

Table des matières

1	Introduction générale	3
2	Assistants virtuels intelligents	4
2.1	Introduction	4
2.2	L'importance du contexte pour un SPA	6
2.3	Caractéristiques principales d'un SPA	6
2.3.1	Sensible au contexte	6
2.3.2	Évolutif	7
2.3.3	Multimodal	7
2.3.4	Anthropomorphe	7
2.3.5	Multi-plateforme et Flexible	8
2.4	Domaines d'applications des SPAs	8
2.4.1	Vie quotidienne	8
2.4.2	Assistance professionnelle	9
2.4.3	E-Apprentissage	9
2.5	Exemples de SPAs	9
2.5.1	Google assistant	9
2.5.2	Apple Siri	11
2.5.3	Amazon Alexa	12
2.5.4	Microsoft Cortana	13
2.6	Conclusion	13
3	Composants de base d'un assistant personnel	14
3.1	Introduction	14
3.2	Notions et aspects théoriques	14
3.2.1	Apprentissage automatique	14
3.2.2	Réseaux de neurones artificiels	15
3.3	Schéma global d'un SPA	21
3.4	Reconnaissance automatique de la parole (ASR)	22
3.4.1	Acquisition du signal et extraction d'attributs	23
3.4.2	Modélisation acoustique et modélisation du lexique	23
3.4.3	Modélisation de la langue	23
3.5	Compréhension du langage naturel (NLU)	23
3.5.1	Classification d'intentions	24
3.5.2	Extraction d'entités	25
3.5.3	Analyse sémantique	26

3.6	Gestion du dialogue	26
3.6.1	Processus de décision Markovien (MDP)	27
3.6.2	État du gestionnaire de dialogue	28
3.6.3	Politique de gestion de dialogue	30
3.7	Génération du langage naturel (NLG)	33
3.7.1	Détermination du contenu	33
3.7.2	Structuration de texte	33
3.7.3	Agrégation de phrases	34
3.7.4	Lexicalisation	34
3.7.5	Génération d'expressions référentielles (REG)	34
3.7.6	Réalisation linguistique	34
3.7.7	Systèmes basés encodeur-décodeur	36
3.8	Conclusion	37
4	Conception du système	38
4.1	Introduction	38
4.2	Architecture du système	38
4.2.1	Couche utilisateur	39
4.2.2	Couche système	39
4.3	Module de reconnaissance automatique de la parole	40
4.3.1	Architecture du module	40
4.3.2	Modèle acoustique	40
4.3.3	Modèle de la langue	42
4.4	Module de compréhension automatique du langage naturel	44
4.4.1	Architecture du module	45
4.4.2	Analyse sémantique avec apprentissage automatique	46
4.5	Module de gestion du dialogue	47
4.5.1	Architecture du module	47
4.5.2	Les ontologies du système	50
4.5.3	Les simulateurs d'utilisateurs	54
4.5.4	Modèles d'apprentissage	57
4.6	Module de génération du langage naturel	61
4.7	Conclusion	62

To correct

To explain more deeply

Chapitre 1

Introduction générale

- Ici on parlera des motivations qui ont aboutis à ce projet, des objectifs de ce dernier ainsi que ses perspectives

Chapitre 2

Assistants virtuels intelligents

2.1 Introduction

Depuis la commercialisation du premier ordinateur grand public (Xerox PARC Alto) en 1973, le monde découvrit pour la première fois ce qui allait devenir l'apparence basique de chaque ordinateur moderne. En effet, la compagnie Xerox fut la première à proposer une interface graphique dotée de fenêtres, d'icônes et d'une souris pour se déplacer et d'un clavier pour écrire du texte. Bien que basique, cette idée lança alors plusieurs autres grandes marques sur le même chemin (IBM, Apple, Compaq ...). Par la suite, beaucoup ont essayé d'améliorer la façon dont l'homme utilisait sa machine : souris plus précise, écran doté d'une plus grande résolution, clavier plus enrichi, voire même l'introduction des écrans tactiles dans certains systèmes embarqués.

Cependant, certains voyaient encore cette façon d'utiliser la machine comme trop primitive, et peu intuitive. En effet laissez un enfant devant un ordinateur et il prendrait un bon moment pour apprendre à éditer ne serait ce qu'un simple fichier. Pour citer Donald A. Norman :

"We must design for the way people behave, not for how we would wish them to behave."^[1]

que nous pouvons traduire par :

"Nous devons concevoir selon le comportement des utilisateurs, et non pas selon la façon dont nous voudrions qu'ils se comportent."

L'humanité a fait beaucoup de chemin depuis les années 70, l'utilisation d'un ordinateur de nos jours avec les moyens classiques (souris, clavier, écran ...) est devenue une tâche triviale, voire même une **seconde nature, cela reste cependant dû au fait que de plus en plus de jeunes enfants sont exposés depuis leur plus jeune âge au monde technologique qui les entoure, le processus d'apprentissage reste cependant présent, l'effort d'utiliser les outils communs reste lui aussi présent.**

La plus naturelle et plus ancienne façon de communiquer pour l'homme a toujours été la parole. Le développement de langues toutes aussi riches et complexes les unes que les autres a permis à l'humanité de briser plusieurs **barrières sociales**. L'avancement le

plus naturel pour cette façon de communiquer serait donc de l'étendre aux machines que l'homme a su construire et améliorer au fil des années.

Motivé par cette manière que l'on a de communiquer entre nous, et épaulé par les récentes technologies telles que l'apprentissage automatique, le traitement automatique du langage naturel et l'intelligence artificielle, les plus brillants des chercheurs ont entamé leurs travaux dans cette toute nouvelle direction.

Les Assistants Virtuels Intelligents (Smart Personal Assistant, SPA [2]) sont donc le produit de plusieurs années de recherche, visant tout d'abord à faciliter certaines tâches pour l'utilisateur. Les premiers SPAs étaient conçus comme des agents de conversation ou Chatbots, limités dans leurs actions et dépendant toujours d'un moyen de communication textuel, ce n'était pas la forme désirée du SPA. Avec l'avancement des recherches sur la reconnaissance automatique de la parole (Automatic Speech Recognition, ASR) et l'émergence de l'apprentissage automatique, les tout premiers assistants virtuels utilisant l'ASR étaient spécialisés dans certains domaines comme des systèmes médicaux d'aide à la décision. Il a ensuite été plus aisé de briser la barrière et de réaliser ce qui était encore une esquisse d'un SPA personnalisé. Aujourd'hui, et ce depuis l'avènement de l'apprentissage profond et la popularisation des Smartphones, de nouveaux SPAs comme Apple Siri (voir 2.5.2) et Google assistant (voir 2.5.1) et Amazon Alexa (voir 2.5.3) ont fait leurs apparitions, offrant de plus en plus de services personnalisés et spécifiques à chaque utilisateurs.

Dans la suite de ce chapitre nous essayerons de mieux détailler ce qu'est un SPA, ce qui est demandé d'un tel système, ses domaines d'application, en enchaînant par une description d'une pseudo-architecture potentielle de ce système, pour enfin conclure sur les limitations actuelles et les motivations de ce projet.

2.2 L'importance du contexte pour un SPA

Informellement, un SPA est un type d'agent (voir 2.3.2) logiciel qui peut effectuer certaines tâches et proposer des services dédiés aux utilisateurs qui vont d'une simple tâche (Ouvrir une fenêtre, lancer une application ...) à la réalisation de requêtes un peu plus complexes comme réserver une table dans un restaurant en passant un appel vocal (voir 2.5.1.1). Pour répondre efficacement à toutes sortes de requêtes, un SPA se doit donc de garder trace du contexte courant de sa conversation avec l'utilisateur. Il doit disposer d'un système capable d'enregistrer les informations pertinentes et de savoir les réutiliser, mais aussi de pouvoir déduire lesquelles de ces informations sont manquantes. On parle ici de Context-Awareness ou Sensibilité au contexte, comme vu dans [2].

D'après [2] et [3], *Day* et *Abwod* définissent un contexte comme suit :

"A context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves"

qui peut être traduit par :

"Un contexte est une information qui peut être utilisé pour caractériser l'état d'une entité. Une entité peut être une personne une place ou un objet, considérée comme pertinente à l'interaction entre l'utilisateur et l'application, ainsi qu'à ces deux derniers eux mêmes"

Il en découle que pour parvenir à développer un système qui puisse répondre aux besoins individuels et spécifiques de chaque personne, modéliser et prendre en compte le contexte semble être une solution prometteuse.

2.3 Caractéristiques principales d'un SPA

À partir de [2], nous pouvons dégager certaines caractéristiques principales qui peuvent être vues comme primordiales pour qualifier un assistant virtuel comme étant intelligent.

2.3.1 Sensible au contexte

Comme précédemment vu dans la définition du contexte (section 2.2), ce dernier peut être interprété comme tout aspect d'une entité (position d'un objet, couleur d'un objet, température d'une chambre, etc.). Un assistant dit intelligent doit donc être capable de capturer le concept du contexte, d'utiliser et de traiter toute information catégorisée comme contextuelle. Pour être plus précis, un SPA doit être sensible à l'évolution du contexte courant, par le biais de capteurs optiques, de microphones, ou tout ce qui pourrait amener l'utilisateur à faire évoluer la requête qu'il a émise. L'assistant devra donc proposer un système de mise à jour du contexte pour éliminer les informations inutiles et garder celles qui pourraient aider à répondre à la requête de l'utilisateur.

2.3.2 Évolutif

Comme vu dans la section 2.2, un SPA peut être vu comme un type d'agent. Pour rappel, d'après Russel et Norvig dans [4], un agent est une entité autonome pouvant interagir avec son environnement afin d'accomplir certaines tâches et peut être de plusieurs types :

- Agent à réflexes simples : agent exécutant ses actions à base de règles conditionnelles simples (c.à.d Si *Condition* alors exécuter *actions*), ils sont ainsi très simplistes et limités dans la portée de leurs actions.
- Agent basé modèle : semblable aux agents à réflexes simples, il est doté d'un modèle interne complexe censé représenter le monde extérieur auquel l'agent a accès. Cependant, il applique les actions de la même manière que le précédent type d'agents.
- Agent à but : ce type représente une amélioration des agents simples puisqu'il est doté d'un ensemble d'états buts à atteindre d'une façon ou d'une autre.
- Agent à utilité : il s'agit ici d'agents à buts qui tentent d'aboutir à leurs buts d'une manière optimisée (intelligente) utilisant une fonction de mesure adéquate pour le choix des différents états à atteindre.
- Agent apprenant : agent à utilité enrichi par un module d'apprentissage qui sert de juge pour répondre aux "critiques" des actions qu'il entreprend. Le terme agent évolutif est aussi employé.

Pour ce qui est des SPAs, les plus récents systèmes (ex : Amazon Alexa qui améliore son module de reconnaissance de la parole après chaque réponse non [réfutée](#) par l'utilisateur) peuvent être considérés comme des agents apprenants, répondant de ce fait à la contrainte évolutive imposée. Cependant, le domaine de l'auto-évolution des systèmes intelligents est encore un domaine nouveau qui se voit [aidé](#) par les récentes avancées dans l'apprentissage automatique [2].

2.3.3 Multimodal

Afin d'assurer une aisance d'utilisation, les SPAs sont fréquemment amenés à récupérer les requêtes (ou données) en entrée de la manière la plus naturelle possible (par exemple par le biais de la parole). Cependant, pour garantir une expérience d'utilisation adéquate, l'assistant sera souvent confronté à récupérer ces requêtes de différentes manières, que ce soit à travers une interface graphique (écran tactile) ou à travers un texte brut tapé au clavier, voire même à travers des expressions faciales ou des états cognitifs/émotionnels [5], pour ensuite produire une réponse qui elle aussi pourrait éventuellement être de la forme textuelle, sonore ou les deux. Cette capacité à recevoir en entrée et/ou produire une sortie de plusieurs façons différentes est appelée la multi-modalité [6]. Cette caractéristique permet de masquer à l'utilisateur toute la complexité d'acquisition de ses requêtes.

2.3.4 Anthropomorphe

Plusieurs auteurs tendent à attribuer une grande importance à l'anthropomorphisme des SPAs [7], qui est

"Un mécanisme qui pousse les êtres humains à induire qu'une entité non-humanoïde possède des caractéristiques et comportements propres à l'homme"[8]

Ce comportement humanoïde pousserait donc l'utilisateur à se sentir plus à l'aise avec l'assistant, le conduisant ainsi à adopter une façon de communiquer plus humaine et moins structurée qu'avec les autres machines. Ceci est une caractéristique majeure d'un SPA se disant personnalisé.

2.3.5 Multi-plateforme et Flexible

Malgré leurs récentes prouesses, certains SPAs sont encore restreints à un écosystème fortement dépendant du fabricant. Cowan et al. mentionnent dans [9] que Apple Siri est limité à l'environnement constitué des produits de la firme à la pomme, n'ouvrant par défaut que les applications de cette dernière quand une requête lui est transmise. C'est un comportement que les assistants devraient éviter, car une indépendance des plateformes utilisées est, certes, très complexe à instaurer, mais offre plus de possibilités aux utilisateurs et aux développeurs pouvant ainsi exploiter la puissance de certaines plateformes (Smartphones, TV connectées, etc). Avec l'émergence de l'IoT (Internet of Things) et des maisons intelligentes par exemple, c'est un tout nouveau terrain de jeu qui est présenté aux SPAs, offrant plus d'opportunités pour les utilisateurs.

2.4 Domaines d'applications des SPAs

Après avoir vu les différents aspects que les SPAS doivent traiter, nous nous intéresserons maintenant aux types de services et applications que ces derniers pourraient fournir pour démontrer qu'ils peuvent bel et bien faciliter certaines tâches à l'homme.

2.4.1 Vie quotidienne

À la base, les SPAs étaient destinés à un usage très personnel comme la gestion des achats dans les supermarchés, ou des guides touristiques de plusieurs destinations de voyage. Cette spécificité a commencé à s'estomper petit à petit avec l'émergence de nouveaux systèmes dédiés à des applications plus générales, comme les maisons intelligentes ou les assistants de planification de tâches. Ceci a permis de mettre encore plus l'accent sur cet aspect de convivialité que les tout premiers SPAs ont tenté de perfectionner. Ainsi, ces assistants spécialisés dans des domaines restreints (Tourisme, shopping, détente, etc) ont été regroupés dans un seul système plus polyvalent, capable de répondre à des besoins quotidiens divers et variés, allant même à fournir une assistance aux personnes âgées pour leur faciliter les tâches rudimentaires devenues trop fatigantes.

2.4.2 Assistance professionnelle

Les SPAs ont aussi une place dans le monde professionnel. Dans [10] il est cité que dans les situations où la marge d'erreur est très petite (par exemple dans les système de manufacturing¹) l'assistance d'un SPA est nécessaire, servant d'un aide à l'humain pour la prise de décision.

Par exemple, dans un environnement de travail hétérogène (Nouveaux/anciens employés, Hiérarchies des postes ...) les SPAs pourraient décharger les employés les plus expérimentés de la tâche d'assister les nouveaux arrivants, pour ainsi aider ces derniers dans leurs tâches et permettre aux autres de se focaliser sur les leurs.

2.4.3 E-Apprentissage

Les SPAs peuvent aussi être utiles dans l'enseignement, aussi bien dans un milieu académique que professionnel. D'une part ils pourraient occuper plusieurs rôles dans les établissements scolaires (correcteur automatique de copies, enseignant interactif ...) [11] et, d'autre part, accompagner les employés durant leurs formations professionnelles.

Ainsi, en considérant les caractéristiques d'un SPA, la sensibilité au contexte est reliée aux expériences antérieures de l'apprenant, permettant au SPA d'adapter son processus d'enseignement en conséquence.

En ce qui concerne l'aspect évolutif du SPA, il lui permet de préférer une approche d'enseignement à une autre selon les résultats de ses apprenants.

2.5 Exemples de SPAs

Pour illustrer la puissance des SPAs les plus récents, nous présentons dans cette section les quatre produits qui dominent le marché courant :

2.5.1 Google assistant

Lancé en 2016 sous forme d'un chatbot intégré dans l'application Google Allo, Google Assistant s'est vu ensuite être directement intégré sur les système d'exploitation Android (que ce soit sur smartphones ou tablettes, et plus récemment sur Google Home²). Google Assistant est un assistant à tout faire qui a été développé par les ingénieurs de Google dans le but de faciliter la recherche sur internet, la planification des tâches, l'ajustement des réglages de l'appareil, etc. Son point fort est sa capacité à engager une conversation bidirectionnelle avec l'utilisateur, assurant ainsi une interaction personnalisée variant d'un utilisateur à un autre. Cette capacité lui permet par exemple de proposer certains résultats

1. Manufacturing ici dans le sens chaîne de montage industrielle, par exemple dans des usines.

2. Appareil servant à contrôler les composants d'une smart-house ainsi que l'utilisation des différents services de Google

de recherche selon les précédentes interactions avec l'utilisateur ou de lui proposer une activité si ce dernier lui mentionne qu'il s'ennuie (voir figure 2.1).

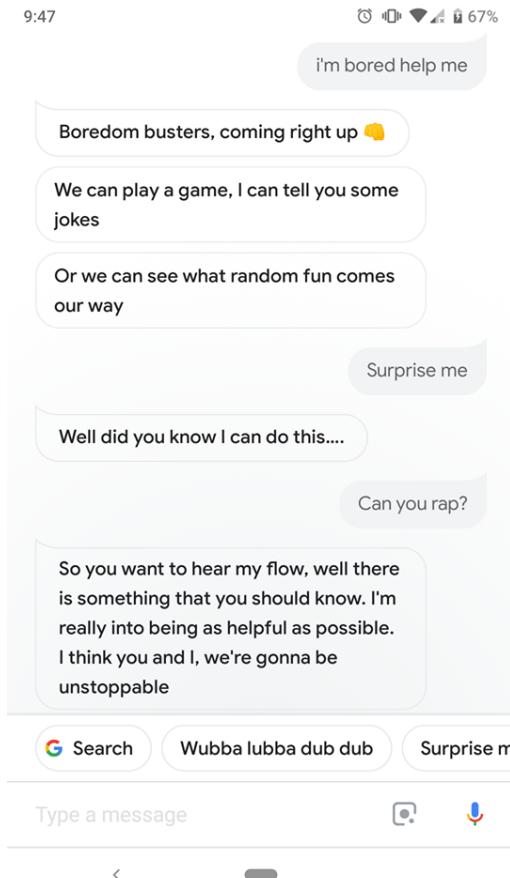


Figure 2.1 – Conversation aléatoire avec Google Assistant

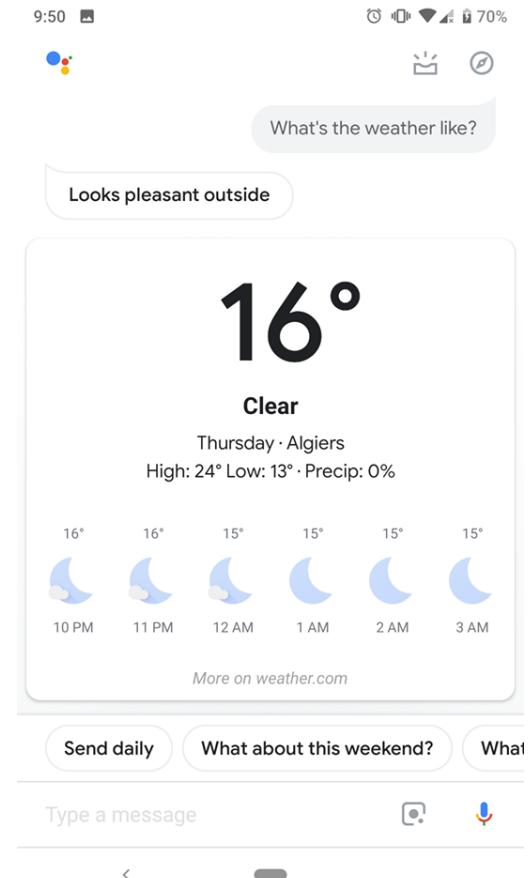


Figure 2.2 – Requête simple formulée à Google Assistant

2.5.1.1 Google duplex

Une des nouveautés impressionnante de Google Assistant est la fonctionnalité Google Duplex. Toujours en phase de développement, ce module est capable de passer des appels à de vraies personnes et d'avoir une conversation avec elles afin de réaliser une tâche demandée par l'utilisateur comme par exemple réserver une chambre d'hôtel, une table au restaurant, etc.



Figure 2.3 – *Google duplex réservant une place dans un salon de coiffure*

2.5.2 Apple Siri

Siri est l'assistant virtuel développé par Apple. Contrairement aux SPAs durant sa sortie, Siri proposait une nouvelle façon de communiquer avec l'utilisateur, à travers une interface de requêtes vocales, et une façon de **converser** très humanoïde (satisfaisant ainsi le critère d'anthropomorphisme 2.3.4). Siri est capable de répondre à des questions précises (voir figure 2.6), de proposer des recommandations, déléguer la requête à des services web ou d'autres applications (voir figures 2.4 et 2.5). Il a l'avantage (et l'inconvénient) d'être uniquement disponible que sur les appareils qui composent l'écosystème d'Apple (MacBook, iPhone, iWatch, etc.).



Figure 2.4 – *Intégration aux applications [12]*

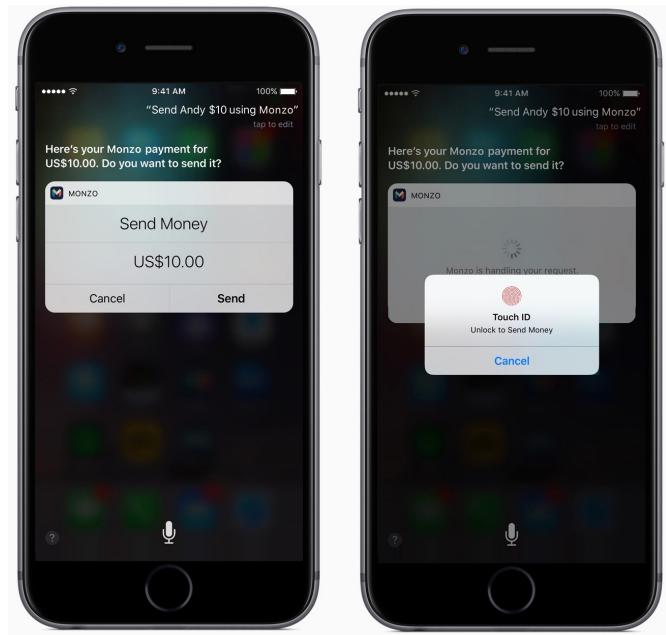


Figure 2.5 – Service paiement 1 [12]

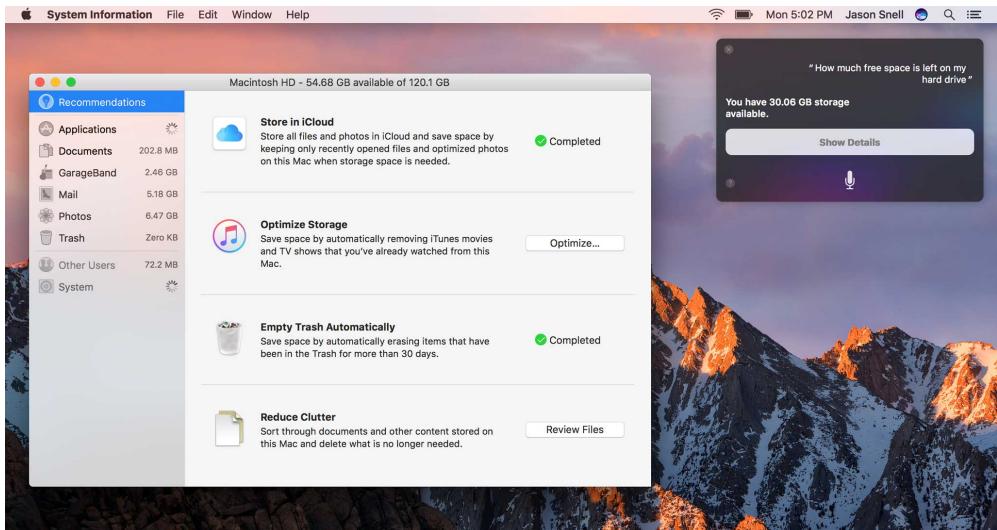


Figure 2.6 – Siri sur un laptop [13]

2.5.3 Amazon Alexa

Amazon Alexa est un assistant exclusivement intégré au dispositif Amazon Echo (un haut-parleur portatif). À l'instar de Siri, il est aussi capable de communiquer avec l'utilisateur par le biais de la parole, pouvant ainsi exécuter diverse commandes comme jouer de la musique, réciter des livres audios, annoncer des news en temps réel (Résultats sportifs, tendances politiques, etc). Son atout majeur est sa capacité à s'intégrer à plusieurs appareils-connectés (Contrôleur de thermostat ou de lumières ambiantes dans une Smart-House) ainsi que la possibilité d'ajout des Skills (ou compétences) de la part des développeurs tiers pour enrichir la panoplie de services que peut offrir Alexa.

