new PearsonCorrelationSimilarity(dataModel);

            var recommender = new GenericItemBasedRecommender(dataModel, similarity);
            var recommendedItems = recommender.MostSimilarItems(preferredEventIds, 5);
            var EventIds = recommendedItems.Select(ri => (int)ri.GetItemID()).ToArray();

            recommendedEvents =  _eventRepository.GetEventsByIds(EventIds);
            return recommendedEvents;
        }

        2 références
        private IDataModel GetDataModel()
        {
            var cacheKey = "RecommenderDataModel";
            IDataModel dataModel = _memoryCache.Get<IDataModel>(cacheKey);
            if (dataModel != null)
            {
                return dataModel;
            }

            var eventRatings = _ratingRepository.GetAll();

            FastByIDMap<IList<IPreference>> data = new FastByIDMap<IList<IPreference>>();
            foreach (var eventRating in eventRatings)
            {
                var userPreferences = data.Get(eventRating.AccountId);
                if (userPreferences == null)
                {
                    userPreferences = new List<IPreference>(5);
                    data.Put(eventRating.AccountId, userPreferences);
```

```
                    data.Put(eventRating.AccountId, userPreferences);
                }

                userPreferences.Add(new GenericPreference(eventRating.AccountId, eventRating.A_ObjectId, (float)eventRating.Value));
            }

            var newData = new FastByIDMap<IPreferenceArray>(data.Count());
            foreach (var entry in data.EntrySet())
            {
                var prefList = (List<IPreference>)entry.Value;
                newData.Put(entry.Key, (IPreferenceArray)new GenericUserPreferenceArray(prefList));
            }

            dataModel = new GenericDataModel(newData);

            _memoryCache.Set(cacheKey, dataModel);

            return new GenericDataModel(newData);
        }

    }
}
```

```
using TourismApp.Core.Entities;
using System;
using System.Collections.Generic;

namespace TourismApp.Core.Interfaces.Services
{
    9 références
    public partial interface IAttractionService
    {
        3 références
        List<Attraction>GetSimilarPrefferedAttractions(long[] preferredAttractionIds);
        3 références
        List<Attraction>GetAttractionsWithSameUserProfile(long[] preferredAttractionIds);
    }
}
```

```
using Microsoft.Extensions.Caching.Memory;
using TourismApp.Core.Entities;
using TourismApp.Core.Interfaces;
using TourismApp.Core.Interfaces.Services;
using NReco.CF.Taste.Impl.Common;
using NReco.CF.Taste.Impl.Model;
using NReco.CF.Taste.Impl.Neighborhood;
using NReco.CF.Taste.Impl.Recommender;
using NReco.CF.Taste.Impl.Similarity;
using NReco.CF.Taste.Model;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using TourismApp.Core.Interfaces.IRepositories;

namespace TourismApp.Core.Services
{
    6 références
    public partial class AttractionService : IAttractionService
    {

        private readonly IAttractionRepository _attractionRepository;
        private readonly IRatingRepository _ratingRepository;
        private readonly IMemoryCache _memoryCache;
        private readonly IDataModel _dataModel;


        2 références
        public AttractionService(IAttractionRepository attractionRepository, IRatingRepository ratingRepository, IMemoryCache memoryCache)
        {
            _attractionRepository = attractionRepository;
            _ratingRepository = ratingRepository;
            _memoryCache = memoryCache;
            _dataModel = GetDataModel();

        }
```

```
3 références
public List<Attraction>GetSimilarPrefferedAttractions(long[] preferredAttractionIds)
{
    List<Attraction> recommendedAttractions = new List<Attraction>();

    var dataModel = GetDataModel();
    var similarity = new UncenteredCosineSimilarity(dataModel);

    var recommender = new GenericItemBasedRecommender(dataModel, similarity);
    var recommendedItems = recommender.MostSimilarItems(preferredAttractionIds, 5);

    var AttractionIds = recommendedItems.Select(ri => (int)ri.GetItemID()).ToArray();

    recommendedAttractions =  _attractionRepository.GetAttractionsByIds(AttractionIds);

    return recommendedAttractions;
}

3 références
public List<Attraction>GetAttractionsWithSameUserProfile(long[] preferredAttractionIds)
{
    List<Attraction> recommendedAttractions = new List<Attraction>();

    var dataModel = GetDataModel();
    var plusAnonymModel = new PlusAnonymousUserDataModel(dataModel);
    var prefArr = new GenericUserPreferenceArray(preferredAttractionIds.Length);
    prefArr.SetUserID(0, PlusAnonymousUserDataModel.TEMP_USER_ID);
    for (int i = 0; i < preferredAttractionIds.Length; i++)
    {
        prefArr.SetItemID(i, preferredAttractionIds[i]);
        prefArr.SetValue(i, 5); // lets assume max rating
    }
    plusAnonymModel.SetTempPrefs(prefArr);
    var similarity = new EuclideanDistanceSimilarity(plusAnonymModel);
    var neighborhood = new ThresholdUserNeighborhood(2, similarity, plusAnonymModel);
```

```
    var recommender = new GenericUserBasedRecommender (plusAnonymModel, neighborhood, similarity);
    var recommendedItems = recommender.Recommend(PlusAnonymousUserDataModel.TEMP_USER_ID, 5, null);

    var AttractionIds = recommendedItems.Select(ri => (int)ri.GetItemID()).ToArray();

    recommendedAttractions =  _attractionRepository.GetAttractionsByIds(AttractionIds);

    return recommendedAttractions;
}


3 références
private IDataModel GetDataModel()
{
    var cacheKey = "RecommenderDataModel";
    IDataModel dataModel = _memoryCache.Get<IDataModel>(cacheKey);
    if (dataModel != null)
    {
        return dataModel;
    }

    var attractionRatings = _ratingRepository.GetAll();

    FastByIDMap<IList<IPreference>> data = new FastByIDMap<IList<IPreference>>();
    foreach (var attractionRating in attractionRatings)
    {
        var userPreferences = data.Get(attractionRating.AccountId);
        if (userPreferences == null)
        {
            userPreferences = new List<IPreference>(5);
            data.Put(attractionRating.AccountId, userPreferences);
        }

        userPreferences.Add(new GenericPreference(attractionRating.AccountId, attractionRating.A_ObjectId, (float)attractionRating.Value));
    }
```

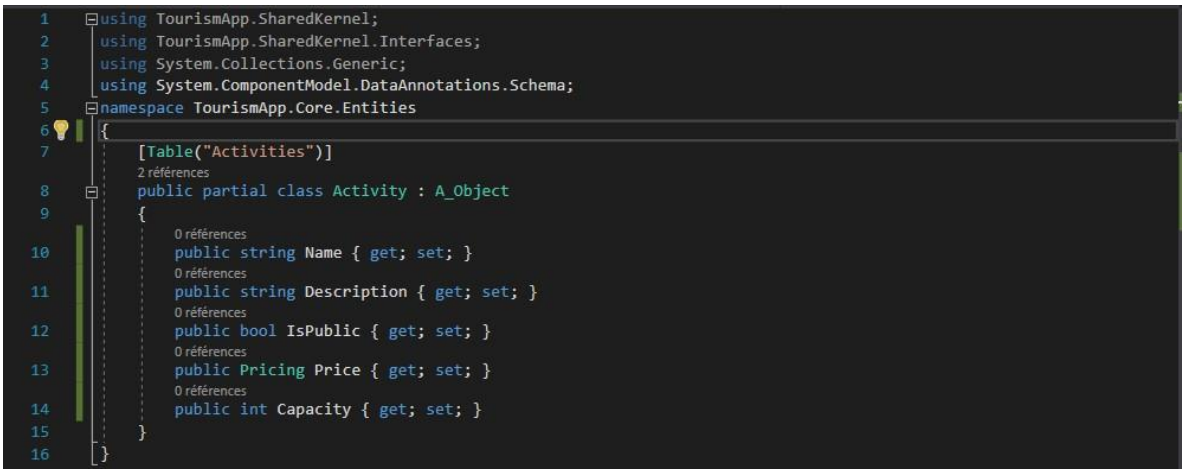
```
    var newData = new FastByIDMap<IPreferenceArray>(data.Count());
    foreach (var entry in data.EntrySet())
    {
        var prefList = (List<IPreference>)entry.Value;
        newData.Put(entry.Key, (IPreferenceArray)new GenericUserPreferenceArray(prefList));
    }

    dataModel = new GenericDataModel(newData);

    _memoryCache.Set(cacheKey, dataModel);

    return new GenericDataModel(newData);
}

}
}
```

```cs
using TourismApp.SharedKernel;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
namespace TourismApp.Core.Entities
{
    [Table("A_Objects")]
    5 références
    public partial class A_Object : BaseEntity , IAggregateRoot
    {
        5 références
        public List<Action> Actions { get; set; }
    }
}
```

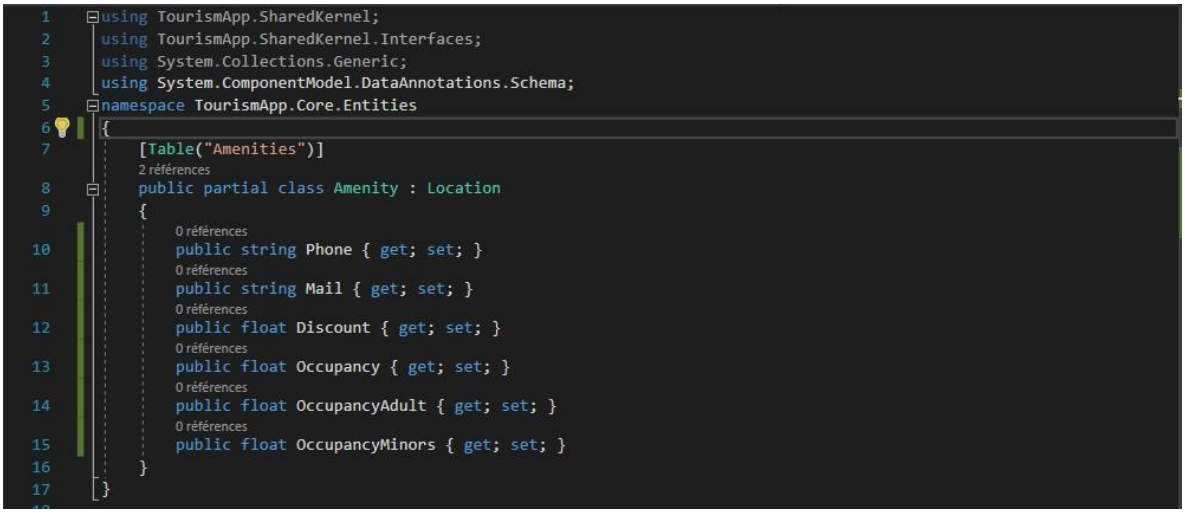
```cs
using TourismApp.SharedKernel;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
namespace TourismApp.Core.Entities
{
    [Table("Accounts")]
    3 références
    public partial class Account : BaseEntity , IAggregateRoot
    {
        0 références
        public int UserId { get; set; }
        0 références
        public User User { get; set; }
        0 références
        public List<Action> Actions { get; set; }
        0 références
        public string Login { get; set; }
        0 références
        public string PassWord { get; set; }
    }
}
```

```cs
using TourismApp.SharedKernel;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System;

namespace TourismApp.Core.Entities
{
    [Table("Actions")]
    6 références
    public partial class Action : BaseEntity , IAggregateRoot
    {
        10 références
        public int AccountId { get; set; }
        0 références
        public Account Account { get; set; }
        5 références
        public int A_ObjectId { get; set; }
        2 références
        public A_Object A_Object { get; set; }
        0 références
        public DateTime DateTime { get; set; }
    }
}
```

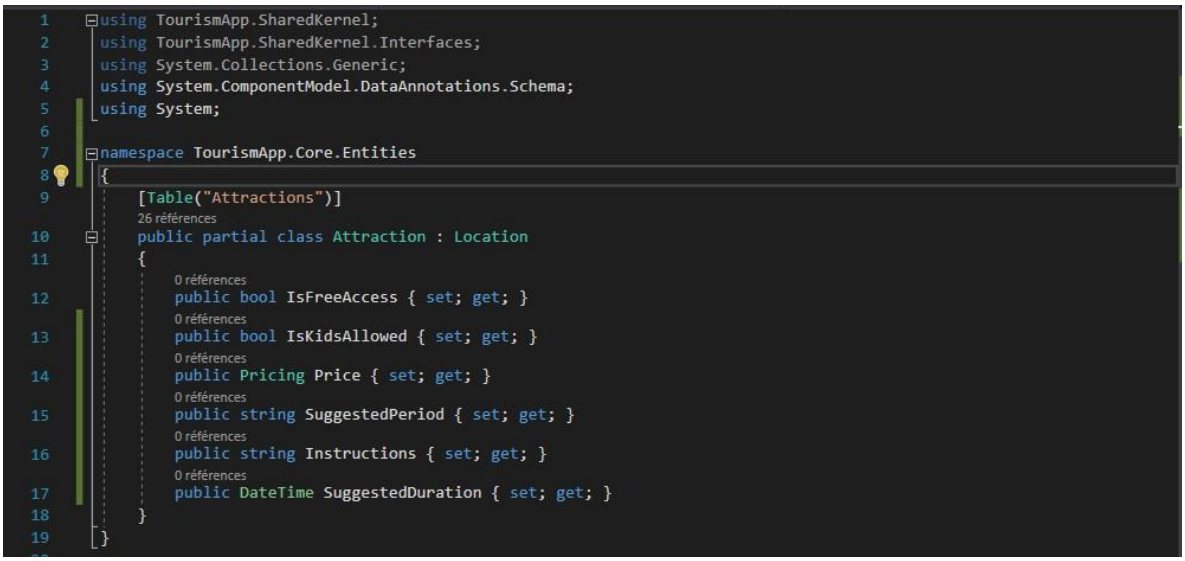
```cs
using System.ComponentModel.DataAnnotations.Schema;
using System;

