



# Apprentissage basé sur l'usage en interaction humaine avec un assistant adaptatif

Clément Delgrange

## ► To cite this version:

Clément Delgrange. Apprentissage basé sur l'usage en interaction humaine avec un assistant adaptatif. Environnements Informatiques pour l'Apprentissage Humain. Université de Lyon, 2018. Français. NNT : 2018LYSE1290 . tel-02077000

**HAL Id: tel-02077000**

**<https://theses.hal.science/tel-02077000>**

Submitted on 22 Mar 2019

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



N° d'ordre NNT : 2018LYSE1290

## THÈSE DE DOCTORAT DE L'UNIVERSITÉ DE LYON

opérée au sein de  
l'Université Claude Bernard Lyon 1

École Doctorale ED476  
Neurosciences et Cognition

Spécialité de doctorat : Sciences Cognitives  
Discipline : Informatique

Soutenue publiquement le 13/12/2018, par :  
**Clément DELGRANGE**

---

# Apprentissage basé sur l'usage en interaction humaine avec un assistant adaptatif

---

Devant le jury composé de :

HASSAS Salima, Professeure des Universités, Université Claude Bernard Présidente

AUFAURE Marie-Aude, Présidente (HDR), Datarvest

FILLIAT David, Professeur, ENSTA ParisTech

OUDEYER Pierre-Yves, Directeur de Recherche, INRIA

Rapporteure

Rapporteur

Examineur

DOMINEY Peter Ford, Directeur de Recherche, CNRS

DUSSOUX Jean-Michel, Responsable R&D, Cloud-Temple

Directeur de thèse

Invité

## Résumé

Aujourd'hui, un utilisateur peut interagir avec des assistants virtuels, comme Alexa, Siri ou Cortana, pour accomplir des tâches dans un environnement numérique. Dans ces systèmes, les liens entre des ordres exprimés en langage naturel et leurs réalisations concrètes sont précisées lors de la phase de conception. Une approche plus adaptative consisterait à laisser l'utilisateur donner des instructions en langage naturel ou des démonstrations lorsqu'une tâche est inconnue de l'assistant. Une solution adaptative devrait ainsi permettre à l'assistant d'agir sur un environnement numérique plus vaste composé de multiples domaines d'application et de mieux répondre aux besoins des utilisateurs. Des systèmes robotiques, inspirés par des études portant sur le développement du langage chez l'humain, ont déjà été développés pour fournir de telles capacités d'adaptation. Ici, nous étendons cette approche à l'interaction humaine avec un assistant virtuel qui peut, premièrement, apprendre le lien entre des commandes verbales et la réalisation d'actions basiques d'un domaine applicatif spécifique. Ensuite, il peut apprendre des liens plus complexes en combinant ses connaissances procédurales précédemment acquises en interaction avec l'utilisateur. La flexibilité du système est démontrée par sa forte adaptabilité au langage naturel, sa capacité à apprendre des actions dans de nouveaux domaines (Email, Wikipedia,...), et à former des connaissances procédurales hybrides en utilisant plusieurs services numériques, par exemple, en combinant une recherche Wikipédia avec un service de courrier électronique.

# Table des matières

Liste des figures . . . . .	5
Liste des tableaux . . . . .	6
<b>Avant-propos</b>	<b>6</b>
<b>1 Introduction</b>	<b>8</b>
1.1 Motivations . . . . .	10
1.2 Contributions . . . . .	12
1.3 Plan de thèse . . . . .	13
<b>2 Contexte industriel et systèmes adaptatifs</b>	<b>15</b>
2.1 Gestion d'infrastructure <i>Cloud Computing</i> . . . . .	15
2.2 Les besoins en adaptation . . . . .	16
2.3 Les systèmes adaptatifs . . . . .	17
2.4 Interface dialogique pour l'adaptation des systèmes . . . . .	18
<b>3 Interaction par parole pour la réalisation de tâches</b>	<b>23</b>
3.1 Émulation d'un dialogue . . . . .	23
3.2 Les principaux composants d'un système de dialogue . . . . .	26
3.3 Application pour les agents instructibles . . . . .	28
3.3.1 Plow . . . . .	28
3.3.2 Lia . . . . .	29
<b>4 Approche cognitive du langage et mise en œuvre en robotique</b>	<b>31</b>
4.1 Contexte . . . . .	31
4.2 Développement du langage . . . . .	32
4.3 Application en robotique . . . . .	34
4.4 Transfert vers le monde numérique . . . . .	36
<b>5 Agent virtuel apprenant</b>	<b>38</b>
5.1 Principe de fonctionnement . . . . .	38
5.2 Environnement partagé . . . . .	40
5.3 Perception de l'environnement . . . . .	41
5.3.1 Structure des événements . . . . .	42
5.3.2 Représentation du sens . . . . .	43
5.3.3 Mémoriser les constructions . . . . .	45
5.4 Gestion du dialogue et apprentissage . . . . .	46
5.4.1 Interaction avec les autres modules . . . . .	46

---

5.4.2	Flux de contrôle du module <i>InterpreterProcess</i> . . . . .	48
5.5	Agir dans l'environnement . . . . .	50
5.6	Intégration des modules : Exemple . . . . .	52
5.7	Création des variables . . . . .	53
5.8	Cas d'utilisation . . . . .	54
5.9	Évaluation du système . . . . .	58
5.10	Évaluation cross-linguistique . . . . .	60
<b>6</b>	<b>Variabilité linguistique</b>	<b>65</b>
6.1	Une approche basée sur la similarité sémantique de paires de phrases . . . . .	66
6.2	Module de similarité sémantique . . . . .	67
6.2.1	Explication du modèle . . . . .	67
6.2.2	Adaptation des données et de la mesure de performance . . . . .	70
6.2.3	Mise en œuvre et résultats préliminaires . . . . .	70
6.3	Module de la variabilité linguistique . . . . .	72
6.3.1	Principe de fonctionnement . . . . .	72
6.3.2	La similarité sémantique pour la recherche de procédures candidates . . . . .	73
6.3.3	La similarité sémantique pour la détection d'arguments . . . . .	74
6.3.4	Mise en œuvre et résultats préliminaires . . . . .	76
6.4	Intégration du module de variabilité linguistique . . . . .	78
6.4.1	Adaptation du comportement de l'agent . . . . .	78
6.4.2	Exemples de cas d'utilisation . . . . .	79
6.5	Conclusion . . . . .	82
<b>7</b>	<b>Implémentation et infrastructure</b>	<b>83</b>
7.1	Choix technologiques . . . . .	83
7.2	Structuration du projet . . . . .	84
7.3	Étendre le système avec de nouveaux services . . . . .	86
<b>8</b>	<b>Discussion et conclusion</b>	<b>88</b>
8.1	Résumé . . . . .	88
8.2	Pertinence de l'approche . . . . .	89
8.3	Futurs travaux . . . . .	91
<b>A</b>	<b>Diagrammes de séquence : « What is a tweet ? »</b>	<b>93</b>
A.1	Déclenchement d'une procédure . . . . .	94
A.2	Réalisation d'une action . . . . .	95
A.3	Prise en compte du résultat d'une action . . . . .	96
A.4	Réponse à l'utilisateur . . . . .	97
<b>B</b>	<b>Article : Usage-Based Learning in Human Interaction with an Adaptive Virtual Assistant</b>	<b>98</b>
	<b>Bibliographie</b>	<b>114</b>

---

# Table des figures

1.1	Un opérateur apprenant à un robot des actions basiques . . . .	11
1.2	Un opérateur exécutant un plan coopérativement avec un robot	11
2.1	Documentation d'un service web pour les e-mails : Ensemble des actions possibles . . . . .	19
2.2	Documentation d'un service web pour les e-mails : Description d'un message . . . . .	20
2.3	Un client e-mail en ligne de commande . . . . .	21
2.4	Un client e-mail graphique . . . . .	21
3.1	Architecture du système de dialogue Jaspis . . . . .	27
3.2	Architecture du système de dialogue OpenDial . . . . .	27
3.3	Le système Plow . . . . .	28
3.4	Le système TRIPS . . . . .	28
3.5	Ensemble des commandes du système LIA . . . . .	29
3.6	Ensemble d'entraînement pour le système LIA . . . . .	29
3.7	Interface du système LIA . . . . .	30
4.1	Exemple de constructions grammaticales . . . . .	33
4.2	Interaction Homme – Robot . . . . .	35
4.3	L'architecture Chris . . . . .	35
5.1	Architecture générale de l'agent virtuel . . . . .	39
5.2	Environnement partagé . . . . .	40
5.3	Mise à jour du modèle d'environnement . . . . .	43
5.4	Exemple d'une construction grammaticale ancrée sur des ac- tions primitives . . . . .	44
5.5	Exemple de procédure apprise . . . . .	45
5.6	Diagramme d'activité du module <i>InterpreterProcess</i> . . . . .	47
5.7	Une procédure illustrant la compositionnalité sémantique. . . . .	50
5.8	Algorithme « <i>One-shot Learning</i> » . . . . .	51
5.9	Cas d'utilisation : Envoi d'un message . . . . .	57
5.10	Email sent in scenario 3 . . . . .	58
5.11	Email sent in scenario 4 . . . . .	59
5.12	Évaluation du système . . . . .	60
6.1	Architecture du modèle ECNU . . . . .	67

6.2	Boxplot du résultat de dix <i>4-fold cross validation</i> sur un jeu de données appliquées à l'agent . . . . .	72
6.3	Diagramme de communication du module de variabilité linguistique . . . . .	73
6.4	Étapes algorithmiques du module de variabilité linguistique . . .	75
7.1	Infrastructure client–serveur de la plateforme agent . . . . .	85
8.1	Espace des politiques . . . . .	92
A.1	Diagramme de séquence illustrant l'interprétation de la commande utilisateur « <i>What is a tweet ?</i> ». Dans cette étape, la phrase est liée à une procédure précédemment apprise et l'assistant commence à l'exécuter. . . . .	94
A.2	Diagramme de séquence illustrant l'interprétation de la commande utilisateur « <i>What is a tweet ?</i> ». Dans cette étape, l'assistant attend de percevoir le résultat de sa première action, remplir un champ, avant de déclencher l'action suivante : lancer la recherche. . . . .	95
A.3	Diagramme de séquence illustrant l'interprétation de la commande utilisateur « <i>What is a tweet ?</i> ». À cette étape, l'assistant attend que la page Wikipédia affiche le résultat de la recherche. . . . .	96
A.4	Diagramme de séquence illustrant l'interprétation de la commande utilisateur « <i>What is a tweet ?</i> ». Ici, l'assistant prend le premier paragraphe de la page Wikipédia et l'envoie à l'utilisateur via le <i>chat</i> . . . . .	97

# Liste des tableaux

6.1	Exemples de variation de la forme d'un énoncé. . . . .	66
6.2	Corrélation de Pearson sur un jeu de données de test appliquées à l'agent ( <i>4-fold cross validation</i> ) . . . . .	71
6.4	Scores obtenus pour la détection d'arguments . . . . .	77
6.6	Scores obtenus pour des phrases inconnues . . . . .	77



# Avant-propos

Cette thèse a été préparée dans le cadre d'une collaboration académique-industrielle via une bourse CIFRE. Je remercie Peter Ford Dominey, Jean-Michel Dussoux et Cloud-Temple de m'avoir permis cette opportunité. Le contexte CIFRE est très motivant et dynamique. Il associe le besoin de prendre conscience des objectifs de la recherche industrielle et celui de faire progresser l'état de la technique du point de vue académique. Bien sûr, cela crée des défis. En fin de compte, j'espère avoir satisfait ces deux exigences de manière modeste mais réelle. Le système que j'ai développé est désormais le point de départ du développement futur d'un assistant intelligent pour notre partenaire industriel Cloud-Temple. Sur la base de ces résultats, un nouveau doctorat CIFRE est en cours afin d'étendre les recherches que j'ai commencées. De même, je continuerai à travailler sur ce projet chez Cloud-Temple et avec l'équipe de l'INSERM après le doctorat. D'un point de vue académique, mon développement de l'approche basée sur l'usage pour la conception d'un assistant intelligent fait l'objet d'un article en cours d'évaluation, après révision et resoumission à l'IEEE Transactions sur les systèmes cognitifs et développementaux.

# Chapitre 1

## Introduction

Cette thèse s’inscrit dans le contexte d’un système qui apprend à partir d’interactions avec un humain à réaliser des tâches sur un environnement numérique. Dans la réalisation d’un tel système, l’approche peut être envisagée par deux extrêmes. D’une part, celle où l’ingénieur doit anticiper l’adaptabilité du système aux interactions humaines et à l’ensemble des tâches réalisables, et donc intégrer un moteur d’analyse linguistique sophistiqué, implémenter des ontologies qui fournissent des connaissances de base, un moteur de planification, etc. Cette approche peut s’appuyer sur les outils existants et est prête à être mise en œuvre pour accomplir des tâches complexes dès le départ. Le système PLOW développé par James Allen et ses collègues entre dans cette catégorie (Allen et al. 2007). Cette approche a ses limites. En particulier, l’important travail d’ingénierie pour intégrer un grand nombre de capacités dès le début rend difficile l’adaptation du système à de nouveaux domaines (par exemple l’adaptation à un nouveau système de messagerie électronique), c’est également le cas pour l’adaptation à d’autres langues. À l’autre extrémité se trouverait un système adaptatif, ayant peu de capacités, autre que celles de percevoir l’environnement partagé avec l’être humain et d’effectuer des actions fondamentales dans cet espace partagé. C’est la position que nous adoptons dans cette thèse.

Il est intéressant de noter que cette distinction est analogue à un débat dans le domaine de la cognition humaine. Les nativistes soutiennent que le nourrisson humain est pré-équipé de modules puissants comme la grammaire universelle – un système d’analyse du langage et de grammaire hautement développé qui n’a besoin d’être paramétré que lors de l’apprentissage du langage chez l’enfant (Chomsky 1995) – et d’autres modules connexes pour la théorie de l’esprit, l’agentivité, la numérosité, la physique, la psychologie, la géométrie, les nombres, etc (Spelke 2000 ; Spelke et Kinzler 2007 ; Carey 2000 ; Spelke, Bernier et Skerry 2013). Un argument majeur en faveur de cette approche « intégrée » est l’affirmation selon laquelle le monde est si complexe qu’il serait impossible d’apprendre de telles capacités. C’est particulièrement le cas pour le langage, où l’argument de la pauvreté du stimulus qui soutient que les données auxquelles l’enfant est exposé sont si pauvres en information qu’il serait impossible d’apprendre le langage sans une grammaire universelle intégrée (Chomsky

---

1995). D'autres approches ont commencé à faire valoir que le stimulus n'est pas si faible et que, grâce au riche lien social de l'attention jointe, la perception des enfants serait dirigée de sorte que le langage et les autres capacités cognitives pourraient effectivement être appris par l'expérience (Mandler 1992 ; Tomasello 2009, 2003b ; Dominey et al. 2003 ; Mandler 2012).

Ainsi, notre choix d'avoir un système « basé sur l'usage » qui tire les leçons de l'expérience, permettra d'obtenir deux résultats potentiellement intéressants. Il peut fournir une méthode pour développer un assistant intelligent, et s'il le fait, il fournira également un argument en faveur de la position constructiviste dans le développement cognitif humain. Nous considérons maintenant l'environnement numérique dans lequel l'agent va agir.

Notre environnement numérique est constitué d'un nombre grandissant de services applicatifs augmentant ainsi les possibilités d'usages et d'interactions avec ces services. Pour autant, un utilisateur devra souvent les orchestrer « manuellement » dans l'accomplissement d'une tâche spécifique. La raison est que l'automatisation possède un coût et demande du temps. Un concepteur devra recueillir, spécifier, implémenter et évaluer une solution pour chaque nouveau besoin. Ce cycle long a du mal à s'adapter à la rapidité avec laquelle les utilisateurs s'approprient les services numériques et créent de nouveaux besoins.

L'*Internet des Objets* (IdO) est un cas particulièrement représentatif, où les objets connectés, comme un réfrigérateur, se multiplient rapidement. Ces objets, par leurs interactions, offrent un large éventail de nouveaux cas d'usage, comme connecter son réfrigérateur à une liste de courses. Ainsi, pour créer des systèmes qui permettent à un utilisateur de les contrôler, les questions de coût et de rapidité des développements seront particulièrement importantes : comment anticiper l'usage d'un nouvel objet, comment ne pas brider les interactions entre ces objets, comment créer une interaction efficace avec l'utilisateur quand le nombre de fonctionnalités augmente. L'objectif de simplifier ces interactions et de concevoir des systèmes qui s'adaptent au plus proche des besoins des utilisateurs est donc un défi de plus en plus nécessaire à relever.

Ces questions sont également un enjeu pour la conception d'applications métiers dont l'environnement évolue vite. Pour illustrer ce point, nous pouvons prendre l'exemple de la société *Cloud Temple*, dont le métier est de fournir une solution de virtualisation d'infrastructures informatiques sur lesquels repose le système d'information de leurs clients. Cette virtualisation consiste à simuler numériquement des équipements qui sont normalement physiques comme un disque de stockage, une unité centrale, un câble réseau et où des fournisseurs de technologies proposent leurs services applicatifs pour les gérer. Le morcellement de ces technologies et de l'expertise nécessaire pour les maîtriser, impliquent pour la société, d'offrir une vue simplifiée de cette infrastructure virtuelle qui s'adapte aux besoins de leurs clients. Cela nécessite de concevoir des outils d'automatisation et donc d'arbitrer les différents développements nécessaires au détriment d'une adéquation optimale avec leurs besoins.

## 1.1 Motivations

Un des axes permettant de répondre à cette problématique réside dans la conception d’interfaces Homme – Machine adaptatives capables d’interactions naturelles. Ce type de systèmes comprend l’emploi du langage naturel mais également la mise en place d’interactions dialogiques proches de celles réalisées entre humains dans la réalisation d’une tâche. Ces systèmes offrent de bonnes perspectives car, d’une part, ils permettent de simplifier l’interaction entre un utilisateur et son environnement numérique en donnant à l’utilisateur un rôle plus actif dans l’établissement du contexte d’une tâche. Par exemple, l’utilisateur peut exprimer un besoin cible, comme *envoyer un email à quelqu’un*, là où le système pourra s’adapter et faire correspondre cette demande à l’exécution d’un service spécifique. D’autre part, le langage naturel est également un moyen efficace pour transmettre notre connaissance, offrant ainsi à l’utilisateur la possibilité de jouer un rôle essentiel pour recueillir, voir combler de nouveaux besoins au sein même du système.

La possibilité d’instruire un système en langage naturel est pourtant peu exploitée. En effet, les systèmes de dialogue sont le plus souvent conçus pour fournir leur rôle adaptatif dès la fin de leur conception. Ceci peut s’expliquer par la complexité des modules mis en œuvre. Bien que le besoin d’instruire ces systèmes diminue à mesure qu’ils deviennent performants, cette possibilité est rendue également plus difficile à réaliser.

Les recherches portant sur le développement du langage suggèrent qu’il joue un rôle important dans notre capacité à donner un sens aux événements qui nous entourent et qu’il s’établit progressivement dès le plus jeune âge, où l’expérience sensorielle est accompagnée par le langage naturel. Ainsi, les enfants commencent par associer des mots aux objets et aux agents de leur scène perceptuelle, puis apprennent des représentations primaires autour d’événements répétitifs jusqu’à acquérir des représentations de plus en plus abstraites et généralisées. Les premiers systèmes numériques à exploiter ces travaux sont réalisés dans le domaine de la robotique. Comme l’illustre la Figure 1.1, les expériences montrent des humains apprenant aux robots à réaliser des actions en utilisant plusieurs modalités, comme le langage, des démonstrations physiques ou kinesthésiques. Cette étape correspond au premier niveau d’apprentissage du langage chez l’enfant, où les mots sont ancrés dans des primitives sensorimotrices. Les opérateurs sont ensuite en mesure d’exécuter un plan coopérativement avec le robot en utilisant le langage naturel comme illustré par la Figure 1.2.

Cette méthode d’apprentissage par l’usage permet au robot de s’affranchir d’un modèle préétabli d’un langage et de l’environnement qui l’entoure. En contrepartie, il dispose des structures et des processus lui permettant d’apprendre à appréhender son environnement grâce à des interactions avec un humain et son environnement.

Pour les interfaces Homme – Machine en langage naturel, l’apprentissage du langage par l’usage permet à l’utilisateur d’établir lui-même le rapprochement entre son système symbolique de communication et les systèmes symboliques

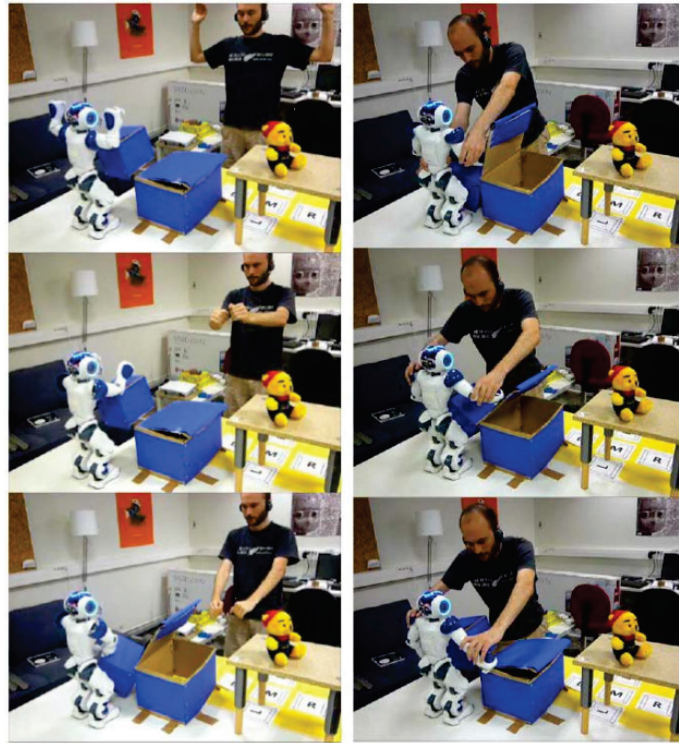


FIGURE 1.1 – Un opérateur apprenant à un robot à réaliser des actions en utilisant plusieurs modalités : langage naturel, démonstration physiques et démonstrations kinesthésiques.

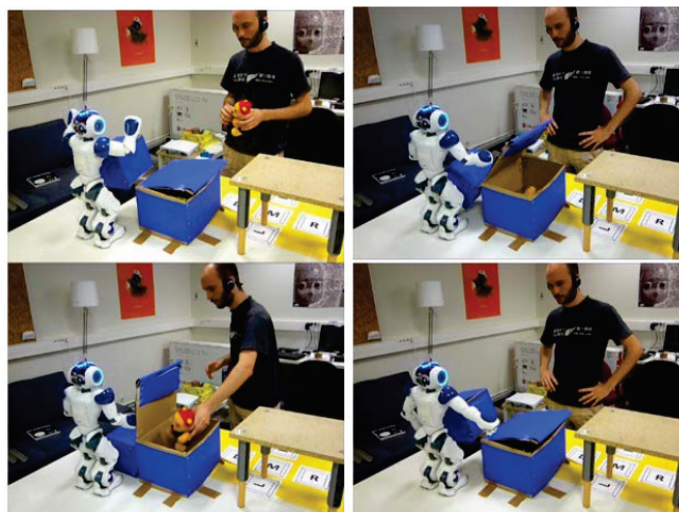


FIGURE 1.2 – Un opérateur exécutant un plan coopérativement avec un robot.

des services qu'il utilise. Cette méthode procure entre autres les avantages suivants :

- De construire une infrastructure légère dont le nombre de services peut être plus facilement dimensionné et ainsi fournir à l'utilisateur l'adaptation nécessaire pour orchestrer un nombre arbitraire de services dès le début du cycle de vie du système.
- D'améliorer l'utilisabilité du système quand celui-ci n'est pas en mesure de comprendre une demande.
- D'adapter plus facilement l'interface dans des domaines où des mots ne font pas partie du langage courant ou ont une autre signification.

## 1.2 Contributions

Dans cette thèse, je mets en œuvre un système qui apprend à la fois par démonstrations humaines et par des explications en langage naturel à orchestrer des services numériques. J'ai développé un environnement commun où l'humain et le système peuvent agir et surtout, où le système peut observer les démonstrations effectuées par l'humain. Le système apprend la correspondance entre des commandes en langage naturel telles que « envoyer un e-mail à john@inserm.fr au sujet de la fête et dire que je serai en retard » et l'enchaînement des actions nécessaires pour réaliser le but de la commande. Le système permet de généraliser ces commandes de façon transparente, pour pouvoir naturellement répondre à une nouvelle commande comme « envoyer un message à mary@inserm.fr à propos de la réunion et dire que nous devons la faire demain ». Le système est conçu pour que l'être humain puisse enseigner incrémentalement les commandes. Ainsi les nouveaux apprentissages peuvent s'appuyer sur des apprentissages précédents en les combinant et ainsi optimiser les étapes d'apprentissage.

Le système a été conçu pour répondre à de fortes exigences adaptatives. Ainsi aucune connaissance propre à une langue ou à un environnement de tâches n'est nécessaire. En outre, le système permet d'accumuler des données riches issues de sa propre expérience, qui peuvent ensuite être exploitées pour offrir de nouvelles capacités. Un exemple d'utilisation de ces données a été le développement d'un algorithme de détection de similarité sémantique, qui une fois intégré au système permet de généraliser les commandes en langage naturel qui contiennent des synonymes ou diverses locutions comme « envoie un e-mail » et « envoie un message s'il te plaît ».

En fin de compte, le système permet de montrer l'intérêt et la faisabilité d'une approche cognitive constructiviste, initialement développé dans les sciences du développement humain, pour la conception d'assistants virtuels dialogiques.



## 1.3 Plan de thèse

Une description du déroulement du manuscrit, chapitre par chapitre, est fournie ci-dessous :

### **Chapitre 2 : Contexte industriel et systèmes adaptatifs**

Ce chapitre présente le contexte industriel qui a initié le projet et le système cible souhaité sous l'angle des systèmes adaptatifs. Nous commençons par décrire la problématique de l'adaptabilité des systèmes aux changements des exigences utilisateur et de l'évolution de ces systèmes lors de l'ajout et de la maintenance de fonctionnalités. Puis, une approche est proposée pour améliorer l'adaptabilité des systèmes à ces facteurs. Elle consiste en l'élaboration d'une interface dialogique avec laquelle un utilisateur peut instruire un système en langage naturel afin de l'accommoder à ses besoins.

### **Chapitre 3 : Interaction par parole pour la réalisation de tâches**

L'utilisation du langage naturel au sein d'un système introduit de nombreux besoins en terme d'adaptation. Des facteurs tels que l'environnement ambiant, la langue de l'utilisateur, le domaine d'application ou les interactions dialogiques requièrent une certaine flexibilité dans la mise en œuvre de ces systèmes. Dans ce chapitre, nous présentons les principales caractéristiques d'un système de dialogue et de ses composants. Nous expliquons ensuite les principales approches employées dans la réalisation du module de compréhension du langage naturel et leur application au sein de systèmes qui peuvent être instruits incrémentalement par un utilisateur en langage naturel. Pour conclure, nous mettons en perspective ces approches avec la conception d'un système de dialogue qui s'adapte à la fois aux exigences utilisateur et à l'évolution de l'environnement des tâches.

### **Chapitre 4 : Approche cognitive du langage et mise en œuvre en robotique**

Ce chapitre introduit les concepts clefs d'une approche cognitive constructiviste de l'apprentissage du langage, en quoi cette approche diffère de l'approche générative et comment elle pourrait être exploitée dans la réalisation d'interfaces Homme – Machine dialogiques. En effet, cette approche a déjà été exploitée par des systèmes robotiques où le principal intérêt était de créer des robots qui soient en mesure de communiquer et d'être contrôlés par des humains dans un monde ouvert. Ces derniers partagent un environnement commun, ce qui fait de la robotique un domaine d'application propice pour mener ce genre d'expérimentations. Ainsi,

nous présentons le fonctionnement de ces systèmes et le transfert de ces techniques pour le monde numérique.

### **Chapitre 5 : Agent virtuel apprenant**

Ce chapitre présente la plateforme agent développée au cours de la thèse. Après une explication générale du principe de fonctionnement du système, chaque module est décrit un à un, afin de détailler leur rôle, leur structuration des données et leur relation avec les autres modules de la plateforme. L'interaction globale entre les modules et les principaux algorithmes mis en œuvre sont ensuite expliqués autour d'un scénario d'usage spécifique. Pour finir une liste exhaustive de cas d'utilisation, avec des exemples de dialogues, est donnée afin d'illustrer le comportement des agents dans des conditions réelles d'utilisation.

### **Chapitre 6 : Variabilité linguistique**

L'un des problèmes récurrents lors d'interactions en langage naturel avec un système est la nécessité pour l'utilisateur de simplifier son langage et le sentiment de devoir « parler comme un robot » pour être compris. Ce chapitre traite spécifiquement de ce point et montre comment l'agent virtuel peut améliorer sa tolérance à la variabilité linguistique des énoncés et permettre ainsi aux utilisateurs de s'exprimer avec moins de contraintes.

### **Chapitre 7 : Implémentation et infrastructure**

Ce chapitre présente les outils et les technologies utilisés lors du développement de la plateforme agent et du module de variabilité linguistique. Il explique également le travail nécessaire pour étendre les fonctionnalités de la plateforme avec de nouveaux services. Enfin une description détaillée des modules annexes pouvant être réutilisés dans d'autres contextes est donnée en fin de chapitre.

### **Chapitre 8 : Discussion et conclusion**

Ce chapitre conclut la dissertation avec un bref résumé des travaux recherches et une mise en perspective avec l'état de l'art actuel des systèmes cognitifs apprenants et des interfaces dialogiques Homme – Machine. Nous finissons avec une présentation des futurs travaux envisagés dans le cadre du projet.



## Chapitre 2

# Contexte industriel et systèmes adaptatifs

Ce chapitre présente le contexte industriel qui a initié le projet et le système cible souhaité sous l'angle des systèmes adaptatifs. Nous commençons par décrire la problématique de l'adaptabilité des systèmes aux changements des exigences utilisateur et de l'évolution de ces systèmes lors de l'ajout et de la maintenance de fonctionnalités. Puis, une approche est proposée pour améliorer l'adaptabilité des systèmes à ces facteurs. Elle consiste en l'élaboration d'une interface dialogique avec laquelle un utilisateur peut instruire un système en langage naturel afin de l'accommoder à ses besoins.

### 2.1 Gestion d'infrastructure *Cloud Computing*

La société Cloud Temple une Société de Service et d'Ingénierie Informatique (SSII) spécialisée en infogérance sur *Cloud Computing*. Ce type d'infogérance consiste à virtualiser les briques matérielles qui composent un système d'information classique (une unité centrale, un disque de stockage, un câble réseaux, etc) afin de rendre plus flexible leur maintenance. Ces dernières années, la taille et la complexité des infrastructures de type *Cloud Computing* ont explosé avec l'essor de nouvelles applications, rendant la gestion de ces infrastructures par les administrateurs de la société de plus en plus difficile. Ainsi, la société Cloud Temple possède une centaine de clients issus de tous les secteurs du tissu économique français : commerçants web, municipalités, banques, gestionnaires d'actifs, industriels ou encore média et qui possèdent des besoins propres. Les systèmes d'information de ces clients s'exécutent sur une infrastructure répartie sur de plus de 2000 serveurs hébergés dans quatre *data centers* localisés en France.

Parallèlement à cette accroissement de complexité, les infrastructures se développent autour de la notion de services de ressources tels que les serveurs physiques, les machines virtuelles, les capacités disques ou encore les adresses réseaux et récemment autour du concept de *Virtual Data Center* (VDC). La notion de VDC a été introduite pour offrir aux utilisateurs mais aussi aux administrateurs une plus grande flexibilité dans l'usage de leurs ressources. A

cette fin un VDC organise ses ressources matérielles et réseaux en pools définis spécifiquement pour répondre aux besoins métiers d'un client. Dès lors, ce dernier doit disposer d'une complète flexibilité dans l'affectation des ressources des pools au gré de ses besoins opérationnels. Ce niveau de flexibilité nécessite d'accéder en un point unique à l'ensemble des informations du VDC (l'état courant dans lequel se trouve l'infrastructure en terme de ressources allouées, consommées, restantes par type) et des opérations (création/suppression de machines virtuelles, de disques, de réseaux, etc.). Le client doit également accéder à la gestion des services logiciels existants ou à venir : sauvegarde, supervision, VPN-SSL, BigData, ou encore VOIP auxquels il aurait souscrit.

Un scénario typique de gestion de l'infrastructure, impliquant les opérateurs de la société et/ou ses clients, nécessite la réalisation coordonnée d'activités sur plusieurs services ou composants de l'infrastructure. Ces activités correspondent dans la majorité des cas à des tâches de diagnostic et de production/exécution de plans d'actions (retrouver les origines d'une panne, appliquer une procédure de remédiation). Dans les deux cas, la prise de décision repose sur la perception d'un environnement complexe. De la même façon, établir une suite d'actions susceptible de recouvrer une situation nominale requière une multitude de connaissances expertes impossibles à maîtriser dans leur globalité. En conséquence, les opérateurs sont régulièrement engagés dans une activité d'investigation pour déterminer quelle information est utile dans un contexte particulier ou quelle compétence sera nécessaire pour réaliser l'objectif. Avec l'expérience, certaines tâches deviennent par la suite fastidieuses, comme l'enchaînement des actions élémentaires d'un plan établi dans un contexte bien connu.

## 2.2 Les besoins en adaptation

La société fait face à des coûts importants en développement logiciel et en support. Afin de mieux comprendre les responsabilités qui incombent aux différents acteurs qui interviennent dans l'établissement d'une solution de *Cloud Computing* nous listons ci-dessous chacun de leur rôle :

**Le client** Le client possède un rôle double, celui d'utilisateur final qui consomme la solution de virtualisation et celui d'opérateur qui adapte une solution à ses besoins métiers.

**Le fournisseur de technologies** La société repose sur des fournisseurs de technologies de virtualisation qui mettent à disposition des services numériques pour contrôler les ressources virtuelles. Ces services numériques se présentent soit sous la forme de systèmes complets avec interface utilisateur, soit sous la forme d'Interfaces de Programmation Applicative (API) à partir desquels le consommateur de ces APIs peut construire une application plus aboutie.

**Les développeurs** Les fournisseurs de technologies offrent des services applicatifs morcelés et d'un niveau d'abstraction trop bas pour être directement utilisés par les clients. La société doit donc développer une couche

applicative qui correspond aux besoins métiers de leurs clients et leurs permettre d'accéder en un point unique à leur infrastructure.

**Les superviseurs** Les clients ont également besoin de support pour gérer au quotidien leur infrastructure. Les opérateurs de la société supervisent donc une partie des systèmes d'information de leurs clients afin de remédier aux pannes et d'assister les clients dans l'adaptation d'une solution à leurs besoins opérationnels.

La société possède plusieurs objectifs. Celui de contrôler les coûts en développement et en supervision mais également de s'adapter à des besoins spécifiques pour chaque client, de suivre les innovations opérées par les fournisseurs de technologies de virtualisation, d'aligner la vitesse des développements à l'évolution des besoins de leurs clients, etc. Lorsque cet environnement devient trop complexe, la conception de systèmes adaptatifs devient un levier pour atteindre l'ensemble de ces objectifs (Salehie et Tahvildari 2009).