2.5.4 Microsoft Cortana

Cortana est la tentative de la part de Microsoft d'intégrer un assistant dans son système d'exploitation Windows 10 et WindowsPhone. Il propose divers services de base tel que planifier des tâches, exécuter des commandes via la parole, et analyser des résultats de recherche sur le moteur de recherche de Microsoft, Bing, pour répondre à des questions.

2.6 Conclusion

À travers les sections précédentes, nous avons essayé de présenter les différents aspects d'un assistant virtuel intelligent (caractéristiques, exemples, architectures possibles, etc). Nous avons donc pu apprécier la potentielle puissance d'un tel système s'il venait à être perfectionner d'avantage.

En effet, en examinant les domaines d'applications, il est facile de déduire que le recours à un SPA peut grandement faciliter certaines tâches, que ce soit celles qui sont les plus triviales pouvant retarder d'autres tâches plus importantes, ou bien celles qui doivent faire appel à la précision et à la grande capacité de calcul des machines, assurant ainsi des résultats précis et rapidement délivrés.

À la fin de ce chapitre nous pouvons donc mettre en valeur la place primordiale que pourraient avoir les SPAs s'ils arrivaient à maturité, c.à.d briser la barrière qui sépare les humains de la machine, parvenant ainsi à faire partie de la vie quotidienne des utilisateurs.

Dans le prochain chapitre nous allons principalement aborder les aspects techniques des différents composants du SPA que nous désirons réaliser.

Chapitre 3

Composants de base d'un assistant personnel

3.1 Introduction

Durant ce chapitre, nous allons détailler un peu plus l'aspect technique d'une architecture typique pour un SPA, nous commencerons d'abord par définir des notions de base ainsi que des techniques d'apprentissage automatique, nous traiterons celles qui ont été les plus utilisées dans les travaux que nous avons examiné. La suite du chapitre sera organisé en sections qui décriront chacune le fonctionnement d'une partie du SPA, en citant les travaux et références qui relatent de cette dernière. Nous terminerons sur une conclusion qui introduira le chapitre suivant.

3.2 Notions et aspects théoriques

Dans cette section nous essaierons de présenter différentes notions liées au domaine de l'apprentissage automatique, pour ensuite les citer dans les modules qui les utilisent.

3.2.1 Apprentissage automatique

Le plus souvent, un domaine scientifique peut être défini à travers le, ou les types de problèmes dont il essaye de trouver une solution, d'après [14], l'auteur a défini ce problème sous forme d'une question :

"How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes?"[14]

que nous pouvons traduire par :

"Comment pourrions nous développer un système informatique qui pourrait s'améliorer à travers l'expérience, et quelles seraient les lois qui régiraient le processus d'apprentissage ?"

D'après [14] l'apprentissage automatique est un paradigme qui stipule qu'un système informatique peut apprendre à effectuer un ensemble de tâches T , sachant qu'il dispose d'un ensemble de données E , tout en améliorant sa performance P .

Il existe plusieurs sous catégories d'apprentissage automatique, elles diffèrent principalement par la manière dont le système apprend, du type de données sur les quelles il apprend, ainsi que du but de son apprentissage (classification, régression,...). Nous pouvons citer les catégories suivantes :

- **Apprentissage supervisé** : Les algorithmes de cette catégories ont besoin d'une assistance externe, les données doivent être séparées en deux parties (ensemble d'apprentissage et de test), un *label* ou classe doit être associée à chaque instance pour permettre à l'algorithme de calculer un certain taux d'erreur qu'il essayera de minimiser au fur et à mesure qu'une nouvelle instance lui est présentée [15]. Idéalement le système pourra apprendre une certaines fonction $\hat{f} : X \rightarrow Y$ qui liera les entrées X aux sorties Y en minimisant l'erreur E_Y
- **Apprentissage non supervisé** : Ici, les algorithmes ne disposent pas d'un étiquetage des données, ils essayeront donc d'apprendre des *pattern* ou motifs fréquents pour grouper les données similaires. De tels algorithmes ne se préoccupent pas de la classe, mais de la similarité entre un groupe de données [16]
- **Apprentissage par renforcement** : Cette dernière catégories d'algorithmes apprennent par un système de *trial and error*¹ en interagissant avec l'environnement pour accomplir une tâche (voir 3.6.3.3)

3.2.2 Réseaux de neurones artificiels

Un réseau de neurones artificiels est une structure d'appariement non-linéaire inspiré de la façon dont les systèmes nerveux biologiques fonctionnent. Ils permettent de modéliser les relations sous-jacentes des données. ils sont composés d'un nombre arbitrairement large de plus petites unités de calcul interconnectées appelées neurones, qui traient l'information d'une manière parallèle dans le but de résoudre un problème bien spécifique [17]. Ils ont notamment connu un très grand succès lors des dernières années dans différents domaines comme la reconnaissance d'images [18], la reconnaissance automatique de la parole [19, 20] ou encore la classification de textes [21, 22].

Il existe une variété d'architectures de réseaux de neurones, nous traiterons dans cette section de trois d'entre elles :

- Réseaux de neurones multicouches denses
- Réseaux de neurones profonds
- Réseaux de neurones récurrents et leurs variantes

1. Essais et erreur?

3.2.2.1 Réseaux de neurones multicouches denses

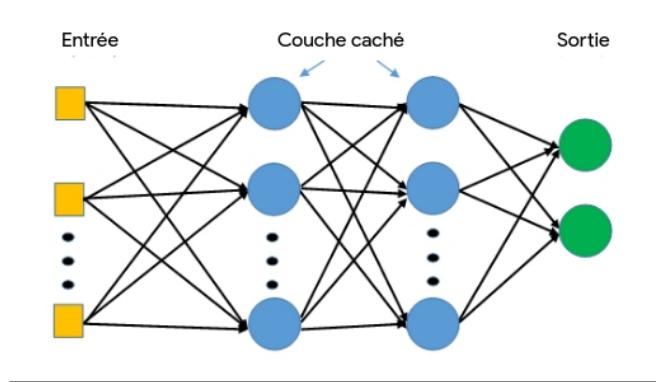


Figure 3.1 – Architecture basique d'un réseaux de neurones multicouches

La forme la plus basique que peut avoir un réseau de neurones est celle d'un réseau multicouches comme montré dans 3.1, elle se compose de trois parties :

- **Une couche d'entrée :** cette couche est composé d'un ensemble de neurones ou *perceptron* [23], elle représente l'information en entrée codifié en un vecteur numérique χ
- **Une ou plusieurs couches cachées :** le cœur du réseau, c'est une succession de couche de neurones dont chaque couche ω_i reçoit un signal sous forme d'une ou plusieurs valeurs numériques depuis une couche antérieure ω_{i-1} ou bien la couche d'entrée χ , puis envoie en sortie un autre signal de même nature qui est une combinaison non-linéaire du signal en entrée vers une couche suivante ω_{i+1} ou bien la couche de sortie
- **Une couche de sortie :** elle permet de calculer une valeur y qui peut être vu comme la prédiction du modèle Φ par rapport à son entrée χ :

$$y = f_{\Phi}(\chi) \quad (3.1)$$

Le réseau calculera une erreur $e(y, \hat{y})$ en fonction de sa valeur en sortie et de la valeur exacte puis corrigera cette erreur au fur et à mesure du parcours des données d'apprentissage [24]

3.2.2.2 Réseaux de neurones profonds

Les réseaux de neurones à une seule couche sont dit *shallow*, ils présentaient l'avantage d'être assez rapide durant la phase d'apprentissage, cependant les limites computationnelles d'antan se sont vite brisées avec le développement de processeurs plus puissants, de plus avec l'explosion des données sur le web, toutes les conditions nécessaires pour la mise en place d'architectures plus complexes étaient réunies. Les réseaux de neurones profonds sont une adaptation des réseaux multi-couches classiques avec généralement plus de 3 couches cachées.

même si le principe reste le même, l'apprentissage profond est puissant car les architectures les plus complexes permettent d'extraire automatiquement les caractéristiques qui sont importantes mais non visibles, c'est le cas des réseaux de neurones convolutif et récurrents [25]

3.2.2.3 Réseaux de neurones récurrents

Un aspect que les réseaux de neurones (profonds ou pas) ne peuvent capturer est la notion de séquentialité, en effet beaucoup de problèmes qui sont de nature séquentielle ne peuvent être modélisés par les architectures dites classiques, comme l'analyse d'un texte, l'introduction d'une notion de séquence permet donc de capturer des dépendances entre certains états et leurs voisins, on parle ici de contexte [26].

Les réseaux de neurones récurrents (RNNs) sont des réseaux de neurones *feedforward* dont certaines connections en sortie sont réintroduites comme entrées dans le réseau durant une étape ultérieure du processus d'apprentissage, ceci introduit la notion de temps dans l'architecture, ainsi à un instant t , un neurone récurrent recevra en entrée la données $x^{(t)}$ ainsi que la valeur de l'état caché $h^{(t-1)}$ résultante de l'étape précédente du réseau, la valeur en sortie $\hat{y}^{(t)}$ est calculée en fonction de l'état caché $h^{(t)}$, les équations suivantes montrent les calculs effectués :

$$h^{(t)} = \tanh(W^{hx} \times x^{(t)} + W^{hh} \times h^{(t-1)} + b_h) \quad (3.2)$$

$$\hat{y}^{(t)} = \text{softmax}(W^{hy} \times h^{(t)} + b_y) \quad (3.3)$$

Où W^{hx} est la matrice de poids entre la couche d'entrée et l'état caché et W^{hh} est la matrice de poids de récurrence (entre l'état t et $t - 1$) les deux vecteurs b_h et b_y sont les vecteurs de biais et \times est l'opération du produit matriciel[26]

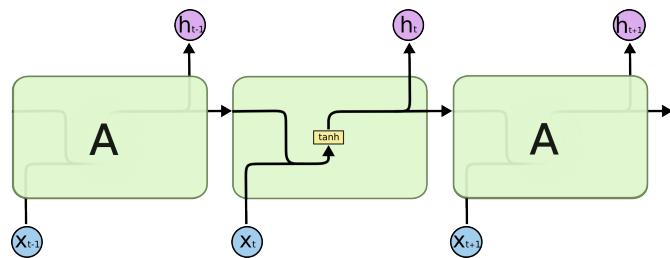


Figure 3.2 – Architecture interne d'un réseau de neurones récurrent à un instant t [27]

Un des principaux problèmes que rencontrent les RNNs est celui du *Vanishing gradient* traduit par la *Problème de disparition du gradient* [28], les relations à long termes entre les séquences ne sont donc pas capturées. Ainsi, pour remédier à ce problème, des architectures de réseaux de neurones dotées d'un module de mémoire ont été introduites.

3.2.2.4 Réseaux de neurones récurrents à mémoire court et long terme (LSTM)

Premièrement introduit en 1997 par *Sepp Hochreiter* et al. dans [29], cette architecture de réseaux de neurones récurrents est dotée d'un système de *portes* qui filtrent l'information qui y passe ainsi que d'un état interne de la cellule mémoire d'un LSTM , ainsi nous pouvons décortiquer les composantes d'une cellule LSTM (voir 3.3) comme suit :

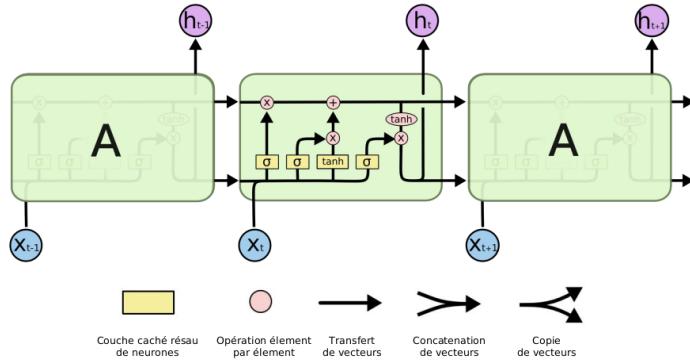


Figure 3.3 – Architecture interne d'une cellule mémoire dans un réseau LSTM [27]

- Porte d'entrée (Input gate) :** C'est une unité de calcul qui laisse passer certaines informations en entrée en utilisant une fonction d'activation sigmoïde pour pondérer les composantes du vecteur d'entrée à une étape t et celles du vecteur d'état interne à l'étape $t - 1$ (1 : laisser passer, 0 : ne pas laisser passer) et ainsi générer un vecteur candidat \tilde{C}_t [29].

$$\begin{aligned} i_t &= \sigma(W_i \times [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \times [h_{t-1}, x_t] + b_C) \end{aligned} \quad (3.4)$$

- Porte d'oubli (Forget gate) :** De manière similaire, cette porte permet de spécifier au fur et à mesure de l'apprentissage les informations à oublier, qui sont donc peu importantes [29, 26].

$$f_t = \sigma(W_f \times [h_{t-1}, x_t] + b_f) \quad (3.5)$$

- État interne de la cellule (Internal cell's state) :** C'est une sorte de convoyeur qui fait circuler l'information à travers la cellule, cet état est mis à jour à travers la combinaison des deux valeurs précédemment filtré par les portes f_c et i_c [29].

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (3.6)$$

- Porte de sortie (Output gate) :** Pour renvoyer un résultat comme état interne du réseau, la cellule filtre son vecteur d'état et le combine avec la donnée en entrée et l'état du réseau précédent pour ne laisser passer que certaines informations [27, 29, 26].

$$\begin{aligned} o_t &= \sigma(W_o \times [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \quad (3.7)$$

3.2.2.5 Modèle basé N-grammes

Très souvent quand il est nécessaire de traiter un ensemble de mots, phrases ou bien tout entité atomique qui constitue une séquence, il serait intéressant de pouvoir assigner une probabilité de vraisemblance à une séquence (de mots par exemple) en particulier.

Dans un contexte textuel, un N-gramme est une suite de N mots consécutifs qui forment une sous-chaîne S' d'une chaîne de caractères S . Un exemple d'un ensemble de bi-grammes (2-grammes) pour la phrase "Ouvre le fichier home" serait donc : (Ouvre,le), (le,fichier), (fichier,home).

Ainsi, en disposant d'un corpus de texte assez large et diversifié et d'une méthode de comptage efficace, il serait possible de calculer la vraisemblance d'apparition d'un mot w après une certaine séquence de mots t sous forme d'une probabilité P [30].

$$P(w|t) = \frac{\text{Comptage}(t, w)}{\text{Comptage}(t)} \quad (3.8)$$

3.2.2.6 Modèle de Markov caché (Hidden Markov Models HMM)

Informellement, un modèle de Markov caché (HMM) est un outil de représentation pour modéliser la distribution de probabilité d'une séquence d'observations. Il est dit *caché* pour deux raisons. Premièrement, il est assumé qu'une observation O à un instant t (dénotée O_t) est le résultat d'un certain processus (souvent stochastique) dont l'état S_t est caché à l'observateur. Deuxièmement, l'état S_t du processus caché ne dépend uniquement que de son état à l'instant $t - 1$, il est alors dit que ce processus est Markovien ou qu'il satisfait la propriété de Markov [31, 32].

D'un façon plus formelle, un HMM est un 5-tuple $\langle S, V, \Pi, A, B \rangle$ [33] où :

- $S = \{s_1 \dots s_N\}$ est l'ensemble fini des N états du processus sous-jacent.
- $V = \{v_1 \dots v_M\}$ est l'ensemble des M symboles qui constituent un certain vocabulaire.
- $\Pi = \{\pi_i\}$ est une distribution initiale des probabilités d'états tel que π_i est la probabilité de se trouver à l'état i à l'instant $t = 0$, bien évidemment $\sum_{i=1}^N \pi_i = 1$
- $A = \{a_{ij}\}$ est une matrice $N \times N$ dont chaque entrée a_{ij} est la probabilité de transition d'un état i à un état j avec $\sum_{j=1}^N a_{ij} = 1$ pour tout $i, j = 1 \dots N$.
- $B = \{b_i(v_k)\}$ est l'ensemble des distributions de probabilités d'émission (ou d'observation), donc $b_i(v_k)$ est la probabilité de générer un symbole v_k du vocabulaire étant donné un certain état i avec $\sum_{k=1}^M b_i(v_k) = 1$ pour tout $i = 1 \dots N$.

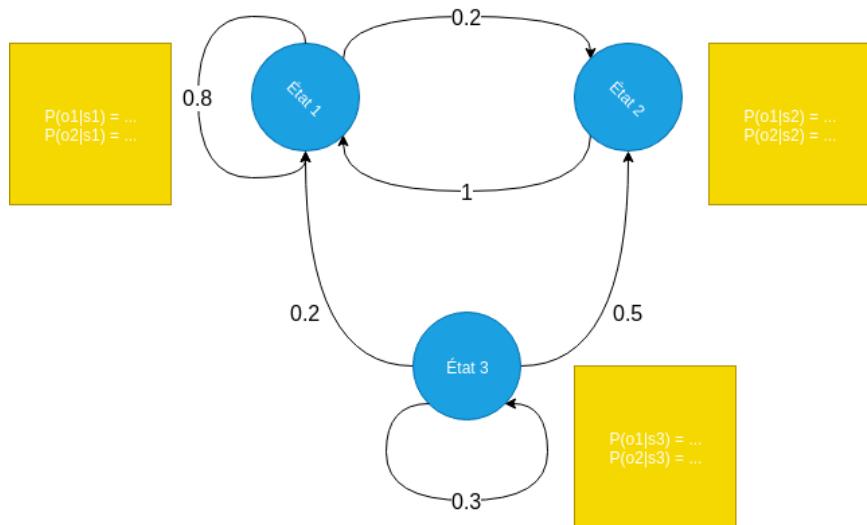


Figure 3.4 – Exemple d'une distribution de probabilité de transition ainsi qu'une distribution de probabilité d'observations pour un HMM à 3 états et 2 observations

Le but serait de trouver la séquence d'états $S_{1:K}$ qui maximiserait la probabilité [31] :

$$P(Y_{1:K}|O_{1:K}) = P(O_1)P(O_1|S_1) \prod_{t=2}^K P(S_t|S_{t-1})P(O_t|S_t) \quad (3.9)$$

Pour y parvenir d'une manière efficace, un décodeur qui implémente l'algorithme de viterbi peut être utilisé [34, 35]

3.3 Schéma global d'un SPA

Durant nos lectures des différents travaux sur le domaine, nous avons pu dresser un schéma assez générale qui englobe les principaux modules d'un SPA :

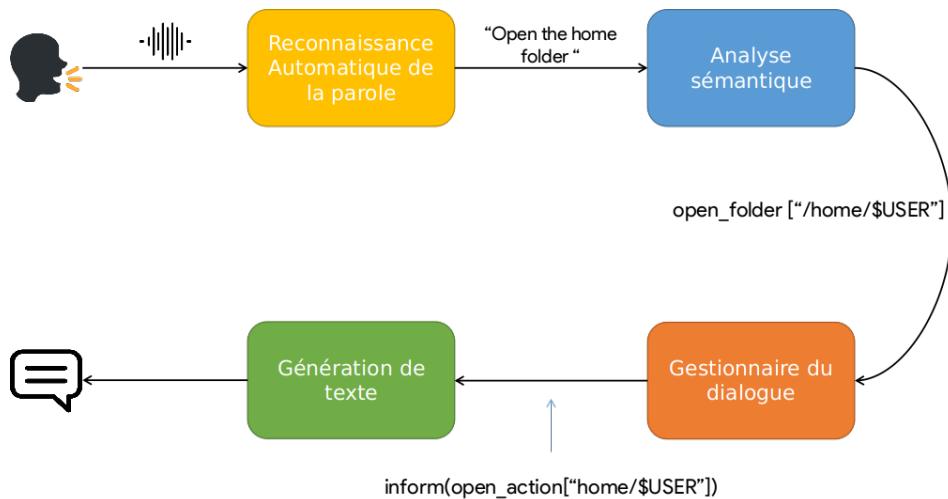


Figure 3.5 – Schéma abstractif d'un SPA [36]

Le processus peut être résumé en des étapes cruciales (qui seront détaillées dans les sections suivantes) :

- L'utilisateur énonce une requête en utilisant sa voix, le signal est ensuite transformé en sa version textuelle.
- La requête étant sous un format textuel brut ne peut pas être interprétée ou comprise par la machine, une représentation sémantique est générée qui compactera les principales informations contenues dans la requête, à savoir l'intentions de l'utilisateur et ses arguments.
- Au fur et à mesure que l'utilisateur énonce des requêtes, le système doit pouvoir être capable d'utiliser le contexte du dialogue pour interagir avec ce dernier pour pouvoir atteindre le but final du dialogue (exécuter une commande système, trouver des informations sur le web ou sur la machine locale ...), le gestionnaire du dialogue communique avec son environnement en utilisant la même représentation sémantique produite par l'analyse précédente.
- Puisqu'il est préférable de cacher à l'utilisateur tout comportement interne au système, il sera préférable de transformer la trame sémantique (voir 3.9) en texte naturelle pour l'afficher en sortie.

Il est à noter que chacun des modules seront indépendant pour ce qu'il est de leur fonctionnement interne, ainsi aucun n'assumera un fonctionnement arbitraire des autres modules, seul le format des informations qui circulent entre eux devra être établi au préalable pour un fonctionnement durable et robuste au changement.

Pour ce qui en est de la suite du chapitre, nous allons présenter les différents modules ainsi que les techniques et architectures qui sont considérés comme étant l'état de l'art du domaine.