namespace TourismApp.Core.Entities
{
    [Table("Ratings")]
    4 références
    public partial class Rating : Action
    {
        3 références
        public double Value { get; set; }
        0 références
        public string Text { get; set; }
        0 références
        public DateTime DateOfVisit { get; set; }
    }
}
```



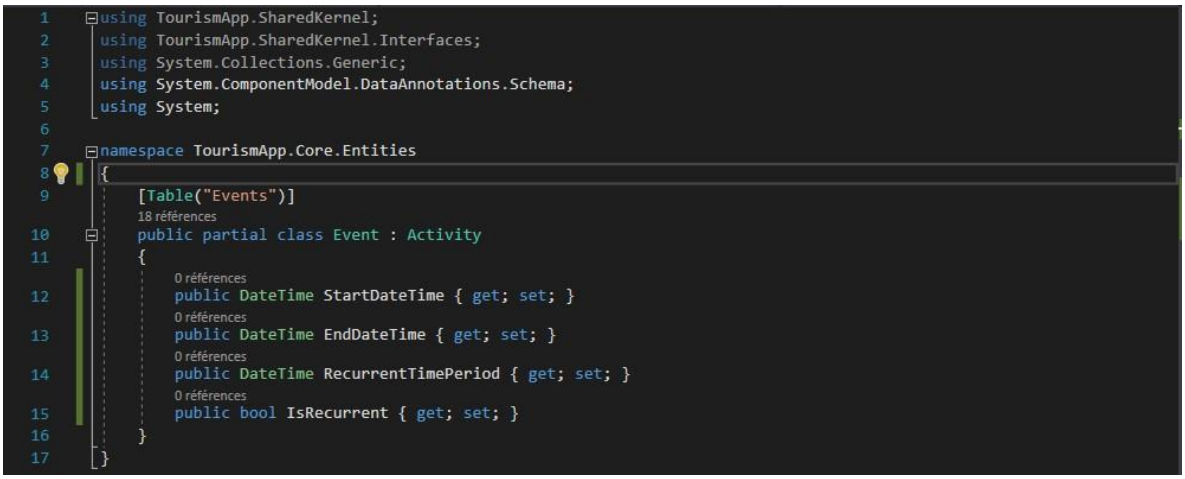
```
using TourismApp.SharedKernel;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
namespace TourismApp.Core.Entities
{
    [Table("Activities")]
    2 références
    public partial class Activity : A_Object
    {
        0 références
        public string Name { get; set; }
        0 références
        public string Description { get; set; }
        0 références
        public bool IsPublic { get; set; }
        0 références
        public Pricing Price { get; set; }
        0 références
        public int Capacity { get; set; }
    }
}
```



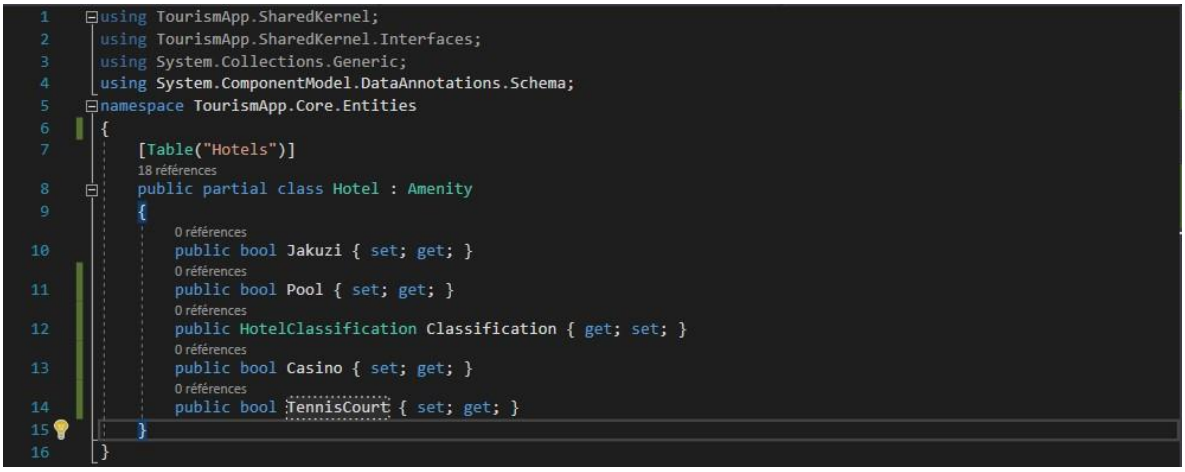
```
using TourismApp.SharedKernel;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
namespace TourismApp.Core.Entities
{
    [Table("Amenities")]
    2 références
    public partial class Amenity : Location
    {
        0 références
        public string Phone { get; set; }
        0 références
        public string Mail { get; set; }
        0 références
        public float Discount { get; set; }
        0 références
        public float Occupancy { get; set; }
        0 références
        public float OccupancyAdult { get; set; }
        0 références
        public float OccupancyMinors { get; set; }
    }
}
```



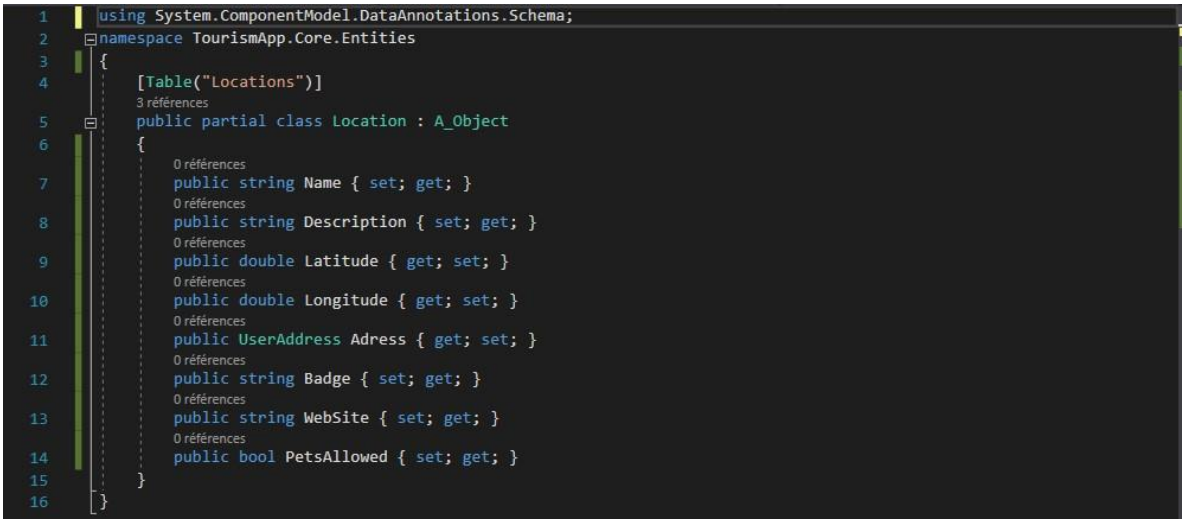
```
using TourismApp.SharedKernel;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System;

namespace TourismApp.Core.Entities
{
    [Table("Attractions")]
    26 références
    public partial class Attraction : Location
    {
        0 références
        public bool IsFreeAccess { set; get; }
        0 références
        public bool IsKidsAllowed { set; get; }
        0 références
        public Pricing Price { set; get; }
        0 références
        public string SuggestedPeriod { set; get; }
        0 références
        public string Instructions { set; get; }
        0 références
        public DateTime SuggestedDuration { set; get; }
    }
}
```



```
using TourismApp.SharedKernel;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System;

namespace TourismApp.Core.Entities
{
    [Table("Events")]
    18 références
    public partial class Event : Activity
    {
        0 références
        public DateTime StartDateTime { get; set; }
        0 références
        public DateTime EndDateTime { get; set; }
        0 références
        public DateTime RecurrentTimePeriod { get; set; }
        0 références
        public bool IsRecurrent { get; set; }
    }
}
```



```
using TourismApp.SharedKernel;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
namespace TourismApp.Core.Entities
{
    [Table("Hotels")]
    18 références
    public partial class Hotel : Amenity
    {
        0 références
        public bool Jakuzi { set; get; }
        0 références
        public bool Pool { set; get; }
        0 références
        public HotelClassification Classification { get; set; }
        0 références
        public bool Casino { set; get; }
        0 références
        public bool TennisCourt { set; get; }
    }
}
```



```
using System.ComponentModel.DataAnnotations.Schema;
namespace TourismApp.Core.Entities
{
    [Table("Locations")]
    3 références
    public partial class Location : A_Object
    {
        0 références
        public string Name { set; get; }
        0 références
        public string Description { set; get; }
        0 références
        public double Latitude { get; set; }
        0 références
        public double Longitude { get; set; }
        0 références
        public UserAddress Adress { get; set; }
        0 références
        public string Badge { set; get; }
        0 références
        public string WebSite { set; get; }
        0 références
        public bool PetsAllowed { set; get; }
    }
}
```

```
using TourismApp.SharedKernel;
using TourismApp.SharedKernel.Interfaces;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System;

namespace TourismApp.Core.Entities
{
    [Table("Users")]
    2 références
    public partial class User : BaseEntity , IAggregateRoot
    {
        0 références
        public List<Account> Accounts { get; set; }
        0 références
        public string FirstName { get; set; }
        0 références
        public string LastName { get; set; }
        0 références
        public string Email { get; set; }
        0 références
        public DateTime DateOfBirth { get; set; }
        0 références
        public string Phone { get; set; }
        0 références
        public string Position { get; set; }
        0 références
        public Gender Gender { get; set; }
        0 références
        public UserRole Role { get; set; }
        0 références
        public int NumberOfKids { get; set; }
        0 références
        public EducationalLevel EducationalLevel { get; set; }
    }
}
```

```
using Microsoft.Extensions.Caching.Memory;
using System;
using System.Collections.Generic;
using TourismApp.Core.Entities;
using TourismApp.Core.Interfaces.IRepositories;
using TourismApp.Core.Interfaces.Services;
using TourismApp.Core.Services;
using TourismApp.Infrastructure;
using TourismApp.Infrastructure.Repositories;


namespace TourismApp.ConsoleTestApp
{
    0 références
    public partial class Program
    {
        0 références
        public static void Main(string[] args)
        {
            Test1();

            Test2();

            Test3();

            Test4();
        }
```

```
static void Test1()
{
    using (TourismAppContext context = new TourismAppContext())
    {
        List<Attraction> preferredAttractions = null;
        List<Attraction> recommendedAttractions = null;
        IAttractionRepository attractionRepository = new AttractionRepository(context);
        IRatingRepository ratingRepository = new RatingRepository(context);
        IMemoryCache memoryCache  = new MemoryCache(new MemoryCacheOptions());
        IAttractionService attractionService = new AttractionService(attractionRepository, ratingRepository, memoryCache);
        preferredAttractions = attractionRepository.GetAll();
        if (preferredAttractions != null)
        {
            recommendedAttractions = attractionService.GetSimilarPrefferedAttractions(new long[] { preferredAttractions[0].Id });
            foreach (Attraction item in recommendedAttractions)
            {
                Console.Write("{0}\t", item.Id);
            }
            Console.WriteLine();
        }
        else
        {
            Console.Write("Données inexistantes pour cette table !");
        }

    }
}
```

```
static void Test2()
{
    using (TourismAppContext context = new TourismAppContext())
    {
        List<Attraction> preferredAttractions = null;
        List<Attraction> recommendedAttractions = null;
        IAttractionRepository attractionRepository = new AttractionRepository(context);
        IRatingRepository ratingRepository = new RatingRepository(context);
        IMemoryCache memoryCache  = new MemoryCache(new MemoryCacheOptions());
        IAttractionService attractionService = new AttractionService(attractionRepository, ratingRepository, memoryCache);
        preferredAttractions = attractionRepository.GetAll();
        if (preferredAttractions != null)
        {
            recommendedAttractions = attractionService.GetAttractionsWithSameUserProfile(new long[] { preferredAttractions[0].Id });
            foreach (Attraction item in recommendedAttractions)
            {
                Console.Write("{0}\t", item.Id);
            }
            Console.WriteLine();
        }
        else
        {
            Console.Write("Données inexistantes pour cette table !");
        }