Les principaux facteurs que la société souhaite adresser sont le changement fréquent des exigences utilisateurs, le besoin de créer des développements plus rapides et d'être flexible lors de l'ajout, de la suppression ou de la modification d'une fonctionnalité et pour lesquels les systèmes adaptatifs peuvent apporter une réponse (Subramanian et Chung 2001).

## 2.3 Les systèmes adaptatifs

Subramanian (Subramanian et Chung 2001) a répertorié différentes définitions des systèmes adaptatifs dont nous listons certaines d'entre elles ci-dessous :

- « Les systèmes auto-adaptatifs modifient leurs propres comportements en réponse à leur environnement opérationnel. »
- « L'adaptation est définie comme la facilité avec laquelle le système ou une partie du système peut s'adapter à des exigences changeantes. »
- « Un programme est adaptable s'il peut facilement être changé. Il est adaptatif, s'il peut automatiquement changer son comportement en fonction de son contexte. »
- « ... une métrique qualitatif qui peut être utilisée pour évaluer la facilité avec laquelle un logiciel permet à des contraintes techniques et à des besoins utilisateurs d'être satisfaits. »

On peut distinguer différents niveaux d'adaptation caractérisés par le terme « facilité ». À une extrémité on trouve les systèmes adaptables, ils sont conçus comme une boucle ouverte entre le système et le concepteur et nécessitent une intervention pour être adaptés à un nouvel environnement (modification du logiciel, configuration, redéploiement, ...). À l'autre extrémité, les systèmes auto-adaptatifs sont conçus comme une boucle fermée entre le système et son environnement et possèdent des procédures qui fournissent le rôle adaptatif de façon autonome.

Selon les facteurs, les systèmes atteignent un niveau d'adaptation différents. Ainsi, des facteurs comme la performance, la résilience ou la sûreté, bénéficient de techniques comme le *load-balancing*, la répartition des tâches (Fatima et Wooldridge 2001) ou des principes architecturaux (Subramanian et Chung 2001 ; Morandini, Penserini et Perini 2008) évolués pour concevoir des systèmes auto-adaptatifs. Tandis que des facteurs comme les exigences utilisateurs, l'ajout et la maintenance de fonctionnalités sont le plus souvent traités comme des besoins non-fonctionnels et profitent de techniques méthodologiques ou d'implémentation comme la programmation par aspect, les architectures en micro-services ou orientés composants (Subramanian et Chung 2001) dans le but de concevoir des systèmes adaptables.

## 2.4 Interface dialogique pour l'adaptation des systèmes

Rendre un système adaptatif à la fois à l'évolution des besoins d'un utilisateur et à l'évolution de ses fonctionnalités peut également profiter à des domaines où les usages et les technologies avancent vite comme l'Internet des Objets (IdO), la domotique ou la mobilité. C'est dans ce contexte que la société Cloud Temple a entrepris de concevoir un système où un utilisateur final peut lui-même exprimer ses besoins et fournir les instructions nécessaires pour les combler.

Le projet consiste en l'élaboration de cet outil sous la forme d'un assistant dialogique qui au travers de multiples interactions en langage naturel avec un utilisateur, est capable d'établir coopérativement l'état cible de l'environnement, la situation courante et la suite d'actions appropriées vers le but tout en tenant compte de son utilisateur (niveau d'expertise, contrainte sur l'ordre des actions, disponibilité des ressources...). L'un des aspects important du projet est que l'utilisateur puisse instruire le système sur un environnement aussi large et riche que possible.

Ainsi, l'environnement du système est composé d'un ensemble de services applicatifs. Ces services doivent être accessibles en se reposant sur des interfaces standards d'interaction afin de bénéficier d'un large écosystème. Comme illustré par les Figures 2.1, 2.3 et 2.4, cela comprend des applications graphiques, en ligne de commande ou des interfaces de programmation applicatives (par exemple des API Rest). Les services peuvent également être à des niveaux d'abstraction quelconques, c'est-à-dire qu'ils peuvent ne pas être pensé pour un utilisateur final comme illustré par le concept de « *payload* » dans la description d'un message d'une API Rest (Figure 2.2).

Le but du système peut ensuite être décrit par sa capacité à :

- Reconnaître un changement dans son environnement. Comme un nouveau besoin de l'utilisateur (un nouveau but), un nouveau service applicatif, etc.
- Déterminer coopérativement avec l'utilisateur le moyen de s'adapter à ce changement. Par l'élaboration d'un plan d'actions par exemple.

Method	HTTP request	Description
URIs relative to <code>https://www.googleapis.com/gmail/v1/users</code> , unless otherwise noted		
<code>delete</code>	DELETE <code>/userId/messages/id</code>	Immediately and permanently deletes the specified message. This operation cannot be undone. Prefer <code>messages . trash</code> instead.
<code>get</code>	GET <code>/userId/messages/id</code>	Gets the specified message.
<code>insert</code>	POST <code>https://www.googleapis.com/upload/gmail/v1/users/userId/messages</code> and POST <code>/userId/messages</code>	Directly inserts a message into only this user's mailbox similar to IMAP <code>APPEND</code> , bypassing most scanning and classification. Does not send a message.
<code>list</code>	GET <code>/userId/messages</code>	Lists the messages in the user's mailbox.
<code>modify</code>	POST <code>/userId/messages/id/modify</code>	Modifies the labels on the specified message.
<code>send</code>	POST <code>https://www.googleapis.com/upload/gmail/v1/users/userId/messages/send</code> and POST <code>/userId/messages/send</code>	Sends the specified message to the recipients in the To, Cc, and Bcc headers.
<code>trash</code>	POST <code>/userId/messages/id/trash</code>	Moves the specified message to the trash.
<code>untrash</code>	POST <code>/userId/messages/id/untrash</code>	Removes the specified message from the trash.
<code>import</code>	POST <code>https://www.googleapis.com/upload/gmail/v1/users/userId/messages/import</code> and POST <code>/userId/messages/import</code>	Imports a message into only this user's mailbox, with standard email delivery scanning and classification similar to receiving via SMTP. Does not send a message.

FIGURE 2.1 – Documentation d'un service web pour les e-mails : Ensemble des actions possibles

Property name	Value	Description	Notes
id	string	The immutable ID of the message.	
threadId	string	The ID of the thread the message belongs to. To add a message or draft to a thread, the following criteria must be met: <ol style="list-style-type: none"> <li>1. The requested threadId must be specified on the Message or Draft . Message you supply with your request.</li> <li>2. The References and In-Reply-To headers must be set in compliance with the <a href="#">RFC 2822</a> standard.</li> <li>3. The Subject headers must match.</li> </ol>	writable
labelIds[]	list	List of IDs of labels applied to this message.	writable
snippet	string	A short part of the message text.	
historyId	unsigned long	The ID of the last history record that modified this message.	
payload	nested object	The parsed email structure in the message parts.	
payload.partId	string	The immutable ID of the message part.	
payload.mimeType	string	The MIME type of the message part.	
payload.filename	string	The filename of the attachment. Only present if this message part represents an attachment.	
payload.headers[]	list	List of headers on this message part. For the top-level message part, representing the entire message payload, it will contain the standard RFC 2822 email headers such as To, From, and Subject.	
payload.headers[].name	string	The name of the header before the : separator. For example, To.	
payload.headers[].value	string	The value of the header after the : separator. For example, someuser@example.com.	
payload.body	nested object	The message part body for this part, which may be empty for container MIME message parts.	
payload.parts[]	list	The child MIME message parts of this part. This only applies to container MIME message parts, for example multipart/* . For non- container MIME message part types, such as text/plain, this field is empty. For more information, see <a href="#">RFC 1521</a> .	
sizeEstimate	integer	Estimated size in bytes of the message.	
raw	bytes	The entire email message in an RFC 2822 formatted and base64url encoded string. Returned in messages.get and drafts.get responses when the format=RAW parameter is supplied.	writable
internalDate	long	The internal message creation timestamp (epoch ms), which determines ordering in the inbox. For normal SMTP-received email, this represents the time the message was originally accepted by Google, which is more reliable than the Date header. However, for API-migrated mail, it can be configured by client to be based on the Date header.	

FIGURE 2.2 – Documentation d’un service web pour les e-mails : Description d’un message

```
$ mail -s "Hello World" someone@example.com
Cc:
Hi Peter
How are you
I am fine
Good Bye
<Ctrl+D>
```

FIGURE 2.3 – Un client e-mail en ligne de commande

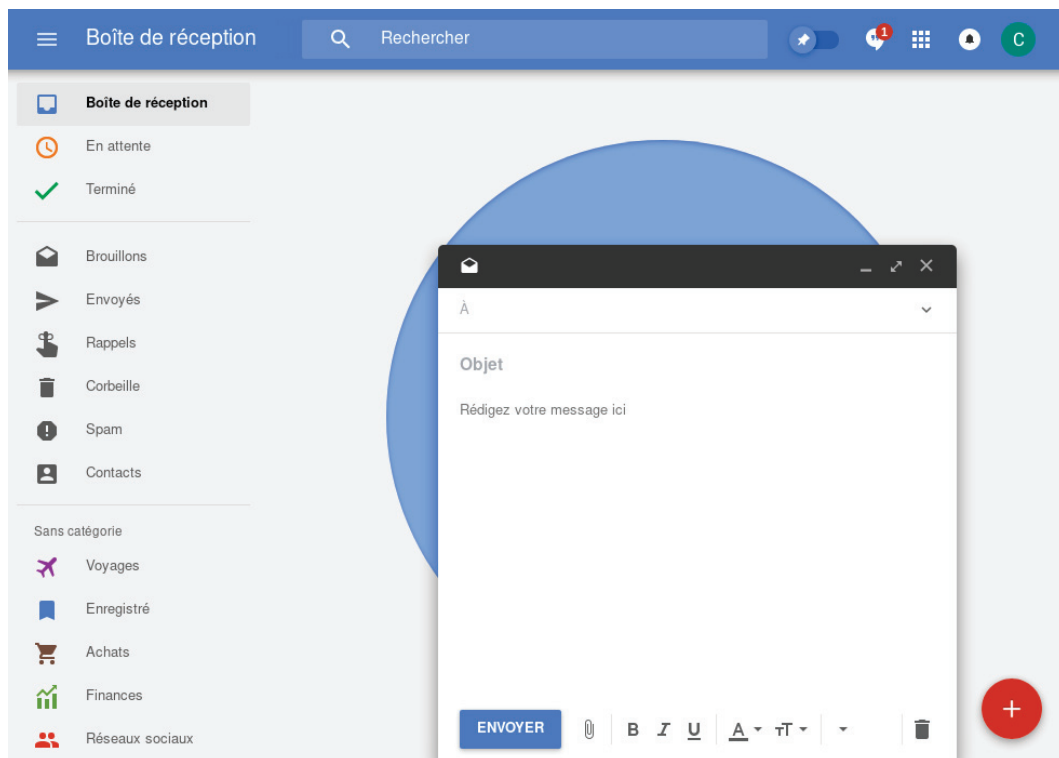


FIGURE 2.4 – Un client e-mail graphique

- D'appliquer des modifications au sein du système pour que le nouveau besoin de l'utilisateur ou le nouveau service ne soient plus perçus comme un changement dans le futur.



# Chapitre 3

## Interaction par parole pour la réalisation de tâches

L'utilisation du langage naturel au sein d'un système introduit de nombreux besoins en terme d'adaptation. Des facteurs tel que l'environnement ambiant, la langue de l'utilisateur, le domaine d'application ou les interactions dialogiques requièrent une certaine flexibilité dans la mise en œuvre de ces systèmes. Dans ce chapitre, nous présentons les principales caractéristiques d'un système de dialogue et de ses composants. Nous expliquons ensuite les principales approches employées dans la réalisation du module de compréhension du langage naturel pour systèmes qui peuvent être instruits incrémentalement par un utilisateur en langage naturel.

### 3.1 Émulation d'un dialogue

Le dialogue est une activité collaborative, dans laquelle deux ou plusieurs participants communiquent en vue de réaliser divers fonctions sociales. L'une de ces fonctions peut être la réalisation d'une tâche, le dialogue est alors caractérisé par un but partagé par l'ensemble des participants. Les systèmes de dialogue en langage naturel, conçu pour la réalisation de tâches, sont des systèmes qui émulent ce type de dialogue pour permettre à un utilisateur de contrôler un système informatique au moyen d'interactions en langage naturel afin d'accomplir des tâches sur un environnement.

Un large spectre des capacités cognitives humaines sont impliquées lors d'un dialogue, son émulation est donc une tâche difficile qui comprend de nombreux constituants. S'il s'agit d'un dialogue oral, l'analyse du signal en fait partie, aussi bien que le traitement du langage utilisé pour la communication, les capacités des participants à raisonner, argumenter, décider, ou bien les différentes activités coopératives qu'engendrent la réalisation d'une tâche à plusieurs. Les principaux points à prendre en considération sont les suivants :

**Le langage** Le langage qui sert la communication.

**L'acte de langage** L'acte de langage est la plus petite unité d'un dialogue qui modélise une prise de parole d'un participant comme une action sur

les autres participants et qui est définie par une performative comme « *request* », « *query* » ou « *inform* ».

**Les tours de parole** Un dialogue est défini comme une succession de prises de parole par les participants. Elles doivent être organisées pour que la communication soit effective.

**La tâche** Les activités coopératives dans lesquelles les participants s'engagent comme la prise de connaissance du domaine, l'établissement du contexte de la tâche, la création d'un plan partagé.

À titre d'exemple, une retranscription d'un dialogue entre deux humains issu du domaine du *cloud computing* est donnée ci-dessous. L'objectif est d'*installer une nouvelle machine virtuelle nommée TX5A* :

1. Domaine

Les deux participants ne possèdent pas les mêmes niveaux de connaissances, l'objectif est d'obtenir une représentation partagée du domaine (concept, objet, relation) entre les participants.

**Dialogue 3.1.1.**

**Apprenti** Qu'est-ce que TX5A ?

**Expert** TX5A est une machine virtuelle de type Y.

**Dialogue 3.1.2.**

**Apprenti** Donc TX5A peut être installée sur le serveur S1 ?

**Expert** Oui.

2. Co-construction de contextes

Les deux participants partagent l'information qu'ils perçoivent de l'environnement.

**Dialogue 3.1.3.**

**Expert** Installons TX5A sur le serveur S2.

**Apprenti** Est-ce que S2 possède un stockage suffisant ?

**Expert** Attends, je regarde.

3. Co-construction de plans

La construction du plan implique une prise de décision partagée.

**Dialogue 3.1.4.**

**Expert** Le serveur S2 est disponible et dispose d'une grande capacité de stockage.

**Apprenti** Sinon, le serveur S1 dispose d'une plus grande capacité réseau.

**Expert** Oui mais TX5A requière seulement une bonne capacité de stockage.

**Dialogue 3.1.5.**

**Apprenti** J'aurai besoin du serveur S2 pour d'autres opérations.

**Expert** Je mettrai le serveur S3 à ta disposition pour les autres opérations.

**Apprenti** Très bien, j'installe TX5A sur le serveur S2.

Ainsi comme l'illustre cet exemple, remplacer l'« apprenti » par un agent artificiel est une tâche ambitieuse. Si certains systèmes s'approchent de cet objectif (comme illustré par la suite avec le système PLOW) le coût ingénierique pour transposer ces systèmes d'un domaine à l'autre reste très élevé. Ceci s'explique par l'utilisation de nombreuses techniques d'IA symboliques (base de connaissance logique, moteur de planification, etc). Une stratégie consiste à simplifier certains constituant du dialogue (le langage, les tours de parole, la répartition des rôles, etc) pour offrir des systèmes moins « intelligents » mais plus maintenables, comme l'illustre des système tel que Siri, Google Now, ou Cortana (Canbek et Mutlu 2016). Ainsi par exemple, le protocole de communication peut être prédéfini. Ci-dessous, l'utilisateur expose un but et l'agent exécute l'action et demande l'information manquante le cas échéant :

**Dialogue 3.1.6.**

**Utilisateur** Envoi un e-mail à Peter Dominey.

**Agent** Quel est le sujet ?

**Utilisateur** Réunion Aujourd'hui

**Agent** Quel est le message ?

**Utilisateur** La réunion est décalée à 14h.

**Agent** Le message a été envoyé.

Voir encore plus simple lorsque l'utilisateur contrôle toutes les transitions dans le dialogue :

**Dialogue 3.1.7.**

**Utilisateur** Envoyer un e-mail.

**Agent** Ok

**Utilisateur** Mettre le sujet : Réunion Aujourd'hui

**Agent** C'est bon.

**Utilisateur** Mettre le message : La réunion est décalée à 14h.

**Agent** C'est bon.

**Utilisateur** Envoyer le message.

**Agent** C'est bon.

## 3.2 Les principaux composants d'un système de dialogue

Les principaux composants fonctionnels mis en œuvre dans un système de gestion de dialogue sont les suivants :

**Reconnaissance vocale** Cette fonctionnalité transforme le signal vocal en texte.

**Compréhension du langage** Ce module convertit le texte brut en une représentation sémantique qui puisse être traitée par le système.

**Gestionnaire de dialogue** Le gestionnaire de dialogue maintient le contexte du dialogue et décide quel sera le prochain acte dialogique. Il est souvent couplé ou confondu avec le module qui gère la tâche à réaliser.

**Génération du langage** Cette fonctionnalité convertit une action communicative sous une forme compréhensible pour un humain.

**Synthèse vocale** Cette fonctionnalité transforme l'utterance en un signal vocal.

Les architectures en *pipe line* sont les plus simples à mettre en œuvre. Elles consistent à abouter chacune de ces briques fonctionnelles dans l'ordre, de la reconnaissance vocale à la synthèse vocale. Cependant, chacun de ces modules pourrait bénéficier d'informations fournies par les modules qui les succèdent. C'est le cas par exemple pour la reconnaissance vocale qui peut être facilitée par un lexique contextuel dérivé du module de compréhension du langage. Ce dernier pourrait à son tour lever des ambiguïtés syntaxiques à partir d'information sémantique du contexte global du dialogue. Ainsi, pour répondre au besoin d'analyser les signaux d'entrées de façon incrémentale et gérer le passage d'informations en amont et en aval des architectures en *blackboard* sont mises en place. L'architecture Jaspis, en Figure 3.1, montre un exemple de ce type d'architecture dans lequel l'information est centralisée et où chacun des modules réagit en fonction des écritures faites dans le module *Information Manager*.

Comme expliqué dans la section précédente, l'émulation de dialogue est une tâche complexe ce qui conduit à une variété de choix de simplification. Ainsi, il n'existe pas une bonne manière de réaliser le module de compréhension du langage ou une bonne manière de concevoir un gestionnaire de dialogue. Le gestionnaire de dialogue OpenDial par exemple, illustré en Figure 3.2, permet de s'adapter au score de confiance du résultat d'un traitement d'une utterance. Le module de compréhension du langage doit alors être en mesure d'établir ce score lors de son analyse. Ainsi, bien que la modularisation des systèmes de dialogue soit balisée, la diversité des méthodes et l'intrication des modules limite la réutilisation.

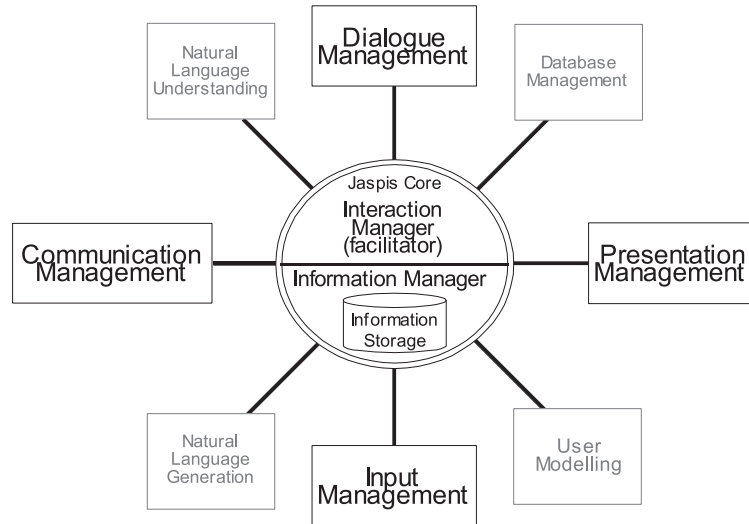


FIGURE 3.1 – Architecture du système de dialogue Jaspis

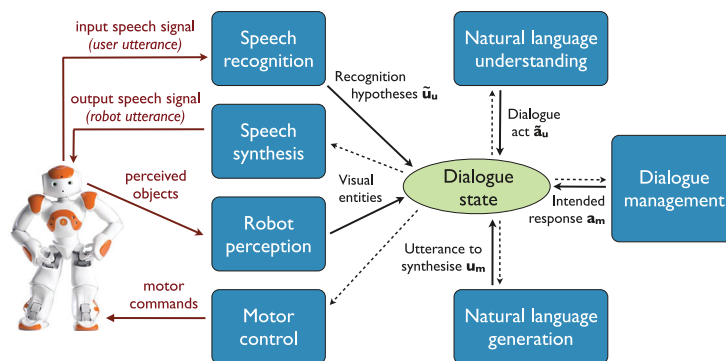


FIGURE 3.2 – Architecture du système de dialogue OpenDial

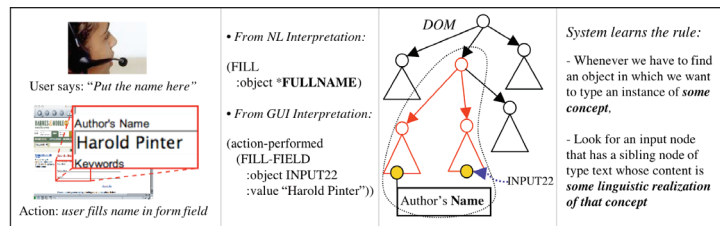


FIGURE 3.3 – Le système Plow

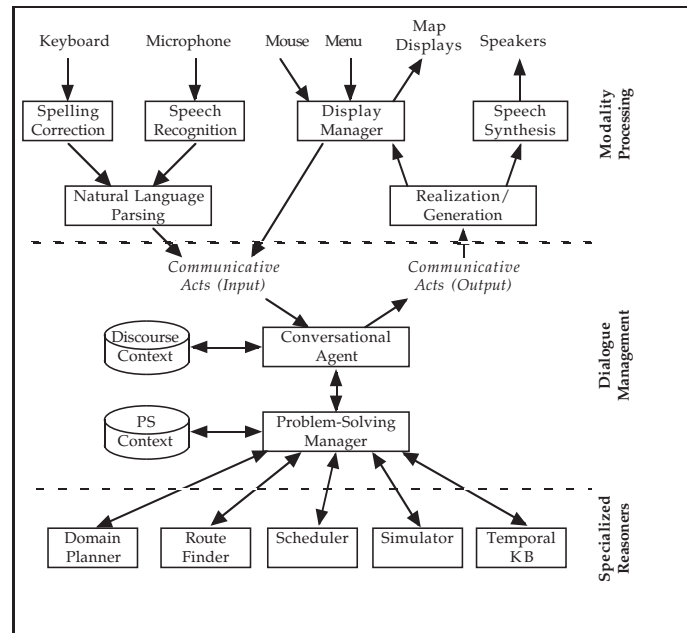


FIGURE 3.4 – Le système TRIPS

### 3.3 Application pour les agents instructibles

Dans cette section nous présentons deux systèmes de dialogue dont la particularité est qu'ils peuvent être instruits par l'utilisateur en langage naturel.

#### 3.3.1 Plow

Le système PLOW développé par James Allen (Allen et al. 2007), en Figure 3.3, permet à un utilisateur d'apprendre une nouvelle tâche au système par une démonstration unique. Il est construit comme un système d'automatisation de tâches dans un navigateur web. Il analyse la page web, et par ce moyen accède aux représentations html, aux urls des pages, et aux événements. Un système de dialogue gère l'interaction entre la personne et une représentation de la tâche à accomplir. En joignant le geste à la parole, l'utilisateur peut expliciter ce qu'elle réalise. PLOW en montre sa compréhension en surlignant les résultats de l'action, ou en généralisant l'action de l'utilisateur à plusieurs pages web. Pour cela, PLOW maintient un contexte de dialogue, un

Word	Syntactic Category	Logical Form
set	((S/PP_StringV)/MutableField)	(lambda x y (setFieldFromString x y))
to	PP_StringV/StringV	(lambda x x)
subject	FieldName	subject
send	S/InstanceName	(lambda x (send x))
email	InstanceName	email
set	((S/PP_FieldVal)/MutableField)	(lambda x y (setFieldFromFieldVal x y))
to	PP_FieldVal/FieldVal	(lambda x x)
and	(S/S)\S	(lambda x y (doSeq x y))
's	((Field\InstanceName)/FieldName)	(lambda x y (getFieldByInstanceNameAndFieldName y x))
forward	(S/InstanceName)	(lambda x (doSeq (doSeq (doSeq (doSeq (createInstanceByConceptName outgoingemail) (setFieldFromFieldVal (getMutableFieldByFieldName subject) (evalField (getFieldByInstanceNameAndFieldName email subject)))) (setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField (getFieldByInstanceNameAndFieldName email body)))) (setFieldFromFieldVal (getMutableFieldByFieldName recipient) (evalField (getFieldByInstanceNameAndFieldName email)))) (sendEmail)))

FIGURE 3.5 – Ensemble des commandes du système LIA

Text Command	Logical Form
set the subject to time to go	(setFieldFromString (getMutableFieldByFieldName subject) (stringValue "time to go"))
send the email	(send email)
set body to email's body and send email	(doSeq (setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField (getFieldByInstanceNameAndFieldName email body)))) (send email))
add length as a field in table	(addFieldToConcept table (stringNoun "length"))
forward to charlie	(doSeq (doSeq (doSeq (doSeq (createInstanceByConceptName outgoingemail) (setFieldFromFieldVal (getMutableFieldByFieldName subject) (evalField (getFieldByInstanceNameAndFieldName email subject)))) (setFieldFromFieldVal (getMutableFieldByFieldName body) (evalField (getFieldByInstanceNameAndFieldName email body)))) (setFieldFromFieldVal (getMutableFieldByFieldName recipient) (evalField (getFieldByInstanceNameAndFieldName email)))) (sendEmail)))

FIGURE 3.6 – Ensemble d'entraînement pour le système LIA

contexte de démonstration (par l'exemple), et un contexte de représentation de l'enchaînement des tâches apprises à réaliser. Les tâches pouvant inclure des boucles et des branchements conditionnelles.

Le système de dialogue repose sur le système TRIPS (Ferguson, Allen et al. 1998) (Figure 3.4). Ce système fournit un ensemble de capacités évoluées permettant de gérer des dialogues riches, ce qui inclut une base de connaissance qui gère la temporalité des événements, un planificateur ou un gestionnaire de dialogue qui gère des dialogues de type délibératif. L'ensemble de ces modules nécessite une analyse fine des utterances de l'utilisateur. Cette analyse repose sur la logique épisodique (Schubert et Hwang 2000) dont EPILOG en est l'implémentation (Schaeffer et al. 1993). Cette logique a la particularité d'avoir une syntaxe proche de celle du langage naturel. Afin de rendre cette logique calculable, des modules spécialisés sont développés pour gérer certains types de connaissance, comme les nombres par exemple, et qui rendent l'implémentation d'un modèle complexe. Chaque module repose sur des techniques indépendantes du domaine, cependant pour adapter le système global à un domaine spécifique, chaque module doit être adapté de façon consistante ce qui implique un travail conséquent et une expertise élevée.

### 3.3.2 Lia

LIA, développé par Azaria (Azaria, Krishnamurthy et Mitchell 2016), est un agent qui peut également être instruit par l'utilisateur en langage naturel. Dans ce système il n'y a pas de démonstration graphique, les utilisateurs peuvent apprendre à exécuter de nouvelles séquences d'actions pour obtenir de

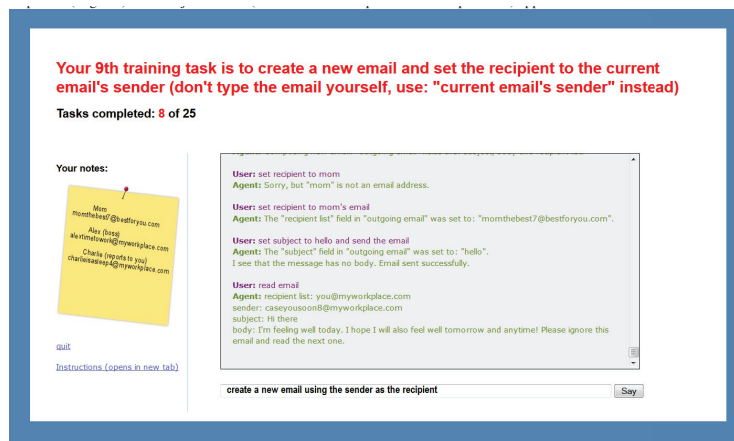


FIGURE 3.7 – Interface du système LIA

nouvelles commandes en utilisant uniquement le langage naturel (Azaria, Krishnamurthy et Mitchell 2016). LIA est composé de deux parties : un analyseur sémantique qui assigne une représentation sémantique à chaque commande en langage naturel et un *back-end* qui exécute ces commandes. Préalablement, le système contient un ensemble de fonctions primitives pour un domaine particulier (comme les e-mails) ainsi que leur forme logique correspondante (Figure 3.5). L'association du langage naturel aux formes logiques est pré-entraînée (Figure 3.6), puis à partir de l'interface du système (Figure 3.7), l'utilisateur peut enseigner de nouvelles actions ou de nouveaux concepts.

Afin d'accommoder le système à un domaine spécifique, l'analyseur sémantique doit être entraîné sur des liaisons préexistantes entre le lexique du langage naturel et les fonctions primitives du domaine. Ainsi, sans possibilité de démonstration, LIA impose que le concepteur pré-programme l'ensemble des fonctions du *back-end*, comme par exemple *sendEmail*.



# Chapitre 4

## Approche cognitive du langage et mise en œuvre en robotique

Ce chapitre introduit les concepts clefs d’une approche cognitive constructiviste de l’apprentissage du langage, en quoi cette approche diffère de l’approche générative et comment elle pourrait être exploitée dans la réalisation d’interfaces Homme – Machine dialogiques. En effet, cette approche a déjà été exploitée par des systèmes robotiques où le principal intérêt était de créer des robots qui soient en mesure de communiquer et d’être contrôlés par des humains dans un monde ouvert. Ces derniers partagent un environnement commun, ce qui fait de la robotique un domaine d’application propice pour mener ce genre d’expérimentations. Ainsi, nous présentons le fonctionnement de ces systèmes et le transfert de ces techniques pour le monde numérique.

### 4.1 Contexte

Du point de vue de l’acquisition du langage, l’approche générative par opposition à l’approche constructiviste peut être caractérisée en termes de ce qui est inné par opposition à ce qui est appris.

L’approche générative soutient qu’une grammaire universelle (ou un système symbolique évolué) est innée, et que l’acquisition du langage est un processus de paramétrage de ce système (Chomsky 1995). Ceci est évoqué comme nécessaire en raison de l’insolubilité de l’apprentissage des langues. L’argument de la pauvreté du stimulus soutient que le problème de l’apprentissage des langues est si sous-déterminé qu’il nécessite cette lourde pré-programmation de la grammaire universelle (Pullum et Scholz 2002).

L’approche constructiviste soutient qu’à partir de l’expérience sensorimotrice et grâce à des processus humains hautement évolués, comme de l’attention jointe, l’intractabilité et l’argument de la pauvreté du stimulus sont éliminés (Tomasello 2003a ; Dominey et Dodane 2004). Cette approche implique d’expliquer comment le système symbolique s’ancre dans le système de représentation sensorimoteur et comment ce dernier évolue pour maîtriser le langage. Pour Mandler (Mandler 1988, 1992, 2005), l’utilisation de symboles s’effectue dès les premiers mois qui suivent la naissance. Elle décrit un proces-

sus, appelé *Perceptual Meaning Analysis* (PMA), comme le premier moyen permettant d'apprendre des schémas représentant des concepts primitifs comme *motion*, *containment*, *path*, etc. Ces schémas sont définis comme étant une représentation symbolique simplifiée (contenant moins d'information) des données présentes dans le système perceptuel. Ces schémas servent ensuite de base pour la création d'images et plus tard dans l'ancrage du langage. Par la suite, l'apprentissage du langage chez les enfants commence autour de modèles fixes, des formes précoces de constructions grammaticales, qu'ils réutilisent, et commencent graduellement à étendre et à généraliser. Au cours des premières phases de ce modèle d'apprentissage et de réutilisation, le langage est déjà très productif (Tomasello et al. 1997).

Du point de vue du développement de systèmes, l'approche constructiviste signifie que nous pouvons construire une infrastructure légère tout en maintenant la possibilité d'offrir un service appréciable dès les premiers instants du cycle de vie du logiciel. Dans le domaine de la robotique, des formes d'apprentissage du langage basé sur l'expérience perceptuelle physique du robot ont été mises en œuvre (Dominey et Boucher 2005a), ainsi que pour l'exécution d'actions et l'apprentissage de plans partagés (Hinaut et al. 2014). Dans cette thèse, cette approche est étendue au développement d'assistants virtuels interagissant avec des services numériques. Nous montrons que l'apprentissage des constructions grammaticales (et la composition d'actions), à partir d'une expérience partagée avec l'utilisateur permet au système de s'adapter aux besoins de l'utilisateur à travers l'apprentissage.

## 4.2 Développement du langage

L'apprentissage du langage basée sur l'usage est une approche sociale-pragmatique de l'acquisition du langage dans laquelle les enfants apprennent des structures linguistiques en comprenant les intentions et en trouvant des modèles dans leurs interactions discursives avec les autres (Tomasello 2003a). Dès le début de la vie, les enfants et les adultes interagissent autour d'événements répétitifs comme les repas, changer les couches, aller au lit, etc. et qui sont accompagnés par le langage des adultes (Clark 2003). Ceci fournit un espace partagé riche où la langue peut enrichir le sens de ces événements (Nomikou et al. 2016). Dans le concept d'acquisition du langage basée sur l'usage, les premiers mots de l'enfant permettent de se référer à des personnes et des objets qui sont au centre de la vie quotidienne (Tomasello 2003a ; Clark 2003). Après cette forme simple d'annotation, les premières constructions grammaticales sont fixées autour de verbes spécifiques et d'actions spécifiques qui tendent à être répétées et ritualisées dans l'environnement social des nourrissons (Tomasello 2000b, 2003a ; Lieven et al. 2003). Les constructions deviennent alors plus abstraites et généralisées à mesure que le monde de l'enfant s'enrichit (Goldberg 1995 ; Tomasello 2003a).