3.4 Reconnaissance automatique de la parole (ASR)

Le premier module qui compose le système est celui de la reconnaissance automatique de la parole (ASR), le but d'un tel système (ou sous-système dans notre cas) est de convertir un signal audio correspondant la locution d'un utilisateur en un texte qui peut être interprété par la machine [37]. Différentes approches ont été développées au court des années, une architecture s'est ensuite dégagée, les systèmes passent par deux phases : La phase d'apprentissage et la phase de reconnaissance. La première consiste à collecter les données qui constituent le corpus d'apprentissage, un ensemble de fichiers audios avec leurs transcriptions en texte et en phonèmes, le signal est ensuite traité pour en extraire des vecteurs de caractéristiques (ou attributs), la suite de l'apprentissage consiste à initialiser le HMM, lui passer les vecteurs précédemment extraits puis de l'enregistrer pour la phase suivante qui est la reconnaissance. Dans la deuxième phase le signal audio passe par le même procédé d'extraction des attributs, en utilisant un algorithme de décodage approprié [35], la séquence d'observations passe par le HMM et la meilleure séquence de mots est sélectionnée [20].

Une architecture assez générale s'est dégagée, quatre modules sont impliqués dans le processus de la reconnaissance, nous les citerons dans les sections qui suivent en énumérant les modèles utilisés dans chacun.

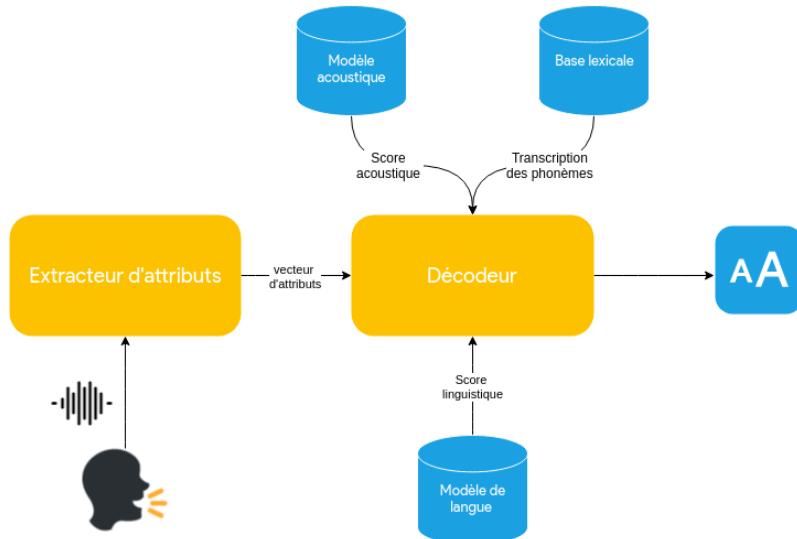


Figure 3.6 – Architecture des systèmes de reconnaissance de la parole [20]

3.4.1 Acquisition du signal et extraction d'attributs

l'étape consiste à extraire une séquence de vecteurs caractéristiques à partir du signal audio, fournissant une représentation compacte de ce dernier. Le processus démarre par la segmentation du signal en *trames*, une transformation de fourrier est ensuite appliquée à ces dernières pour engendrer un spectre de magnitude qui sera ensuite passé à un module de transformation en spectre de Mel (Mel-Spectrum) qui est une sorte de changement d'échelle, finalement, une transformation inverse de fourrier est appliquée pour engendrer le Mel-Cepstrum qui est le vecteur d'attributs que nous recherchions [38]. Un tel processus peut utiliser plusieurs techniques pour la mise à l'échelle : MFCC (Mel-frequency cepstrum) [39], LPC (Linear Predictive coding) [40] et RASTA (RelAtive SpecTrAl) [41], chacune possédant ses avantages et ses inconvénients.

3.4.2 Modélisation acoustique et modélisation du lexique

C'est le cœur du système de reconnaissance, le but est de construire un modèle permettant d'identifier une séquence de phonèmes à partir d'une séquence d'observations de vecteurs d'attributs, ceci peut être fait en utilisant un modèle HMM et en disposant d'un corpus de fichiers audio accompagnés de leurs transcriptions en phonèmes et en texte [42, 33], ou bien un réseau de neurones profonds [20], ou encore une hybridation de ces derniers [43]. Le modèle acoustique procède après la reconnaissance de la séquence de phonèmes au décodage de ces derniers, ceci est fait en utilisant un dictionnaire linguistique qui transcrit chaque mot du vocabulaire en les phonèmes qui le constituent, ceci peut induire à un cas d'ambiguïté où plusieurs mots peuvent avoir la même transcription phonétique (ou bien aucun mot ne correspond à cette séquence). Pour lever cette ambiguïté un modèle de langue est nécessaire pour décider quelle est la séquence de mots la plus probable qui coïncide avec la séquence de phonèmes observée.

3.4.3 Modélisation de la langue

Cette étape consiste à modéliser les aspects linguistiques de la langue que le système essaye de traiter, souvent spécifique au domaine d'application pour restreindre l'espace de recherche des mots reconnaissables par ce dernier et déterminer la séquence de mots en sortie la plus plausible. Les modèles utilisés sont basés soit sur des grammaires à contexte libre dans le cas où les séquences de mots reconnaissables sont peu variées et peuvent être modélisées par le biais de règles de la langue [44]. Dans un cadre plus récent, les modèles probabilistes basés sur les N-grammes sont utilisés. Disposant d'un volume de données textuelles assez conséquent, l'utilisation de modèles basés sur les N-grammes ont prouvé leur utilité [30, 20, 45]

3.5 Compréhension du langage naturel (NLU)

Après avoir obtenu la requête de l'utilisateur, le module qui prend le relai est celui de la compréhension du langage naturel. Ce domaine fait partie du traitement automatique du langage naturel (NLP), il est principalement focalisé sur les tâches qui traitent le niveau sémantique voir même pragmatique de la langue tel que la classification de texte, l'analyse de sentiments ou encore le traitement automatiques des e-mails. Dans le contexte d'un SPA, le but final d'un module de compréhension du langage naturel est de construire une représentation sémantique de la requête de l'utilisateur, étant donné une telle requête préalablement transformé en texte brut dans un langage naturel le module essayera d'en extraire une information primordiale qui est l'intention du locuteur. Une intention (ou intent) est une abstraction sémantique d'un but ou d'un sous-but de l'utilisateur, plusieurs requêtes de l'utilisateur peuvent avoir le même intent, ce qui rend l'utilisation d'une telle représentation essentielle pour que la machine puisse mieux comprendre l'utilisateur.

3.5.1 Classification d'intentions

La classification d'intentions à partir d'un texte est une tâche réalisable avec des techniques récentes d'apprentissage automatique, plus précisément en utilisant des modèles basé sur les réseaux de neurones récurrents à mémoire court et long terme (LSTM) (voir 3.3). Dans [46] et [47] deux architectures ont fait leurs preuves, dans le premier travail les auteurs ont utilisé une architecture BLSTM (LSTM Bidirectionnel) pour capturer les contextes droit et gauche d'un mot donné [48], le dernier état caché retourné par la cellule LSTM est ensuite utilisé pour la classification, une couche d'attention est ajoutée [49] pour permettre au modèle de se focaliser sur certaines parties du texte en entrée. La deuxième architecture est un peu plus simpliste, en effet elle utilise des cellules LSTM basiques avec une couche de classification sur le dernier état, l'avantage est le temps d'apprentissage assez réduit au détriment d'une erreur de classification moins accrue mais toujours dans les normes.

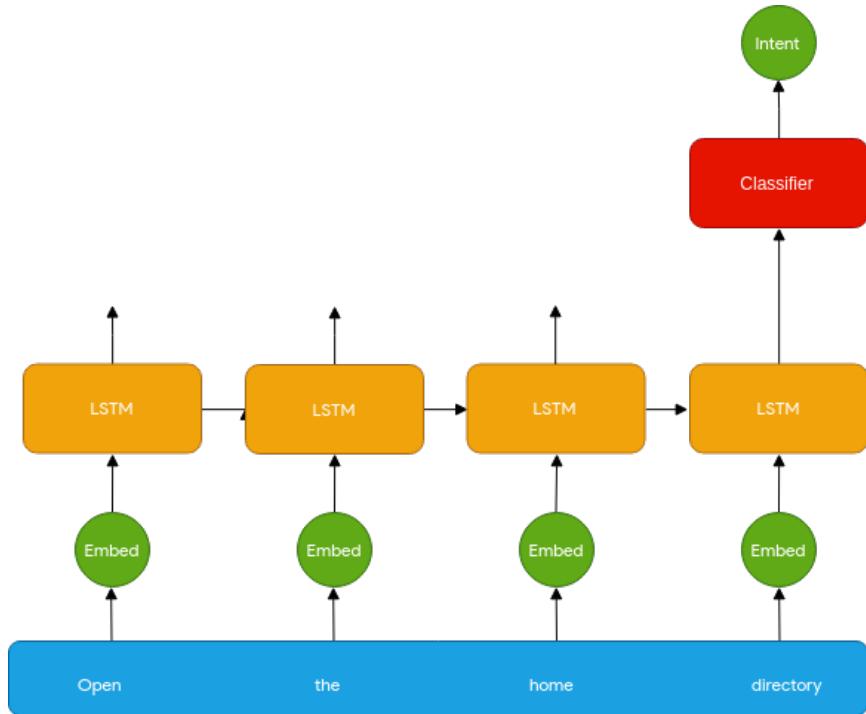


Figure 3.7 – Architecture de base d'un classificateur d'intents [46]

3.5.2 Extraction d'entités

Déterminer l'intention d'un utilisateur ne s'arrête pas qu'au stade de l'identification de ce dit *intent*, en effet pour mieux représenter l'information récoltée depuis la requête il faut en extraire des entités (nommés ou spécifiques au domaine) qui seront des arguments de l'*intent* identifié, cette tâche consiste donc à, pour un *intent* I donné, extraire les arguments *Args* qui lui sont appropriés depuis le texte de la requête. Plusieurs approches sont possibles, dans [47] les auteurs ont taclé le problème comme étant la traduction d'une séquence de mots en entrée en une séquence d'entités en sortie, le schéma suivant explicite le processus :

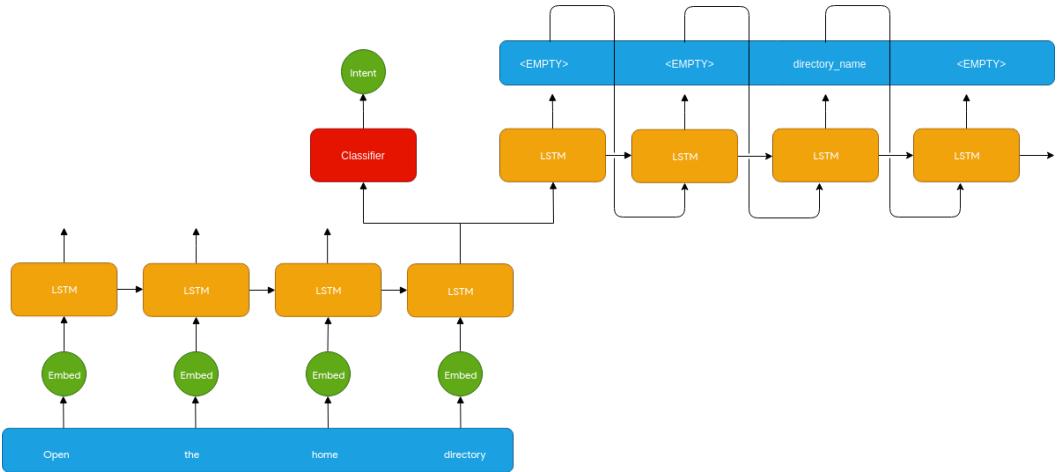


Figure 3.8 – Architecture de base d'un classificateur d'intents doublé d'un extracteur d'entité [47]

3.5.3 Analyse sémantique

Disposant des deux modèles précédemment cités, le module NLU peut maintenant construire une représentation sémantique adéquate qui va être transmise au module suivant qui sera le gestionnaire du dialogue, l'avantage d'avoir en sortie une structure sémantique universelle est la non dépendance par rapport à la langue de l'utilisateur, en effet le système pourra encore fonctionner si une correspondance entre la nouvelle langue et le format choisi pour la représentation sémantique peut être trouvée. Le module donnera donc en sortie une trame sémantique (Semantic frame) qui comprendra les informations préalablement extraites, à savoir l'intent et les arguments (slots) qui lui sont propres [46, 47, 50].

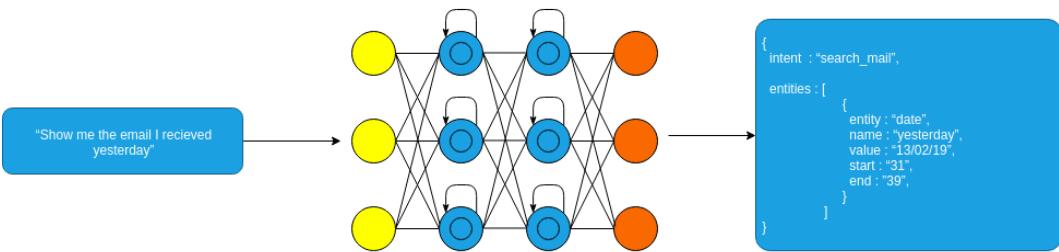


Figure 3.9 – Exemple d'une trame sémantique pour une requête donnée

3.6 Gestion du dialogue

La compréhension du langage naturel permet de transformer un texte en une représentation sémantique. Afin qu'un système puisse réaliser un dialogue aussi anthropomorphe que possible, il doit décider, à partir des représentations sémantiques reçues au cours du dialogue, quelle action à prendre à chaque étape de la conversation. Celle-ci est transmise au générateur du langage naturel 3.7 pour afficher un résultat à l'utilisateur. On distingue deux principaux modules généralement présents dans les systèmes de gestion de dialogue :

- Un module qui met à jour l'état du gestionnaire de dialogue.
- Un module qui détermine la politique d'action du gestionnaire de dialogue.

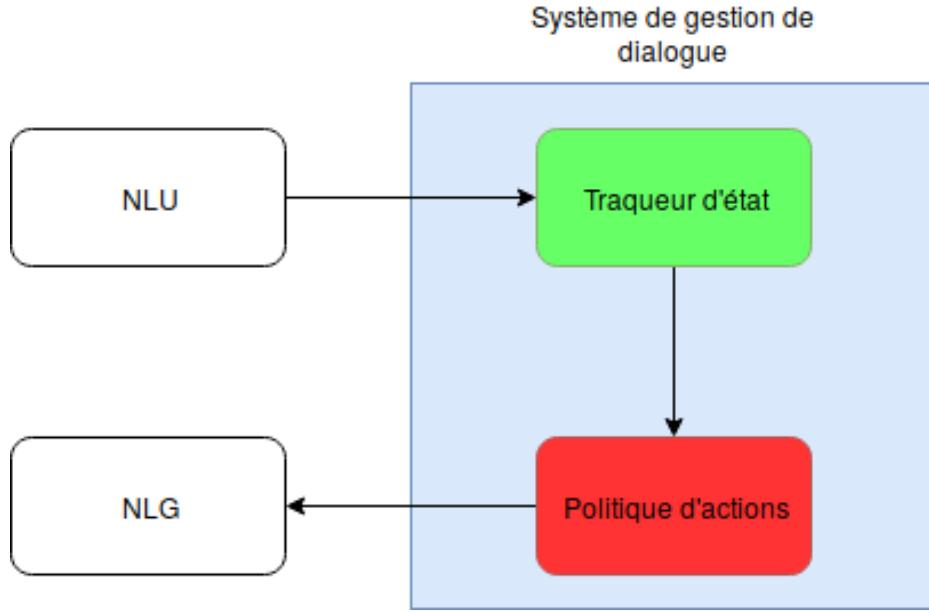


Figure 3.10 – Schéma général d'un gestionnaire de dialogue

3.6.1 Processus de décision Markovien (MDP)

Un gestionnaire de dialogue peut être modélisé par un processus de décision Markovien[51]. Ce dernier est modélisé par un 4-tuple (S, A, P, R) (*):

- S : ensemble des états du système.
- A : ensemble des actions du système.
- P : distribution de probabilités de transition entre états sachant l'action prise. $P(s'/s, a)$ est la probabilité de passer à l'état s' sachant qu'on était à l'état s après avoir pris l'action a .
- R : est la récompense reçue immédiatement après avoir changé l'état avec une action donnée. $R(s'/s, a)$ est la récompense reçue après avoir passé à l'état s' sachant qu'on était à l'état s après avoir pris l'action a .

D'après (*) un MDP, à tout instant t , est dans un état s . Dans notre cas c'est l'état du gestionnaire de dialogue. Il peut prendre une action a afin de passer à un nouvel état s' . Sur ce fait, il reçoit une récompense qui, dans notre cas, est une mesure sur les performances du système de dialogue. Le but est de maximiser les récompenses reçues.

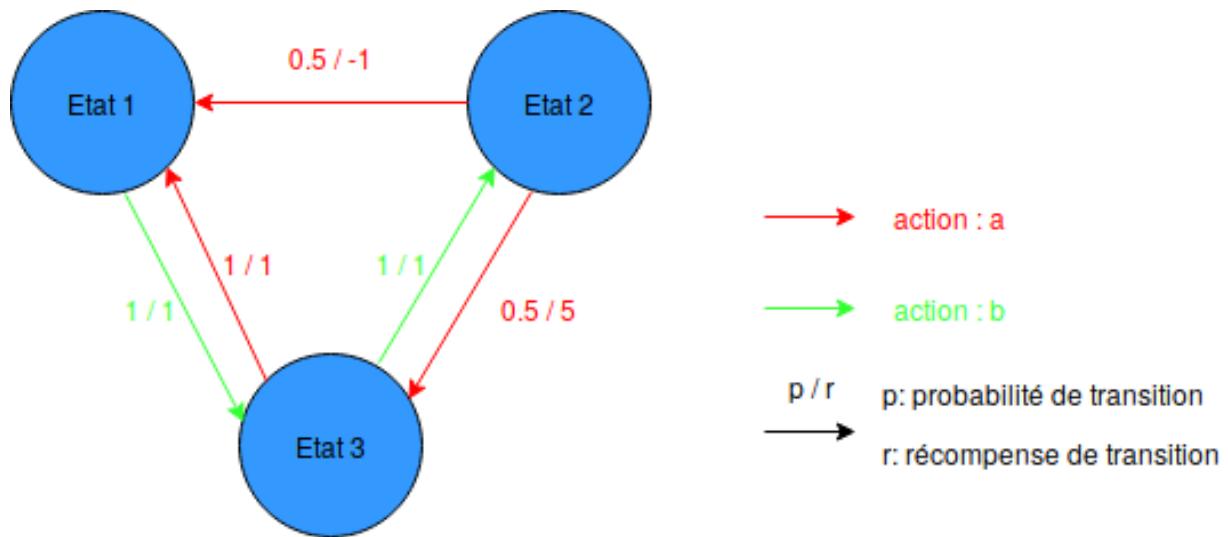


Figure 3.11 – Schéma représentant les transitions entre états dans un MDP

3.6.2 État du gestionnaire de dialogue

L'état d'un système de dialogue est une représentation sémantique qui contient des informations sur le but final de l'utilisateur ainsi que l'historique de la conversation. La représentation souvent utilisée dans les systèmes de dialogue est celle du cadre sémantique [52]. Cette structure contient des emplacements à remplir sur un domaine donné, la figure 3.12 illustre un exemple de cadre sémantique.

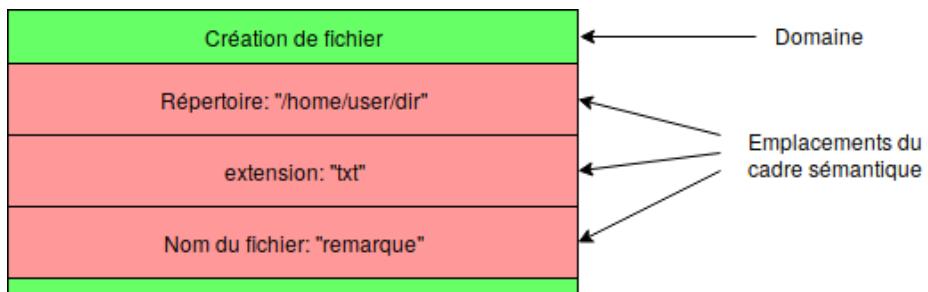


Figure 3.12 – Schéma représentant un cadre sémantique avec comme domaine : création de fichier

À l'arrivée d'une nouvelle information, un module dédié met à jour l'état du gestionnaire de dialogue. Comme l'action du système de dialogue est décidée à partir de son état, cette tâche est donc essentielle au bon fonctionnement du système. Plusieurs méthodes ont été proposées pour gérer le suivi de l'état du gestionnaire de dialogue.

3.6.2.1 Suivi de l'état avec une base de règles

La méthode traditionnelle utilisée consiste à écrire manuellement les règles à suivre lors de l'arrivée d'une nouvelle information pour mettre à jour l'état[53]. Cependant, les bases de

règles sont très susceptibles à faire des erreurs[52] comme ils sont peu robuste face aux incertitudes.

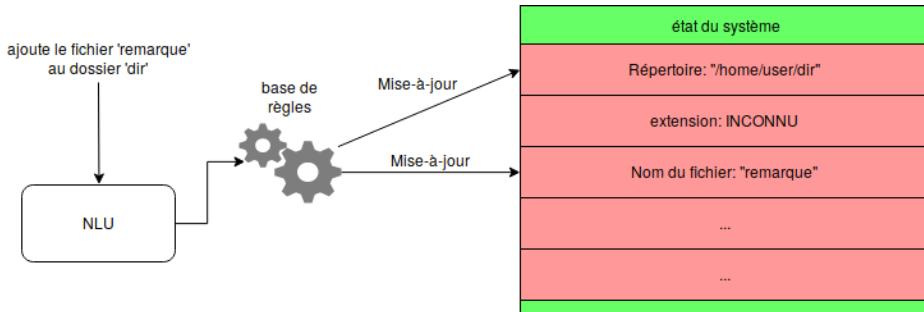


Figure 3.13 – Schéma représentant la mise-à-jour de l'état par un système basé règles

3.6.2.2 Suivi de l'état avec des méthodes statistiques

Le suivi dans ce cas se fait en gardant une distribution de probabilités sur l'état du système. Doù, utilisation des processus de décision markovien partiellement observable (POMDP)[54] quon introduira par la suite. Dans ce cas, le système garde une distribution de probabilités sur les valeurs possibles des différents emplacements du cadre sémantique.

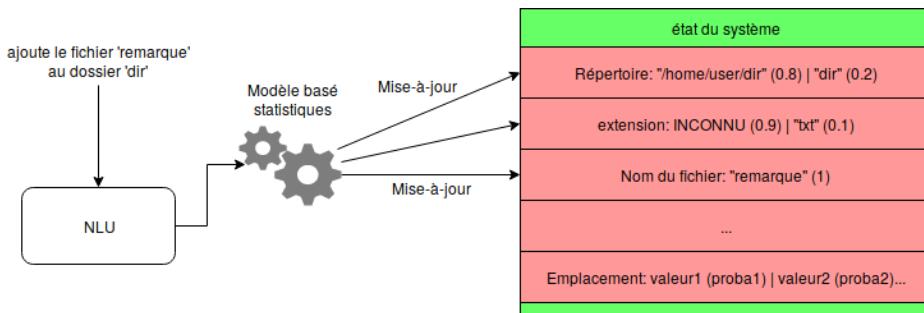


Figure 3.14 – Schéma représentant la mise-à-jour de l'état par un système basé statistiques

3.6.2.3 Processus de décision markovien partiellement observable (POMDP)

Comme dans les processus de décision markovien, un POMDP[55] passe dun état à un autre en prenant une des actions possibles. Cependant, ce dernier ne connaît pas l'état exacte dans lequel il se trouve à un instant t . Il reçoit par contre une observation, dans notre cas cest l'action de l'utilisateur, à partir de laquelle il peut estimer une distribution de probabilités sur l'état actuel. Pour résumer cela, un POMDP est un 6-tuple (S, A, P, R, M, O) :

- Les 4 premiers composants sont les mêmes que ceux dun MDP 3.6.1.
- M : lensemble des observations.
- O : distribution de probabilités sur les observations o en connaissant l'état et l'action prise pour y arriver. $O(o|s, a)$ est la probabilité d'observer o sachant qu'on se trouve à l'état s et qu'on a pris l'action a pour y arriver.

