UNIVERSITÉ DE
SHERBROOKE
Faculté de génie
Génie électrique et génie informatique

Architecture générique pour la découverte de la signification d'un message

Mémoire de maîtrise ès sciences appliquées
Spécialité: génie électrique

Simon-Pierre MORIN

Jury: Charles-Antoine BRUNET
Ruben GONZALEZ-RUBIO
Chantal-Edith MASSON

Sherbrooke (Québec) Canada Décembre 2008

Your file Votre référence
ISBN: 978-0-494-53413-7
Our file Notre référence
ISBN: 978-0-494-53413-7

À tous ceux et celles qui m'ont soutenu et encouragé à persévérer.

RÉSUMÉ

Ce mémoire présente la conception, la réalisation et les tests effectués pour une architecture générique permettant à une machine de reconnaitre le sens d'un message en utilisant le modèle de la cognition linguistique de l'humain et un environnement commun à celui des humains.

À ce jour, les machines reconnaissent des mots clés et répondent en suivant un modèle préfabriqué, tous deux préprogrammés par des humains. Donc, chaque agent informatique muni d'une forme d'interaction avec le public se voit attribuer un certain nombre de questions potentielles avec les réponses associées, parfois préconstruite. C'est-à-dire que la machine possède une base de mots avec un ordre prédéfini de ceux-ci qu'elle peut utiliser, parfois une phrase déjà entièrement construite, que la machine utilise telle qu'elle.

L'objectif principal de ce projet est de démontrer qu'il est possible pour la machine de s'approcher du modèle proposé par les linguistes, principalement un modèle proposé par Kleiber, adjoint à un modèle de cognition, celui du STI, et d'extraire le sens d'un message dans le but de l'interpréter. Ainsi, il est possible d'établir une forme de dialogue entre un être humain et une machine. Cet objectif est atteint en proposant une nouvelle architecture générique pour le traitement du langage naturel. Contrairement à ce qui est fait habituellement dans ce genre de problématique, les réponses obtenues ne doivent pas être des réponses préconçues, mais bien des phrases générées par la machine à partir de la grammaire de la langue.

Les résultats obtenus montrent qu'il est possible de donner un sens aux mots composant un message de manière à ce qu'une machine soit en mesure de l'interpréter dans un langage qui lui est propre. Cela est fait de manière à ce que cette même machine puisse répondre à son interlocuteur, voire éventuellement prendre une décision en rapport avec la conversation.

Mots clés : architecture logicielle, linguistique, prolog, signification du message, intelligence artificielle, interprétation, analyse, phrase générée.

REMERCIEMENTS

Merci à Charles-Antoine Brunet, mon directeur de recherche, pour ses explications, sa direction, son support indéniable et son intérêt dans ce projet. Sans lui, rien de tout cela n'aurait été possible. Un grand merci de m'avoir supporté dans ce projet hors des sentiers battus. Merci à Chantal-Édith Masson, une collaboratrice importante, pour ses explications en linguistique et l'orientation qu'elle a permis de donner au projet. Sans elle, le projet aurait eu un volume trop important pour une seule personne. Merci à Ruben Gonzales-Rubio pour ses demandes de précision que ceux qui étaient trop impliqué ne voyaient plus. Merci à Daniel Morin et Danielle L'Heureux-Morin, mes parents, pour votre amour et pour m'avoir encouragé à continuer et à me surpasser. Merci à Marie-Andrée Morin, ma petite soeur, qui a mis mon projet à l'épreuve lors de certains tests. Merci à Kristel L'Espérance, mon amoureuse, pour son support durant l'écriture de ce mémoire. Merci à tous ceux qui ont eu un regard de près ou de loin dans les versions préliminaires de ce mémoire, que ce soit au niveau du contenu ou de la forme. Merci à mes ami(e)s qui m'ont permis de penser à autre chose quand j'en ai eu besoin. Merci également à l'Université de Sherbrooke pour m'avoir donné les connaissances nécessaires pour la réussite de ce projet. Merci à tous les lecteurs futurs de vous intéresser au contenu de ce document.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LEXIQUE

Conjugueur Néologisme adopté en 2004 par l'OQLF. Logiciel fournissant à tous les temps la conjugaison des verbes d'une langue [OQLF, 2008].

Dictionnaire Dans le présent document, le dictionnaire fait référence à un vocabulaire terminologique fortement monosémique, une base de mots n'ayant qu'une seule définition pouvant être utilisés. Étant d'usage plus commun pour les lecteurs, le terme dictionnaire, bien que plus englobant, sera utilisé.

Entité conceptuelle Une entité conceptuelle n'a pas de support physique dans l'un des univers selon la définition du terme *environnement*. Parmi ces entités se trouvent : les sentiments, les concepts généraux et les lois universelles. Les entités conceptuelles se retrouvent souvent en paires puisque, contrairement aux entités physiques, qui ont un support dans leurs réalités respectives, les entités conceptuelles ne sont pas perçues par les sens. C'est donc la connaissance du concept inverse qui définit l'entité conceptuelle. Par exemple, il n'est pas possible de définir la notion de *bien* sans définir la notion du *mal* en même temps.

Entité physique Une entité physique est toute réalité pouvant se retrouver dans l'un des univers selon la définition du terme *environnement*. Il peut s'agir d'un être vivant ou bien d'un objet concret. Une entité physique peut être perçue par les sens appartenant au même univers.

Environnement Dans le présent document, il s'agit d'une référence aux trois univers suivants :

Univers physique : cette réalité est celle dans laquelle les entités physiques évoluent et sont perçues.

Univers imaginaire et spirituel : cette réalité, qui découle de la réalité physique, est formée de légendes, de rêves, de fables et de croyances.

Univers virtuel : cet univers, plus récent, est l'ensemble du monde numérique et des possibles non actualisés.

Grammaire Dans ce document, la grammaire se limite aux règles d'accord et de syntaxe de base.

Mémoire La mémoire est la capacité qu'ont les êtres vivants et les machines intelligentes d'enregistrer un certain nombre d'informations. Trois types de mémoire existent. La mémoire à court terme, qui analyse ponctuellement ce qui se passe, la mémoire à moyen terme, aussi appelée mémoire de travail, et la mémoire à long terme. Cette dernière se subdivise en trois sous-catégories :

Mémoire sémantique : Ensemble des connaissances acquises. (savoir linguistique et savoir structuré)

Mémoire procédurale : Mémoire des automatismes, tels marcher, courir et manger.

Mémoire épisodique : Souvenirs proprement dit des personnes. (mémoire autobiographique)

Également, à l'intérieur de la mémoire épisodique se retrouve la mémoire sensorielle, telle iconique (visuelle) et échoïque (auditive), qui englobe les souvenirs reliés à un sens ou à une émotion.

Micro-environnement Environnement de référence servant de lien entre l'univers physique (environnement humain) et l'univers virtuel (environnement machine).

Monosémique Mot qui n'a qu'un seul et unique sens [DUBOIS, 2001].

Polysémie Propriété d'un mot à avoir plusieurs sens [DUBOIS, 2001].

Prototype En linguistique : Instance typique d'une catégorie [DUBOIS, 2001].

En informatique : représentant objectal du concept qui représente le mieux le concept et la classe entière associée [OQLF, 2008].

Référence La référence est le principe par lequel un mot est remplacé par un pronom.

Référent Le référent est l'entité physique ou conceptuelle qui est représentée par l'utilisation d'un mot de la langue.

Sémantique Étude du sens d'un mot par l'analyse de leur phonème (base du mot) et morphème (variation) [DUBOIS, 2001].

Sémantique du prototype Capacité de trouver le meilleur exemplaire (représentant) associé à une catégorie. Cette définition se rapproche de la théorie du stéréotype de la sociologie, mais cette fois, appliquée à la structure du langage humain [KLEIBER, 1990].

Taxon Groupe d'êtres vivants ayant des traits caractéristiques communs. Dans la classification internationale, les taxons correspondent aux unités systématiques utilisées à chaque niveau du système de classification. Ce sont le règne, l'ordre, la famille, le genre, l'espèce et la variété.

Terminologie Ensemble de termes, définis rigoureusement, qui désigne les notions qui sont utiles à un domaine spécialisé.

Traitement du langage Dans le présent document, référence à l'ensemble du traitement de la parole et du message.

Traitement du message Dans le présent document, la recherche du sens du message écrit ou parlé.

CHAPITRE 1

INTRODUCTION

Depuis déjà plusieurs décennies, des recherches sont faites pour établir une communication réelle entre les humains et les machines. C'est un rêve humain de vouloir produire une machine dotée de caractéristiques humaines. D'ailleurs, des androïdes dotés de parole sont souvent présents dans les productions de science-fiction. Le dialogue humain-machine n'est pas chose facile et, pourtant, cette communication ouvrirait des portes à toutes sortes d'applications. Jusqu'à ce jour, le dialogue entre un système informatique et un humain se fait principalement par reconnaissance de mots clés. Plusieurs applications se sont construites autour de cette méthode, comme les robots munis de reconnaissance vocale et les agents conversationnels (*chatbots*).

Ce projet de recherche dote la machine d'une représentation permettant une compatibilité entre les langages humain et machine. L'idée est simple, mais sa mise en oeuvre l'est beaucoup moins. Une des difficultés rencontrées est que la machine, n'ayant pas de conscience, ne connait pas le sens de ce qu'elle dit ou de ce qu'elle entend. Ce problème est observable dans différentes situations. Chaque fois, le même raisonnement se produit : si ce terme existe, c'est qu'il doit avoir un sens. Concrètement, en partant du fait que «Le sens n'a de sens que si on lui en donne un. (Anonyme)», pour qu'une machine soit en mesure de comprendre un langage humain, il faut lui faire comprendre que chaque mot à une signification réelle.

Dans le cadre de cette recherche, il est question de démontrer qu'il est possible de donner un sens à un mot de manière à ce qu'une machine soit en mesure de l'interpréter dans un langage qui lui est propre afin qu'elle puisse répondre à son interlocuteur, voire prendre une décision en rapport avec la conversation. Une des applications possibles qui utiliserait ce concept pourrait servir, avec l'aide d'un support robotisé, à l'assistance de personne en manque d'autonomie. Notamment, les personnes paraplégiques pourrait bénéficier d'un tel système. En effet, il n'existe, à ce jour, que peu d'alternatives pour répondre à leurs besoins. À plus grande échelle il serait possible d'associer des éléments du langage avec, par exemple, des actions concrètes à effectuer. Toutefois, il reste beaucoup de travail à faire afin que le langage de l'humain et le langage de la machine soient compatibles au niveau de l'interprétation du sens des mots.

L'objectif principal de ce projet est de démontrer qu'il est possible pour la machine de s'approcher du modèle linguistique, ici un modèle proposé par Kleiber, et de s'arrimer à un modèle cognitif, ici celui du STI, dans le but d'interpréter un message et de répondre de manière *réfléchie* à ce message. Il serait alors possible d'établir un dialogue entre un être humain et une machine. La contribution du projet est de proposer une nouvelle architecture générique pour le traitement du langage naturel en accord avec un modèle combinant un des modèles de Kleiber, impliquant que les concepts ne sont pas fermés, mais des classes floues et ouvertes, jumelé au modèle cognitif du STI, décrivant le processus de compréhension d'un message chez l'humain. Cette architecture est par la suite validée avec une implémentation concrète. De prime abord, le problème est simplifié de manière à ce que la majorité des mots utilisés par la machine n'aient qu'un seul sens (vocabulaire normalisé) et que l'environnement soit restreint pour une application donnée.

Pour la réussite de la démonstration, la machine ne doit pas utiliser un patron de réponse, mais plutôt des phrases qu'elle construit elle-même. Pour ce faire, l'environnement de référence de la machine doit se rapprocher suffisamment de celui des humains pour considérer qu'il y a une base commune aux deux représentations. Il faut également mettre en oeuvre une grammaire permettant à la machine de construire ses propres phrases à partir des éléments de l'environnement et de la grammaire de la langue désirée, le français dans ce cas-ci.

Afin de limiter la taille du projet, des contraintes ont été fixées. Compte tenu des contraintes inhérantes au contexte académique de la présente recherche, il importait de bien circonscrire le micro-environnement sous étude et d'y limiter strictement le vocabulaire, lequel est presqu'exclusivement monosémique. Seuls les expressions nécessaires pour le bon fonctionnement de l'application pour le micro-environnement choisi sont définies. La démonstration du fonctionnement de l'architecture est faite avec un micro-environnement limité à un cas d'utilisation précis. Il s'agit d'un ordinateur dans une pièce avec des informations sur ce qui l'entoure. L'environnement se limite donc au vocabulaire nécessaire pour intéragir avec le contenu de cette pièce. De plus, afin de réduire la complexité, les termes sont réduits à une seule définition par mot, à quelques exceptions près. Ainsi, le problème de la polysémie est mis de côté pour les tests puisque le vocabulaire est fortement monosémique. De plus, la grammaire se limite aux accords du pluriel et du féminin ainsi qu'à la syntaxe de base des phrases, mais non le choix des mots la formant. C'est-à-dire que cette grammaire comprend l'axe syntagmatique, mais non l'axe paradigmatique. Finalement, le dictionnaire s'apparente davantage à une terminologie qu'à un dictionnaire proprement

dit dû à la monosémie forcée, mais le terme dictionnaire sera quand même utilisé pour une question de généralisation et de simplification.

Pour ce projet, une architecture informatique basée sur un modèle linguistique est proposée. Il n'est donc pas question de reconnaissance de mots clés, mais plutôt de fonctionnalité et de sens des mots. Cette approche, bien que nécessitant plus de ressources du processeur qu'une approche classique, permettra une interaction plus naturelle entre une machine et un être humain puisque l'architecture donnera à la machine et à l'humain une référence commune à leur langage respectif.

Le document est structuré de la manière suivante. Il est décrit au chapitre 2 l'état actuel des recherches au niveau informatique et les théories linguistiques en lien avec ce projet. La transition entre le modèle humain et le modèle du projet est expliquée à la section 3.1. La conception et la réalisation des modules mentionnés précédemment sont décrites au chapitre 3. Finalement, le chapitre 4 montre les tests, les résultats et les analyses des différents modules. La conclusion est présentée au chapitre 5.

CHAPITRE 2

REVUE DE LA LITTÉRATURE

Ce chapitre discute des modèles proposés par les linguistes puis des réalisations dans le domaine informatique, ainsi qu'une introduction au langage qui sera utilisé pour unifier ces concepts. Il démontre qu'il existe un écart considérable entre les modèles apportés par les liguistes et ce qu'un ordinateur est en mesure de faire à ce jour. Les termes *compréhension* et *reconnaissance* du message sont utilisés dans ce document. Par compréhension, il est question du fait que le message a été décodé, que son sens a été reconnu et que, si une réponse doit en ressortir, que celle-ci puisse être générée. Par reconnaissance, il est simplement question du fait que les mots formant la phrase soient reconnus, par leur sens, leur définition ou leur fonction, sans traitement significatif de ceux-ci.

2.1 Modèles linguistiques

Les linguistes se sont penchés depuis plusieurs décennies sur la compréhension du sens du message. La compréhension est le fondement de la génération d'un message. Ils se sont également penchés sur le principe du dialogue lui-même. Il existe des centaines de modèles linguistiques, certains plus intéressants que d'autre. Entre autre, un modèle qui ne se base pas strictement sur les conditions nécessaires et suffisantes pour déterminer la nature ou le sens d'un mot. Kleiber a d'ailleurs écrit plusieurs ouvrages sur le sujet. Il indique les principaux concepts du langage en général en s'appuyant sur des exemples en français afin de faire ressortir les différents éléments qui composent le message dans le but de mieux le comprendre. Parmi ces concepts se trouvent la sémantique et la prototypicalité [KLEIBER, 1990] ainsi que la polysémie, la définition et la référence [KLEIBER, 1999].

Les concepts de sémantique et de prototypicalité sont utilisés en linguistique pour représenter les interrelations qui existent entre certains mots. Ensemble, ils donnent naissance à la sémantique du prototype. Le prototype représente le meilleur exemplaire d'une catégorie et il est possible de déduire que les entités s'en approchant suffisamment font également partie de cette catégorie. Les classifications classique se font sous forme de pyramide où sont présents deux types de classification, horizontale et verticale, comme le montre la figure 2.1.

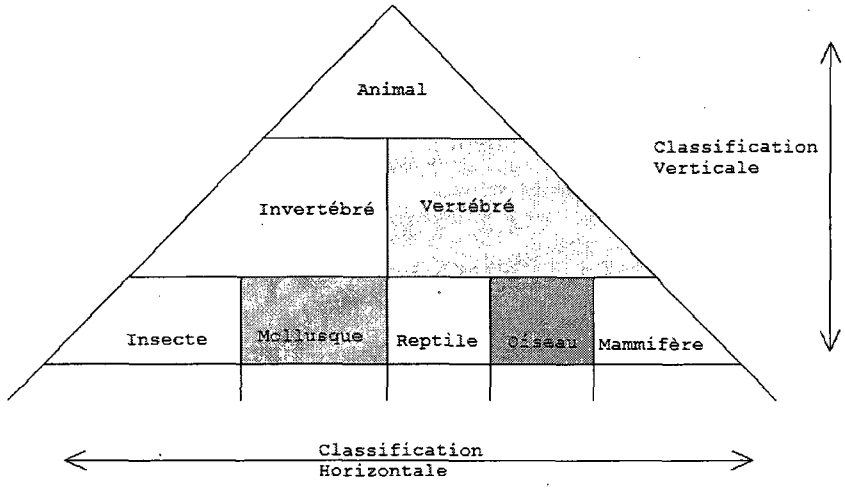

Figure 2.1 – Exemple de pyramide de classification classique

Cette pyramide de classification contient de plus en plus de représentants à chaque étage. Cette pyramide permet une intégration des concepts rapides et efficaces du point de vue de la visualisation et chaque enfant hérite des caractères de ses parents. À noter que cette figure n'est pas exhaustive, mais montre seulement quelques exemples pour les besoins de l'explication. De ce principe, le système de classification international a vu le jour. Le tableau 2.1 présente deux exemples qui illustrent la classification internationale classique.

Taxons	**Exemple1**	**Exemple2**
règne	animal	végétal
ordre	mammifère	arbre
famille	canidé	feuillu
genre	chien	chêne
espèce	caniche	vert
variété	miniature	nain

TABLEAU 2.1 – Exemples de la classification internationale

Le tableau 2.1 montre qu'à chacune des sous-catégories correspond un certain représentant. Ce représentant peut différer d'un individu à un autre. Par exemple, certains diront que le berger allemand serait le meilleur représentant du chien, toutefois, le caniche miniature appartient aussi à la catégorie du chien.

Telles que présentées par Kleiber, les catégories forment des classes ouvertes. En pratique cependant, comme le mentionne Kleiber, il faut limiter ce qui appartient ou non à une catégorie. Par exemple, un objet correspondant au modèle avec un certain taux d'erreur sera associé au modèle qui lui est le plus près horizontalement. Par exemple, le kiwi

(l'animal) présente des caractéristiques appartenant aux mammifères et aux oiseaux. En regardant uniquement ces deux catégories, si un taux d'erreur de 0.3 aux oiseaux et de 0.6 aux mammifères (nombres arbitraires) est déterminé, la catégorie des oiseaux sera choisie. La figure 2.2 illustre le principe de superposition des classes qui permet l'ouverture des prototypes.

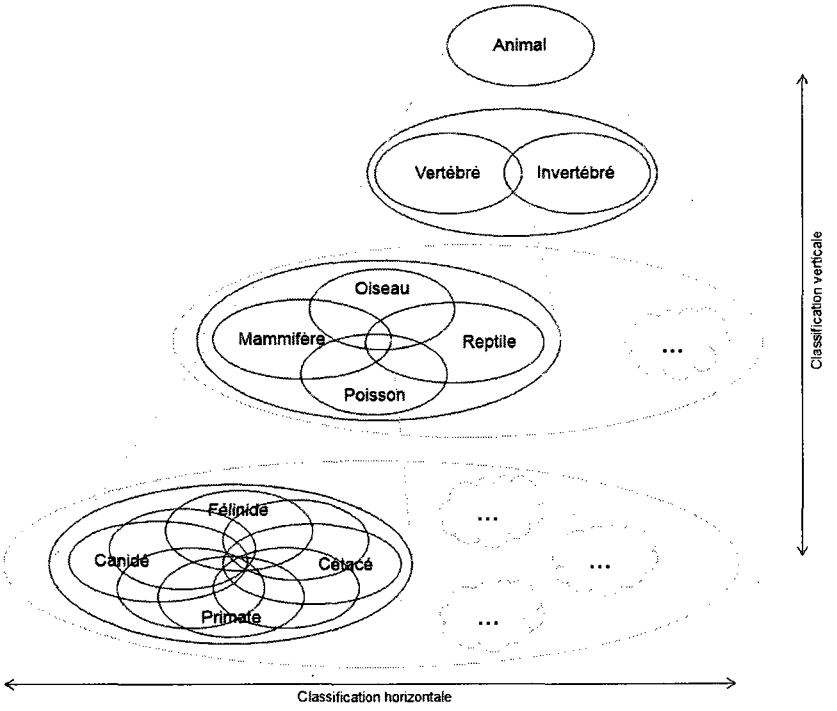

Figure 2.2 – Exemple de pyramide de classification ouverte

Il est à noter sur la figure 2.2, que pour chaque *étage* de la pyramide, il n'y a qu'une seule classe qui est développée pour alléger l'illustration. Ainsi selon les caractéristiques d'un individu, celui-ci peut appartenir à différentes classes. Son niveau d'appartenance déterminera à quelle classe il appartient. De la même manière, lorsqu'on détermine le meilleur représentant d'une classe plus générale, une classe plus spécifique ne présentant que des caractéristiques de cette classe générale est plus suseptible d'être un meilleur représentant de cette classe. Pour reprendre l'exemple du kiwi (l'animal), il présente à la fois des caractéristiques des mamifères et des oiseaux, même s'il fut déterminé que celui-ci était un oiseau par l'exemple précédent. De ce fait, le kiwi n'est pas le meilleur représentant des oiseaux et ne sera pas prévilégié pour être choisi comme représentant possible des oiseaux face au merle, par exemple.

La recherche de caractéristiques au sein d'une ligne de la pyramide est référée comme classification horizontale. Ainsi, chaque entité physique ou conceptuelle appartenant à

une même catégorie se retrouve associée à un taxon commun, indépendamment de sa représentativité pour la catégorie. Ensuite, les différentes classes identifiées sont hiérarchisées de manière à regrouper des classes présentant des similitudes. Cette hiérarchisation est référée comme classification verticale.