La distinction essentielle dans ce contexte est que le langage est considéré comme étant un inventaire structuré de liens entre la formes des énoncés et leur signification, appelés constructions grammaticales (Goldberg 1995). D'un

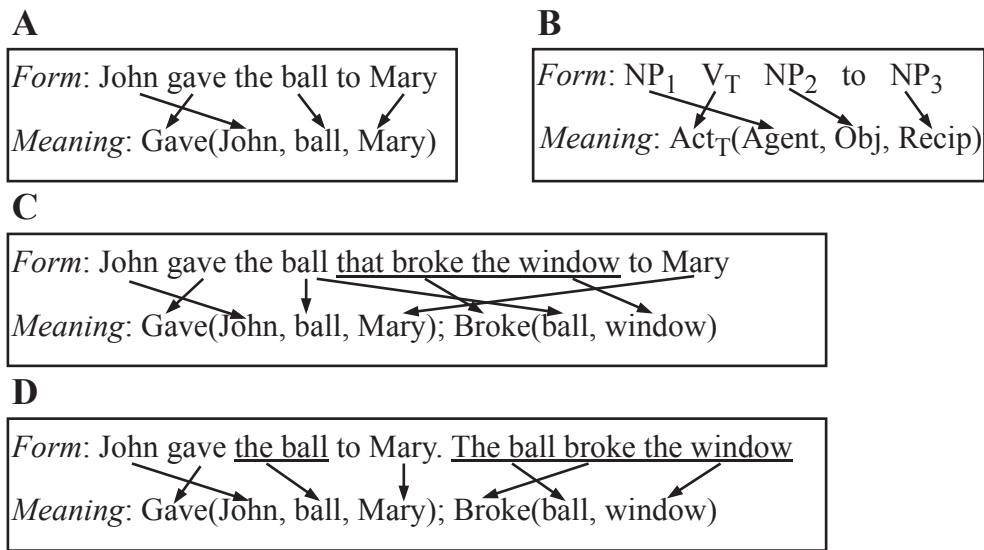


FIGURE 4.1 – Exemple de constructions grammaticales. (A) Exemple de lien forme-sens. (B) Représentation généralisée d’une construction. (C) Lien forme-sens avec une phrase relative. (D) Lien forme-sens avec une phrase relative extraite.

côté, des mots simples et des « holophrases » fixes comme « *Gimme that* » sont traités comme des éléments « holistiques » non analysés (Tomasello 2003a). A l’autre extrême se trouvent des constructions complexes qui permettent l’utilisation de phrases comme celle-ci. Entre les deux, des constructions qui permettent l’expression d’événements spatio-temporels du quotidien qui sont à la base de l’expérience humaine, y compris les constructions transitives actives (ex. John a pris la voiture) et ditransitives (ex. Mary a donné une nouvelle recette à ma mère) (Goldberg 1995). La Figure 4.1 illustre la nature des constructions telles que les liens de la forme de phrase à une représentation sémantique est explicite.

Dans ce contexte, l’approche constructiviste soutient que le nourrisson commence l’acquisition du langage par l’apprentissage de constructions très simples couvert par des ressources calculatoires modestes (Clark 2003 ; Tomasello 2003a) et dont le développement progressif permet ensuite d’en complexifier le traitement. Cela contraste avec l’« hypothèse de continuité » issue de la philosophie de la grammaire générative, dans laquelle toute la complexité syntaxique est disponible sous la forme d’une grammaire universelle et est utilisée au début de l’apprentissage des langues (voir (Tomasello 2000b) et les commentaires concernant le débat sur l’hypothèse de continuité). Dans le contexte génératif, la grammaire universelle est là dès le début, et le défi est de comprendre comment elle s’est formée. Dans l’approche constructiviste, le traitement initial est simple et devient de plus en plus complexe, et le défi est d’expliquer les mécanismes qui permettent la pleine productivité et la composition (composer de nouvelles constructions à partir de constructions existantes). Cette question est partiellement abordée dans la recherche actuelle.

Les grammaires de construction sont une bonne approche pour l’approche constructiviste de l’acquisition du langage. Dans ce framework, la notion de construction devenant de plus en plus abstraites est largement développée par Tomasello dans le chapitre 5 – *Abstract Syntactic Constructions, in Constructing a Language : A Usage-Based Theory of Language Acquisition* (Tomasello 2003a). La structure argumentative des constructions est d’abord directement extraite du modèle perceptuel (Goldberg 2003). Ensuite, par des processus d’abstraction, de recherche de modèle et d’analogie ces constructions deviennent plus généralisées. Gaspers et al (Gaspers et al. 2017) explique comment l’abstraction prend place. Ce processus est basé sur un mécanisme de *bootstrapping* qui repose sur la découverte de régularités pour segmenter la parole en unités (en « mots ») à partir de régularités précédemment apprises. Basé sur une segmentation initiale, le modèle induit une nouvelle construction grammaticale qui à son tour, par une analyse « *top-down* », guide la segmentation de nouvelle phrase en mots. Un modèle est proposé qui en premier apprend des « *holophrases* » idiomatique par un lien direct vers une représentation sémantique, puis où certains mots deviennent des arguments en formant des constructions de type « *iles verbales* » (Tomasello 2000a) et enfin où tous les éléments de classe ouverte deviennent des arguments abstraits au sein d’une construction (Clark et Kelly 2006)

### 4.3 Application en robotique

Cangelosi et al (Cangelosi et Riga 2006 ; Stramandinoli, Marocco et Cangelosi 2017 ; Tikhanoff, Cangelosi et Metta 2011) ont développé des simulations et des expériences robotiques qui montrent comment le sens des mots peut être ancré dans la perception pour simuler l’apprentissage du langage. Leurs travaux les plus récents se sont concentrés sur l’étude des structures grammaticales dans le cas des couples verbe-nom pour la réalisation d’actions orientées vers un but (Stramandinoli, Marocco et Cangelosi 2017). L’apprentissage de constructions grammaticales a été envisagé aussi bien du point de vue de l’évolution du langage (Steels 2005) que d’une perspective développementale ou d’apprentissage (Dominey et Boucher 2005a ; Bergen, Chang et Narayan 2004 ; Dominey et Boucher 2005b).

Les premières expérimentations menées par Dominey et Boucher dans (Dominey et Boucher 2005a) ont été de créer un système perceptuel qui apprend à cartographier des représentations d’événements perçus tels que *push(moon, cylinder)* sur des phrases telles que « *The moon pushed the cylinder* », ou « *The cylinder was pushed by the moon* ». Ils ont ensuite étendu cette approche pour inclure à la fois la description des événements perçus et l’exécution des commandes vocales dans le système Cooperative Human Robot Interaction System (Chris).

L’architecture Chris est une architecture cognitive indépendante du robot développée sur la base d’une analyse du développement cognitif humain. Ses caractéristiques principales consistent en la capacité d’extraire le sens de la perception, d’apprendre à composer de nouvelles actions à partir d’actions

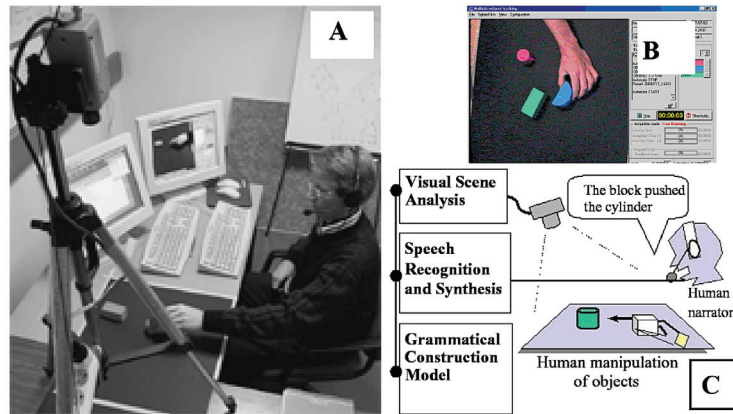


FIGURE 4.2 – Interaction Homme – Robot. A. Un utilisateur humain interagit avec des cubes, narre des événements et écoute le robot retranscrire une narration d'événements. B. Prise de vue d'une scène tel que perçue par la caméra CCD. C. Architecture de la plateforme : 3 modules distincts pour le traitement de la vision, de la parole et du dialogue. Les constructions grammaticales sont apprises et utilisées pour la compréhension et la production de phrases.

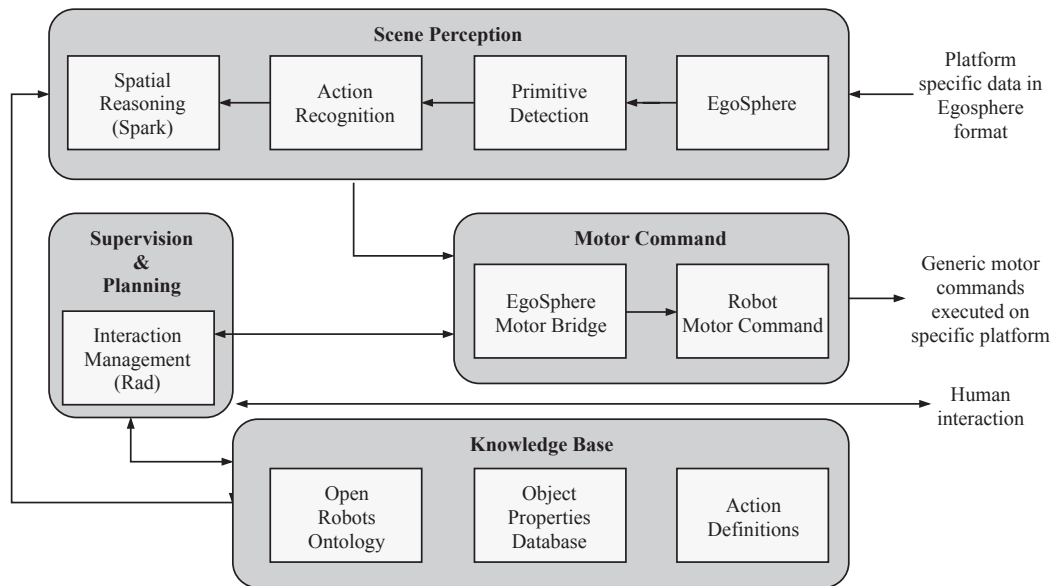


FIGURE 4.3 – L'architecture Chris : Les données sensorielles fournissant des coordonnées 3D en temps réel sont encodées dans le module *EgoSphere*. Les primitives qui incluent par exemple « *contact* » et « *motion* » sont détectées et utilisées pour reconnaître des actions comme « *A gave B to C* ». Ces actions peuvent être utilisées comme une entrée du module *Motor Command* pour l'imitation d'actions. Le module *Knowledge Base* encode les propriétés des objets et apprend les définitions des actions et des plans partagés. L'interaction avec l'humain est accompli par le module *Supervision and Planning*.

Figure reproduite à partir de (Stephane Lallée et al. 2012)

primitives et apprises, et d'apprendre et d'exécuter en coopération des plans ou des procédures partagées avec un être humain par le biais d'une interaction en langage parlé (Stephane Lallée et al. 2012). Une architecture simplifiée est montrée dans Figure 4.3. L'extraction du sens de la perception est réalisée à l'aide d'un ensemble de primitives incluant le « mouvement » et le « contact », qui permettent la reconnaissance d'actions comme « prendre » et « donner » dans le module *Scene Perception*.

De même, les primitives motrices telles que saisir, déplacer et relâcher sont définies et utilisées pour construire des actions plus complexes dans le module *Motor Command*. Ensuite, dans l'ancrage d'une phrase comme « *put the block1 on the block2* » dans le sens, il y a une transition de la phrase dans une représentation schématisée du sens, *put(block1, block2)*. Cette représentation de l'action est ensuite décomposée en commandes motrices primitives comme *grasp*, *grave*, *release* et en primitives perceptuelles comme *object1*, *object2* qui sont codées dans la base de connaissances. Les capacités de traitement du langage et d'apprentissage de la planification sont fournies par le module *Planning and Supervision* et permettent à un humain, soit par des descriptions en langage naturel, soit par des démonstrations physiques, de spécifier à l'exécution comment lier une unité linguistique telle que *block1* à un élément perceptuel tel que *object1* ainsi que la décomposition de *put(object, location)* dans la séquence *grasp(object)*, *move(location)*, *release(object)*.

Pour les nouvelles actions (qui ne sont pas encore définies dans la *Knowledge Base*) le système utilise l'ensemble des observations primitives à partir d'une segmentation temporelle pour créer un schéma générique d'actions primitives. Ainsi, un nouveau prédicat *cover(x,y)* est appris comme un schéma de primitives : « *moving(x)*, *contact(x, y)*, *!visible(y)*, *!moving(x)* » qui est généré par le système perceptuel. C'est ensuite ce schéma sur lequel la phrase « *Cover y with x* » peut être associée (Stéphane Lallée et al. 2010). Ces actions sont également associées avec des états. En conséquence, après l'exécution de l'action *cover(x,y)*, l'état résultant est « *contact(x,y)*, *visible(x)*, *on(x,y)* ». Par le même mécanisme, Lallée et al (Stéphane Lallée et al. 2010) ont démontré l'utilisation de 4 autres prédicats, incluant *uncover*, *take*, *give*, et *put*. Ces actions peuvent ensuite être assemblées dans des constructions de plus haut niveau qui sont appelées plan partagé, permettant ainsi à l'humain et au robot d'atteindre l'état désiré. Ce travail fournit un framework pour démontrer comment des structures prédicats-arguments comme *give*, *put*, etc peuvent émerger.

## 4.4 Transfert vers le monde numérique

Comme le démontre l'architecture Chris, un robot peut apprendre à parler d'une entité perceptuelle, comme par exemple, un humain ramassant un objet (Stéphane Lallée et al. 2010), et comment participer à un jeu coopératif comme la découverte d'un jouet (Stéphane Lallée et al. 2013). De la même manière, un assistant virtuel pourrait apprendre à reconnaître et à parler d'un courriel et à envoyer un courriel en coopération avec l'humain.

Dans ce qui suit, nous montrons comment adapter les techniques employées



dans le domaine de la robotique pour construire un assistant virtuel. Nous nous appuyons sur les trois aspects majeurs suivants :

1. **L'existence de primitives symboliques dérivées du système perceptuel** : Nous avons adapté l'ensemble des primitives symboliques à un monde numérique pour être en mesure d'observer les actions de l'humain sur un GUI. Nous avons donc développé et exploité des primitives perceptuelles comme remplir un champs texte, cliquer, pointer, sélectionner un texte, etc.
2. **L'ancrage du langage dans ces primitives** : Nous montrons comment le langage est lié à ces primitives pour composer de nouvelles actions grâce à des démonstrations de l'utilisateur.
3. **L'apprentissage du langage et son utilisation lors d'activité coopérative avec un humain** : Nous montrons comment un utilisateur peut enseigner au système à interpréter des phrases compositionnelles en utilisant des instructions en langage naturel et comment le système peut généraliser des procédures par la création d'arguments.

# Chapitre 5

## Agent virtuel apprenant

Ce chapitre présente la plateforme agent développée au cours de la thèse. Après une explication générale du principe de fonctionnement du système, chaque module est décrit un à un, afin de détailler leur rôle, leur structuration des données et leur relation avec les autres modules de la plateforme. L'interaction globale entre les modules et les principaux algorithmes mis en œuvre sont ensuite expliqués autour d'un scénario d'usage spécifique. Pour finir une liste exhaustive de cas d'utilisation, avec des exemples de dialogues, est donnée afin d'illustrer le comportement des agents dans des conditions réelles d'utilisation.

### 5.1 Principe de fonctionnement

L'architecture de l'agent virtuel, montrée dans la Figure 5.1, regroupe les composants suivants :

1. Le module *Environment* qui transforme les services numériques dans un format que l'agent peut percevoir et dans lequel il peut agir.
2. Le module *SensorProcess* qui convertit les percepts dans des représentations sémantiques grâce au module *ConstructionMemory*.
3. Le module *InterpreterProcess* qui gère les interactions avec l'utilisateur dans le but de coopérativement exécuter des procédures ou les apprendre. Le module *ContextMemory* est utilisé comme une mémoire de travail pour la tâche en cours d'exécution et le module *EpisodicMemory* enregistre tous les événements qui occurred dans la plateforme, utilisé par la suite par l'algorithme *one-shot learning* pour encoder de nouvelles constructions.
4. Le module *MotorProcess* qui convertit à son tour des représentations sémantiques vers des actions exécutables dans l'environnement.

Dans le but d'illustrer le fonctionnement de l'agent nous donnons dans un premier temps un exemple de scénario. Cela sera expliqué plus en détail dans les sections suivantes :

Un utilisateur demande à envoyer un e-mail « *send an email to peter.dominey@inserm.fr about the meeting and say that I will be*



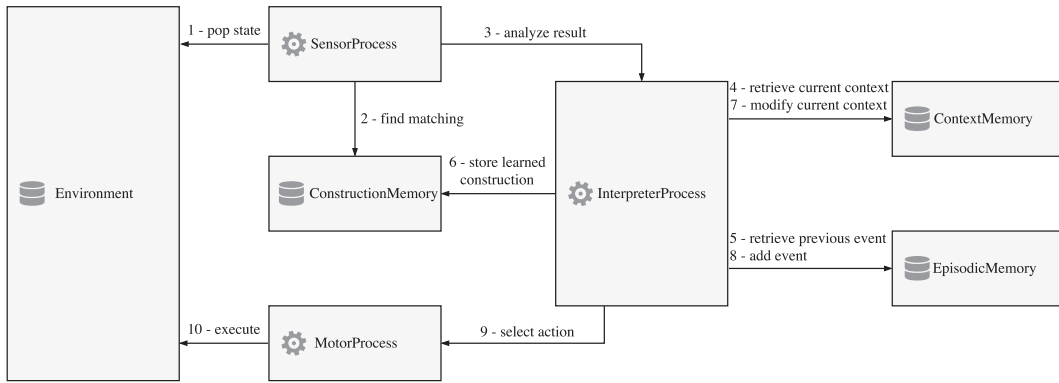


FIGURE 5.1 – Architecture générale de l’agent virtuel : Pendant une interaction, le module *SensorProcess* monitor l’état courant de l’environnement et cherche à faire correspondre les phrases de l’utilisateur avec les constructions présentes dans le module *ConstructionMemory*. Si une correspondance est trouvée, le module *InterpreterProcess* commence à sélectionner les actions contenues dans la construction pour les exécuter via le module *MotorProcess*. Dans le cas contraire, le système demande une démonstration ou une explication à l’utilisateur et enregistre les événements dans le module *EpisodicMemory*. À la fin de la démonstration, ces événements sont consolidés dans le module *ConstructionMemory* sous la forme d’une nouvelle construction qui lie la demande (forme) avec la démonstration (sens).

*early* ». Si l’agent virtuel ne connaît pas cette tâche, le comportement d’apprentissage est activé. Ensuite, l’utilisateur peut décrire les étapes qui composent cette tâche en utilisant soit (a) le langage naturel (en utilisant les constructions précédemment apprises), comme « *create a draft* », « *set the subject* », « *write the content* », etc. ou (b) en effectuant une démonstration sur une interface graphique que l’agent peut observer. Le comportement d’apprentissage enregistre tous les événements perceptuels venant du module *SensorProcess* ainsi que les phrases de l’utilisateur dans le module *EpisodicMemory*. À la fin de l’apprentissage, un algorithme consolide les connaissances extraites du module *EpisodicMemory* et enregistre les nouvelles constructions correspondantes dans le module *ConstructionMemory*. La nouvelle construction est une association entre la forme de l’énoncé et une procédure d’exécution. Quand l’utilisateur demande à nouveau l’exécution de cette action « *send an email to clement.delgrange@inserm.fr about the software and say that it functions perfectly* », l’agent virtuel trouvera une correspondance avec une construction apprise, extraira les arguments contenus dans la phrase de l’utilisateur etinstanciera la procédure correspondante. Le comportement d’exécution est ensuite activé. Il conduit l’interaction avec l’utilisateur et exécute les actions primitives via le module *MotorProcess* dans le but d’accomplir la procédure. Nous décrivons plus en détails les modèles des

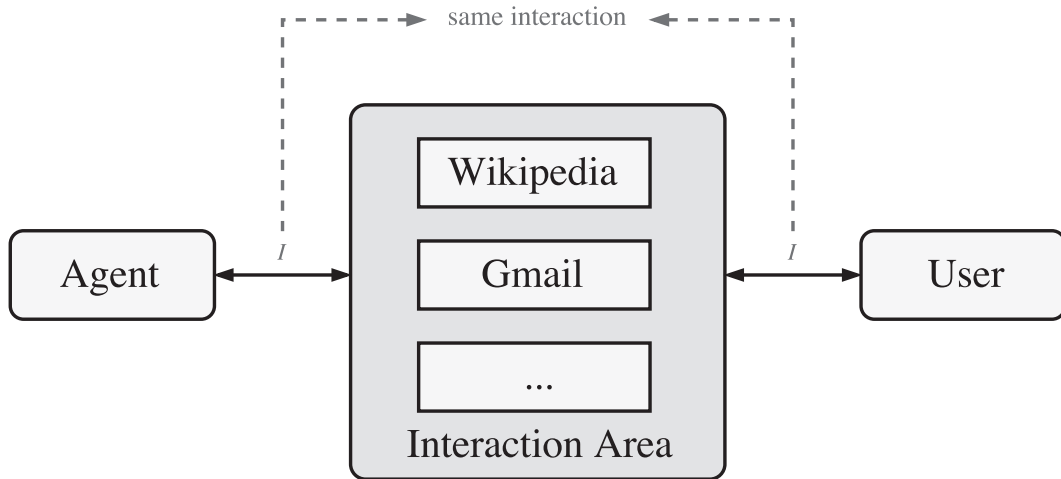


FIGURE 5.2 – Environnement partagé : L’espace d’interaction permet à l’utilisateur d’effectuer des actions qui peuvent être observées par l’agent pendant un apprentissage par démonstration et qu’il peut par la suite exécuter par lui-même.

modules *SensorProcess* et *MotorProcess*, l’algorithme d’extraction de sens et l’algorithme *one-shot learning* dans les prochaines sections.

## 5.2 Environnement partagé

Les première constructions grammaticales sont ancrées dans l’expérience sensorimotrices du système. Ce qui implique que les fonctions offertes par les modules sensoriels et moteurs possèdent une représentation préétablie au sein du système analogue aux primitives perceptuelles dans le domaine des robots.

Dans l’objectif d’apprendre les premières représentation du sens et leur procédure d’interprétation par démonstration (indépendamment d’expression en langage naturel), nous devons définir un modèle pour la perception que l’agent pourra acquérir par observation. Par conséquent, l’agent virtuel doit être en mesure d’observer les actions de l’utilisateur, par exemple remplir le *sujet* d’un e-mail, et facilement transférer cette observation vers ces propres primitives moteurs. La Figure 5.2 illustre l’environnement partagé entre un utilisateur et l’agent virtuel comme étant une espace d’observation et une représentation partagée.

La représentation de l’environnement partagé est une simplification des interactions possibles avec une interface graphique. Pour cela, nous avons défini un ensemble d’actions de base qu’un utilisateur peut effectuer avec une interface graphique qui est la possibilité de lire et d’écrire des informations textuelles dans des champs textes et de déclencher une transition au sein du système (par exemple, cliquer sur un bouton). À partir de l’espace partagé, l’agent peut observer et agir sur cette représentation indépendamment de ses compétences langagières. Nous définissons les éléments primitifs de la percep-

tion comme étant des *frames*, des *data*, des *fields* et des *actions*.

**Frame** Une *frame* est utilisée pour structurer l'information fournie par le système et pour rassembler les informations reliées. Elle est composée d'un label et d'un ensemble d'éléments enfants : *frames*, *data*, *fields* et *actions*. Elle peut être vue comme une représentation en arbre de l'information, par exemple : Tout les paragraphes dans un article *Wikipedia*.

**Data** Une *data* donne l'état courant d'une ressource. Elle est composée d'un label, d'une liste de valeurs qui représente l'état de la ressource et d'une *frame* parent. Par exemple : Le contenu d'un paragraphe dans un article *Wikipedia*.

**Field** Un *field* décrit et maintient une information textuelle fournie par l'utilisateur. Il est composé d'un label, d'une liste de valeurs et d'une *frame* parent. Par exemple : Le champs texte de la boite de recherche sur une page *Wikipedia*.

**Action** Une *action* définit une transition possible dans le système. Elle est composée d'un label et d'une *frame* parent. Par exemple : Cliquer sur le bouton « recherche » d'une page *Wikipedia*.

En complément de ces primitives perceptuelles, nous définissons un ensemble de primitives moteurs, ce qui inclut :

**Fill** Utilisé pour remplir un champs texte dans l'environnement.

**Execute** Utilisé pour déclencher une action, par exemple, cliquer sur un bouton.

**Speak** Utilisé pour communiquer en langage naturel à travers une messagerie.

L'espace d'observation partagé permet à l'agent d'observer les actions de l'utilisateur. Nous avons développé un outil de synchronisation qui maintient la consistance entre le module *Environment* et l'interface réelle sur laquelle l'utilisateur agit. De cette manière, l'agent virtuel peut lire et agir de façon similaire sur ses propres services numériques que ceux de l'utilisateur. Cette outil a été adapté aussi bien sur une application cliente riche, qu'une application web et une API. Rest<sup>1</sup>

## 5.3 Perception de l'environnement

Notre approche est basée sur l'application des méthodes développementales utilisées en robotique. Comme expliqué dans la Section 4.2, la compréhension du langage est définie comme un processus qui lie des éléments d'un espace de forme (par exemple, des expressions en langage naturel) vers un espace de sens (par exemple, une séquence de schéma d'actions). Trois types de représentations doivent être prises en compte, une pour modéliser les éléments de l'espace des formes, une pour modéliser les éléments de l'espace de sens et une dernière pour modéliser les données nécessaires pour lier les formes aux sens, appelés construction (Goldberg 1995).

---

1. *Representational State Transfer*

Du point de vue développemental et pour une architecture robotique cognitive comme Chris, l'espace des formes est composé des éléments qui peuvent être perçus et sur lesquels l'agent peut possiblement agir. Ces éléments sont typiquement des phrases, des objets ou des agents. L'espace de sens correspond aux actions et leur paramétrisation qui peuvent être exécutées par des fonctions primitives comme les commandes moteurs *grasp*, *move* ou *release*. C'est similaire aux primitives sensorimotrices décrites dans le développement chez l'humain (Mandler 1988, 1992, 2005). Les constructions sont les structures de données qui sont apprises à partir de l'expérience et qui agissent comme des instructions pour le module de compréhension du langage afin de construire une représentation du sens paramétrable (Goldberg 2003).

Dans le contexte de l'agent virtuel (voir la Figure 5.1), le rôle du module *SensorProcess* est de convertir les événements venant de l'environnement dans une représentation que le système peut traiter. Pour cela, le module tire les événements à partir du module *Environnement*, l'espace des formes, paramétrise la représentation sémantique de l'événement grâce au module *Construction-Memory*, où sont stockées les constructions, et transmet le résultat au module *InterpreterProcess*.

### 5.3.1 Structure des événements

L'environnement est composé de l'utilisateur, de l'agent virtuel, du modèle d'environnement décrit en Section 5.2 et des énoncés produits par l'utilisateur. L'environnement est représenté comme une queue d'événements consommés par le module *SensorProcess*, les événements sont les suivants :

**NewUtterance** Cet événement est créé chaque fois qu'un message en langage naturel est produit par l'un des acteurs. Il est composé du message et de sa source, soit l'agent soit l'utilisateur. Par exemple : « *user : Search for Boston* ».

**ExecuteEvent** Cet événement est généré chaque fois qu'un élément *action* est déclenché dans l'environnement. Il est composé de la référence à l'élément déclenché et de la source, soit l'agent, soit l'utilisateur. Par exemple, lorsque l'utilisateur déclenche le bouton « recherche » dans une page Wikipedia.

**FillEvent** Cet événement est généré quand un élément *field* est mis à jour dans l'environnement. Il est composé de l'élément mis à jour, de la nouvelle valeur et de la source, soit l'agent, soit l'utilisateur. Par exemple, quand l'utilisateur remplit un champs texte sur une page Wikipedia.

**FocusEvent** Cet événement est généré chaque fois qu'un élément est sélectionné dans l'environnement. Il est composé de l'élément et de la source. Nous avons développé une fonction spécifique qui permet à l'utilisateur de montrer un élément sur son interface graphique, par exemple un paragraphe dans une page Wikipedia. Ceci peut être utilisé par l'utilisateur pour montrer comment exploiter le résultat d'une action, comme copier une recherche Wikipedia dans un e-mail.

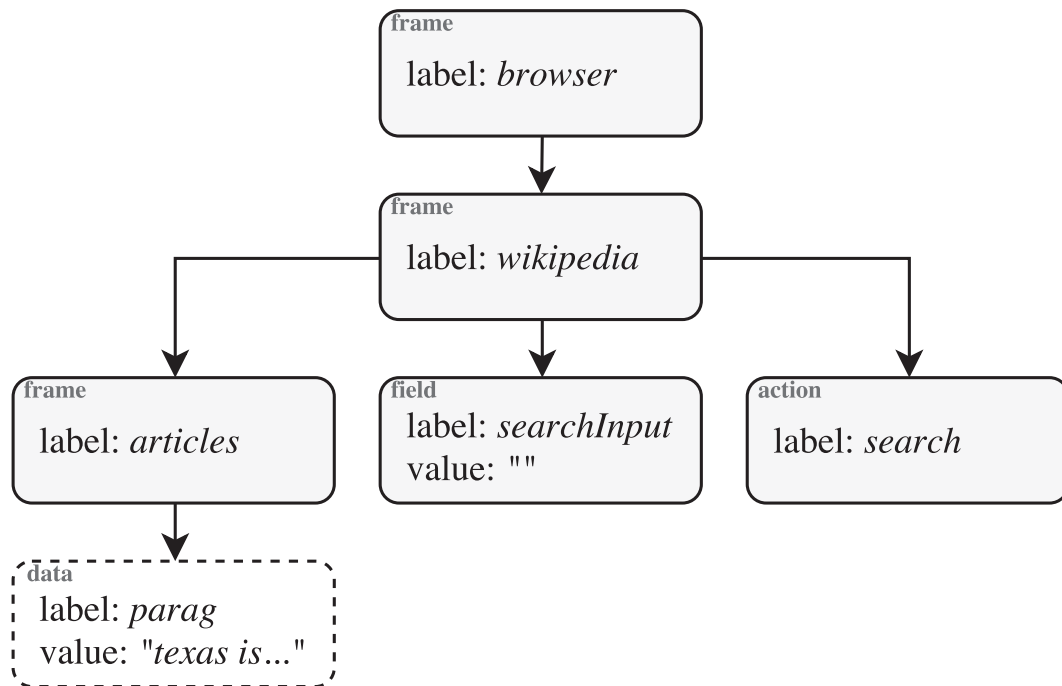


FIGURE 5.3 – Un exemple illustrant une page Wikipédia en train d'être chargée dans le navigateur. La structure hiérarchique forme des chemins de label depuis l'élément racine vers les feuilles de l'arbre. Dans cet exemple, lorsque l'article sera complètement chargé, un événement *AddEvent* sera généré et contiendra un nouveau nœud de type *DataNode* avec comme chemin de label *browser/wikipedia/articles/parag*.

**AddEvent and DeleteEvent** Ces événements sont générés quand la structure du modèle de l'environnement change. Comme décrit précédemment, le modèle de l'environnement possède une structure en arbre, ces événements correspondent respectivement à l'ajout ou la suppression d'un nœud feuille (*action*, *data*, *field*). Ils sont composés de l'élément feuille ainsi que d'une référence qui est le chemin des labels depuis l'élément racine vers la feuille. La Figure 5.3 montre un exemple du modèle de l'environnement lorsqu'une page Wikipédia est chargée dans le navigateur.

### 5.3.2 Représentation du sens

Le sens est extrait d'énoncés qui commandent l'agent d'effectuer une certaine action, comme « *send an email to Peter* ». La principale capacité d'apprentissage de l'agent virtuel est d'apprendre à lier de façon déclarative de tels énoncés à une séquence d'actions primitives qui exécutent le but attendu. Cette séquence d'actions correspond à une procédure dans notre représentation du sens et nous définissons 6 actions primitives qui les composent. Trois de ces actions primitives sont les commandes moteurs définis dans la Section 5.2 (*Fill*,

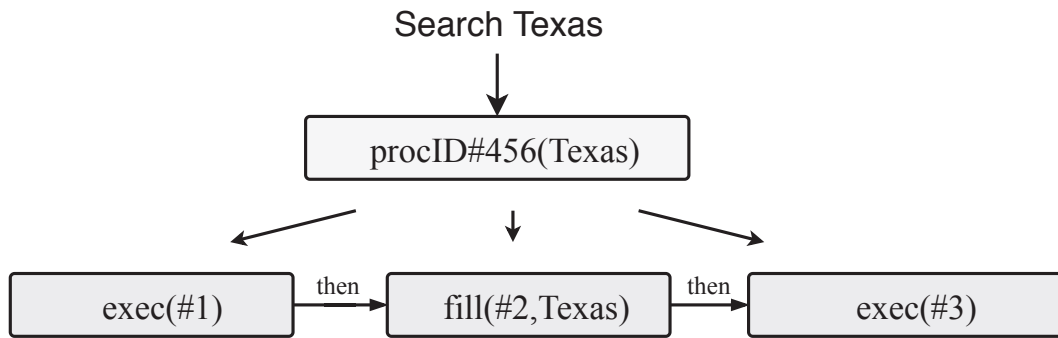


FIGURE 5.4 – Exemple d’une construction grammaticale ancrée sur des actions primitives.

*Execute*, *Speak*) et nous définissons trois actions supplémentaires de nature différente :

**Focus** Utilisé pour pointer un élément dans l’environnement.

**Retain** Utilisé pour lier un élément pointé avec une expression en langage naturel.

**End** Utilisé pour signaler la fin d’un apprentissage.

Ces trois dernières actions ont été introduites seulement pour permettre le fonctionnement de l’assistant et décrites en détails dans les sections suivantes. Le *Focus* est utilisé pour diriger l’attention, le *Retain* pour indiquer à l’assistant de retenir une information, ensemble, ces actions permettent de réaliser une forme de copier-coller pour échanger de l’information entre deux opérations. L’action *End* déclenche l’apprentissage *One-Shot* de constructions.

La première étape est d’ancrer le langage directement sur ces 6 actions. Comme illustré dans la Figure 5.4 où “Search Texas” est lié à une structure sous forme de prédicat, qui à son tour est ancrée sur les actions primitives.

Ensuite, la Figure 5.5 illustre comment une procédure complexe peut se décomposer en d’autres procédures de façon hiérarchique pour finalement s’ancrer sur 6 actions primitives. Les procédures sont toujours définies en termes de ces 6 actions primitives, mais lorsqu’une nouvelle procédure est apprise, elle peut être définie en termes des procédures apprises précédemment. Une action primitive est une fonction qui est directement implémentée au sein de la plateforme qui n’évolue pas et est indépendante d’un domaine.

Une procédure est une séquence ordonnée d’actions primitives décontextualisées (possédant des variables). Par exemple, « *send an email to RECIPIENT about SUBJECT and say that CONTENTS* », est décomposée comme une séquence d’actions primitives *fill* et *execute*, et les trois variables qui seront liées à la propriété « valeur » des actions *fill* (RECIPIENT, SUBJECT et CONTENTS). Une procédure peut être contextualisée en instanciant ses variables à partir d’un énoncé de l’utilisateur, par exemple, « *send an email to pfdominey@gmail.com about the meeting and say that I will be late* ».