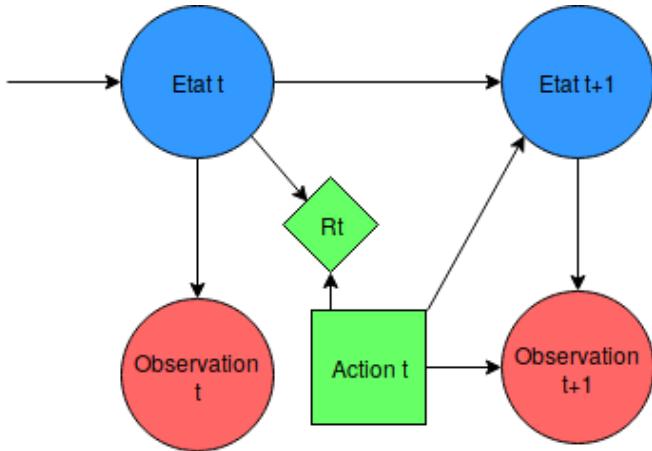


Figure 3.15 – Diagramme d'influence dans un POMDP

3.6.2.4 Suivi de l'état avec réseaux de neurones profonds

Récemment, des approches utilisant les réseaux de neurones profonds 3.2.2 ont fait leurs apparitions. En effet, l'utilisation des architectures profondes permet de capter des relations complexes entre les caractéristiques d'un dialogue et ainsi mieux estimer l'état du système. Le réseau de neurones estime les probabilités de toutes les valeurs possibles d'un emplacement du cadre sémantique [56]. En conséquence, il peut être utilisé comme modèle de suivi d'état pour un processus partiellement observable.

3.6.3 Politique de gestion de dialogue

La première partie était dédiée au module qui suit l'état du système de dialogue. Dans cette partie, Nous allons présenter des approches proposées afin d'arriver au but du MDP, c'est à dire quelles actions prendre pour maximiser la somme des récompenses obtenues.

3.6.3.1 Gestion de dialogue avec une base de règles

Les premières approches utilisaient des systèmes de règles destiné à un domaine bien spécifique. Elles étaient déployées dans plusieurs domaines d'application pour leur simplicité. Cependant, le travail manuel nécessaire reste difficile à faire et, généralement, aboutit pas à des résultats flexibles qui peuvent suivre le flux du dialogue convenablement [57].

3.6.3.2 Gestion de dialogue par apprentissage

La résolution d'un MDP revient à trouver une estimation de la fonction de récompense afin de pouvoir choisir la meilleure action. La majorité des approches récentes utilise l'apprentissage par renforcement pour estimer la récompense obtenue par une action et un état donnés. Cette préférence par rapport aux approches supervisées revient à la difficulté de produire des corpus de dialogues [58], encore moins des corpus annotés avec les récompenses à chaque transition. Néanmoins, il existe des approches de bout en bout

qui exploitent des architectures avec réseaux de neurones profonds et traitent le problème comme Seq2Seq3.2.2.3 afin de produire directement une sortie à partir des informations reçues de l'utilisateur[59, 60].

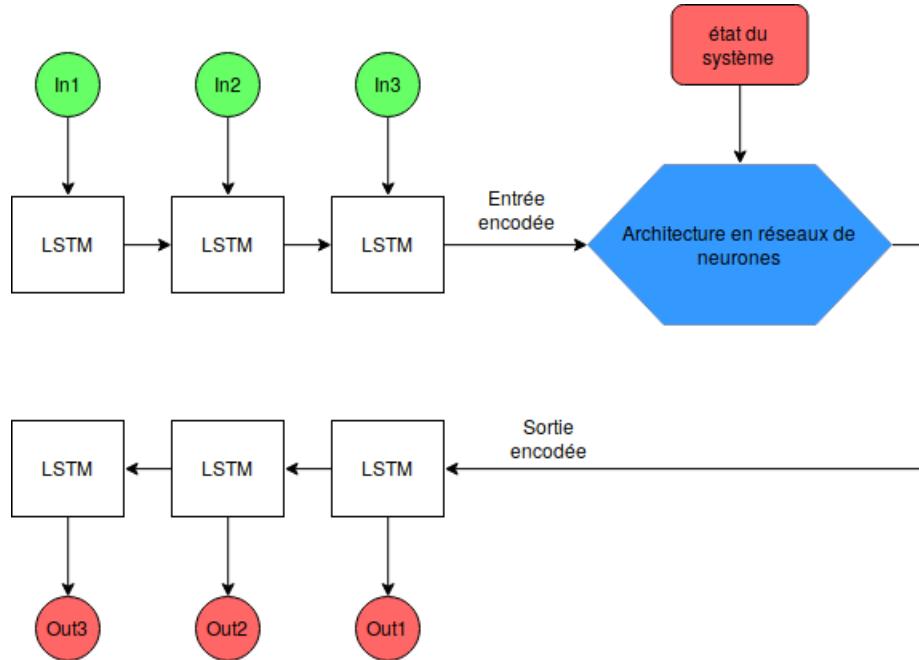


Figure 3.16 – Schéma de gestion de dialogue de bout en bout avec architecture Seq2Seq

3.6.3.3 Apprentissage par renforcement

L'apprentissage par renforcement est une approche qui a pour but d'estimer une politique d'actions à prendre dans un environnement. Celle-ci doit maximiser une mesure d'évaluation qui est sous forme de récompenses obtenues après chaque action[61]. L'environnement est souvent modélisé comme un MDP, ou éventuellement POMDP. L'agent d'apprentissage passe donc d'un état à un autre en prenant des actions dans cet environnement. L'apprentissage se fait dans ce cas en apprenant par l'expérience de l'agent, à savoir les récompenses obtenues par les actions prises et de quels états elles étaient prises. Il peut ainsi estimer la fonction de récompense pour pouvoir faire le choix d'actions optimales.

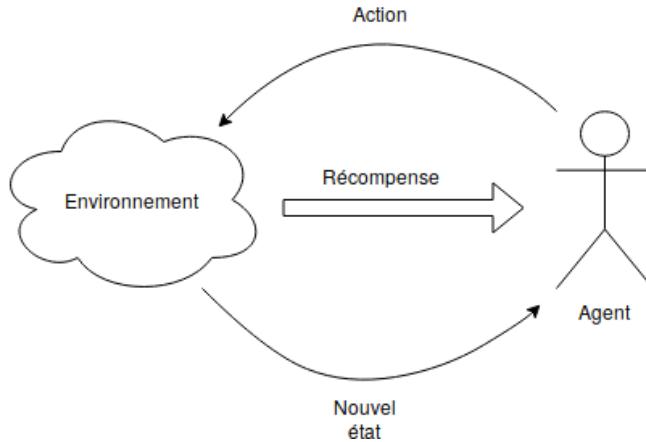


Figure 3.17 – Schéma d’interaction agent-environnement dans l’apprentissage par renforcement

Il existe plusieurs méthodes que l’agent peut utiliser pour estimer la fonction de récompense [62], notamment Deep Q Learning (DQN) [63] qui utilise les réseaux de neurones profonds pour trouver une approximation à cette fonction. Dans cette approche, à chaque fois que l’agent prend une action, il compare la récompense reçue avec celle estimée par le réseau de neurones, il applique ensuite l’algorithme de rétro-propagation pour corriger les erreurs du réseau.

Le système de dialogue est modélisé par un MDP. Seulement, ce dernier inclut l’utilisateur comme partie de l’environnement. En conséquence, pour appliquer l’apprentissage par renforcement, il est nécessaire qu’un utilisateur communique avec le système pour qu’il apprenne. Donc, la nécessité d’utiliser les simulateurs d’utilisateur.

Simulateur d’utilisateur Un simulateur d’utilisateur est un programme qui se comporte comme un humain interagissant avec un système de dialogue. Celui-là doit pouvoir estimer sa satisfaction après une interaction. En d’autres termes, il doit comporter d’une fonction de récompense. Il existe plusieurs méthodes pour créer un simulateur d’utilisateur :

- Les simulateurs d’utilisateur basé règles : dans ce cas une liste de règles est écrite manuellement que le simulateur doit suivre pour communiquer avec le système[64].
- Les simulateurs d’utilisateur basé n-grammes : ils traitent un dialogue comme une séquence d’actions. Ils prennent les n-1 actions pour estimer l’action la plus probable que peut prendre le simulateur à partir des statistiques tirées d’un corpus de dialogues[65].
- Les simulateurs d’utilisateur basé HMM : les états du modèle sont les états du système, les observations sont ses actions. Ainsi un HMM peut estimer l’état le plus probable du système pour prendre une action. Il existe d’autres variantes, IHMM et IOHMM, qui incluent les probabilités conditionnelles des actions de l’utilisateur sachant l’état du système ou l’action du système directement dans le modèle HMM[66].
- Les simulateurs d’utilisateur avec apprentissage par renforcement : Comme le gestionnaire de dialogue, le simulateur apprend par renforcement au même temps. Dans

ce cas la fonction de récompense pour les deux agents peut être apprise à partir des dialogues humain-humain[67].

3.7 Génération du langage naturel (NLG)

Le domaine de la génération automatique du langage naturel est l'un des domaines dont les bordures sont difficiles à définir[68]. Il est vrai que la sortie d'un tel système est clairement du texte. Cependant, l'ambiguïté se trouve dans ses entrées, c'est à dire, sur quoi se basera le système pour générer le texte. D'après (Reiter & Dale, 1997)[69] la génération du langage naturel est décrite comme étant le sous domaine de l'intelligence artificielle qui traite la construction des systèmes de génération de texte à partir d'une représentation non-linguistique de l'information. Celle-ci peut être une représentation sémantique, des données numériques, une base de connaissances ou même des données visuelles (images ou vidéos). Ceci dit, d'autres travaux, comme Labbé & Portet (2012)[70], utilisent les mêmes techniques pour des entrées linguistiques. Enfin la génération du langage naturel peut être très proche de la gestion de dialogue[71], en effet, le texte généré doit prendre en compte l'historique de la conversation et le contexte de l'utilisateur.

Il existe six tâches trouvées fréquemment dans les systèmes de génération de texte [69].

3.7.1 Détermination du contenu

Cette partie consiste à sélectionner les informations de l'entrée dont le système veut transmettre leur contenu sous forme de texte naturel à l'utilisateur. En effet, les données en entrée peuvent contenir plus d'informations que ce que l'on désire communiquer[72]. De plus, le choix de l'information peut aussi dépendre de l'utilisateur et de ses connaissances[71]. Ce qui requiert de mettre au point un système qui détecte les informations pertinentes à l'utilisateur.

3.7.2 Structuration de texte

Après la détermination du contenu, le système doit ordonner les informations à transmettre. Ceci dépend grandement du domaine d'application qui peut exiger des contraintes d'ordre temporelle ou de préférence par importance des idées. Les informations à transmettre en elles-mêmes sont souvent reliées par sens ce qui implique une certaine structuration de texte à respecter.

3.7.3 Agrégation de phrases

Certaines informations peuvent être transmises dans une même phrase. Cette partie introduit des notions de la linguistique afin que le texte généré soit plus lisible et éviter les répétitions. Un exemple de cela peut être la description de la météo à Alger au cours de la matinée :

- Il va faire 16°C à Alger à 7h.
- Il va faire 17°C à Alger à 8h.
- Il va faire 18°C à Alger à 9h.
- Il va faire 18°C à Alger à 10h.

Ceci peut être agrégé en un texte plus compacte : "La température moyenne à Alger sera de 17°C entre 7h et 10h."

3.7.4 Lexicalisation

Le système choisit les mots et les expressions à utiliser pour communiquer le contenu des phrases sélectionnées. La difficulté de cette tâche revient à l'existence de plusieurs manières d'exprimer la même idée. Cependant, certains mots ou expressions sont plus appropriés en certaines situations que d'autres. En effet, inscrire un but est une façon inadéquate d'exprimer un but contre son camp[73].

3.7.5 Génération d'expressions référentielles (REG)

Cette partie du système se focalise dans la génération d'expressions référentielles qui peuvent être entre autres : noms propres, groupes nominaux ou pronoms et ceci a pour but d'identifier les entités du domaine. Cette tâche semble être très proche de sa prédécesseur ; elle savère néanmoins plus délicate dû à la difficulté de confier suffisamment d'information sur l'entité afin de la différencier des autres[69]. Le système doit faire un choix de l'expression référentielle en se basant sur plusieurs facteurs, par exemple Mohammed, Le professeur ou Il font référence à la même personne. Le choix entre eux dépend de comment l'entité a été mentionnée auparavant, si elle l'a été, et des détails qui l'ont accompagnés.

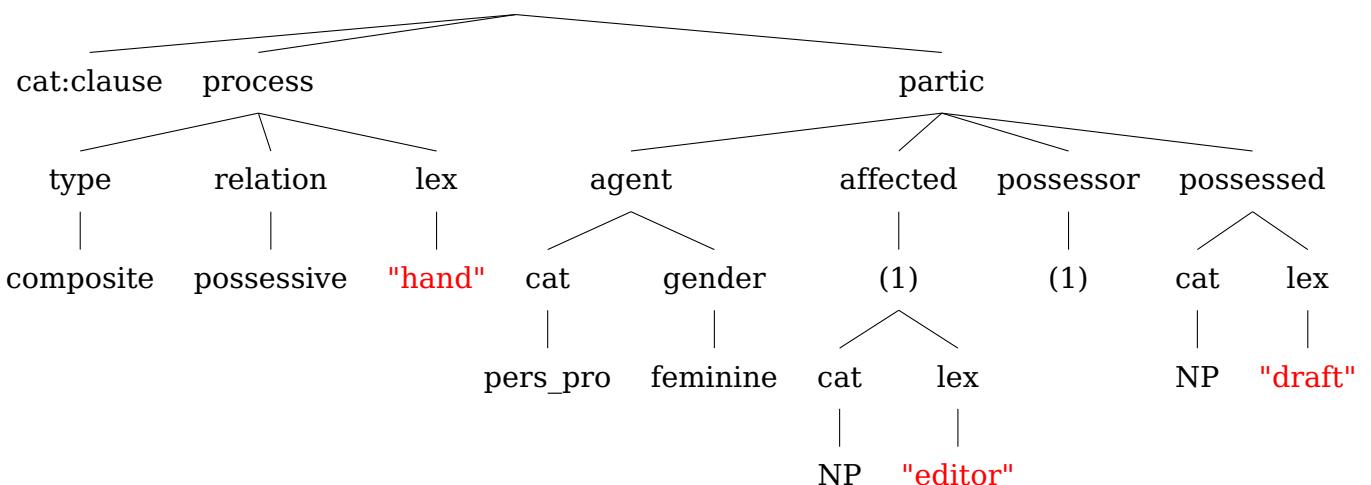
3.7.6 Réalisation linguistique

La dernière tâche consiste à combiner les mots et expressions sélectionnés pour construire une phrase linguistiquement correcte. Ceci requiert l'utilisation des bonnes formes morphologiques des mots, les ordonner, et éventuellement l'addition de certains mots du langage afin de réaliser une structure de phrase grammaticalement et sémantiquement correcte. Plusieurs méthodes ont été proposées, principalement les méthodes basées sur des règles manuellement construites (modèles de phrases, systèmes basés grammaires) et des approches statistiques [73].

Modèles de phrases : La réalisation se fait en utilisant des modèles de phrases pré-définis. Il suffit de remplacer des espaces réservés par certaines entrées du système. Par exemple, une application dans un contexte météorologique pourrait utiliser le modèle suivant : la température à [ville] atteint [température] le [date].

Cette méthode est utilisée lorsque les variations des sorties de l'application sont minimales. Son utilisation a l'avantage et l'inconvénient d'être rigide. D'un côté il est facile de contrôler la qualité des sorties syntaxiquement et sémantiquement tout en utilisant des règles de remplissage complexe [74]. Cependant, lorsque le domaine d'application présente beaucoup d'incertitude, cette méthode exige un travail manuel énorme, voire impossible à faire, pour réaliser une tâche pareille. Bien que certains travaux ont essayé de faire un apprentissage de modèles de phrases à partir d'un corpus[75] cette méthode reste inefficace lorsqu'il s'agit d'applications qui nécessitent un grand nombre de variations linguistiques.

Systèmes basés grammaire : La réalisation peut se faire en suivant une grammaire du langage. Celle-ci contient les règles morphologiques et de structures de la langue, notamment la grammaire systémique fonctionnelle (SFG)[76] a été largement utilisé comme dans NIGEL[77] ou KPML[78]. L'exploitation des grammaires dans la génération du texte nécessite généralement des entrées détaillées. En plus des composantes du lexique sélectionnées, des descriptions de leurs rôles ainsi que leurs fonctions grammaticales sont souvent exigées. Un exemple d'entrée d'un système basé grammaire est celui de SURGE[79] :



Qui génère la phrase : She hands the draft to the editor.

Comme les modèles de phrases, les systèmes basés grammaire nécessitent un énorme travail manuel. En particulier, il est difficile de prendre en compte le contexte en définissant les règles de choix entre les variantes possibles du texte résultat à partir des entrées[73].

Approches statistiques : Il existe plusieurs méthodes basées sur des statistiques pour la tâche de réalisation. Certains se basent sur des grammaires probabilistes, cette dernière a l'avantage de minimiser le travail manuel tout en couvrant plus de cas de réalisation. Il

existe principalement deux approches lutilisant[73] :

- La première se base sur une petite grammaire qui génère plusieurs alternatives qui sont ensuite ordonnés selon un modèle statistique basé sur un corpus pour sélectionner la phrase la plus probable (par exemple Langkilde-Geary (2000)[80]).
- La deuxième méthode utilise les informations statistiques directement au niveau de la génération pour produire la solution optimale (exemple : Belz (2008)[81]).

Dans les deux méthodes sus-citées la grammaire de base peut être manuellement faite. Dans ce cas, les informations statistiques aideront à la détermination de la solution optimale. Elle peut être aussi extraite à partir des données, comme lutilisation des Treebanks² pour déduire les règles de grammaire[82].

Dautres approches statistiques nutilisent pas des grammaires mais se basent sur des classificateurs. Ces derniers peuvent être cascades de telle sorte à décider quel constituant utiliser dans quelle position ainsi que les modifications nécessaires pour générer un texte correcte. À noter quune telle approche, ne nécessitant pas lutilisation de grammaire, utilise des entrées plus abstraites et moins détaillées linguistiquement. À voire même la possibilité de sétendre aux autres tâches de NLG, cest à dire un système qui accomplit plusieurs tâches de NLG en parallèle en utilisant les entrées initiales. Dans la suite de ce travail nous allons présenter certains de ces systèmes qui sont plus utilisés récemment.

3.7.7 Systèmes basés encodeur-décodeur

Une architecture souvent utilisée dans le traitement du langage naturel est lencodeur-décodeur. En particulier, son utilisation dans les tâches seq2seq3.2.2.3 ce qui permet de mettre en correspondance une séquence de taille variable en entrée avec une autre séquence en sortie. Les modèles seq2seq peuvent être adapter pour convertir une représentation abstraite de linformation en langage naturel[83].

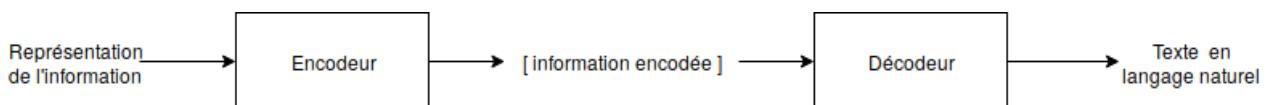


Figure 3.18 – Schéma d'une architecture encodeur-décodeur pour NLG

Beaucoup dapproches de génération de langage naturel en gestion de dialogue utilise des encodeur-décodeur. Wen et al. (2015)[84] utilise par exemple des LSTMs3.3) sémantiquement conditionnés ; il ajoute aux LSTMs classiques une couche contenant des informations sur laction prise par le gestionnaire de dialogue pour assurer que la génération représente le sens désiré. Dautres travaux utilisent des réseaux de neurones récurrent pour encoder létat du gestionnaire de dialogue et lentrée reçue suivie par un décodeur pour générer le texte de la réponse[85, 60, 86].

2. un Treebank est un texte analysé qui contient des informations syntaxiques ou sémantiques sur les structures de phrases

3.8 Conclusion

Au terme de ce chapitre et après avoir étudié l'état de l'art sur les systèmes existants, nous avons pu construire un bagage théorique assez complet dans le domaine des SPAs. En assimilant les différentes approches, il est maintenant temps de passer à la conception de notre propre système. Le chapitre suivant introduira nos propositions pour le développement de notre propre SPA (qui sera assez simpliste du coté opérationnel et applicatif au début) qui sera destiné à la manipulation d'un ordinateur.

Chapitre 4

Conception du système

4.1 Introduction

Durant ce chapitre, nous allons présenter en détail les étapes de conceptions de notre système, tout d'abord une architecture générale est présentée et décortiquée, ensuite chaque module du système sera détaillé du point de vue des composants qui le constituaient en donnant un maximum de détails tout en restant assez concis. Une conclusion viendra clôturer ce chapitre pour introduire ensuite le suivant.