Au niveau de la langue, c'est habituellement le taxon du genre qui est utilisé pour référer à une entité de manière générale. Un individu dira qu'il a vu un chien, sans nécessairement préciser de quelle espèce il s'agissait. Toutefois, selon les entités à décrire, un taxon plus général ou plus spécifique sera utilisé. C'est ainsi que la hiérarchie est décrite, selon Kleiber, comme faisant partie des connaissances et de la culture des êtres humains. Cette hiérarchisation est faite pour toutes les entités qui sont rencontrées par des êtres humains. La classification internationale est bien connue pour les êtres vivants et il en va de même pour les mots dans une langue, par exemple, aimer est un verbe du premier groupe et les verbes sont des mots. Pour réussir à déterminer où un mot se situe dans la double classification, c'est-à-dire une classification horizontale et verticale combinée, les quatre points de comparaison suivants sont utilisés :

1. Appartenance à une catégorie : vérification de la nature du mot par critères non analytiques. Comme dans l'exemple de la classification internationale du tableau 2.1, un mot est catégorisé dans une classe de mots et il est regroupé par un terme plus général. Ainsi, un mot décrivant une action a plus de chance d'être un verbe alors qu'un mot indiquant un objet a plus de chance d'être un nom.

2. Sens et référence : l'expression référée par un mot (généralité) de la catégorie est basée sur le sens du mot (spécificité). Par exemple, le pronom prendra le sens du nom qu'il remplace. Dans l'expression «Apporte-le-moi.», *moi* fait référence à l'interlocuteur et *le* à un objet qui aurait été défini lors d'une phrase précédente. Ainsi, une catégorie qui prend en compte les objets inclut également les référents directs de ces objets qui sont membres de la catégorie.

3. Double parallélisme : extension (générique) et intension (spécifique) du mot. Un exemple concret est la substitution du contenant pour le contenu. «Passe-moi le sel» ne fait pas référence au sel lui-même, mais bien la salière. Ici, la catégorie détermine un lien étroit entre certains mots de catégories différentes (parallèles). Le sel faisant partie des condiments et la salière des récipients.

4. Polysémie présente : sens multiples. Par exemple, le mot *gorge* peut signifier autant une partie du corps qu'une vallée étroite.

À la suite de ces points de comparaison, la catégorisation sera confirmée en considérant la flexibilité et la stabilité de la structure. Un principe important étant «L'exception confirme la règle[...]»(FLAUBERT, 1966). En effet, les linguistes, comme d'autre spécialistes de disciplines différentes, partent du principe qu'il est inutile de définir une règle si aucune exception ne peut se produire. De la même façon, les entités purement conceptuelles n'auront pas de nom tant et aussi longtemps que son opposé ne sera pas identifié, par exemple la définition du bien et du mal. Comme le démontre Rosch [ROSCH et coll., 1993], certaines entités conceptuelles, telles les couleurs, ou celles qui n'appartiennent pas à un continuum bipolaire, comme chaud, tiède et froid, existent malgré l'absence d'une dualité inverse. Les points de comparaisons 3 et 4 rendent parfois impossible le classement d'un mot uniquement par lui-même. Dans ce cas, le sens de la phrase doit être observé. C'est principalement le cas pour les mots polysémiques et les mots référents. Pour l'analyse d'une phrase, six points d'étude doivent être pris en compte :

1. Phonétique et phonologie : étude des sons de la linguistique (au niveau oral)
2. Morphologie : partie de la grammaire qui étudie la formation des mots
3. Syntaxe : partie de la grammaire qui étudie la relation structurale entre les mots
4. Sémantique : étude du sens lui-même
5. Pragmatique : étude du langage utile pour atteindre un but
6. Discours : étude de la parole ou de la langue concrétisée

Il existe d'autres approches traîtant du côté émotionnel du message comme étant le véhicule premier du sens de ce message [WINOGRAD, 1978]. Cependant, vu la quasi impossibilité actuelle, du côté informatique, à faire ce genre de traitement, celles-ci seront mise de côté.

L'approche présentée par Kleiber fait partie d'une multitude d'autres théories qui ne sont pas présentés dans ce rapport. Celle-ci ne se base pas que sur des méthodes classiques de classification par des conditions nécessaires et suffisantes. Elle permet une prototypicalité par l'intersection des critères et leur relativisation. Ceci ouvre la prise en charge du flou et du ponctuel caractéristique au discours humain. C'est pour cette raison que l'approche présentée par Kleiber sera conservée aux fins du présent projet de recherche.

2.2 Réalisations informatiques

Cette section présente les différentes recherches qui ont été effectuées sur la capacité de l'univers virtuel à comprendre et à s'exprimer de manière cohérente dans l'univers physique. Il est d'abord question des différentes recherches en traitement du langage, puis sont présentées certaines techniques d'intelligences artificielles.

2.2.1 Recherches courantes en traitement du langage naturel

Plusieurs approches ont été traitées selon les domaines pour le traitement du langage naturel. Les approches programmées de micro-environnement et de traitement de langage diffèrent des approches proposées par les linguistes.

L'un des outils les plus avancés en terme de langage naturel, le *Natural Language ToolKit* (NLTK) [KLEIN, 2006], permet d'effectuer l'analyse syntaxique (en anglais) de ce qui est écrit. Par exemple, si un mot peut être à la fois un nom ou un verbe, il vérifie les séquences de mots précédents. Si cela ne lui permet pas de déterminer de quel type de mot il s'agit, il vérifie le nombre d'occurrences statistiques du langage que ce mot possède pour chaque fonction. Il peut également répondre de façon aléatoire à des questions sur la base de réponses disponibles ou, par une technique inspirée de la logique du premier ordre, indiquer la réponse la plus probable sur un sujet donné. Cependant, toutes les réponses données par NLTK sont préconstruites. Par la suite, il vérifie que la structure de la phrase correspond au modèle de phrase standard. La représentation graphique de cette technique est donnée à la figure 2.3 tirée de la documentation de NLTK [KLEIN, 2006]. Selon cette figure, NLTK commence tout d'abord par trouver la fonction de chaque mot

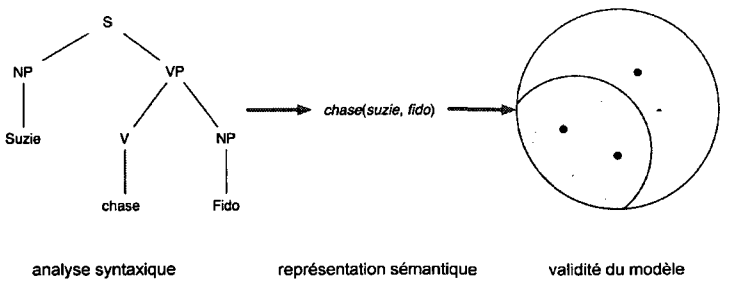

Figure 2.3 – Représentation graphique de la recherche sémantique de NLTK

dans la phrase. Ensuite, il convertit cette structure en logique du premier ordre (présenté à la section 2.2.2). Ensuite, il valide que le modèle qu'il a identifié correspond à une architecture valide selon la grammaire anglaise. Si oui, alors il donne la fonction du mot,

sinon il refait une itération tant que toutes les fonctions de chaque mot n'aient été vérifiées et qu'aucun modèle de phrase n'ait été identifié.

Cependant, NLTK n'est pas un interpréteur de sens. Pour le moment, il ne fait que décortiquer la phrase et ne semble pas en mesure de générer une phrase par ses propres moyens, ses réponses étant préconstruites.

Un système important dans l'analyse de texte par ordinateur est SATO, développé par le groupe ATO [MEUNIER, 2008] de l'UQAM. Au même titre que NLTK, SATO fait l'analyse du texte, soit pour retrouver des mots clés, soit pour analyser la sémantique du texte, en français cette fois. Néanmoins, bien que l'analyse soit faite de manière performante, elle n'est pas en mesure de répondre à un texte écrit en lien avec ce dont il était question dans le texte.

Il existe aussi des générateurs de texte, tel Nigel [MANN, 1983]. Mais, comme le suggère leur nom, ceux-ci ne font que générer du texte, sans pour autant comprendre le sens de ce qui a été dit ou écrit précédemment par un interlocuteur. Nigel est principalement utilisé dans l'optique de vérifier la syntaxe d'une phrase et d'en proposer une correction si celle-ci est déficiente. Ce principe a également été repris par la compagnie Druide pour son logiciel Antidote [BRUNELLE et coll., 2003], en français. Donc, pour ce qui est de l'analyse et de la génération de phrase, il existe déjà des outils qui ont été développés et qui sont utilisés couramment. Toutefois, ces outils ne permettent pas la compréhension du sens du message, seulement une génération automatique structurellement correcte du texte qui le compose.

D'autres travaux de recherches tentent de concevoir un processeur qui peut faire du traitement de langage naturel en temps réel [DEMARA et MOLDOVAN, 1991]. Il y a aussi des guides d'exposition ou de musée, par exemple Spartacus [MICHAUD et coll., 2005] et Repliee Q2 [ISHIGURO, 2006]) ou pour site Web comme Cybelle [ÉQUIPE AGENT LAND, 2006] et les autres agents conversationnels. Ceux-ci se basent principalement sur la reconnaissance de mots clés au sein d'une base de données et la réponse donnée est une phrase préconstruite ou partiellement préconstruite. Cette alternative est plutôt efficace dans ce genre d'application, mais reste totalement ineffective lorsque les questions ou les phrases utilisées sortent du contexte d'utilisation. En effet, des programmeurs doivent mettre à jours la base de données, ou encore, il doit y avoir un algorithme d'apprentissage basé sur la réponse de l'utilisateur, afin de rendre plus *intelligent* l'agent conversationnel. Bien qu'étant intéressants et pratiques, ces travaux ne permettent pas non plus de comprendre le sens du message ou ils le font de manière préprogrammée. Toutefois, ils offrent une

base intéressante pour une application de reconnaissance de messages écrits, c'est-à-dire la reconnaissance des mots et de leur fonction sans pour autant en comprendre le sens, et pour la génération de réponses.

Il y a également le principe de Web sémantique [HAWKE et coll., 2008] utilisé par certains moteurs de recherche. Le Web sémantique est une nouvelle manière de coder les pages Web. Il est normalement utilisé conjointement avec le langage de balise structurelle de marquage XML et avec des espaces de nommage. À l'aide de balise de type *META*, il est possible d'inscrire de quoi il est question dans la page visitée. De plus, les balises, ou étiquettes, peuvent être nommée à volonté et dans les termes *éloquents*, d'où l'idée de *sémantique*. Ainsi, les moteurs de recherche peuvent regrouper différents sites Web traitant du même sujet et suggèrent même des liens entre eux. Toutes ces technologies utilisent la reconnaissance de mots clés et nécessitent un lourd travail humain pour chaque cas d'utilisation. Aussi, du travail supplémentaire est requis pour permettre à une autre langue d'être utilisée. À nouveau, cet outil est pratique, mais ne permet toujours pas de comprendre le sens du message, il ne fait que reconnaître les mots.

D'autres générateurs de textes touchant aux différentes notions de la langue existent, tant au plan théorique que fonctionnel, comme le générateur de texte à expressions temporelles [GAGNON, 1996]. Ce générateur de texte utilise un journal d'événements passés, présents et futurs, qui sont définis de manière absolue ou relative dans le temps. Il permet de générer des phrases en utilisant le temps de verbe adéquat selon chaque situation. En 2005, les concepts développés par GAGNON devinrent un point important dans le développement du Web sémantique et de la correction automatique (Correcteur 101). Par contre, tout comme les autres techniques présentées précédemment, le sens même du message n'est pas traité. Cependant, la génération de texte à expressions temporelles ouvre la voie à la compréhension des messages dont le sens est passé ou futur.

Il existe donc des techniques pour faire du traitement de langage au sens large. Cependant, plusieurs autres recherches théoriques n'ont jamais abouti sur le plan pratique. Par exemple, le principe de génération syntaxique par prédicat [KOZLOWSKI et coll., 2003] ou encore l'étude des relations entre la sémantique et la programmation logique [JAUME, 1999]. Ces études sont intéressantes, car elles montrent une mathématisation du langage par des linguistes en logique du premier ordre. Ce langage étant déjà adapté pour des ordinateurs, l'utilisation de ces théories permettrait de faire apprendre certaines facettes du langage humain à la machine. Il existe également des théorèmes prometteurs dont la grammaire générative [CHOMSKY et coll., 2007] ainsi que le dictionnaire combina-

toire [MEL'CHUK, 1999]. De nos jours, il serait possible de revisiter ces théories, pour vérifier leur validité. Bien que, pour ce projet, ces théories n'ont pas été retenues, le principe d'utiliser une mathématisation du langage pour *instruire* la machine et le côté génératif du langage est conservé.

2.2.2 Intelligence artificielle

Les techniques d'intelligence artificielle tentent de reproduire des comportements naturels de l'univers physique par leur fonctionnement dans leur univers virtuel. La logique floue, les systèmes à bases de connaissances, les réseaux de neurones, les algorithmes génétiques et la logique du premier ordre sont quelques exemples parmi les plus connus [NEGNEVITSKY, 2005; RUSSELL et NORVIG, 2003]. Cependant, ce ne sont pas toutes ces techniques qui sont profitables pour des applications linguistiques. Les techniques présentées dans cette section sont celles qui semblent offrir des possibilités intéressantes pour ce projet de recherche.

La logique floue a été inventée par Zadeh et améliorée par la suite [ZADEH, 1965]. Son principal atout est au niveau de la mathématisation de certains concepts de la langue, mais elle peut être étendue à une palette bien plus large. La logique floue permet de quantifier certains termes, comme la température. Elle permet ainsi de fixer des seuils et de déterminer le degré d'appartenance à un groupe, comme pour la température qui appartient au concept de chaud, froid ou tiède. Cette technique donne donc, en plus d'établir le lien entre le terme température et les termes chaud, tiède et froid, un moyen à la machine de transformer une information quantitative en donnée qualitative et de redonner sa réponse sous forme qualitative ou de la retransformer en donnée quantitative. Cette méthode est très utilisée notamment dans les contrôles automatiques et la classification. Il pourrait être intéressant de réutiliser cette méthode pour réaliser certaines définitions et la classification de concepts mathématisables, telles la température (chaud, froid, tiède) et la grandeur (petit, grand, etc.). D'ailleurs, cette approche avait été envisagée par Zadeh lui-même dans un papier traitant de la logique floue au service du traitement sémantique [ZADEH, 1982]. Par contre, la logique floue est limité à des notions mathématisables, donc physiques. Elle doit être annexé de d'autres techniques pour analyser des concepts purement qualitatifs.

Les algorithmes génétiques [GOLDBERG et HOLLAND, 1988], pour leur part, sont principalement utilisés pour l'optimisation ou pour la recherche dans un espace d'état hautement complexe. Que ce soit au niveau quantitatif ou qualitatif, les algorithmes génétiques font varier certains critère de décision et choisissent, selon un seuil du critère de qualification, si

les résultats obtenus sont meilleurs ou pires avec les changements effectués. Ils réutilisent les résultats en les croisant, en leur faisant subir des mutations ou en les gardant tels quels sur plusieurs *générations* jusqu'à l'obtention d'un résultat acceptable. Cette technique s'avèrerait utile pour la classification horizontale (voir section 2.1) puisqu'elle permettrait de déterminer quel pourrait être le meilleur représentant d'une classe. En effet, avec un algoritme génétique, l'application pourrait tester itérativement différents représentants de la classe afin de déterminer de manière autonome lequel répond le mieux aux critères de qualification de la classe. Le représentant peut changer de manière dynamique lorsque de nouveaux membres sont ajoutés à une classe.

La logique du premier ordre est utilisée principalement en linguistique. Celle-ci décrit une base de faits tel *Spot est un chien, un ordinateur est une machine* et ainsi de suite. Elle se compose aussi d'une base de règles. Ainsi, il est possible d'inférer de nouveaux faits à partir des règles, établissant certaines relations entre les faits ou de déduire que quelque chose est vrai en s'appuyant sur des faits. Par exemple, puisque Spot est un chien, une base de règles pourrait déduire que Spot est un mammifère et que Spot a de la fourrure, à condition que les faits «les chiens sont des mammifères» et «les mammifères ont de la fourrure» soient présents dans la base de connaissances. Par analogie, on peut considérer le dictionnaire et l'encyclopédie comme une base de faits, puis le guide grammatical et le guide de conjugaisons comme une base de règles. En les mettant en commun, des phrases structurellement univoques, c'est à dire bien formée, peuvent être générées. Cette technique est très intéressante dans l'optique de cette recherche puisqu'elle se rapproche beaucoup du langage humain que ce projet tente de donner à une machine. En effet, plusieurs travaux de recherche des dernières décennies ont été mis sur papier sous cette forme, par exemple le modèle syntaxique [RATTÉ, 1995] et le modèle interprétatif [TANGUY, 1997]. Effectivement, lorsqu'ils présentent leurs travaux, plusieurs exemples démontrant leurs modèles sont faits en logique du premier ordre. Il est donc possible de réutiliser ces travaux en les informatisant. De plus, puisque la logique du premier ordre possède une forte ressemblance avec la langue structurée, il a été possible de transposer les dictionnaires et grammaires dans ce langage de manière à permettre à un ordinateur de les utiliser. Bien que ce modèle semble ouvrir la porte à des phrases déduites d'une longueur infinie, certaines limitations pratiques sont de mise pour éviter ce phénomène. Ces limitations correspondent aux règles de syntaxe d'usage de la langue basées sur les limites de la cognition humaine.

En ce qui a trait aux autres techniques, comme les réseaux de neurones artificiels, elles ne semblent pas, à première vue, adaptées pour la reconnaissance sémantique, mais pourraient cependant s'avérer un atout dans l'amélioration des performances du système.

Il existe donc des similitudes entre certains outils d'intelligence artificielle et certains théorèmes de linguistiques. Il est possible d'envisager un parallèle utilisant des bases de connaissances de logique du premier ordre pour détailler le dictionnaire et la grammaire. De plus, des principes d'algorithmes génétiques pourraient servir à identifier le meilleur représentant d'une classe. Finalement, la logique floue pourrait déterminer les concepts appartenant à des continuums bipolaires. Ainsi, il est envisageable de transcrire les concepts linguistiques dans un langage compris par une machine. Depuis quelques temps, diverses applications ont vu le jour, telles que des correcteurs automatiques comme le Correcteur 101 [GAGNON, 1996] et Antidote [BRUNELLE et coll., 2003]. Cependant, ceux-ci sont construites de façon statique basé sur des lexicographes d'allégence mel'chukienne [MEL'CHUK, 1999]. Elles sont peu adaptatives et formées principalement d'une table de connaissances. L'intégration de l'intelligence artificielle pourrait être un atout considérable pour ce genre d'application.

2.3 Prolog

Prolog (PROgrammation LOGique) est un langage de programmation dont le fonctionnement est basé sur la logique du premier ordre (LPO). Quoique la syntaxe soit différente de la LPO, on y retrouve tous les éléments nécessaires pour y concevoir une base de connaissances formée de prédicats, de faits et de règles. Prolog a été créé par Colmerauer et Roussel vers 1972 [MONGENET et coll., 2006]. À l'origine, ce langage fut inventé pour permettre l'utilisation de l'expression logique plutôt qu'une succession d'instructions à exécuter. Sa syntaxe et sa sémantique ont été conçues dans l'optique de fournir un outil pour les linguistes n'ayant aucune ou très peu de connaissances de l'informatique. Les concepts fondamentaux utilisés par ce langage sont l'unification, la récursivité et le retour sur trace. Une des particularités de Prolog est que l'on peut construire une base de connaissances indépendamment de l'ordre d'inscription des informations dont Prolog se servira pour résoudre des problèmes logiques.

Aujourd'hui, Prolog est utilisé dans de nombreux programmes d'intelligence artificielle et dans le traitement linguistique, entre autres le traitement des langages naturels par ordinateur. Prolog permet d'établir des relations, plutôt que des fonctions, entre les mots, ce qui rapproche le comportement de l'ordinateur à celui des humains. Pour ce qui est de l'intelligence artificielle, Prolog est utilisé, entre autres, pour l'informatisation de la logique du premier ordre et des modèles probabilistique pour la génération de séquence de Markov [RUSSELL et NORVIG, 2003]. Ces deux facettes de Prolog offrent une voie

intéressante pour l'accomplissement du projet qui nécessite à la fois le côté relationnel et le côté logique de ce langage.

Selon l'implémentation de Prolog choisie, il est facile de combiner à la fois les forces de Prolog avec les forces d'un langage de programmation procédural. Ainsi, il est possible, par exemple, de consulter la base de connaissances à partir d'un programme écrit en C++ ou en Java. Il est également possible de faire l'inverse. C'est pour ces raisons que Prolog est utilisé comme principal langage pour cette recherche. Il permet de mettre en oeuvre les dictionnaires et les grammaires ainsi que les connaissances nécessaires pour le micro-environnement afin que le message puisse être décodé. De plus, il permet à des modules externes de le consulter. Il est facile de concevoir qu'une application existante nécessitant une compréhension de ce qui est écrit, un *chatbot* (agent conversationnel) par exemple, peut inclure la base de connaissances en Prolog dans son code. Ainsi, la base de connaissances serait portable à n'importe quelle application.

Pour une étude approfondie des différentes possibilités qu'offre Prolog ainsi que des exemples concrets des différentes utilisations et résolutions de problèmes avec ce langage, le livre «Programmer en Prolog» [FORD, 1993] traite de ce sujet et s'adresse à un public intermédiaire ou avancé.

Plusieurs méthodes et théories existent donc du côté informatique et linguistique. Il s'agit de trouver le moyen de les rapprocher en créant une architecture qui permet de faire ressortir le sens du message afin de le comprendre. La compréhension du message sera donc atteinte uniquement lorsque la machine sera en mesure de reconnaitre les mots utilisés, d'en extraire leur sens et de former une réponse par elle-même en se basant sur la grammaire et à partir des connaissances de la machine sur son environnement.

Par exemple, pour trouver le sens de la phrase «Le livre est sur la table», il faut connaitre, comme le décrivent les linguistes, les définitions et les références des mots utilisés ainsi que leur fonction. Il faut donc avoir une connaissance de l'environnement, un dictionnaire pour les sens, une grammaire pour l'ordre et un conjugueur pour la temporalité et l'agent afin de déterminer la totalité de ces informations pour chaque mot. Pour simplifier le problème, le vocabulaire est considéré monosémique. Ainsi, *le* et *la* sont des articles qui impliquent qu'on connait l'existence du nom qui les suit. *Sur* est une préposition de lieu indiquant que l'entité précédente est en contact avec la partie supérieure de l'entité qui suit. *Livre* et *table* sont des objets connus dans l'environnement du locuteur. *Est* est le verbe être, au présent, utilisé comme verbe copule servant à relier le complément au sujet. Par la suite, *le livre* est reconnu comme étant le sujet de la phrase et *la table* comme

complément. Connaissant toutes ces informations, il est possible de savoir que le sens de la phrase est «L'objet connu sous l'appellation livre se trouvant dans l'environnement actuel du locuteur est localisé en contact avec la surface supérieure de l'objet connu sous l'appellation table dans ce même environnement.» Ainsi, en donnant à la machine les connaissances sufisantes pour déterminer la définition, la référence et la fonction des mots d'une phrase, il est possible de lui apprendre à déduire le sens d'une phrase.