    }
}
```

```
static void Test3()
{
    using (TourismAppContext context = new TourismAppContext())
    {
        List<Hotel> preferredHotels = null;
        List<Hotel> recommendedHotels = null;
        IHotelRepository hotelRepository = new HotelRepository(context);
        IRatingRepository ratingRepository = new RatingRepository(context);
        IMemoryCache memoryCache  = new MemoryCache(new MemoryCacheOptions());
        IHotelService hotelService = new HotelService(hotelRepository, ratingRepository, memoryCache);
        preferredHotels = hotelRepository.GetAll();
        if (preferredHotels != null)
        {
            recommendedHotels = hotelService.GetHotelsWithSameUerProfile(new long[] { preferredHotels[0].Id });
            foreach (Hotel item in recommendedHotels)
            {
                Console.Write("{0}\t", item.Id);
            }
            Console.WriteLine();
        }
        else
        {
            Console.Write("Données inexistantes pour cette table !");
        }

    }
}
```

```
static void Test4()
{
    using (TourismAppContext context = new TourismAppContext())
    {
        List<Event> preferredEvents = null;
        List<Event> recommendedEvents = null;
        IEventRepository eventRepository = new EventRepository(context);
        IRatingRepository ratingRepository = new RatingRepository(context);
        IMemoryCache memoryCache  = new MemoryCache(new MemoryCacheOptions());
        IEventService eventService = new EventService(eventRepository, ratingRepository, memoryCache);
        preferredEvents = eventRepository.GetAll();
        if (preferredEvents != null)
        {
            recommendedEvents = eventService.GetSimilarPrefferedEvents(new long[] { preferredEvents[0].Id });
            foreach (Event item in recommendedEvents)
            {
                Console.Write("{0}\t", item.Id);
            }
            Console.WriteLine();
        }
        else
        {
            Console.Write("Données inexistantes pour cette table !");
        }

    }
}
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Cors;
using TourismApp.Core.Interfaces.Services;

namespace TourismApp.API.Controllers
{
    [Route("api/[controller]")]
    [EnableCors("CorsPolicy")]
    [ApiController]

    2 références
    public partial class HotelController : ControllerBase
    {
        private readonly IHotelService _HotelService;

        0 références
        public HotelController(IHotelService HotelService)
        {
            _HotelService = HotelService;
        }
    }
}
```

```
using TourismApp.Core.Entities;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace TourismApp.API.Controllers
{
    2 références
    public partial class HotelController : ControllerBase
    {
        [HttpPost("GetHotelsWithSameUerProfile")]
        0 références
        public List<Hotel>GetHotelsWithSameUerProfile(long[] hotelsId)
        {
            return _HotelService.GetHotelsWithSameUerProfile(hotelsId);
        }
    }
}
```

```
using TourismApp.Core.Entities;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using Microsoft.AspNetCore.Cors;
using TourismApp.Core.Interfaces.Services;

namespace TourismApp.API.Controllers
{
    [Route("api/[controller]")]
    [EnableCors("CorsPolicy")]
    [ApiController]

    2 références
    public partial class EventController : ControllerBase
    {
        private readonly IEventService _EventService;

        0 références
        public EventController(IEventService EventService)
        {
            _EventService = EventService;
        }
    }
}
```

```
using TourismApp.Core.Entities;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace TourismApp.API.Controllers
{
    2 références
    public partial class EventController : ControllerBase
    {
        [HttpPost("GetSimilarPrefferedEvents")]
        0 références
        public List<Event>GetSimilarPrefferedEvents(long[] eventsId)
        {
            return _EventService.GetSimilarPrefferedEvents(eventsId);
        }
    }
}
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Cors;
using TourismApp.Core.Interfaces.Services;

namespace TourismApp.API.Controllers
{
    [Route("api/[controller]")]
    [EnableCors("CorsPolicy")]
    [ApiController]

    2 références
    public partial class AttractionController : ControllerBase
    {
        private readonly IAttractionService _AttractionService;

        0 références
        public AttractionController(IAttractionService AttractionService)
        {
            _AttractionService = AttractionService;
        }
    }
}
```

```
using TourismApp.Core.Entities;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;

namespace TourismApp.API.Controllers
{
    2 références
    public partial class AttractionController : ControllerBase
    {
        [HttpPost("GetSimilarPrefferedAttractions")]
        0 références
        public List<Attraction>GetSimilarPrefferedAttractions(long[] attractionsId)
        {
            return _AttractionService.GetSimilarPrefferedAttractions(attractionsId);
        }

        [HttpPost("GetAttractionsWithSameUserProfile")]
        0 références
        public List<Attraction>GetAttractionsWithSameUserProfile(long[] attractionsId)
        {
            return _AttractionService.GetAttractionsWithSameUserProfile(attractionsId);
        }
    }
}
```

```
using TourismApp.Core.Interfaces;
using TourismApp.Core.Services;
using TourismApp.Infrastructure;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.OpenApi.Models;
using TourismApp.Core.Interfaces.Services;
using TourismApp.Core.Interfaces.IRepositories;
using TourismApp.Infrastructure.Repositories;


namespace TourismApp.API
{
    2 références
    public partial class Startup
    {
        0 références
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        2 références
        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        0 références
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddCors(options => options.AddPolicy("CorsPolicy", builder =>
            {
                builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();
            }));
```

```
            services.AddControllers();
            services.AddControllersWithViews().AddNewtonsoftJson(options =>
                 options.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore);
            services.AddDbContext<TourismAppContext>(options =>
                 options.UseSqlServer(Configuration.GetConnectionString("DevConnection")));


            services.AddSwaggerGen(c => {
                c.SwaggerDoc("v1", new OpenApiInfo
                {
                    Title = "Customer.API",
                    Version = "v1"
                });
            });
```

```
50              // Add application services.
51              services.AddTransient<IRatingRepository, RatingRepository>();
52
53
54              services.AddTransient<IEventRepository, EventRepository>();
55              services.AddTransient<IEventService, EventService>();
56
57              services.AddTransient<IHotelRepository, HotelRepository>();
58              services.AddTransient<IHotelService, HotelService>();
59
60              services.AddTransient<IAttractionRepository, AttractionRepository>();
61              services.AddTransient<IAttractionService, AttractionService>();
62
63              services.AddTransient<IAttractionRepository, AttractionRepository>();
64              services.AddTransient<IAttractionService, AttractionService>();
65
66
67              services.AddMemoryCache();
68          }
69
70          // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
            0 références
71          public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
72          {
73              app.UseSwagger();
74              // Enable middleware to serve swagger-ui (HTML, JS, CSS, etc.),
75              // specifying the Swagger JSON endpoint.
76              app.UseSwaggerUI(c =>
77              {
78                  c.SwaggerEndpoint("/swagger/v1/swagger.json", "My API V1");
79              });
80
81              if (env.IsDevelopment())
82              {
83                  app.UseDeveloperExceptionPage();
84              }
```

```
85
86              app.UseHttpsRedirection();
87
88              app.UseAuthentication();
89
90              app.UseRouting();
91
92              app.UseCors("CorsPolicy");
93
94              app.UseAuthorization();
95
96              app.UseEndpoints(endpoints =>
97              {
98                  endpoints.MapControllers();
99              });
100         }
101     }
102 }
```

ANNEXE V CODE DU PROGRAMME DE LA CLASSIFICTION DE COMMANDES VOCALE

```
# Installer les dépendances nécessaires
!pip install torch
!pip install torchaudio
!pip install pandas
!pip install transformers
!pip install simpletransformers
!pip install openpyxl
!pip install huggingsound


drive.mount('/content/gdrive')

# Dossier contenant les fichiers audio et les fichiers .txt
audio_text_folder = "gdrive/My Drive/Data/audio_dataset"

# Liste des noms d'origine des classes
class_names = ["hotel", "event", "userbased", "itembased", "hotel_userbased", "event_itembased"]

# Créer le dictionnaire des classes avec les noms d'origine
class_dictionary = {class_name: class_name for class_name in class_names}


# Charger le modèle préentraîné pour la transcription audio en texte
from huggingsound import SpeechRecognitionModel
from transformers import Wav2Vec2ForCTC, Wav2Vec2Tokenizer
model = SpeechRecognitionModel("jonatasgrosman/wav2vec2-large-xlsr-53-english")



# Créer le DataFrame avec les données renommées et les transcriptions
data = []
```

```
for class_name in class_names:
    class_folder = os.path.join(audio_text_folder, class_name)
    audio_files = [file for file in os.listdir(class_folder) if file.endswith(".wav")]

    for audio_file in audio_files:
        txt_file = audio_file.replace(".wav", ".txt")
        audio_path = os.path.join(class_folder, audio_file)
        txt_path = os.path.join(class_folder, txt_file)

        # Transcription de l'audio
        transcriptions = model.transcribe([audio_path])
        transcription = transcriptions[0]

        # Lecture du fichier .txt correspondant pour obtenir la phrase
        with open(txt_path, 'r') as f:
            txt_phrase = f.read()

        data.append({'audio': audio_file, 'phrase': txt_phrase, 'transcription':
transcription['transcription'].upper(), 'class': class_dictionary[class_name]})

# Convertir le DataFrame en fichier Excel
df = pd.DataFrame(data)
df.to_excel(f'gdrive/My Drive/Data/audio_text_dataset.xlsx', index=False)

import numpy as np
import pandas as pd

def word_error_rate(r, h):
    r = r.split()
    h = h.split()

    d = np.zeros((len(r) + 1) * (len(h) + 1), dtype=np.uint8)
    d = d.reshape((len(r) + 1, len(h) + 1))
    for i in range(len(r) + 1):
        for j in range(len(h) + 1):
            if i == 0:
```

```
                d[0][j] = j
            elif j == 0:
                d[i][0] = i

    for i in range(1, len(r) + 1):
        for j in range(1, len(h) + 1):
            if r[i - 1] == h[j - 1]:
                d[i][j] = d[i - 1][j - 1]
            else:
                substitution = d[i - 1][j - 1] + 1
                insertion = d[i][j - 1] + 1
                deletion = d[i - 1][j] + 1
                d[i][j] = min(substitution, insertion, deletion)

    return float(d[len(r)][len(h)]) / len(r) * 100

def character_error_rate(r, h):
    r = r.replace(" ", "")
    h = h.replace(" ", "")

    d = np.zeros((len(r) + 1) * (len(h) + 1), dtype=np.uint8)
    d = d.reshape((len(r) + 1, len(h) + 1))
    for i in range(len(r) + 1):
        for j in range(len(h) + 1):
            if i == 0:
                d[0][j] = j
            elif j == 0:
                d[i][0] = i

    for i in range(1, len(r) + 1):
        for j in range(1, len(h) + 1):
            if r[i - 1] == h[j - 1]:
                d[i][j] = d[i - 1][j - 1]
            else:
                substitution = d[i - 1][j - 1] + 1
                insertion = d[i][j - 1] + 1
```

```
                deletion = d[i - 1][j] + 1
                d[i][j] = min(substitution, insertion, deletion)

    return float(d[len(r)][len(h)]) / len(r) * 100

# Lire le fichier Excel
df = pd.read_excel('gdrive/My Drive/Data/audio_text_dataset_with_phrases.xlsx')

wer_scores = []
cer_scores = []

# Calculer le WER et le CER pour chaque ligne et stocker les scores dans des listes
for index, row in df.iterrows():
    wer_score = word_error_rate(row['phrase'], row['transcription'])
    cer_score = character_error_rate(row['phrase'], row['transcription'])

    wer_scores.append(wer_score)
    cer_scores.append(cer_score)

# Calculer la moyenne des WER et CER
average_wer = np.mean(wer_scores)
average_cer = np.mean(cer_scores)

# Afficher les résultats
print("Moyenne du Word Error Rate (WER): {:.2f}%".format(average_wer))
print("Moyenne du Character Error Rate (CER): {:.2f}%".format(average_cer))
```

```
!pip install librosa
!pip install soundfile
!pip install pydub
!pip install soundfile
!pip install librosa --upgrade
!pip install torch
!pip install torchaudio
!pip install huggingsound


import os
import pandas as pd
import torch
import glob
from google.colab import drive

drive.mount('/content/gdrive')
dataset_path = 'gdrive/My Drive/Data'


import random
import numpy as np
import soundfile as sf
import os

# Fonction pour perturbation temporelle
def perturbation_temporelle(audio, shift_range=200):
    shift = random.randint(-shift_range, shift_range)
    augmented_audio = np.roll(audio, shift)
    return augmented_audio

# Fonction pour perturbation de vitesse
def perturbation_vitesse(audio, speed_factor_range=(0.8, 1.2)):
    speed_factor = random.uniform(speed_factor_range[0], speed_factor_range[1])
    num_samples = len(audio)
    new_num_samples = int(num_samples / speed_factor)
    indices = np.round(np.arange(0, num_samples, speed_factor)).astype(int)
    augmented_audio = audio[indices[:new_num_samples]]
    return augmented_audio

# Fonction pour ajout de bruit
def ajout_bruit(audio, noise_factor_range=(0.01, 0.05)):
    noise_factor = random.uniform(noise_factor_range[0], noise_factor_range[1])
```

```
    noise = np.random.randn(len(audio))
    augmented_audio = audio + noise_factor * noise[:, np.newaxis]
    return augmented_audio

# Fonction pour variation de volume
def variation_volume(audio, volume_range=(-10, 10)):
    volume_db = random.uniform(volume_range[0], volume_range[1])
    augmented_audio = audio * (10 ** (volume_db / 20))
    return augmented_audio