4.2 Architecture du système

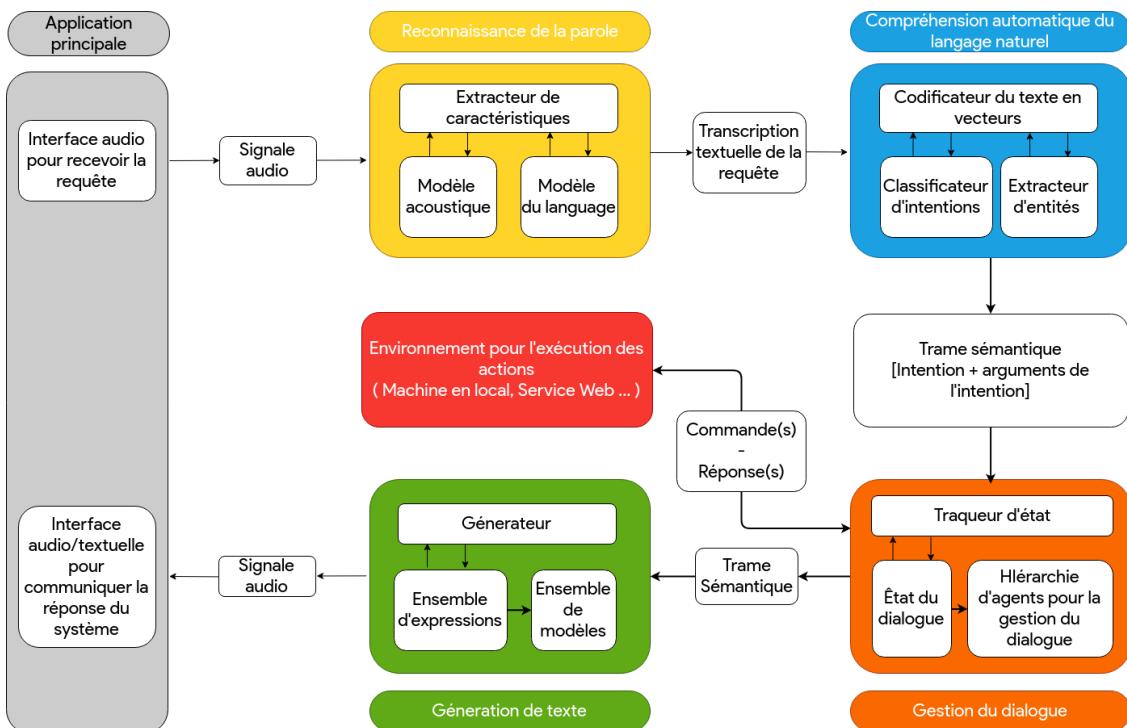


Figure 4.1 – Architecture générale de notre système

Comme montré dans la figure ci dessus (voir 4.1) et comme cité dans le chapitre précédent (voir 3.3) le système se présente comme l'interconnexion de cinq parties dont une interface¹ et quatre modules internes, chaque module forme ainsi un maillon d'une chaîne qui représente une partie du cycle de vie du système. L'architecture du système est une pipeline (chaîne de traitement) de processus qui s'exécutent de manière indépendante mais qui font circuler un flux de données entre eux dans un format préalablement décidé (voir 3.5). Nous pouvons séparer ces parties en deux catégories :

4.2.1 Couche utilisateur

Cette couche représente ce que l'utilisateur peut voir comme entrée/sortie et les interfaces qui lui sont accessibles, puisque l'assistant est un processus qui communique majoritairement avec l'utilisateur à travers des échanges verbaux, nous avons pensé à implémenter l'interface du système comme une processus qui s'exécute en arrière plan et qui attend d'être activer (pour le moment par un événement physique, c.à.d clique sur un bouton/icône ou raccourci clavier), l'assistant pourra ensuite répondre en affichant un texte à l'écran qui sera vocalement synthétisé envoyer à l'utilisateur via l'interface de sortie de son choix (afficher le texte et sa transcription vocale pourrait palier au manque d'un périphérique de sortie audio).

4.2.2 Couche système

Cette couche quant à elle représente ce que l'utilisateur ne voit pas et fait donc partie du fonctionnement interne du système, elle regroupe les quatre grandes étapes d'un cycle de vie pour une commande reçue de la couche utilisateur. Comme mentionné dans le chapitre précédent (voir 3.3), la requête passe par un module de reconnaissance de la parole, qui traduira en texte le signal audio correspondant à la requête, le module suivant va extraire l'intention de l'utilisateur et ses arguments (par exemple "*open the home folder*" pourrait donner une intention du type `open_file_desire[file_name="home",parent_directory=?]`), le gestionnaire de dialogue gardera trace de l'ensemble des échanges effectués entre l'utilisateur et l'assistant et essayera d'atteindre le but final formulé par la requête (la plus récente ou la plus ancienne), pour ce faire il aura besoin d'interagir avec ce qu'on a appelé un environnement d'exécution, qui peut être la machine où l'assistant réside ou une API² qui aura accès à un service à distance (sur internet par exemple) ou locale (dans un réseau domestique). Finalement une action spéciale qui servira à informer l'utilisateur sera envoyé au module suivant pour être transformé en son équivalent en langage naturel, puis le texte sera vocalement synthétisé et envoyé vers l'interface de sortie de l'application.

Nous allons maintenant détailler la conception des différents modules en précisant à chaque fois le ou les processus de sa mise en uvre.

1. Ici nous par interface nous entendons le sens abstrait du terme et non obligatoirement le sens interface graphique

2. Application programming interface ou interface de programmation applicative

4.3 Module de reconnaissance automatique de la parole

Premier module du système, il joue un rôle clé dans la véracité du dialogue entre l'utilisateur et la machine, en effet il devait être assez robuste et précise dans la transcription de la requête en entrée minimiser les erreurs et les ambiguïtés qui peuvent survenir dans le reste de la pipeline. C'est pour cela que nous avons décidé de ne pas développer un sous-système en partant de zéro, par faute de temps et par soucis de précision nous avons décidé de nous baser sur un outil open-source nommé DeepSpeech [87], naturellement du fait que ce soit un projet open-source nous avons pu avoir accès à différentes informations concernant le modèle d'apprentissage, d'inférence et la nature des données utilisé pour le premier et le teste du deuxième.

4.3.1 Architecture du module

Le module possède une architecture en pipeline dont chaque composant exécute un traitement sur la donnée reçue par son précurseur

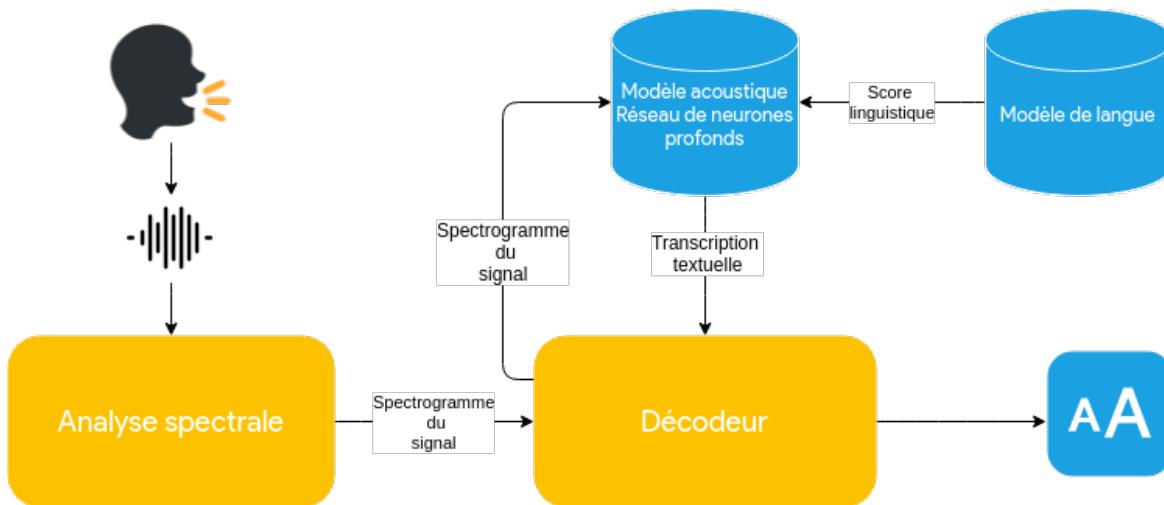


Figure 4.2 – Architecture de notre module de reconnaissance de la parole

4.3.2 Modèle acoustique

Type du modèle

Le modèle d'apprentissage (qui est principalement le modèle acoustique à l'exception d'une partie consacrée au modèle linguistique) possède une architecture en réseau de neurones avec apprentissage de bout-en-bout composé de trois parties :

- Deux couches de convolution spatiale : pour capturer les patrons dans la séquence du spectrogramme du signal audio.
- Sept couches de récurrence (Réseaux de neurones récurrents) pour analyser la séquence de patrons (ou caractéristiques) engendré par les couches de convolutions.

- Une couche de prédiction utilisant un réseau de neurone complètement connecté pour prédire le caractère correspondant à la fenêtre d'observation du spectrogramme du signal audio. La fonction d'erreur prend en compte la similarité du caractère produit avec le véritable caractère ainsi que la vraisemblance de la séquence produite par rapport à un modèle de langue basé sur les N-grammes (voir 3.2.2.5)

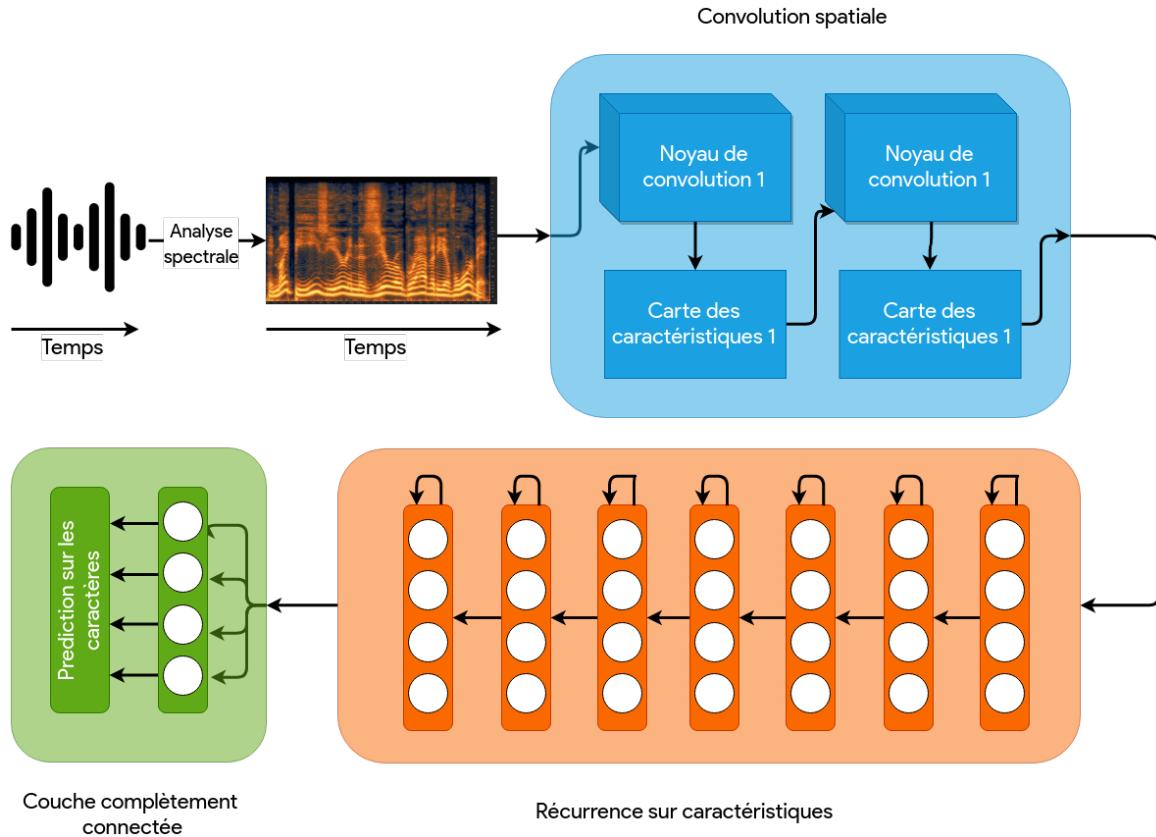


Figure 4.3 – Architecture du modèle DeepSpeech [87]

Données d'apprentissage

Pour entraîner le modèle acoustique, Mozilla a lancé le projet Common Voice³ une plateforme en ligne pour récolter des échantillons d'audio avec leurs transcriptions textuelles, chaque lot (batch) de données reçus est alors manuellement validé par l'équipe de Mozilla pour l'inclure dans la banque de données d'exemples. À ce jour plus et pour la langue anglaise, la plateforme a récolté plus de 22Go de données, soit 803 heures d'enregistrement appartenant à plus de 30 000 voix différentes dont 582 heures ont été validées. Mais ces données ne sont rien comparées à celle déjà utilisées pour l'apprentissage initial, en effet plusieurs sources ont été combinées pour construire cet ensemble de données, dans [87] il a été mentionné que trois ensembles d'apprentissage existants ont été utilisés dont WSJ (Wall Street Journal)⁴, Switchboard⁵ et Fisher⁶ qui cumulent 2380 heures d'enregistrement.

3. <https://voice.mozilla.org/fr>

4. <http://www.cstr.ed.ac.uk/corpora/MC-WSJ-AV/>

5. <https://catalog.ldc.upenn.edu/LDC97S62>

6. <https://catalog.ldc.upenn.edu/LDC2004S13>

ments audio en anglais et plus de 27 000 voix différentes, vient s'ajouter à cela l'ensemble Baidu⁷ avec 5000 heures d'enregistrements et 9600 locuteurs.

4.3.3 Modèle de la langue

Type du modèle

Pour ce qui est du type du modèle de langue, c'est un modèle basé sur les N-grammes (3-grammes pour être plus précis) qui est utilisé, en effet il permet de façon assez simple et intuitive de capturer l'enchaînement des mots dans une langue donnée, rendant ainsi la transcription assez proche de la façon dont les mots sont distribués dans le corpus d'apprentissage.

Données d'apprentissage

À l'origine, DeepSpeech utilise un modèle de langue dont la source n'est pas dévoilée par les chercheurs dans [87], mais son volume est approximativement de 220 million de phrases avec 495 000 mots différents. Cependant puisque ce corpus nous reste inconnu et qu'il a probablement été construit pour reconnaître des séquence de mots de phrase en anglais assez générales, nous avons décidé de construire notre propre modèle de langue en récoltant des données depuis des dépôts sur le site Github, plus précisément les fichiers README.md des dépôts qui font office de manuels d'utilisation d'un projet hébergé sur le site, ce type de fichier renferment généralement des instructions de manipulation de fichiers, de lancement de commandes ..., ce qui offre un bon corpus pour le modèle de langue, car en effet notre système se concentre plus sur l'aspect de manipulation d'un ordinateur, donc la probabilité de trouver certaines séquences de mots qui appartiennent au domaine du Tech est plus élevé en théorie. La procédure suivie est la suivante :

7. <https://ai.baidu.com/broad/introduction>

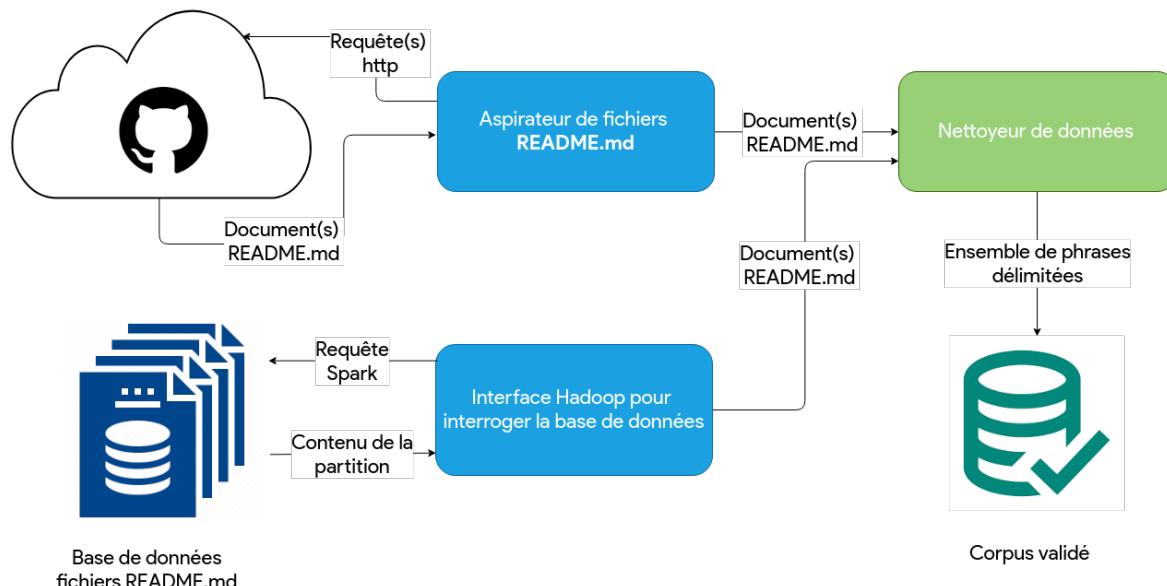


Figure 4.4 – Processus de génération du corpus pour le modèle de langue

- L'acquisition des données dans leur format brut .md (markdown) se fait de deux manières :
 - Depuis le site officiel de GitHub en faisant des requêtes http au serveur en suivant le patron suivant des urls :

`'http://raw.githubusercontent.com/' +NOM_DÉPOT+ '/master/README.md'`

La liste des noms de dépôt est disponible dans un fichier⁸ en free open acces au format .csv dont les colonnes sont *Nom_Utilisateur* et *Nom_Dépot*

- En lisant une base de 16 millions de fichiers différents dont la taille totale atteint 4.5 Go
- Les deux sources de données envoient ensuite les fichiers récoltés au nettoyeur de fichiers pour en extraire seulement les parties qui ont du sens (c.à.d paragraphes, titres, instructions ...).
- Le corpus final est ensuite construit à partir des paragraphes extraits à l'étape précédente après les avoir segmenté en phrases (en utilisant un modèle de segmentation prédéfini) donnant le format suivant

```
<s>Phrase1</s>
<s>Phrase2</s>
...
<s>PhraseN</s>
```

4.4 Module de compréhension automatique du langage naturel

Second module du système, son rôle et de faire office de couche d'abstraction entre la requête de l'utilisateur formulée dans un langage naturel et le fonctionnement interne du système qui lui comprend (et parle) un langage plus formel, on parle ici de la construction d'un représentation sémantique de la requête. Pour ce faire nous avons opté pour l'approche par apprentissage automatique, compte tenu des bons résultats obtenus par certaines architectures [47],[46] et ceux malgré le petit de nombre de données d'apprentissage, cette option nous paru plus abordable que la construction d'un analyseur basé sur règles, qui sont souvent assez rigide.

8. https://data.world/vmarkovtsev/github-readme-files/file/top_broken.tsv

4.4.1 Architecture du module

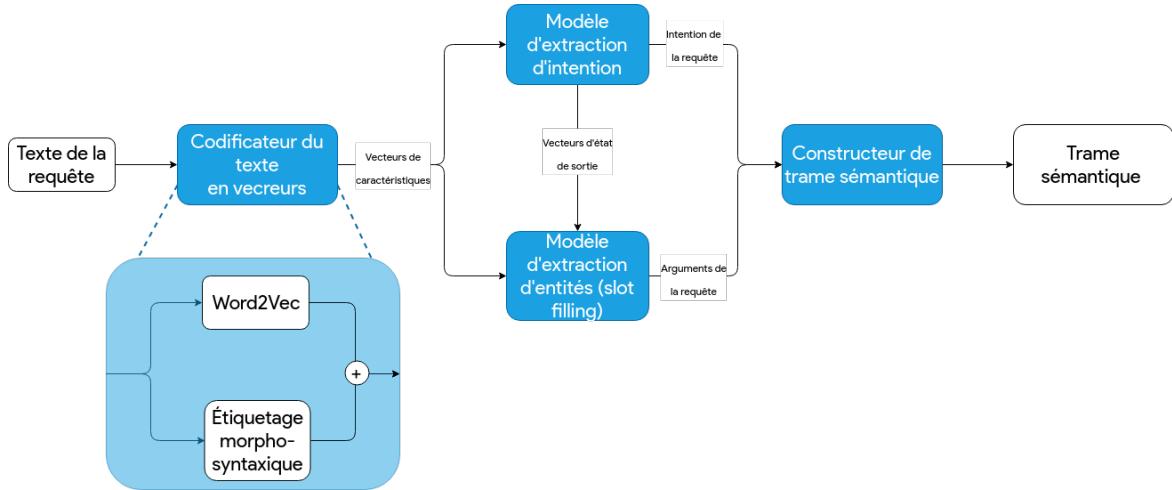


Figure 4.5 – Architecture du module compréhension automatique du langage naturel

Comme précédemment cité (voir 4.2.2), le module possède une architecture en pipeline qui reçoit en entrée le texte brut de la requête, sa codification varie selon les approches que nous avons exploré et qui seront plus expliciter dans le chapitre suivant Réalisation et Expérimentations, pour mieux capturer l'aspect sémantique des mots dans le texte, nous avons décidé d'utiliser un modèle pré-entraîné par Google de Word2Vec (entraîné sur 100 milliard de mots) pour produire un vecteur de taille fixe pour chaque mots, pour encoder l'information syntaxique de la requête nous avons concaténé au vecteur de prolongement de chaque mot de la requête (Word Embedding Vector) le vecteur codifiant son étiquette morphosyntaxique. Après avoir codifié la séquence de mots, elle est envoyée aux modèles de classification d'intentions et d'extraction d'entités⁹, qui sont en fait un seul modèle joint dont l'architecture est détailler dans 4.4.2.1. Ces deux informations sont ensuite décodées et passées au constructeur de trame sémantique qui structurera ces dernières en une seule entité sémantique dans le format suivant : donnant le format suivant

```
{
    intent :"open_file_desire",
    entities:[
        {
            entity :"string",
            name :"file_name",
            value :"tes.py",
            start :"31",
            end   :"37",
        }
    ]
}
```

9. Par entité nous entendons les arguments de l'intention

4.4.2 Analyse sémantique avec apprentissage automatique

4.4.2.1 Modèle(s) utilisé

Comme vu dans le chapitre précédent (voir 3.5) l'architecture adopté est une architecture mono-entrée/multi-sorties dont l'entrée est une séquence de mots codifiés et les sorties sont une séquence d'étiquettes et une classe associé au texte. Nous pouvons distinguer les deux parties qui sont l'encodage et le décodage de la séquence, l'encodage sert à la fois à l'attribution de la classe (l'intention) et à l'initialisation de la séquence de décodage (pour l'attribution de son étiquette à chaque mot).

L'encodage se fait en utilisant un réseau de neurones récurrent de type BLSTM (Bidirectionnel Long Short Term Memory) pour mieux capturer le contexte droit (respectivement gauche) de chaque entrée, le dernier vecteur en sortie est ensuite utilisé comme vecteur d'entrée pour un réseau de neurones Fully Connected (Complètement connecté) dont la dernière couche est une couche de prédiction sur une distribution de probabilités des intentions possibles. Ce dernier vecteur sert aussi de d'état initial au décodeur qui est aussi un réseau de neurones récurrent de type BLSTM, à chaque étape de l'inférence une étiquette est produite en sortie pour chaque position du texte en entrée (les longueurs des séquences d'entrée et de sortie sont donc égales) en utilisant un autre réseau de neurone Fully Connected sur chaque vecteur d'état de sortie des cellules LSTM du décodeur (voir 3.8)

4.4.2.2 Les données d'apprentissage

Ne disposant pas d'un ensemble d'apprentissage pré-existant pour les intentions que nous avons développé, nous avons tenté d'en construire un nous même en l'enrichissant avec quelque modifications. Dans [88] il a été noté que pour une tâche assez simple (comme pour notre cas l'exploration des fichiers dans un premier temps) une grande quantité de données n'est pas nécessaire (une cinquantaine d'exemples par intentions approximativement) si les exemples ne sont pas facilement confondus, surtout si l'espace des possibilité pour les requête est assez réduit et peut facilement être expliciter. En jouant sur l'ordre des mots nous avons pu générer pour les XX intentions YYY patrons d'exemple au total (TOTAL PEUT ENCORE ÊTRE CHANGÉ 361 pour l'instant), un patron d'exemple une structure contenant des placeholders (compartiment) pouvant être rempli avec des valeurs générées programmatiquement, par exemple :

```
delete the {file_name:} file under {parent_directory:}
```

Ces placeholders servent à la fois à générer plus d'exemples mais aussi à étiqueter le texte en choisissant les valeurs de ces variables comme valeur de l'étiquette, un exemple d'un entrée de l'ensemble d'apprentissage avant affectation des variables est le suivant :

```
{
  "id": 6,
  "text": "I want to open the {file_name:} folder",
  "intent": "open_file_desire"
},
```

Pour remplir l'ensemble des placeholders, nous commençons d'abord par scanner le répertoire de la machine avec une profondeur max égale à 5. Les noms des répertoires sont donc nettoyé à l'aide d'expressions régulière et transformé en un format universel établi à l'avance **nom_du_fichier** en choisissant "_" comme séparateur, en bouclant sur ces noms de répertoire nous pourrons donc construire plusieurs exemples comme une entré dans un dictionnaire dont le format est le suivant :

```
{
  'id': 79372,
  'intent': 'close_file_desire',
  'postags': ['NN', 'VB', 'DT', 'NN', 'VBN', 'NN', 'NNS'],
  'tags': 'NUL NUL NUL NUL NUL file_name file_name',
  'text': 'please close the file named platform notifications'
}
```

4.5 Module de gestion du dialogue

Le but de ce module est de décider quelle action à prendre à chaque instant du dialogue. D'abord nous allons présenter l'architecture globale de ce module notamment la représentation des informations reçues et la politique d'action. Ensuite, nous allons détailler la conception de chaque partie.