CHAPITRE 3

CONCEPTION ET RÉALISATION

Ce chapitre présente les détails de l'architecture conçue et réalisée pour la recherche du sens d'un message. Dans les diagrammes de ce chapitre, les ellipses représentent un traitement sur les données, les rectangles sont des bases de données et les flèches sont des transmissions de données. À noter que les ellipses grisées sont des modules qu'il serait nécessaire d'ajouter pour une application complète et que les ellipses pointillées sont présentes pour faciliter la compréhension, mais sont détaillées dans leur section respective. Afin d'illustrer le fonctionnement des modules de l'architecture, l'exemple qui est utilisé est celui d'un robot domestique appelé *RoDo* qui reçoit des ordres vocaux de *Hugo*, un être humain.

3.1 Adaptation des théories

Comme en fait état le chapitre 2, un écart existe entre les modèles humains et les modèles informatiques. Cet écart est principalement dû à un retard technologique et au fait que, du côté informatique, les modèles des linguistes sont peu connus. Aujourd'hui, les systèmes sont plus performants et la compréhension des modèles est meilleure, ce qui permet d'envisager de réduire cet écart. Puisque la machine et l'humain ne sont pas conçus de la même manière, il faut d'abord trouver un modèle permettant d'avoir une base commune.

Comme point de départ pour la conception de cette architecture, le comportement humain est un bon modèle. En effet, les êtres humains, dès leur jeune âge, apprennent à interpréter le sens de ce qui leur est dit. Afin de rapprocher la compréhension de la machine et celle des humains, l'utilisation des modèles humains pour la compréhension de la langue est le point de départ pour chaque module de l'architecture.

3.1.1 Le traitement du langage par les humains

Tout d'abord, il faut considérer le fonctionnement du cerveau humain. Celui-ci est à la base du traitement de l'information chez l'être humain. Le cerveau reçoit des informations diverses de son environnement sous forme d'influx nerveux et chaque partie du cerveau ayant sa spécialité traite cette information afin de lui donner un certain sens pour ensuite

prendre une décision sur ce qui doit être fait par rapport à l'information qui vient d'être reçue.

Les entrées du cerveau sont donc les données provenant de l'environnement (sons, odeurs, sensation, goût et images). Celles-ci sont comparées avec les connaissances acquises. Elles sont ensuite traitées par différentes sections du cerveau pour finalement reformer un message en réponse au stimulus. S'il s'agit d'une réponse sous forme de parole, le cerveau envoie au système verbomoteur les informations afin de produire la phrase désirée en sortie. [TORTORA et coll., 1999]

Le fonctionnement du cerveau humain serait donc une bonne base, mais sa compréhension se fait à trop haut niveau, car la connaissance n'est pas encore assez développée. Il faut donc se baser sur d'autres modèles.

3.1.2 Approche du système de traitement de l'information (STI)

Certains linguistes se sont penchés sur la sémantique cognitive et un modèle du processus à été proposé comme le montre la figure 3.1.

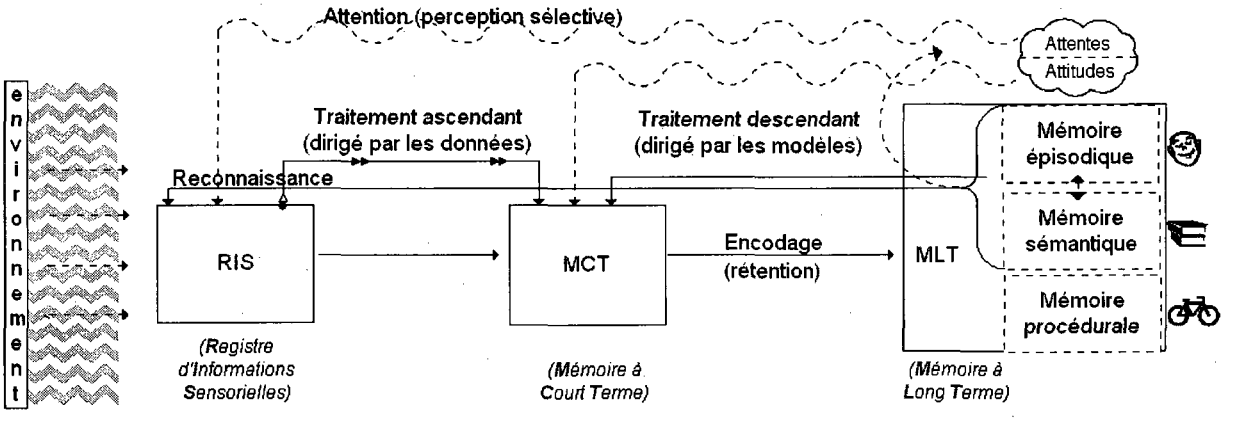

Figure 3.1 – Processus cognitif selon l'approche STI [MASSON, 2003]

L'architecture de la figure 3.1 se base sur les concepts du cerveau humain à plus bas niveau, il n'est donc pas étonnant que l'environnement soit l'entrée du système. Les informations venant de l'environnement sont traitées par le registre d'information sensoriel (RIS) en suivant, lorsqu'elles sont connues, les informations reconnues par la mémoire à long terme (MLT) épisodique (MLT-e) et sémantique (MLT-s) ou en ajoutant les informations lorsqu'elles ne sont pas connues. Ensuite, les données sont traitées par la mémoire à court terme (MCT). La MCT considère à la fois les informations provenant de l'environnement ainsi que les informations présentes dans la MLT. Ensuite, s'il s'agit d'une information

devant être archivée, elle est envoyée vers la MLT pour devenir une connaissance (MLT-s) ou un souvenir (MLT-e). S'il s'agit d'une demande de réponse, la MCT renvoie la réponse vers les zone du cerveau servant à la communication. La facilité à traiter de l'information est reliée à l'attitude et à l'attention de la personne.

La MLT-p, qui comporte principalement les automatismes acquis, tels que marcher, faire du vélo ou manger avec une fourchette, ne sera pas abordée puisqu'elle relève de la robotique procédurale et non pas de la reconnaissance du message. La MLT-e comprend tous les souvenirs d'une personne et s'apparente grossièrement à la mémoire événementielle d'une machine. La MLT-s contient toutes les connaissances acquises d'une personne, ce qui correspond à une base de connaissances d'une machine. La MCT permet le traitement de l'information à l'intérieur du cerveau, ce qui équivaut aux calculs et aux traitements exécutés par un ordinateur. Finalement, le RIS qui correspond aux signaux des cinq sens, il est équivalant à des capteurs sur l'environnement de la machine ou, dans le cas présent, la reconnaissance de son environnement. Pour ce qui est de l'attention de la machine, dans le cadre du projet, elle sera écartée puisque la machine n'a pas de baisse d'attention proprement dit, mis à part des ralentissements dus au traitement en cours.

Il est possible d'adapter l'approche du STI afin d'obtenir un système machine qui s'apparente au fonctionnement cognitif de l'être humain pour la compréhension du message. C'est cette nouvelle architecture qui est présentée à la section 3.2 qui est la base du projet. Les concepts utilisés pour cette architecture reprennent donc l'approche proposée par Kleiber, présentée au chapitre 2, ainsi que la représentation STI.

3.2 Architecture globale

L'architecture présentée à la figure 3.1 doit être adaptée afin d'être implantée d'un point de vue logiciel dans ce projet. Tout d'abord, le module d'environnement est un environnement virtuel (section 3.5), mais l'architecture est prévue pour qu'il puisse être adjoint à un module d'acquisition de données sur un environnement réel. Dans ce document, le terme générique *environnement* est utilisé. Pour une machine, le traitement et le RIS se font à l'intérieur même du module d'acquisition des informations sur l'environnement, à l'exception de l'ouïe qui est traitée séparément pour la conversion parole à texte. La MLT épisodique devient l'historique des connaissances des éléments dynamiques, appelée *connaissances dynamiques* (section 3.6). La MLT sémantique pour sa part renferme toutes les connaissances nécessaires pour la compréhension et la génération de la langue. Ainsi, les règles d'accords, de syntaxe, de conjugaison ainsi que le vocabulaire sont pré-

sentes dans ce module. La MLT sémantique renferme donc les informations linguistiques, nommée ci-après module *bloc linguistique* (section 3.4). Finalement, la MCT qui fait le traitement pour reconnaitre et comprendre le message est le module qui sert à interpréter le sens du message, il est appelé *interpréteur de sens* (section 3.3). La figure 3.2 montre l'adaptation du STI pour l'architecture de ce projet et les interrelations qui existent entre les différents modules.

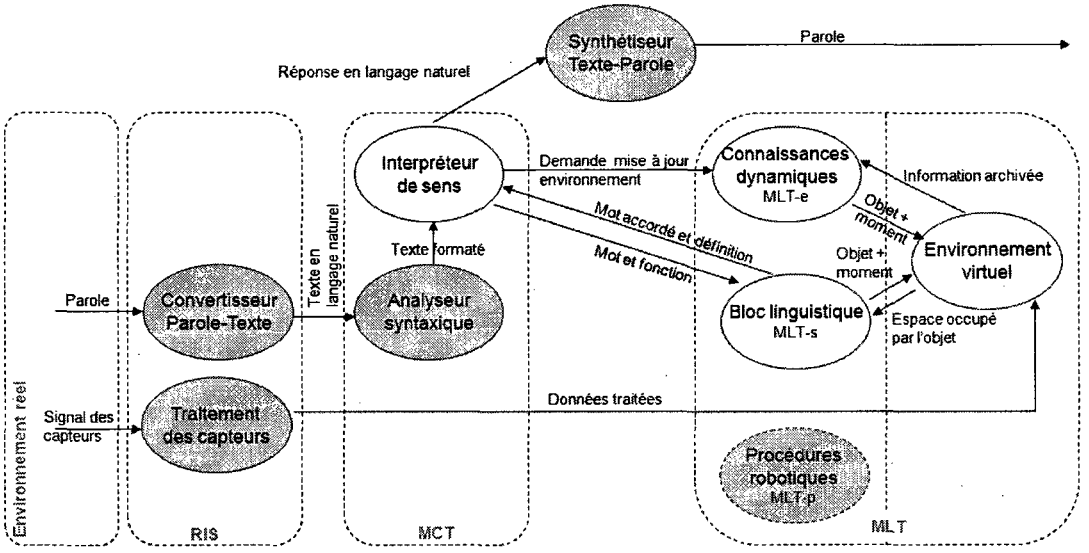

Figure 3.2 – Architecture globale du système

Étant donné que seul le module *bloc linguistique* est dépendant de la langue, pour une application où la langue serait différente, il est souhaitable que seul le contenu de ce module soit modifié pour correspondre à la langue désirée.

En suivant la figure 3.2, la parole entre dans le module de conversion pour être transformée en texte. Par la suite, ce texte est analysé par un analyseur syntaxique, par exemple SATO [MEUNIER, 2008], afin d'obtenir une forme structurée. Cette structure est ensuite formatée de façon à respecter le format voulu par l'*interpréteur de sens*. Le texte formaté est alors interprété par l'*interpréteur de sens*. Pour ce faire, les mots, ainsi que leurs fonctions, trouvés à l'aide de l'*analyseur syntaxique*, sont envoyés vers le module *bloc linguistique*. Selon les cas, le *bloc linguistique* envoie une définition avec le mot accordé en fonction de la phrase à générer ou il interroge les informations connues sur l'*environnement* pour répondre adéquatement à la demande de l'utilisateur. Selon le temps de verbe utilisé, le temps de verbe est modifié en moment de réponse, présent, passé ou futur, afin d'interroger l'historique des événements et renvoyer l'information au temps demandé. Lorsque le mot accordé et sa définition reviennent dans l'*interpréteur de sens*, celui-ci prend une

décision afin d'effectuer une action ou simplement répondre par une phrase qu'il construit à l'aide du *bloc linguistique*. La figure 3.3 présente un exemple de cette séquence pour les modules développés lors de ce projet. Par la suite, le texte généré par l'*interpréteur de sens* peut être acheminé vers un module de synthèse pour être donné sous forme de parole à l'utilisateur ou simplement affichée sous forme de texte à l'écran.

Le module *connaissances dynamiques* reçoit des informations semblant provenir à la fois de l'interpréteur et de l'environnement. En fait, tel que le montre le diagramme de séquence de la figure 3.3, ces interractions avec le module de *connaissances dynamiques* sont fait à des moment bien différent. Les détails des accès seront discutés dans les sections respectives de chaque module. Notez simplement que ce qui provient de l'interpréteur vers le module de *connaissances dynamiques* ne génère aucune réponse de la part de ce dernier. Seul un accusé de réussite ou d'échec est donné pour informer que l'action de mise à jour a été complétée. Toutefois, puisque chaque action dans l'architecture engendre une confirmation semblable, afin d'alléger les diagrammes ainsi que le texte, ces confirmations ne sont pas considérées comme étant un message en transit parmi les modules.

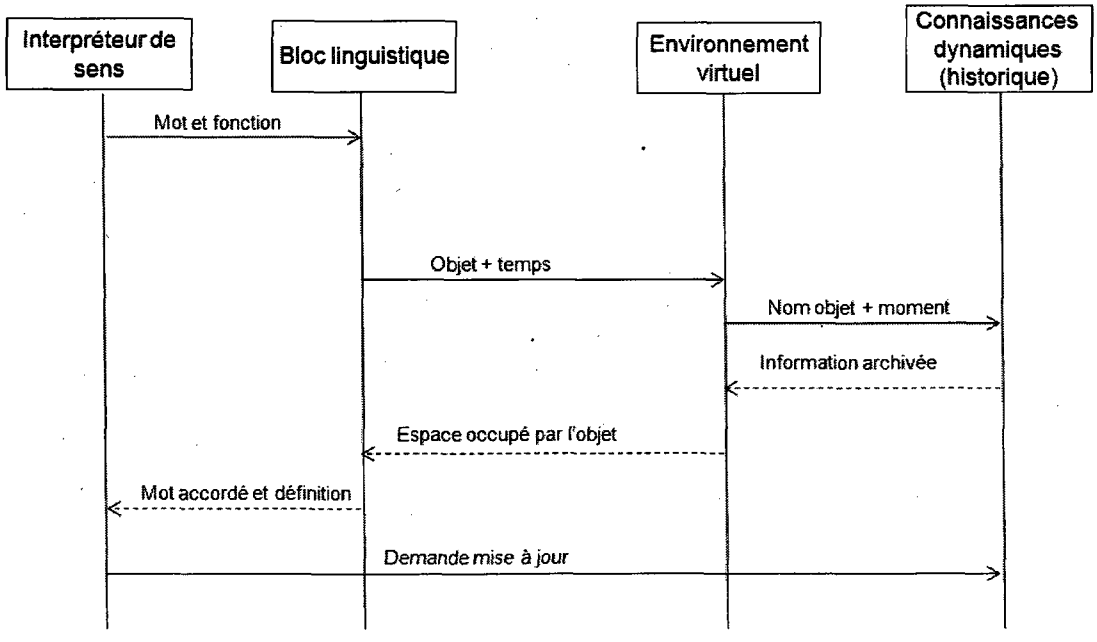

Figure 3.3 – Diagramme de séquence de l'architecture globale

L'exemple suivant illustre le fonctionnement global de l'architecture. Hugo veut savoir où se trouve son livre et utilise RoDo afin de l'aider. Pour ce faire, Hugo demande : «Où est le livre ?». L'*interpréteur de sens* demande d'abord au *bloc linguistique* ce que signifie le terme *où* et reçoit immédiatement l'information que ce terme indique une interrogation sur une localisation. Suite à cette information, RoDo cherche ce qui doit être localisé, soit

le livre. Le livre est un objet qui se trouve dans l'environnement. Le temps du verbe de la phrase est au présent, donc il n'est pas nécessaire d'interroger les connaissances passées ou à venir. Alors, le *bloc linguistique* recherche l'objet dans l'environnement au moment actuel. Le livre se trouvant à ce moment sur la table, cette information est donnée au *bloc linguistique* qui, à son tour, donne cette information à l'*interpréteur de sens*. Puisque la question concernait uniquement la localisation du livre, RoDo n'a pas à exécuter aucune action. Toutefois, il doit donner sa réponse à Hugo. L'*interpréteur de sens* demande donc au *bloc linguistique* de générer une phrase en français qui détermine une localisation du livre sur la table. Cette phrase se fait donc accorder puis RoDo répond à Hugo : «Le livre est sur la table.».

Les sections qui suivent permettent de voir plus en détail la conception et le fonctionnement de chacun des modules présentés à la figure 3.2.

3.3 Interpréteur de sens

Le module *interpréteur de sens* est le coeur de la découverte de la signification du message. Son rôle est d'appeler le module *bloc linguistique* pour décortiquer le message, de découvrir le sens de ce qui a été dit dans le message et de formuler une réponse qui répond ou réagit correctement au message. La figure 3.4 montre globalement ce qui est fait par l'*interpréteur de sens*.

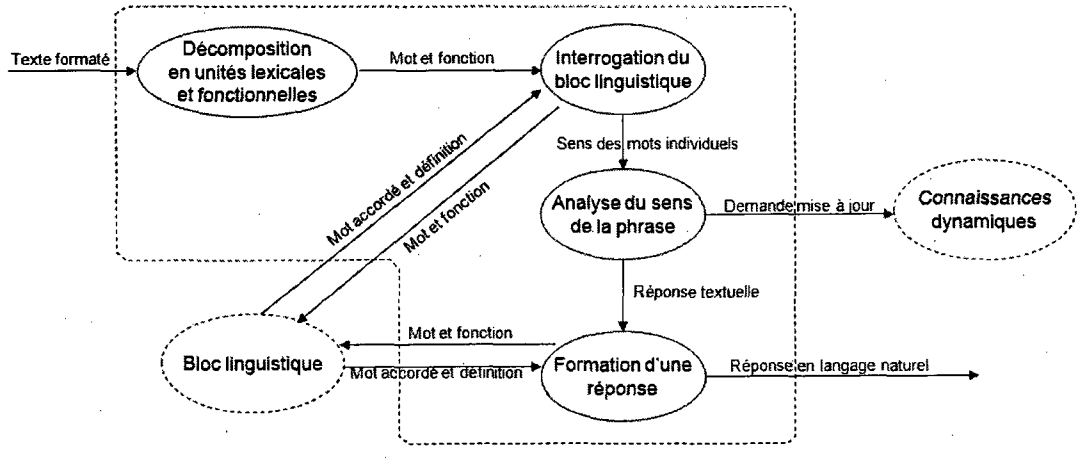

Figure 3.4 – Détails du module *Interpréteur de sens*

L'*interpréteur de sens* reçoit le texte formaté de l'*analyseur syntaxique*. L'*analyseur syntaxique* place chaque terme dans un ordre prédéterminé afin que ceux-ci soient analysés

par l'*interpréteur de sens*. Il est inutile d'essayer d'interpréter un sujet de phrase comme si celui-ci était le complément, ou encore un verbe comme si celui-ci était un déterminant. Cet ordonnancement des mots donne à l'*interpréteur de sens* quel mot possède quelle fonction. Par la suite, les mots sont pris l'un après l'autre et sont envoyés vers le *bloc linguistique* afin d'obtenir leur définition. Par la suite, une fois que toutes les définitions ont été trouvées, le sens de la phrase est analysé. Si une action doit être prise, elle est exécutée à ce moment. Une mise à jour est faite sur l'état et les connaissances de la machine lorsque la phrase demande une prise d'action comme bouger. L'action peut être faite en parallèle à une réponse textuelle. Cependant, elle n'est pas faite systématiquement, seulement lorsque l'analyse de la phrase le demande. Ensuite, une réponse textuelle est générée. Pour ce faire, l'*interpréteur de sens* requiert l'aide du *bloc linguistique* à nouveau afin d'organiser et d'accorder les mots adéquatement afin de former une phrase dans un français correct. Ces phrases ne sont pas préconstruites, mais bâties en utilisant les règles grammaticales du *bloc linguistique* afin de construire, à partir des mots formant la réponse, une phrase grammaticalement correcte. Par réponse en langage naturel, il est question d'un message textuel affiché à l'écran, compréhensible par les êtres humains, et qui pourrait être envoyé vers un engin de synthèse vocale, comme suggéré à la figure 3.2.

En reprenant l'exemple présenté à la section 3.2, une question telle que « *Où est le livre ?* » résulte en une réponse de type «*Le livre est sur la table*», sans aucune mise à jour des connaissances. Une phrase telle «*Je m'appelle Simon-Pierre.*» résulte en un changement d'interlocuteur ayant le nom *Simon-Pierre* et la machine répond par une salutation du type «*Bonjour Simon-Pierre*». La première phrase est construite sur le vif lors de l'interprétation, contrairement à la seconde qui est partiellement préconstruite. Le fait de répondre automatiquement *Bonjour* ne constitue pas une lacune dans l'architecture puisque l'être humain répond lui aussi par réflexe à une salutation. L'utilisation de formes automatiques de réponses, dans certains cas précis, ne compromet donc pas l'objectif global du projet et est utilisé pour accélérer le temps de réponse. De plus, cette réponse permet d'indiquer à l'utilisateur que la machine a bien interprété le sens de la phrase et que les actions demandées ont bien été effectuées. Le type de réponses et leurs constructions sont discutés dans le module *bloc linguistique*, section 3.4, et au chapitre 4, lors de la présentation des résultats.

Globalement, l'interpréteur de sens reçoit des mots en paramètres déjà positionné par leur fonction générale dans la phrase par l'*analyseur syntaxique*. La forme *interprete(Sujet, Verbe)* ou *interprete(Sujet, Verbe, Complément)* est utilisée afin de lancer la recherche de

la signification et de la forme semblable est utilisée pour initier la génération de la phrase par le *bloc linguistique*.

3.4 Bloc linguistique

Le *bloc linguistique* regroupe les éléments qui sont nécessaires à l'utilisation de la langue à interpréter. Ce module regroupe la grammaire, le conjugueur, le dictionnaire de l'architecture ainsi que la classification des connaissances. Lorsqu'un changement de langue est désiré, ce module est remplacé par son équivalent dans la nouvelle langue. Le détail des sous-modules est présenté dans les sections qui suivent. Les détails internes des interactions du *bloc linguistique* sont présentés à la figure 3.5.