# Fonction pour perturbation de fréquence
def perturbation_frequence(audio, pitch_factor_range=(-2, 2)):
    pitch_factor = random.uniform(pitch_factor_range[0], pitch_factor_range[1])
    num_samples = len(audio)
    time_axis = np.arange(num_samples) / sr
    frequency_shift = 2 * np.pi * pitch_factor * time_axis
    augmented_audio = audio * np.cos(frequency_shift[:, np.newaxis])
    return augmented_audio

# Appliquer des perturbations aléatoires à l'audio
def perturber_audio(audio):
    perturbations = [perturbation_temporelle, perturbation_vitesse, ajout_bruit, variation_volume,
perturbation_frequence]
    num_perturbations = random.randint(3, len(perturbations))
    perturbed_audio = audio.copy()
    for _ in range(num_perturbations):
        perturbation = random.choice(perturbations)
        perturbed_audio = perturbation(perturbed_audio)
    return perturbed_audio

# Dossier contenant les audios
audio_dir = 'gdrive/My Drive/audio_dataset/'

# Parcourir les sous-dossiers du dossier audio_dataset
for root, dirs, files in os.walk(audio_dir):
    # Ignorer les sous-dossiers "augmented"
    if os.path.basename(root) == 'augmented':
        continue

    for sub_dir in dirs:
        if sub_dir != 'augmented':
          sub_dir_path = os.path.join(root, sub_dir)
```

```
output_dir = os.path.join(sub_dir_path, 'augmented')

# Vérifier si le dossier "augmented" existe, sinon le créer
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

#if sub_dir != 'augmented':

# Parcourir les fichiers audio dans chaque sous-dossier
for file in os.listdir(sub_dir_path):
    if file.endswith('.wav'):
        audio_path = os.path.join(sub_dir_path, file)

        # Ignorer les fichiers se trouvant dans un dossier "augmented"
        if os.path.basename(os.path.dirname(audio_path)) == 'augmented':
            continue

        # Charger l'audio
        audio, sr = sf.read(audio_path)

        # Liste pour stocker les audios augmentés
        augmented_audios = []

        # Augmentation avec perturbation temporelle
        augmented_audio = perturbation_temporelle(audio)
        augmented_audios.append(augmented_audio)

        # Augmentation avec perturbation de vitesse
        augmented_audio = perturbation_vitesse(audio)
        augmented_audios.append(augmented_audio)

        # Augmentation avec ajout de bruit
        augmented_audio = ajout_bruit(audio)
        augmented_audios.append(augmented_audio)

        # Augmentation avec variation de volume
        augmented_audio = variation_volume(audio)
        augmented_audios.append(augmented_audio)

        # Augmentation avec perturbation de fréquence
        augmented_audio = perturbation_frequence(audio)
        augmented_audios.append(augmented_audio)
```

```
                    # Augmentation avec des perturbations aléatoires
                    augmented_audio = perturber_audio(audio)
                    augmented_audios.append(augmented_audio)

                    # Sauvegarder les audios augmentés
                    for i, augmented_audio in enumerate(augmented_audios):
                        output_file = f'{os.path.splitext(file)[0]}.augmente_{i+1}.wav'
                        output_path = os.path.join(output_dir, output_file)
                        sf.write(output_path, augmented_audio, sr)


import os
import pandas as pd
from huggingsound import SpeechRecognitionModel

# Charger le modèle SpeechRecognition préentraîné
model = SpeechRecognitionModel("jonatasgrosman/wav2vec2-large-xlsr-53-english")


import os
import torch
import pandas as pd


# Définir le chemin du dossier contenant les audios
audio_dataset_folder = "gdrive/My Drive/audio_dataset"

# Liste pour stocker les données de transcription
transcriptions_data = []

# Fonction récursive pour parcourir les fichiers audio dans le dossier et ses sous-dossiers
def parcourir_audio_dossier(folder_path, intent):
    for file_name in os.listdir(folder_path):
        file_path = os.path.join(folder_path, file_name)

        # Ignorer les fichiers dans le dossier "augmented"
        if os.path.isdir(file_path) and file_name == "augmented":
            continue

        if os.path.isfile(file_path) and file_name.endswith(".wav"):
            # Transcription de l'audio avec SpeechRecognitionModel
            transcription = model.transcribe([file_path])
```

```
            transcription = transcription[0]

            # Extraire la classe/intent à partir du chemin du fichier
            if "hotel" in file_path:
                intent = "hotel"
            elif "event" in file_path:
                intent = "event"
            else:
                intent = "none"

            # Ajouter les informations à la liste de données
            transcriptions_data.append({
                "son": os.path.basename(file_path),
                "transcription": transcription['transcription'].upper(),
                "intent": intent
            })

        elif os.path.isdir(file_path):
            # Appel récursif pour parcourir les sous-dossiers
            parcourir_audio_dossier(file_path, intent)

# Parcourir les sous-dossiers du dossier audio_dataset
for root, dirs, files in os.walk(audio_dataset_folder):
    for sub_dir in dirs:
        sub_dir_path = os.path.join(root, sub_dir)

        # Appel de la fonction pour parcourir les fichiers audio dans le sous-dossier
        parcourir_audio_dossier(sub_dir_path, intent=sub_dir)

# Créer un DataFrame à partir des données de transcription
df_transcriptions = pd.DataFrame(transcriptions_data)

# Enregistrer les transcriptions dans un fichier Excel
output_file = "gdrive/My Drive/audio_dataset.xlsx"
df_transcriptions.to_excel(output_file, index=False)


print(f"Les transcriptions ont été enregistrées dans le fichier : {output_file}")
```

```
import pandas as pd

# Charger le DataFrame à partir du fichier Excel
df = pd.read_excel("audio_dataset_equalized_100_type_of_recommendation.xlsx")

# Afficher le nombre total d'entrées avant la suppression des doublons
total_entries = len(df)
print(f"Nombre total d'entrées : {total_entries}")

# Supprimer les doublons basés sur les colonnes 'transcription' et 'intent'
df_unique = df.drop_duplicates(subset=['transcription', 'intent'])

# Afficher le nombre d'entrées uniques après la suppression des doublons
unique_entries = len(df_unique)
duplicates_removed = total_entries - unique_entries
print(f"Nombre d'entrées uniques après la suppression des doublons : {unique_entries}")
print(f"Nombre d'entrées doublons supprimées : {duplicates_removed}")


# Calculer le nombre d'entrées pour chaque intent
counts = df_unique['intent'].value_counts()
min_count = counts.min()  # Nombre minimum d'entrées parmi tous les intents

if min_count == 0:
    print("Tous les intents ont le même nombre d'entrées. Aucune égalisation n'est nécessaire.")
else:
    # Créer un DataFrame vide pour stocker les données égalisées
    df_equalized = pd.DataFrame()
    entries_equalized = 0

    for intent in counts.index:
        # Filtrer le DataFrame pour ne conserver que les entrées correspondant à l'intent
        df_intent = df_unique[df_unique['intent'] == intent]

        if len(df_intent) > min_count:
            # Supprimer les entrées excédentaires pour atteindre le même nombre d'entrées que
min_count
            df_intent = df_intent.sample(min_count, replace=False, random_state=42)
            entries_equalized += len(df_intent) - min_count

        # Ajouter les entrées égalisées au DataFrame final
        df_equalized = pd.concat([df_equalized, df_intent])
```

```
    # Réinitialiser les index du DataFrame égalisé
    df_equalized = df_equalized.reset_index(drop=True)

    # Afficher le nombre d'entrées après l'égalisation
    equalized_entries = len(df_equalized)
    print(f"Nombre d'entrées après l'égalisation : {equalized_entries}")
    print(f"Nombre d'entrées supprimées pour l'égalisation : {entries_equalized}")

    # Enregistrer le DataFrame égalisé dans un nouveau fichier Excel
    output_file_equalized = "audio_dataset_equalized_100_type_of_recommendation.xlsx"
    df_equalized.to_excel(output_file_equalized, index=False)

    print(f"Les doublons ont été supprimés et le nombre d'entrées a été égalisé. Le fichier égalisé
a été enregistré : {output_file_equalized}")



import pandas as pd
import random

def complete_excel(input_file, output_file):
    # Lire le fichier d'entrée contenant 600 entrées
    df_input = pd.read_excel(input_file)

    # Lire le fichier de sortie contenant moins de 100 entrées par intent
    df_output = pd.read_excel(output_file)

    # Compléter chaque intent pour atteindre 100 entrées
    intents = df_output['intent'].unique()  # Récupérer les intents uniques

    for intent in intents:
        # Vérifier combien d'entrées sont déjà présentes pour l'intent
        entries_count = df_output[df_output['intent'] == intent]['transcription'].nunique()

        if entries_count < 100:
            # Sélectionner les entrées de l'intent dans le fichier d'entrée
            available_entries = df_input[df_input['intent'] == intent]

            # Exclure les entrées déjà présentes dans le fichier de sortie
            existing_entries = df_output[df_output['intent'] == intent]['transcription'].unique()
```

```
            available_entries =
available_entries[~available_entries['transcription'].isin(existing_entries)]

            # Limiter le nombre d'entrées à ajouter si nécessaire
            entries_to_add = available_entries.sample(n=min(100 - entries_count,
len(available_entries)), random_state=42)

            # Ajouter les nouvelles entrées à la fin du fichier de sortie
            df_output = pd.concat([df_output, entries_to_add])

    # Afficher le nombre d'entrées uniques par intent
    unique_entries_count = df_output.groupby('intent')['transcription'].nunique()
    print("Nombre d'entrées uniques par intent :\n", unique_entries_count)

    # Supprimer la colonne "son" du fichier de sortie
    #df_output = df_output.drop(columns=['son'])

    # Trier le fichier de sortie par intent
    df_output = df_output.sort_values(by='intent')

    # Réinitialiser les index du fichier de sortie
    df_output = df_output.reset_index(drop=True)

    # Écrire le fichier de sortie complété
    df_output.to_excel(output_file, index=False)

# Appel de la fonction en spécifiant les noms des fichiers d'entrée et de sortie
complete_excel('audio_dataset_equalized.xlsx',
'audio_dataset_equalized_100_type_of_recommendation.xlsx')


import pandas as pd

def interleave_entries(input_file, output_file):
    # Lire le fichier d'entrée trié par intent
    df_input = pd.read_excel(input_file)

    # Extraire les intents uniques
    intents = df_input['intent'].unique()

    # Créer une liste vide pour stocker les entrées entrelacées
    interleaved_entries = []
```

```
    # Calculer le nombre maximum d'entrées par intent
    max_entries_per_intent = df_input.groupby('intent').size().max()

    # Parcourir jusqu'au nombre maximum d'entrées par intent
    for i in range(max_entries_per_intent):
        # Parcourir chaque intent
        for intent in intents:
            # Filtrer les entrées pour l'intent courant
            entries = df_input[df_input['intent'] == intent]

            # Vérifier si l'indice actuel est valide pour cet intent
            if i < len(entries):
                # Ajouter l'entrée à la liste entrelacée
                interleaved_entries.append(entries.iloc[i].to_dict())

    # Créer un nouveau DataFrame à partir de la liste entrelacée
    df_output = pd.DataFrame(interleaved_entries)


    # Écrire le DataFrame dans un fichier Excel
    df_output.to_excel(output_file, index=False)

# Spécifier le nom du fichier d'entrée et de sortie
input_file = 'audio_dataset_equalized_100_type_of_recommendation.xlsx'
output_file = 'audio_dataset_equalized_100_type_of_recommendation.xlsx'

# Exécuter la fonction de transformation
interleave_entries(input_file, output_file)

# Afficher le contenu du fichier d'entrée
print("Contenu du fichier d'entrée :")
df_input = pd.read_excel(input_file)
print(df_input)

# Afficher le contenu du fichier de sortie
print("Contenu du fichier de sortie :")
df_output = pd.read_excel(output_file)
print(df_output)
```

```python
import pandas as pd

# Charger le fichier Excel
df = pd.read_excel("audio_dataset_label.xlsx")

# Vérifier si la colonne "intent" existe
if "intent" not in df.columns:
    # Créer la colonne "intent" avec des valeurs vides
    df["intent"] = ""

# Mettre à jour la colonne "intent" en fonction des valeurs de "son"
df.loc[df["son"].str.contains("userbased", case=False), "intent"] = "userbased"
df.loc[df["son"].str.contains("itembased", case=False), "intent"] = "itembased"
df.loc[~df["intent"].isin(["userbased", "itembased"]), "intent"] = "none"

# Compter le nombre d'entrées par intent
counts = df["intent"].value_counts()

# Enregistrer les modifications dans le fichier Excel
df.to_excel("audio_dataset_label.xlsx", index=False)

# Afficher les résultats
print(counts)
```

```python
import pandas as pd

# Charger le fichier Excel
df = pd.read_excel("audio_dataset.xlsx")

# Obtenir le nombre d'entrées uniques dans la colonne "transcription"
unique_entries = df["transcription"].nunique()

# Afficher le nombre d'entrées uniques
print("Nombre d'entrées uniques dans la colonne 'transcription':", unique_entries)
```