Comme dans le système d’apprentissage d’actions de Lallée (Stéphane Lallée et al. 2010), il n’y a pas de prédicats prédéfinis. Toutes les représenta-

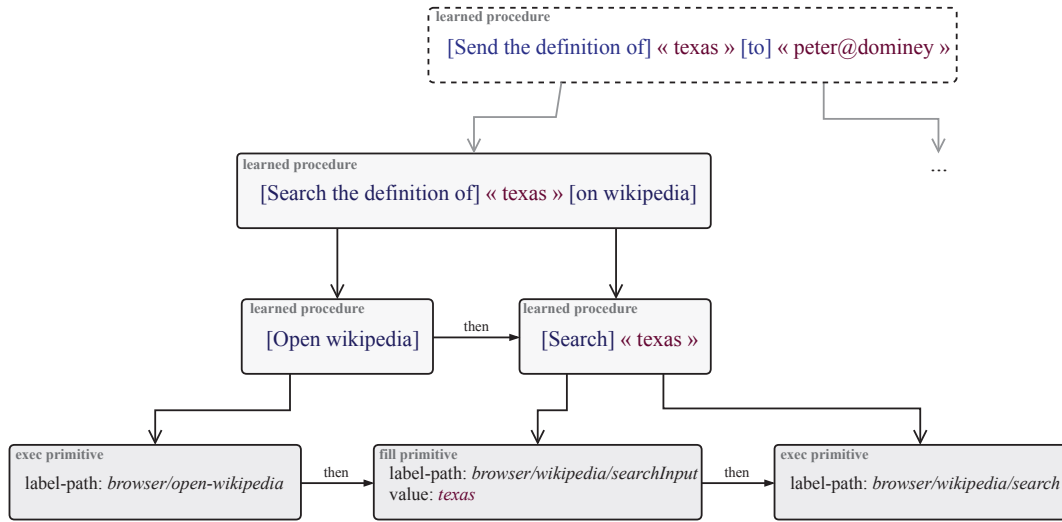


FIGURE 5.5 – Procédure apprise illustrant la compositionnalité de la représentation du sens. Au niveau le plus bas, la procédure est ancrée sur les actions primitives. Ceci illustre la possibilité de structures compositionnelles dans la définition de nouvelles procédures.

tions du sens sont ancrées dans les primitives perceptuelles de l'interface, ce qui correspond à remplir un champ, cliquer sur un bouton, etc. Du point de vue de l'agent, ces primitives sont analogues aux schémas CONTAINER, MOVE, CONTACT, LOCATION, START PATH, END PATH et LINK de Mandler (Mandler 1992). L'absence de représentations sémantiques explicites est illustrée dans la Figure 5.5. Ainsi, la commande de haut niveau « *send the definition of texas to peter@dominey* » est définie par un niveau intermédiaire, comme « *search the definition of texas in Wikipedia* » et qui à son tour se décompose en des actions plus simples. À la fin, le sens repose sur les schémas les plus primitifs « *fill* » et « *execute* ». Les arguments des plus hauts niveaux sont propagés vers les niveaux les plus bas. En conséquence, l'abstraction est organisée autour du langage.

### 5.3.3 Mémoriser les constructions

Le module *ConstructionMemory* enregistre les constructions comme un lien entre une partie « forme » et une partie « sens ». La partie « forme » correspond à des schémas d'événements provenant de l'environnement et la partie « sens » aux procédures. Comme illustré précédemment, les schémas contiennent des emplacements pour les variables qui sont liés aux variables de la procédure et qui permettent ainsi de l'instancier quand une procédure est invoquée. La notion de construction provient du domaine de la linguistique cognitive, où les constructions grammaticales sont des liens entre la forme d'une phrase et son sens (Goldberg 1995, 2003). Dans ce contexte, nous exploitons la distinction entre les mots de classes fermés (mots ayant une fonction grammaticale) utilisés dans la caractérisation de la partie forme des constructions



et les mots de classes ouvertes (noms, verbes, adjectifs, etc) quiinstancient les variables dans les constructions (Dominey et al. 2003 ; Hinaut et Dominey 2013).

Le module *SensorProcess* utilise ce lien entre la forme et le sens pour trouver une correspondance entre le schéma d'un événement de la partie forme des constructions contenues dans le module *ConstructionMemory* et un événement courant venant de l'environnement. Par exemple, pour une phrase de l'utilisateur comme « *send an email to clement@gmail.com about the paper and say finish it* », le module *SensorProcess* compare la phrase avec l'ensemble des schémas du module *ConstructionMemory*. Quand une correspondance est trouvée, les variables de la construction sont instanciées et propagées dans la procédure. L'évènement courant, l'instanciation des variables et la procédure constituent un résultat d'analyse et est ensuite transmit au module *InterpreterProcess*.

## 5.4 Gestion du dialogue et apprentissage

Le module *InterpreterProcess* maintient l'état de l'agent virtuel et du scénario dans lequel il est engagé avec l'utilisateur. Ceci implique les scénarios dans lesquels un utilisateur peut coopérativement exécuter un plan d'actions avec l'agent virtuel et les scénarios dans lesquels l'agent virtuel apprend de nouveaux plans d'actions. Pour cela, le module *InterpreterProcess* a accès à l'ensemble des modules sur lesquels il peut exécuter les actions primitives et retrouver l'état courant. Cela inclue, le module *ConstructionMemory*, le module *ContextMemory*, le module *EpisodicMemory* et le module *MotorProcess*. Les opérations du module *InterpreterProcess* sont illustrées dans le diagramme d'activité dans la Figure 5.6.

Nous présentons en premier les interactions que le module entretient avec les autres, puis nous illustrons son fonctionnement autour de différents exemples. Enfin, nous présentons l'algorithme *One-shot learning*.

### 5.4.1 Interaction avec les autres modules

#### ContextMemory

L'état du module *InterpreterProcess* est caractérisé par 2 variables dans le module *ContextMemory*. L'une maintient l'instance de la procédure courante et l'autre la phase dans laquelle l'agent virtuel est engagé, soit :

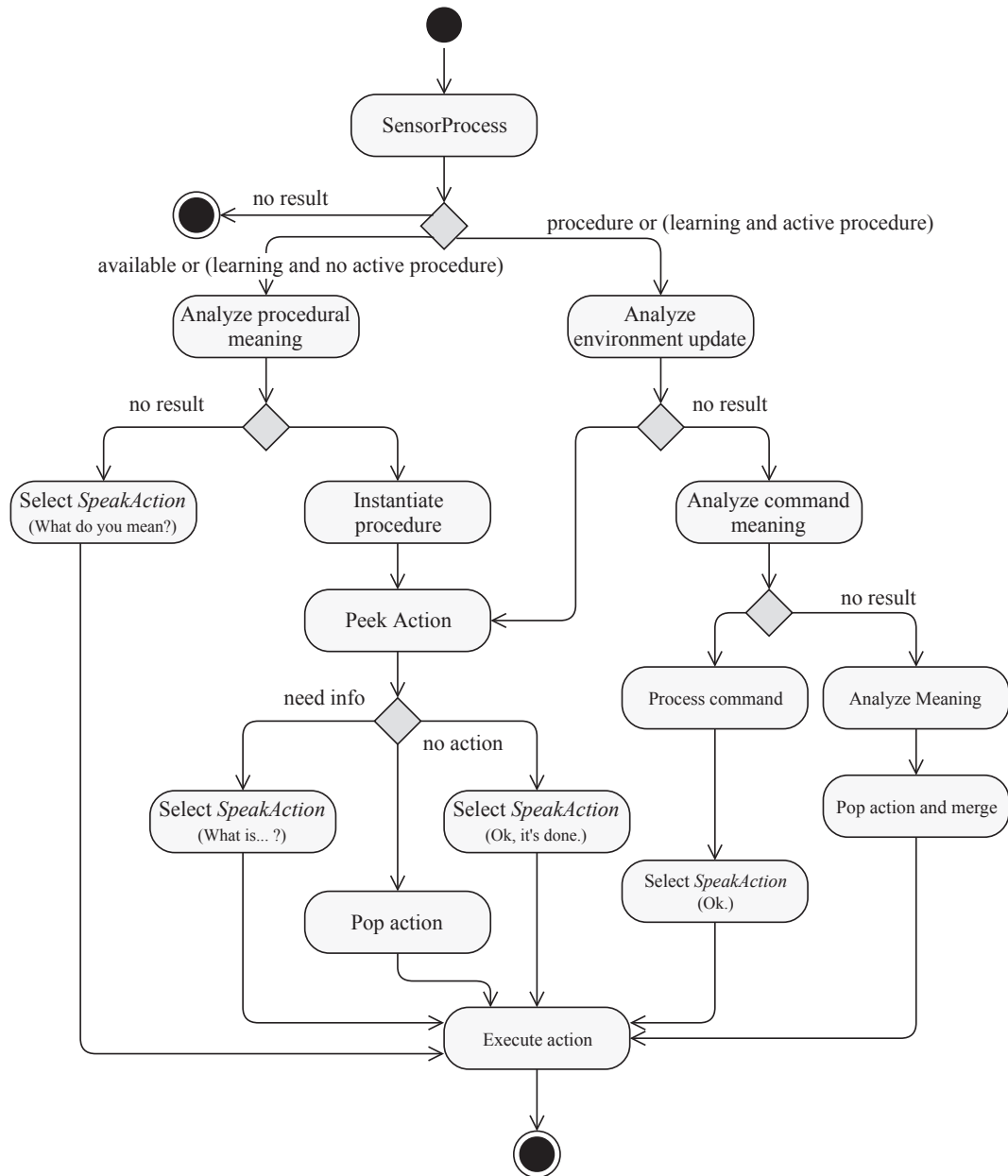
**Available** Indique que l'agent ne participe à aucune activité, ni d'apprentissage ni d'exécution.

**Learning** Indique que l'agent est en train d'apprendre un nouveau plan d'actions.

**Executing** Indique que l'agent est en train d'exécuter un plan d'actions.

Le contenu du module *ContextMemory* et le résultat d'analyse fournie par le module *SensorProcess* vont déterminer la prochaine transition à effectuer.



FIGURE 5.6 – Diagramme d'activité du module *InterpreterProcess*.

## EpisodicMemory

Quand une nouvelle procédure est en train d'être apprise, le module *InterpreterProcess* stocke chaque événement et action primitive exécutée dans le module *EpisodicMemory*. Cette information est utilisée par l'algorithme d'apprentissage pour rejouer toutes les actions exécutées pendant la phase d'apprentissage, de sorte que ces actions peuvent alors constituer la partie sens de la nouvelle construction créée. Pendant l'apprentissage, l'utilisateur peut invoquer des fonctions primitives, et peut également invoquer des procédures apprises qui sont ancrées dans des fonctions primitives, comme illustré dans la Figure 5.5.

## ConstructionMemory

Le module *ConstructionMemory* est mis à jour lorsque l'algorithme d'apprentissage produit une nouvelle construction. Ceci utilise une méthode d'apprentissage « *one-shot* », où la démonstration retenue dans le module *EpisodicMemory* est transformée en une procédure avec des variables correspondant à la partie sens d'une nouvelle construction. Le module *ConstructionMemory* est également utilisé par le *SensorProcess* pour déterminer si un énoncé courant de l'utilisateur correspond à une construction déjà apprise. Dans ce cas, la construction apprise est instanciée et exécutée.

## MotorProcess

Le module *MotorProcess* est utilisé pour exécuter les actions motrices primitives, pendant l'apprentissage et l'exécution d'une procédure.

### 5.4.2 Flux de contrôle du module *InterpreterProcess*

L'assistant virtuel peut être engagé dans deux modalités principales : l'une lorsqu'il exécute une procédure et l'autre lorsqu'il en apprend une nouvelle.

#### Exécuter une procédure apprise

La situation la plus simple est lorsque l'assistant virtuel exécute une procédure et que l'utilisateur fournit tout le contexte en une seule fois. Par exemple : « recherche Texas sur Wikipédia ». Dans cette situation, le module *InterpreterProcess* exécutera la procédure apprise qui a été récupérée lorsque l'énoncé a été trouvé dans le module *ConstructionMemory*. A ce stade, l'assistant virtuel sera dans un état *Available*. Le systèmeinstanciera la procédure et propagera les variables. C'est-à-dire que toutes les actions primitives seront complètement instanciées. L'assistant virtuel exécutera la procédure et notifiera l'utilisateur lorsqu'elle sera terminée. L'exécution de la procédure correspondra aux événements suivants :

- Fill("browser/wikipedia/searchInput", "texas")
- Execute("browser/wikipedia/search")

— Speak(“Ok, It’s done”) [*À cette étape, le GUI montre l’article Wikipédia*].

### Exécuter une procédure apprise lorsque de l’information manque

Une deuxième situation est celle où l’utilisateur ne fournit pas tout le contexte dans l’énoncé, comme « recherche la définition d’un terme ». La procédure qui sera récupérée sera similaire à l’exemple précédent, sauf que le terme de la recherche est manquant. Ceci laissera la variable de la primitive *Fill* non instanciée dans la procédure récupérée. Dans un tel cas, lorsqu’une variable non liée est rencontrée, le module *InterpreterProcess* générera une action primitive *Speak* afin de demander la variable manquante telle que : « Quelle est la requête ? ». Le prochain énoncé de l’utilisateur sera interprété comme la valeur de la variable manquante. Ceci mène à la troisième situation, quand l’assistant virtuel ne sait pas encore comment demander la variable manquante.

### Apprendre à demander de l’information

Afin de générer une question, l’assistant virtuel doit apprendre à associer un élément de l’environnement avec le langage naturel. Pour cela, le module *InterpreterProcess* demandera d’abord à l’utilisateur d’expliquer un élément de son interface soit par le langage naturel tel que « Qu’est-ce que c’est que ça : /browser/wikipedia/searchbox/searchInput ? » ou par pointage sur le GUI. La réponse de l’utilisateur sera liée à la référence de l’élément dans le *ConstructionMemory* et sera réutilisée pour une prochaine itération pour poser une question, comme illustré dans le second scénario de la Section 5.8.

### Apprentissage « One-shot » d’une procédure

Les scénarios d’apprentissage sont déclenchés lorsqu’aucune construction n’est trouvée et que l’état actuel de l’agent est *Available*. Dans ce cas, si l’utilisateur produit un énoncé comme « *Search Texas on Wikipedia* », le système invitera l’utilisateur à expliquer ce que le système doit faire ou à fournir une démonstration via une interface graphique. Une marque sera insérée dans le module *EpisodicMemory* pour indiquer le début de la phase d’apprentissage.

Le module *InterpreterProcess* enregistrera tous les événements se produisant dans l’environnement dans le module *EpisodicMemory*, y compris ceux résultant de l’exécution de procédures apprises pendant l’enseignement. Le module *InterpreterProcess* permet également de combiner plusieurs procédures apprises dans une nouvelle procédure. Pour composer ces sous-procédures, les actions primitives *focus* et *retain* peuvent être utilisées pour sauvegarder temporairement l’évaluation d’une procédure dans une variable et la réutiliser dans une autre sous-procédure.

Cela permet d’apprendre à interpréter des énoncés tels que : « envoyer la définition de Texas à peter.dominey@inserm.fr » où « la définition de Texas » doit être évalué et appliqué à « envoyer  $x$  à peter.dominey@inserm.fr ». Dans cet exemple, également illustré dans la Figure 5.7, l’utilisateur peut combiner les procédures précédemment apprises « rechercher Texas sur Wikipedia » et

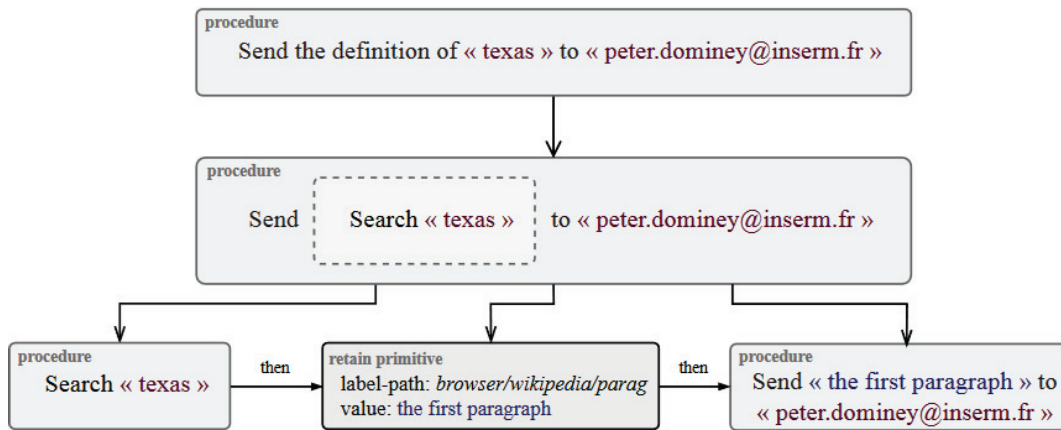


FIGURE 5.7 – Une procédure illustrant la compositionnalité sémantique.

« envoyer un e-mail à peter.dominey@inserm.fr ». Entre les invocations de ces deux procédures, l'utilisateur peut demander à conserver la définition de Texas, et indiquer dans l'interface graphique l'élément à conserver. Le module *InterpreterProcess* créera la variable « définition de Texas » pointant vers cet élément dans le module *ContextMemory* et cette variable pourra être réutilisée par l'utilisateur dans la procédure, par exemple :

**Assistant** What is the content of the message ?

**User** The definition of Texas.

Lorsque la procédure est terminée, l'utilisateur doit déclencher l'algorithme d'apprentissage « *One-Shot* » avec l'action primitive *End*. Comme nous venons de le décrire, pendant l'apprentissage, l'agent virtuel stocke tous les événements correspondant à l'exécution des fonctions motrices primitives et des mises à jour perceptuelles. A la fin de l'apprentissage, l'algorithme crée la nouvelle construction comme illustré dans l'Algorithme 1 et la Figure 5.8.

## 5.5 Agir dans l'environnement

Le rôle du module *MotorProcess* est de transformer les schémas primitifs correspondant en des fonctions motrices exécutables dans l'environnement. Par exemple, le schéma « *execute* » fait référence (par un chemin de label) à un bouton dans l'environnement. Ce bouton est représenté dans la mémoire sensorielle par plusieurs propriétés, dont par exemple un identifiant unique. Le schéma ne dispose pas de la valeur cette propriété, le module *MotorProcess* doit donc enrichir le schéma d'informations présentes contextuellement dans la mémoire sensorielle pour être en mesure d'agir concrètement sur l'environnement.

Une fois l'action effectuée, l'environnement générera d'autres événements qui pourront à leur tour être traités par le module *SensorProcess*. Par exemple, appliquer la fonction *execute* sur un élément de type *Action* dans le modèle d'environnement, générera un événement *ExecuteEvent* qui sera traité

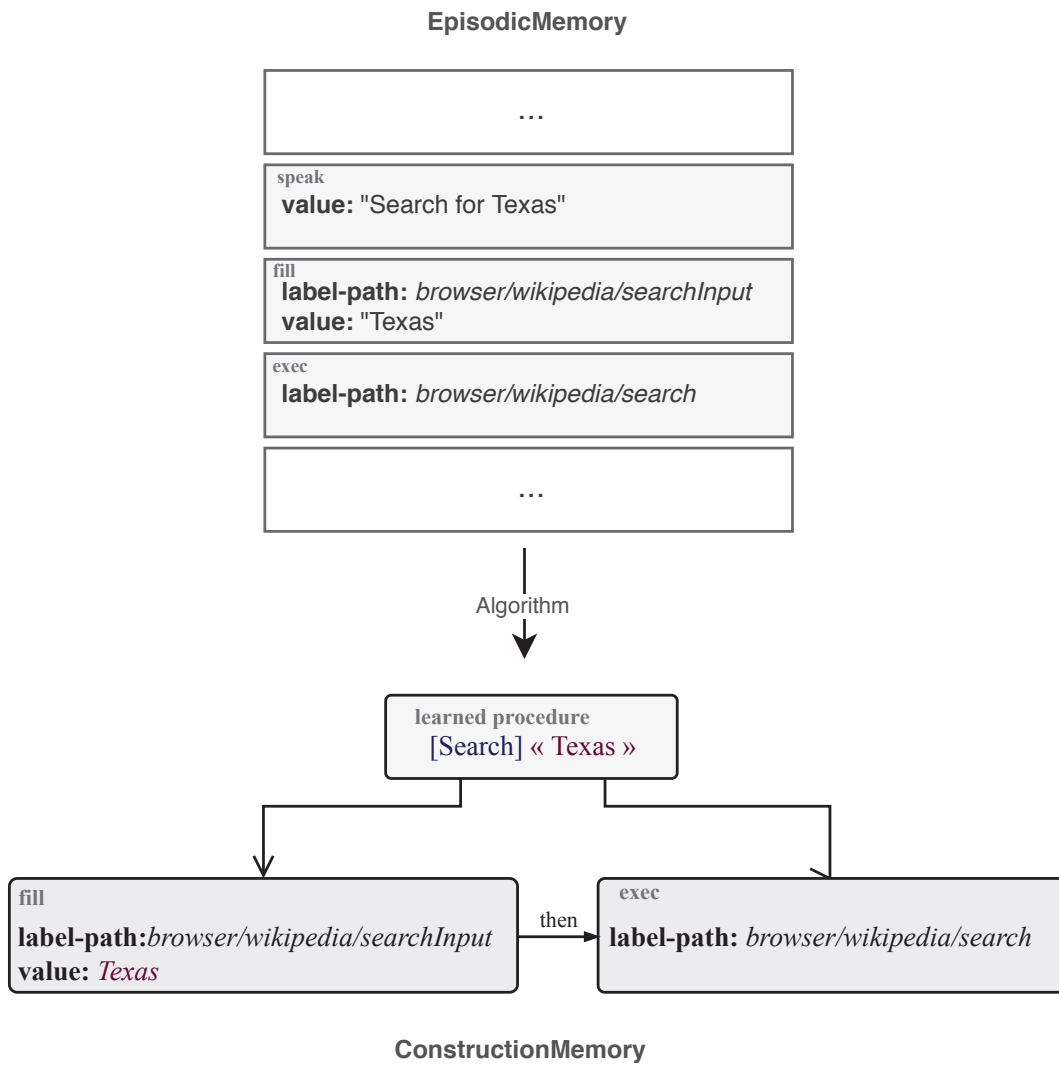


FIGURE 5.8 – Algorithme « *One-shot Learning* » appliqué à l'exemple "Search Texas".

**input** : events a sequence of events  
           utterance the original user utterance  
**output**: procedure a sequence of primitive actions  
           pattern a natural language pattern

```

1 pattern ← utterance ;
2 foreach event of events do
3   switch event do
4     case Exec or Retain do
5       | procedure ← updateProcedure(procedure, event);
6     end
7     case Fill or Speak do
8       | variable ← generateVariable();
9       | if event.value in utterance then
10        | pattern ← updatePattern(event.value, variable,
11        | pattern);
12      end
13      event.value ← variable ;
14      procedure ← updateProcedure(procedure, event);
15    end
16 end

```

**Algorithm 1:** Algorithme « *One-shot Learning* »

par l'agent virtuel ainsi que d'autres événements générés par l'environnement lui-même. Concrètement, cela correspond par exemple, à cliquer sur le bouton « recherche » de la page *Wikipédia*.

## 5.6 Intégration des modules : Exemple

Les sections précédentes expliquaient comment chaque module de l'architecture traite ses propres données. Dans cette section, nous montrons comment ces modules interagissent sur un exemple concret. La Section 5.8 donne des exemples plus détaillés de procédures d'apprentissage et d'exécution, tandis qu'ici nous fournissons un exemple simple d'exécution. L'exemple illustre comment la phrase « Qu'est-ce qu'un tweet ? » est interprétée par l'assistant virtuel pour répondre à la question. Nous supposons que l'assistant virtuel a accès à l'environnement *Wikipédia* et que l'utilisateur a déjà appris à l'assistant virtuel comment répondre aux questions de la forme « *What is a FOO* » basée sur *Wikipédia*.

Dans la première étape, l'énoncé de l'utilisateur est ajouté à l'environnement et un *UtteranceEvent* est ajouté à la file d'événements du module *Environment*. Cet événement est composé de l'énoncé de l'utilisateur « Qu'est-ce qu'un tweet » et de la source, en l'occurrence l'utilisateur. Cet étape est décrite par un diagramme de séquence en Annexe A.1.

Le module *SensorProcess* essaiera de faire correspondre cet événement

avec la partie forme de l'ensemble des constructions contenues dans le module *ConstructionMemory*, afin de trouver une procédure correspondante. Une construction avec le schéma « Qu'est-ce que x ? » sera trouvée et sa procédure correspondante sera sélectionnée pour former le résultat de l'analyse. La procédure est composée des fonctions primitives suivantes :

- Fill(ref1="/browser/wikipedia/searchbox/searchInput", x="tweet")
- Execute(ref2="/browser/wikipedia/searchbox/search")
- Retain(ref3="browser/wikipedia/articles/parag", "the definition")
- Speak("A tweet is...") [*note : c'est le résultat de la recherche Wikipédia enregistrée dans la variable temporaire « the definition »*].

Dans une deuxième étape, le résultat de l'analyse est transmis au module *InterpreterProcess*. L'assistant virtuel est actuellement disponible, ilinstanciera la procédure dans le module *ContextMemory* et commencera à exécuter la procédure, voir Annexe A.2. La première action à effectuer est de remplir l'élément *Field ref1* avec "tweet" (la valeur correspondante de *x*), le module *MotorProcess* exécutera cette action dans l'environnement.

Dans l'Appendice A.3, l'assistant virtuel interceptera les événement suivants déclenchant l'action primitive suivante qui est d'exécuter *ref2*, de la même manière que le module *MotorProcess* exécutera cette action dans l'environnement.

Ensuite, le système *Wikipédia* mettra à jour son état en chargeant une nouvelle page web et sera propagé au module *Environment*. Ceci générera un nouvel événement *AddEvent* qui contiendra une donnée représentant, dans notre exemple, une définition de tweet. Cet événement sera capturé et le module *InterpreterProcess* sélectionnera l'action primitive suivante qui est l'action « *retain* ». La variable définie dans cet action sera instanciée avec la définition de tweet et stockée dans le module *ContextMemory*. Enfin, dans l'Annexe A.4, la dernière action sera exécutée et consiste à retransmettre la valeur de cet variable à l'utilisateur.

## 5.7 Création des variables

Les variables peuvent être créées de 2 façons. Dans un premier cas, pendant l'apprentissage d'une nouvelle construction. Lorsque l'utilisateur répète une chaîne de caractères lors de sa démonstration, et qui est également présente dans sa requête originale, alors cette chaîne est interprétée comme une variable. Par exemple, si l'utilisateur dit « *find the definition of texas* », et que l'utilisateur réalise une démonstration en ouvrant Wikipédia et en tapant « *texas* » dans le champ de recherche, alors « *texas* » est identifié comme une variable dans la construction créée « *find the definition of X* ».

Les variables peuvent également être créées en interne pendant l'apprentissage de procédures qui implique l'action primitive « *retain* ». Par exemple, dans le scénario 3 5.8 des cas d'utilisation, à un certain point, l'utilisateur dit « *retain the definition* ». Pour le système cela signifie qu'une variable « *the definition* » doit être créée. L'action « *retain* » est associée à l'action « *focus* »,

ainsi l'utilisateur portera ensuite l'attention du système sur un élément de son GUI grâce à un petit cercle vert qu'il peut déplacer sur sa page. Pour notre exemple, l'utilisateur pointera l'attention sur le premier paragraphe de la page Wikipédia et le système associera l'élément pointé avec la variable précédemment créée « *the definition* ». Plus tard dans le scénario, l'utilisateur pourra réutiliser cette variable. Ainsi, quand le système demande « *What is the message ?* », l'utilisateur répond « *the definition* ». Le système interprétera cette phrase comme étant le contenu de l'élément qui est associé à cette variable. De cette façon, la variable "the definition" sera toujours associé au premier paragraphe de la page Wikipédia au moment de l'exécution de la construction « *Send the definition of X to Y* ».

## 5.8 Cas d'utilisation

Dans cette section, nous fournissons des exemples de scénarios d'utilisation pertinents qu'un utilisateur peut réaliser avec l'implémentation courante du système. Ces démonstrations sont basées sur le site web Wikipédia et le site web *Gmail*. Nous présentons 4 scénarios par ordre chronologique. Les scénarios commencent lorsque l'utilisateur se connecte pour la première fois avec son assistant virtuel et, dans le scénario final, le système peut interpréter l'énoncé suivant : « *Could you send an email to cl\*\*\*ge@protonmail.com and write him something about Texas ?* ».

### Scénario 1

L'utilisateur apprendra deux actions de base de *Gmail* à son assistant virtuel : comment créer un nouveau brouillon et comment définir le sujet. Dans cette situation, l'utilisateur est connecté à son assistant virtuel avec une boîte de discussion ouverte et est déjà connecté à sa boîte de réception *Gmail*.

**User** Create a draft

**Assistant** What do you mean ?

**User** *The user creates a draft on the Gmail GUI. (The system perceives the event primitives, generated by the user's actions in the GUI, and stores them in the EpisodicMemory).*

**Assistant** Ok (*the virtual assistant has received events from the GUI*)

**User** End (*the user indicates the end of the learning procedure*)

A ce stade, lorsque l'utilisateur dit « End », une nouvelle construction est créée dans le module *ConstructionMemory* avec « *Create a draft* » comme schéma de forme, et les séquences d'événements (procédure) stockées dans le module *EpisodicMemory* comme schéma de sens.

**User** Set the subject

**Assistant** What do you mean ?

**User** *The user fills the subject field on the Gmail GUI*



**Assistant** Ok

**User** End

Encore une fois, quand l'utilisateur dit « End », une nouvelle construction est créée dans le module *ConstructionMemory*. De même, l'utilisateur peut enseigner d'autres actions de base sur *Gmail* telles que l'ouverture de sa boîte de réception, l'envoi d'un brouillon, la définition des destinataires ainsi que d'autres services tels que la recherche d'un terme sur *Wikipedia*. Une vidéo illustrant ces exemples est disponible en ligne<sup>2</sup>.

## Scénario 2

Lorsque l'assistant virtuel a besoin de savoir comment parler d'un élément de l'environnement, il peut demander cet élément. Dans ce scénario, l'assistant apprend comment demander le sujet d'un message afin que la prochaine fois, l'utilisateur n'ait pas à interagir avec son interface graphique pour définir le sujet d'un message. Une vidéo complète de cet exemple est disponible en ligne.<sup>3</sup>

**User** Set the subject (*this matches with a previously learned procedure, from scenario 1, where the user filled the subject field.*)

**Assistant** What does “/browser/Inbox/Subject” mean? (*In the current execution, the “subject” field in the GUI is not yet set, and this leads to an unbound variable corresponding to the subject input box in the Gmail GUI that can also be highlighted*)

**User** The subject of the message (*the user specifies that this field is the subject of the message*)

**Assistant** What is the subject of the message? (*this is still an unbound variable, so the assistant asks for the content*)

**User** Meeting schedule

**Assistant** Ok, It's done.

## Scénario 3

Après quelques apprentissages, l'assistant virtuel est capable d'effectuer des actions de base sur les sites web *Wikipedia* et *Gmail*. Dans ce scénario, nous montrons comment un utilisateur peut enseigner une nouvelle procédure qui combine des procédures précédemment apprises sur différents domaines de tâches comme illustré dans une vidéo en ligne<sup>4</sup>.

**User** Send the definition of Boston to cl\*\*\*ge@gmail.com

---

2. Usage-based Learning Virtual Assistant [Part 1: Learning basic actions]

[https://www.youtube.com/watch?v=Lj\\_CW0OIU88&list=PLKnvrLyZkzbaNzpcz\\_ql3HS-mLvYOu1mB&index=2](https://www.youtube.com/watch?v=Lj_CW0OIU88&list=PLKnvrLyZkzbaNzpcz_ql3HS-mLvYOu1mB&index=2)

3. Usage-based Learning Virtual Assistant [Part 2: Learning GUI elements]

[https://www.youtube.com/watch?v=TthKpUKlrck&list=PLKnvrLyZkzbaNzpcz\\_ql3HS-mLvYOu1mB&index=3](https://www.youtube.com/watch?v=TthKpUKlrck&list=PLKnvrLyZkzbaNzpcz_ql3HS-mLvYOu1mB&index=3)

4. Usage-based Learning Virtual Assistant [Part 4: Learning complex tasks]

[https://www.youtube.com/watch?v=Kreke7tQ7ik&index=5&list=PLKnvrLyZkzbaNzpcz\\_ql3HS-mLvYOu1mB](https://www.youtube.com/watch?v=Kreke7tQ7ik&index=5&list=PLKnvrLyZkzbaNzpcz_ql3HS-mLvYOu1mB)

**Assistant** What do you mean ?  
**User** Go to Wikipedia  
**Assistant** Ok, It's done  
**User** Search for Boston  
**Assistant** Ok, It's done  
**User** retain the definition of Boston  
**User** *The user uses the focus to point the element to retain in its GUI interface.*  
**Assistant** Ok, It's done *(The system binds the indicated paragraph to the variable "the definition of Boston")*  
**User** Close Wikipedia  
**Assistant** Ok, It's done  
**User** Send an email to ce\*\*\*ge@gmail.com  
**Assistant** What is the subject of the message ? *(Note that here the assistant asks a question in order to establish the binding for this variable)*  
**User** Boston's definition  
**Assistant** What is the message ?  
**User** The definition of Boston  
**Assistant** Ok, It's done. *(The contents of the variable "the definition of Boston" is set as the contents of the message)*  
**User** End

La Figure 5.9 fournit une vue de l'interface graphique pendant le déroulement de cette exécution.

Le résultat de cette commande génère le courrier illustré dans Figure 5.10. Il est important de noter que lorsque l'utilisateur explique ce que signifie cette phrase, il peut réutiliser les procédures apprises existantes en les évoquant en langage naturel, ainsi qu'en utilisant des actions plus primitives sur l'interface graphique. Cela illustre la capacité du système à apprendre des structures compositionnelles.

Il convient de noter que le système généralise immédiatement l'utilisation de la même construction avec des arguments différents. Ainsi, l'utilisateur peut maintenant dire « *Send the definition of Texas to john@gmail.com* » et le système exécutera la procédure apprise de manière appropriée.