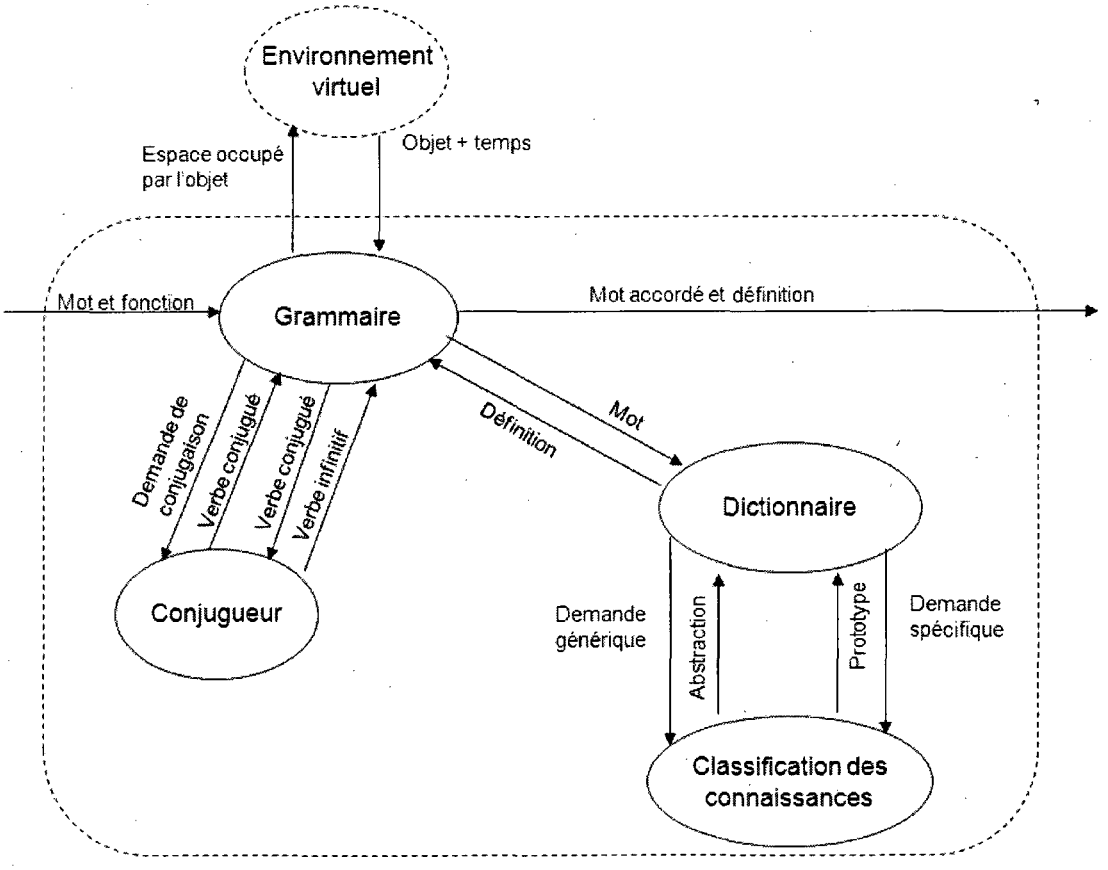

Figure 3.5 – Détails du module *Bloc linguistique*

Tout d'abord, ce module reçoit de l'*interpréteur de sens* un mot et sa fonction pour être défini et, au besoin, accordé. Par exemple, lorsque Hugo demande à RoDo «Où sont les livres ?», RoDo aura besoin de définir «livre», en interrogeant son dictionnaire, afin de savoir ce que Hugo lui demande. Par la suite, RoDo devra accorder «livre» au pluriel pour

répondre «Les livres sont dans la bibliothèque». Que ce soit pour la définition ou l'accord, les mots sont d'abord traités par la grammaire. Si le mot est un verbe, il est envoyé vers le *conjugueur* pour être mis à l'infinitif et retrouver sa définition ou il est conjugué pour générer une phrase. S'il s'agit d'un autre mot qu'un verbe, la *grammaire* ramène le mot sous sa forme simple, appelée forme canonique, et l'envoie au *dictionnaire* pour avoir sa définition. La détection du verbe ou non verbe est faite dans l'*analyseur syntaxique* et le *bloc linguistique* reçoit cette information de l'*interpréteur de sens*. La *grammaire* ne fait que diriger le mot vers le bon module pour le neutraliser. Selon le type de mot, le *dictionnaire* peut faire appel au module de *classification* pour avoir le générique du mot où encore un spécimen prototypique spécifique du mot. Dans les autres cas, le *dictionnaire* renvoie simplement la définition fonctionnelle connue du mot. Cette définition est renvoyée à la *grammaire* et le mot est accordé selon les besoins, par exemple en génération de phrase. La grammaire consulte également le *dictionnaire* pour les questions d'élision. Le *dictionnaire* renvoie les formes alternatives existantes à la *grammaire*. Finalement, la *grammaire* peut demander au module *environnement virtuel* des informations sur les mots étant définis comme des objets présents dans l'environnement afin d'avoir plus d'information sur l'objet en question. Le mot accordé et sa définition sont obtenus et, par la suite, renvoyés vers l'*interpréteur de sens*.

Par exemple, si Hugo demande à RoDo «Où sont les livres?», le *bloc linguistique* agit comme suit. Le *où* se trouve dans le *dictionnaire* et sa définition est *marqueur interrogatif de lieu*, le *sont* est un verbe et donc, il passe dans le *conjugueur* où il est défini en tant que *verbe être*, le *les* est dans le *dictionnaire* et sa définition est *déterminant* et, finalement, le *livre* passe par le *dictionnaire* qui l'envoie vers la *classification* découvre que c'est un *objet*. Ces informations sont toutes données à l'*interpréteur de sens* qui, par la suite, demande au *bloc linguistique* de générer une phrase de localisation avec [*livre,être,localisation du livre*], puisque l'*interpréteur de sens* ne connait pas cette information et que RoDo doit répondre à cette question pour former sa phrase. La *grammaire* génère le sujet *les livres*, puis fait accorder *être* au présent avec *les livres*. Le *conjugueur* lui renvoie *les livres sont*. La *grammaire* consulte ensuite les informations sur l'*environnement* et découvre que l'objet *livres* est *dans* la *bibliothèque*. Finalement, le complément *la bibliothèque* est formé et la phrase «Les livres sont dans la bibliothèque» est formée.

Le module *bloc linguistique* est conçu de manière à pouvoir être mis à jour rapidement et efficacement. Il comporte des faits et des règles, tels que retrouvés dans la langue utilisée. Ainsi, pour ajouter ou modifier une règle de grammaire, ou un mot dans le dictionnaire avec sa définition, il suffit d'y apporter les modifications en suivant le format des autres

règles ou faits présents. Par exemple, pour ajouter dans le dictionnaire le mot *livre*, il faut indiquer le type et le genre sous la forme *nom(commun,mm,livre)*, *mm* indiquant un mot strictement masculin, et sa définition venant de la classification, une entrée de type *classup(livre,objet)* se trouve dans le module *classification des connaissances*. De plus, le livre se trouve dans l'*environnement virtuel*, donnant des dimentions et une forme au livre. Ces définitions, bien que minimalistes, permettent à la machine de conceptualiser les mots. Un système de définitions plus complexes pourrait être implémenté en modifiant le dictionnaire, en s'assurant cependant que l'entrée soit toujours un mot neutralisé et qu'en sortie les mots renvoient les informations nécessaires pour son accord et une définition fonctionnel du mot. Également, l'ajout de la connaissance des synonymes pourrait être fait en ajoutant un sous-module qui interagirait avec le dictionnaire ou simplement en ajoutant une interface reliant les mots ayant la même définition.

3.4.1 Grammaire

Dans cette section, lorsqu'il est question de grammaire, il n'est question que de l'axe syntagmatique de la grammaire (accords et syntaxe) et non pas de l'axe paradigmatique (liens avec les substituts potentiels) de celle-ci. Dans le langage humain, un certain nombre de règles de grammaire définissent un modèle *règles et exceptions*. Celles-ci se retrouvent, par exemple, dans le *Précis de grammaire française* [GREVISSE, 1995]. Ainsi, pour qu'une machine soit en mesure d'utiliser et de comprendre les règles grammaticales qu'une langue utilise, il faut encoder ces règles grammaticales dans un langage qu'elle comprend. Donc, en encodant les règles et les exceptions, il est possible de permettre à la machine d'avoir un équivalent grammatical de la langue désirée dans son propre langage. La figure 3.6 montre le flot de données à l'intérieur du module *grammaire*. Un diagramme de séquence est également donné à la figure 3.7.

Un mot est soumis à la *grammaire* et est neutralisé afin d'obtenir sa forme canonique telle que présente dans le dictionnaire. Pour des fins d'exemple, *chatte* et *mangeait* sont utilisés. Si le mot est un verbe, il passe par le *conjugueur*, sinon, la *grammaire* utilise ses règles pour transformer un pluriel en singulier et utilise ses règles pour retrouver le masculin à partir du féminin. Si l'une des transformations est impossible, par exemple, *fleur* ne peut pas être mis au masculin, la transformation est tout simplement ignorée. Avec les exemples, les mots *chat* et *manger* sont obtenus. Avec ces mots, la *grammaire* interroge le *dictionnaire* et les informations sur l'*environnement* afin d'obtenir des informations supplémentaires sur les mots, comme leurs définition, leurs formes élidées ou leur emplacement dans l'environnement. Ici, le mot *chat* est associé avec la définition *nom com-*

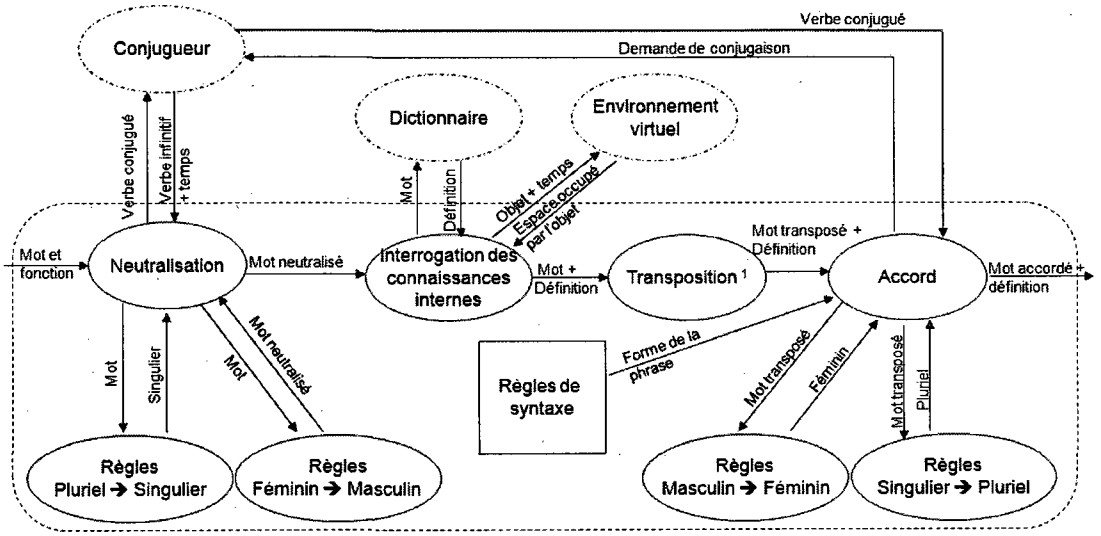

Figure 3.6 – Détails du module *Grammaire*

mun, mammifère félinidé et le mot *manger* avec *verbe action.* Pour le moment, *manger* n'a pas besoin d'une définition plus élaborée puisqu'il n'est considéré que sous sa forme intransitive dans tous les cas et que, sous cette forme, il n'a pas d'impacte directe dans l'*environnement virtuel* de la machine. Toutefois, celle-ci pourrait impliquer le fait que le complément soit *ingéré* par le sujet dans le cas où la compréhension de la forme transitive du verbe serait ajoutée.

Par la suite, le mot subit un certain nombre de transpositions si nécessaire pour prendre une forme plus adéquate. Les différentes transpositions possibles sont l'élision, la contraction et le changement de personnes. Ces transposition sont détaillées un peu plus loin dans cette section. Si aucune transposition n'est nécessaire, cette action est ignorée et le mot reste inchangé. Finalement, le mot est réaccordé selon les besoins de la phrase. Pour un verbe, le module *grammaire* envoie le mot au *conjugueur* pour conjuguer le verbe au bon temps et à la bonne personne. Pour les noms et les adjectifs, le module utilise les règles du féminin suivit de celles du pluriel, au besoin, pour accorder les mots selon l'accord désiré dans la phrase à générer. Dans ce cas, *chat* redevient *chatte* et *manger* devient *mangeait.* Ensuite, le mot accordé avec sa définition est envoyée en sortie pour obtenir le sens de la phrase à partir du sens des mots, tout en conservant les mots initiaux.

La *grammaire* est donc importante pour la compréhension de la langue qui sert à retrouver les unités lexicales et ainsi trouver leurs sens. La figure 3.7 montre un diagramme de séquence pour les deux cas qui viennent d'être expliqués.

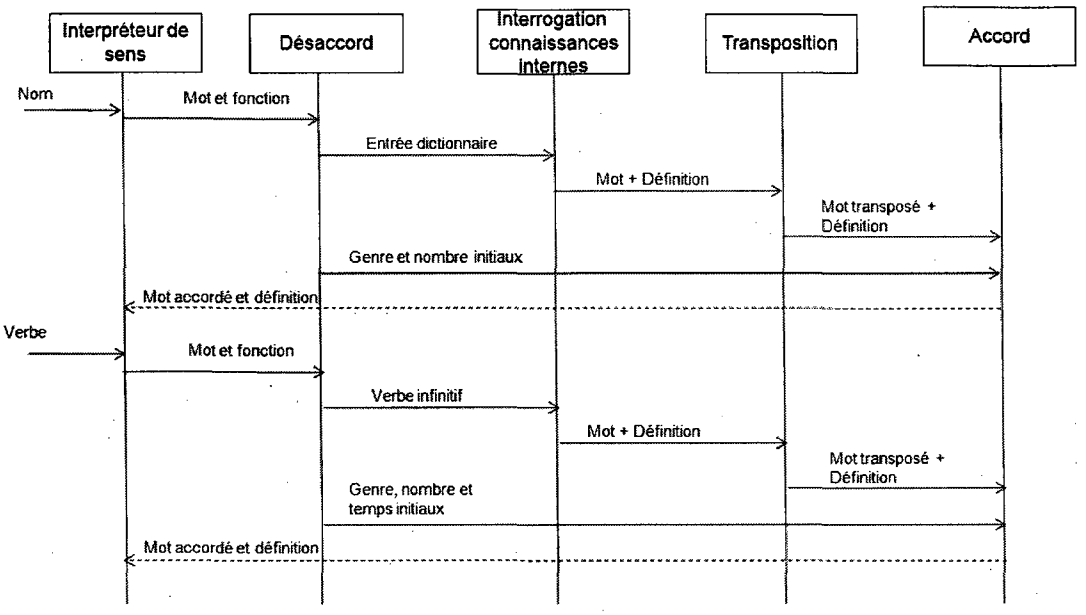

Figure 3.7 – Diagramme de séquence du module *Grammaire*

La *grammaire* est essentiellement composée de règles et de leurs exceptions. Dans la grammaire utilisée pour ce projet, les règles de formation du féminin et du pluriel des noms et des adjectifs ainsi que la syntaxe d'une phrase pour sa construction en place permettent la formation automatisée du féminin et du pluriel selon les besoins de la phrase. Ces règles sont de la forme *Regle(In,Out) :- concat(ST,Term,In), concat(ST,Term2,Out)* où *Regle* est la règle qu'il faut appliquer, soit *pluriel* ou *féminin*, *In* est le mot entré, *Out* le mot accordé, *ST* est le radical du mot, *Term* est la terminaisons qui doit être vérifiée et *Term2* la terminaison qui doit être mise pour faire l'accord. Par exemple, pour le pluriel des adjectifs en *al*, la règle sera *plurAdjQlf(Sing,Plur) :- concat(ST,'al',Sing), concat(ST,'aux',Plur)*. Les exceptions pour leur part sont écrit simplement en marquant le mot initial et sa version accordée. Par exemple, *femininNoms(homme,femme)* indique que le féminin de *homme* est *femme*. Dès qu'une exception est détectée, les règles ne sont pas vérifiées pour ce mot.

Toutefois, cette transformation n'est possible que si le dictionnaire autorise cet accord. En effet, certains mots sont invariables, certains pour ce qui est du genre, d'autres pour ce qui est du nombre et certains termes sont épicène, c'est à dire qu'ils ont les deux genres. Outre le cas des épicènes pour qui les règles s'appliquent, la *grammaire* considère ces mots comme des exceptions et n'appliquera pas la règle correspondante.

À partir des règles du pluriel et du féminin, il est également possible de faire le chemin inverse et de retrouver le masculin et le singulier d'un mot, toujours si le dictionnaire

permet l'accord. Toutefois, il y a certaines anomalies dues à des règles qui se chevauchent, par exemple, les noms en *-el* et en *-eau* ont une forme féminine en *-elle.* Ainsi, pour éviter ce problème, il faut ajouter une vérification de la présence des mots dans le dictionnaire. Donc la grammaire, lorsqu'elle recherche le masculin d'un mot, vérifie que celui-ci existe dans son dictionnaire avant d'affirmer que c'est bel et bien le masculin du mot demandé. Ceci limite un peu la fonctionnalité, mais est nécessaire pour ne pas avoir de fausse forme du masculin à partir du féminin.

Il existe trois types de transpositions, la première est le changement du *je* en *tu* et inversement. En effet, lorsque le premier interlocuteur demande au second une phrase du genre «*Comment vas-tu ?*», il faut s'attendre à ce que le second interlocuteur lui réponde «*Je vais bien.*» et non pas «*Tu vas bien.*» Il en va de même pour le *nous* et le *vous.* Une simple formule mathématique permet de faire cette transposition. Le détail de cette formule se trouve à la section 4.2.1.

La seconde transposition est l'élision. Celle-ci consiste à modifier certains mots se terminant par des voyelles lorsque le mot qui les suit commence également par une voyelle ou un «H» muet. Par exemple, «le oiseau» devient «l'oiseau» et «le hôpital» devient «l'hôpital». Le dictionnaire connaît ce que sont les voyelles, et leurs formes dérivées, ainsi que la lettre «H». Un appel au dictionnaire est donc fait pour identifier si un nom commence par une voyelle où un «H» suivit d'une voyelle. Bien que la réelle détection doivent être fait au niveau de la phonétique du mot, et non pas de la présence de la voyelle, cette technique a été utilisé afin de simplifier l'architecture. Le dictionnaire est également utilisé pour savoir si une forme élidée existe pour un mot donné. Lorsque le *dictionnaire* identifie que la première lettre du mot demandé est une voyelle, et ce, indépendamment du fait que le mot existe dans le dictionnaire ou non, la grammaire vérifie dans le dictionnaire que le mot qui le précède dans la phrase possède une forme élidée. Si c'est le cas, la *grammaire* doit modifier le mot précédent pour que celui-ci prenne sa forme élidée. Le *dictionnaire* ne tient pas compte de la phonétique des mots puisque son dictionnaire phonétique n'a pas été implémenté par choix de simplification du design. Donc, la différence entre un «H» muet et un «H» aspiré n'est pas détectée comme il le devrait ce qui cause que tous les mots commençants par un «H» et dont la seconde lettre est une voyelle sont considérés comme devant générer une élision. Par exemple, pour les mots *éléphant* et *hôpital* , l'article *le* est élidé en *l'* alors que dans le cas du mot *moteur*, l'article est laissé tel quel. Le mot *hangar*, qui devrait normalement laisser l'article tel quel, provoque néanmoins l'élision. Des pistes de solutions sont présentées au chapitre 4 dans le but de régler cette situation.

La dernière transposition qui peut être effectuée est la contraction. Si la phrase, après le passage de l'élision, comporte la séquence *à le* ou la séquence *de le*, elle les remplace par *au* ou *du* selon le cas. Il faut cependant attendre la fin de l'élision avant de prendre cette décision puisque *à l'* suivit d'un nom masculin ne devient pas *au*. Par exemple, lors d'une localisation, la phrase «*l'oiseau est à le nord de le château*» est générée, elle doit être transposée suivant la contraction «*l'oiseau est au nord du château*» alors que «*le château est à le sud de l'oiseau*» devient «*le château est au sud de l'oiseau*».

3.4.2 Conjugueur

Pour le projet, le *conjugueur* fait référence à une simplification du conjugueur proprement dit puisqu'il ne traite pas toutes les flexions, c'est-à-dire toutes les formes conjugales possibles. Il contient les temps suivants : l'infinitif présent, l'indicatif présent, l'indicatif imparfait, l'indicatif futur simple, le participe présent, le participe passé, le conditionnel présent et l'indicatif passé composé. De plus, il se limite à certains verbes ciblés. Le module *conjugueur* sert à faire du traitement sur les verbes, au même titre que la grammaire qui agit sur les autres mots. Il s'agit soit de prendre un verbe conjugué et de le ramener à l'infinitif pour analyser son sens, soit de prendre l'infinitif et de conjuguer le verbe au temps et à la personne adéquate pour formuler une réponse. Le conjugueur est construit selon une base de règles pour les verbes de formes régulières, comme le verbe *aimer* et une liste d'exceptions pour les formes irrégulières, comme le verbe *être*. Ces règles et ces exceptions forment la table de conjugaison. Le modèle utilisé est donc un modèle semblable à ce qui est présenté par *L'Art de conjuguer* [BESCHERELLE, 1980]. La figure 3.8 montre le traitement des données à l'intérieur de ce module.

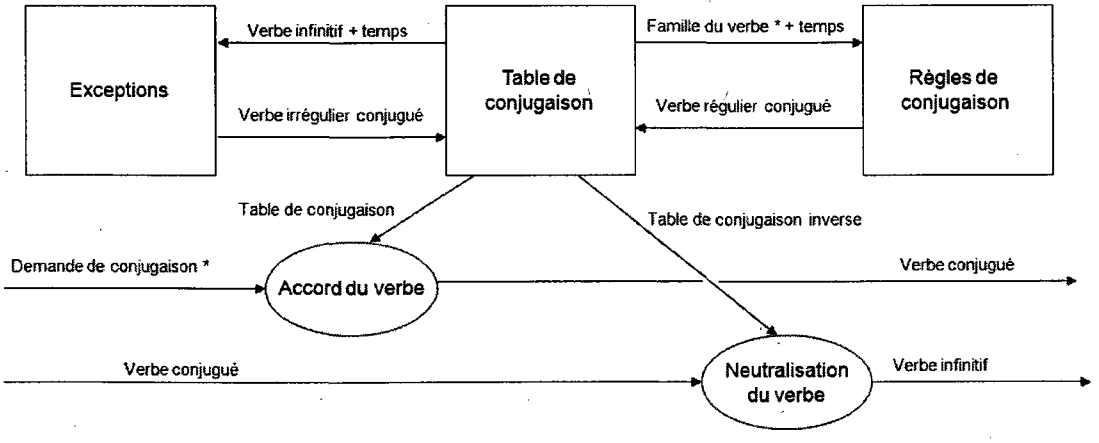

Figure 3.8 – Détails du module *Conjugueur*

Sur la figure 3.8, la famille du verbe comprend l'infinitif du verbe, son groupe de terminaison et son radical. La demande de conjugaison comprend, pour sa part, l'infinitif du verbe, la personne et le temps désiré.