4.5.1 Architecture du module

Comme nous avons déjà vu, l'architecture typique des gestionnaires de dialogue se compose de deux parties principales :

- Un module qui suit l'état du dialogue : Pour gérer le dialogue avec l'utilisateur, le gestionnaire doit représenter l'état du dialogue de façon à pouvoir répondre aux actions de l'utilisateur. Ce module sert à suivre cet état après chaque étape du dialogue.
- Une politique d'action : Celle-ci détermine l'action à prendre à partir d'un état donné.

4.5.1.1 État du dialogue

Avant de détailler les deux modules du gestionnaire. Il est nécessaire d'introduire une représentation de l'état du dialogue. Classiquement, les trames sémantiques sont utilisés 3.6.2. Le suivi d'état se fait dans ce cas en gardant trace des emplacements remplis durant

le dialogue 3.6.2.1. Nous avons opter à utiliser une représentation plus riche; les graphes de connaissances sont une forme de représentation où les connaissances sont décrites sous forme d'un graphe orienté étiqueté. Des travaux ont déjà utilisé des graphes de connaissances[89] et des ontologies[90] pour représenter l'état du système de dialogue. Celui-là est suivi d'une base de règles pour décider l'action à prendre directement du graphe de connaissances. L'avantage par rapport à l'utilisation des trames sémantiques apparaît dans la flexibilité et le dynamisme des graphes de connaissances. En effet, pour une tâche comme la navigation dans les fichiers, l'état de l'arborescence des fichiers est sujet à des changements fréquents : ajout, suppression, modification, etc. Il est difficile de faire une représentation de l'information dans ce cas avec de simples emplacements à remplir.

4.5.1.2 Le suivi de l'état du dialogue

Le rôle du premier module est de mettre à jour l'état du système au cours du dialogue. Il reçoit l'action de l'utilisateur ou du gestionnaire et il produit un nouvel état comme le montre la figure4.5.1.2.

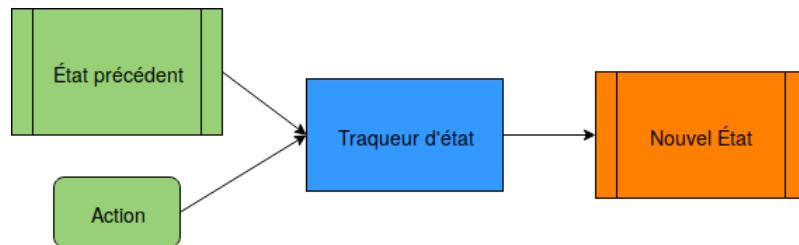


Figure 4.6 – Schéma du traqueur d'état

Dans notre cas, le module NLU produit toujours un trame sémantique à partir du texte contenant l'intention de l'utilisateur ainsi que ses paramètres. C'est alors le travail du traqueur d'état d'injecter le résultat du NLU dans le graphe de connaissances. Ceci consiste à transformer le trame sémantique en un graphe selon des règles de transformation qui est ensuite ajouté au graphe d'état. Plus de détails seront donnés dans la section suivante4.5.2 où nous construirons une ontologie pour définir un vocabulaire de dialogue et comment elle peut être utilisé pour passer du résultat du NLU en un graphe de connaissances.

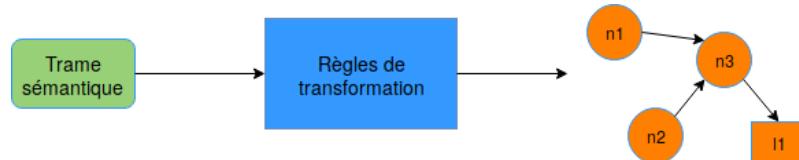


Figure 4.7 – Schéma de transformation de trame sémqntique en graphe

4.5.1.3 La politique d'action

La politique d'action peut être écrite manuellement ou apprise à partir d'un corpus ou en utilisant l'apprentissage par renforcement. Dans ce dernier cas, un agent doit interagir avec un utilisateur qui évalue ses performances afin qu'il puisse apprendre. Étant donné que l'apprentissage par renforcement nécessite un nombre important d'interactions, il est primordial d'utiliser un simulateur d'utilisateur. Ce dernier peut être basé règles, ou un modèle statistique extrait à partir d'un corpus de dialogue.

Dans les trois cas de figure, il est difficile de réaliser un modèle varié et qui peut accomplir plusieurs tâches. D'un coté, un corpus contenant des dialogues sur toutes les tâches possibles, si ces derniers sont nombreux et spécifiques à une application précise, est difficile à acquérir. De l'autre coté, écrire les règles d'un système de dialogue ou d'un simulateur d'utilisateur s'avère compliqué et nécessite un travail manuel énorme pour gérer toutes les tâches possibles.

Pour pallier à cela, nous proposons d'utiliser une architecture multi-agents hiérarchique. Dans laquelle, les agents feuilles sont des agents qui peuvent répondre à une tâche ou une sous tâche bien précise. Tandis que les agents parents sélectionnent l'agent fils capable de répondre à l'intention de l'utilisateur.

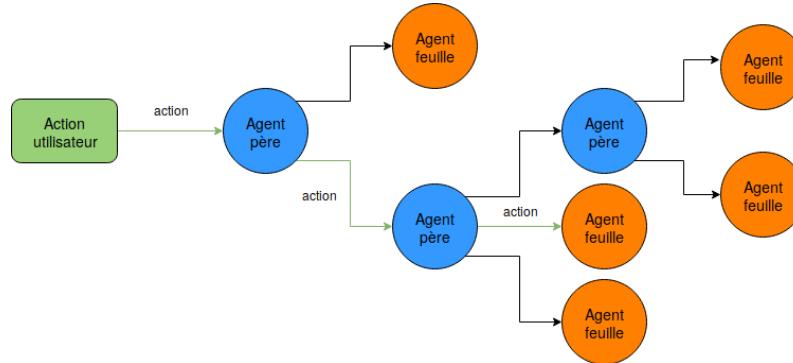


Figure 4.8 – Schéma de l'architecture multi-agents

L'avantage d'une telle architecture est de permettre la division du problème en plusieurs sous problèmes indépendants. En effet, un simulateur d'utilisateur ou un corpus qui est destiné pour une seule tâche est considérablement plus abordable. Cependant, un travail supplémentaire s'avère nécessaire qui est celui des agents parents. Celui-ci est relativement simple, il suffit de faire un apprentissage supervisé des agents parents avec les simulateurs d'utilisateurs des agents fils. à tour de rôle et avec des probabilités de transitions entre les simulateurs d'utilisateurs, ces derniers communiquent avec l'agent parent. Comme on connaît pour chaque simulateur l'agent fils qui lui correspond, il est donc possible de faire un apprentissage supervisé où les entrées sont les actions des simulateurs et l'état du système, tandis que la sortie est l'agent fils qui peut répondre à l'action.

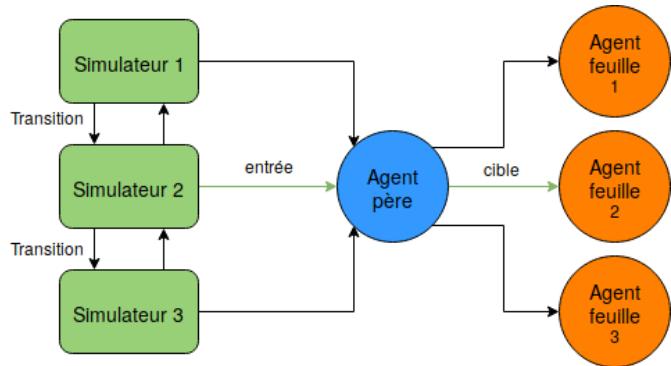


Figure 4.9 – Schéma représentant l'apprentissage des agents parents avec les simulateurs des agents feuilles

Pour résumer l'architecture globale du gestionnaire de dialogue, lorsque une nouvelle action utilisateur arrive au système, le traqueur d'état la reçoit. Il met à jour l'état du système en transformant l'action en un graphe de connaissances pour l'ajouter au graphe d'état. Ce nouvel graphe d'état ainsi que la dernière action reçue sont transmis à une architecture multi-agents hiérarchique qui va décider quelle action le système de dialogue doit prendre.

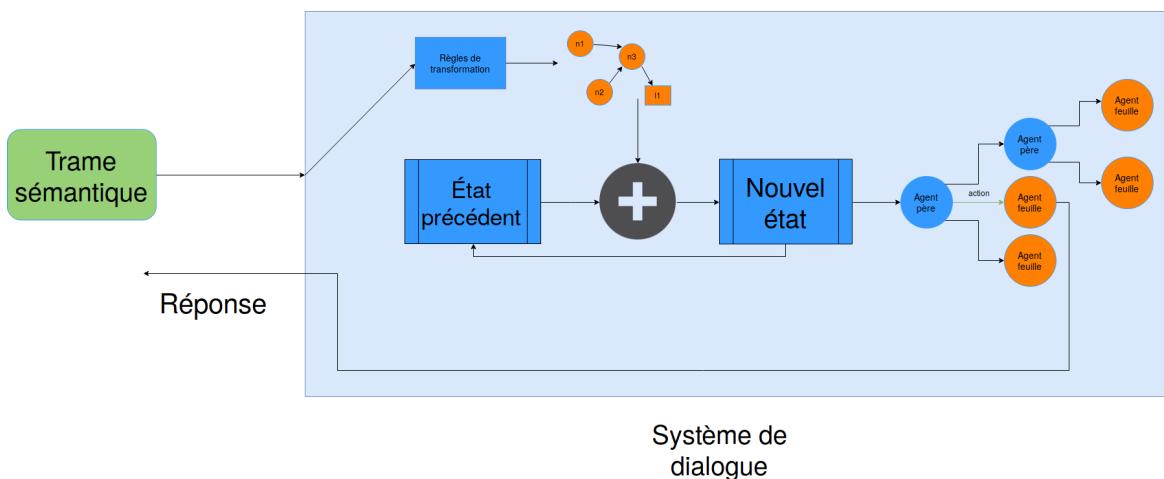


Figure 4.10 – Schéma global du gestionnaire de dialogue

4.5.2 Les ontologies du système

Une ontologie est une représentation des concepts et les relations d'un domaine donné, elle permet de définir un vocabulaire pour ce domaine afin que les programmes intelligents puissent comprendre et communiquer sur des données reliées à ce domaine.

Nous définissons une ontologie de dialogue ainsi que des ontologies pour chaque tâche réalisable par notre assistant. Ce qui permettra à notre gestionnaire de comprendre le dialogue et les tâches qu'il peut réaliser.

4.5.2.1 Ontologie de dialogue

D'abord on définit une ontologie de dialogue qui contient des concepts qui peuvent aider un assistant d'ordinateur pour gérer son dialogue.

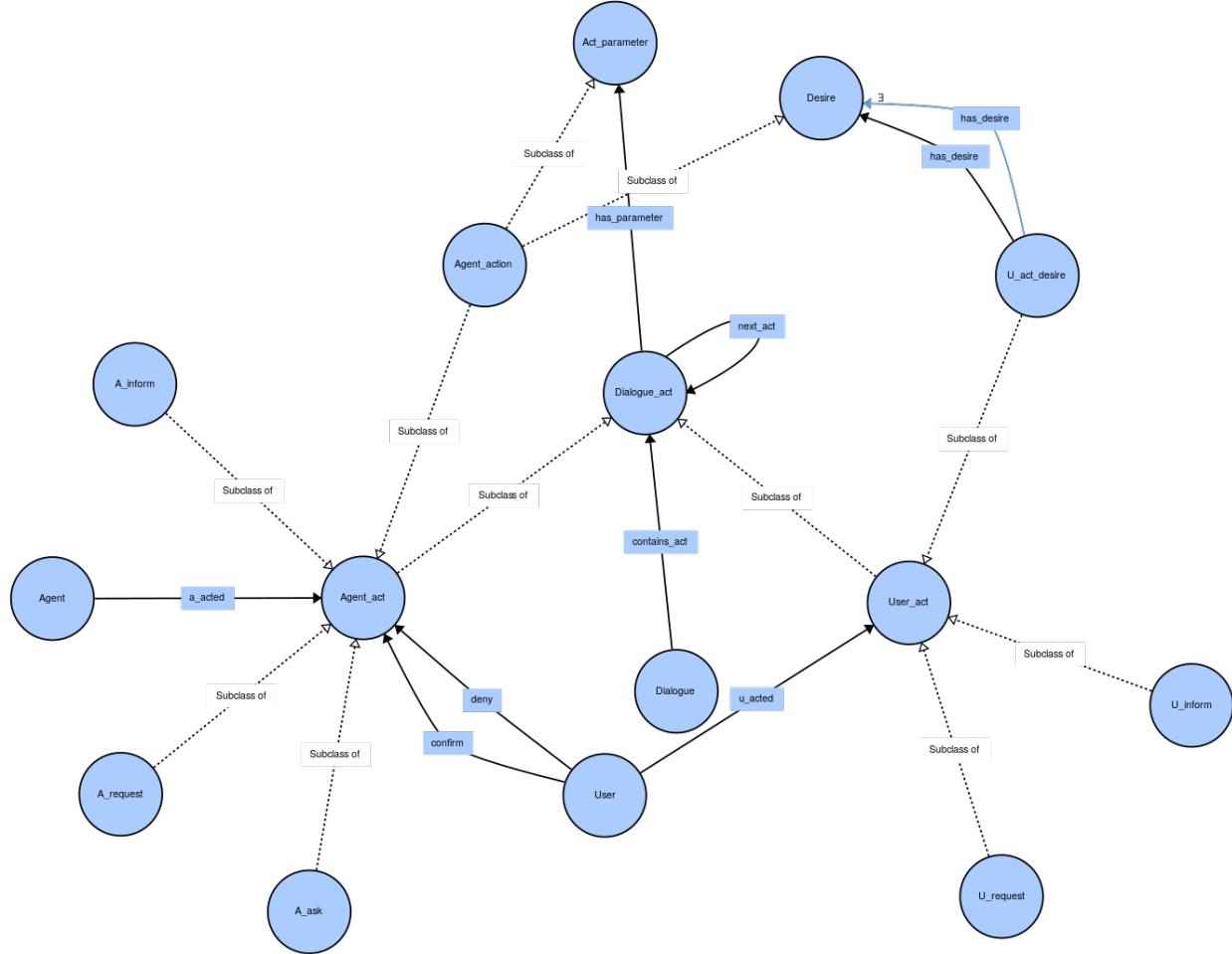


Figure 4.11 – Graphe de l'ontologie de dialogue

Principalement l'ontologie se compose de six classes mères :

- *Agent* et *User* : ce sont les classes qui représentent l'utilisateur et l'agent qui participent au dialogue.
- *Dialogue* : l'agent et l'utilisateur participe à un dialogue, ce dernier contient les actions des deux cotés.
- *Dialogue_act* : la classe qui représente une action du dialogue, elle a deux sous-classes *Agent_act* et *User_act* qui représentent les actions de l'agent et de l'utilisateur respectivement.
- *Act_parameter* : C'est la classe mère des paramètres que peuvent prendre les actions de dialogue. Par exemple, l'action d'informer peut avoir en paramètre le nom d'un fichier.
- *Act_desire* : C'est la classe mère des actions de l'agent que l'utilisateur peut demander. Par exemple, il peut demander l'ouverture d'un fichier donné.

Le reste des classes sont des classes filles qui détaillent plus les concepts du dialogue agent-utilisateur.

À l'arrivée d'une nouvelle action, le traqueur d'état va créer le graphe correspondant. Un exemple abstrait de cela est représenté dans la figure 4.5.2.1. Une nouvelle action utilisateur est créée ainsi que ses paramètres et les relations entre eux.

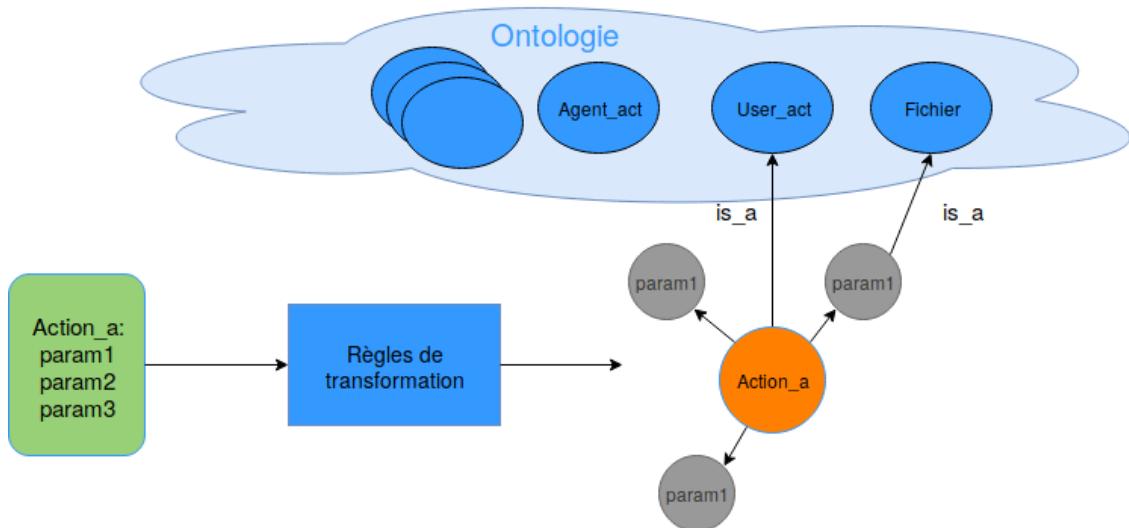


Figure 4.12 – Schéma de transformation d'une action en graphe

Ontologie pour l'exploration de fichiers

Un exemple d'ontologie pour la compréhension d'une tâche réalisable par l'assistant est celle de l'exploration de fichiers.

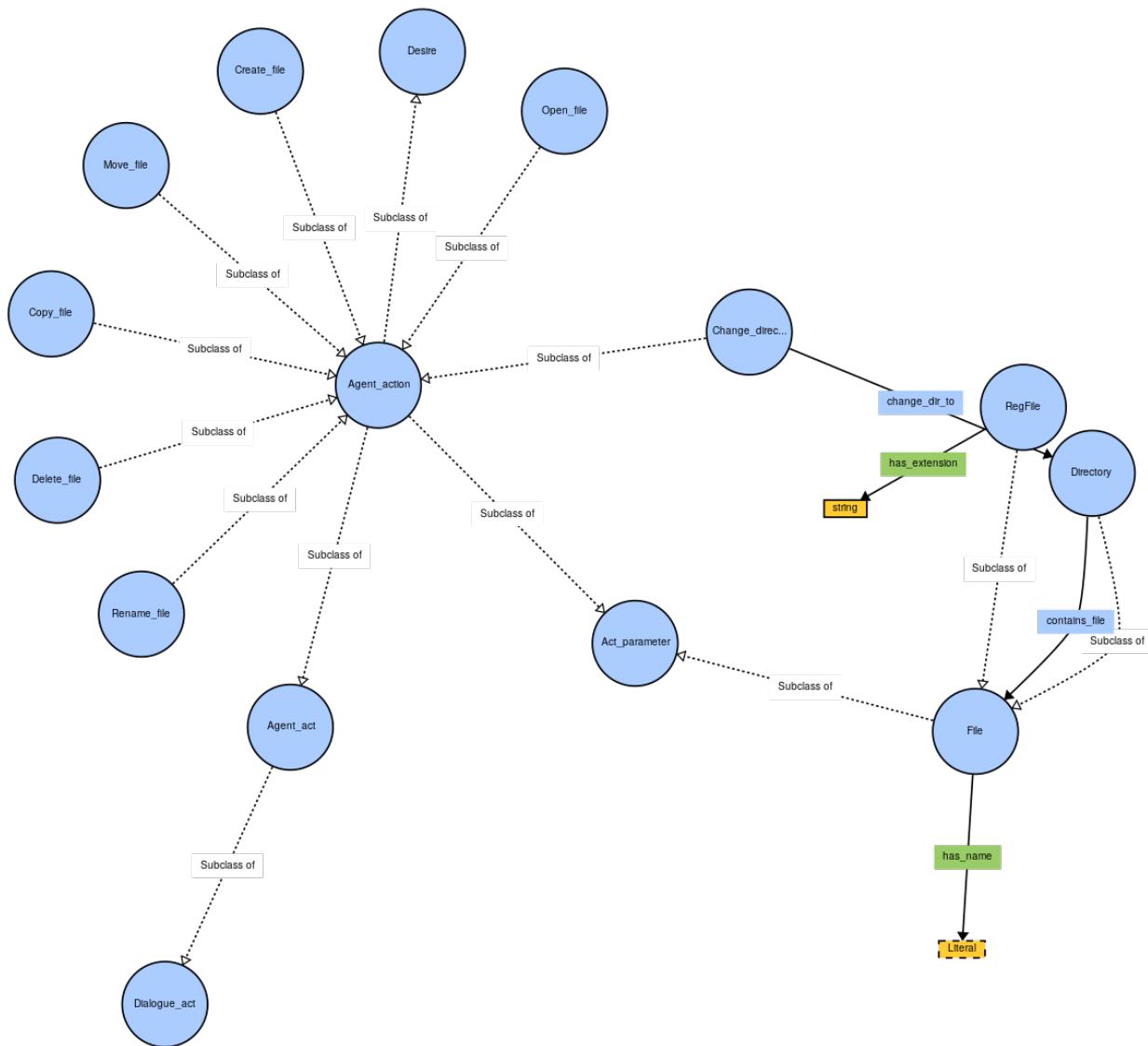


Figure 4.13 – Graphe de l'ontologie de l'exploration de fichiers

L'ontologie contient essentiellement :

- Des actions sur les fichiers : Créer un fichier, supprimer un fichier, changer de répertoire, etc. Ces actions sont des sous-classes de la classe *Agent_act* vu précédemment ainsi que les classes *Act_parameter* et *Act_desire*. Ce qui veut dire que ces actions peuvent être des paramètres d'autres actions comme demander à l'utilisateur s'il veut que l'assistant réalise une action donnée et que l'utilisateur peut demander à l'assistant de faire une de ces actions.
- Les concepts qui ont relation avec l'exploration de fichiers sont des sous-classes de *Act_parameter* étant donné qu'ils peuvent être des paramètres d'actions, par exemple l'action d'ouvrir un fichier a comme paramètre un fichier.
- Des relations entre ces concepts sont aussi définies comme un répertoire peut contenir des fichiers, ou une action de changement de répertoire doit avoir comme paramètre un répertoire cible.