#### Scénario 4

L'assistant virtuel peut apprendre à interpréter des phrases plus élaborées de la même manière qu'il apprend les procédures. C'est-à-dire que nous pouvons dire une phrase différente qui a le même sens qu'une procédure apprise précédemment, puis lorsque le système demande « *What do you mean ?* », nous énonçons la signification comme dans la formulation apprise. De cette façon, une procédure précédemment apprise peut être reformulée. Ici, nous montrons

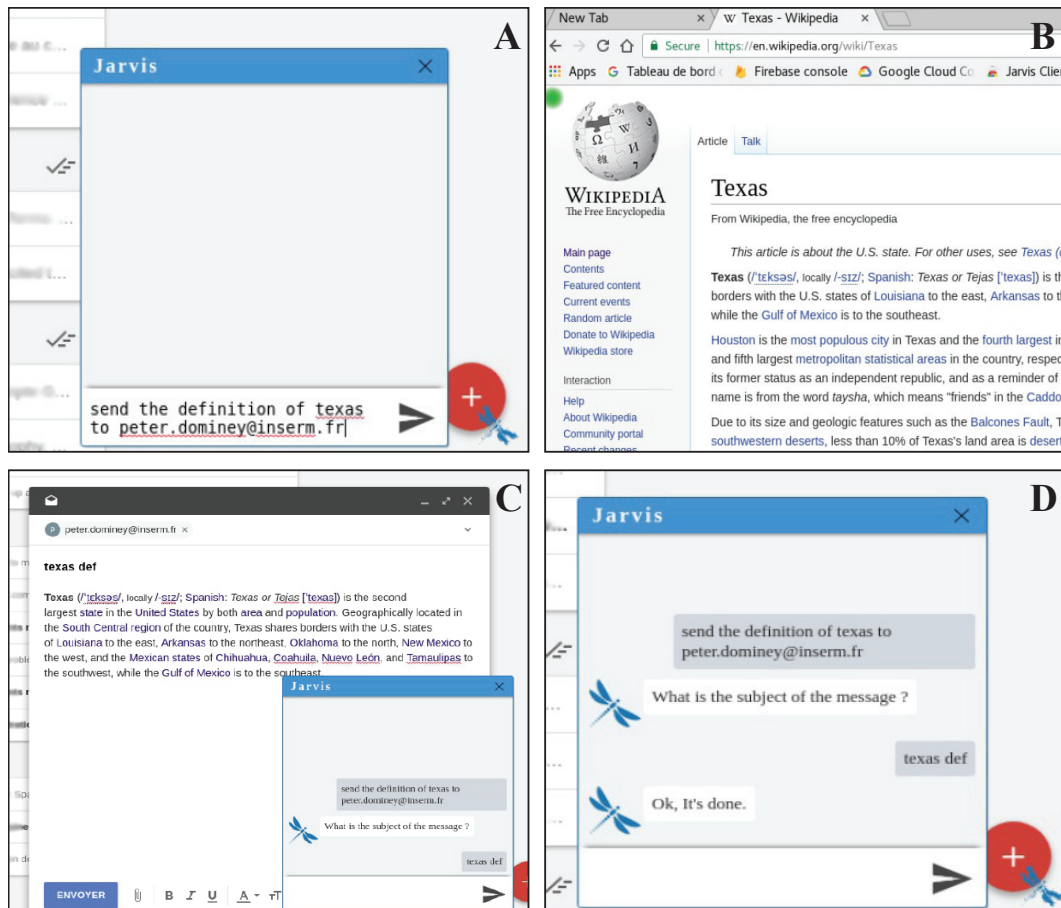


FIGURE 5.9 – (A) L'utilisateur écrit dans sa fenêtre de chat « *send the definition of texas to peter.dominey@inserm.fr* ». (B) Le système trouve une construction et exécute la procédure associée : *open Wikipedia, write « texas » into the search input*. (C) Une fois l'article chargé, l'agent copie le premier paragraphe dans le corps de l'e-mail. (D) Il demande ensuite le sujet du message et l'envoie.

```
MIME-Version: 1.0
Message-ID: <CAMVLL=CkwqWV***om>
Date: Tue, 01 Aug 2017 10:09:58 +0000
Subject: Boston's definition
From: "Clement Delgrange" <ct***vice@gmail.com>
To: cl***ge@gmail.com
Content-Type:multipart/alternative;
boundary="089e08210a4c8773740555ae59f0"

Boston is the capital city and most populous municipality
of the Commonwealth of Massachusetts in the United States.
The city proper covers 48 square miles (124 km2) with an
estimated population of 687,584 in 2017, making it also
the most populous city in the New England region of the
northeastern United States. Boston is the seat of Suffolk
County as well, although the county government was disbanded
on July 1, 1999. The city is the...
```

FIGURE 5.10 – Email sent in scenario 3

le résultat final et une vidéo en ligne met en évidence cette fonctionnalité dans un contexte d'interactions textuelles sur un smartphone<sup>5</sup> :

**User** Could you send an email to cl\*\*\*ge@protonmail.com and write him something about Texas

**Assistant** What is the subject of the message?

**User** Texas definition

**Assistant** Ok, It's done

L'e-mail résultant est affiché dans la Figure 5.11. Ces scénarios illustrent comment un ensemble de commandes primitives peuvent être associées en une commande d'appel de haut niveau, sous la forme d'une construction grammaticale encodée dans le module *ConstructionMemory*.

## 5.9 Évaluation du système

Dans cette section nous montrons comment le système peut être efficace pour réduire le nombre d'interventions nécessaires de l'utilisateur dans la réalisation d'une tâche. La Figure 5.12 illustre comment l'apprentissage et la réutilisation de procédures apprises dans la création de nouvelles constructions permettent de réduire le nombre d'interventions de l'utilisateur sur le long terme. Pour cela nous comparons 3 stratégies qu'un utilisateur peut suivre :

---

5. Usage-based Learning Virtual Assistant [Part 6: Use learned skills on a smartphone]

[https://www.youtube.com/watch?v=2m0YUBIetKc&index=7&list=PLKnvrLyZkzbaNzpcz\\_ql3HS-mLvyOu1mB](https://www.youtube.com/watch?v=2m0YUBIetKc&index=7&list=PLKnvrLyZkzbaNzpcz_ql3HS-mLvyOu1mB)

```

MIME-Version: 1.0
Message-ID: <CAMVLL=BPwT6T***com>
Date: Tue, 01 Aug 2017 10:11:18 +0000
Subject: Texas definition
From: "Clément Delgrange" <ct***vice@gmail.com>
To: cl***ge@protonmail.com
Content-Type:multipart/alternative;
boundary="94eb2c0ea38e4c17910555ae5eec"

Texas is the second largest state in the United States by
both area and population. Geographically located in the South
Central region of the country, Texas shares borders with
the U.S. states of Louisiana to the east, Arkansas to the
northeast, Oklahoma to the north, New Mexico to the west, and
the Mexican states of Chihuahua, Coahuila, Nuevo Leon, and
Tamaulipas to the southwest, while the Gulf of Mexico is to
the southeast...

```

FIGURE 5.11 – Email sent in scenario 4

**No Agent** La stratégie *No Agent* signifie que l'utilisateur n'utilise pas l'agent et réalise toutes ses actions sur un GUI.

**All-in-one** La stratégie *All-in-one* signifie que l'utilisateur minimise les interactions en langage naturel pendant l'apprentissage. C'est-à-dire que l'apprentissage par démonstration est toujours préféré et que lors de l'apprentissage il n'y a aucune réutilisation de procédures précédemment apprises.

**Reuse** La stratégie *Reuse* signifie que l'utilisateur maximise l'utilisation du langage naturel. C'est-à-dire que l'apprentissage par explication est toujours préféré et que l'utilisateur organise l'apprentissage pour réutiliser au maximum les apprentissages précédents, en commençant par les actions les plus simples, puis en les composant.

Pour évaluer ces 3 stratégies, dans un premier temps, nous comptons le nombre d'interactions (agir sur un GUI ou parler à l'agent compte chacun pour une interaction) nécessaires pour réaliser 3 tâches (chercher sur Wikipédia, envoyer un e-mail, envoyer une définition par e-mail) à partir d'une session réelle avec un utilisateur et son agent. Puis, nous simulons une mise à l'échelle en multipliant le nombre de tâches et leur répétition. Ainsi, le panneau A représente le nombre d'interactions nécessaires pour finir 3 tâches 3 fois, qui est le nombre maximum après lequel l'agent a totalement appris la tâche et où il n'y a plus d'interactions liées à l'étape d'apprentissage. Et le panneau B représente le nombre d'interactions nécessaires pour finir 100 tâches 100 fois, simulant ainsi un utilisateur qui utilise intensivement le système.

Lorsqu'il y a peu de tâches et qu'elles sont réalisées peu fréquemment, il n'y a pas d'avantage à utiliser l'agent (Panneau A). En effet, les étapes sup-

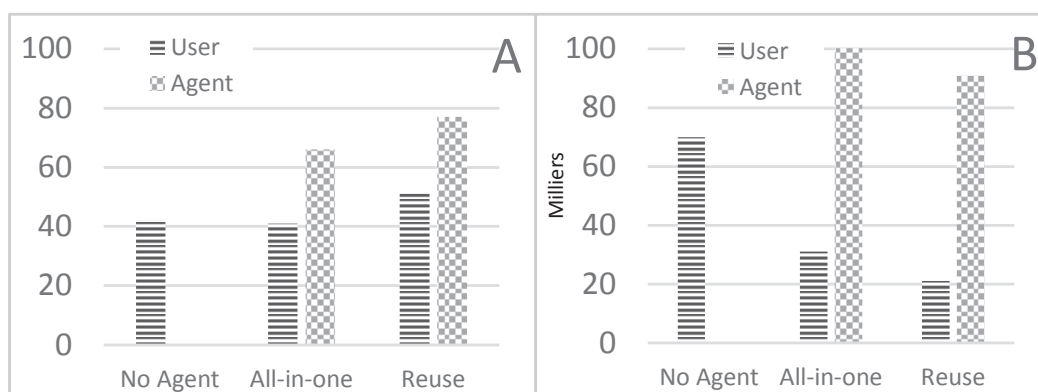


FIGURE 5.12 – Évaluation du système : Nombre d'actions exécutées par l'utilisateur et par l'agent lors de la réalisation de tâches. Nous comparons 3 stratégies, quand l'utilisateur n'utilise pas l'agent, lorsqu'il ne réutilise pas les apprentissages précédents et enfin lorsqu'il les réutilise. A. Pour l'apprentissage de 3 tâches. B. Pour l'apprentissage de 100 tâches et une utilisation intensive.

plémentaires pour enseigner à l'agent ne sont pas suffisamment rentabilisées. Lorsque le nombre de tâches croît ainsi que leur utilisation, on voit une nette réduction du nombre d'interactions de la part de l'utilisateur lorsqu'il utilise l'agent. L'emploi du langage naturel et de la réutilisation permet également de réduire davantage le nombre d'interactions. Ceci ne constitue pas une étude utilisateur mais permet d'avoir une mesure quantitative qui met en lumière l'efficacité du système lorsque l'utilisation et le nombre de tâches possibles augmentent.

## 5.10 Évaluation cross-linguistique

Comme mentionné dans l'introduction, l'un des avantages de notre approche « basée sur l'usage » est le peu de connaissances nécessaires au fonctionnement du programme. Ainsi, nous considérons dans cette section la problématique de la validité cross-linguistique (Bates et al. 1982 ; Bates, Wulfeck et MacWhinney 1991), qui permet à un apprenant d'acquérir des constructions en langue Anglaise qui soit également valide dans d'autres langues comme le Français ou le Japonais.

Le système d'apprentissage de constructions que nous avons développé dans cette thèse respecte la contrainte que certains éléments d'une phrase indiquent les arguments (par exemple des agents, des actions, des objets, des destinataires), tandis que d'autres mots jouent un rôle structurel dans l'identification des relations entre les arguments. C'est ce que l'on appelle traditionnellement la distinction entre classe ouverte et classe fermée (Bates, Wulfeck et MacWhinney 1991). Dans notre système, les mots de classes ouvertes sont identifiés par le système d'appariement d'arguments qui identifie les variables dans les phrases (comme « chercher la définition de Texas » où « texas » est identifié comme une variable parce qu'il apparaît dans la phrase initiale puis à nouveau

dans la recherche Wikipédia). Les mots de classes fermés sont tous les autres mots qui ne jouent pas ce rôle d'argument.

En ce qui concerne la question de la validité interlinguistique, l'idée est que, d'une langue à l'autre, cette distinction entre classes ouvertes et fermées se maintient à un niveau plus ou moins standard. Si tel est le cas, le modèle que nous avons développé devrait pouvoir fonctionner dans n'importe quelle langue. Comme première démonstration de ce concept, nous utiliserons le français.

Dans ce qui suit, nous montrons comment utiliser le français, comme pour la version anglaise, l'utilisateur peut enseigner au système les opérations de base, puis utiliser la composition hiérarchique afin de construire des procédures plus complexes basées sur les procédures qui viennent d'être apprises.

Les 6 dialogues suivants montrent l'apprentissage d'action de base par démonstration.

**Dialogue 5.10.1.** *Apprentissage d'actions de base : recherche Wikipedia*

User Cherche un article sur Paris  
Assistant What do you mean?  
User *L'utilisateur effectue la démonstration sur son GUI*  
Assistant Ok  
User End

**Dialogue 5.10.2.** *Apprentissage d'actions de base : créer un brouillon*

User Crée un brouillon  
Assistant What do you mean?  
User *L'utilisateur effectue la démonstration sur son GUI*  
Assistant Ok  
User End

**Dialogue 5.10.3.** *Apprentissage d'actions de base : spécifier le destinataire*

User Mets le destinataire  
Assistant What do you mean?  
User *L'utilisateur effectue la démonstration sur son GUI*  
Assistant Ok  
User End

**Dialogue 5.10.4.** *Apprentissage d'actions de base : spécifier le sujet*

User Mets le sujet  
Assistant What do you mean?  
User *L'utilisateur effectue la démonstration sur son GUI*  
Assistant Ok  
User End



**Dialogue 5.10.5.** *Apprentissage d'actions de base : spécifier le message*

**User**       Écris le message  
**Assistant** What do you mean ?  
**User**       *L'utilisateur effectue la démonstration sur son GUI*  
**Assistant** Ok  
**User**       End

**Dialogue 5.10.6.** *Apprentissage d'actions de base : envoi du message*

**User**       Envoie le message  
**Assistant** What do you mean ?  
**User**       *L'utilisateur effectue la démonstration sur son GUI*  
**Assistant** Ok  
**User**       End

Bien que l'apprentissage soit en Français, on peut remarquer que l'agent pose les questions en Anglais. Ces questions pourraient être facilement étendues ou traduites mais celles-ci resteraient non apprises. Une évolution intéressante du système pourrait être de laisser l'agent mimer la forme des questions d'un utilisateur (par exemple lorsque celui-ci demande au système « What is tweet ? ») pour ensuite formuler lui-même ses questions. Les prochains dialogues montrent comment ces actions de base sont réutilisées pour former des actions plus complexes.

**Dialogue 5.10.7.** *Apprentissage d'action composées : envoyer un e-mail*

**User**       Envoie un email  
**Assistant** What do you mean ?  
**User**       Crée un brouillon  
**Assistant** Ok  
**User**       Mets le destinataire  
**Assistant** What does "/browser/Inbox/To" mean ?  
**User**       le destinataire  
**Assistant** What is le destinataire  
**User**       cl.delgrange@gmail.com  
**Assistant** Ok, It's done.  
...  
**User**       Envoie le message  
**Assistant** Ok, It's done.  
**User**       End



Ce dialogue, par le mixage entre langue Anglaise et Française rend explicite les parties de l'interaction qui sont « *hard-codé* ». On pointe ici une amélioration qui pourrait être apportée pour la prise en compte des réponses de l'utilisateur lors de questions (par exemple la façon dont la réponse « le destinataire » et utilisée par l'agent dans « *What is le destinataire* »). Comme déjà expliqué, nous ne disposons pas d'outil d'analyse fine du langage, ainsi nous ne sommes pas en mesure de discriminer un déterminant d'un nom par exemple, ce qui contraint l'utilisateur à produire des réponses simples (et éviter des réponses du type « C'est le destinataire de l'e-mail »). Toutefois, par l'observation de dialogues répétés, l'agent pourrait apprendre à identifier les mots importants dans le but de formuler une question.

Les dialogues suivants montrent la composition d'actions. On pourra noter que l'interaction repose sur certains mots clefs comme (« *end* », « *retain* » ou « *retient* »). Ces mots sont introduits dans le système comme *bootstrap* mais peuvent être traduits en différentes langues. S'affranchir de ces mots clefs se révèle plus complexe, le mot clef « *retain* » dénote une action interne non observable par l'agent et le mot clef « *end* » signale la fin d'un apprentissage. Pour que l'agent soit en mesure d'apprendre à les interpréter d'autres techniques doivent être mise en œuvre comme l'apprentissage autonome par essais-erreurs, l'analyse statistique de son expérience, etc. Ce point est discuté dans la conclusion de la thèse.

**Dialogue 5.10.8.** *Apprentissage d'action composées : envoyer une définition*

**User**        Envoie la définition de Paris à cl.delgrange@gmail.com  
**Assistant**   What do you mean ?  
**User**        Cherche un article sur Paris  
**Assistant**   Ok, It's done.  
**User**        Retiens la définition  
**User**        *L'utilisateur effectue la démonstration sur son GUI*  
**Assistant**   Ok  
**User**        Envoie un email  
**Assistant**   What is le destinataire  
**User**        cl.delgrange@gmail.com  
**Assistant**   What is le sujet  
**User**        Définition de Paris  
**Assistant**   Ok, It's done.  
**Assistant**   What is le message  
**User**        la définition  
**Assistant**   Ok, It's done.

**Dialogue 5.10.9.** *Réutilisation d'actions composées : envoyer une définition*

**User**        Envoie la définition de Dijon à cl.delgrange@protonmail.com

**Assistant** What is le sujet

**User** Définition de Dijon

**Assistant** Ok, It's done.

# Chapitre 6

## Variabilité linguistique

L'un des problèmes récurrents lors d'interactions en langage naturel avec un système est la nécessité pour l'utilisateur de simplifier son langage et le sentiment de devoir « parler comme un robot » pour être compris. Ce chapitre traite spécifiquement de ce point et montre comment l'agent virtuel peut améliorer sa tolérance à la variabilité linguistique des énoncés et permettre ainsi aux utilisateurs de s'exprimer avec moins de contraintes.

Comme expliqué dans le Chapitre 5, l'agent virtuel dispose d'un premier niveau d'adaptation à la variabilité linguistique des énoncés. Il consiste à réapprendre une procédure en reformulant l'énoncé non compris ou en donnant une nouvelle démonstration. Par exemple, suite à un apprentissage, l'agent virtuel peut correctement répondre à une demande de l'utilisateur telle que « *Send an email to cl.delgrange@protonmail.com* », mais si l'utilisateur formule sa demande ainsi « *Please, send an email to cl.delgrange@proton-mail.com* », l'agent virtuel ne sera pas en mesure d'établir le lien avec la construction précédemment apprise. Dans ce cas précis, l'utilisateur a la possibilité de reformuler sa demande et ainsi de spécifier lui-même la correspondance sémantique entre ces deux énonciations.

Ainsi, une part importante des énoncés non interprétés est seulement due à une variation dans la forme de l'énoncé bien que la procédure correspondante soit connue et réutilisable. L'utilisateur doit alors se souvenir des précédents échanges avec son agent virtuel afin de reformuler, d'effectuer plusieurs essais-erreurs ou de réapprendre la procédure à l'agent. Dans la pratique, ces tâches s'avèrent être trop lourdes pour l'utilisateur et nuisent à l'utilisation du système. Comme illustré dans la Table 6.1, les déclinaisons possibles d'un énoncé sont nombreuses (synonymie, ajout de locutions, ...), il est donc important de prendre en compte la variabilité linguistique en permettant à l'agent d'être proactif dans l'établissement du rapprochement sémantique entre deux énoncés légèrement différents.

Énoncé de référence
Send an email to cl.delgrange@protonmail.com
Please, send an email to cl.delgrange@protonmail.com

Énoncé de référence
Send an email to cl.delgrange@protonmail.com
Could you send an email to cl.delgrange@protonmail.com
Send an email to cl.delgrange@protonmail.com right now !
Send a mail to cl.delgrange@protonmail.com
Write an email to cl.delgrange@protonmail.com

TABLE 6.1: Exemples de variation de la forme d'un énoncé.

Les travaux présentés dans ce chapitre ont été réalisés en collaboration avec Nicolas Lair et David Mugisha dans le cadre du projet collaboratif dans lequel s'inscrit cette thèse.

## 6.1 Une approche basée sur la similarité sémantique de paires de phrases

Dans l'optique de répondre à la problématique de la variabilité linguistique, nous avons proposé une méthode qui s'appuie sur les travaux effectués dans le champ de l'évaluation de la similarité sémantique, un domaine de recherche très actif car au cœur des problématiques liées à la compréhension automatique du langage.

Il existe plusieurs compétitions internationales autour de ces sujets et les jeux de données sont nombreux et facilement accessibles (Baudis 2018). La compétition SemEval (*International Workshop on Semantic Evaluation* 2018) a proposé entre 2012 et 2017 une épreuve intitulée *Semantic Textual Similarity* (STS) (Agirre 2018) dont les résultats sont encourageants. Le challenge était de pouvoir donner un score de similarité sémantique entre 0 et 5 pour une paire de phrases quelconques. En 2017, l'épreuve a été étendue aux paires dont les phrases sont exprimées dans des langues différentes.

Les techniques employées par les vainqueurs sont relativement variées (Cer et al. 2017). Certains appliquent des algorithmes d'apprentissage profond (Henderson et al. 2017; Shao 2017; Maharjan et al. 2017) à partir de représentation numériques des phrases (*Word Embedding* et *Sentence Embedding*), d'autres construisent des variables issues de l'analyse sémantique des phrases pour nourrir des modèles d'apprentissage automatique plus classiques (Henderson et al. 2017; Tian et al. 2017), enfin d'autres encore tentent d'estimer directement le score de similarité entre les phrases en s'appuyant sur une analyse et les tags sémantiques de *WordNet* ou *BabelNet* (Wu et al. 2017; Hassan et al. 2017).

Nous avons choisi de nous appuyer sur les travaux de l'équipe ECNU (Tian et al. 2017), vainqueur de la compétition, qui ont adopté une approche d'ap-

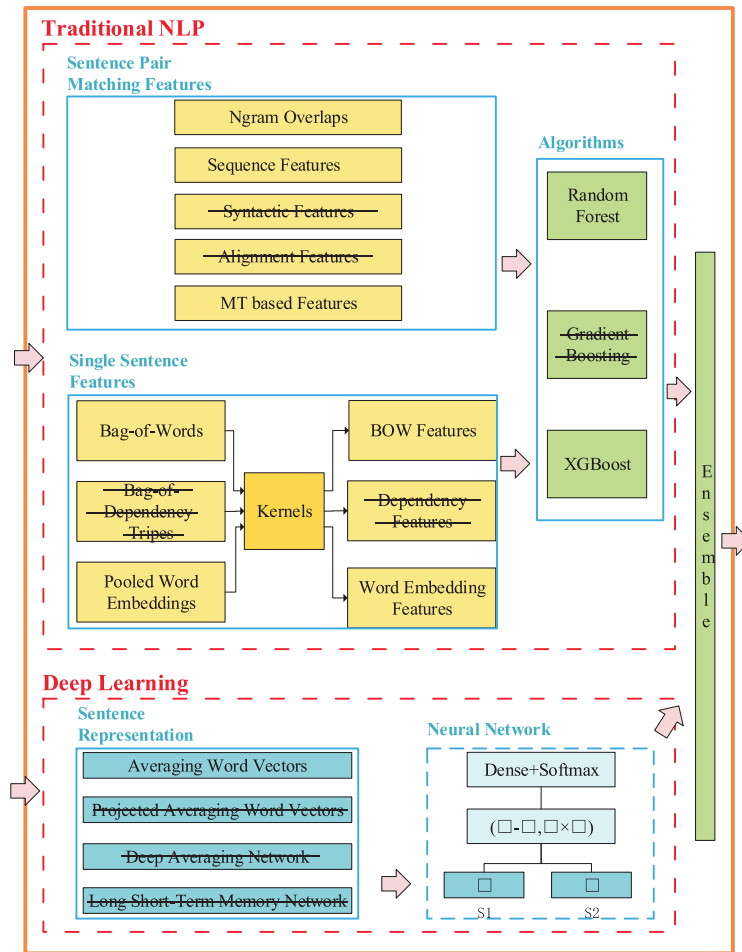


FIGURE 6.1 – Architecture du modèle ECNU où les modules non réimplémentés ont été rayés. *Figure extraite de l'article original (Tian et al. 2017).*

prentissage automatique ensembliste, c'est-à-dire qu'ils ont combiné les résultats de plusieurs modèles pour en améliorer les performances globales. Nous avons ainsi combiné des modèles neuronaux fonctionnant à partir de représentations de mots et des modèles d'arbres de décision s'appuyant sur des variables extraites de l'analyse des phrases.

## 6.2 Module de similarité sémantique

### 6.2.1 Explication du modèle

Le modèle utilisé est donc un modèle ensembliste dont les sous-modèles peuvent être séparés en deux groupes : des modèles neuronaux qui s'appuient sur des représentations vectorielles de mots (exemple : Paragram-WS353 (Wieting, Bansal, Gimpel, Livescu et Roth 2015), GloVe (Pennington, Socher et Manning 2014)) et des modèles d'arbres de décision s'appuyant sur des variables obtenues à partir de l'analyse et de la comparaison de chaque paire de phrases. La Figure 6.1 reprend l'architecture du modèle ECNU en laissant de

côté les modules que nous avons choisi de ne pas utiliser.

### Les modèles neuronaux

Il s'agit de perceptrons multicouches. Les données d'entrée des réseaux sont transformées de la façon suivante :

1. Les phrases sont divisées en « mots ».
2. Chaque mot est remplacé par sa représentation vectorielle (*word embedding*) à partir d'une base de données préconstruites.
3. La représentation vectorielle des phrases est calculée en moyennant celles des mots.
4. Enfin, la représentation vectorielle d'une paire de phrases est calculée en concaténant le produit de Hadamard (produit terme à terme) et la distance  $l1$  des représentations vectorielles des deux phrases de la paire.

Les représentations vectorielles des paires passent ensuite à travers deux couches complètement connectées et un *softmax*. En sortie, le réseau fournit donc un vecteur qui encode le score de similarité.

Plusieurs approches ont été adoptées pour encoder ce score. Une première approche dite « binaire », où le réseau fournit alors la probabilité d'avoir une similarité parfaite. Une seconde approche, dite « multiclasse » qui consiste à encoder la similarité comme 6 classes distinctes correspondant à une similarité de plus en plus forte (classe 0 : complètement différent / classe 5 : complètement similaire). Cette seconde technique est issue de l'approche observée dans plusieurs articles (Tai, Socher et Manning 2015 ; Kiros et al. 2015).

Nous avons implémenté deux modèles neuronaux, un pour chaque encodage des scores. On observe que sur le jeu de données SemEval, les performances pour ces deux encodages sont très similaires. Nous avons tenté plusieurs *word embedding* classiques (GloVe et Paragram) et nos résultats corroborent ceux obtenus par l'équipe ECNU : la représentation Paragram nous a donné les meilleurs résultats sans que le gain ne soit significatif.

La particularité de ces modèles neuronaux est qu'ils ne s'appuient que sur le sens des mots et ne tiennent pas compte de l'ordre des mots, de la syntaxe ou de la grammaire. Il serait intéressant de rajouter un modèle neuronal qui tiennent compte de l'ordre des mots, comme un LSTM par exemple. L'équipe ECNU avait trouvé les performances du LSTM décevante et ont tenté de l'expliquer par le fait que le jeu de données est trop varié (Wieting, Bansal, Gimpel et Livescu 2015), notamment les données vues en entraînement peuvent être très différentes de celles en test. Dans notre cas d'usage, les données d'entraînement devraient être relativement proches des données réelles et donc un modèle LSTM pourrait donner de bons résultats.

### Les modèles d'arbres

Nous avons utilisé deux types de modèles, tous deux à base d'arbres de décision binaire, un modèle de forêt aléatoire et un modèle *xgboost* (Chen et

Guestrin 2016). Ces modèles s'appuient sur des variables numériques exclusivement, préalablement calculées :

- Un premier type de variable mesurent directement la distance ou la similarité entre les deux phrases d'une même paire. On utilise :
  - Des mesures classiques de la qualité de traduction (*Bleu score*).
  - Plusieurs *n-gram overlaps* (au niveau des caractères et des mots).
  - La distance de Levenshtein (nombre minimal d'insertions, délétions, ou substitutions pour transformer une chaîne de caractères en une autre).
- Un second type de variable s'appuie sur une représentation vectorielle de chaque phrase et une transformation à l'aide de fonctions noyaux. Deux types de représentations vectorielles sont utilisés :
  - *Bag of Words*, chaque mot étant pondéré par son IDF (*Inverse Document Frequency*).
  - *Continuous Word Embedding* : chaque mot est transformé dans un *embedding* continu (Glove et Paragram dans notre cas), pondéré par l'IDF. Chaque phrase est alors représentée par la concaténation des *min*, *max* et *average poolings*.

Dans ces deux cas, les phrases sont donc représentées dans des espaces relativement grands. Pour réduire la dimension, chaque paire est transformée à l'aide de plusieurs fonctions noyaux. Les noyaux utilisés sont :

- linéaires : *cosine*, distance de manhattan, distance euclidienne.
- statistiques : corrélation de Pearson, corrélation de Spearman, tau de Kendall.
- non-linéaires : *sigmoid*, RBF (*Radial Basis Function*), laplacien.

Dans un souci de simplification et d'efficacité, nous n'avons pas cherché à reproduire toutes les fonctionnalités utilisées par l'équipe gagnante. Nous avons privilégié les plus simples à implémenter en s'assurant qu'elles étaient de natures différentes. On s'aperçoit qu'il manque notamment des fonctionnalités issues de l'analyse syntaxique (arbres de dépendance et *POS tagging* notamment) qui pourraient apporter une nouvelle nature d'informations pertinentes au modèle. Cependant pour des raisons d'implémentations et de performances, nous n'avons pas pu les prendre en compte dans le modèle actuel. En effet, l'application que nous souhaitons en faire nous contraint à respecter une certaine performance en terme de temps de calcul.

## Le modèle ensembliste

Pour fournir le score final, nous utilisons la moyenne des scores prédits par chacun des modèles. Cependant, le score global du modèle peut être optimisé en affectant un poids différent à chacun des modèles. Dans nos tests préliminaires nous observons un apport différent des modèles selon le contexte. Une analyse plus fine des résultats nous permettra de mieux comprendre la nature de ces différences et d'apporter alors des méthodes d'optimisation.

### 6.2.2 Adaptation des données et de la mesure de performance

Pour adapter le modèle à l'agent virtuel nous avons utilisé deux jeux de données : celui de SemEval 2017 et un jeu de données propres à l'agent virtuel extraites à partir des traces de sessions réelles d'utilisation de l'agent. Nous avons la répartition suivante :

**SemEval 2017** Environ 8000 paires de phrases dans des domaines très variés : tirées de titres de journaux ou des descriptifs d'images.

**Agent virtuel** Jeu de données relativement petit puisqu'il ne compte que 528 paires de phrases et seulement 32 phrases différentes. Nous avons mis en place un questionnaire sur Qualtrics pour nous permettre d'augmenter la taille de ce jeu de données et obtenir des paires de phrases de nature plus variées.

La mesure de performance nécessite également d'être prise en compte dans notre adaptation. La mesure de la performance des modèles dans SemEval est réalisée en mesurant la corrélation de Pearson (corrélation linéaire) entre les prédictions du modèle et les labels sur un jeu de données spécifiquement utilisé pour les tests. Sur le jeu de données SemEval 2017, l'équipe gagnante a réalisé un score de 0.85 sur les données en anglais.

En ce qui concerne notre application, la mesure de la corrélation de Pearson ne peut suffire pour mesurer les résultats de notre dispositif, comme expliqué par la suite. Une mesure sur la pertinence du lien fait entre deux phrases par l'agent serait plus intéressante dans notre cas, par exemple une mesure de type *accuracy*. Cela n'a pas encore été mis en place et dans la suite, nous continuerons à utiliser la corrélation de Pearson pour établir les performances des modèles.

### 6.2.3 Mise en œuvre et résultats préliminaires

#### Mise en œuvre

Le module de similarité sémantique a été implémenté en langage Python et est rendu accessible au reste de la plateforme par l'intermédiaire d'une API Rest. Les consommateurs qui désirent obtenir un score de similarité entre des phrases peuvent appeler le service en fournissant (au format Json) : une phrase à scorer, une liste de phrases de référence et le nombre maximal  $x$  de résultats attendus. Pour chaque appel, le service calculera le score de similarité entre la phrase à scorer et chacune des phrases présentes dans la liste de phrases de référence, il retournera ensuite le score des  $x$  meilleurs résultats. Les Listings 6.1 et 6.2 illustrent respectivement un appel et un résultat retourné par le module de similarité sémantique.

```
{
  "sentence": 'Please , send an email to peter.dominey@in-
               serm . fr ',
```



```

"sentences": [
  'Send an email to cl.delgrange@inserm.fr ',
  'Search the definition of Texas',
  'Set the subject to Meeting'
],
"maxResult": 2
}

```

Listing 6.1 – Exemple d’un appel au module de similarité sémantique

```

{
  "results": [
    {
      "index": 0,
      "score": 4.5
    },
    {
      "index": 2,
      "score": 1.9
    }
  ]
}

```

Listing 6.2 – Exemple d’un retour du module de similarité sémantique

## Résultats

La Table 6.2 présente les performances en terme de corrélation de Pearson calculées lors d’une *4-fold cross validation* à partir du jeu de données propre à l’agent. On observe que le modèle ensembliste est presque toujours meilleur que chaque modèle séparément et qu’il est bien le meilleur modèle en moyenne.

Modèles	Pearson correlation				
	1	2	3	4	Moyenne
Xgboost	0.789	0.742	0.776	0.769	0.769
Random forest	0.777	0.739	0.754	0.843	0.778
Neural model (binary)	0.769	0.725	0.843	0.882	0.805
Neural model (multiclass)	0.836	0.788	0.883	0.918	0.857
Ensemble model	0.849	0.819	0.869	0.925	0.866

TABLE 6.2: Corrélation de Pearson sur un jeu de données de test appliquées à l’agent (*4-fold cross validation*)

On peut remarquer que le modèle neuronal « multiclasse » se révèle particulièrement performant en comparaison des autres modèles. Il faut toutefois nuancer cette observation aux vues des résultats obtenus sur les données de SemEval, et où aucune différence notable n’avait été observée entre les deux

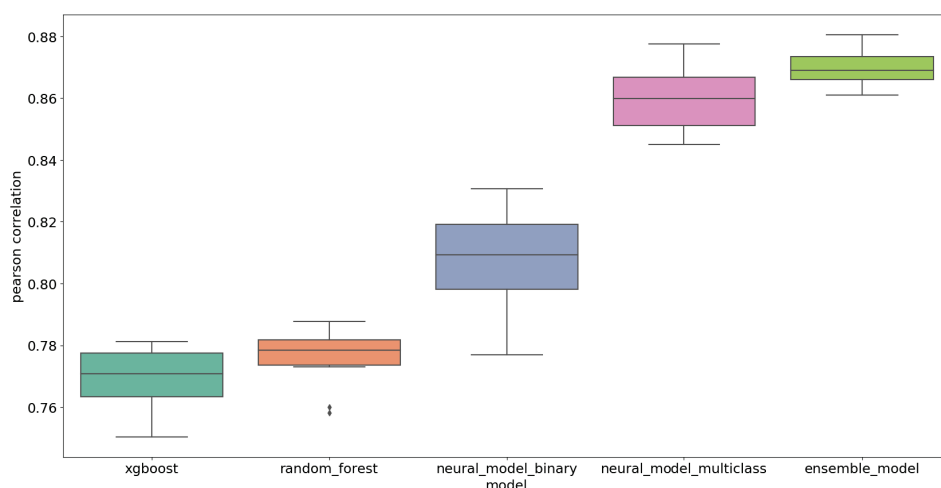


FIGURE 6.2 – Boxplot du résultat de dix *4-fold cross validation* sur un jeu de données appliquées à l’agent

modèles neuronaux. Ainsi, cette différence de performance peut provenir de la particularité du jeu de données propre à l’agent, notamment par le fait que ces données contiennent uniquement des scores entiers qui correspondent bien à l’encodage « multiclasse ».

Les autres modèles ont des performances similaires, légèrement meilleures pour le modèle neuronal binaire.

L’ensemble de données issues de l’agent présente l’inconvénient d’être petit en taille et surtout de ne contenir qu’une petite variété de phrases. On peut donc s’interroger sur la stabilité des résultats obtenus en effectuant une seule validation croisée. Nous avons effectué une dizaine de *4-fold cross validation* et comparé les résultats obtenus, cela permet de tracer les résultats illustrés dans la Figure 6.2 et d’observer la stabilité des résultats. Nous remarquons alors que le modèle ensembliste possède la plus petite variance et semble bien être le plus performant. Ces résultats sont encourageants et permettent de confirmer l’apport de chacun des modèles.