Afin de limiter la grandeur de la table, les règles de conjugaisons ont été implémentées plutôt qu'une simple table de conjugaisons remplie verbe par verbe à chaque temps et à chaque personne. C'est-à-dire que l'utilisation de la forme *radical + terminaison* est préférée à un tableau de conjugaison. Les règles de conjugaison ont donc besoin, en paramètre, du groupe de terminaison, le temps du verbe, la personne et le nombre. La règle renvoie la terminaison et le *conjugueur* ajoute cette terminaison au radical du verbe. Également, vu le grand nombre d'exceptions, un changement de radical, pour certains temps de verbe, a été intégré de façon à pouvoir réutiliser les règles et ainsi éviter d'avoir un trop grand nombre d'exceptions. À l'instar de certaines règles de grammaire, dont les exceptions sont également régies par des règles, plusieurs verbes, même irréguliers, respectent les règles de conjugaisons du moment que le radical est modifié. Par exemple, le verbe *faire*, qui est considéré irrégulier, n'est en fait irrégulier, au présent de l'indicatif, que pour la deuxième et la troisième personne du pluriel. Ainsi il n'y a que les exceptions *vous faites* et *ils/elles font* et les règles de conjugaison régulières sont utilisées pour toutes les autres personnes de ce temps.

3.4.3 Dictionnaire

Il faut représenter le dictionnaire de la machine de manière semblable aux dictionnaires utilisés par les humains afin que la machine et l'humain aient une base commune sur les mots et leurs significations. Cependant, étant donné qu'il n'est pas possible pour le module *dictionnaire* d'interpréter une définition inscrite sous forme de phrases, comme un dictionnaire *humain*, tel *Le petit Robert* [ROBERT, 1996], il faut trouver une forme de définition simple que la machine peut comprendre.

Trois formes de définitions sont utilisées :

1. les définitions directes, par exemple *le* est un déterminant identifiant un item *connu* par les interlocuteurs,

2. les définitions formalisables, comme la localisation, par exemple *sur* suppose que la base de l'objet est posé sur la face supérieur d'un autre objet,

3. les définitions hiérarchiques, telles que *chien* est un *mammifère*.

La dernière forme est traitée plus en détail dans la section 3.4.4. Pour ce qui est des deux premières formes, les définitions directes sont sous forme de table de faits et les définitions formalisables sont sous forme de traitement d'informations et de prises de décision. La figure 3.9 montre l'architecture du module *dictionnaire*.

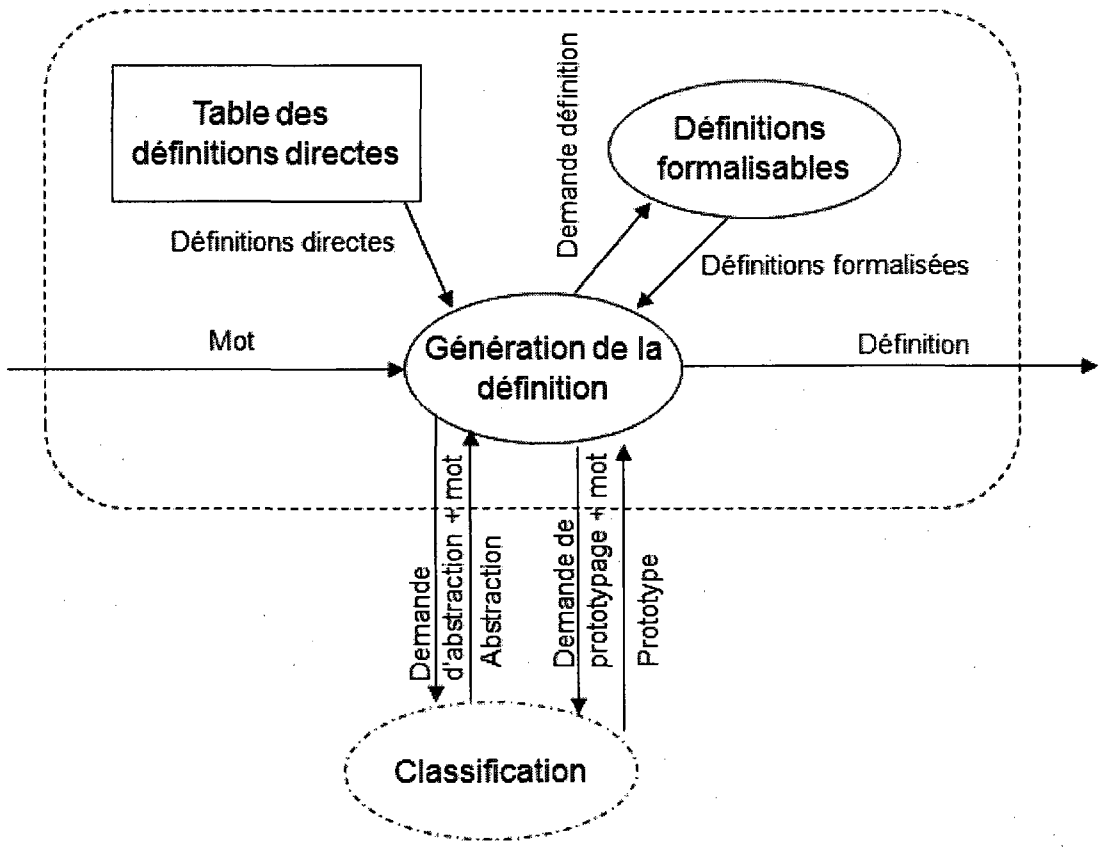

Figure 3.9 – Détails du module *Dictionnaire*

Lorsqu'un mot est soumis au *dictionnaire*, celui-ci regarde les différents types de définitions afin de trouver celle qui correspond au mot reçu.

Les définitions directes sont représentées sous forme de faits représentant les caractéristiques du mot, comme son genre, son type et d'autres informations particulières selon le type. Contrairement à un dictionnaire comme *Le petit Robert*, le module *dictionnaire* se base sur les fonctions plutôt que sur les mots, l'idée étant d'aider à la structure des phrases. Typiquement, on cherche un déterminant et non pas tous les mots du dictionnaire qui sont des déterminants. Cependant, une fonction d'interface qui inverse le fait de chercher une fonction par les mots ayant cette fonction pourrait servir pour un dictionnaire basé sur les mots. Ainsi, dans le *dictionnaire*, les mots sont entrés sous la forme *fonction(Mot,Definition)* plutôt que mot(Fonction,Definition). Par exemple, les entrées *de-*

terminant(le,Def) et *nom(commun,m,homme,Def)* font partie du *dictionnaire*, avec leur définition.

Pour les définitions formalisées, elles sont écrites sous forme de règles. Par exemple, *sur* est défini comme étant le fait que la surface inférieure d'un premier objet est positionnée à une valeur d'élévation égale à la surface supérieure d'un deuxième objet, tout en étant entre les limites de bordure de ce dernier. Mathématiquement, il est possible de représenter la relation ainsi : $X_{2i} \leq X_1 \leq X_{2s} \& Y_{2i} \leq Y_1 \leq Y_{2s} \& Z_{1i} = Y_{2s}$, où l'indice 1 représente l'objet évalué et l'indice 2 l'objet sur lequel il est déposé, l'indice i étant la limite inférieure et l'indice s la limite supérieur et X, Y et Z les positions selon un référentiel donné.

Ainsi, en obtenant les mesures qui viennent de l'environnement ou en voulant indiquer quelque chose en relation avec l'environnement, la notion chiffrée de ces connaissances permet à la machine d'avoir la même définition que son interlocuteur. Présentement, les informations de type flou, comme la température et la grandeur, sont fixées par le programmeur, mais il serait possible de les rendre adaptatives.

Pour ce qui est des définitions hiérarchiques, le dictionnaire fait appel au module de *classification des connaissances* (section 3.4.4) pour obtenir soit le prototype (parmis les spécifiques) ou l'abstraction (générique) permettant d'avoir la définition ou l'exemple désiré.

3.4.4 Classification des connaissances

Tous les concepts utilisés pour ce module proviennent de la section 2.1 traitant des modèles linguistiques. La *classification des connaissances* sert pour les définitions de la forme hiérarchique du *dictionnaire*. Elle correspond à la classification internationale et elle est en lien avec ce qui a été présenté par Kleiber. Elle reprend le principe des sous-ensembles pour retrouver l'entité désirée. Également, les principes de base de la sémantique du prototype et le principe des conditions nécessaires suffisantes servent à demander à la machine un élément des sous-ensembles. Finalement, elle se base sur de la double classification, horizontale et verticale, pour gérer son arbre de classification.

Par exemple, les chiens, les chats et les humains sont des mammifères. Les mammifères et les oiseaux sont des vertébrés. Les vertébrés et les invertébrés sont des animaux. Les animaux et les plantes sont des êtres vivants. Si l'ancêtre est connu, alors les descendants des ancêtres sont connus. Donc, si Hugo demande à RoDo un animal, RoDo peut lui donner n'importe quel animal qu'il connait en prenant un membre du bon sous-ensemble.

Ainsi, il ne donnera pas un membre de la famille des plantes puisque les plantes ne sont pas présentes dans l'ensemble des animaux. La figure 2.2/*2.1*/ illustre bien la pyramide des classifications. La figure 3.10 présente comment une abstraction ou un prototype est donné selon ce qui est demandé au module.

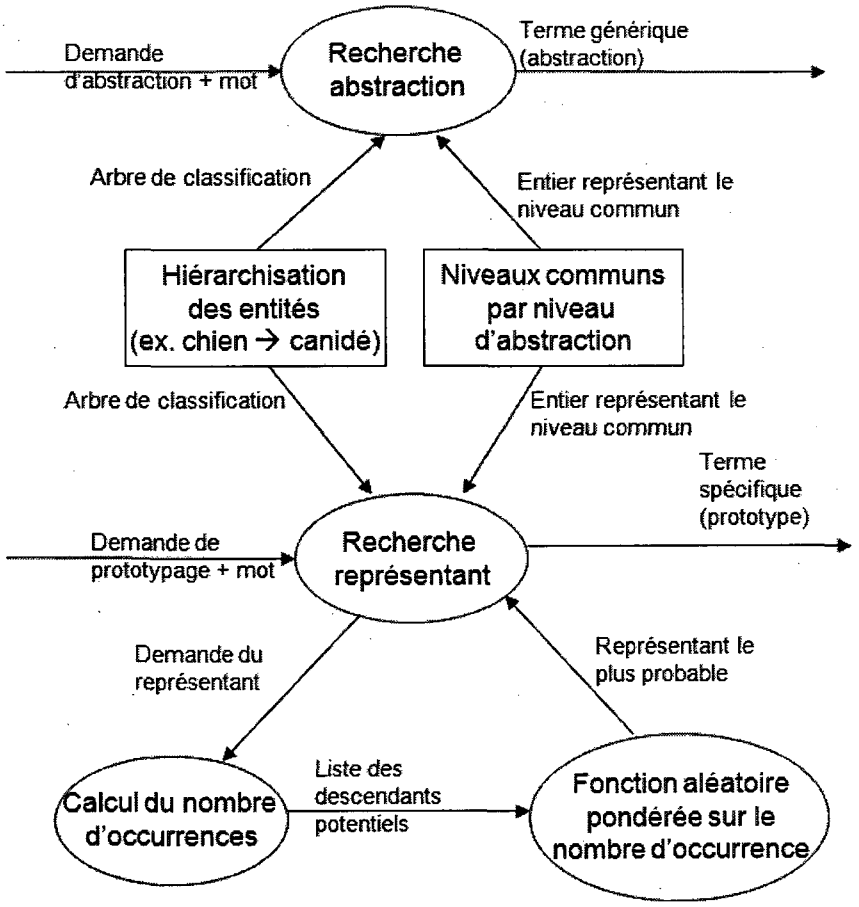

Figure 3.10 – Détails du module *Classification des connaissances*

La sémantique du prototype a été choisie sous une forme simplifiée et a été implémentée à l'aide d'une décision aléatoire pondérée sur le nombre d'occurrences. Contrairement aux êtres humains qui se font un modèle de ce que doit être un oiseau, par exemple, le module utilise comme prototype l'entité rencontrée le plus fréquemment dans son environnement, tout en laissant la possibilité aux autres entités de la même classe d'être sélectionnées. Par exemple, même si le chien et le chat ont plus de chances de ressortir lorsque l'utilisateur demande à la machine un animal, un perroquet, un dauphin, un chimpanzé ou un humain pourrait tout aussi bien être donné en réponse. Pour ce faire, une fonction aléatoire pondérée est utilisée afin de déterminer quelle entité utiliser comme prototype de chaque classe. Le module connait les parents de chaque classe pour permettre la hiérarchie, mais

ne peut pas déterminer par lui-même à quelle classe appartient un objet rencontré pour la première fois.

De plus, pour se rapprocher du modèle humain de la classification, le niveau commun n'est pas le même pour tous les niveaux de la hiérarchie et il n'est pas non plus le même pour les différentes branches de la hiérarchie. Ainsi, le niveau commun ascendant et descendant des entrées est indiqué de façon à ce que la demande d'un antécédent ou d'un descendant se fasse selon le niveau commun de la langue. Par exemple, lorsque la machine se fait demander, *Nomme-moi un animal*, il n'est pas nécessaire qu'elle réponde *un caniche royal argenté*, qui est néanmoins une réponse valide. La réponse *un chien* est une réponse suffisante. Toutefois, *mammifère* ou *canidé*, qui sont des niveaux intermédiaires de la classification, ne sont pas utilisés couramment dans la langue et sont donc des réponses à proscrire. Le niveau commun de la langue est défini selon chaque mot et est configurable au sein même du module en donnant, à chaque niveau, le nombre de niveaux hiérarchiques nécessaire pour atteindre le bon niveau de la langue pour les génériques et les spécifiques.

3.5 Environnement virtuel

Le module *environnement virtuel* sert à la validation des concepts. L'*environnement virtuel* correspond à la représentation *mentale* que se fait la machine de son *environnement réel.* Comme indiqué précédemment, le système complet et les données sur l'environnement réel ne sont pas accessibles, ce qui fait que l'état courant de l'environnement réel doit être entré manuellement. Tel que mentionné précédemment, ce module doit être adjoint à un module faisant l'acquisition des informations sur l'environnement réel pour une application dans un système complet.

Une petite pièce avec différents objets a été choisie comme environnement virtuel. Cet environnement permet, entre autres, de valider les concepts formalisés et hiérarchiques du dictionnaire. Il permet également de créer des déplacements pour la validation du passé et du présent à l'aide du module *connaissances dynamiques* qui est présenté à la section 3.6. L'architecture du module *environnement virtuel* est montré à la figure 3.11. L'architecture présentée ne prend pas en considération l'acquisition des données de l'environnement réel.

L'*environnement virtuel* est conçu de telle sorte que des formes 3D sont disposées géométriquement dans un référentiel cartésien et chacun de ces objets correspond à une entité bien précise. On y retrouve des prismes (lits, tables, tabourets, enveloppes, humains et chiens), des sphères (balles, lampes et chats), et des cubes (blocs et ordinateurs). Ces

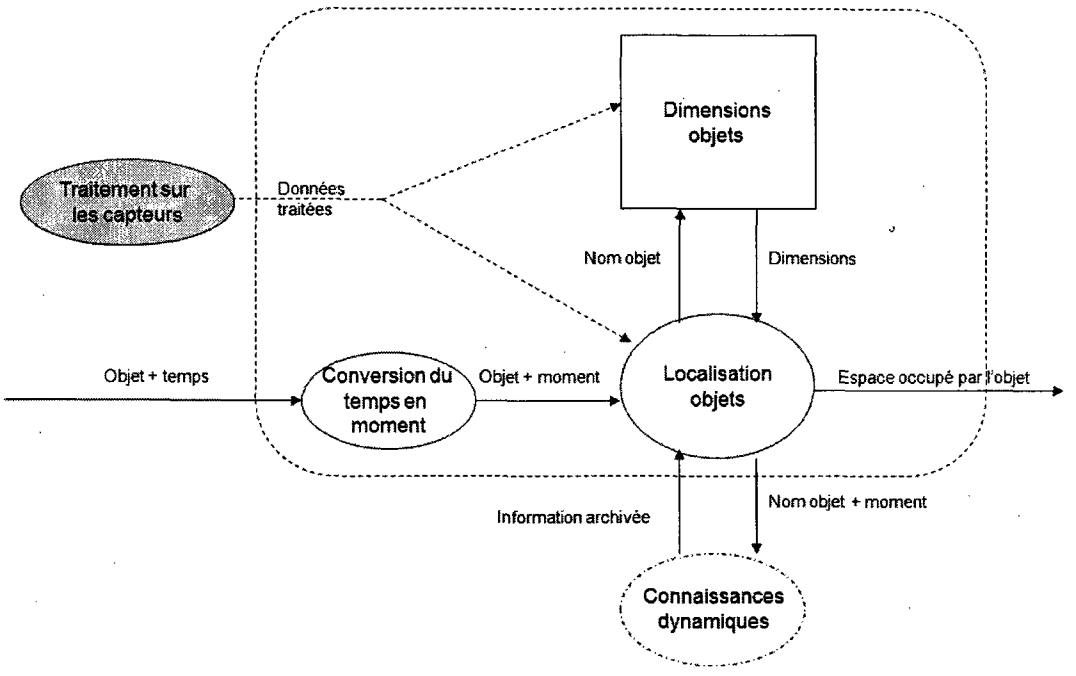

Figure 3.11 – Détails du module *Environnement virtuel*

objets ont des dimensions bien définies et sont disposés dans l'environnement avec des coordonnées cartésiennes (X, Y, Z). L'idée étant de savoir si la machine peut situer les différents objets les uns par rapport aux autres et de faire la différence entre les différentes entités lorsque des questions lui sont posées. Un exemple de positionnement dans l'*environnement virtuel* est présenté à la figure 3.12.

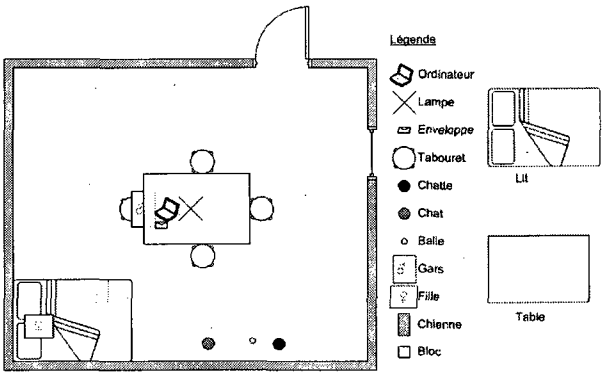

Figure 3.12 – Exemple d'environnement virtuel

De plus, le temps du verbe est utilisé afin de déterminer si l'information doit être recherchée parmi les connaissances actuelles ou les informations archivées (voir la section 3.6).

Afin de différencier chaque entité, le couple (ID, OBJ) permet de rattacher un identifiant à un objet. Grâce à ce couple, il est possible de donner soit l'objet ou son identifiant lors de la génération de la réponse pour parler du même objet. Présentement, l'architecture renvoie le terme utilisé par l'interpréteur, mais il est possible d'utiliser l'autre membre du couple. Ainsi, lorsque la question *Où est Moka ?* est posée, il est aussi facile de répondre *Moka est sur le lit* que *le chat est sur le lit.*

3.6 Connaissances dynamiques

Le module *connaissances dynamiques* est un historique des états passés et présents de l'*environnement virtuel.* Ce module sert à ce que la machine ne donne pas uniquement les états courants, mais qu'elle ait une connaissance générale du temps qui passe puisque l'univers n'est pas statique. Également, la machine peut actualiser certains faits avec ce module, comme le nom de son interlocuteur, les personnes qu'elle connait, ou tout autre information dynamique. La figure 3.13 montre le fonctionnement interne des connaissances dynamiques.

Sur la figure 3.13, il y a en fait deux types d'interactions. La première est la mise à jour des connaissances. Dans ce cas, les faits actuels sont enregistrés comme étant passés et les nouveaux faits sont enregistrés comme étant l'état actuel. Ensuite, la machine actualise ses connaissances selon cette nouvelle situation. La seconde est la demande d'information contenue dans les archives. Dans ce cas, il n'y a aucune modification des connaissances.

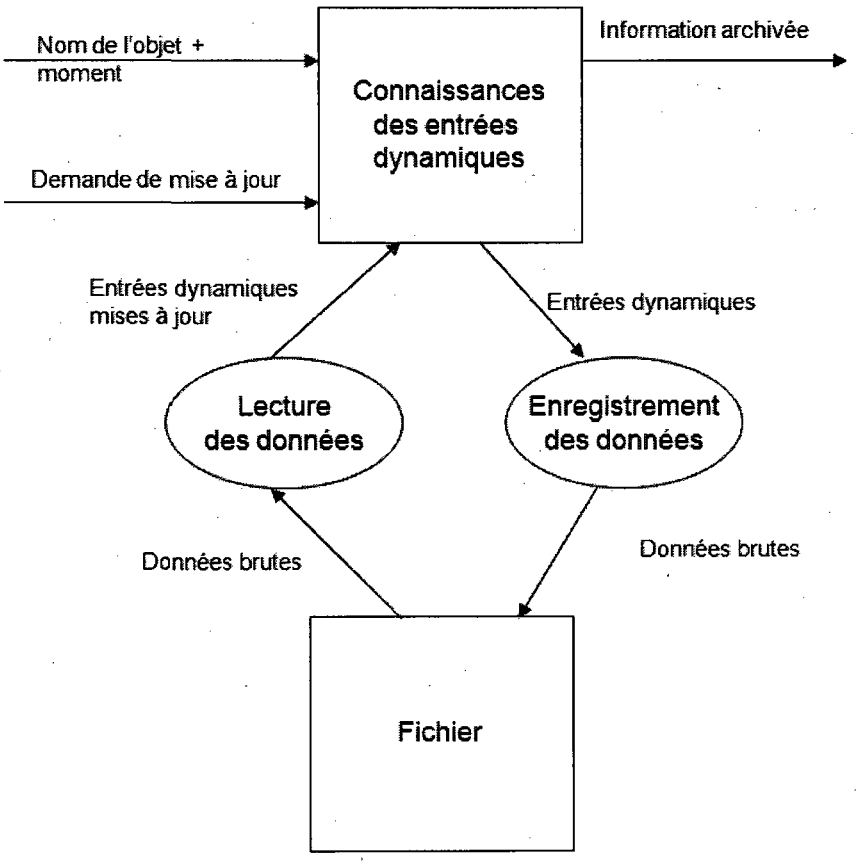

Figure 3.13 – Détail du module *Connaissances dynamiques*

CHAPITRE 4

TESTS, RÉSULTATS ET ANALYSES

Ce chapitre montre différents tests significatifs qui ont été effectués avec leurs critères de réussite ainsi que les résultats obtenus. Les tests montrés dans ce chapitre ne sont pas exhaustifs, car beaucoup d'autres ont été effectués. Seuls les tests les plus significatifs sont donnés ici. Les résultats de ces tests sont ensuite analysés et, dans certains cas, des pistes d'améliorations sont proposées.