```python
import pandas as pd

# Charger le fichier Excel
df = pd.read_excel("audio_dataset_equalized.xlsx")
```

```
# Vérifier si la colonne "intent" existe déjà
if "intent" not in df.columns:
    # Créer la colonne "intent" si elle n'existe pas
    df["intent"] = ""

# Parcourir chaque ligne du DataFrame
for index, row in df.iterrows():
    son = row["son"]
    segments = son.split(".")
    intent = segments[0] if len(segments) > 0 else "none"
    df.at[index, "intent"] = intent

# Afficher le DataFrame mis à jour
print(df)

# Enregistrer le DataFrame dans un nouveau fichier Excel
df.to_excel("audio_dataset_equalized.xlsx", index=False)
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
import shutil
import os

# Chemin du dossier que vous souhaitez supprimer
chemin_dossier = "nlp"

# Vérifier si le dossier existe
if os.path.exists(chemin_dossier):
    # Supprimer le dossier
    shutil.rmtree(chemin_dossier)
    print("Le dossier a été supprimé.")
else:
    print("Le dossier n'existe pas.")
```

```
import shutil
```

```
# Chemin du dossier source que vous souhaitez copier
chemin_dossier_source = "gdrive/MyDrive/Recommender_Classifier/"

# Chemin du dossier de destination où vous souhaitez copier le dossier source
chemin_dossier_destination = "nlp"

# Copie du dossier source vers la destination
shutil.copytree(chemin_dossier_source, chemin_dossier_destination)

# Installation des dépendances spécifiées dans le fichier "requirements.txt" du dossier "nlp"
!pip install -r ./nlp/requirements.txt


# Recherche d'hyperparamètres sur le jeu de données de classification de l'objet de la
recommandation
!python nlp/search_hyperparams_class_dataset.py --data_dir nlp/data/recommender_class_audio_dataset/
--parent_dir nlp/experiments/recommender_class_model/


# Synthèses des résultats des expériences du modèle
!python nlp/synthesize_results.py --parent_dir nlp/experiments/recommender_class_model/


# Liste des répertoires des différents modèles à évaluer
model_dirs = [

    "nlp/experiments/recommender_class_model/learning_rate_2e-05",
    "nlp/experiments/recommender_class_model/learning_rate_3e-05",
    "nlp/experiments/recommender_class_model/learning_rate_4e-05"

]

# Boucle pour évaluer chaque modèle dans la liste
for model_dir in model_dirs:
    command = f"python nlp/evaluate.py --data_dir nlp/data/recommender_class_audio_dataset/ --
model_dir {model_dir}"
    !{command}
    print(model_dir)


# Exécution du script plot.py pour générer le graphique
!python nlp/plot.py --parent_dir nlp/train/train_class_model.txt --learning_rate 4e-05 --output_file
class_model_plot.png
from IPython.display import Image
```

```
# Affichage du graphe généré
Image('class_model_plot.png')


# Recherche d'hyperparamètres sur le jeu de données de classification du type de recommandation
!python nlp/search_hyperparams_type_dataset.py --data_dir nlp/data/recommender_type_audio_dataset/ -
-parent_dir nlp/experiments/recommender_type_model/


# Synthèses des résultats des expériences du modèle
!python nlp/synthesize_results.py --parent_dir  nlp/experiments/recommender_type_model/


# Liste des répertoires des différents modèles à évaluer
model_dirs = [
    "nlp/experiments/recommender_type_model/learning_rate_2e-05",
    "nlp/experiments/recommender_type_model/learning_rate_3e-05",
    "nlp/experiments/recommender_type_model/learning_rate_4e-05"
]

# Boucle pour évaluer chaque modèle dans la liste
for model_dir in model_dirs:
    command = f"python nlp/evaluate.py --data_dir nlp/data/recommender_type_audio_dataset/ --
model_dir {model_dir}"
    !{command}
    print(model_dir)


# Exécution du script plot.py pour générer le graphique
!python nlp/plot.py --parent_dir nlp/train/train_type_model.txt --learning_rate 2e-05 --output_file
type_model_plot.png
from IPython.display import Image
# Affichage du graphe généré
Image('type_model_plot.png')


import random
import numpy as np
import os
import sys
from transformers import BertTokenizer
import torch
from torch.autograd import Variable
```

```
import utils

class DataLoader(object):
    """
    Gère tous les aspects des données. Stocke les dataset_params, le vocabulaire et les tags avec
leurs correspondances d'indices.
    """
    def __init__(self, data_dir, params):
        """
        Charge les dataset_params, le vocabulaire et les tags. Assurez-vous d'avoir exécuté
`build_vocab.py` sur data_dir avant d'utiliser cette classe.

        Args:
            data_dir: (string) répertoire contenant le jeu de données
            params: (Params) hyperparamètres du processus d'entraînement. Cette fonction modifie les
params et ajoute dataset_params (comme la taille du vocabulaire, le nombre de tags, etc.) à params.
        """

        # chargement des dataset_params
        json_path = os.path.join(data_dir, 'dataset_params.json')
        assert os.path.isfile(json_path), "Aucun fichier JSON trouvé à {}, exécutez
build_vocab.py".format(json_path)
        self.dataset_params = utils.Params(json_path)

        # chargement du vocabulaire (nous en avons besoin pour mapper les mots à leurs indices)
        vocab_path = os.path.join(data_dir, 'sentences.txt')
        self.vocab = {}
        with open(vocab_path, encoding='latin-1') as f:
            for i, l in enumerate(f.read().splitlines()):
                self.vocab[l] = i

        # chargement des tags (nous en avons besoin pour mapper les tags à leurs indices)
        tags_path = os.path.join(data_dir, 'intent_label.txt')
        self.tag_map = {}
        with open(tags_path) as f:
            for i, t in enumerate(f.read().splitlines()):
                self.tag_map[t] = i

        # ajout des paramètres du jeu de données à params (par ex. taille du vocabulaire)
        params.update(json_path)
```

```
    def load_sentences_labels(self, sentences_file, labels_file, d):
        """
        Charge les phrases et les tags à partir de leurs fichiers respectifs. Mappe les tokens et
les tags à leurs indices et les stocke dans le dictionnaire fourni d.

        Args:
            sentences_file: (string) fichier contenant les phrases avec les tokens séparés par des
espaces
            labels_file: (string) fichier contenant les tags pour les phrases dans labels_file
            d: (dict) un dictionnaire dans lequel les données chargées sont stockées
        """

        sentences = []
        intent_labels = []
        labels = []

        with open(sentences_file) as f:
            for sentence in f.read().splitlines():
                sentences.append(sentence)

        with open(labels_file) as f:
            for label in f.read().splitlines():
                intent_labels.append(label)
                # remplacer chaque tag par son index
                label_index = self.tag_map[label]
                labels.append(label_index)

        # vérification pour s'assurer qu'il y a un tag pour chaque token
        assert len(labels) == len(sentences)

        # stockage des phrases et des tags dans le dictionnaire d
        d['data'] = sentences
        d['labels'] = labels
        d['size'] = len(sentences)


    def load_data(self, types, data_dir):
        """
        Charge les données pour chaque type dans types depuis data_dir.

        Args:
```

```
            types: (list) contient un ou plusieurs des éléments ‘train’, ‘val’, ‘test’ en fonction
des données requises
            data_dir: (string) répertoire contenant le jeu de données

        Returns:
            data: (dict) contient les données avec les tags pour chaque type dans types

        """
        data = {}

        for split in [‘train’, ‘val’, ‘test’]:
            if split in types:
                sentences_file = os.path.join(data_dir, split, "sentences.txt")
                labels_file = os.path.join(data_dir, split, "labels.txt")
                data[split] = {}
                self.load_sentences_labels(sentences_file, labels_file, data[split])

        return data

    def data_iterator(self, data, params, shuffle=False):
        """
        Renvoie un générateur qui génère des lots de données avec les tags. La taille du lot est
params.batch_size. Expire après un seul passage sur les données.

        Args:
            data: (dict) contient les données qui ont les clés ‘data’, ‘labels’ et ‘size’
            params: (Params) hyperparamètres du processus d’entraînement.
            shuffle: (bool) indique si les données doivent être mélangées

        Yields:
            batch_data: (Variable) dimension batch_size x seq_len avec les données de la phrase
            batch_labels: (Variable) dimension batch_size x seq_len avec les tags correspondants

        """
        tokenizer = BertTokenizer.from_pretrained(‘bert-base-uncased’)
        # crée une liste qui décide de l’ordre dans lequel nous parcourons les données- cela évite
le mélange explicite des données
        order = list(range(data[‘size’]))
        if shuffle:
            random.seed(230)
            random.shuffle(order)
```

```
        # un seul passage sur les données
        for i in range((data['size']+1)//params.batch_size):
            # récupère les phrases et les tags
            batch_sentences = [data['data'][idx] for idx in
order[i*params.batch_size:(i+1)*params.batch_size]]
            batch_tags = [data['labels'][idx] for idx in
order[i*params.batch_size:(i+1)*params.batch_size]]
            # calcule la longueur de la phrase la plus longue dans le lot
            batch_max_len = max([len(s) for s in batch_sentences])

            # prépare un tableau numpy avec les données, en initialisant les données avec pad_ind et
            tous les tags avec -1
            # l'initialisation des tags à -1 différencie les tokens avec les tags des tokens de p
            adding
            batch_data = tokenizer(batch_sentences, max_length=128, padding='max_length',
                       truncation=True, return_tensors="pt")
            batch_labels = batch_tags

            # puisque toutes les données sont des indices, nous les convertissons en tenseurs
            LongTensors torch
            batch_labels = torch.LongTensor(batch_labels)
            input_ids = torch.LongTensor(batch_data["input_ids"])
            attention_mask = torch.LongTensor(batch_data["attention_mask"])
            token_type_ids = torch.LongTensor(batch_data["token_type_ids"])

            # transférer les tenseurs sur le GPU s'ils sont disponibles
            if params.cuda:
                input_ids, attention_mask, token_type_ids, batch_labels = input_ids.cuda(),
attention_mask.cuda(), token_type_ids.cuda(), batch_labels.cuda()

            yield input_ids, attention_mask, token_type_ids, batch_labels


import numpy as np
import torch
import torch.nn as nn
import torch.nn.functional as F
from transformers import BertModel
from torchmetrics.classification import MulticlassF1Score, MulticlassRecall

class Net(nn.Module):
    """
```

```
    C’est la façon standard de définir son propre réseau dans PyTorch. Vous choisissez généralement
les composants
    (par exemple des LSTMs, des couches linéaires, etc.) de votre réseau dans la fonction __init__.
Ensuite, vous appliquez ces couches
    sur l’entrée étape par étape dans la fonction forward. Vous pouvez utiliser torch.nn.functional
pour appliquer des fonctions
    telles que F.relu, F.sigmoid, F.softmax. Assurez-vous de vous assurer que les dimensions sont
correctes après chaque étape.

    Nous vous encourageons à regarder le réseau dans pytorch/vision/model/net.py pour mieux
comprendre comment vous pouvez
    définir votre propre réseau.

    La documentation pour tous les différents composants disponibles est ici:
http://pytorch.org/docs/master/nn.html
    """

    def __init__(self, params):
        """
        Nous définissons un réseau récurrent qui prédit les tags NER pour chaque token de la phrase.
Les composants
        nécessaires sont:

        - une couche d’embedding: cette couche associe chaque index dans
l’intervalle(params.vocab_size) à un vecteur de params.embedding_dim
        - lstm: en appliquant le LSTM sur l’entrée séquentielle, on obtient une sortie pour chaque
token de la phrase
        - fc: une couche complètement connectée qui convertit la sortie du LSTM pour chaque token en
une distribution sur les tags NER

        Args:
            params: (Params) contient vocab_size, embedding_dim, lstm_hidden_dim
        """
        super(Net, self).__init__()

        self.bert = BertModel.from_pretrained(‘bert-base-uncased’)
        self.dropout = nn.Dropout(0.2)
        self.classifier = nn.Linear(params.hidden_size, params.number_of_tags)

    def forward(self, input_ids, attention_mask, token_type_ids):
        """
```

```
        Cette fonction définit comment nous utilisons les composants de notre réseau pour
fonctionner sur un lot d'entrée.

        Args:
            input_ids: (Variable) contient un lot de phrases, de dimension batch_size x seq_len, où
seq_len est
                la longueur de la phrase la plus longue dans le lot. Pour les phrases plus courtes
que seq_len, les tokens restants sont des tokens de remplissage (PADding). Chaque ligne est une
phrase avec chaque élément correspondant à l'index du token dans le vocabulaire.

        Returns:
            out: (Variable) dimension batch_size*seq_len x num_tags avec les log probabilités des
tokens pour chaque token
                 de chaque phrase.

        Note: les dimensions après chaque étape sont fournies
        """
        #                                -> batch_size x seq_len
        outputs = []
        bert_output = self.bert(input_ids=input_ids, attention_mask=attention_mask,
token_type_ids=token_type_ids)
        pooled_output = bert_output.pooler_output
        pooled_output = self.dropout(pooled_output)

        # appliquer la couche complètement connectée et obtenir la sortie
        logits = self.classifier(pooled_output)
        output = F.relu(logits)
        return output


def loss_fn(outputs, labels, params):
    """
    Calcule la perte d'entropie croisée donnée les sorties du modèle et les étiquettes pour chaque
phrase.