Nous avons également réalisé des tests préliminaires de temps de calcul. Pour un échantillon de 500 paires de phrases, nous obtenons un temps de calcul de 8 millisecondes en moyenne pour calculer le score de similarité sémantique d’une paire de phrases. Ce temps diminue de manière non linéaire lorsque le nombre de paires à scorer augmente et pourra être optimisé. Il permet toutefois de conclure qu’il n’est pas réhibitoire pour notre cas d’application.

## 6.3 Module de la variabilité linguistique

### 6.3.1 Principe de fonctionnement

Le module de similarité sémantique expliqué dans la Section 6.2.1 permet d’évaluer la proximité sémantique d’une paire de phrases quelconques. L’objectif est d’intégrer ce module au sein de l’agent virtuel pour que le module

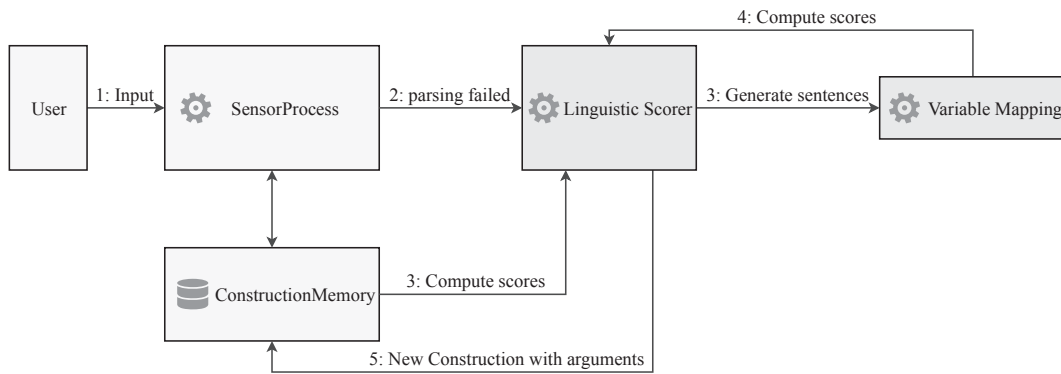


FIGURE 6.3 – Diagramme de communication du module de variabilité linguistique.

*SensorProcess* puisse analyser une demande et proposer un résultat d'analyse même lorsqu'aucune correspondance directe ne peut être établie avec une construction issue du module *ConstructionMemory*. Ainsi comme le montre les Dialogues 6.3.1 et 6.3.2, l'objectif est d'améliorer l'efficacité de l'interaction en évitant à l'utilisateur de devoir reformuler sa demande et en permettant à l'agent de proposer lui-même une correspondance.

#### Dialogue 6.3.1. Sans le module de variabilité linguistique

**Humain** Please, send an email to cl.delgrange@protonmail.com.  
**Agent** What do you mean ?  
**Humain** Send an email to cl.delgrange@protonmail.com. *(l'utilisateur doit se souvenir de cette phrase)*  
**Agent** What is the subject ?  
**Humain** ...

#### Dialogue 6.3.2. Avec le module de variabilité linguistique

**Humain** Please, send an email to cl.delgrange@protonmail.com.  
**Agent** Did you mean : Send an email to cl.delgrange@protonmail.com ?  
**Humain** yes.

### 6.3.2 La similarité sémantique pour la recherche de procédures candidates

Afin d'évaluer une proximité sémantique nous avons donc besoin au minimum de deux phrases. L'une provient de l'énoncé courant de l'utilisateur tandis que l'autre doit provenir d'apprentissages précédents et pour laquelle l'agent connaît l'interprétation. Comme le montre la Figure 6.3 le comportement du module *SensorProcess* doit être modifié afin d'orienter la demande courante de l'utilisateur dans le cas où elle ne serait pas reconnue vers le module de la variabilité linguistique. Le module *ConstructionMemory* doit également être

modifié afin d'enregistrer conjointement avec les modèles de phrases (partie *forme* des constructions) des exemples de phrases instanciées rencontrées au cours de l'expérience de l'agent, par exemple :

- « *Send an email to cl.delgrange@protonmail.com* »
- « *Send an email to peter.dominey@inserm.fr* »
- « *Send an email to cde@cloud-temple.com* »

doivent être enregistrées conjointement avec le modèle de phrase « *Send an email to X* ».

Cette étape préliminaire permet donc de retrouver une construction susceptible d'interpréter une demande en suivant cette démarche :

1. Calculer le score de similarité entre l'énoncé courant et l'ensemble des phrases instanciées enregistrées dans le module *ConstructionMemory*.
2. Sélectionner la paire de phrases ayant obtenu le meilleur score.
3. Retrouver la construction liée à la phrase sélectionnée.

À cette étape, le *SensorProcess* est donc en mesure de fournir la procédure correspondant à la demande de l'utilisateur, pour autant, elle ne dispose pas de valeur pour ses arguments. En effet, l'instanciation de la construction, et donc la possibilité de retrouver dans la nouvelle forme de l'énoncé la valeur des arguments, nécessite une étape supplémentaire. La première raison est que la modification de la forme de l'énoncé peut introduire un décalage de l'ordre des mots dans le modèle de phrase ou leur modification, par exemple « *Please, write an email to cl.delgrange@protonmail.com* » où « *Please* » décale la position de l'argument « *cl.delgrange@protonmail.com* » et « *send* » est substitué par « *write* ». La seconde, est que la valeur d'un argument peut être très variée, par conséquent il sera très probable qu'une valeur particulière n'est jamais été rencontrée et donc absente du module *ConstructionMemory*, comme par exemple « *Send an email to clement.delgrange@in-serm.fr* » en référence à notre exemple précédent.

### 6.3.3 La similarité sémantique pour la détection d'arguments

Nous disposons à cette étape de la demande courante de l'utilisateur ainsi que la construction qui permettrait de l'interpréter. Le modèle de phrase de la construction ne permet pas d'en extraire directement les arguments comme expliqué dans la Section 6.3.2, ainsi l'objectif est de créer une nouvelle forme qui puisse correctement propager la valeur des arguments présents dans la demande courante vers la procédure existante et donc inférer une nouvelle construction.

La Figure 6.4 illustre l'ensemble des étapes algorithmiques pour y parvenir. La première partie correspond à l'étape préliminaire décrite en Section 6.3.2. Dans notre exemple, « *Please, send an email to peter.dominey@in-serm.fr* » est l'énoncé courant de l'utilisateur et la construction sélectionnée offrant le meilleur score est « *Send an email to Z* » → *SendEmailProcedure(Z)*.



FIGURE 6.4 – Étapes algorithmiques du module de variabilité linguistique.

La seconde partie consiste à générer un ensemble de phrases instanciées en utilisant le modèle de phrase de la construction sélectionnée précédemment et de l'énoncé courant. Pour cela nous listons l'ensemble des instanciations possibles des arguments du modèle de phrase en utilisant l'ensemble des mots de l'énoncé courant. Les phrases ainsi obtenues font le lien entre les variables du modèle de phrase et les valeurs possibles dans l'énoncé courant. Comme le montre la Figure 6.4, les phrases peuvent présenter des défauts syntaxiques ou sémantiques comme « *Send an email to send* ». L'hypothèse est qu'en calculant à nouveau le score de similarité de ces phrases avec l'énoncé courant, celle obtenant le meilleur score soit exempte d'erreurs et proche de l'intention de l'utilisateur.

La troisième partie permet donc de retenir la phrase générée la plus proche de l'énoncé courant, dans notre exemple « *Send an email to peter.dominey@inserm.fr* », en utilisant à nouveau le calcul du score de similarité sémantique. Cette dernière étape nous permet donc d'inférer la place des arguments dans l'énoncé courant et de générer une nouvelle construction équivalente à la première :

$$\begin{aligned} \ll \textit{Send an email to Z} \gg &\rightarrow \textit{SendEmailProcedure}(Z) \\ &\Leftrightarrow \\ \ll \textit{Please, send an email to Z} \gg &\rightarrow \textit{SendEmailProcedure}(Z) \end{aligned}$$

### 6.3.4 Mise en œuvre et résultats préliminaires

Le processus de détection d'arguments, décrit précédemment, repose sur l'hypothèse forte que le modèle sera suffisamment performant pour discriminer le sens des phrases lorsque l'on génère les phrases candidates. Nous avons d'abord mesuré que cette hypothèse semble être validée sur un cas relativement simple et par la suite réalisé des sessions réelles d'utilisation de l'agent qui montre de bons résultats (décrites dans la section suivante). Il sera toutefois nécessaire de procéder à d'autres tests pour s'assurer de la robustesse de l'approche.

#### Pertinence des scores

La Table 6.4 montre la performance des différents modèles ainsi que du modèle ensembliste lorsque l'on effectue l'exercice sur un cas simple. Les deux phrases choisies sont les suivantes :

- Phrase 1 : « *Send an email to peter.dominey@inserm.fr* »
- Phrase 2 : « *Please send an email to peter.dominey@inserm.fr* »

On observe que les modèles neuronaux ne font quasiment aucune différence entre les phrases candidates. Ce comportement est cohérent avec le fait qu'ils ne tiennent pas compte de l'ordre des mots mais seulement du sens global des mots. En revanche les modèles d'arbres enregistrent de meilleures performances pour ce cas, certainement aidés par l'email dont la séquence de caractères est

Sentence 1	Sentence 2	Xgboost	Random Forest	Neural model (binary)	Neural model (multi-class)	Ensemble model
Please send an email to peter.dominey@inserm.fr	Send an email to Please	0.8	2.0	4.4	4.0	2.8
	Send an email to send	3.2	2.1	5.0	5.0	3.8
	Send an email to an	0.6	1.3	4.4	4.0	2.6
	Send an email to email	0.5	1.3	4.4	4.0	2.6
	Send an email to to	0.6	1.3	4.4	4.0	2.6
	<b>Send an email to peter.dominey@inserm.fr</b>	<b>4.0</b>	<b>4.3</b>	<b>4.4</b>	<b>4.0</b>	<b>4.2</b>

TABLE 6.4 – Scores obtenus pour la détection d’arguments

Sentence 1	Sentence 2	Xgboost	Random Forest	Neural model (binary)	Neural model (multi-class)	Ensemble model
Please send a email to peter.dominey@inserm.fr	Send an email to cl.delgrange@gmail.com	0.7	1.3	4.4	4.0	2.6
Could you send an email to Peter	Send an email to peter.dominey@inserm.fr	1.9	2.8	4.5	4.1	3.3
Could you send an email to Peter	Could you send an email to Clément	4.5	4.4	4.9	5.0	4.7
Please tell me the definition of Boston	Please send the definition of Boston	4.0	4.6	3.0	2.4	3.5
Please tell me the definition of a bank	Give me the definition of a bank	4.5	4.6	5.0	5.0	4.8
Give me the definition of bank	What is a bank ?	4.0	2.9	2.7	2.7	2.7
Give me the definition of bank	Send an email to peter.dominey@inserm.fr	0.3	0.7	0.0	0.0	0.3

TABLE 6.6 – Scores obtenus pour des phrases inconnues

assez particulière. Bien qu’il s’agisse d’un résultat isolé (confirmé par la suite), le test est réussi puisque c’est bien la phrase correcte qui a été identifiée avec un score de 4.2/5.

La Table 6.6 présente d’autres tests de phrases encore jamais vues par les modèles afin d’évaluer leurs performances. On observe dans la Table 6.6, exemple 1, la confirmation de ce qui avait pu être supposé précédemment, lorsque l’email change trop, les modèles de régression se trompent. L’exemple 3 montre que les modèles se trompent moins lorsque ce n’est pas l’email qui est passé en argument mais un nom.

L’exemple 5 est assez éloquent puisque les termes sont assez différents et que les modèles s’en sortent tous correctement. L’exemple 6 en revanche voit échouer les modèles neuronaux tandis que les modèles de régression se révèlent performants.

L’exemple 7 montre que les modèles savent reconnaître deux phrases complètement différentes.

## Temps de calcul

La détection d'argument introduit également un calcul combinatoire mais qui n'entrave pas l'utilisation voulue. En effet, dans un cas moyen où une phrase comporte 9 mots dont 2 variables, le nombre d'arrangements possibles est de 72. Avec un temps de calcul de 8 millisecondes par paire, le temps par interaction sera de 576 millisecondes. Dans un cas plus difficile, avec 25 mots et 4 variables le nombre d'arrangements est de l'ordre de 300000. Le temps de calcul sera alors approximativement de 25 minute par interaction, ce qui n'est pas envisageable.

Cependant, 300000 phrases correspond au nombre de phrases pouvant être contenues dans le module *ConstructionMemory* lors d'une mise à l'échelle du système et ainsi la détection d'arguments n'aura pour effet que de doubler le temps nécessaire. Par conséquent le principal levier pour permettre une mise à l'échelle du système est la réduction du temps de calcul nécessaire pour calculer le score de similarité d'une paire de phrases qui pourra être complété par la possibilité de filtrer et ainsi de réduire le nombre de phrases à scorer.

## 6.4 Intégration du module de variabilité linguistique

### 6.4.1 Adaptation du comportement de l'agent

Comme expliqué dans le Chapitre 5, le module *SensorProcess* a la responsabilité d'analyser les phrases de l'utilisateur et de produire un résultat unique d'analyse qu'il transfère ensuite au module *InterpreterProcess*. Ce dernier n'a pas accès à la façon dont le résultat de l'analyse d'une phrase est produit, le résultat doit ainsi contenir toute l'information nécessaire pour que le module *InterpreterProcess* puisse guider l'interaction de façon cohérente.

L'introduction du module de variabilité linguistique implique le besoin de traiter une nouvelle nature d'information suite à l'incertitude qu'introduit la notion de score. En effet les conséquences sont que pour une phrase « utilisateur » plusieurs résultats d'analyse sont possibles et que chacun de ces résultats possède un score qu'il faut alors interpréter.

Afin de conserver l'isolement entre la production d'un résultat d'analyse et sa consommation il convient de répartir les responsabilités dans l'adaptation au module de variabilité linguistique entre le module *SensorProcess* et le module *InterpreterProcess*.

Une première expérience a été de confier au module *SensorProcess* la responsabilité d'interpréter les scores en ordonnant les résultats d'analyse en fonction des scores et le cas échéant en validant automatiquement une proposition créée par le module de la variabilité linguistique. Nous avons défini un seuil de score plancher et un seuil de score plafond de façon empirique. Le seuil plancher était fixé à un score de 2.5, ainsi tout résultat inférieur à ce score était ignoré et un seuil plafond de 4.5 où tout résultat avec un score supérieur



était automatiquement validé et la nouvelle construction ajoutée au module *ConstructionMemory*.

Les résultats situés entre le seuil plancher et le seuil plafond étaient transférés au module *InterpreterProcess*. Ce dernier a été modifié pour permettre d'interpréter une liste de résultats ordonnés et d'en valider le meilleur. Pour cela le comportement de l'agent a également été modifié pour impliquer l'utilisateur dans la décision. Ainsi lorsqu'une phrase encore inconnue implique plusieurs possibilités d'interprétation, l'agent demande à l'utilisateur de valider ou non les hypothèses dans une limite de deux tentatives comme illustré ci-dessous :

#### Dialogue 6.4.1.

**Humain** Send an email. (*phrase inconnue*)

**Agent** Did you mean : create a draft ?

**Humain** no.

**Agent** Did you mean : search for email ?

**Humain** no.

**Agent** What do you mean ? (*retour au comportement d'apprentissage*)

Malgré les bons résultats du module de variabilité linguistique, des premiers tests nous ont montré qu'une erreur pouvait impacter l'intégrité du module *ConstructionMemory* de façon durable dû au fait qu'en l'état, l'agent n'a pas la possibilité de corriger un apprentissage faux.

Nous avons dans la suite défini un unique seuil plancher à 3.5 et considéré que toute hypothèse dépassant ce seuil devrait faire l'objet d'une validation par l'utilisateur. Nos premiers résultats sont encourageants avec un faible taux d'erreurs qui doit être nuancé par la proximité des données utilisées pour l'entraînement du module de similarité sémantique avec les données de production.

### 6.4.2 Exemples de cas d'utilisation

La modification du comportement de l'agent implique de nouvelles possibilités d'interactions que nous détaillons dans cette section. Pour chaque nouveau cas d'utilisation, nous décrivons les prérequis (ce que l'agent doit connaître préalablement à l'interaction), le dialogue généré, ainsi qu'une explication du résultat. Une vidéo est en ligne<sup>1</sup> illustrant ces nouvelles interactions.

Dans un premier temps nous décrivons les cas où l'agent trouve une correspondance, que nous déclinons pour les phrases simples (sans variable) et un peu plus complexes (avec variable). Puis nous présentons les cas où l'agent échoue ou nécessite plusieurs essais avant de trouver une correspondance.

---

1. Adaptive Usage-based Learning Virtual Assistant [Full Video]

[https://www.youtube.com/watch?v=DzM\\_JsLqLLE](https://www.youtube.com/watch?v=DzM_JsLqLLE)

### A. Cas où l'agent s'adapte avec succès, sans variable

**Prérequis** L'agent possède une base de connaissances vierges à l'exception d'une seule construction qui lie *Open wikipedia* avec la procédure d'ouverture de page de *Wikipédia*.

**Dialogue 6.4.2.** *Dialogue réalisé pour s'adapter à : Go to wikipedia*

**Humain** Go to wikipedia.

**Agent** Did you mean : open wikipedia ?

**Humain** yes.

**Agent** (*l'agent ouvre une page Wikipédia*)

**Explication** L'agent ne possède pas de constructions qui lie *Go to wikipedia* avec une procédure apprise. Il fait donc appel au module de variabilité linguistique. Celui-ci calcule le score de similarité de cette phrase avec l'ensemble des phrases contenues dans sa mémoire de construction, qui se compose alors uniquement de la construction *Open wikipedia*. Le module de similarité linguistique retourne un score de 4.35 qui dépasse le seuil de 3.5. La construction *Open wikipedia* devient donc une hypothèse que l'agent confirme auprès de l'utilisateur. Lorsque la construction est validée par l'utilisateur, une nouvelle construction est créée afin de lier *Go to wikipedia* avec la procédure de la construction *Open wikipedia*. Et enfin l'ordre est exécuté.

### Cas où l'agent s'adapte avec succès, avec variable

**Prérequis** L'agent a maintenant appris à rechercher un terme sur *Wikipédia*. Il possède donc une construction de la forme « *Search for X* » et pour laquelle il a accumulé de l'expérience en recherchant les termes « *Paris* » et « *Boston* ».

**Dialogue 6.4.3.** *Dialogue réalisé pour s'adapter à : Please, search for Texas*

**Humain** Please, search for Texas.

**Agent** Did you mean : Search for Texas ?

**Humain** yes.

**Explication** L'agent va calculer le score de similarité entre « *Please, search for Texas* » et l'ensemble des phrases présentes dans sa mémoire de construction. Au vue de l'expérience de l'agent, celle-ci possède entre autres les phrases « *Search for Boston* » et « *Search for Paris* ». A ce niveau le meilleur résultat obtenu est « *Search for Boston* » avec un score de 4.23 (proche de « *Search for Paris* » avec 4.19). La phrase retenue ayant des variables, l'agent va tenter de retrouver la valeur courante de la variable dans la phrase inconnue et ainsi être en mesure de propager sa valeur dans la procédure correspondante. Pour cela il va générer l'ensemble des arrangements possibles à partir de la forme de la construction retenue et de la phrase inconnue, soit :

— « *Search for Please* »

- « *Search for for* »
- « *Search for Texas* »
- « *Search for search* »

L'agent va ensuite calculer les scores de similarité de la phrase inconnue avec l'ensemble de ces phrases. « *Search for Texas* » obtient tout juste le meilleur score avec 4.79. L'agent peut donc dériver une nouvelle construction de la forme « *Please, search for X* » en réutilisant la procédure de la construction « *Search for X* ». Une fois l'hypothèse validée par l'utilisateur, la construction pourra être ajoutée au module *ConstructionMemory*.

### Cas où l'agent échoue à s'adapter

**Prérequis** Après plusieurs cycles d'apprentissage, l'agent peut maintenant envoyer un email à quelqu'un et possède une construction de la forme « *Send an email to X* ».

**Dialogue 6.4.4.** *Dialogue réalisé pour s'adapter à* : Send the definition of Boston to clement.delgrange@inserm.fr

**Humain** Send the definition of Boston to clement.delgrange@inserm.fr.

**Agent** Did you mean : Send an email to clement.delgrange@inserm.fr ?

**Humain** no.

**Agent** What do you mean ?

**Humain** (*l'utilisateur explique la procédure*)

**Explication** Comme pour l'exemple précédent, l'agent a pu trouver une correspondance et une proposition de variable dont le score de similarité était supérieur à 3.5. L'hypothèse établie que « *Send the definition of Boston to X* » et « *Send an email to X* » sont deux constructions équivalentes, lorsque l'utilisateur l'invalidé, la construction est rejetée et l'agent retrouve le comportement d'apprentissage expliqué dans le Chapitre 5.

Le comportement de l'agent permet également d'effectuer plusieurs propositions dans le cas où plusieurs hypothèses dépassent le seuil de 3.5. Ainsi, si l'utilisateur reformule sa demande comme illustrée dans le dialogue ci-dessous, une bonne hypothèse pourra être validée par l'utilisateur après plusieurs tentatives.

### Dialogue 6.4.5.

**Humain** Send an email with the definition of Paris to peter.dominey@inserm.fr.

**Agent** Did you mean : Send an email to peter.dominey@inserm.fr ?

**Humain** no.

**Agent** Did you mean : Send the definition of Paris to peter.dominey@inserm.fr ?

**Humain** yes.

De la même façon, une nouvelle construction de la forme « *Send an email with the definition of X to Y* » pourra être dérivée de la construction « *Send the definition of X to Y* ».

## 6.5 Conclusion

Bien que l’agent virtuel puisse être instruit par l’utilisateur pour l’accommoder à son phrasé, dans la pratique, la variabilité linguistique est trop importante pour ne pas nuire à l’utilisabilité du système. Les travaux réalisés dans le domaine de la similarité sémantique nous ont apporté des méthodes afin d’accomplir automatiquement le rapprochement sémantique des énoncés inconnus et connus de l’agent. Nous avons développé un module basé sur ces méthodes, qui une fois intégré à l’agent, permet de bénéficier de la généralisation sur la forme des énoncés mais également de la généralisation opérée par l’agent sur les arguments des procédures.

Nos premiers résultats concernant l’efficacité du module de similarité sémantique sont encourageants bien que nous manquions encore de données pour réaliser des tests de robustesses plus aboutis. Son utilisation dans le cas de la détection d’arguments s’avère convaincante au vu de nos tests pratiques avec l’agent. Cependant, les méthodes utilisées impliquent quelques limitations. Premièrement, elles reposent sur des données difficiles à obtenir. Par exemple, le jeu de données utilisé lors de la compétition SemEval est en langue anglaise, le transposer dans d’autres langues peut être coûteux, là où l’agent virtuel ne dépend d’aucune langue spécifique. Deuxièmement, pour obtenir de meilleurs résultats, ces données devraient être adaptées pour un domaine particulier, cette adaptation requiert de développer des bases de données spécifiques pour chaque domaine.

Les résultats lors de sessions avec l’agent montrent également un apport fonctionnel du module. En effet, bien que les dialogues générés avec le module soient de taille égale ou supérieure en nombre d’interactions à ceux sans le module, la proactivité de l’agent rend le dialogue plus simple du point de vue de l’utilisateur. Ceci pourrait encore être amélioré dans le cas où l’agent n’aurait pas à faire valider chaque rapprochement sémantique par l’utilisateur. Comme expliqué au cours du chapitre, la cause est que l’agent n’est pas tolérant aux erreurs qu’introduirait une validation automatique. Cependant, les données liées au module de la similarité sémantique possèdent une structure légère, une paire de phrases et un score, qui pourraient être facilement partagées par un grand nombre d’utilisateurs et où chacun pourrait bénéficier de l’expérience des autres.

# Chapitre 7

## Implémentation et infrastructure

Ce chapitre présente les outils et les technologies utilisés lors du développement de la plateforme agent. Il permet également de mettre en lumière de façon concrètes les avantages obtenus par notre approche ainsi que les conséquences de certaines simplifications que nous avons dues prendre. Nous commençons par expliquer brièvement nos choix technologiques, puis, dans les grandes lignes, la structuration du logiciel et son implémentation. Nous expliquons par la suite les étapes nécessaires pour étendre les fonctionnalités de la plateforme avec de nouveaux services et permettre ainsi d'évaluer le logiciel pour d'autres domaines que les e-mails et la recherche Wikipédia.

### 7.1 Choix technologiques

L'un des objectifs de la thèse était de créer un prototype qui puisse mettre en œuvre l'agent virtuel présenté dans le Chapitre 5. Il s'agissait également de proposer une solution qui puisse évoluer vers une adaptation industrielle. Cette section présente les principaux concepts et technologies mis en œuvre dans le projet.

Le principale langage de programmation utilisé est Java. Il dispose d'une forte implantation industrielle et est particulièrement indiqué pour le développement de systèmes larges ou complexes nécessitant de nombreuses contributions. Nous l'avons conjointement utilisé avec OSGI, qui est une spécification et un ensemble d'outils permettant la réalisation d'applications modulaires.

La modularisation a joué un rôle important dans le développement du prototype. Premièrement parce que de nombreuses fonctionnalités sont en jeux : un environnement numérique, un analyseur pour le langage naturel, un gestionnaire de dialogue, un moteur d'exécutions d'actions et d'apprentissage de procédures, la reconnaissance et la synthèse vocale, etc. Or, le développement du prototype a été principalement guidé par notre objectif d'illustrer l'apprentissage de constructions et leurs utilisations sans pour autant disposer à priori d'un cadre architectural. En conséquence, des évolutions de bas niveau ont dû être appliquées au cours des différentes itérations, comme illustré dans

le Chapitre 6. Ces évolutions ont été facilitées par le confinement des implémentations et l'emploi de standards. Deuxièmement, elle permet d'obtenir des modules très peu couplés facilitant ainsi leur réutilisation pour d'autres projets ou une adaptation industrielle.

## 7.2 Structuration du projet

Le prototype se compose d'une plateforme agent et d'un ensemble de programmes clients. La plateforme agent permet d'exécuter plusieurs agents virtuels en concurrence et où chaque utilisateur peut s'y connecter par l'intermédiaire des programmes clients. La plateforme contient l'implémentation de tous les modules décrits dans le Chapitre 5 ainsi que des modules auxiliaires qui permettent de gérer les connexions des utilisateurs, la reconnaissance et la synthèse vocale, le cycle de vie des agents, etc.

Les programmes clients sont destinés à être exécutés sur le poste des utilisateurs. Ils permettent de capturer l'environnement numérique des utilisateurs et de communiquer avec les agents. Plusieurs programmes clients ont été développés en s'appuyant sur un système de communication générique, illustré par la Figure 7.1 :

**Application Rest** Une application cliente en Java permettant de discuter en langage naturel avec un agent et de représenter un environnement composé de services Rest qui peuvent être exécutés par l'utilisateur et par l'agent.

**Diagnosis** Diagnosis est un système de gestion de connaissances développé par la société Cloud Temple. Il permet aux utilisateurs de décrire et de documenter la réalisation de procédures métiers. L'une des interfaces du système se compose d'une application Web qui permet aux utilisateurs d'exécuter des procédures métier en ligne de commande ou par des formulaires HTML générés automatiquement. Une extension a été créée pour permettre à un agent d'observer les commandes entrées par l'utilisateur et leurs résultats dans le but que les utilisateurs puissent ensuite invoquer leurs exécutions en langage naturel.

**Xmpp** Un connecteur Xmpp a été développé pour permettre à n'importe quelle application cliente supportant le protocole Xmpp de communiquer avec un agent. Ainsi un utilisateur peut communiquer avec son agent depuis un smartphone comme illustré dans les vidéos de démonstration de la Section 5.8.

**WebBot** Enfin, une extension a été développée en utilisant le langage *Javascript* pour le navigateur *Chrome*. Il permet à l'utilisateur de discuter avec son agent soit par la voix, soit par des messages textuels. De plus, l'extension *Chrome* permet au système de capturer l'environnement Web de l'utilisateur et de contrôler les actions sur des pages Web. Les possibilités d'extension de ce programme client sont décrites dans la section suivante.

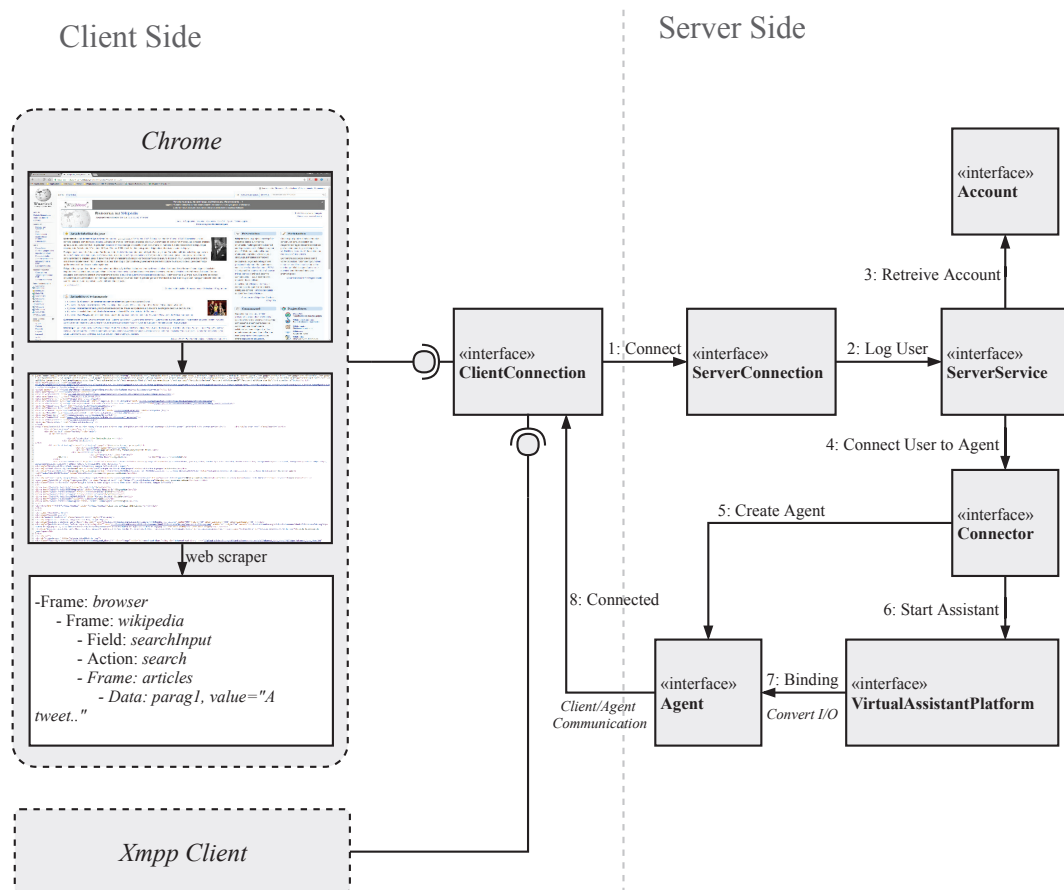


FIGURE 7.1 – Le système a été développé comme une collection de modules qui permettent de facilement adapter notre infrastructure à des protocoles de communication différents (Xmpp, Websocket, ...) et aux entrées possibles d'une plateforme agents (langage naturel, ligne de commande, GUI, son, etc). Ici, nous illustrons comment un utilisateur peut se connecter à son agent personnel depuis une extension Chrome ou une application de messagerie Xmpp.

## 7.3 Étendre le système avec de nouveaux services

Cette section décrit comment de nouveaux services Web peuvent être ajoutés au programme client WebBot. Ce programme permet à l'agent de voir et de contrôler les pages Web dans le navigateur de l'utilisateur. Pour que celui-ci puisse interpréter les pages, le contenu HTML doit être converti dans le modèle de l'environnement, décrit dans le Chapitre 5, qui se compose de *frames*, de *fields*, d'*actions* et de *data*.

Le modèle est suffisamment générique et a permis de représenter les différents environnements présentés dans la section précédente (requête REST, ligne de commande, GUI). Cependant, il a été conçu pour simplifier la tâche de l'agent et ainsi éviter à l'agent d'établir lui-même le lien entre ses actions primitives et les événements qu'il perçoit de l'environnement. Pour cela, chaque action primitive engendre un unique événement correspondant, facilitant l'apprentissage par démonstration, par exemple l'action de remplir un champs *fill* engendre l'événement *fill*. Ainsi lorsqu'il perçoit l'événement *fill* de l'utilisateur il peut facilement établir le lien avec ses propres actions primitives. Une alternative aurait été que plusieurs actions primitives soient nécessaires pour produire un événement observable, par exemple une action *sélection* pour sélectionner un élément et des actions *keyboards* pour écrire des lettres qui produisent l'événement *fill*. Dans ce cas de figure, l'apprentissage par démonstration aurait nécessité une étape supplémentaire dans laquelle l'agent doit apprendre de lui-même à faire la correspondance entre ce qu'il perçoit et ses actions.

En conséquence, l'environnement doit respecter cette contrainte, ce qui dans le cas d'un environnement contraint comme l'est un navigateur Web peut compliquer la tâche de conversion. Ainsi pour être en mesure de supporter de nouveau site web, une fichier de configuration doit être spécialement conçu pour établir le lien entre les éléments DOM de la page HTML et leurs événements avec le modèle de l'environnement. Un exemple de fichier de configuration est montré dans le Listing 7.1. Les éléments importants du fichier sont les « *selectors* » qui possèdent les deux propriétés :

**selector** Correspond au Xpath d'un élément HTML.

**id** Correspond à un élément du modèle de l'environnement qui doit être construit pour représenter l'élément DOM ciblé par le *selector*. Sa structure est de la forme « type » :« label ».

```
{
  "startUrl": 'https://en.wikipedia.org/wiki/Main_Page',
  "selectors": [
    {
      "parentSelectors": [ '_root', 'action:Perform' ],
      "type": 'SelectorElement',
```



```
    "id": 'frame:Search ',
    "selector": 'div#p-search ',
  },
  {
    "parentSelectors": [ 'frame:Search ' ],
    "type": 'SelectorText ',
    "id": 'field:Your query ',
    "selector": 'input#searchInput ',
  },
  {
    "parentSelectors": [ 'frame:Search ' ],
    "type": 'SelectorLink ',
    "id": 'action:Perform ',
    "selector": 'input.searchButton ',
  },
  {
    "parentSelectors": [ 'action:Perform ' ],
    "type": 'SelectorText ',
    "id": 'data:Para 1 ',
    "selector": 'div.mw-parser-output > p:nth-of-type(1) ',
  },
  {
    "parentSelectors": [ 'action:Perform ' ],
    "type": 'SelectorText ',
    "id": 'data:Para2 ',
    "selector": 'p:nth-of-type(2) ',
  }
]
}
```

Listing 7.1 – Exemple simplifié de configuration du service Wikipedia

Ce fichier de configuration peut être réalisé à l'aide d'outils graphiques pour faciliter la conception. Nous avons créé trois fichiers de configuration pour les besoins de démonstration, pour Gmail, Wikipédia et Airbnb.