Les tests ont été effectués sur un environnement simple et normalisé. La figure 4.1a présente l'environnement initial utilisé pour les tests et la figure 4.1b présente l'environnement modifié pour les tests avec le module *connaissances dynamiques*. Il est à noter que sur ces figures, les éléments se trouvant sous quelque chose, comme le lit ou la table, ont été pâlis alors qu'ils sont foncés lorsqu'ils se trouvent par-dessus. Cependant, les tons de gris ne sont pas représentatifs de leur élévation globale. Également à noter que les oreillers, la couverture, les murs, la porte et la fenêtre ont été ajoutés à la figure pour une question d'esthétique, mais n'existent pas dans la version du micro-environnement qui sert aux tests. Cet environnement, bien que simplifié, permet de tester les fonctionnalités générales de l'architecture et de valider l'architecture générique proposée. Cet environnement est constitué du nécessaire pour vérifier le bon fonctionnement des différents modules.

Les modules développés pour cette architecture ont été faits en Prolog. Bien que ce langage ne soit pas optimal pour tous les aspects de l'architecture, cela évite d'avoir à entretenir du code dans des environnements différents, à savoir une base de données, du code en langage orienté objet, du code en Prolog et tout autre environnement qui sont mieux adaptés pour certains modules. D'ailleurs, certaines améliorations sont suggérées lors de l'analyse de chaque module.

Aux fins des tests, des identificateurs donnant un nom spécifique à une entité ont été ajoutés pour permettre de les différencier. Le tableau 4.1 montre certains de ces identificateurs. Pour des raisons de simplification, les noms propres commencent par des lettres minuscules, mais les majuscules sont également traitées, comme le démontre l'identifiant de *femme*.

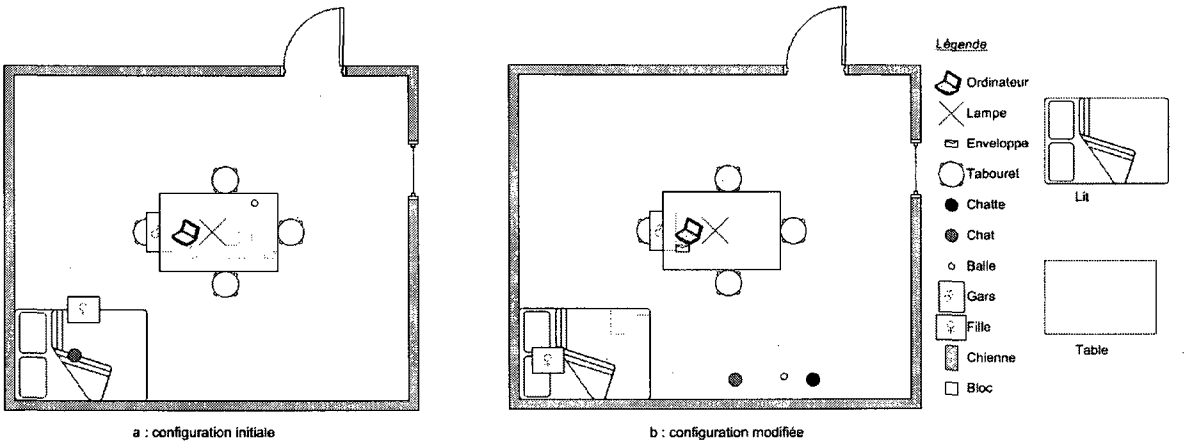

Figure 4.1 – Environnement virtuel de tests

Identificateur	Entité	Identificateur	Entité
moka	chat	nala	chatte
bijoue	chienne	mia	ordinateur
simon-pierre	homme	Marie-Andrée	femme

TABLEAU 4.1 – Table d'association des identificateurs

Certains tests unitaires sont triviaux et ne sont pas détaillés dans ce chapitre, car ils ont peu d'intérêt. Ils sont indiqués pour confirmer qu'ils ont été effectués et qu'ils se sont déroulés normalement. De plus, même si la plupart des modules ne décèlent pas toutes les nuances de la langue française, la fonctionnalité de l'architecture et des modules est quand même démontrée et le fait de compléter ces modules, par exemple en augmentant le dictionnaire, permettrait uniquement une plus grande compréhension de la langue.

Dans les sections suivantes, les tests par modules sont détaillés et les tests d'intégration de l'architecture complète sont présentés à la fin de ce chapitre à la section 4.5.

4.1 Interpréteur de sens

L'*interpréteur de sens* a deux fonctions majeures. La première est de recevoir la phrase reçue par un analyseur syntaxique, tel SATO, et de prendre chaque mot venant de cet analyseur pour en découvrir le sens. La seconde est de générer une phrase ou de poser une action en réponse à la phrase reçue. Étant donnée la complexité de ce module, les tests ont dû être effectués dans un ordre qui permettait d'assurer que le problème est issu du module en cours de test et non pas d'ailleurs. Ces tests nécessitaient pour la plupart la présence d'un *bloc linguistique* fonctionnel. De ce fait, certains tests effectués dans ce module clé

de l'architecture sont des tests qui nécessitent des modules supplémentaires, même s'ils n'étaient pas complétés. Les tests d'intégration de l'architecture seront présentés à la section 4.5, mais ici sont présentés les tests qui devaient être effectués avant l'intégration complète et définitive du module *interpréteur de sens*.

Il faut premièrement vérifier la capacité du module à accéder à la grammaire pour reconstituer une phrase intelligible en langage humain. L'objectif étant de passer de la forme normalisée *texte(présent,chat,être,table,localisation)*, qui est le format utilisé par la machine pour la compréhension de la phrase, à la forme «*Le chat est sous la table*», la forme utilisée par les humains. Lors des premiers tests, le *bloc linguistique* était minimal pour les phrases testées. Toutefois, puisque l'*interpréteur de sens* était constamment utilisé par la suite pour recevoir les réponses et tester le fonctionnement des autres modules, au fur et à mesure que le *bloc linguistique* se complétait, des tests régressifs ont été fait. Donc, des tests en continu furent appliqués à cette fonction et la génération de texte formait une phrase adéquate dans un français correct.

Deuxièmement, la détection du type de phrase a été testée, telles la localisation, la salutation, la présentation, l'identification et la connaissance. En effet, il est possible de faire la distinction entre «*Où est le chat ?*», «*Je me nomme Myriam.*», «*Qu'est-ce qu'un chat ?*» et «*Nomme-moi un animal*». Pour ces tests, le dictionnaire devait également être en service pour obtenir la définition des mots. Ainsi, le module devait identifier la première phrase comme une *interrogation sur un lieu*, la seconde comme une *présentation*, la troisième comme une *interrogation sur une définition* et la quatrième comme étant une *interrogation sur une connaissance*. Plusieurs phrases des différents types, formatées pour la machine sous la forme *q(Marqueur,Verbe,Complément)*, où le marqueur peut être soit un sujet, soit un marqueur interrogatif, ont été utilisées pour les tests. Le résultat de cette requête était le type du message sans se rendre à la réponse finale puisque d'autres fonctions étaient nécessaires pour ressortir la compréhension du message. Suite à chacun de ces tests, le résultat était bel et bien celui attendu. Le bon type était donné à chaque essai, ce qui a permis par la suite d'effectuer certains traitements différents pour certains types de phrase. À ce moment des tests, aucune réponse n'était attendue par le module.

Voici quelques exemples qui démontre ce qui se passe à ce niveau des tests. La fonction *q* des lignes de tests signifie *query*, requête en anglais.

```
TEST: q(où,être,chat).
RÉSULTAT : localisation.
TEST: q(que,être,chat).
RÉSULTAT : définition.
TEST: q(je,'se nommer',myriam).
RÉSULTAT : présentation.
TEST: q(tu,nommer,animal).
RÉSULTAT : connaissance.
```

4.2 Bloc linguistique

Le *bloc linguistique* intègre les résultats de tests obtenus par la *grammaire*, le *conjugueur*, le *dictionnaire* et la *classification des connaissances*, ainsi que les modules spécifiques à la langue. Il vérifie aussi la compréhension de la langue.

Premièrement, il fallait vérifier qu'un mot accordé pouvait être retrouvé dans le dictionnaire. Le test visait donc à faire passer un mot dans la grammaire pour être neutralisé et ensuite être retrouvé dans le dictionnaire. Deuxièmement, le même test a été fait au niveau des verbes. Puis, finalement, la recherche d'une définition hiérarchique à partir d'un nom accordé au féminin pluriel, à savoir que les *chattes* sont des *mammifères*, a été testée. Ce dernier test oblige le passage dans trois des quatre modules du *bloc linguistique*. Ces tests démontrent que le *bloc linguistique* est fonctionnel.

Voici quelques exemples représentatifs des tests du *bloc linguistique*. La fonction *dicoform* demande à la grammaire de neutraliser le mot pour retrouver l'unité lexicale (forme du dictionnaire). Une fois toute l'architecture en place, la fonction *grpcompl* est utilisée par la grammaire afin de construire le groupe complément de la phrase. Cependant, dans le test présenté, il ne sert qu'à faire le lien vers la classification des connaissances afin d'obtenir le terme générique *Generalisation*.

Le premier test présenté neutralise le mot *chiennes* pour retrouver l'unité lexicale *chien* qui est présent dans le dictionnaire. Le second test prend le verbe *aime* et retrouve sa forme infinitive *aimer*, ainsi que son temps de conjugaison *présent*. Le prédicat *accord* sert autant pour retrouver la forme infinitive que pour accorder le verbe. Le dernier test présenté recherche une généralisation pour le mot *chattes*. Celui-ci donne, comme terme générique, le mot *mammifère* qui correspond au niveau commun de langage recherché.

```
test      : dicoform(chiennes,UniteLexicale).
résultat  : UniteLexicale = chien.
test      : accord(_,aime,Infinitif,Temps).
résultat  : Infinitif = aimer, Temps = pres.
test      : grpcompl(chattes,_,Generalisation,classification,_).
résultat  : Generalisation = mammifère
```

À noter que, suite à l'intégration de tous les sous-modules du *bloc linguistique*, la réponse obtenue au dernier test est le suivant :

```
test      : grpcompl(chattes,_,Generalisation,classification,_).
résultat  : Generalisation = des mammifères
```

Ce dernier test est effectué directement sur le modèle de construction du complément de la phrase et non directement sur l'arbre de classification puisqu'il doit être appelé depuis la grammaire pour démontrer la fonctionnalité. En effet, lors de l'intégration finale, la phrase sera générée en prenant comme sujet *les chattes* et en recevant la généralisation *mammifère*. Le module construira alors la phrase *les chattes sont des mammifères*, puisque le terme *mammifère* doit être accordé en genre et en nombre avec son sujet *chattes* avec l'utilisation du verbe d'état *être*.

4.2.1 Grammaire

Un premier type de test fut sur la formation du féminin des noms. Un test typique pour les noms *ami, colonel, chameau, chien, berger, époux, bourgeois* et *veuf* qui sont régis par des règles ainsi que *garçon, homme, vieux* et *chat* qui sont des exceptions. La sortie attendue étant bien sûr le véritable féminin de ces noms. La sortie obtenue fut bien celle attendue, par exemple :

```
test      : feminin(chien,F).
résultat  : F = chienne.
test      : feminin(homme,F).
résultat  : F = femme.
```

Un deuxième type de test est d'obtenir le masculin en appliquant ces mêmes règles, mais dans l'autre sens et de s'assurer que le masculin obtenu n'est pas le résultat d'une règle de chevauchement. La recherche du masculin à partir des règles du féminin s'est avérée efficace puisque le masculin vérifie en même temps la présence du mot dans le dictionnaire. Toutefois, si un mot n'est pas dans le dictionnaire, le module donne comme résultat que

le masculin de ce mot n'existe pas, au même titre qu'un mot strictement féminin comme *lampe*. Des tests similaires pour le masculin et le féminin ont été effectués pour les adjectifs avec résultats équivalents. Voici deux exemples de test qui retrouvent la forme masculine du mot :

```
test      : masculin(chatte,M).
résultat  : M = chat.
test      : masculin(lampe,M).
résultat  : mot strictement féminin.
```

Un troisième type de tests est la formation du pluriel et le retour à la forme du singulier. Des tests semblables à la formation du féminin ont été appliqués pour le pluriel et ont été réussis. Cependant, pour reformer le singulier à partir de la forme plurielle, des ajustements ont été faits. En effet, il est possible que le mot au pluriel soit au féminin, et donc, par le fait même, absent du dictionnaire de base. Donc, pour assurer que le singulier obtenu est la bonne forme du pluriel initial, la recherche du masculin du mot est appliquée et c'est celui-ci qui détermine si le mot est présent dans le dictionnaire. Avec cette vérification, les résultats de conversion entre pluriel et singulier fonctionnent correctement. Voici quatre exemples de ces tests pour la formation du pluriel et le retour au singulier :

```
test      : pluriel(singe,P).
résultat  : P = singes.
test      : pluriel(naval,P).
résultat  : P = navals.
test      : singulier(chevaux,S).
résultat  : S = cheval.
test      : singulier(chattes,S).
résultat  : S = chatte.
```

Un quatrième type de test est effectué pour la formation des phrases. Le *conjugueur* devait être minimalement fonctionnel pour ces tests puisqu'il est nécessaire que le verbe soit conjugué pour vérifier que la phrase est construite correctement. Donc, les verbes utilisés pour les tests devaient être présents dans le *conjugueur*.

En donnant un temps, un sujet, un verbe et un complément, le *bloc linguistique* devait être en mesure de former une phrase en français correct. Lorsque l'entrée est *texte(présent, je, être, garçon)*, ce qui, dans le langage de la machine, signifie «fais-moi une phrase au *présent* ayant pour sujet *je*, utilisant le verbe *être* et comme complément *garçon*», la sortie

doit être *je suis un garçon.* Au départ, les articles et déterminants étaient omis, mais au fur et à mesure que le *dictionnaire* et la *grammaire* étaient remplis, ce test a toujours donné les résultats attendus. Par la suite, des tests plus complexes ont été implémentés en accord avec ce que l'*interpréteur de sens* demandait selon le type de phrase demandée et le type de complément désiré. Par exemple, l'*interpréteur de sens* pouvait demander à la *grammaire* de former une phrase indiquant une localisation ou une identification, ce qui change la forme de la phrase à construire. Ainsi, *texte(imparfait, chat, être, table, localisation)* et *texte(présent, spot, être, chien, identification)* modifiait l'article devant le nom pour donner «***le** chat était sous **la** table*» et «*spot est **un** chien*». Pour chaque cas traité, la réponse donnée avait la forme désirée. Il est toutefois à noter que toutes les formes possibles de la langue française n'ont pas été implémentées et que, par exemple, une question du type «*Comment me rendre à l'hôpital ?*» ferait en sorte que l'*interpréteur de sens* enverrait une demande de phrase de type *indication* ou *moyen* que la grammaire ne saurait pas traiter pour le moment. Toutefois, pour les types connus par *grammaire*, celle-ci construit une phrase qui fait du sens.

Un cinquième type de test est fait pour l'élision des mots. Les entrées du type *le oiseau, le beau avion, la plante* et *le joli avion* furent utilisés. Le type de sorties attendues était *l'oiseau, le bel avion* et les deux derniers devaient rester inchangés. Les élisions se sont faites correctement. Toutefois, pour ce qui est de la non-différenciation entre les «H» muets et aspirés, le correctif suggéré serait de mettre dans les mots du dictionnaire leur prononciation sous forme phonétique et de baser le traitement sous cette forme lorsqu'un «H» est détecté pour déterminer la nécessité d'élider ou non un mot. Pour le moment, tous les mots commençants par un «H» et dont la seconde lettre est une voyelle génèrent une élision.

Un sixième type de test est fait pour la contraction. Des entrées du type *de le, de la, de l', à le, à la* et *à l'* ont été soumis au test et les sorties reçues étaient, dans l'ordre, *du, de la, de l', au, à la* et *à l'*, ce qui était prévu.

Un septième type de test a été effectué pour la transformation. Il s'agit uniquement de faire en sorte que si la question était sous la forme *je* ou *tu*, alors la réponse est l'inverse. Le *je* devient un *tu* et le *tu* devient un *je.* Lors des tests, chaque fois que se présente l'un ou l'autre de ces sujets, la réponse est bien transposée. Il en va de même pour le *nous* et le *vous.* Tous les autres sujets, *il, elle, ils, elles, on* et les noms, restent inchangés. Lorsqu'une transposition est faite, par exemple un *je* qui devient un *tu*, une formule mathématique a été mise en oeuvre pour convertir la personne du sujet. Cette formule est la suivante :

$((Personne\%3) \times (^{-}1)) + 3$. Le % est l'opérateur modulo qui donne le reste de la division entière. Cette formule mathématique est donc pratique pour cette conversion plutôt que de faire un traitement cas par cas de chaque sujet potentiel de la langue française. Le tableau 4.2 montre quelques exemples de transformation de sujets.

Sujet initial	**Valeur numérique (I)**	$^{-}(I\%3)+3=S$	**Sujet de sortie**
je	1	$^{-(1\%3)+3=S}$ $^{-}1+3=2$	tu
tu	2	$^{-(2\%3)+3=S}$ $^{-}2+3=1$	je
nous	1	$^{-(1\%3)+3=S}$ $^{-}1+3=2$	vous
vous	2	$^{-(2\%3)+3=S}$ $^{-}2+3=1$	nous
il	3	$^{-(3\%3)+3=S}$ $0+3=3$	il
elles	3	$^{-(3\%3)+3=S}$ $0+3=3$	elles
Spot	3	$^{-(3\%3)+3=S}$ $0+3=3$	Spot

TABLEAU 4.2 – Exemple de transformation de sujets.

4.2.2 Conjugueur

Le *conjugueur* est bâti sur des règles de conjugaison et d'exceptions, il faut donc tester les règles ainsi que la gestion des exceptions. Le premier type de tests consistait à prendre un verbe purement irrégulier, en occurrence, l'auxiliaire *être*, et de tester les différents temps et les différentes personnes. La sortie attendue était que le verbe soit accordé correctement et que les règles de conjugaison ne soient pas prises en compte. En effet, *être* se trouvait entièrement dans la table de conjugaison, c'est-à-dire que chaque accord de chaque personne de chaque temps retenu était entré manuellement. Il ne fallait pas retrouvé comme conjugaison une forme du genre *nous étons* au présent de l'indicatif. Le second type de tests fut celui d'un verbe purement régulier du premier groupe, soit *nommer*, et d'obtenir la bonne conjugaison en utilisant uniquement les règles de conjugaison. Le troisième type de tests était de prendre un verbe qui respectait partiellement les règles de conjugaisons et de vérifier qu'il prend en compte les exceptions du verbe, mais qu'il utilise les règles le reste du temps. Dans ce dernier cas, le verbe *aller* fut utilisé. Dans les trois cas, la conjugaison s'est effectuée sans problèmes dans tous les temps intégrés dans le *conjugueur*. Voici quelques exemples illustrant ces tests.

```
test      : verbe(aimer,pres,3,2,Conjugaison).
résultat  : Conjugaison = aiment.
test      : verbe(aimer,futur,3,2,Conjugaison).
résultat  : Conjugaison = aimeront.
test      : verbe(avoir,pres,2,1,Conjugaison).
résultat  : Conjugaison = as.
test      : verbe(être,imparf,2,2,Conjugaison).
résultat  : Conjugaison = étiez.
test      : verbe(trouver,passcomp,3,1,Conjugaison).
résultat  : Conjugaison = 'a trouvé'
test      : verbe(Infinitif,_,_,_,mange).
résultat  : Infinitif = manger
```

La syntaxe de ces tests est la suivante : verbe(Infinitif, Temps, Personne, Nombre, Conjugaison). Comme vu à la section 4.2, il est également possible de demander l'infinitif. En fait, le prédicat *accord* utilisé dans le *bloc linguistique* appelle le prédicat *verbe* du *conjugueur*. Selon le fait de laisser l'*Infinitif* ou la *Conjugaison* variable, il est possible de récupérer l'un ou l'autre. Infinitif est donc l'entrée en cas d'accord et la sortie en cas de désaccord. Le *Temps* est le temps de conjugaison, soit présent, futur, imparfait, participe passé, passé composé, participe présent ou conditionnel présent. Des acronymes ont été utilisés pour simplifier le code. La *Personne* prend la valeur 1, 2 ou 3 et le *Nombre* la valeur 1 pour le singulier et 2 pour le pluriel. La *Conjugaison* est la valeur de retour si le *conjugueur* effectue un accord ou la valeur d'entrée en cas de désaccord.

4.2.3 Dictionnaire

Le dictionnaire contient trois types de définitions : directes, formalisables et hiérarchisées. Il faut vérifier que chaque type renvoie la bonne définition.

Les définitions directes sont les plus simples. Si un mot est recherché, sa définition fonctionnelle est donnée. Puisque ce sont des faits, les résultats des tests sont facilement vérifiés. Par exemple, le mot *où* est un mot intérrogeant sur une localisation, le mot *le* est un déterminant indiquant que l'objet est connu et le mot *je* est un pronom désignant l'interlocuteur.

Pour les définitions hiérarchiques, puisqu'elles sont dans le module de *classification des connaissances*, ces tests sont discutés à la section 4.2.4. La définition d'une donnée hiérarchique correspond à sa valeur générique dans l'arbre de classification.

Les définitions formalisables devaient être testées plus en profondeur, car elles sont écrites sous forme de règles, par exemple, les définitions servant à la localisation relative des objets entre eux. Les exemples utilisés correspondent à l'environnement tel qu'affiché sur la figure 4.1a. Seuls les termes *au dessus*, *en dessous*, *sur* et *sous* sont présentés, mais le même type de test a été effectué pour *devant*, *derrière*, *à gauche*, *à droite* qui sont aussi des définitions formalisables présentes dans le dictionnaire. Bien sûr, pour réaliser ces tests, le module *environnement virtuel* doit être en fonction.

En se référant à la figure 4.1a, le tableau 4.3 montre l'emplacement relatif de quelques objets entre eux.