La figure suivante 4.5.2.1 représente l'arrivée d'une nouvelle action : à créer un fichier nommé travail. L'action de l'utilisateur est donc représentée par un nouveau nud de type *U_act_desire* qui désigne une action utilisateur qui demande une action de l'assistant. Cette dernière a comme paramètre un nud de type *Create_file* qui est l'action de l'agent que l'utilisateur veut réaliser. Cette action à son tour a des paramètres comme, dans ce cas, le fichier qu'on veut créer.

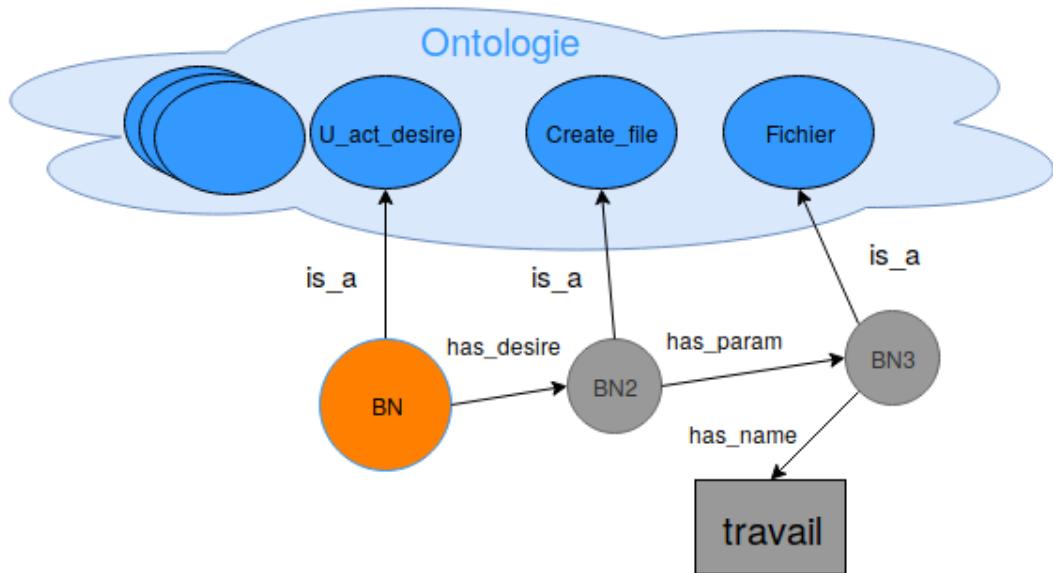


Figure 4.14 – Schéma de transformation d'une action de demande de création de fichier en graphe

Les autres actions sont créées de façon similaire avec des règles à suivre lorsqu'une nouvelle action arrive.

4.5.3 Les simulateurs d'utilisateurs

Plusieurs méthodes peuvent être utilisées pour la création de simulateurs d'utilisateurs 3.6.3.3. Les simulateurs basés sur des méthodes d'apprentissage sont les plus robustes. Cependant, ils nécessitent un nombre important de données. L'alternative c'est d'utiliser des simulateurs basés règles. Nous nous sommes inspirés des simulateurs basés agenda [64] qui sont des variantes des simulateurs basés règles pour créer nos propres simulateurs. Leur fonctionnement est simple, Ils commencent par générer un but. Pour y arriver, une agenda est créée, celle-ci contient les informations que doit convoyer le simulateur ainsi que les informations qu'il doit recevoir. Les actions sont sélectionnées en suivant des probabilités conditionnelles sur l'état de l'agenda. Enfin, les récompenses sont en fonction des informations reçues de l'agent.

Simulateur pour l'exploration de fichiers

L'exploration de fichiers ne dépend pas de simples informations à transmettre et d'autres à recevoir comme dans les cas d'envoyer un e-mail, chercher une information sur internet ou bien lancer de la musique. Il s'agit d'une tâche dynamique dont la situation de départ est variante. Pareillement, le nombres d'actions change d'un état à un autre. Effectivement, le nombre de fichiers qu'on peut supprimer ou le nombre de répertoires qu'on peut y accéder n'est pas fixe par exemple. D'abord une arborescence aléatoire est générée qui représente la situation initiale du système. Ensuite, le simulateur duplique cette arborescence en y introduisant des modifications pour générer une arborescence but. Enfin, le simulateur essaye de guider l'agent pour arriver au but en utilisant les actions utilisateurs possibles. En addition des actions de création et suppression de fichiers ainsi que les changements de répertoires qui peuvent guider l'agent au but. D'autres sous-but peuvent être créés suivant une distribution de probabilité comme copier ou couper un fichier, renommer un fichier, ouvrir un fichier etc. Dans ce cas le simulateur donne la priorité aux sous-but avant de reprendre les actions menant au but final.

L'algorithme suivi par le simulateur est le suivant :

Algorithm 1 Algorithme simulateur

```
1 : procedure Step(entrées : action_agent; sorties : recompense, fin, succès)
2 :   tour ← tour+1
3 :   if tour > max_tour then
4 :     fin ← true
5 :     succès ← échec
6 :     réponse_utilisateur ← reponse_fin()
7 :   else
8 :     succès ← update_state(action_agent)
9 :     if succès then
10 :       fin ← true
11 :       réponse_utilisateur ← reponse_fin()
12 :     else
13 :       réponse_utilisateur ← decider_action(action_agent)
14 :     recompense ← calculer_recompense(action_agent, succès)
15 :   return réponse_utilisateur
```

- **lignes 3-6** : cas échec on met *fin* à *vrai* et on retourne une réponse de fin.
- **ligne 8** : la fonction **update_state** met à jour l'état du simulateur et vérifie si on arrive à un état de succès.
- **lignes 9-11** : cas succès on met *fin* à *vrai* et on retourne une réponse de fin.
- **ligne 13** : le simulateur décide quelle action à prendre selon l'action de l'agent.

Quant à la décision de l'action à prendre, elle est résumée par le diagramme suivant :

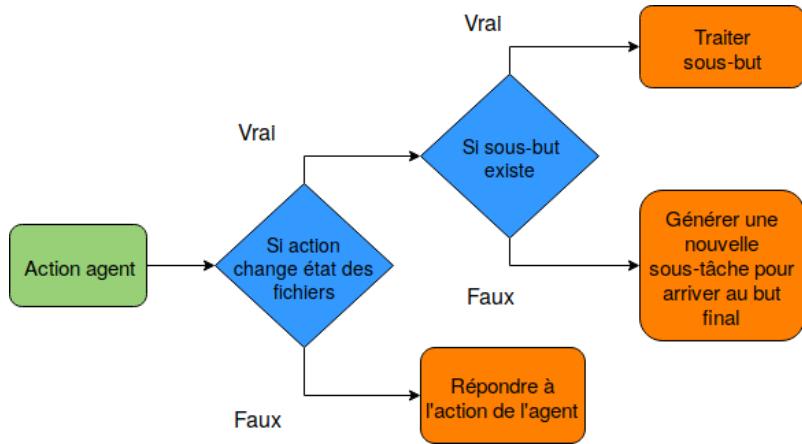


Figure 4.15 – Diagramme de décision de l'action à prendre

En ce qui concerne la fonction de récompense, celle-ci est évaluée à partir des changements effectués sur l'état de l'utilisateur, le tableau suivant ?? associe les changements avec leurs récompenses :

Événements	Récompenses
Similarité améliorée	2
Succès	2
Sous-but réalisé	2
Similarité diminuée	-3
Confirmation d'une question	0
Autre	-1

Table 4.1 – Tableau des récompenses

- **similarité améliorée** : On calcule la similarité entre l'arborescence courante et l'arborescence but. La valeur de la similarité est égale à n_sim/n_diff avec :
 - n_sim : nombre de fichiers qui existent dans les deux arborescences.
 - n_diff : nombre de fichiers qui n'existent que dans une des deux arborescences.
 Si cette valeur s'améliore, on donne une récompense positive à l'agent.
- **Succès** : c'est à dire, réaliser le but final du simulateur. Pour arriver à cet événement, l'agent ne fait qu'améliorer la similarité, c'est pourquoi les deux événements ont la même valeur de récompense.
- **Sous-but réalisé** : c'est la récompense donnée quand l'agent arrive au sous-but du simulateur.
- **Similarité diminuée** : la récompense est dans ce cas négative et supérieure en valeur absolue à celle que l'agent obtient lorsqu'il améliore la similarité. Ce choix a pour but d'éviter que l'agent boucle sur des actions dont la somme des récompenses est supérieure ou égale à zéro. Par exemple il peut créer ensuite supprimer le même fichier, si la somme de ces deux actions est supérieur ou égale à zéro, l'agent peut boucler sur ces actions indéfiniment sans pour autant recevoir des récompenses négatives.

- **Confirmation d'une question** : l'utilisateur peut confirmer une action à l'agent. La récompense est nul pour que l'agent puisse demander une confirmation quand il n'est pas sûr de ce qu'il doit faire sans diminuer le cumul des récompenses reçues.
- **Autre** : La récompense est de -1 pour éviter que le dialogue dure longtemps.

Pour résumer le fonctionnement du simulateur, à l'arrivée d'une nouvelle action agent, celle-ci met à jour l'état du simulateur. L'état du simulateur se compose de deux parties : un simulateur d'arborescence de fichiers qui simule l'état de l'arborescence courant, et des variables d'état qui contiennent d'autres informations comme l'état des sous-but, le fichier en cours de traitement, les informations reçues de l'agent, le répertoire courant, etc. Après la mise à jour de l'état, si l'action de l'agent nécessite une réponse immédiate comme la demande d'une information ou la permission d'exécuter une action, celle-ci est traité directement, sinon le simulateur initie le traitement d'une nouvelle sous-tâche. C'est à dire, Si un sous-but existe, une action qui le traite est générée, sinon une action qui traite le but final est générée.

4.5.4 Modèles d'apprentissage

Comme on l'a déjà cité dans le chapitre précédent, il existe plusieurs algorithmes d'apprentissage par renforcement comme Q-Learning ou State-Action-Reward-State-Action (SARSA)[91]. Cependant ces algorithmes, en essayant d'estimer la fonction Q de récompense, traitent le problème comme un tableau état/action et essayent d'estimer pour chaque état et action la récompense résultante. Ceci implique que ces algorithmes ne peuvent pas estimer la fonction de récompense pour des état qu'ils n'ont pas vu pendant l'apprentissage. Pour pallier à ce problème, Deep Q Learning (DQL)[63] utilise un réseau de neurones comme estimateur de la fonction Q. Ce qui lui permet d'avoir une notion de similarité entre les états; ainsi il peut estimer la récompense pour des états jamais vus auparavant.

Encodeur de graphe

La flexibilité des graphes les rend difficile à introduire dans un réseau de neurones vu que ce dernier n'accepte que des entrées de tailles fixes. Des méthodes ont été utilisées pour introduire les graphes dans des réseaux de neurones notamment les convolutions sur les graphes avec Graph Convolution Networks (GCN)[92] qui s'avère être des variantes des Gated Graph Neural Networks (GGNN)[93]. Ce dernier utilise des réseaux de neurones récurrent entre chaque deux nuds reliés par une arête pour transférer l'information d'un nud à un autre, ce qui résulte en des vecteurs ayant un encodage de l'information associé à chaque nud. Cette étape est répétée k fois pour que chaque nud aie des informations des nuds qui sont à un maximum de k pas de distance, avec k un paramètre empirique. Enfin, les vecteurs d'états de chaque nud sont sommés pour produire un vecteur fixe encodant tout le graphe qui peut être relié au reste du réseau de neurones pour l'apprentissage.

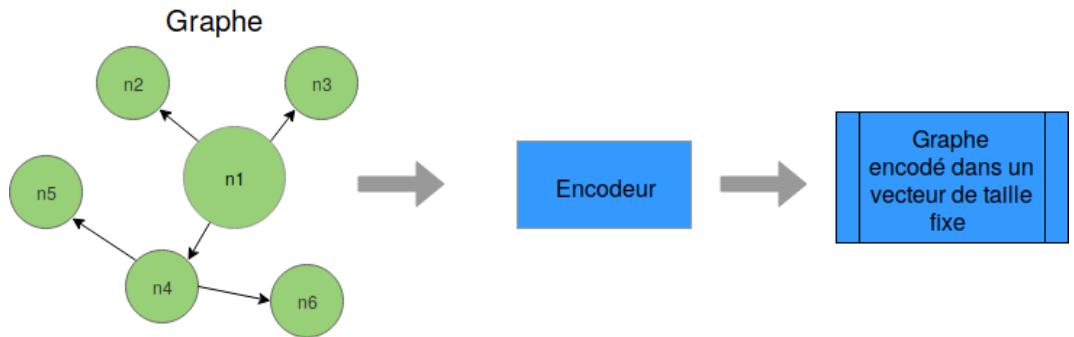


Figure 4.16 – Schéma représentant un encodeur de graphe

Ces méthodes nécessitent tout le graphe pour l'encoder. Cependant, dans notre cas, après chaque action, le graphe augmente de taille ce qui nécessite de refaire l'encodage dès le début. Nous proposons de traiter le graphe comme une séquence de triplets : nānudă; arcă; nudăz ou násujetă; predicată; objetaž comme dans le framework Resource Description Framework (RDF). L'encodage se fait avec une architecture encodeur-décodeur basé sur des réseaux de neurones récurrent (RNN). Ainsi, à l'arriver de nouveaux triplets, il suffit d'utiliser l'état précédent pour y encoder les nouveaux triplets.

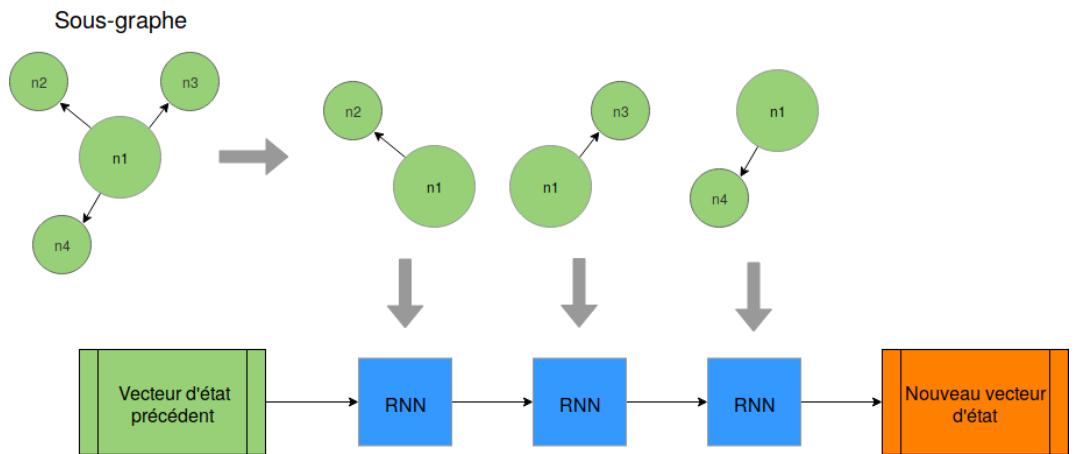


Figure 4.17 – Schéma représentant un encodeur séquentiel de graphe

Pour faire l'apprentissage de cet architecture, il est possible de générer des graphes aléatoirement qu'on fait passer triplet par triplet dans l'encodeur. Celui-ci est un RNN qui prend en entrée l'état précédent du réseau et un triplet du graphe et qui produit en sortie un nouvel état. L'état final du RNN, après avoir fait passer tous les triplets du graphe, est utilisé dans le décodeur qui est un autre RNN. Ce dernier essaye de reconstruire le graphe triplet par triplet à partir de l'état reçu comme sortie de l'encodeur. Ainsi, si on peut reconstruire le graphe, on peut dire que le dernier état encode tout le graphe dans un vecteur de taille fixe.

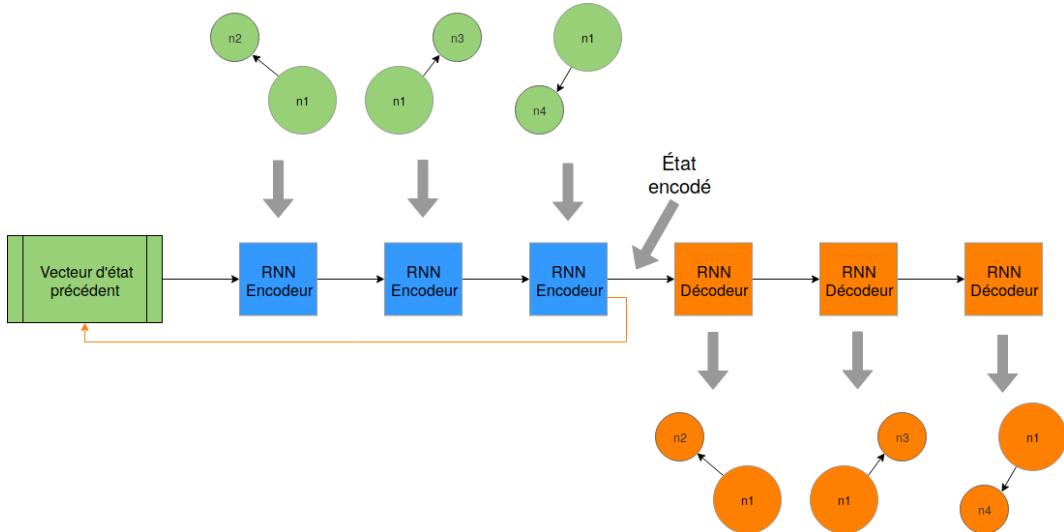


Figure 4.18 – Schéma de l'apprentissage d'un encodeur séquentiel de graphe

Les agents feuilles

Comme nous en avons parlé précédemment, les agents feuilles sont les agents responsables de répondre aux intentions de l'utilisateur. Pour faire l'apprentissage par renforcement d'un agent feuille, on utilise le simulateur d'utilisateur comme environnement de cet agent. Ce dernier interagit avec le simulateur pour but d'estimer la fonction de récompense en fonction de son état. On utilise pour cela un réseau de neurones profond qui prend en entrée l'état encodé de l'agent, c'est à dire le graphe encodé, et il produit pour chaque action la récompense correspondante. Comme le nombre d'actions est variable, il est impossible d'utiliser une architecture de réseau de neurones qui produit une sortie pour chaque action, où chaque sortie est la récompense de l'action qui y correspond. Par conséquent, il est nécessaire de donner au réseau, en addition de l'état encodé, l'action candidate.

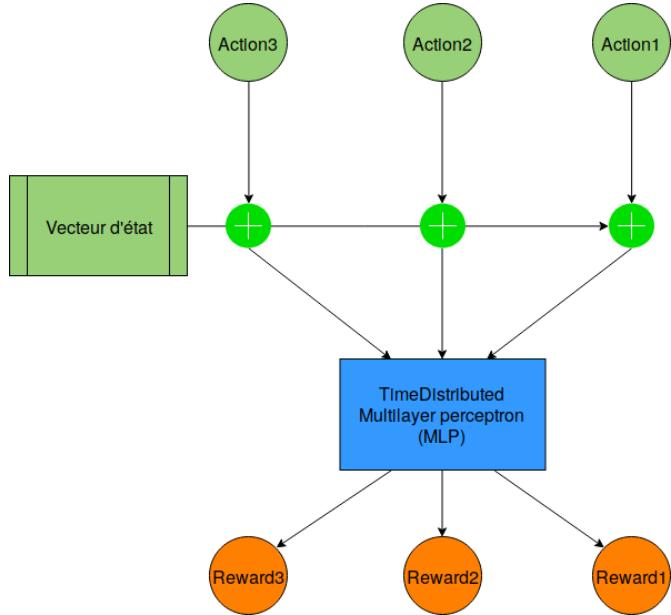


Figure 4.19 – Schéma du réseau DQN

L'algorithme utilisé pour l'apprentissage par renforcement est double DQN en rejouant l'expérience. En addition de l'utilisation des réseaux de neurones pour estimer la fonction Q, deux améliorations ont été ajoutées :

- rejouer l'expérience : l'agent interagit avec le simulateur plusieurs fois en gardant dans une mémoire ses interactions. Après chaque k épisode¹⁰ l'agent reprend la mémoire pour entraîner le réseau de neurones.
- Double DQN : la fonction Q est donnée par la formule $Q(s, a) = r + \alpha * \max_j(Q(s', a_j))$ avec :
 - s : état de l'agent.
 - a : l'action qu'on veut estimer.
 - R : récompense immédiate.
 - α : paramétrage de réduction.
 - s' : nouvel état après avoir effectué l'action a de l'état s .
 - a_j : les actions possibles à partir de l'état s' .

On remarque la récurrence dans cette formule qui nécessite la réutilisation du réseau pour estimer le terme $\max_j(Q(s', a_j))$. Il a été démontré que l'utilisation d'un autre réseau qu'on fixe lors de l'apprentissage pour l'évaluation de la récompense dans ce terme améliore les résultats[63]. La formule devient donc : $Q(s, a) = r + \alpha * Q(s', \text{argmax}_{a_j}(s', a_j))$. Cette valeur est donc utilisée pour calculer l'erreur et appliquer l'algorithme de retro-propagation pour l'apprentissage automatique.

Une autre architecture possible serait de relier l'encodeur de graphe directement avec le réseau DQN pendant l'apprentissage. Ainsi l'erreur de l'apprentissage pour l'encodeur est calculée à partir de la fonction de récompense de l'apprentissage par renforcement. L'avantage de relier l'encodeur avec le réseau de DQN est de permettre à l'encodeur de

10. un épisode est un ensemble d'interactions agent-simulateur jusqu'à finir avec un succès ou échec

contrôler quelle partie du graphe encodé à oublié. En effet, la taille fixe du vecteur dont on encode le graphe a une limite de nombre de triplets supportable. L'utilisation des cellules de réseaux de neurones récurrent comme les LSTMs ou GRUs qui ont des porte d'oubli rend cette architecture possible.

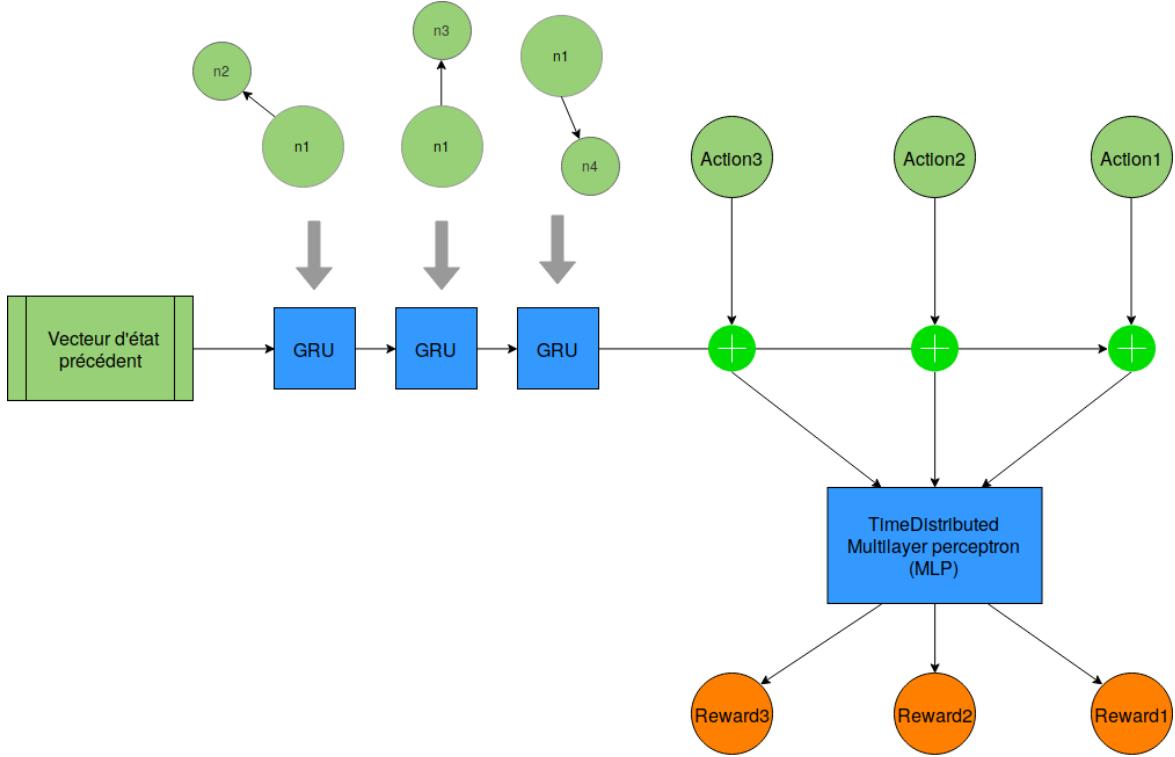


Figure 4.20 – Schéma du réseau DQN relié avec l'encodeur directement

L'agent coordinateur

L'agent coordinateur utilise la même architecture que celle des agents feuilles. La différence se trouve au niveau de la sortie. Dans le cas des agents coordinateurs, ils essayent de prédire quelle agent fils peut répondre à la dernière action reçue.