# Chapitre 8

## Discussion et conclusion

Ce chapitre conclut la dissertation avec un bref résumé des travaux de recherche et une mise en perspective avec l'état de l'art actuel des systèmes cognitifs apprenants et des interfaces dialogiques Homme – Machine. Nous finissons avec une présentation des futurs travaux envisagés dans le cadre du projet.

### 8.1 Résumé

Cette recherche s'inscrit dans un contexte industriel où la possibilité de concevoir des systèmes adaptatifs aux besoins des utilisateurs et qui puissent être mis à l'échelle sur un grand nombre de fonctionnalités est de plus en plus demandée.

Le projet consiste en la création d'un système qui puisse être instruit par un utilisateur en langage naturel, et qui ensuite, puisse assister l'utilisateur dans la réalisation de ses tâches quotidiennes sur un environnement numérique.

Ma contribution a été de développer une approche du traitement du langage naturel qui ne sacrifie pas l'adaptabilité du système. Elle repose sur des techniques développées dans le domaine de la robotique développementale et d'études sur le développement du langage chez l'enfant humain. Le prototype que j'ai développé permet à un utilisateur d'enseigner en langage naturel à réaliser des actions sur un environnement numérique, de les réutiliser et de les combiner sans qu'un modèle sémantique préétabli ne soit fourni au système. Quoique l'implémentation reste simple, le système présente des intérêts immédiats pour l'utilisateur, comme la possibilité de réaliser des tâches qui nécessitent l'interaction de plusieurs applications. Le système est ainsi engageant pour l'utilisateur et permet de collecter des données difficiles à obtenir, comme des connaissances procédurales, dès les premiers instants d'utilisation. L'adaptabilité à la variabilité linguistique est un exemple d'utilisation de ces données qui enrichit le système. D'autres techniques permettant d'exploiter l'expérience de l'assistant et de l'enrichir en fonctionnalités sont en cours de développement dans la continuité de ce travail.

Ainsi, nous avons présenté un système dans lequel un utilisateur final est capable d'enseigner à un assistant virtuel comment faire fonctionner un ensemble

de services numériques arbitraires en utilisant le langage naturel. L'assistant virtuel utilise la structure inhérente aux commandes en langage naturel de l'utilisateur pour organiser les éléments perceptuels primitifs en procédures. Cela permet à l'assistant de s'adapter aux différents services et à la variabilité des choix des utilisateurs quant à l'utilisation de la langue. L'apprentissage a d'abord été utilisé pour apprendre à ancrer les énoncés du langage naturel dans des commandes perceptuelles et motrices indépendantes du domaine afin d'exécuter des actions de base (par exemple, ouvrir *Gmail*, créer un brouillon de courrier, définir le sujet, etc). Sur la base de ces constructions apprises, l'assistant virtuel a ensuite appris à exécuter des tâches plus complexes, caractérisées par la composition de ces constructions précédemment apprises et les procédures correspondantes, avec le passage d'argument entre elles (par exemple, le terme de recherche de *Gmail* étant passé à *Wikipedia*, et la définition résultante étant renvoyée à *Gmail*). Ainsi, il est démontré que les procédures apprises sont utilisées dans une définition récursive ou compositionnelle d'une nouvelle procédure comme « envoyer un courriel à pfdominey@gmail.com au sujet de la programmation avec la définition de python », que l'utilisateur peut expliquer en langage naturel en décomposant dans la procédure « envoyer un courriel », et une autre procédure apprise qui obtient des définitions de *Wikipedia*. La capacité d'ancrer les commandes dans des actions primitives par l'observation d'un utilisateur sur son interface graphique, puis la capacité de composer ces commandes purement par le langage illustre comment des mécanismes récursifs simples pour créer des structures étiquetées fournissent un puissant mécanisme de composition pour spécifier des interactions complexes avec le monde (Goucha, Zaccarella et Friederici 2017).

Nous montrons qu'en prenant une perspective développementale le système obtenu présente également plusieurs avantages. Cette conception permet d'éviter la nécessité d'une ingénierie des connaissances inhérente à une langue et à un domaine spécifique, tout en offrant une capacité d'apprentissage substantielle. Dans ce contexte, d'autres systèmes offrent la possibilité pour un utilisateur d'enseigner à un assistant virtuel, mais avec l'exigence que l'interaction avec différents domaines (e-mail, moteur de recherche, etc.) doit être pré-spécifiée. Le système est plus résilient aux défauts présents dans les phrases de l'utilisateur tels que les mots rares ou spécifiques à un domaine non présents dans le lexique, des mots mal orthographiés ou des entités nommées inconnues. Et enfin l'utilisateur peut obtenir une meilleure appréhension des limites du système. Par l'expérience partagée qu'il a avec son assistant, il connaît les demandes qui seront correctement interprétées. Dans le cas contraire, il a la possibilité d'expliquer comment les réaliser dans le même flux d'interactions. Ceci s'oppose aux systèmes qui rompent l'interaction en exécutant une commande par défaut, comme une recherche sur un moteur de recherche.

## 8.2 Pertinence de l'approche

Sun et al (Sun, Chen et Rudnicky 2016) ont investigué comment des agents virtuels pouvaient assister les utilisateurs à réaliser des actions de haut niveau

comme « *help me plan an evening out with my friends* » lorsque plusieurs applications distinctes sont impliquées (par exemple : Yelp, OpenTable et Messenger). L'utilité de tels agents est que les utilisateurs ont souvent le besoin de réaliser ce genre d'actions complexes mais que la plupart des assistants actuels sont conçus pour être performants dans un domaine spécifique. Notre exemple d'intégration du service Gmail avec Wikipédia, qui reste encore simple, est une première démonstration de ce but. Tandis que le système de Sun nécessite une phase dans laquelle le langage et les actions sont collectés avant que le système ne puisse fournir son rôle premier, notre système permet à l'utilisateur de lier le langage aux actions pendant son utilisation.

Dans le contexte du développement du langage, notre système fait face à un problème similaire à celui de l'enfant humain : la manière dont la relation entre la structure argumentative du langage et la structure des événements liés à une action est apprise. On peut considérer ce point dans le contexte du système d'analyse perceptuel du sens (PAM) de Mandler. Ce système produit un ensemble de représentations primitives comme des chemins, des déplacements ou des relations spatiales à partir de données sensorimotrices (Mandler 2012). Ces représentations servent de base pour interpréter les événements qui sont observés et sur lesquels le langage s'ancrent et les enrichissent de structures additionnelles (Mandler 2012). D'une façon similaire, notre module *Sensor-Process* produit ces représentations primitives (*fill*, *execute*, *focus* et *retain*) à partir de l'interaction de l'utilisateur sur une page Web. Ces représentations servent de base au module *InterpreterProcess* pour créer des constructions grammaticales qui lient ces primitives à la phrase de l'utilisateur. Ainsi, il n'y a pas de représentations pré-programmées sous forme de prédicats au sein du système. La structure argumentative de la phrase est directement liée à la représentation des actions résultantes. Cette représentation est une spécification préliminaire de comment la structure argumentative peut émerger.

Pour Tomasello, un point clef de la cognition humaine est sa capacité à construire des structures compositionnelles qui peuvent être réutilisées pour construire à nouveaux des structures de plus haut niveau (Tomasello 2009; Goucha, Zaccarella et Friederici 2017). La recherche actuelle fait progresser l'état de l'art dans le domaine des assistants intelligents en fournissant une capacité d'apprentissage qui permet de construire une structure compositionnelle labellisée. Il est intéressant de noter que cette capacité a été identifiée comme étant au cœur de la capacité humaine à traiter le langage (Goucha, Zaccarella et Friederici 2017). Ainsi, une méthode a été mise en place pour créer de nouveaux éléments à partir d'éléments existants de façon récursive et où chaque éléments peut être étiqueté. La capacité d'étiqueter ces nouveaux éléments permet de les réutiliser et d'aller et venir entre une représentation interne et un acte de langage (Goucha, Zaccarella et Friederici 2017). La méthode employée est simple, mais a permis d'illustrer l'intérêt de cette capacité linguistique humaine pour la conception de système « intelligent ».

## 8.3 Futurs travaux

Deux axes peuvent être entrepris dans la continuité de ce travail. Le premier consiste à rendre le système plus utile dès les premiers instants d'utilisation et à collecter d'autres types de données. Cela se traduit en continuant l'approche que nous avons entreprise dans cette thèse et d'ajouter des fonctionnalités comme la possibilité de gérer des structures conditionnelles (du type *if... then...*) ou des structures itératives (*while... do*). Mais également, la capacité de générer des constructions narratives qui permettent à l'assistant virtuel d'apprendre à comprendre et à raconter une séquence d'événements possédant des relations de dépendance, notamment des relations de causalités. Ce travail pourra être adapté de travaux connexes en robotique (Mealier et al. 2017). En effet, la capacité du récit à imposer une structure supplémentaire aux représentations de l'expérience humaine est cruciale dans la capacité humaine à représenter une réalité partagée avec d'autres (J. S. Bruner 1990 ; J. Bruner 1991), et jouera un rôle important dans le développement futur des assistants adaptatifs.

Un second axe est celui de l'exploitation des données collectées pour rendre l'agent plus flexible et offrir plus d'autonomie dans son apprentissage. Cette recherche peut être située dans le contexte plus large de l'apprentissage automatique et du transfert de connaissances humaines vers ces algorithmes. La Figure 8.1 est basée sur le framework développé par Oudeyer et ses collègues (Oudeyer 2012) pour caractériser comment un système peut exploiter des démonstrations humaines dans le but d'apprendre de nouvelles politiques pour réaliser une tâche.

Ainsi, une partie des tâches peut être apprise en autonomie par le système tout en gardant la possibilité d'être guidé par un humain. Cette technique peut par exemple être intéressante pour rendre notre représentation de l'environnement plus flexible comme expliqué dans le Chapitre 7. La perception et les actions n'auraient plus besoin d'être en relation directe (observer des pixels et agir sur un clavier).

Ce projet représente un point départ pour d'importants futurs travaux de recherche. Cela représente la continuité entre l'apprentissage du langage basé sur l'usage dans le domaine de la robotique vers de futures recherches en apprentissage automatique. Comme un premier terrain d'expérimentation, nous étendrons le présent système pour inclure une capacité d'exploration active intrinsèque.

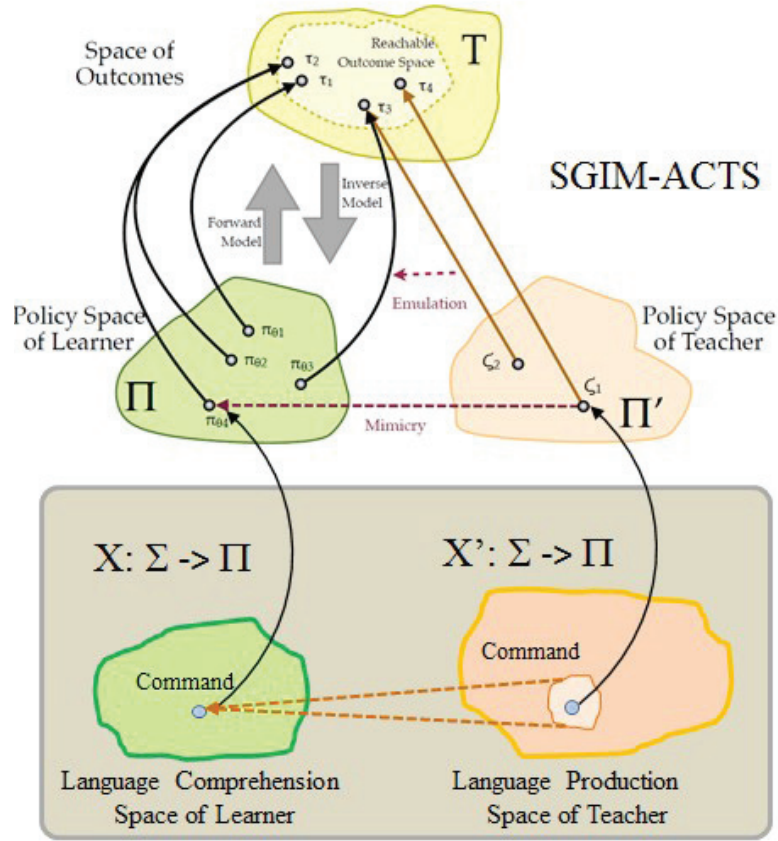


FIGURE 8.1 – La figure illustre l’espace des politiques (« *know how* ») du système sur la gauche et celui de l’humain sur la droite. Le système peut choisir de soit copier la démonstration elle-même – mimétisme, ou d’en copier le résultat – émulation. Dans notre implémentation courante, notre système utilise le mimétisme. Nous avons étendu la figure de Nguyen et Oudeyer (Oudeyer 2012) qui inclut un espace de langage pour l’humain et le système et les liens entre le langage et une politique. Dans cette version étendue, l’humain produit une commande (en bas à droite) qui est transmise à l’apprenant. Si la commande est reconnue la politique correspondante est exécutée. Dans le cas contraire, l’apprentissage par mimétisme est invoqué. La variabilité linguistique est représentée par l’espace autour de l’espace de langage de l’enseignant, qui ensuite converge vers un énoncé dans l’espace de compréhension.

## Annexe A

Diagrammes de séquence :  
« What is a tweet ? »

## A.1 Déclenchement d'une procédure

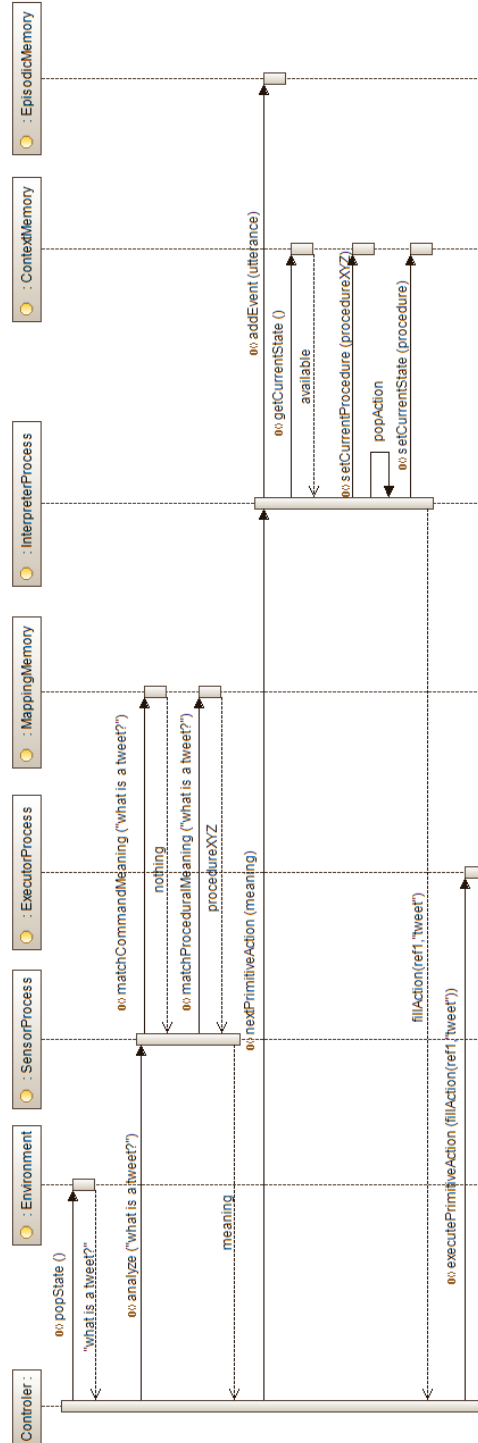


FIGURE A.1 – Diagramme de séquence illustrant l'interprétation de la commande utilisateur « *What is a tweet ?* ». Dans cette étape, la phrase est liée à une procédure précédemment apprise et l'assistant commence à l'exécuter.



## A.2 Réalisation d'une action

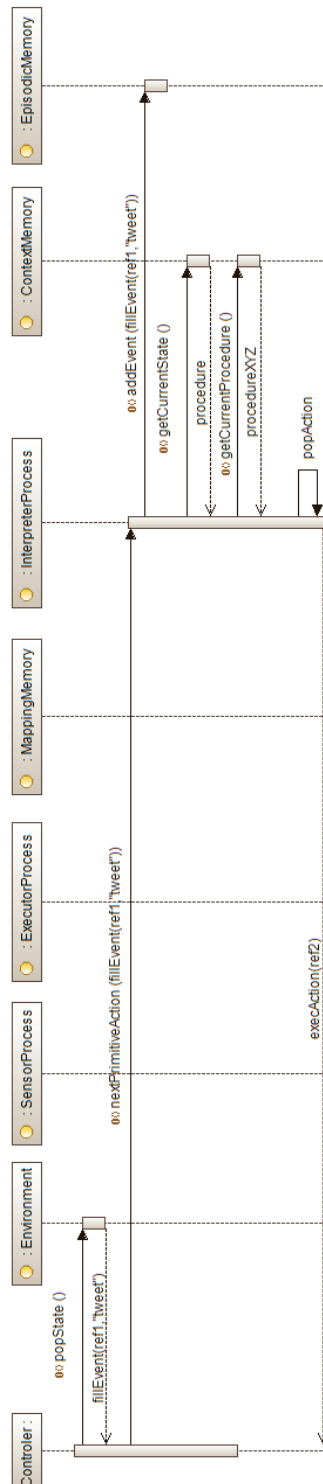


FIGURE A.2 – Diagramme de séquence illustrant l'interprétation de la commande utilisateur « *What is a tweet?* ». Dans cette étape, l'assistant attend de percevoir le résultat de sa première action, remplir un champ, avant de déclencher l'action suivante : lancer la recherche.

### A.3 Prise en compte du résultat d'une action

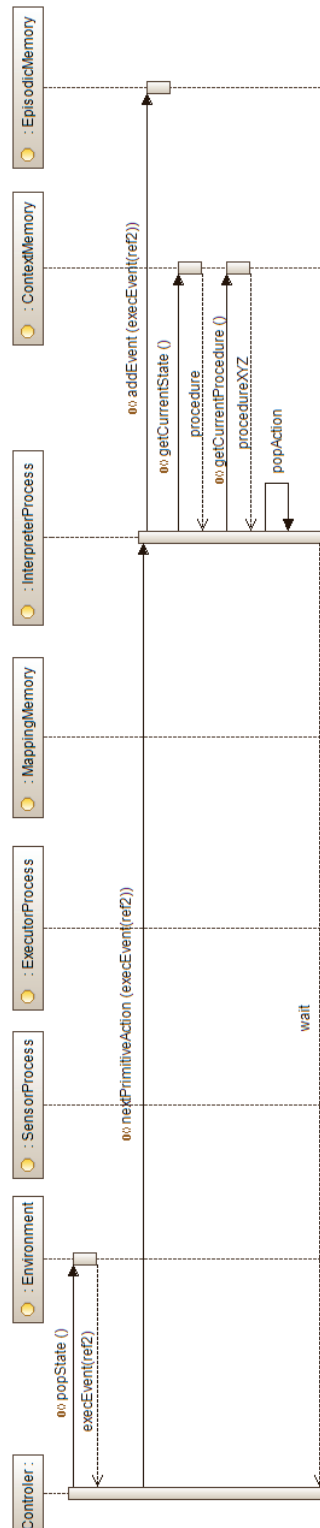


FIGURE A.3 – Diagramme de séquence illustrant l'interprétation de la commande utilisateur « *What is a tweet ?* ». À cette étape, l'assistant attend que la page Wikipédia affiche le résultat de la recherche.

## A.4 Réponse à l'utilisateur

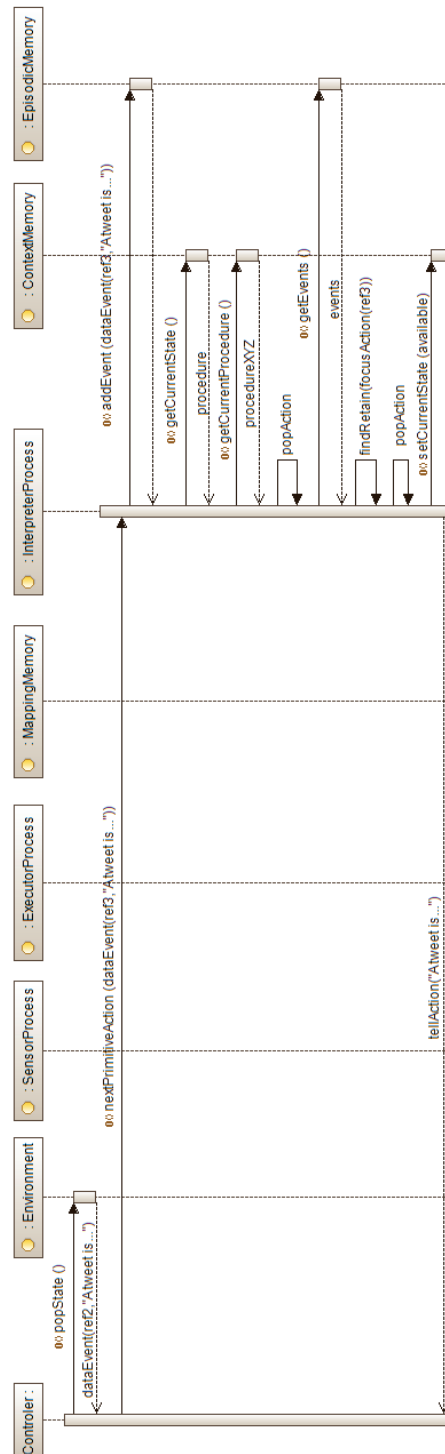


FIGURE A.4 – Diagramme de séquence illustrant l'interprétation de la commande utilisateur « *What is a tweet ?* ». Ici, l'assistant prend le premier paragraphe de la page Wikipédia et l'envoie à l'utilisateur via le *chat*.

## Annexe B

### Article : Usage-Based Learning in Human Interaction with an Adaptive Virtual Assistant

# Usage-Based Learning in Human Interaction with an Adaptive Virtual Assistant

Clément Delgrange, Jean-Michel Dussoux, Peter Ford Dominey

**Abstract**—Today users can interact with popular virtual assistants such as Siri to accomplish their tasks on a digital environment. In these systems, links between natural language requests and their concrete realizations are specified at the conception phase. A more adaptive approach would be to allow the user to provide natural language instructions or demonstrations when a task is unknown by the assistant. An adaptive solution should allow the virtual assistant to operate a much larger digital environment composed of multiple application domains and providers and better match user needs. We have previously developed robotic systems, inspired by human language developmental studies, that provide such a usage-based adaptive capacity. Here we extend this approach to human interaction with a virtual assistant that can first learn the mapping between verbal commands and basic action semantics of a specific domain. Then, it can learn higher level mapping by combining previously learned procedural knowledge in interaction with the user. The flexibility of the system is demonstrated as the virtual assistant can learn actions in a new domains (Email, Wikipedia,...), and can then learn how email and Wikipedia basic procedures can be combined to form hybrid procedural knowledge.

**Index Terms**—human-system interfaces, intelligent assistant, language, usage-based learning.

## I. INTRODUCTION

NATURAL language is a concise and effective way to interact with software and cloud services. Natural-language interfaces have demonstrated their usefulness in areas such as information extraction from databases [1], robot control [2] or in the execution of daily tasks through companions like Google Now, Siri or Cortana [3]. A limitation of these systems is the inability to benefit from user's knowledge when dealing with unexpected situations. Such situations could be a speech input requesting an action unknown to the system, an unknown lexical unit or the inability to generate a valid course of actions. To overcome such limitations, a system should have the opportunity to learn from its own experience and from external sources by the means of natural language instructions and perceptual demonstrations. We determined that this objective can be achieved by taking advantage of concepts from usage-based learning in human language acquisition.

Our objective is to provide a method that allows the user maximum flexibility in teaching new behaviors to the assistant using natural language by minimizing the need of specific knowledge engineering of task domains or languages.

We first review related works in the domain of virtual assistants that exhibit the need to design systems that are

adaptive to the evolution of the functionalities of a digital environment, in term of application domain and tasks, and to the changing user needs.

We then present robotic works inspired by developmental studies on which our current work is grounded. Here, we do not attempt to duplicate usage-based learning and generalization of grammatical constructions as in the human, but rather, we exploit three particular aspects of usage-based learning of grammatical constructions, and we then apply this to an intelligent assistant. First, we adopt the concept that language is a collection of form-to-meaning mappings [4] where in our case form is the natural language command from the user and meaning is the demonstrated execution. Second, we adopt the notion of abstraction over arguments, where a given construction can be re-used over a set of different arguments. Finally, perhaps most importantly, we adopt the generative capacity for compositionality, where new constructions can be composed of existing ones, inheriting their argument structure.

In order to evaluate our proof-of-concept system we provide a set of use cases transcriptions and an analysis of the number of interactions needed to complete a set of tasks.

We finally discuss how state-of-the-art machine learning techniques could improve the generalization of natural language user utterances to compete with actual virtual assistants and lead, then, to a realistic user study.

## A. Domain Adaptive Assistant

User tasks generally require the use of multiple applications, for example to plan an evening out with friends that require to interact with a messaging service, a calendar and a search engine. One task the user would want to achieve is to send the list of the nearest pubs to his friend and the virtual assistant should so be able to interact with multiple application providers and of different domains.

Sun et al [5] investigate how a future agent could organize activity across multiple applications (e.g. Yelp, OpenTable and Messenger) in order to achieve a high level goal like “help me plan an evening out with my friends”. Here, the association between user intentions and the use of specific applications are learned thanks to user traces on a smartphone. The value of such an agent is that it can allow the user to build their own virtual assistant based on their favorite applications without the need for each provider to share a common semantic. The long term goal of such agents is to operate a larger digital environment composed of a larger number of possible tasks.

C. Delgrange and P.F. Dominey are with the Stem Cell and Brain Research Institute, Inserm U1208, 18 Avenue Doyen Lépine, 69500-Bron, France.

J.M. Dussoux is with Cloud Temple, 215 Avenue Georges Clemenceau, 92000 Nanterre, France.

### B. User-Task adaptive assistant

Another aspect of adaptation is to allow the user to define the interpretation of a natural language order. This can be done by building instructable agents where a user can teach to the assistant how to accomplish a task by demonstrations or instructions at runtime and relies on one shot learning techniques.

1) *PLOW*: The *PLOW* system, developed by James Allen [6], allows the user to teach new tasks to an assistant based on GUIs demonstration. The range of problems covered by *PLOW* is broader than procedural knowledge acquisition and includes a set of modules allowing for example to emulate deliberative dialogues. Language understanding is performed by converting user natural language sentences into logical forms [7] and relies on the *TRIPS* system for dialogue management and task planning [8].

While the *PLOW* system can learn new task with the user, a number of domain specific competences must be engineered for the system to work. This makes the system difficult to maintain as the number of applications increases.

2) *LIA*: *LIA* is another example of an instructable agent that users can teach to perform new action sequences to achieve new commands using solely natural language [9]. *LIA* is composed of two parts: a semantic parser which assigns executable semantics to each learned natural language command and a back-end which executes these commands. The back-end contains a set of primitive functions for a particular domain along with their corresponding logical form. *LIA* use a Combinatory Categorical Grammar (CCG) to parse natural language utterances. The parser must first be trained to associate natural language lexicon with the set of primitive functions. The system can then derive more complex logical forms from user inputs.

While the *LIA* system has a lighter infrastructure than *PLOW*, it requires the use of predefined high-level primitives, like *sendEmail*. This prevents the user to use the same kinds of one shot learning technique to accommodate the virtual assistant to new domains or new applications (extending the back-end in *LIA*).

## II. USAGE-BASED LANGUAGE UNDERSTANDING

Here we consider how these limitations can be addressed by taking inspiration from usage-based learning.

Usage-based learning is a social-pragmatic approach to language acquisition in which children learn linguistic structures through understanding intentions and finding patterns in their discourse interactions with others [10]. From the outset of life, children and adults interact in feeding, changing diapers, getting ready for bed, etc. in repeating rituals that are accompanied by language from the adults [11]. This provides a rich shared space where language can enrich meaning at multiple levels [12]. In the usage-based concept of language acquisition, the child's first words allow reference to people and objects that are of central prominence in everyday life [10], [11]. After words, the first grammatical constructions are fixed around specific verbs, and specific actions that tend to be repeated and ritualized in the infants' social environment [10], [13], [14].

Constructions then become more abstract and generalized as the child's world becomes enriched [4], [10].

This learning requires access to perceptual and motor meaning via mechanisms that have been characterized in the context of developmental robotics [15]. The essential idea is that primitive innate perceptual and motor capabilities can be composed into more complex percepts or behaviors, and that these structured representations become associated with language [16]. We have previously applied this methodology of structuring perceptual-motor primitives around language in the context of human-robot cooperative interaction [17]–[21]. These works concentrate on three major aspects:

- 1) The existence of symbolic primitives grounded in the perceptual system.
- 2) Grounding the language in these primitives.
- 3) Learning the use of natural language by engaging the robot into cooperative activities with a human.

Our objective is to extend this work into the domain of cooperative interaction with a virtual assistant. Part of the motivation is to allow the assistant to acquire human expertise through cooperative interaction with humans.

### A. Symbolic Primitives

1) *Developmental Theory*: In the context of language development in human, a fundamental question is how the symbolic structure of language is grounded into the perceptual representation system.

Mandler established the existence of symbolic primitive structures [22], called schema. Schemas are described as a simplification of the information contained in the perceptual system. They are learned early by the infant through a process, called Perceptual Meaning Analysis (PMA), that maps perceptual representation with a set of early developed structures such as *PATH*, *MOTION* or *CONTAINMENT* [23]. Inputs to the PMA are temporal aspects of objects moving along paths, “into or out of containers, objects going into or out of sight, and contingencies between paths, such as one object chasing another” [24]. The PMA generates representations that are the interpretation or construal of the events that are being observed. These image schemas can then be further enriched by language, which can add additional structure that the PMA cannot supply [24].

The use of perceptual primitives such as *MOTION* and *CONTACT* do not assume any form of innate predicate structure, but instead, they are the base which allow a symbolic grounding of language or images.

2) *Application in Robotic*: As depicted in Figure 1, robots share the same space as humans and one task is to convert information extracted from a camera into a meaning structure linked to natural language.

Perceptual robotics is a research domain in which robots are equipped with perceptual systems that can produce internal representations of the physical world, sensory systems provide a re-coding or representation of the world that can provide the meaning component for this grounding. Thus, in Dominey and Boucher [26], a computer vision system recognized coloured objects and provided outputs that could

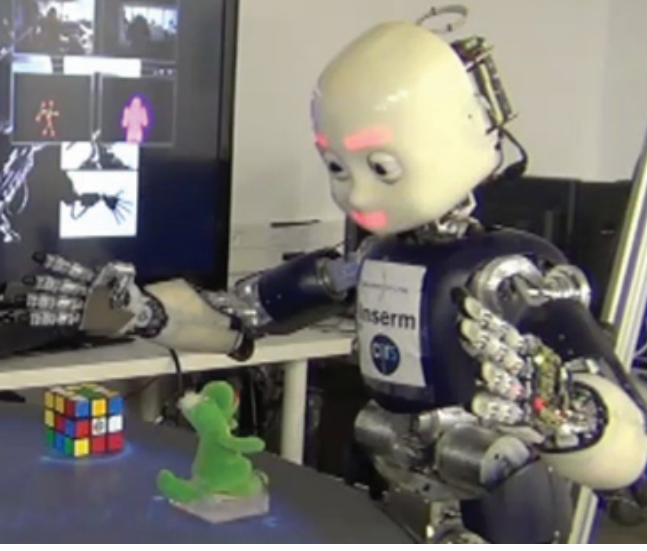


Fig. 1. iCub in front of different objects. Extracted from Mealier et al. [25]

be used to ground the meaning of words in sentences. That is, the words could be learned to refer to the outputs produced by the visual system. Interestingly, this grounding applied to nouns and verbs. Nouns, like “block”, “moon”, and “cylinder” were grounded in the objects identified by the visual system. The visual system also extracted dynamic properties of these objects, including the direction and speed of motion (related to Mandler’s MOTION), and whether objects were in contact (related to Mandler’s CONTACT). Interestingly we determined that actions like “push” and “give” could be defined in terms of sequences of perceptual primitives related to contact. We thus developed an event parser which could detect GIVE( $A, B, C$ ), meaning “A gives B to C” as the following sequence of primitive events: CONTACT( $A, B$ ), MOVE( $A$ ), MOVE( $B$ ), CONTACT( $B, C$ ), MOVE( $A$ ), END-CONTACT( $A, B$ ).

In the same manner, Lallec et al [27] developed a system that learned to recognize actions including COVER, UNCOVER, GIVE and TAKE, based on the perceptual primitives VISIBLE, MOVE and CONTACT. These were encoded as  $\langle \text{enabling state}, \text{action}, \text{resulting state} \rangle$  triplets. This research thus provides a demonstration of how basic meaning representation and higher level actions can be extracted from the perceptual stream, based on a initial set of simple perceptual primitives.

## B. Grammatical Construction: Grounding Language in Symbolic Primitives

1) *Developmental Theory*: In a constructionist approach, very limited initial perceptual and motor abilities form the primitives that are assembled and structured as part of the process of usage-based language learning. In this context, language is considered to consist of a structured inventory of mappings between the surface forms of utterances and meanings, referred to as grammatical constructions [4]. These mappings vary along a continuum of complexity. At one end are single words and fixed “holophrases” such as “Gimme that” that are processed as unparsed “holistic” items [10]. At

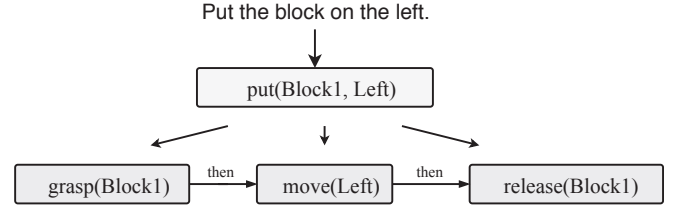


Fig. 2. Grammatical construction grounding: The sentence “Put the block on the left” is mapped on a predicate-argument structure which in turn is grounded on perceptual primitive schemas.

the other extreme are complex abstract argument constructions that allow the use of sentences like this one. In between are the workhorses of everyday language, abstract argument constructions that allow the expression of spatiotemporal events that are basic to human experience including active transitive (e.g., John took the car) and ditransitive (e.g., Mary gave my mom a new recipe) constructions [4].

The notion of constructions becoming more abstract is a prevalent notion in language acquisition in the construction grammar framework. Tomasello extensively develops this point in Chapter 5 – Abstract Syntactic Constructions, in *Constructing a Language: A Usage-Based Theory of Language Acquisition* [10]. The essential argument is that basic argument structure constructions are directly associated with dynamic, experientially grounded scene gestalts [28]. Then, through processes of abstraction, and pattern finding, analogy and distributional analysis, constructions become progressively more abstract. Gaspers et al [29] address how abstraction can take place in a usage-based computational model of language acquisition that is based on bootstrapping mechanisms that relies on discovered regularities to segment speech into word-like units. In the top-down aspects, based on this initial segmentation, the model induces a construction grammar that in turn acts as a top-down prior that guides the segmentation of new sentences into words. We proposed a model that first learned idiomatic holophrases that mapped directly onto meaning representations, then certain arguments became liberated in verb island constructions, and finally all open class elements become liberated in abstract argument constructions [30].