Objet1 ⇓ /*Objet2* ⇒	lit	table	bloc	enveloppe	ordinateur
lit	–	devant	devant	devant	devant
table	derrière	–	au dessus	sur	sous
bloc	derrière	en dessous	–	à gauche	à droite
enveloppe	derrière	sous	à droite	–	à droite
ordinateur	derrière	sur	à gauche	à gauche	–

TABLEAU 4.3 – Position relative de quelques objets de l'environnement

La fonction *pos_rel* qui est utilisée, fait le lien entre le *dictionnaire* et l'*environnement virtuel* afin de trouver la position relative d'un objet par rapport à un autre. La syntaxe est pos_rel(Objet1,POS,Objet2,Moment) est l'équivalent français de demander «où se situe l'*Objet1* par rapport à l'*Objet2* en ce *Moment*» et la position *POS* est donnée en réponse. *Moment* peut prendre uniquement la valeur *maintenant* ou *avant* dans la conception actuel du module *connaissances dynamiques*. Voici des exemples pour ce test.

```
Test : pos_rel(ordinateur,POS,table,maintenant).
Résultat : POS = sur.
Test : pos_rel(enveloppe,POS,table,maintenant).
Résultat : POS = sous.
Test : pos_rel(lampe,POS,table,maintenant).
Résultat : POS = au dessus.
Test : pos_rel(bloc,POS,table,maintenant).
Résultat : POS = en dessous.
```

Comme il est possible de constater, les résultats obtenus correspondent aux résultats attendus du tableau 4.3.

4.2.4 Classification des connaissances

Le module de *classification des connaissances* interagit avec le dictionnaire pour lui donner les définitions hiérarchisées. Il est principalement composé de faits et de quelques règles pour les classer entre eux par antécédents et successeurs hiérarchiques. Ainsi, ce module retourne le terme générique ou spécifique d'un mot à un niveau commun du langage. Le tableau 4.4 est utilisé pour les tests de ce module.

Règne	Branche	Ordre	Famille	Genre
Animal	Vertébré	Mammifère	Canidé	Chien
Animal	Vertébré	Mammifère	Canidé	Loup
Animal	Vertébré	Mammifère	Félinidé	Chat
Animal	Vertébré	Mammifère	Félinidé	Puma
Animal	Vertébré	Mammifère	Primate	Singe
Animal	Vertébré	Mammifère	Primate	Humain
Animal	Vertébré	Mammifère	Cétacé	Baleine
Animal	Vertébré	Mammifère	Cétacé	Dauphin
Animal	Vertébré	Mammifère	Cétacé	Épaulard
Animal	Vertébré	Oiseau	Passereau	Hirondelle
Animal	Vertébré	Oiseau	Rapace	Aigle
Animal	Vertébré	Oiseau	Rapace	Faucon
Animal	Invertébré	Insecte		

TABLEAU 4.4 – Table de classification

Le premier type de test vérifie que le module donne bien comme résultat les antécédents hiérarchiques (termes génériques) de chaque objet présent du module. Par exemple, lorsqu'il est demandé de trouver le générique de *mammifère*, *animal* doit être donné, pour *chien*, *canidé* doit être donné, et ainsi de suite. Étant donné que la classification est faite dans ce sens, le générique immédiat d'un mot est bel et bien ce qui a été obtenu. Voici quelques exemples de résultats. Le prédicat *classification* demande l'élément supérieur dans la classification correspondant au terme générique du mot. Le second paramètre est le nombre de niveau qui doit être remonté pour atteindre le niveau commun du langage. Pour ce type de test, la valeur 1 est attribuée.

```
test      : classification(chien,1,Generique).
résultat  : Generique = canidé
test      : classification(chat,1,Generique).
résultat  : Generique = félinidé
```

Le second type de test consiste à valider que le niveau commun du langage est renvoyé. Il faut d'abord tester qu'il est possible d'afficher le terme générique de deux ou trois niveaux

supérieurs, puis ensuite, laisser le module déterminer le niveau pour renvoyer le terme qui correspond au niveau commun du langage choisit dans l'implémentation. Voici quelques exemples de tests. Lorsque le deuxième paramètre est omis, le module détermine lui-même la valeur sur la base du niveau commun déterminé dans l'arbre de classification.

```
test      : classification(chien,2,Generique).
résultat  : Generique = mammifère
test      : classification(chat,3,Generique).
résultat  : Generique = vertébré
test      : classification(singe,4,Generique).
résultat  : Generique = animal
test      : classification(primate,3,Generique).
résultat  : Generique = animal
test      : classification(chat,_,Generique).
résultat  : Generique = mammifère
test      : classification(canidé,_,Generique).
résultat  : Generique = mammifère
test      : classification(mammifère,_,Generique).
résultat  : Generique = animal
```

Les résultats sont bien ceux attendus sur la base du tableau 4.4. Il est possible de remarquer que le terme générique renvoyé par *chat* et *canidé* se trouve au même niveau commun du langage, soit *mammifère*, même si le niveau du terme générique a été omis.

Le troisième type de test est de valider le terme spécifique immédiat. Le module fait une liste de tous les termes spécifiques et il fallait vérifier qu'aucun descendant connu de la machine ne manquait et qu'aucun termes inappropriés ne s'y trouvait. Cette liste est ensuite réutilisée comme entrée pour la fonction aléatoire pondérée.

```
test      : prototype(mammifère,_,Liste).
résultat  : Liste = [canidé, cétacé, félin, primate]
test      : prototype(canidé,_,Liste).
résultat  : Liste = [chien, loup]
```

À noter que dans ce type de tests, le résultat n'est pas représentatif de la pondération. Lors de l'ajout de la pondération, la liste résultante reflète la pondération d'occurrence de chaque terme en indiquant combien de fois on retrouve chaque éléments de la liste.

```
test      : prototype(mammifère,_,Liste).
résultat : Liste = [[8, canidé], [3, cétacé], [10, félin], [5, primate]]
test      : prototype(canidé,_,Liste).
résultat : Liste = [[7, chien], [1, loup]]
```

Le quatrième type de test vérifie le niveau commun du langage en sens descendant. Il fallait tester le fait de se rendre au bon niveau de terme spécifique qui correspond au niveau commun du langage. Cette fois, ce n'est pas une liste des possibilités qui est attendue, mais un des résultats de la liste. À ce stade, il s'agit uniquement d'une fonction aléatoire pondérée qui détermine quel élément est choisi. Les résultats présentés correspondent à une liste de chaque résultat avec le nombre de fois que ceux-ci ont été obtenus. Le prédicat *testCDwn* a été implémenté afin de tester la classification en descendant dans l'arbre afin de trouver les termes spécifiques de chaque noeud. Le premier paramètre est le nombre de tests consécutifs que la machine doit effectuer, dans l'exemple présenté, 20 tests consécutifs sont effectués. Les résultats présentent donc un résumé des résultats de ces 20 tests. La somme des valeurs numériques de la liste est toujours égale au premier paramètre de *testCDwn*.

```
test : testCDwn(20,mammifère,Liste). %Test Classification vers le bas.
résultat : Liste = [[9, chat], [6, chien], [2, dauphin], [3, humain]]
```

Le cinquième type de test vérifie la pondération pour donner un terme spécifique déterminé par le niveau de langage recherché. Pour valider que les éléments ayant la plus grande occurrence reviennent plus souvent, un test de vérification des réponses renvoyées sur 1000 demandes identiques a été effectué. Celui-ci affichait les éléments regroupés et comptabilisés, puis écrits en ordre croissant. Le résultat attendu était que les entités avec la plus grande occurrence se retrouvent en bas de la liste avec une différence notable avec les entités de moindre occurrence. Le tableau 4.5 montre les proportions utilisées par la fonction aléatoire pondérée et le tableau 4.6 montre les résultats obtenus sur trois groupes de tests consécutifs pour la demande d'un terme spécifique de *mammifère*.

Il est possible de remarquer que *chat* et *chien* sont les mammifères qui reviennent le plus souvent, ce qui démontre que la fonction aléatoire pondérée fonctionne et que le module en tient compte pour retourner le terme spécifique. Plutôt que de compter les occurrences où un animal différent est rencontré dans l'environnement de la machine, il serait possible de lui donner un dictionnaire de fréquences, tel qu'il existe en linguistique. Cependant, le fait de compter les occurrences se rapproche davantage du comportement humain, car

Terme	Occurrence
canidé	8
félinidé	10
primate	5
cétacé	3
chien	7
loup	1
chat	9
puma	1
humain	4
singe	1
dauphin	2
épaulard	1

TABLEAU 4.5 – Table d'occurrences pour les mammifères connus par l'architecture

mammifère	obtenu test1	mammifère	obtenu test2	mammifère	obtenu test3
puma	29	épaulard	30	loup	29
singe	39	loup	40	puma	31
épaulard	39	singe	43	épaulard	38
loup	46	puma	44	singe	41
dauphin	81	dauphin	74	dauphin	88
humain	136	humain	167	humain	161
chien	286	chien	274	chien	256
chat	344	chat	328	chat	356
	1000		1000		1000

TABLEAU 4.6 – Résultats de trois tests de 1000 descendants de *mammifères*

un animal qui n'a jamais été rencontré par la machine ne sera jamais nommé, alors qu'un animal rencontré régulièrement sera plus souvent considéré par la machine.

4.3 Environnement virtuel

Le module *environnement virtuel* contient les informations de la machine sur son environnement réel. Dans une application réelle, les données inscrites à l'intérieur du module seraient obtenues par un système de capteurs. Pour ce projet, les données sont entrées manuellement. Ce module connait les coordonnées ainsi que l'espace occupé par les différents objets, et il différencie les différents objets. Il est appelé lorsque la machine doit connaître l'emplacement d'un objet, par exemple, lors de la localisation. Étant donné que ce module est constitué presque exclusivement de faits, le seul test intéressant est celui de la conversion entre l'identifiant et l'objet lui-même afin d'obtenir ses dimensions et son

emplacement. Le résultat de ce test est que la machine sait aussi bien que *Moka* ou *le chat* font référence au même type d'entité et que *Nala* ou *la chatte*, qui est vu également comme étant un *chat* par la machine, font référence à une entité différente de la première. Le module fait cependant la différence entre ces deux entités par l'identifiant et fait la distinction entre ces entités afin de retourner les bonnes coordonnées. Voici deux exemples qui démontrent que les requêtes pour la récupération se fait autant avec l'identifiant que le référent. checkID permet de faire ressortir le référent (Ref) et l'identifiant (ID) et de ressortir les coordonnées X, Y et Z ainsi que les dimension DX, DY et DZ. Puisque les requêtes équivalentes donnent la même réponse, la réponse n'est écrite qu'une seule fois par binôme.

```
requêtes :
checkID(moka,Ref,ID),localisation(ID,X,Y,Z,_),entitee(Ref,DX,DY,DZ).
ou
checkID(chat,Ref,ID),localisation(ID,X,Y,Z,_),entitee(Ref,DX,DY,DZ).
réponse :
Ref = chat, ID = moka, X = 2, Y = 1.5, Z = 2.5,
DX = 1.0, DY = 1.0, DZ = 1.0

requêtes :
checkID(nala,Ref,ID),localisation(ID,X,Y,Z,_),entitee(Ref,DX,DY,DZ).
ou
checkID(chatte,Ref,ID),localisation(ID,X,Y,Z,_),entitee(Ref,DX,DY,DZ).
réponse :
Ref = chatte, ID = nala, X = 6.5, Y = 5.5,Z = 0,
DX = 1.0, DY = 1.0, DZ = 1.0
```

Présentement, l'absence des capteurs oblige de documenter chaque objet de l'environnement, mais il est possible de penser qu'avec l'ajout de capteurs, certaines informations sur les objets environnants soit traitées en temps réel par les capteurs sans nécessairement avoir à retenir cette information en mémoire.

4.4 Connaissances dynamiques

Le module *connaissances dynamiques* contient les états passés et l'état actuel de l'*environnement virtuel*. Il correspond à la mémoire des événements de la machine. Il faut donc vérifier que

ces informations sont bien enregistrées et retrouvées dans le bon ordre lorsque des modifications sont faites sur l'état de l'*environnement virtuel.*

Un premier type de test important est d'assurer que les éléments qui s'y trouvent sont effectivement dynamiques. Ces tests modifient l'état d'un élément de l'environnement et vérifient qu'il a effectivement été modifié. L'affichage de l'état précédent et de l'état courant à l'écran servait de balise d'évaluation. Ensuite, la recherche de l'état actuel devait donner le nouvel état et non l'ancien. Suite aux tests, ces critères étaient respectés et la sortie était bien celle attendue.

Une deuxième série de tests importants pour ce module était l'accès à un fichier en écriture et en lecture pour conserver et récupérer les archives des états de l'*environnement virtuel* qui lui sert de mémoire à long terme (voir figure 3.1). Un test effectué est l'inscription de toutes les entrées dynamiques dans le fichier. L'ouverture du fichier pour l'évaluation visuelle du contenu sert d'évaluation. Un autre test est celui de la lecture du fichier. Il faut pouvoir charger les connaissances dynamiques dans la mémoire de l'architecture, qui correspond à la mémoire à court terme (voir figure 3.1) afin qu'elles puissent être consultées.

Le dernier test effectué pour l'accès au fichier d'archives est celui de la vérification des doublons. À la suite de plusieurs lectures et écritures consécutives de la base de connaissances, le fichier ne doit pas *gonfler* en ajoutant des entrées en double. En effaçant le contenu de la base de connaissances en ce qui a trait aux entrées dynamiques avant la lecture du fichier et en écrasant le fichier d'archives lors de l'écriture, ce problème n'est pas survenu. Effectivement, à chaque relecture et réécriture, ni le fichier, ni la base de connaissances n'ont d'entrées en double.

Avec les connaissances dynamiques, la compréhension du concept de présent et de passé par la machine a aussi été testée. À la suite d'un changement dans l'*environnement virtuel* correspondant à un déplacement du chat du lit vers le sol à droite du lit, deux questions ont été posées à la machine : «Où est le chat par rapport au lit ?» et «Où était le chat par rapport au lit ?». Les réponses attendues étaient respectivement «Le chat est à droite du lit» puis «Le chat était sur le lit». Ce fut effectivement les réponses obtenues. Ensuite, la balle fut déplacée de la table vers la droite du chat. Il fut alors demandé à la machine «Où était la balle par rapport au chat ?» puis «Où est la balle par rapport au chat ?». Les réponses attendues étaient «au dessus» puis «à droite». Cependant, les réponses obtenues étaient les deux fois «à droite». L'explication de cette apparente erreur vient du fait que les déplacements ne sont pas situés dans le temps et le module considère que tous les

déplacements se font au même instant. De ce fait, même si la balle a été déplacée après le chat, puisque tous les évènements du passé sont considérés au même moment, il est considéré que la balle et le chat se sont déplacés en même temps. C'est un choix de design pour le moment, car l'ajout en prolog d'une étiquette de temps devant être gérée manuellement pour savoir si l'événement a eu lieu avant un autre n'était pas nécessaire pour démontrer la validité de l'architecture. Ainsi, ce n'est pas une erreur du système. Pour remédier à cette situation, il s'agirait d'indiquer le moment des déplacements et de faire un traitement sur le temps relatif de chacun. La performance actuelle de cette architecture est suffisante, mais elle pourrait être améliorée en remplaçant le fichier servant de MLT par une base de données contenant l'historique des déplacements ainsi que le moment où ceux-ci sont survenus.

4.5 Architecture globale - intégration

Les tests d'intégration des modules de l'architecture sont sous une forme différente : ils sont séparés par ensembles fonctionnels. Chaque série de tests vérifie une fonctionnalité et démontre que les réponses sont bien distinctes lorsqu'elles doivent l'être et identiques lorsque le contexte s'y attend. Il est important de rappeler que, pour une question de simplification, les majuscules ne sont pas prises en compte. Le fait que la première lettre d'une phrase soit en minuscule ne constitue pas une erreur de l'architecture, mais un choix de design pour la démonstration de la fonctionnalité de l'architecture.

4.5.1 Localisation

Les tests de localisation permettent à la fois de

- vérifier que l'architecture analyse correctement l'*environnement virtuel*,
- vérifier l'association entre l'identifiant et l'entité correspondante,
- rechercher correctement les informations archivées ou l'état actuel lors de la localisation,
- différencier la forme féminine de la forme masculine,
- localiser par rapport à n'importe quel objet ou même sans objet de référence.

L'état de l'*environnement virtuel* présent et passé est présenté à la figure 4.1 et les identificateurs sont présentés au tableau 4.1, au début de ce chapitre. La fonction q, qui signifie

query (requête en anglais), est la syntaxe utilisée par l'*interpréteur de sens* pour consulter le *bloc linguistique.*

En premier lieu, il est vérifié que *chat* et *moka* sont correctement associés. Pour ce faire, des requêtes interchangeant *moka* et *chat* doivent retourner la même valeur.

```
test1a :- q(où,est,chat,lit).        %où est le chat par rapport au lit
resultat : le chat est à droite du lit.
test1b :- q(où,est,moka,lit).        %où est moka par rapport au lit
resultat : moka est à droite du lit.
test1c :- q(où,était,chat,lit).      %où était le chat par rapport au lit
resultat : le chat était sur le lit.
test1d :- q(où,était,moka,lit).      %où était moka par rapport au lit
resultat : moka était sur le lit.
```

Ces résultats démontrent bien que moka et le chat sont bien associés puisque les réponses correspondent. Ils démontrent également que la gestion du moment présent et passé est faite puisque le chat est passé de «*sur le lit*» à «*à droite du lit*»

Ensuite, il fallait démontrer que, malgré le fait que chat et chatte réfèrent à la même unité lexicale, soit le mot *chat*, ils sont différenciés dans l'environnement.

```
test1e :- q(où,est,chatte,lit). %où est la chatte par rapport au lit
resultat : la chatte est à droite du lit.
test1f :- q(où,était,chatte,lit). %où était la chatte par rapport au lit
resultat : la chatte était derrière le lit.
```

En comparant avec les tests 1a et 1c avec les tests 1e et 1f, le chat et la chatte sont bien différenciés par l'architecture.

Il est également important de vérifier que la requête est réversible, c'est-à-dire que lorsque les paramètres sont inversés, la réponse est logiquement inversée.

```
test1g :- q(où,est,lit,chat). %où est le lit par rapport au chat
résultat : le lit est à gauche du chat.
```

En comparant avec le test 1a qui demande la question inverse, ce test démontre qu'effectivement, la réponse est inversée.

Tous les tests présentés précédemment localisent un objet par rapport au lit. Il est donc important de s'assurer qu'il est possible de localiser par rapport à d'autres objets, qu'ils

soient fixes ou mobiles. Les tests qui suivent localisent le chat par rapport à la femme et la chatte par rapport à la table.

```
test1h :- q(où,est,chat,femme). %où est le chat par rapport à la femme
résultat : le chat est à droite de la femme.
test1i :- q(où,était,chat,femme). %où était le chat ~
résultat : le chat était devant la femme.
test1j :- q(où,est,chatte,table). %où est la chatte ~ la table
résultat : la chatte est devant la table.
test1k :- q(où,était,chatte,table). %où était la chatte ~
résultat : la chatte était en dessous de la table.
```

Ces tests démontrent bien que la localisation par rapport à tout objet de l'environnement est possible.

Finalement, il est intéressant de voir le résultat d'une recherche qui ne spécifie pas par rapport à quel objet l'utilisateur souhaite faire la localisation.

```
test1l :- q(où,est,femme). %où est la femme (sans référence)
résultat : la femme est sur le lit.
```

Le résultat donnée est «*sur le lit*» qui correspond à l'objet fixe le plus près de l'objet à localiser.

Bref, ces 12 tests démontrent que l'architecture réussit à interpréter les localisations dans l'*interpréteur de sens* à l'aide du *bloc linguistique*, à retrouver les unité lexicale du *dictionnaire* tout en les différenciant à l'aide de la *grammaire*, à rechercher dans l'*environnement* les informations nécessaires pour formuler une réponse en accord avec les règles de la *grammaire*.

4.5.2 Interrogation des définitions hiérarchiques

Ces tests servent à vérifier que le niveau commun du langage est respecté lorsqu'un terme générique ou spécifique est demandé et que l'ascension ou la descente dans l'arbre de classification se fait correctement sur plusieurs niveaux. Du même coup, la fonction aléatoire pondérée sur les occurrences est également testée. Le pourcentage d'occurrence est présenté plus spécifiquement aux résultats du test *2e*. Le féminin et le pluriel sont aussi présentés pour démontrer que l'architecture fait abstraction du genre et du nombre pour la recherche du résultat, mais les considèrent dans sa réponse.

Tout d'abord, les tests demandant les termes génériques. Ceux-ci sont plus simples puisque chaque terme n'a qu'un seul terme générique. Il faut uniquement vérifier que le terme renvoyé correspond à un niveau commun du langage.

```
test2a :- q(que,est,chat).        %qu'est-ce qu'un chat
résultat : le chat est un mammifère.
test2b :- q(que,sont,chiennes).   %que sont les chiennes
résultat : les chiennes sont des mammifères.
```

Les résultats de ces deux tests démontrent qu'effectivement le niveau commun est obtenu et non pas un niveau intermédiaire, tel que *canidé* ou *félinidé* qui sont les termes génériques, ou parents, directs dans l'arbre de classification. À noter également que lors de la génération de la réponse, le pluriel du sujet est reflété dans le complément.

Ensuite viennent les tests sur les termes spécifiques. Ceux-ci sont plus complexes puisqu'il faut choisir un membre de la liste des termes spécifiques correspondants à la recherche. Il faut également atteindre le niveau commun, ce qui implique que la fonction aléatoire pondérée doit être utilisée plus d'une fois pour déterminer le terme à choisir. En effet, la décision aléatoire pondérée est appliquée sur chacune des branches de l'arbre de classification pour ne pas développer l'arbre au complet, mais développer uniquement les noeuds qui correspondent à la demande de l'utilisateur. Une partie de l'arbre utilisé pour les tests est présenté au tableau 4.4.

```
test2c :- q(nomme,mammifère).    %nomme-moi un mammifère
résultat : le puma est un mammifère.
test2d :- q(choisit,animal).     %choisit un animal
résultat : le singe est un animal.
```

Tel qu'expliqué à la section 3.4.4, le résultat de ces tests est aléatoire puisque l'architecture doit choisir un enfant dans la liste, puis continuer à partir de celui-ci pour atteindre le niveau commun. Dans ces tests, il y a deux paramètres qui changent. Premièrement, dans le test 2c, la classification part de *mammifère* puis, dans le test 2d, part de *animal*, ceci pour démontrer que le niveau commun est toujours atteint quand même. Deuxièmement, le verbe *nommer* et le verbe *choisir* composent les requêtes 2c et 2d respectivement afin de démontrer que les synonymes peuvent être gérés. Même si le module *synonyme* n'est pas implémenté dans cette démonstration, deux termes présentant une définition similaire, dans un cadre de définitions fonctionnelles simples, sont interprétés de la même manière et donnent le même résultat.