4.6 Module de génération du langage naturel

Le modèle utilisé pour la génération du texte est relativement simple. Il s'agit de préparer des modèles de phrases contenant des emplacements à remplir. Chaque action de l'agent lui correspond un ensemble de modèles et chaque paramètre de l'action lui correspond un ensemble d'expressions. La génération du texte se fait en choisissant d'abord pour chaque paramètre de l'action une expression aléatoirement. Ensuite, de même, un modèle de phrase est choisi aléatoirement. Enfin, les emplacements vides sont remplis avec les expressions des paramètres. La figure suivante illustre un exemple de la génération de texte en utilisant les modèles de phrases.

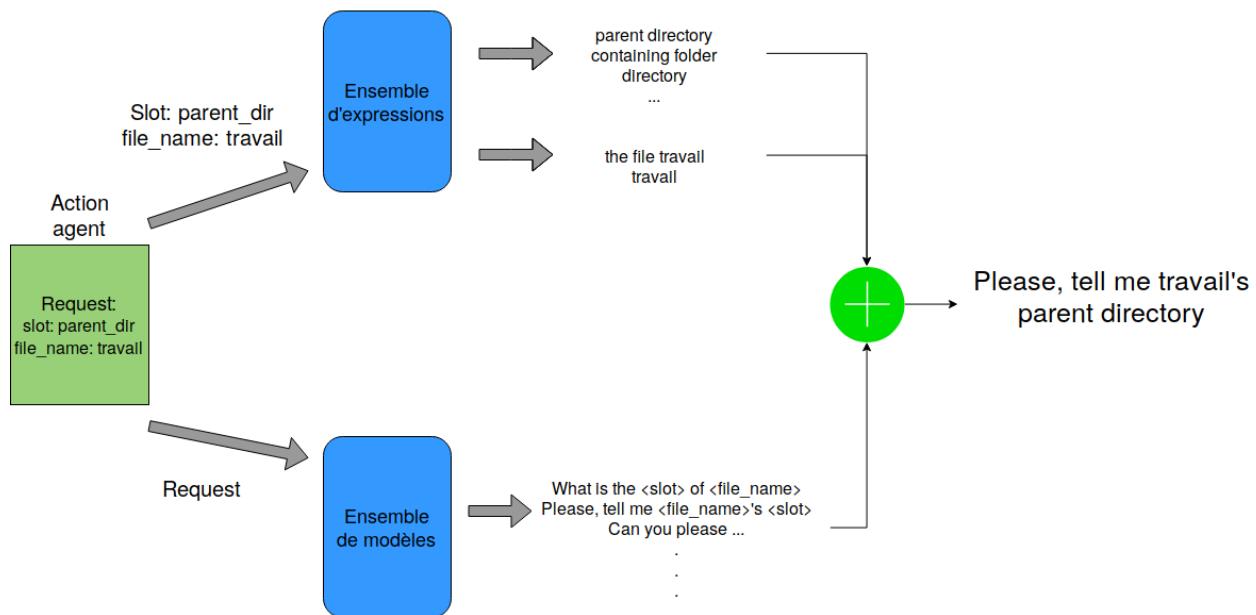


Figure 4.21 – Schéma de fonctionnement du générateur de texte

4.7 Conclusion

Table des figures

2.1 Conversation aléatoire avec Google Assistant	10
2.2 Requête simple formulée à Google Assistant	10
2.3 Google duplex réservant une place dans un salon de coiffure	11
2.4 Intégration aux applications [12]	11
2.5 Service paiement 1 [12]	12
2.6 Siri sur un laptop [13]	12
3.1 Architecture basique d'un réseaux de neurones multicouches	16
3.2 Architecture interne d'un réseau de neurones récurrent à un instant t [27]	17
3.3 Architecture interne d'une cellule mémoire dans un réseau LSTM [27]	18
3.4 Exemple d'une distribution de probabilité de transition ainsi qu'une distribution de probabilité d'observations pour un HMM à 3 états et 2 observations	20
3.5 Schéma abstractif d'un SPA [36]	21
3.6 Architecture des systèmes de reconnaissance de la parole [20]	22
3.7 Architecture de base d'un classificateur d'intents [46]	25
3.8 Architecture de base d'un classificateur d'intents doublé d'un extracteur d'entité [47]	26
3.9 Exemple d'une trame sémantique pour une requête donnée	26
3.10 Schéma général d'un gestionnaire de dialogue	27
3.11 Schéma représentant les transitions entre états dans un MDP	28
3.12 Schéma représentant un cadre sémantique avec comme domaine : création de fichier	28
3.13 Schéma représentant la mise-à-jour de l'état par un système basé règles	29
3.14 Schéma représentant la mise-à-jour de l'état par un système basé statistiques	29
3.15 Diagramme d'influence dans un POMDP	30
3.16 Schéma de gestion de dialogue de bout en bout avec architecture Seq2Seq	31
3.17 Schéma d'interaction agent-environnement dans l'apprentissage par renforcement	32
3.18 Schéma d'une architecture encodeur-décodeur pour NLG	36
4.1 Architecture générale de notre système	38
4.2 Architecture de notre module de reconnaissance de la parole	40
4.3 Architecture du modèle DeepSpeech [87]	41
4.4 Processus de génération du corpus pour le modèle de langue	43
4.5 Architecture du module compréhension automatique du langage naturel	45
4.6 Schéma du traqueur d'état	48

4.7 Schéma de transformation de trame séquentielle en graphe	48
4.8 Schéma de l'architecture multi-agents	49
4.9 Schéma représentant l'apprentissage des agents parents avec les simulateurs des agents feuilles	50
4.10 Schéma global du gestionnaire de dialogue	50
4.11 Graphe de l'ontologie de dialogue	51
4.12 Schéma de transformation d'une action en graphe	52
4.13 Graphe de l'ontologie de l'exploration de fichiers	53
4.14 Schéma de transformation d'une action de demande de création de fichier en graphe	54
4.15 Diagramme de décision de l'action à prendre	56
4.16 Schéma représentant un encodeur de graphe	58
4.17 Schéma représentant un encodeur séquentiel de graphe	58
4.18 Schéma de l'apprentissage d'un encodeur séquentiel de graphe	59
4.19 Schéma du réseau DQN	60
4.20 Schéma du réseau DQN relié avec l'encodeur directement	61
4.21 Schéma de fonctionnement du générateur de texte	62
4.22 A generated graph with TKIZ using alpha version of the TeX exporter of Web-VOWL (version 1.1.3)	63

Bibliographie

- [1] D. A. Norman, *The design of everyday things*. New York : Basic Books, 2002.
- [2] R. Knote, A. Janson, L. Eigenbrod, and M. Söllner, "The what and how of smart personal assistants : Principles and application domains for is research," in *Multikonferenz Wirtschaftsinformatik (MKWI)*, 2018.
- [3] H. Gellersen, *Handheld and Ubiquitous Computing : First International Symposium, HUC'99, Karlsruhe, Germany, September 27-29, 1999, Proceedings (Lecture Notes in Computer Science)*. Springer, 1999.
- [4] S. J. Russell and P. Norvig, *Artificial Intelligence : A Modern Approach*. Pearson Education, 2 ed., 2003.
- [5] T. Dingler, "Cognition-aware systems as mobile personal assistants," in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing Adjunct - UbiComp '16*, ACM Press, 2016.
- [6] E. Luger and A. Sellen, ""like having a really bad PA"," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*, ACM Press, 2016.
- [7] R. Trappi, ed., *Your Virtual Butler*. Springer Berlin Heidelberg, 2013.
- [8] A. Purington, J. G. Taft, S. Sannon, N. N. Bazarova, and S. H. Taylor, ""alexa is my new BFF", in *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '17*, ACM Press, 2017.
- [9] B. R. Cowan, N. Pantidi, D. Coyle, K. Morrissey, P. Clarke, S. Al-Shehri, D. Earley, and N. Bandeira, ""what can i help you with? ", in *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services - MobileHCI '17*, ACM Press, 2017.
- [10] J. Imtiaz, N. Koch, H. Flatt, J. Jasperneite, M. Voit, and F. van de Camp, "A flexible context-aware assistance system for industrial applications using camera based localization," in *Proceedings of the 2014 IEEE Emerging Technology and Factory Automation (ETFA)*, IEEE, sep 2014.
- [11] A. Janson and M. T. de Gafenco, "Engaging the appropriation of technology-mediated learning services - a theory-driven design approach," in *ECIS*, 2015.
- [12] "Apple shares examples of siri's third-party app integration on ios 10." <https://www.idownloadblog.com/2016/09/01/apple-siri-ios-10-app-integration/>. (Accessed on 10/29/2018).
- [13] "macos sierra review : Hey siri, where did my files go? - six colors." <https://sixcolors.com/post/2016/09/sierra-review/>, 2016. (Accessed on 10/29/2018).

- [14] T. M Mitchell, "The discipline of machine learning," 01 2006.
- [15] S. Kotsiantis, I. Zaharakis, and P. Pintelas, "Machine learning : A review of classification and combining techniques," *Artificial Intelligence Review*, vol. 26, pp. 159–190, 11 2006.
- [16] H. Barlow, "Unsupervised learning," *Neural Computation*, vol. 1, no. 3, pp. 295–311, 1989.
- [17] B. Sonali, "Research paper on basic of artificial neural network," p. 1, 2014.
- [18] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015.
- [19] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6645–6649, May 2013.
- [20] D. Yu and L. Deng, *Automatic Speech Recognition*. Springer London, 2015.
- [21] M. Velay and F. Daniel, "Seq2seq and multi-task learning for joint intent and content extraction for domain specific interpreters," *CoRR*, vol. abs/1808.00423, pp. 3–4, 2018.
- [22] P. Liu, X. Qiu, and X. Huang, "Recurrent neural network for text classification with multi-task learning," *CoRR*, vol. abs/1605.05101, 2016.
- [23] F. Rosenblatt, *Perceptron simulation experiments (Project Para)*. 1959.
- [24] F. Murtagh, "Multilayer perceptrons for classification and regression," *Neurocomputing*, vol. 2, pp. 183–197, jul 1991.
- [25] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.
- [26] Z. C. Lipton, "A critical review of recurrent neural networks for sequence learning," *CoRR*, vol. abs/1506.00019, 2015.
- [27] C. Olah, "Understanding lstm networks – colah's blog." <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>, Août 2015. (Accessed on 02/19/2019).
- [28] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, vol. 6, pp. 107–116, Apr. 1998.
- [29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.
- [30] D. Jurafsky and J. H. Martin, *Speech and Language Processing, 2nd Edition*. Prentice Hall, 2008.
- [31] Z. Ghahramani, "Hidden markov models," ch. An Introduction to Hidden Markov Models and Bayesian Networks, pp. 9–42, River Edge, NJ, USA : World Scientific Publishing Co., Inc., 2002.
- [32] J. G. Kemeny and J. Laurie Snell, "Markov processes in learning theory," *Psychometrika*, vol. 22, pp. 221–230, Sep 1957.
- [33] R. Rabiner and B. H. Juang, "An introduction to hidden markov models," *IEEE ASSP Magazine*, vol. 3, pp. 4–16, 1986.

- [34] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, pp. 268–278, March 1973.
- [35] J. Bloit and X. Rodet, "Short-time viterbi for online hmm decoding : Evaluation on a real-time phone recognition task," pp. 2121 – 2124, 05 2008.
- [36] P. Milhorat, S. Schlögl, G. Chollet, B. , A. Esposito, and G. Pelosi, "Building the next generation of personal digital assistants," 03 2014.
- [37] F. Al-Anzi and D. AbuZeina, "Literature survey of arabic speech recognition," pp. 1–6, 03 2018.
- [38] S. Narang and M. D. Gupta, "Speech feature extraction techniques : A review," 2015.
- [39] P. M. Chauhan and N. P. Desai, "Mel frequency cepstral coefficients (mfcc) based speaker identification in noisy environment using wiener filter," in *2014 International Conference on Green Computing Communication and Electrical Engineering (ICGCEE)*, pp. 1–5, March 2014.
- [40] D. D. O'Shaughnessy, "Linear predictive coding," *IEEE Potentials*, vol. 7, pp. 29–32, 1988.
- [41] H. Rahali, Z. Hajaiej, and N. Ellouze, "Robust features for speech recognition using temporal filtering technique in the presence of impulsive noise," *International Journal of Image, Graphics and Signal Processing*, vol. 6, pp. 17–24, 10 2014.
- [42] W. Ghai and N. Singh, "Literature review on automatic speech recognition," *International Journal of Computer Applications*, vol. 41, pp. 42–50, March 2012. Full text available.
- [43] I. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications : An overview," pp. 8599–8603, 10 2013.
- [44] R. Glass and M. Kyle Mccandless, "Automatic acquisition of language models for speech recognition," 10 1994.
- [45] B. Roark, M. Saracclar, and M. Collins, "Discriminative n-gram language modeling," *Comput. Speech Lang.*, vol. 21, pp. 373–392, Apr. 2007.
- [46] B. Liu and I. Lane, "Attention-based recurrent neural network models for joint intent detection and slot filling," *CoRR*, vol. abs/1609.01454, 2016.
- [47] C.-W. Goo, G. Gao, Y.-K. Hsu, C.-L. Huo, T.-C. Chen, K.-W. Hsu, and Y.-N. Chen, "Slot-gated modeling for joint slot filling and intent prediction," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 2 (Short Papers)*, (New Orleans, Louisiana), pp. 753–757, Association for Computational Linguistics, June 2018.
- [48] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks," *Trans. Sig. Proc.*, vol. 45, pp. 2673–2681, Nov. 1997.
- [49] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," *CoRR*, vol. abs/1506.07503, 2015.
- [50] Y. Wang, Y. Shen, and H. Jin, "A bi-model based rnn semantic frame parsing model for intent detection and slot filling," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 2 (Short Papers)*, (New Orleans, Louisiana), pp. 309–314, Association for Computational Linguistics, June 2018.

- [51] t. . A. M. D. P. Richard Bellman" *Indiana Univ. Math. J.*", *fjournal = "Indiana University Mathematics Journal*, vol. 6, pp. 679–684, 1957.
- [52] H. Chen, X. Liu, D. Yin, and J. Tang, "A survey on dialogue systems : Recent advances and new frontiers," *SIGKDD Explor. NewsL.*, vol. 19, pp. 25–35, Nov. 2017.
- [53] D. Goddeau, H. Meng, J. Polifroni, S. Seneff, and S. Busayapongchai, "A form-based dialogue manager for spoken language applications," vol. 2, pp. 701 – 704, 11 1996.
- [54] S. Young, M. Gašić, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu, "The hidden information state model : A practical framework for pomdp-based spoken dialogue management," *Comput. Speech Lang.*, vol. 24, pp. 150–174, Apr. 2010.
- [55] K. J. Åström, "Optimal control of Markov Processes with incomplete state information," *Journal of Mathematical Analysis and Applications*, vol. 10, pp. 174–205, January 1965.
- [56] M. Henderson, B. Thomson, and S. J. Young, "Deep neural network approach for the dialog state tracking challenge," in *SIGDIAL Conference*, pp. 467–471, 2013.
- [57] C. Lee, S. Jung, K. Kim, D. Lee, and G. G. Lee, "Recent approaches to dialog management for spoken dialog systems," *JCSE*, vol. 4, pp. 1–22, 2010.
- [58] J. Henderson, O. Lemon, and K. Georgila, "Hybrid reinforcement/supervised learning of dialogue policies from fixed data sets," *Comput. Linguist.*, vol. 34, pp. 487–511, Dec. 2008.
- [59] T.-H. Wen, M. Gasic, N. Mrksic, L. M. Rojas-Barahona, P. hao Su, S. Ultes, D. Van-dyke, and S. J. Young, "A network-based end-to-end trainable task-oriented dialogue system," in *EACL*, pp. 438–449, 2017.
- [60] I. V. Serban, A. Sordoni, Y. Bengio, A. Courville, and J. Pineau, "Building end-to-end dialogue systems using generative hierarchical neural network models," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI, pp. 3776–3783, AAAI Press, 2016.
- [61] G. Weisz, P. Budzianowski, P. hao Su, and M. Gax0161ix0107, "Sample efficient deep reinforcement learning for dialogue systems with large action spaces," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, pp. 2083–2097, 2018.
- [62] D. Bertsekas, *Dynamic Programming & Optimal Control, Vol II : Approximate Dynamic Programming*. Athena Scientific, 01 2012.
- [63] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.
- [64] J. Schatzmann, B. Thomson, K. Weilhammer, H. Ye, and S. J. Young, "Agenda-based user simulation for bootstrapping a pomdp dialogue system," in *HLT-NAACL*, pp. 149–152, 2007.
- [65] K. Georgila, J. Henderson, and O. Lemon, "Learning user simulations for information state update dialogue systems," in *INTERSPEECH*, pp. 893–896, 2005.

- [66] H. Cuayáhuitl, S. Renals, O. Lemon, and H. Shimodaira, "Human-computer dialogue simulation using hidden markov models," *IEEE Workshop on Automatic Speech Recognition and Understanding, 2005.*, pp. 290–295, 2005.
- [67] S. Chandramohan, M. Geist, F. Lefèvre, and O. Pietquin, "User simulation in dialogue systems using inverse reinforcement learning," in *INTERSPEECH*, p. 10251028, 2011.
- [68] R. Evans, P. Piwek, and L. Cahill, "What is nlg ?," in *Proceedings of the Second International Conference on Natural Language Generation*, pp. 144–151, 2002.
- [69] E. Reiter and R. Dale, "Building applied natural language generation systems," *Natural Language Engineering*, vol. 3, pp. 57–87, Mar. 1997.
- [70] C. Labb  and F. Portet, "Towards an abstractive opinion summarisation of multiple reviews in the tourism domain," *CEUR Workshop Proceedings*, vol. 917, pp. 87–94, 01 2012.
- [71] N. Dethlefs, "Context-sensitive natural language generation : From knowledge-driven to data-driven techniques," *Language and Linguistics Compass*, vol. 8, pp. 99–115, 03 2014.
- [72] J. Yu, E. Reiter, J. Hunter, and C. Mellish, "Choosing the content of textual summaries of large time-series data sets," *Natural Language Engineering*, vol. 13, pp. 25–49, Mar. 2007.
- [73] A. Gatt and E. Krahmer, "Survey of the state of the art in natural language generation : Core tasks, applications and evaluation," *J. Artif. Int. Res.*, vol. 61, pp. 65–170, Jan. 2018.
- [74] M. Theune, E. Klabbers, J. R. De Pijper, E. Krahmer, and J. Odijk, "From data to speech : A general approach," *Natural Language Engineering*, vol. 7, pp. 47–86, Mar. 2001.
- [75] G. Angeli, C. D. Manning, and D. Jurafsky, "Parsing time : Learning to interpret time expressions," in *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, NAACL HLT '12*, (Stroudsburg, PA, USA), pp. 446–455, Association for Computational Linguistics, 2012.
- [76] M. A. K. Halliday and C. M. I. M. Matthiessen, *Introduction to Functional Grammar*. London : Hodder Arnold, 3 ed., 2004.
- [77] W. C. Mann and C. M. I. M. Matthiessen, "Nigel : A systemic grammar for text generation.," 1983.
- [78] J. A. Bateman, "Enabling technology for multilingual natural language generation : The kpml development environment," *Nat. Lang. Eng.*, vol. 3, pp. 15–55, Mar. 1997.
- [79] M. Elhadad and J. Robin, "An overview of surge : a reusable comprehensive syntactic realization component," in *International Natural Language Generation Workshop*, pp. 1–4, 1996.
- [80] I. Langkilde-Geary, "Forest-based statistical sentence generation," in *ANLP*, pp. 170–177, 2000.
- [81] A. Belz, "Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models," *Nat. Lang. Eng.*, vol. 14, pp. 431–455, Oct. 2008.

- [82] D. Espinosa, M. White, and D. Mehay, "Hypertagging : Supertagging for surface realization with ccg," in *ACL*, pp. 183–191, 2008.
- [83] T. C. Ferreira, I. Calixto, S. Wubben, and E. Krahmer, "Linguistic realisation as machine translation : Comparing different mt models for amr-to-text generation," in *INLG*, pp. 1–10, 2017.
- [84] T.-H. Wen, M. Gasic, N. Mrksic, P. hao Su, D. Vandyke, and S. J. Young, "Semantically conditioned lstm-based natural language generation for spoken dialogue systems," in *EMNLP*, pp. 1711–1721, 2015.
- [85] A. Sordoni, M. Galley, M. Auli, C. Brockett, Y. Ji, M. Mitchell, J.-Y. Nie, J. Gao, and W. B. Dolan, "A neural network approach to context-sensitive generation of conversational responses," in *HLT-NAACL*, pp. 196–205, 2015.
- [86] R. Goyal, M. Dymetman, and E. Gaussier, "Natural language generation through character-based rnns with finite-state prior knowledge," in *COLING*, pp. 1083–1092, 2016.
- [87] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, *et al.*, "Deep speech : Scaling up end-to-end speech recognition," *arXiv preprint arXiv:1412.5567*, 2014.
- [88] T. Bocklisch, J. Faulkner, N. Pawlowski, and A. Nichol, "Rasa : Open source language understanding and dialogue management," *CoRR*, vol. abs/1712.05181, 2017.
- [89] S. Stoyanchev and M. Johnston, "Knowledge-graph driven information state approach to dialog," in *AAAI Workshops*, 2018.
- [90] M. Wessel, G. Acharya, J. Carpenter, and M. Yin, *OntoVPAAn Ontology-Based Dialogue Management System for Virtual Personal Assistants : 8th International Workshop on Spoken Dialog Systems*, pp. 219–233. 01 2019.
- [91] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," *Technical Report CUED/F-INFENG/TR 166*, 11 1994.
- [92] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.
- [93] Y. Li, D. Tarlow, M. Brockschmidt, and R. S. Zemel, "Gated graph sequence neural networks," *CoRR*, vol. abs/1511.05493, 2016.