    Args:
        outputs: (Tensor) dimension batch_size x num_classes - logits de sortie du modèle
        labels: (Tensor) dimension batch_size - étiquettes cibles pour chaque phrase

    Returns:
        loss: (Tensor) perte d'entropie croisée pour le lot
    """
```

```
    loss = F.cross_entropy(outputs.view(-1, params.number_of_tags), labels.view(-1))
    return loss


def accuracy(outputs, labels):
    """
    Calcule la précision, donnée les sorties et les étiquettes pour tous les tokens. Exclut les
termes de remplissage (PADding).

    Args:
        outputs: (np.ndarray) dimension batch_size*seq_len x num_tags - sortie log softmax du modèle
        labels: (np.ndarray) dimension batch_size x seq_len où chaque élément est soit une étiquette
dans
                [0, 1, ... num_tag-1], ou -1 s'il s'agit d'un token de remplissage (PADding).

    Returns: (float) précision dans [0,1]
    """

    # remodeler les étiquettes pour donner un vecteur plat de longueur batch_size*seq_len
    labels = labels.ravel()

    # puisque les tokens de remplissage (PADding) ont l'étiquette -1, nous pouvons générer un masque
pour exclure la perte de ces termes
    mask = (labels >= 0)

    # np.argmax nous donne la classe prédite pour chaque token par le modèle
    outputs = np.argmax(outputs, axis=1)

    # comparer les sorties avec les étiquettes et diviser par le nombre de tokens (en excluant les
tokens de remplissage - PADding)
    return np.sum(outputs == labels) / float(np.sum(mask))

def recall(outputs, labels):
    outputs_tensor = torch.from_numpy(outputs)
    labels_tensor = torch.from_numpy(labels)
    metric = MulticlassRecall(num_classes=100)
    recall = metric(outputs_tensor, labels_tensor)

    return recall

def f1_score(outputs, labels):
    outputs_tensor = torch.from_numpy(outputs)
```

```
    labels_tensor = torch.from_numpy(labels)
    metric = MulticlassF1Score(num_classes=100)
    f1_score = metric(outputs_tensor, labels_tensor)
    return f1_score

# maintenir toutes les métriques requises dans ce dictionnaire - celles-ci sont utilisées dans les
boucles d'entraînement et d'évaluation
metrics = {
    'accuracy': accuracy
    # on pourrait ajouter plus de métriques telles que la précision pour chaque type de token
}



"""Lecture, division et sauvegarde du dataset audio pour notre modèle"""

import csv
import os
import sys
import pandas as pd
from sklearn.model_selection import train_test_split

def load_dataset(path_excel):
    """Charge le dataset en mémoire à partir du fichier Excel"""
    # Lire le fichier Excel dans un dataframe pandas
    df = pd.read_excel(path_excel)

    dataset = []
    sentence, label = [], []

    # Chaque ligne du dataframe correspond à une phrase
    for idx, row in df.iterrows():
        text, label = row['transcription'], row['intent']
        # Si le texte n'est pas vide, cela signifie que nous avons atteint une nouvelle phrase
        if len(str(text)) != 0:
            if len(sentence) > 0:
                dataset.append((sentence, label))
                sentence, label = [], []
        try:
            text, label = str(text), str(label)
            sentence.append(text)
        except UnicodeDecodeError as e:
```

```
            print("Une exception a été levée, sautant un mot: {}".format(e))
            pass
    return dataset

def save_dataset(dataset, save_dir):
    """Écrit les fichiers sentences.txt et labels.txt dans save_dir à partir du dataset

    Args:
        dataset: ([["a", "cat"], ["O", "O"]), ...])
        save_dir: (string)
    """
    # Créer le répertoire s'il n'existe pas
    print("Sauvegarde dans {}...".format(save_dir))
    if not os.path.exists(save_dir):
        os.makedirs(save_dir)

    # Exporter le dataset
    with open(os.path.join(save_dir, 'sentences.txt'), 'w') as file_sentences, \
         open(os.path.join(save_dir, 'labels.txt'), 'w') as file_labels:
        for sentence, label in dataset:
            file_sentences.write("{}\n".format(sentence))
            file_labels.write("{}\n".format(label))
    print("- Terminé.")

if __name__ == "__main__":
    # Vérifier que le dataset existe (assurez-vous de ne pas avoir téléchargé le fichier `ner.csv`)
    path_dataset = 'nlp/data/audio_dataset/audio_dataset_equalized.xlsx'
    msg = "Fichier {} introuvable. Assurez-vous d'avoir téléchargé le bon
dataset".format(path_dataset)
    assert os.path.isfile(path_dataset), msg

    # Charger le dataset en mémoire
    print("Chargement du dataset en mémoire...")
    dataset = load_dataset(path_dataset)
    print("- Terminé.")


    # Diviser le dataset en ensembles d'entraînement et de test (70% train, 30% test)
    train_dataset, test_dataset = train_test_split(dataset, test_size=0.3)

    # Diviser l'ensemble de test en ensembles de validation et de test (50% val, 50% test)
    val_dataset, test_dataset = train_test_split(test_dataset, test_size=0.5)
```

```
    # Sauvegarder les datasets dans des fichiers
    save_dataset(train_dataset, 'nlp/data/audio_dataset/train')
    save_dataset(val_dataset, 'nlp/data/audio_dataset/val')
    save_dataset(test_dataset, 'nlp/data/audio_dataset/test')



"""Construit les vocabulaires de mots et d'étiquettes à partir des datasets"""

import argparse
from collections import Counter
import json
import os


parser = argparse.ArgumentParser()
parser.add_argument('--min_count_word', default=1, help="Nombre minimum d'occurrences pour les mots
dans le dataset", type=int)
parser.add_argument('--min_count_tag', default=1, help="Nombre minimum d'occurrences pour les
étiquettes dans le dataset", type=int)
parser.add_argument('--data_dir', default='data/audio_dataset', help="Répertoire contenant le
dataset")


def save_vocab_to_txt_file(vocab, txt_path):
    """Écrit un token par ligne, l'ID de ligne basé sur 0 correspond à l'ID du token.

    Args:
        vocab: (objet itérable) génère des tokens
        txt_path: (chaine de caractères) chemin du fichier vocab
    """
    with open(txt_path, "w") as f:
        for token in vocab:
            f.write(token + '\n')


def save_dict_to_json(d, json_path):
    """Sauvegarde un dictionnaire dans un fichier json

    Args:
        d: (dictionnaire)
        json_path: (chaine de caractères) chemin du fichier json
```

```
    """
    with open(json_path, 'w') as f:
        d = {k: v for k, v in d.items()}
        json.dump(d, f, indent=4)


def update_vocab(txt_path, vocab):
    """Met à jour les vocabulaires de mots et d'étiquettes à partir du dataset

    Args:
        txt_path: (chaine de caractères) chemin du fichier, une phrase par ligne
        vocab: (dictionnaire ou Counter) avec la méthode update

    Returns:
        dataset_size: (entier) nombre d'éléments dans le dataset
    """
    with open(txt_path) as f:
        for i, line in enumerate(f):
            vocab.update(line.strip().split('\n'))

    return i + 1


if __name__ == '__main__':
    args = parser.parse_args()

    # Construire le vocabulaire de mots à partir des datasets d'entraînement et de test
    print("Construction du vocabulaire de mots...")
    sentences = Counter()
    size_train_sentences = update_vocab(os.path.join(args.data_dir, 'train/sentences.txt'),
sentences)
    size_dev_sentences = update_vocab(os.path.join(args.data_dir, 'val/sentences.txt'), sentences)
    size_test_sentences = update_vocab(os.path.join(args.data_dir, 'test/sentences.txt'), sentences)
    print("- Terminé.")

    # Construire le vocabulaire d'étiquettes à partir des datasets d'entraînement et de test
    print("Construction du vocabulaire d'étiquettes...")
    tags = Counter()
    size_train_tags = update_vocab(os.path.join(args.data_dir, 'train/labels.txt'), tags)
    size_dev_tags = update_vocab(os.path.join(args.data_dir, 'val/labels.txt'), tags)
    size_test_tags = update_vocab(os.path.join(args.data_dir, 'test/labels.txt'), tags)
    print("- Terminé.")
```

```
    # Vérifier que le nombre d'exemples est le même dans tous les datasets
    assert size_train_sentences == size_train_tags
    assert size_dev_sentences == size_dev_tags
    assert size_test_sentences == size_test_tags

    # Sauvegarder les vocabulaires dans des fichiers
    print("Sauvegarde des vocabulaires dans des fichiers...")
    save_vocab_to_txt_file(sentences, os.path.join(args.data_dir, 'sentences.txt'))
    save_vocab_to_txt_file(tags, os.path.join(args.data_dir, 'intent_label.txt'))
    print("- Terminé.")

    # Sauvegarder les propriétés des datasets dans un fichier json
    sizes = {
        'train_size': size_train_sentences,
        'dev_size': size_dev_sentences,
        'test_size': size_test_sentences,
        'vocab_size': len(sentences),
        'number_of_tags': len(tags)
    }
    save_dict_to_json(sizes, os.path.join(args.data_dir, 'dataset_params.json'))

    # Affichage des tailles
    to_print = "\n".join("- {}: {}".format(k, v) for k, v in sizes.items())
    print("Caractéristiques du dataset:\n{}".format(to_print))



"""Évalue le modèle"""

import argparse
import logging
import os

import numpy as np
import torch
import utils
import model.net as net
from model.data_loader import DataLoader

parser = argparse.ArgumentParser()
parser.add_argument('--data_dir', default='data/small', help="Répertoire contenant le dataset")
```

```
parser.add_argument('--model_dir', default='experiments/base_model', help="Répertoire contenant
params.json")
parser.add_argument('--restore_file', default='best', help="Nom du fichier dans --model_dir
contenant les poids à charger")


def evaluate(model, loss_fn, data_iterator, metrics, params, num_steps):
    """Évalue le modèle sur `num_steps` lots.

    Args:
        model: (torch.nn.Module) le réseau neuronal
        loss_fn: une fonction qui prend en entrée la sortie du modèle et les étiquettes du lot et
calcule la perte pour le lot
        data_iterator: (générateur) un générateur qui génère des lots de données et d'étiquettes
        metrics: (dict) un dictionnaire de fonctions qui calculent une métrique en utilisant la
sortie et les étiquettes de chaque lot
        params: (Params) hyperparamètres
        num_steps: (int) nombre de lots à évaluer, chaque lot ayant une taille params.batch_size
    """

    # mettre le modèle en mode évaluation
    model.eval()

    # résumé pour la boucle d'évaluation actuelle
    summ = []
    outputs_epoch = []
    labels_epoch = []

    # calculer les métriques sur l'ensemble de données
    for _ in range(num_steps):
        # récupérer le prochain lot d'évaluation
        input_ids, attention_mask, token_type_ids, labels_batch = next(data_iterator)

        # calculer la sortie du modèle et la perte
        output_batch = model(input_ids, attention_mask, token_type_ids)
        loss = loss_fn(output_batch, labels_batch, params)

        # extraire les données de la Variable torch, les déplacer vers le CPU, les convertir en
tableaux numpy
        output_batch = output_batch.data.cpu().numpy()
        labels_batch = labels_batch.data.cpu().numpy()
```

```
        # calculer toutes les métriques sur ce lot
        summary_batch = {metric: metrics[metric](output_batch, labels_batch)
                         for metric in metrics}
        summary_batch['loss'] = loss.item()
        summ.append(summary_batch)
        output_batch = np.argmax(output_batch, axis=1)
        outputs_epoch = np.concatenate((outputs_epoch, output_batch))
        labels_epoch = np.concatenate((labels_epoch, labels_batch))

    recall = utils.recall(outputs_epoch, labels_epoch)
    f1_score = utils.f1_score(outputs_epoch, labels_epoch)

    # calculer la moyenne de toutes les métriques dans le résumé
    metrics_mean = {metric: np.mean([x[metric] for x in summ]) for metric in summ[0]}
    metrics_mean['Recall'] = recall
    metrics_mean['F1_score'] = f1_score
    metrics_string = " ; ".join("{}: {:05.3f}".format(k, v) for k, v in metrics_mean.items())
    logging.info("- Métriques d'évaluation : " + metrics_string)
    return metrics_mean


if __name__ == '__main__':
    """
        Évalue le modèle sur l'ensemble de test.
    """
    # Charger les paramètres
    args = parser.parse_args()
    json_path = os.path.join(args.model_dir, 'params.json')
    assert os.path.isfile(json_path), "Aucun fichier de configuration JSON trouvé à
{}".format(json_path)
    params = utils.Params(json_path)

    # utiliser le GPU si disponible
    params.cuda = torch.cuda.is_available()     # utiliser le GPU si disponible

    # Définir la graine aléatoire pour des expériences reproductibles
    torch.manual_seed(230)
    if params.cuda: torch.cuda.manual_seed(230)

    # Obtenir le journal
    utils.set_logger(os.path.join(args.model_dir, 'evaluate.log'))
```

```
    # Créer le pipeline de données d'entrée
    logging.info("Création du dataset...")