Un dernier test permet de vérifier la proportion dans laquelle les tests 2c et 2d peuvent donner leur réponse. Dans le test qui suit, la fonction *getTest* lance la même requête que le test 2c le nombre de fois indiqué par le premier paramètre. Par la suite, la variable R accumule chaque résultat et fait un tableau puis les classe en ordre croissant.

```
test2e :- getTest(100,animal,R). %test de proportionnalité
.                                %sur 100 spécimens d'animaux
résultat :
 1:pinson      1:singe
 2:faucon      2:hirondelle
 2:loup        5:puma
 7:dauphin     8:aigle
 9:épaulard    13:humain
 21:chien      29:chat
```

Tel qu'indiqué dans le tableau 4.6, il est possible de constater, dans le test 2e, que le chat et le chien sont toujours plus présents que tous les autres membres.

Tout comme la localisation, ces tests font intervenir l'*interpréteur de sens* et le *bloc linguistique*. Cette fois, la *classification des connaissances* est mise à contribution pour valider les connaissances des définitions hiérarchiques. Comme démontré, les phrases données par l'*interpréteur de sens* sont bien accordées et font bien interagir les différents modules de l'architecture.

4.5.3 Identification, transposition et salutation

Ces tests font interagir les relations entre l'identificateur et son identifiant ainsi que la relation entre la 1^{re} et 2^{e} personne du singulier du sujet et finalement, les salutations d'usages et les présentations. La 1^{re} et 2^{e} personne du pluriel ne sont pas directement démontrées ici puisqu'il n'y a pas d'identifiants rattachés pour l'interlocution.

Premièrement, les identifications de base. Ceux-ci démontrent le lien qui existe entre les identificateurs et les entités auxquelles ils sont attachés puis les affichent sous forme de texte.

```
test3a :-
 q(qui,est,moka), q(comment,'se nomme',chatte), q(qui,est,chienne).
 % qui est moka, comment se nomme la chatte, qui est la chienne?
résultat :
```

```
moka est un chat.
la chatte se nomme nala.
la chienne est bijoue.
```

Dans ces tests, tel que montré également lors de la localisation à la section 4.5.1, les identifiants *moka*, *nala* et *bijoue* sont bien rattachés à *chat*, *chatte* et *chienne*. Il est également démontré que la forme pronominale de nommer, *se nommer*, ne donne pas le même résultat que le test 2c qui utilisait également ce verbe sous sa forme normale. La polysémie a été écartée pour le projet, mais un exemple de faisabilité a été mis sur pied pour cette démonstration. Bien que nommer n'est pas fortement polysémique, il présente une différence dans sa définition selon son utilisation. Ainsi, *se nommer* n'est pas interprété de la même manière que *nommer* même si, dans le dictionnaire, il s'agit de la même unité lexicale, à savoir le verbe nommer.

Le prochain test est la transposition du sujet. Pour ce faire, l'identification a également été utilisée. Le référent des interlocuteurs *je* et *tu* et la transposition qui existe pour passer de l'un à l'autre sont présentés dans ce test. L'interlocuteur est *Simon-Pierre* et la machine qui lui répond se nomme *MIA*.

```
test3b :- q(qui,suis,je), q(qui,es,tu). %qui suis-je, qui es-tu?
résultat :
tu es simon-pierre.
je suis mia.
```

Il est possible de constater que lorsque le *je* est utilisé, la réponse utilise le *tu* et inversement. Ce test montre également que le conjugueur exécute correctement sa fonction puisqu'il ne répond pas «*tu suis simon-pierre*» ni «*je es mia*». Ce test montre donc que l'*interpréteur de sens* ne réutilise pas bêtement les mots utilisés par l'utilisateur pour reconstruire sa réponse. D'ailleurs, d'autre tests non représentatifs ont démontré que l'utilisation de la forme infinitive d'un verbe dans une requête ne change pas la forme de la réponse, mais cause parfois une réponse où le temps de conjugaison peut être erroné même s'il est bien conjugué.

Ensuite, les tests de salutation sont triviaux puisque, à l'instar de l'être humain, celle-ci est une reconnaissance de mot avec réponse directe. Celle-ci est comparable à un automatisme linguistique. Cependant, la réponse générée n'est pas préconçue. Elle répond selon un modèle et la machine choisit de manière aléatoire la façon dont elle répond à la salutation.

La machine choisit parmi les salutations qu'elle connaît puis ajoute sur une base *volontaire* le nom de son interlocuteur, si elle le connaît. Voici deux exemple de cas de salutation.

```
test3c :- q(bonjour), q(bonjour).
résultat :
allo, simon-pierre.
salut.
```

Ce test est basé sur une reconnaissance simple de fonctionnalité. Les mots ayant comme définition la fonction de salutation engendre automatiquement une salutation par la machine. Ce contexte de réponse automatique reflète le fait que l'être humain répond parfois par réflexe à certain mots. Toutefois, le processus est le même que pour les autres requêtes. Le mot doit passer par le dictionnaire pour avoir la définition puis par la grammaire pour générer une phrase, bien que dans ce cas précis, il ne s'agisse que d'une phrase sans verbe, aussi connu sous le nom de pseudo-phrase.

Finalement, le test de présentation. Celle-ci engendre un changement d'interlocuteur, une salutation et une présentation de la part de la machine. Il intègre donc le test 3c en plus de faire intervenir une action directe sur les connaissances de la machine. Puisque la présentation entre en compte, la partie aléatoire du test précédent est reflété dans les résultats de ce test.

```
test3d :- q(je,'me nomme',myriam).
résultat : bonjour, myriam, moi, je me nomme mia.
test3e :- q(je,'me nomme','simon-pierre').
résultat : salut, moi, je me nomme mia.
```

Dans le second résultat, le changement d'interlocuteur n'est pas visible. Par contre, la requête «*q(qui,suis,je)*», comme présenté au test 3b, permet de s'assurer que le changement a bien eu lieu.

Tous les tests effectués correspondent aux résultats attendus. Ceux-ci faisaient interagir différents modules du *bloc linguistique*, l'*interpréteur de sens*, et les *connaissances dynamiques*.

En résumé, les tests d'intégration de l'architecture de compréhension du message et de génération de réponse sont concluants. Les phrases ne sont pas préconstruites, mais générées à partir des règles de la grammaire. Les différentes informations venant des différents mo-

dules sont interprétés et utilisés correctement dans les limites de l'implémentation. Seules quelques améliorations restent à faire. Celles-ci sont discutés dans la section suivante.

4.6 Discussion

Bien qu'aucune phrases n'aient été codées dans l'architecture, des réponses bien formé sont bien renvoyé à l'utilisateur selon les règles grammaticales fournies. Les différents modules répondent bien aux critères des tests effectués. L'*interpréteur de sens* demande bien les informations au *bloc linguistique*, les informations sur l'environnement se font bien par l'*environnement virtuel* et l'historique est bien archivé et retrouvé grâce aux *connaissances dynamiques*. Le *bloc linguistique* fait également son travail en appliquant les différentes règles d'accords, de conjugaison et de syntaxe, ainsi qu'en donnant les définitions de trois natures distinctes, soit directes, formalisées ou hiérarchiques. Par contre, des améliorations restent à être effectuées pour être pleinement fonctionnel dans une application réelle.

Parmi les problèmes connus, il y a certaines définitions qui se chevauchent comme *en dessous* et *dans*. En effet, d'un point de vue mathématique, ces deux termes ont une définition similaire pour des objets généraux. Pour ce genre de problème, il serait important d'indiquer à la machine les objets qui peuvent contenir d'autres objets ou qui sont ouverts par endroits. Ainsi, la machine ferait la différence entre *en dessous de la table* et *dans un tiroir de la table*. Ce problème est étroitement lié à la précision de l'*environnement virtuel*. Dans le cas présent de cette recherche, l'utilisation d'un prisme pour représenter la table entière est loin de représenter efficacement une table. En effet, une table a un dessus et des pattes qui ne sont pas représentés par le fait que la table soit un prisme qui inclut la table. Cette simplification géométrique ainsi que la méconnaissance de la possibilité d'être inclusif sont responsables de ce malentendu au niveau de certains termes. Des indications supplémentaires dans les définitions, comme la possibilité de contenir un objet, serait probablement suffisant pour corriger ce chevauchement. Une autre possibilité serait d'avoir une représentation plus précise des objets. Ainsi, un objet à l'intérieur des limites d'un autre serait *dans* cet objet alors qu'un objet qui se trouve entre les limites d'objets qui définissent un objet plus gros sans être dans ces objets serait *en dessous*.

Un autre problème connu est celui du «H» muet et aspiré. En effet, pour le moment, tous les «H» sont considérés comme étant muets. Pour corriger ce problème, il faudrait inclure la prononciation phonétique à l'intérieur du dictionnaire. D'ailleurs, la présence de la notation phonétique pourrait servir pour le *générateur vocal*. Ainsi, si le mot débute

par «H» lorsque vient le temps d'appliquer une élision, celle-ci ne s'appliquera que si le «H» est muet et non s'il est aspiré.

Un autre problème vient du fait que, n'ayant pas entré tous les mots du dictionnaire, certains termes sont inconnus de la machine et elle ne peut pas générer de réponse satisfaisante à partir de requêtes utilisant ces mots. La seule façon de régler ce problème est de pouvoir entrer la totalité du dictionnaire et de la grammaire. Par contre, pour les exemples utilisés pour la démonstration de la validité de l'architecture, il n'y a pas de problèmes de fonctionnement.

Le dernier problème connu est le fait que tous les événements passés sont considérés au même instant par la machine. En fait, il s'agit uniquement d'un manque de précision dans le module des *connaissances dynamiques*. Une façon simple et efficace de régler ce problème serait de modifier ce module et d'utiliser une base de données. L'acquisition de l'information se ferait plus rapidement, et des informations supplémentaires sur le moment précis de l'état pourraient être conservées. Étant donné le but des travaux de recherche, il a été choisi de n'utiliser qu'un seul système et langage, ce qui cause cette limitation.

Les problèmes mentionnés n'empêchent pas de démontrer la fonctionnalité de l'architecture. Il ne s'agit que d'améliorations qui rendraient plus près de la réalité ce que la machine doit faire pour intégrer complètement la langue française.

Il y a également d'autres améliorations qui pourraient être apportées. Une d'entre elles pourrait être d'inclure le dictionnaire d'occurrences plutôt que de compter les occurrences *manuellement*. Cela éviterait une opération à la machine, cependant, celle-ci pourrait donner des résultats qu'elle n'a jamais rencontrés lorsqu'on demande un terme spécifique, ce qui, dans ce cas, l'éloignerait du comportement humain. En effet, un enfant qui n'a jamais vu ou entendu parler de perroquets ne nommera jamais le perroquet lorsque quelqu'un lui demande de nommer un oiseau, ce que le dictionnaire d'occurrences permettrait de faire. Également, il serait possible d'envisager d'avoir des tables de conjugaisons et d'accord plutôt que des règles qui les déterminent dynamiquement. Cependant, le fait que la machine fasse elle-même ses accords ouvre une avenue intéressante puisqu'il n'est plus nécessaire d'entrer la totalité d'un verbe qui est ajouté aux connaissances de la machine, mais uniquement le modèle de conjugaison que ce verbe doit suivre. Cette façon de faire demande moins d'espace mémoire, mais plus de traitement de la part de la machine. Une autre amélioration est d'avoir des modules appartenant à différents systèmes pour optimiser leur fonctionnement respectif. Par exemple, les demandes qui donnent des réponses directes, ou encore les formules mathématiques pour obtenir de l'information, les modules

concernés pourraient être relocalisés dans du code orienté objet, en C++ par exemple. Il serait également intéressant de voir comment l'architecture interagirait en utilisant d'autres techniques d'intelligence artificielle pour remplacer certains modules.

Dans le cas où l'application change de contexte, il n'y a que quelques modification à apporter aux modules. L'exemple d'une application d'agent de dépanage pour téléphone cellulaire sera pris en exemple de comparaison avec l'exemple de l'aide autonome utilisé dans le mémoire. Premièrement, le dictionnaire devra être mis à jour afin d'inclure la terminologie propre à l'application. Le contexte d'agent de dépanage implique, entre autre, que le *téléphone* doit être *consicent* de sa propre conception interne et non des objets à l'intérieur d'une habitation. Il faudra donc inclure les termes concernant les pièces du téléphone, les procédures de résolution de problèmes, la conception haut niveau des logiciels internes et autres informations techniques utiles dans le dictionnaire. Il faut également s'assurer que les verbes d'usages dans ce contexte sont présent. De plus, l'environnement virtuel devra comprendre les composantes du téléphones. Pour ce qui est des règles de grammaire et de conjugaison, elles resteront inchangées dans l'optique ou l'application reste dans la même langue, soit le français. Comme la polysémie et la synonimie n'est pas prise en charge pour le moment, il faudra s'assurer de la monosémie de la terminologie inclue dans le téléphone. Toutefois, le reste de l'application restera inchangé.

En conclusion, les tests démontrent bien la fonctionnalité des modules, mais il reste des améliorations avant de pouvoir l'intégrer dans un système complet d'une application réelle. Toutefois, malgré les améliorations suggérées, l'architecture implémentée démontre la fonctionnalité et la faisabilité d'un tel système.

CHAPITRE 5

CONCLUSION

L'objectif principal de ce projet était de démontrer qu'il est possible pour la machine de s'approcher du modèle linguistique, ici un modèle proposé par Kleiber, et de s'arrimer à un modèle cognitif, ici celui du STI, dans le but d'interpréter un message et de répondre de manière *réfléchie* à ce message. Pour y arriver, ce projet propose une nouvelle architecture générique de traitement du message basé sur les modèles des linguistes ainsi qu'une implémentation de cette architecture.

Le but de ce projet était une démonstration de concept. Pour une application complète, l'intégralité de la grammaire, du dictionnaire et du conjugueur devront être présents, ce qui allongera le traitement sur les données. Toutefois, compte tenu du fait que le temps de réponse du système actuel se fait toujours en moins d'une seconde, un système de traitement du message en temps réel est possible si, tel que suggéré au chapitre 4, les systèmes dédiés, comme les applications en C++, réseaux de neurones artificielles, logique floue et algorithmes génétiques sont présents pour supporter le système de logique. En effet, le modèle proposé par Kleiber montre la force du flou dans la compréhension d'un message. Ce projet n'utilise pas le plein potentiel du modèle, mais sa forme ouvre la voie pour y parvenir.

Au chapitre 4, il a été démontré que l'architecture répondait bien aux demandes qui lui étaient faites. Que ce soit pour la localisation, la recherche dans l'historique des événements, la salutation, la recherche de définitions, l'utilisation de la grammaire pour les accords de mots et la génération des phrases, l'utilisation du conjugueur pour former un verbe conjugué ou revenir à l'infinitif, pour donner un terme générique ou spécifique, l'architecture donnait les résultats souhaités dans les limites de ses connaissances. Ce chapitre a donc démontré qu'il est envisageable qu'une machine comprenne le sens d'un message écrit. Cependant, il fut discuté à la section 4.6 que cela demanderait une grande quantité de mémoire et de temps de calcul. Il est possible d'améliorer les performances de ce prototype en dédiant certaines parties de l'architecture à des systèmes spécialisés. Par exemple, toutes les données fixes pourraient être transférées vers une véritable base de données, les traitements qui donnent un résultat direct ou qui demandent un traitement sur une base aléatoire pourraient être transférés vers une application programmée en C++. Éga-

lement, plutôt que de simuler en Prolog une logique floue ou un algorithme génétique pour la classification des connaissances et certaines définitions de concepts de continuum bipolaire du dictionnaire, il serait avantageux d'avoir un système de type logique floue ou d'algorithme génétique pour faire le traitement. Il ne resterait plus en Prolog que le traitement linguistique tels les règles de grammaire et de conjugaison. Cela donnerait une aide considérable au niveau de la performance. De plus, l'*environnement virtuel* devrait être remplacé par un environnement virtuel plus précis ou encore être remplacé par une analyse par capteurs du monde réel.

D'autres problèmes connus restent à traiter, la polysémie et la synonymie. Un algorithme efficace devra être mis en oeuvre pour prendre en compte les différentes définitions des mots et les différents mots pour une définition. En effet, les linguistes se penchent encore sur ces problèmes récurants et certains linguistes considèrent même impossible d'y arriver. Présentement, l'application reconnait la différence entre le verbe *nommer* simple et pronominal, ce qui porte à croire que le traitement de la polysémie est possible en analysant le contexte de la phrase. Cependant, il reste que cet exemple possède des définitions semblables. Pour les mots avec une plus grande complexité polysémique, il peut en être autrement. Prendre en considération les phrases antérieures dans une discussions peut être une piste de solution au problème et sera nécessaire pour l'analyse des pronoms qui n'a pas été touchée dans cette démonstration, à l'exception du *je* et du *tu* pour qui l'association du terme de remplacement est moins variable.

Bien que le modèle actuel utilise plutôt la définition par fonctionnalité plutôt que par mot, les modifications proposées précédemment feront en sorte que le modèle *humain* pourra être pleinement utilisé. Cette décision, prise pour des raisons de design avec le langage de programmation utilisé, n'affecte en rien les résultats et sa conversion vers le modèle désiré est minimale. Ainsi, l'approche des linguistes semble être celle à considérer pour que la machine soit en mesure de comprendre ce que les êtres humains disent ou écrivent puisque, tel que mentionné, pour que deux *univers* se comprennent, il faut éliminer les différences qui existent entre les deux pour avoir une base commune.

BIBLIOGRAPHIE

BERSINI, H. (avril 2006) *De l'intelligence humaine à l'intelligence artificielle*, Ellipses Marketing, 192 p.

BESCHERELLE (1980) *L'Art de conjuguer*, Hurtubise HMH Ltée, 157 p.

BRUNELLE, E., et coll. (2003) *Antidote : le remède à tous vos mots - Posologie*, Druide informatique, 158 p.

CHOMSKY, N., SAUERLAND, U., GÄRTNER, H.-M. (jan 2007) *Approaching UG from Below*, Studies in Generative Grammar (StGG) : 89, Mouton de Gruyter, p. 1 – 29.

DEMARA, R., MOLDOVAN, D. (avril 1991) *The SNAP-1 Parallel AI Prototype*, ACM SIGARCH Computer Architecture News, vol. 19, n° 3, p. 2–11, Department of Electrical Engineering - Systems University of Southern California.

DUBOIS, J. (2001) *Dictionnaire de linguistique*, Larousse, 514 p.

ÉQUIPE AGENT LAND (2006) *Intelligent Agents and Bots*, http ://agentland.com.

FORD, N. (février 1993) *Programmer en Prolog*, Dunod informatique, 282 p.

GAGNON, M. (1996) *Expression de la localisation temporelle dans un générateur de texte*, Thèse de doctorat, Université de Montréal.

GOLDBERG, D. E., HOLLAND, J. H. (1988) *Genetic algorithms and machine learning*, Machine Learning, vol. 3, p. 95–99.

GREVISSE, M. (1995) *Précis de grammaire française*, Duculot, 319 p.

HAWKE, S., HERMAN, I., PRUD'HOMMEAUX, E., RAGGETT, D., SWICK, R. (2008) *W3c semantic web activity*, http ://www.w3.org/2001/sw/.

ISHIGURO, H. (2006) *Interactive humanoids and androids as ideal interfaces for humans*, *Proceedings of the 11th international conference on Intelligent user interfaces*, p. 2–9.

JAUME, M. (janvier 1999) *Contribution de la sémantique à la programation logique*, Thèse de doctorat, École Nationale des Ponts et Chaussées.

KLEIBER, G. (août 1990) *La sémantique du prototype, Catégories et sens lexical*, Presses Universitaires de France : Linguistique nouvelle, 199 p.

KLEIBER, G. (octobre 1999) *Problèmes de sémantique, la polysémie en question*, Presses Universitaires : Septentrion, publication Sens et structures, 220 p.

KLEIN, E. (décembre 2006) *Computational Semantics in the Natural Language Toolkit*, *Proceedings of the Australasian Language Technology Workshop 2006*, p. 26–33.

KOZLOWSKI, R., MCCOY, K. F., VIJAY-SHANKER, K. (juillet 2003) *Generation of single-sentence paraphrases from predicate/argument structure using lexico-grammatical re-*

sources, Proceedings of the Second International Workshop, University of Delaware, p. 1–8.

MANN, W. C. (juin 1983) *An Overview of the Nigel Text Generation Grammar, 21st Annual Meeting of the Association for Computational Linguistics*, USC/Information Sciences Institute, p. 79–84.

MASSON, C.-E. (2003) *CRM738 - Traitement cognitif de l'information*, http ://pages.usherbrooke.ca/crm738-cem/.

MEL'CHUK, I. (1999) *Experience of the Theory of the Sense–Text Linguistic Models : Semantics and Syntax*, Shkola Yazyki Russkoi Kultury, Moscow.

MEUNIER, J.-G. (2008) *Centre d'analyse de texte par ordinateur*, http ://www.ling.uqam.ca/ato/.

MICHAUD, F., BROSSEAU, Y., CÔTÉ, C., LÉTOURNEAU, D., MOISAN, P., PONCHON, A., RAÏEVSKY, C., VALIN, J.-M., BEAUDRY, E., KABANZA, F. (2005) *Modularity and Integration in the Design of a Socially Interactive Robot, Proceedings IEEE International Workshop on Robot and Human Interactive Communication*, p. 172–177.

MONGENET, M., et coll. (décembre 2006) *Prolog*, Wikipedia France.

NEGNEVITSKY, M. (2005) *Artificial Intelligence : A Guide to Intelligent Systems*, 2e édition, Addison-Wesley, 415 p.

OQLF (2008) *Le grand dictionnaire terminologique*, http ://granddictionnaire.com, Office québécoise de la langue française.

RATTÉ, S. (1995) *Interprétations des structures syntaxiques : une analyse computationnelle de la structure des événements*, Thèse de doctorat, Université du Québec à Montréal.

ROBERT, P. (1996) *Le Nouveau Petit Robert*, DICOROBERT inc, 2551 p.

ROSCH, E., VARELA, F., THOMPSON, E. (1993) *L'inscription corporelle de l'esprit : sciences cognitives et expérience humaine*, Édition du Seuil, 377 p.

RUSSELL, S., NORVIG, P. (2003) *Artificial Intelligence, A Modern Approch*, 2e édition, Prentice Hall, 1081 p.

TANGUY, L. (1997) *Traitement automatique de la langue naturelle et Interprétation : Contribution à l'élaboration d'un modèle informatique de la Sémantique Interprétative.*, Thèse de doctorat, Université de Rennes 1.

TORTORA, G. J., GRABOWSKI, S. R., PARENT, J.-C. (1999) *Principe d'anatomie et de physiologie*, CEC, 1204 p.

WINOGRAD, T. (1978) *Language representation and psychology : On primitives, prototypes, and other semantic anomalies, Proceedings of the 1978 workshop on Theoretical issues in natural language processing*, Association for Computational Linguistics, p. 25–32.

ZADEH, L. A. (1965) *Fuzzy Sets*, Information and Control, vol. 8.

ZADEH, L. A. (1982) *Test-Score Semantics for Natural Languages*, North-Holland Publishing Company, p. 425–430.