    # charger les données
    data_loader = DataLoader(args.data_dir, params)
    data = data_loader.load_data(['test'], args.data_dir)
    test_data = data['test']

    # spécifier la taille de l'ensemble de test
    params.test_size = test_data['size']
    test_data_iterator = data_loader.data_iterator(test_data, params)

    logging.info("- Terminé.")

    # Définir le modèle
    model = net.Net(params).cuda() if params.cuda else net.Net(params)

    loss_fn = net.loss_fn
    metrics = net.metrics

    logging.info("Début de l'évaluation")

    # Recharger les poids à partir du fichier sauvegardé
    utils.load_checkpoint(os.path.join(args.model_dir, args.restore_file + '.pth.tar'), model)

    # Évaluer
    num_steps = (params.test_size + 1) // params.batch_size
    test_metrics = evaluate(model, loss_fn, test_data_iterator, metrics, params, num_steps)
    save_path = os.path.join(args.model_dir, "metrics_test_{}.json".format(args.restore_file))
    utils.save_dict_to_json(test_metrics, save_path)
```

```
import argparse
import re
import matplotlib.pyplot as plt
from IPython.display import Image
import numpy as np

# Parser les arguments en ligne de commande
parser = argparse.ArgumentParser(description="Trace la courbe de perte à partir d'un fichier texte")
```

```
parser.add_argument("--parent_dir", type=str, help="Chemin du fichier texte contenant les
informations de perte")
parser.add_argument("--learning_rate", type=str, help="Valeur du taux d'apprentissage (learning
rate)")
parser.add_argument("--output_file", type=str, help="Nom du fichier de sortie pour l'image")
args = parser.parse_args()

# Vérifier si le chemin du fichier est fourni
if not args.parent_dir:
    raise ValueError("Le chemin du fichier doit être spécifié avec l'argument --parent_dir")

pertes_entrainement = []
pertes_validation = []
nombre_epochs = 0

with open(args.parent_dir, "r") as fichier:
    lignes = fichier.readlines()
    for ligne in lignes:
        if "Epoch" in ligne:
            match = re.search(r"Epoch (\d+)", ligne)
            if match:
                epoch = int(match.group(1))
                if epoch > nombre_epochs:
                    nombre_epochs = epoch
        elif "loss" in ligne:
            correspondance = re.search(r"loss: ([\d.]+)", ligne)
            if correspondance:
                perte = float(correspondance.group(1))
                if "Train" in ligne:
                    pertes_entrainement.append(perte)
                elif "Eval" in ligne:
                    pertes_validation.append(perte)

# Tronquer les listes de pertes si le nombre d'époques est différent
pertes_entrainement = pertes_entrainement[:nombre_epochs]
pertes_validation = pertes_validation[:nombre_epochs]

# Tracer le graphique des pertes
plt.plot(range(1, nombre_epochs + 1), pertes_entrainement, label="Perte Entraînement")
plt.plot(range(1, nombre_epochs + 1), pertes_validation, label="Perte Validation")
plt.xlabel("Époque")
plt.ylabel("Perte")
```

```
plt.title(f"Courbe de perte (taux d'apprentissage = {args.learning_rate})")
plt.xticks(range(1, nombre_epochs + 1))
plt.yticks(np.arange(0.1, max(max(pertes_entrainement), max(pertes_validation))+0.1, 0.1))  #
Réglage de l'échelle des pertes
plt.legend()

# Enregistrer la figure en tant qu'image avec le nom spécifié
fichier_sortie = args.output_file if args.output_file else "courbe_perte.png"
plt.savefig(fichier_sortie)

# Afficher l'image
Image(fichier_sortie)



"""Effectue une recherche d'hyperparamètres"""

import argparse
import os
from subprocess import check_call
import sys

import utils

PYTHON = sys.executable
parser = argparse.ArgumentParser()
parser.add_argument('--parent_dir', default='experiments/learning_rate',
                    help='Répertoire contenant params.json')
parser.add_argument('--data_dir', default='data/SmartTourismDataset', help="Répertoire contenant le
dataset")


def lancer_tache_entrainement(parent_dir, data_dir, nom_tache, params):
    """Lance l'entraînement du modèle avec un ensemble d'hyperparamètres dans parent_dir/nom_tache

    Args:
        parent_dir: (string) répertoire contenant la configuration, les poids et les journaux
        data_dir: (string) répertoire contenant le dataset
        params: (dict) contenant les hyperparamètres
    """
    # Crée un nouveau dossier dans parent_dir avec le nom "nom_tache"
    model_dir = os.path.join(parent_dir, nom_tache)
    if not os.path.exists(model_dir):
```

```
        os.makedirs(model_dir)

    # Écrit les paramètres dans un fichier json
    json_path = os.path.join(model_dir, 'params.json')
    params.save(json_path)

    # Lance l'entraînement avec cette configuration
    cmd = "{python} nlp/train.py --model_dir={model_dir} --data_dir
{data_dir}".format(python=PYTHON, model_dir=model_dir,
                                                                    data_dir=data_dir
)
    print(cmd)
    check_call(cmd, shell=True)


if __name__ == "__main__":
    # Charger les paramètres de référence à partir du fichier json parent_dir
    args = parser.parse_args()
    json_path = os.path.join(args.parent_dir, 'params.json')
    assert os.path.isfile(json_path), "Aucun fichier de configuration JSON trouvé à
{}".format(json_path)
    params = utils.Params(json_path)

    # Effectuer la recherche d'hyperparamètres sur un paramètre
    taux_apprentissage = [2e-5, 3e-5, 4e-5]

    for lr in taux_apprentissage:
        # Modifier le paramètre approprié dans params
        params.learning_rate = lr

        # Lancer la tâche (le nom doit être unique)
        nom_tache = "learning_rate_{}".format(lr)
        lancer_tache_entrainement(args.parent_dir, args.data_dir, nom_tache, params)



"""Agrège les résultats à partir des fichiers metrics_val_best_weights.json dans un dossier
parent"""

import argparse
import json
import os
```

```
from tabulate import tabulate


parser = argparse.ArgumentParser()
parser.add_argument('--parent_dir', default='experiments',
                    help='Répertoire contenant les résultats des expériences')


def agréger_metrics(parent_dir, metrics):
    """Agrège les métriques de toutes les expériences dans le dossier `parent_dir`.

    Suppose que `parent_dir` contient plusieurs expériences, avec leurs résultats stockés dans
    `parent_dir/sous_dossier/metrics_dev.json`

    Args:
        parent_dir: (string) chemin du répertoire contenant les résultats des expériences
        metrics: (dict) sous_dossier -> {'accuracy': ..., ...}
    """
    # Obtient les métriques pour le dossier s'il contient des résultats d'une expérience
    fichier_metrics = os.path.join(parent_dir, 'metrics_val_best_weights.json')
    if os.path.isfile(fichier_metrics):
        with open(fichier_metrics, 'r') as f:
            metrics[parent_dir] = json.load(f)

    # Vérifie chaque sous-dossier de parent_dir
    for sous_dossier in os.listdir(parent_dir):
        if not os.path.isdir(os.path.join(parent_dir, sous_dossier)):
            continue
        else:
            agréger_metrics(os.path.join(parent_dir, sous_dossier), metrics)


def metrics_to_table(metrics):
    # Obtient les en-têtes à partir du premier sous-dossier. Suppose que tout a les mêmes métriques
    en_têtes = metrics[list(metrics.keys())[0]].keys()
    table = [[sous_dossier] + [valeurs[h] for h in en_têtes] for sous_dossier, valeurs in
metrics.items()]
    res = tabulate(table, en_têtes, tablefmt='pipe')

    return res


if __name__ == "__main__":
```

```
    args = parser.parse_args()

    # Agrège les métriques à partir du répertoire args.parent_dir
    metrics = dict()
    agréger_metrics(args.parent_dir, metrics)
    table = metrics_to_table(metrics)

    # Affiche le tableau dans le terminal
    print(table)

    # Sauvegarde les résultats dans parent_dir/results.md
    fichier_sauvegarde = os.path.join(args.parent_dir, "results.md")
    with open(fichier_sauvegarde, 'w') as f:
        f.write(table)


"""Entraîne le modèle"""

import argparse
import logging
import os
from torch.utils.tensorboard import SummaryWriter
import numpy as np
import torch
import torch.optim as optim
from tqdm import trange

import utils
import model.net as net
from model.data_loader import DataLoader
from evaluate import evaluate


parser = argparse.ArgumentParser()
parser.add_argument('--data_dir', default='data/SmartTourismDataset',
                    help="Répertoire contenant le jeu de données")
parser.add_argument('--model_dir', default='experiments/base_model',
                    help="Répertoire contenant params.json")
parser.add_argument('--restore_file', default=None,
                    help="Optionnel, nom du fichier dans --model_dir contenant les poids à charger
avant \
                    l'entraînement")  # 'best' or 'train'
```

```
def train(model, optimizer, loss_fn, data_iterator, metrics, params, num_steps):
    """Entraîne le modèle sur `num_steps` lots.

    Args:
        model: (torch.nn.Module) le réseau neuronal
        optimizer: (torch.optim) optimiseur pour les paramètres du modèle
        loss_fn: une fonction prenant batch_output et batch_labels et calculant la perte pour le lot
        data_iterator: (générateur) un générateur qui génère des lots de données et d'étiquettes
        metrics: (dict) un dictionnaire de fonctions qui calculent une métrique en utilisant la
sortie et les étiquettes de chaque lot
        params: (Params) hyperparamètres
        num_steps: (int) nombre de lots sur lesquels s'entraîner, chacun de taille params.batch_size
    """

    # mettre le modèle en mode entraînement
    model.train()

    # résumé pour la boucle d'entraînement actuelle et objet de moyenne mobile pour la perte
    summ = []
    loss_avg = utils.RunningAverage()
    writer = SummaryWriter()

    outputs_epoch = []
    labels_epoch = []

    # Utiliser tqdm pour la barre de progression
    t = trange(num_steps)
    for i in t:
        # récupérer le lot d'entraînement suivant
        input_ids, attention_mask, token_type_ids, labels_batch = next(data_iterator)

        # calculer la sortie du modèle et la perte
        output_batch = model(input_ids, attention_mask, token_type_ids)
        loss = loss_fn(output_batch, labels_batch, params)
        writer.add_scalar('Loss', loss, i)


        # effacer les gradients précédents, calculer les gradients de toutes les variables par
rapport à la perte
        optimizer.zero_grad()
        loss.backward()
```

```
        # effectuer les mises à jour en utilisant les gradients calculés
        optimizer.step()

        # évaluer les résumés seulement de temps en temps
        if i % params.save_summary_steps == 0:
            # extraire les données de la variable torch, les déplacer sur le cpu, les convertir en
tableaux numpy
            output_batch = output_batch.data.cpu().numpy()
            labels_batch = labels_batch.data.cpu().numpy()

            # calculer toutes les métriques sur ce lot
            summary_batch = {metric: metrics[metric](output_batch, labels_batch)
                             for metric in metrics}
            summary_batch['loss'] = loss.item()
            summ.append(summary_batch)

            output_batch = np.argmax(output_batch, axis=1)

            outputs_epoch = np.concatenate((outputs_epoch, output_batch))
            labels_epoch = np.concatenate((labels_epoch, labels_batch))

        # mettre à jour la perte moyenne
        loss_avg.update(loss.item())
        t.set_postfix(loss='{:05.3f}'.format(loss_avg()))


    # calculer la moyenne de toutes les métriques dans le résumé
    metrics_mean = {metric: np.mean([x[metric]
                                     for x in summ]) for metric in summ[0]}
    metrics_string = " ; ".join("{}: {:05.3f}".format(k, v)
                                for k, v in metrics_mean.items())
    logging.info("- Métriques d'entraînement : " + metrics_string)


def train_and_evaluate(model, train_data, val_data, optimizer, loss_fn, metrics, params, model_dir,
restore_file=None):
    """Entraîne le modèle et évalue à chaque époque.

    Args:
        model: (torch.nn.Module) le réseau neuronal
        train_data: (dict) données d'entraînement avec les clés 'data' et 'labels'
        val_data: (dict) données de validation avec les clés 'data' et 'labels'
```

```
        optimizer: (torch.optim) optimiseur pour les paramètres du modèle
        loss_fn: une fonction prenant batch_output et batch_labels et calculant la perte pour le lot
        metrics: (dict) un dictionnaire de fonctions qui calculent une métrique en utilisant la
sortie et les étiquettes de chaque lot
        params: (Params) hyperparamètres
        model_dir: (string) répertoire contenant la configuration, les poids et les journaux
        restore_file: (string) optionnel- nom du fichier à restaurer (sans son extension .pth.tar)
    """
    # charger à nouveau les poids à partir de restore_file si spécifié
    if restore_file is not None:
        restore_path = os.path.join(
            args.model_dir, args.restore_file + '.pth.tar')
        logging.info("Restauration des paramètres à partir de {}".format(restore_path))
        utils.load_checkpoint(restore_path, model, optimizer)

    best_val_acc = 0.0

    for epoch in range(params.num_epochs):
        # Effectuer une époque
        logging.info("Époque {}/{}".format(epoch + 1, params.num_epochs))

        # calculer le nombre de lots dans une époque (un passage complet sur l'ensemble
d'entraînement)
        num_steps = (params.train_size + 1) // params.batch_size
        train_data_iterator = data_loader.data_iterator(
            train_data, params)
        train(model, optimizer, loss_fn, train_data_iterator,
              metrics, params, num_steps)

        # Évaluer pendant une époque sur l'ensemble de validation
        num_steps = (params.val_size + 1) // params.batch_size
        val_data_iterator = data_loader.data_iterator(
            val_data, params, shuffle=False)
        val_metrics = evaluate(
            model, loss_fn, val_data_iterator, metrics, params, num_steps)

        val_acc = val_metrics['accuracy']
        is_best = val_acc >= best_val_acc

        # Sauvegarder les poids
        utils.save_checkpoint({'epoch': epoch + 1,
                               'state_dict': model.state_dict(),
```

```
                                'optim_dict': optimizer.state_dict()},
                               is_best=is_best,
                               checkpoint=model_dir)

        # Si best_eval, best_save_path
        if is_best:
            logging.info("- Nouvelle meilleure précision trouvée")
            best_val_acc = val_acc

            # Sauvegarder les meilleures métriques de validation dans un fichier JSON dans le
répertoire du modèle
            best_json_path = os.path.join(
                model_dir, "metrics_val_best_weights.json")
            utils.save_dict_to_json(val_metrics, best_json_path)

        # Sauvegarder les dernières métriques de validation dans un fichier JSON dans le répertoire
du modèle
        last_json_path = os.path.join(
            model_dir, "metrics_val_last_weights.json")
        utils.save_dict_to_json(val_metrics, last_json_path)


if __name__ == '__main__':

    # Charger les paramètres à partir du fichier JSON
    args = parser.parse_args()
    json_path = os.path.join(args.model_dir, 'params.json')
    assert os.path.isfile(
        json_path), "Aucun fichier de configuration JSON trouvé à {}".format(json_path)
    params = utils.Params(json_path)

    # utiliser le GPU s'il est disponible
    params.cuda = torch.cuda.is_available()

    # Fixer la graine aléatoire pour des expériences reproductibles
    torch.manual_seed(230)
    if params.cuda:
        torch.cuda.manual_seed(230)

    # Fixer le journal
    utils.set_logger(os.path.join(args.model_dir, 'train.log'))
```

```
    # Créer le pipeline de données d'entrée
    logging.info("Chargement des ensembles de données...")

    # charger les données
    data_loader = DataLoader(args.data_dir, params)
    data = data_loader.load_data(['train', 'val'], args.data_dir)
    train_data = data['train']
    val_data = data['val']

    # spécifier les tailles des ensembles d'entraînement et de validation
    params.train_size = train_data['size']
    params.val_size = val_data['size']

    logging.info("- Terminé.")

    # Définir le modèle et l'optimiseur
    model = net.Net(params).cuda() if params.cuda else net.Net(params)
    optimizer = optim.Adam(model.parameters(), lr=params.learning_rate)

    # obtenir la fonction de perte et les métriques
    loss_fn = net.loss_fn
    metrics = net.metrics

    # Entraîner le modèle
    logging.info("Début de l'entraînement pour {} époque(s)".format(params.num_epochs))
    train_and_evaluate(model, train_data, val_data, optimizer, loss_fn, metrics, params,
args.model_dir,
                       args.restore_file)
```

```
import json
import logging
import os
import shutil
from sklearn.metrics import recall_score
from torchmetrics.classification import MulticlassF1Score, MulticlassRecall
import torch
import numpy as np


class Params():
    """Classe qui charge les hyperparamètres à partir d'un fichier JSON.
```

```
    Exemple :
    ```
    params = Params(json_path)
    print(params.learning_rate)
    params.learning_rate = 0.5 # changer la valeur de learning_rate dans params
    ```
    """

    def __init__(self, json_path):
        with open(json_path) as f:
            params = json.load(f)
            self.__dict__.update(params)

    def save(self, json_path):
        with open(json_path, 'w') as f:
            json.dump(self.__dict__, f, indent=4)

    def update(self, json_path):
        """Charge les paramètres à partir d'un fichier JSON"""
        with open(json_path) as f:
            params = json.load(f)
            self.__dict__.update(params)

    @property
    def dict(self):
        """Donne un accès de type dictionnaire à l'instance Params en utilisant
`params.dict['learning_rate']`"""
        return self.__dict__


class RunningAverage():
    """Une classe simple qui maintient la moyenne mobile d'une quantité.

    Exemple :
    ```
    loss_avg = RunningAverage()
    loss_avg.update(2)
    loss_avg.update(4)
    loss_avg() = 3
    ```
    """
```

```
    def __init__(self):
        self.steps = 0
        self.total = 0

    def update(self, val):
        self.total += val
        self.steps += 1

    def __call__(self):
        return self.total / float(self.steps)


def set_logger(log_path):
    """Définit le logger pour enregistrer les informations dans le terminal et le fichier
`log_path`.

    En général, il est utile d'avoir un logger pour enregistrer toutes les sorties du terminal dans
un fichier permanent.
    Ici, nous l'enregistrons dans `model_dir/train.log`.

    Exemple :
    ```
    logging.info("Démarrage de l'entraînement...")
    ```

    Args:
        log_path: (string) chemin où enregistrer les logs
    """
    logger = logging.getLogger()
    logger.setLevel(logging.INFO)

    if not logger.handlers:
        # Enregistrement dans un fichier
        file_handler = logging.FileHandler(log_path)
        file_handler.setFormatter(logging.Formatter('%(asctime)s:%(levelname)s: %(message)s'))
        logger.addHandler(file_handler)

        # Enregistrement dans la console
        stream_handler = logging.StreamHandler()
        stream_handler.setFormatter(logging.Formatter('%(message)s'))
        logger.addHandler(stream_handler)
```

```
def save_dict_to_json(d, json_path):
    """Enregistre un dictionnaire de nombres en JSON.

    Args:
        d: (dict) contient des valeurs numériques pouvant être transformées en float (np.float, int,
float, etc.)
        json_path: (string) chemin du fichier JSON
    """
    with open(json_path, 'w') as f:
        # Nous devons convertir les valeurs en float pour le JSON (il n'accepte pas np.array,
np.float, etc.)
        d = {k: float(v) for k, v in d.items()}
        json.dump(d, f, indent=4)


def save_checkpoint(state, is_best, checkpoint):
    """Enregistre le modèle et les paramètres d'entraînement à l'emplacement checkpoint +
'last.pth.tar'.
    Si is_best==True, enregistre également à l'emplacement checkpoint + 'best.pth.tar'

    Args:
        state: (dict) contient le state_dict du modèle, peut contenir d'autres clés telles que
l'époque, le state_dict de l'optimiseur, etc.
        is_best: (bool) True si c'est le meilleur modèle jusqu'à présent
        checkpoint: (string) dossier où sauvegarder les paramètres
    """
    filepath = os.path.join(checkpoint, 'last.pth.tar')
    if not os.path.exists(checkpoint):
        print("Le répertoire de sauvegarde n'existe pas! Création du répertoire
{}".format(checkpoint))
        os.mkdir(checkpoint)
    else:
        print("Le répertoire de sauvegarde existe! ")
    torch.save(state, filepath)
    if is_best:
        shutil.copyfile(filepath, os.path.join(checkpoint, 'best.pth.tar'))


def load_checkpoint(checkpoint, model, optimizer=None):
    """Charge les paramètres du modèle (state_dict) à partir du fichier checkpoint.
```

```
    Args:
        checkpoint: (string) nom du fichier à charger
        model: (torch.nn.Module) modèle pour lequel les paramètres sont chargés
        optimizer: (torch.optim) facultatif : optimiseur à reprendre depuis le fichier checkpoint
    """
    if not os.path.exists(checkpoint):
        raise ("Le fichier n'existe pas {}".format(checkpoint))
    checkpoint = torch.load(checkpoint)
    model.load_state_dict(checkpoint['state_dict'])

    if optimizer:
        optimizer.load_state_dict(checkpoint['optim_dict'])

    return checkpoint


def recall(outputs, labels):
    """Calcule le rappel (recall) des prédictions du modèle par rapport aux étiquettes réelles.

    Args:
        outputs: (np.array) prédictions du modèle
        labels: (np.array) étiquettes réelles

    Returns:
        recall: (float) le rappel calculé
    """
    outputs_tensor = torch.tensor(outputs)
    labels_tensor = torch.tensor(labels)
    metric = MulticlassRecall(num_classes=3)
    recall = metric(outputs_tensor, labels_tensor)

    return recall


def f1_score(outputs, labels):
    """Calcule le score F1 des prédictions du modèle par rapport aux étiquettes réelles.

    Args:
        outputs: (np.array) prédictions du modèle
        labels: (np.array) étiquettes réelles

    Returns:
```

```
        f1_score: (float) le score F1 calculé
    """
    outputs_tensor = torch.from_numpy(outputs)
    labels_tensor = torch.from_numpy(labels)
    metric = MulticlassF1Score(num_classes=3)
    f1_score = metric(outputs_tensor, labels_tensor)

    return f1_score


!pip install transformers
!pip install huggingface-hub
!pip install huggingsound
!pip install pydub

from transformers import BertForSequenceClassification, BertTokenizer
from huggingsound import SpeechRecognitionModel
from IPython.display import display, Javascript
from google.colab import output
from base64 import b64decode
from io import BytesIO
from pydub import AudioSegment

# Charger le modèle de transcription wav2vec
wav2vec_model = SpeechRecognitionModel("jonatasgrosman/wav2vec2-large-xlsr-53-english")

# Enregistrer l'audio depuis le microphone
def record(sec=10):
    print("")
    print("Parlez maintenant...")
    display(Javascript(RECORD))
    s = output.eval_js('record(%d)' % (sec*1000))
    print("Enregistrement terminé !")
    b = b64decode(s.split(',')[1])
    with open('audio.wav', 'wb') as f:
        f.write(b)
    return 'audio.wav'

# Transcription de l'audio
audio = record()
transcriptions = wav2vec_model.transcribe([audio])
transcription = transcriptions[0]
```

```
transcription_text = transcription['transcription'].strip()

# Phase 1 : Classification basée sur les mots-clés
keywords = {
    "Hotel": ["HOTEL", "HOSTEL", "ROOM", "LODGING", "ACCOMMODATION"],
    "Event": ["EVENT", "FESTIVAL", "CONCERT", "CELEBRATION", "GATHERING"],
    "User-based": ["USER BASED", "USER-BASED", "USERBASED", "USER RECOMMENDATION", "USER
PREFERENCE"],
    "Item-based": ["ITEM BASED", "ITEM-BASED", "ITEMBASED", "ITEM RECOMMENDATION", "ITEM
PREFERENCE"]}

# Fonction pour la classification basée sur les mots-clés
def classify_by_keywords(text, keywords):
    predicted_classes = []
    for label, keyword_list in keywords.items():
        for keyword in keyword_list:
            if keyword in text.upper() and label not in predicted_classes:
                predicted_classes.append(label)
    return predicted_classes

predicted_classes_first_phase = classify_by_keywords(transcription_text, keywords)

# Phase 2 : Classification en utilisant le classificateur chargé de détecter l'objet de la
recommandation
second_phase_model_path = "recommender_class_model.pth.tar"
tokenizer_second_phase_path = "recommender_class_model.pth.tar"
second_phase_classification_model =
BertForSequenceClassification.from_pretrained(second_phase_model_path,
num_labels=num_classes_second_phase)
tokenizer_second_phase = BertTokenizer.from_pretrained(tokenizer_second_phase_path)

# Phase 3 : Classification en utilisant le classificateur chargé de détecter le type de
recommandation
third_phase_model_path = "recommender_type_model.pth.tar"
tokenizer_third_phase_path = "recommender_type_model.pth.tar"
third_phase_classification_model =
BertForSequenceClassification.from_pretrained(third_phase_model_path,
num_labels=num_classes_third_phase)
tokenizer_third_phase = BertTokenizer.from_pretrained(tokenizer_third_phase_path)

# Phase 2 : Classification en utilisant le premier classificateur
```

```
inputs_second_phase = tokenizer_second_phase(transcription_text, return_tensors="pt", padding=True,
truncation=True)
predictions_second_phase = second_phase_classification_model(**inputs_second_phase)
predicted_class_index_second_phase = predictions_second_phase.logits.argmax(dim=1).item()
predicted_class_second_phase =
train_df_second_phase['class'].unique()[predicted_class_index_second_phase]

# Phase 3 : Classification en utilisant le deuxième classificateur
inputs_third_phase = tokenizer_third_phase(transcription_text, return_tensors="pt", padding=True,
truncation=True)
predictions_third_phase = third_phase_classification_model(**inputs_third_phase)
predicted_class_index_third_phase = predictions_third_phase.logits.argmax(dim=1).item()
predicted_class_third_phase =
train_df_third_phase['class'].unique()[predicted_class_index_third_phase]

# Afficher la transcription et les prédictions des trois phases
print("Transcription:", transcription_text)
print("Classification basée sur les mots clés :", predicted_classes_first_phase)
print("Classification de l'objet de la recommandation :", predicted_class_second_phase)
print("Classification du type de recommandation:", predicted_class_third_phase)
```