Figure 2 illustrates the nature of simple constructions as mappings from sentence form to a predicate-argument meaning representation grounded into primitive schemas, they are the form of construction mainly used in robotics. We then present more advanced tools for abstracting language.

2) *Application in Robotics*: A central issue in language learning is grounding: establishing the link between a word and its referent.

We initially studied these form-to-meaning mappings in the domain of perceptual robotics [31] using the perceptual capabilities described above in Section II-A. There, a perceptual system learned to map perceived event representations such as *push(Moon1, Cylinder1)* onto sentences such as “The moon pushed the cylinder”, or “The cylinder was pushed by the moon”. We then extended this approach to include both the description of perceived events, and the execution of spoken commands in the Cooperative Human Robot Interaction Sys-



tem (CHRIS) architecture [19].

For new actions (that have not yet been defined in the Knowledge Base) the system uses the set of observed primitives from Temporal Segmentation to generate a generic pattern of primitives to define the action. Thus, a new predicate *cover(x,y)* is learned as a pattern of primitives:  $\langle \text{moving}(x), \text{contact}(x, y), \neg \text{visible}(y), \neg \text{moving}(x) \rangle$  which is generated by the perceptual system. This then is the pattern onto which “Cover y with x” can be associated [27]. These actions were also associated with states. Thus, for example, after the action *cover(x,y)*, the resulting state is *contact(x,y), visible(x), on(x,y)*. By this same mechanism, Lalle et al [27] demonstrated 4 other complex predicate-argument actions, including uncover, take, give, and put. These actions could then be assembled into higher level constructs that we called shared plans, allowing the human and robot together to achieve a desired state. This research thus provides a framework for demonstration of how predicate-argument structure for high level actions like *cover*, *put*, etc. can emerge.

In this context of grounding and language learning, Cangelosi and colleagues [32]–[34] have developed a number of simulation and robotics experiments that demonstrate how word meaning can be grounded in perception for language learning. Their more recent work begins to look at grammatical structure in the case of verb-noun pairs that link to object-oriented action [33]. Indeed, it is likely that meaning is more pertinent in the context of goal oriented action schemas or frames [35]. The learning of grammatical constructions has been approached both from a language evolution perspective [36], and from a developmental or learning perspective [26], [31], [37]. An important common point is that they rely on some form of organization of meaning in terms of pragmatic frames, or repetitive goal-oriented action schemas.

3) *Models and Tools*: Powerful algorithms for grammar induction combine statistical processing for pattern extraction, and rule-based structural generalization based on these discovered patterns [38]. This results in unprecedented capabilities for extracting grammatical structure, without addressing the problem of how such structure maps onto meaning. In this context of form to meaning mappings, we initially investigated neural networks that learned to associate the pattern of closed class elements in a sentence with the ordered mapping of open class (semantic) words onto their respective thematic roles in the meaning [39]. We subsequently showed that with larger networks and training corpora, such systems could actually generalize to new constructions that were not used in the training data [40].

Gaspers and Cimiano [41] exploited cross situational statistics that link words to their referents across multiple scenes, in order to then build up progressively more elaborated grammatical constructions. Gaspers et al [29] then employ a powerful method where bottom up and top down learning mechanisms mutually reinforce each other in learning form to meaning mappings. An initial lexicon is formed bottom up from phonemic sequences that appear frequently enough to establish initial form-meaning mappings. These initial words can then be mapped into perceived sentences in order to define alignments between words in the sentence and their roles in

predicate argument representations of meanings, which allow identification of grammatical constructions. Once an initial set of constructions is learned, they can be used in a top down manner to identify the meanings of new words in sentences that correspond to these learned constructions. This allows for a synergistic interaction between bottom-up and top-down learning. We [26], [31], [42] exploited these two processes in where semantic bootstrapping provides initial word meaning which allows the learning of the mapping between lexical elements in the sentence and their corresponding role in the predicate-argument structure of the meaning. This results in learning of grammatical constructions that can be exploited in top-down syntactic bootstrapping, where novel words are more easily learned by using known constructions that directly map them to their referent in the meaning representation.

In the current research we exploit a direct mapping between the user’s top down spoken language input, which forms a pattern from the bottom-up demonstration to map onto. This is in line with a recent perspective proposed by McCauley and Christiansen [43]. They argue that in addition to distributional information, computational models of language learning should take more account of meaning, in terms of the agents, objects and actions that the language is referring to.

### C. Learning and Using Language in Cooperative Plans

1) *Developmental Theory*: A crucial aspect in the development of human cooperation is the ability for the child and adult to engage in a triadic interaction where the two agents interact around the third element – the object of their cooperative interaction. By 12-14 months of age, the child engages in these triadic interactions with the adult, and demonstrates that they are motivated to share experience with others, and importantly, to help them toward their goals [44]. Language allows the developing child to align shared plans with the adult, and to create new language-based representations that persist over time. In development, language is thus a tool for interacting minds, which allows the expansion of new spaces for interaction and cooperation [45].

Following Tomasello we consider cooperation as the case when two individuals establish a common goal, and develop a shared plan to achieve that goal [44]. In such cooperative scenarios, it is typically an adult “leader” that establishes the shared plan as a turn-taking sequence of actions by the two participants in order to achieve the shared goal.

2) *Application in Robotics*: The robotic CHRIS system demonstrates how to integrate the core features that consist of the ability to extract meaning from perception, to learn how to compose new actions from primitive and learned actions, and to cooperatively learn and execute shared plans or procedures with a human via a spoken language interaction [19]. As described above, extracting meaning from perception is performed using a set of primitives including MOTION and CONTACT, which allow recognition of actions like TAKE and GIVE in the perception module of the system. Then, in the grounding of a sentence like “put the block1 on the block2” in meaning, there is the transition from the sentence to a schematized representation of the meaning *put(block1, block2)* in the



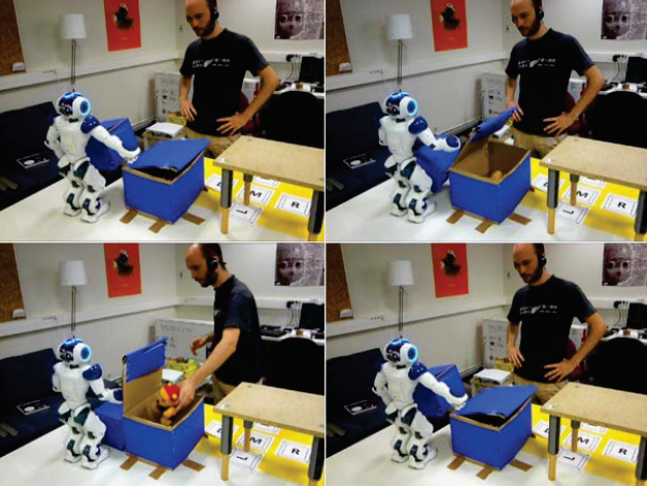


Fig. 3. Execution of a shared plan to put a toy into a box. From upper left to lower right, the robot opens the box, Gregoire puts the toy in the box, and finally the robot closes the box.

learned grammatical construction. This representation of the put action is then decomposed into primitive motor commands like GRASP, MOVE, RELEASE and into perceptual primitives like OBJECT1, OBJECT2 that are encoded in the knowledge base. The language processing and plan learning capabilities are provided by a planning and supervision module and allow a Human, either by natural language descriptions or by physical demonstrations, to specify at runtime how to link a linguistic unit such as “block1” to a perceptual element like OBJECT1 as well as the composition of  $put(Object1, Location1)$  into the sequence  $GRASP(Object1), MOVE(Location1), RELEASE$ .

We have used this mapping of sentences to robot action in order to allow the human user to specify shared plans in cooperative human-robot interaction [2], [19]. An example is illustrated in Figure 3, where the execution of a shared plan to put a toy into a box is elaborated. Gregoire has specified the shared plan as “I reach the toy, then I reach the trash box then you open the trash box then I put the toy in the trash box then you close the trash box”. In Figure 3 we see the successive execution of these steps. In that work, the user is also able to teach the robot how to do new actions, like opening and closing the trash box.

### III. TRANSFER TO THE DIGITAL UNIVERSE

As the CHRIS architecture demonstrates, a robot can learn how to speak about a perceptual entity like a human agent picking up an object [27] and how to participate in a cooperative game like uncovering a toy [20]. In the same way, a virtual assistant could learn how to recognize and speak about an email and how to cooperatively send an email.

In the following, we show how to adapt the techniques employed in the robotic domain to build a virtual assistant:

- 1) **Primitives:** We adapt the set of symbolic primitives to a digital world so that the system can observe the human actions on the GUI. We thus develop and exploit perceptual primitives that correspond to filling fields, clicking, highlighting, copying, pasting.

- 2) **Grounding:** We show how the language is linked to these primitives to compose new actions thanks to user demonstrations.
- 3) **Language and Cooperation:** We show how a user can teach the system to interpret compositional sentences using natural language instructions that refer to learned construction and how the system can generalize a procedure over a set of arguments.

#### A. Overall Functioning

The virtual assistant architecture, shown in Figure 4, is made up of the following components:

- 1) The *Environment* which renders the digital services in a model that the assistant can perceive and act upon. This includes a chat window.
- 2) The *SensorProcess* which converts percepts to meaningful representation thanks to the *ConstructionMemory*.
- 3) The *InterpreterProcess* which handles the interactions with a user in order to cooperatively execute procedures and to learn them. This uses the *ContextMemory* that is a working memory for the current tasks and the *EpisodicMemory* which stores event traces that occur within the platform. This is used in the one-shot learning to encode new constructions.
- 4) The *MotorProcess* which converts meaningful representations to executable actions in the environment.

In order to illustrate the information flow we first provide a general description of a usage scenario. This will then be explained in detail in the following sections:

A user requests to send an email in natural language, “send an email to peter.dominey@inserm.fr about the meeting and say that I will be early”. If the virtual assistant does not know this action, a learning behavior is activated. Then, the user can describe the steps that compose this action using either (a) natural language (using language constructions that have been previously learned), such as “create a draft”, “set the subject”, “write the content”, etc. or (b) by performing a demonstration on a graphical user interface that the virtual assistant can observe. The learning behavior stores all perceptual events coming from the *SensorProcess* including user utterances in the *EpisodicMemory*. At the end of the learning, an algorithm consolidates this knowledge from the *EpisodicMemory* and stores the resulting new construction in the *ConstructionMemory*. The new construction is a map from the language form to the execution meaning. When the user again requests the execution of this action “send an email to clement.delgrange@inserm.fr about the software and say that it functions perfectly”, the virtual assistant will match the utterance with the learned construction, extract the arguments contained in the user utterance and instantiate the corresponding procedure. The procedural behavior is then activated. It leads the interaction with the user, monitors the *SensorProcess* and executes primitive actions thanks

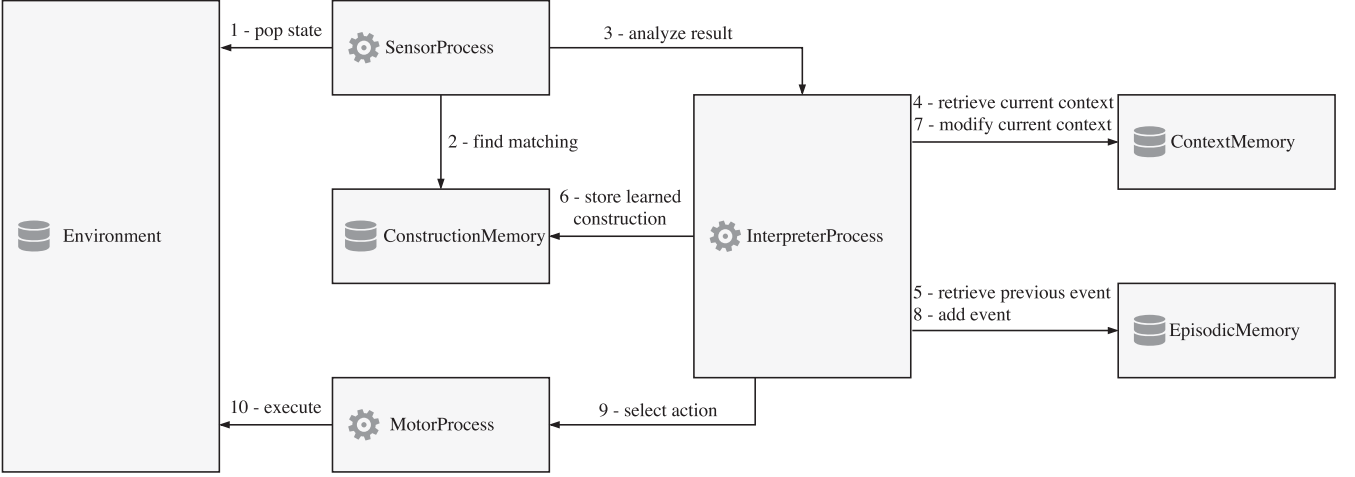


Fig. 4. Virtual Assistant Architecture Overview: During an interaction, the *SensorProcess* pops the state (e.g. user request “send an email”), and then searches for a match of this pattern in the *ConstructionMemory*. If a match is found, the *InterpreterProcess* begins to select actions from the retrieved sequence of actions, to execute via the *MotorProcess*. If no match is found for the user request, the system asks for a demonstration. Demonstrated events are stored in the *EpisodicMemory*, and then when the demonstration is finished, the sequences of events is consolidated via one-shot learning, and stored as construction that pairs the request (form) with the demonstration (meaning).

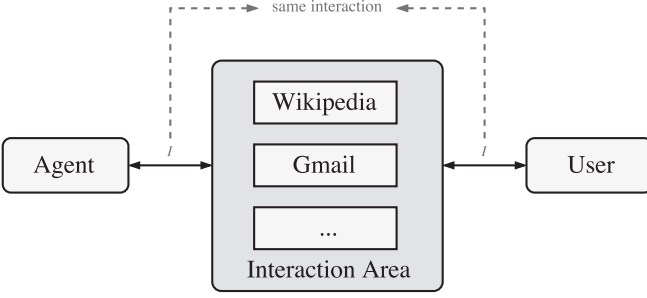


Fig. 5. Shared Environment: The interaction area allows the agent to perform actions that can be observed by the agent during learning by demonstration, and that can be performed by the agent and observed by the user.

to the *MotorProcess* in order to complete the procedure. We describe in more details the *SensorProcess* and the *MotorProcess* models, the meaning extraction algorithm and the one-shot procedural learning algorithm in the next sections.

## B. Shared Environment and Primitives

1) *Principle*: In robotics, the human and the robot share a commons space where the grounding of meaning in the perceptual and motor modalities relies on the physics of objects in space. The perceptual model represents objects in the world with attributes such as a position and an orientation while the motor model defines the most primitive commands that the effectors of a robot can implement such as to point, grasp, move, or release [19]. These primitive commands have regular consequences in the perceptual model in term of position and orientation, helping for plan recognition, procedural learning and finally, meaning extraction.

For the virtual assistant, the perceptual model should share the same characteristics as in robotics, that is, it must be domain independent, there must be a common space for the

user and the agent where this latter should be able to learn by demonstration by observing the user actions, e.g. filling the “subject” field in an email, and easily transfer its observations to its own motor commands. Figure 5 illustrates the shared environment between a virtual assistant and a user as a shared environment representation and a shared observation area.

The shared environment representation is an abstraction of a GUI based system interaction. We model the most primitive interactions that a user performs with a GUI, which are the ability to read and write textual information in a field, and to trigger a system transition (i.e. to enter text into a field, and click in a button). From the shared space, the virtual assistant is able to monitor and act on this representation independently of any natural language skills. We define the perceptual primitive structures as being frames, data, fields and actions.

**Frame** A frame is used to structure the information provided by the system and gather related information. It is composed of a label and a set of the other elements: frame, field, data and action. It could be viewed as a tree structure of information and thus, each component has a parent reference. E.g., all the paragraphs in a *Wikipedia* article.

**Data** A data gives the current state of a resource. It is composed of a label, a list of values which stands for the current state and a parent frame. E.g., the content of a paragraph in a *Wikipedia* article.

**Field** A field describes and holds textual information provided by the user. It is composed of a label, a list of values which hold the current information and a parent frame. E.g., the search input box on the *Wikipedia* website.

**Action** An action defines a possible transition from the current state. It is composed of a label and a parent frame. E.g., clicking on the search button on the *Wikipedia* website.

In complement to these perceptual structures, we define a set of primitive motor commands:

**Fill** Used to fill in a field with a natural language text in the environment.

**Execute** Used to trigger an action in the environment (clicking on a button).

**Speak** Used to send utterance via the chat.

The shared observation area allows the virtual assistant to observe the user actions. We developed a synchronization tool which maintains the consistence between the *Environment* module of the virtual assistant and a local or remote system interface. In this manner, the virtual assistant can read and act similarly on its own digital services and those that a user connects to it. Our synchronization tool has been adapted on top of a rich client application (RCA), web based applications, and a Rest<sup>1</sup> API.

Thus, we have a shared space where observations come from and the perceptual and motor primitive structures that correspond to a kind of Mandler’s primitive schemas on which language is grounded. We will now define how the effective observations are converted to these structures.

2) *Interactions between the Environment and the Sensor-Process modules*: Concrete observations diverge depending if the virtual assistant is interacting with a RCA, a web application or a Rest API. In our current research, we employ a direct mapping between these concrete types of observation and our primitive structures model. We respectively relied on a specific graphical library, HTML parsing methods and Json resources description for these conversions. The *Environment* module acts as an abstraction of these different types of conversion and is represented as a queue of events which will be popped by the *SensorProcess*. The events are the following:

**NewUtterance** This event is created each time a new natural language message is transmitted to one of the actors. It is composed of the message content and its source, either the agent or the user. For example “user: search for Boston”.

**ExecuteEvent** This event is generated each time an *Action* element is triggered in the environment model. It is composed of the element and its source, either the agent or the user. For example, when the user triggers the “search” button in *Wikipedia*.

**FillEvent** This event is generated each time a *Field* element is updated in the environment model. It is composed of the element, its updated value and its source, either the agent or the user. For example, when the user fills the query field in *Wikipedia*.

**FocusEvent** This event is generated each time an element is focused in the environment model. It is composed of the focused element and its source, either the agent or the user. We developed a special focus function that allows the user to indicate a particular item of interest in a GUI, e.g., a paragraph on a *Wikipedia* page. This will be of use when the user needs to demonstrate to the assistant to do something with the result of an action (e.g., to paste the result of a *Wikipedia* search into an email).

**AddEvent and DeleteEvent** These events are generated each time the structure of the environment model changes.

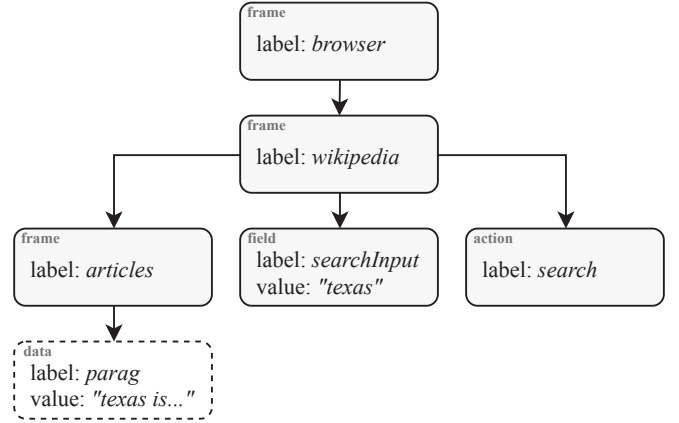


Fig. 6. An example of the environment model that illustrates a *Wikipedia* page loaded in a browser. The hierarchical structure forms label paths from the root node to the leaf nodes. In the current case, when an article will be fully loaded, an *AddEvent* will be dynamically generated and will contain a new *DataNode* with the label path set to *browser/wikipedia/articles/parag*.

As previously described, the environment model can be represented as a tree structure, those events are composed respectively of the added leaf nodes or deleted leaf nodes (*Action*, *Data*, *Field*) and a reference which is a label path from the root node. Figure 6 shows an example of the environment model structure when a *Wikipedia* page is loaded in a browser.

The *Environment* module is made up of the digital services and utterances produced by the user and the virtual assistant. When the *SensorProcess* pops events, those coming from the digital services will remain unchanged as they are already represented in term of our primitive structures however utterances will have to be mapped on those structures as explained in the next section.

### C. Language grounding

1) *Principle*: As stated in Section II-B, language understanding is defined as a process of mapping elements from a form space (e.g., natural language expressions) into a meaning space (e.g., a sequence of actions). Three kinds of representation must be taken into account, one to model elements in the form space (user utterances), one to model elements in the meaning space (primitives functions) and one to model data used by the process itself linking the form and meaning, called constructions. This is based on the construction grammar framework as characterized by Goldberg [4].

Meaning is extracted from goal directed utterances that command the agent to perform an action, such as “send an email to Peter”. The main learning capability of the virtual assistant is to learn how to map such utterances to a sequence of primitive functions that execute the intended goal. The structured sequence of primitives that results from this interaction with the human in a goal directed activity is thus associated with the surface form of a grammatical construction. This is similar to the pragmatic frame characterization of language

<sup>1</sup>Representational State Transfer

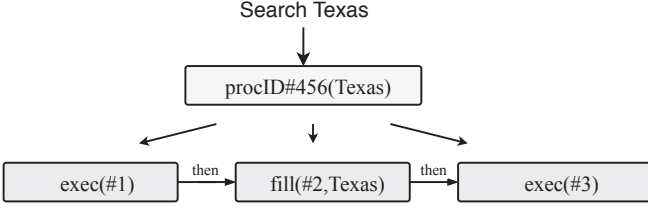


Fig. 7. Example of a learned grammatical construction grounded in the primitive functions.

learning developed by Rohlfing et al [35]. This sequence corresponds to a procedure in our meaning representation.

Mandler sought to define a level of representation for the PMA that corresponds to the sensory capabilities of the developing child. In the current research, we make an analogy between the developing child in the physical world, and an adaptive agent in a digital world of GUIs. In that world, we identified a set of sensory-motor actions that we refer to as primitives because they are the lowest level actions that a user can perform on a GUI and they are independent of the particular application. Thus, in a similar way, we define six primitive functions that compose a procedure. Three of these primitive functions are the primitive motor commands defined in Section III-B1 (fill, execute, speak) and we define three additional functions of a different nature:

**Focus** Used to highlight an element in the environment.

**Retain** Used to map an highlighted element with a natural language expression.

**End** Used to trigger the one-shot learning algorithm.

These three later functions have been introduced for the virtual assistant functioning only and further described in the next sections. The *Focus* is used to direct attention, the *Retain* is used to tell the assistant to memorize something, together they can be used as a form of copy-past for passing information between operations. The *End* is used to indicate the end of a learning stage.

The first step is to ground language directly on these 6 primitives. As illustrated by Figure 7 where “Search Texas” is mapped to a learned predicate-argument structure (a procedure in our meaning representation) which in turn is grounded to the primitives.

Then, Figure 8 illustrates how a complex procedures can be composed from existing procedures in a hierarchical structure that is finally grounded in the 6 primitive functions. Procedures are always grounded in these six functions, but when a new procedure is being learned, it can be defined in terms of previously learned procedures and the primitives.

A procedure is a decontextualized (i.e., its variables are not instantiated) ordered sequence of the primitive functions that set their parameters either with variables or constants. The decontextualized procedure is created by the one-shot-learning mechanism, where “decontextualization” is a process in which the instantiated arguments of fills are identified as variables. For example, “send an email to RECIPIENT about SUBJECT and say that CONTENTS”, is decomposed as a sequence of “fill” and “execute” primitive functions, and three

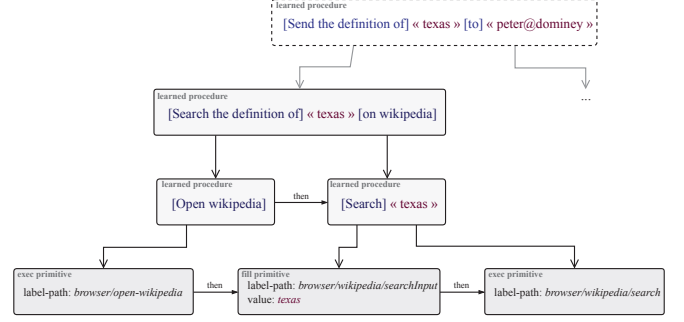


Fig. 8. Learned Procedure illustrating compositional meaning representation. At the lowest level, the procedure is grounded in the six primitive functions – fill, execute, speak, focus, retain and end. Here we see two “exec” and one “fill” actions. At the highest level, a procedure is defined purely in terms of previously learned procedures, which finally ground out in the six primitives. This illustrates the possibility for compositional structure in defining new procedures.

variables that will be bound to the “value” property of the “fill” functions (RECIPIENT, SUBJECT and CONTENTS). A procedure can be contextualized by instantiating its variables in a command input to the agent, for example, “send an email to pfdominey@gmail.com about the meeting and say that I will be late.”

2) *Interactions between the SensorProcess and the ConstructionMemory modules:* In a construction such as “Send an email to peter@gmail.com about the meeting and say that I will be ready as planned”, the system learns a form of complex item based construction “Send an email to VARX about VARY and say that VAR Z.” In this case VARX, VARY and VARZ correspond to the recipient, the subject and the message content of an email.

The *ConstructionMemory* stores the constructions as form-to-meaning mappings. It is a map of user utterance event patterns to procedures. As just illustrated, the patterns contain place holders for variables which are bound to the procedure variables when the procedure is contextualized or invoked. This notion of construction is borrowed from the domain of human linguistics, where grammatical constructions are mappings from sentence forms to meanings [4], [28]. In this context, we have exploited the distinction between closed class words (grammatical function words) used in the form characterization of constructions, and open class words (nouns, verbs, adjectives, etc.) that instantiate variables in the constructions in learning grammatical constructions [39], [40].

The *SensorProcess* uses this form to meaning mapping to find a match with the current user utterance event and the set of mappings in the *ConstructionMemory*. For example, when the user utterance is “send an email to clement@gmail.com about the paper and say finish it”, the *SensorProcess* compares this with the set of patterns in the *ConstructionMemory*. When a match is found, the pattern variables are instantiated and propagated to the procedure variables. The current utterance, the instantiated pattern variables and the decontextualized procedure constitutes the meaning which is forwarded to the *InterpreterProcess*.

As in the action learning system of Lallee et al [27], there



is no predefined predicate representation, they are learned in interaction with the user. This process is explained in the next section.

#### D. Using and Learning Language in Cooperative Procedure

The *InterpreterProcess* is responsible for maintaining the state of the virtual assistant coherent with the scenario it is engaged in with the user. This includes scenarios in which the user can cooperatively execute a plan of actions with the virtual assistant, and scenarios in which the virtual assistant learns a new plan of actions. For this, the *InterpreterProcess* has access to all the modules on which it can execute primitive actions and retrieve the current state. This includes the *ConstructionMemory*, the *ContextMemory*, the *EpisodicMemory* and the *MotorProcess*.

##### 1) Interaction with other modules:

a) *ContextMemory*: The *InterpreterProcess*, see Figure 4, works as a state transition system. States are characterized by two variables in the *ContextMemory*. One holds the current procedure instance being executed, and the other holds the current phase of the virtual assistant which can be:

**Available** Indicates that the virtual assistant is not engaged in a learning or executing scenario.

**Learning** Indicates that the virtual assistant is learning a new plan of actions.

**Executing** Indicates that the virtual assistant is executing a plan of actions.

The content of the *ContextMemory* and the meaning result in inputs will determine the next transition to perform.

b) *EpisodicMemory*: As a new procedure is being learned, the *InterpreterProcess* stores each executed primitive action in the *EpisodicMemory*, as well as each transformation of the environment model and the result of the *focus* and the *retain* primitive actions. This information is used by the learning algorithm to replay all the executed actions during the learning phase, so that these actions can then constitute the meaning component of the new learned construction. Note that during learning, the user can invoke primitive functions, and can also invoke learned procedures that are grounded in primitive functions, as illustrated in Figure 8.

c) *ConstructionMemory*: The *ConstructionMemory* is updated when the learning algorithm produces a new construction. This employs a “one-shot” learning method, where the demonstration that is encoded in the *EpisodicMemory* is transformed into a procedure with variables that corresponds to the meaning component of the form-to-meaning mapping. Then the *ConstructionMemory* is also used by the *SensorProcess* to determine if a current utterance produced by the user matches with a learned construction. In this case, the learned construction is instantiated and executed.

d) *MotorProcess*: The *MotorProcess* is used to execute the primitive motor actions, during learning and execution of a learned construction.

2) *InterpreterProcess Operation Modes*: The virtual assistant can be engaged in two main modalities: one when it executes a procedure and one when it learns a new one.

##### a) Execution Mode:

**Executing a learned procedure** The simplest situation is when the virtual assistant executes a procedure and the user provides all the context in one utterance. For Example: “search texas on Wikipedia.” In this situation, the *InterpreterProcess* will execute the learned procedure that was retrieved when the utterance was matched in the *ConstructionMemory*. At this stage, the virtual assistant will be in an “available” state. The system will instantiate the procedure in the retrieved meaning, and propagate the variables. That is, all the primitive actions will be grounded. The virtual assistant will execute the procedure and report when it is finished. Execution of the procedure will correspond to the following events:

- Fill(“browser/wikipedia/searchInput”, “texas”)
- Execute(“browser/wikipedia/search”)
- Speak(“Ok, It’s done”) [At this stage, the GUI is showing the article].

##### Executing a learned procedure with missing information

A second situation is when the user does not provide all the context in the utterance, such as “Search the definition of a term.” The result meaning will be retrieved as a procedure for searching on *Wikipedia*, similar to the previous example, except that the term of the search is missing. This will leave the variable of the *Fill* primitive unbound in the retrieved procedure. In such a case, where an unbound variable is encountered, the *InterpreterProcess* will generate a *Speak* primitive action in order to request the missing variable such as: “What is the query?” The next user utterance will be interpreted as the value of the missing variable. This leads to the third situation, when the virtual assistant does know yet how to request the missing variable.

##### b) Learning Mode:

**Learning how to request information** In order to generate the utterance, the virtual assistant must learn the mapping between the environment model element and the natural language pattern. For this, the *InterpreterProcess* will first ask the user to explain an element of its interface either by natural language such as “What is that: /browser/wikipedia/searchbox/searchInput?” or by GUI pointing. The response from the user will be bound to the element reference in the *ConstructionMemory* and will be reused for a next iteration as in the second scenario just illustrated in Section III-D2a.

##### Learning a new procedure with a one-shot algorithm

Learning scenarios are triggered when the meaning result does not contain a procedural meaning representation and the current state is available. In such case, if the user produces an utterance like “Search Texas on Wikipedia”, the system will invite the user to either explain what the system has to do or to provide a GUI demonstration and mark in the *EpisodicMemory* the beginning of the learning phase.

During learning, the meaning representations are of the same format for procedures evoked by natural language utterances and for environment model events demon-

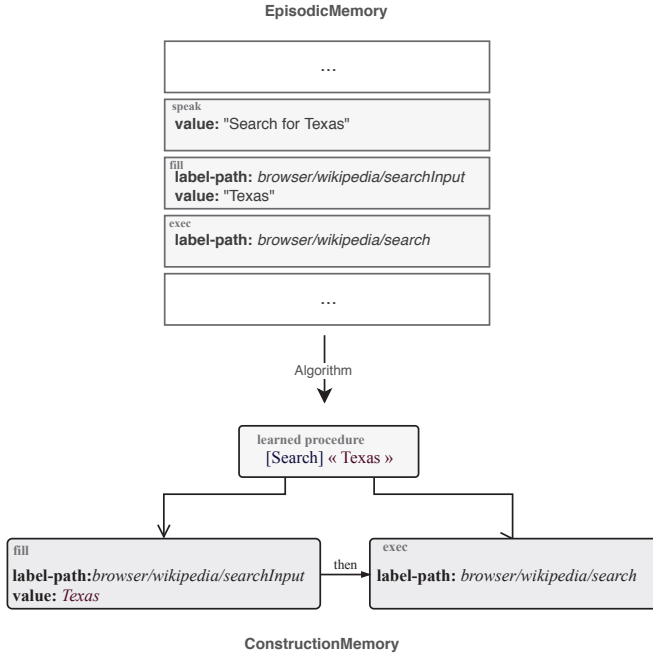


Fig. 9. One-shot Learning Algorithm applies on the example “Search Texas”.

strated on the GUI. Thus, the *InterpreterProcess* will handle these representation in the same way, except that for procedures evoked by language, the virtual assistant will execute the primitive action, whereas for GUI user demonstrations it is the user. Note that the use of existing procedures to define new procedures is a powerful mechanism for compositionality [46]. The virtual assistant logs all events occurring in the environment in the *EpisodicMemory*, including those from executing learned procedures during the teaching.

When the procedure is completed, the user must trigger the one-shot learning algorithm with a sentence indicating the end of the learning step. This sentence will be bound to the primitive function *End*. As just described, during learning, the virtual agent stores all events corresponding to the execution of primitive motor functions and perceptual updates. At the end of the learning, the algorithm creates the new construction as illustrated in Figure 9. The algorithm will generalize by creating variables in the construction. During the learning of a new procedure, when the user is explaining, if he/she repeats a string that is present in the original request, then that string is interpreted as a variable. For example, if the user says “Search Texas”, the system asks what do you mean, and the user shows that you open Wikipedia, and then type “Texas” in to the search window, “Texas” is identified as a string that is actually a variable in the learned construction “Search VARX”.

3) *Motor process*: The role of the *MotorProcess* is to transform primitive motor function representations into execution units in the environment. As for the sensory event representation, these representations are considered to be innate

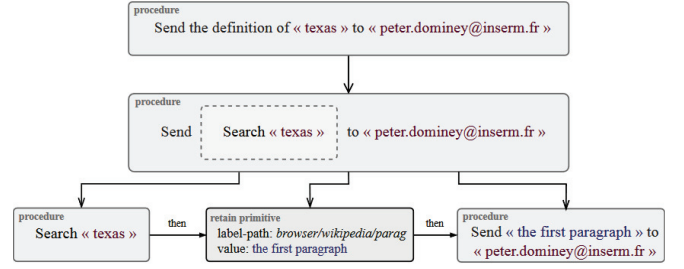


Fig. 10. A procedure illustrating the compositional meaning of two distinct sub-procedures.

primitives, and are specified at the conception phase and must be kept stable along the life-cycle of the virtual agent, as all the learned procedures rely on them. The transformations are simple algorithms that apply the function representation into the environment. This will in turn generate observable events that will be processed by the *SensorProcess*.

For example the *Execute* primitive function will be apply to an *Action* element in the environment model, this will generate an *ExecuteEvent* that will be processed by the virtual assistant as well as other event generated by the environment itself. Concretely, this corresponds, for example, to clicking on the search button after filling in the search target in *Wikipedia*.

### E. Compositionality

The previous sections explained how each module of the architecture interact to learn new procedures. In this section we show in more detail how the system can learn to interpret a compositional sentence.

As explained, the *InterpreterProcess* allows the combination of multiple learned procedure into a new one. In order to compose these sub-procedures, the *Focus* and *Retain* primitive functions can be used to temporarily save the evaluation of one procedure into a variable and to reuse it in another sub-procedure. This allows to learn how to interpret such utterances as: “Send the definition of Texas to peter.dominey@inserm.fr” where “the definition of Texas” must be evaluated and applied to “send  $x$  to peter.dominey@inserm.fr” as shown in Figure 10. In this example the user can combine the previously learned procedures “Search Texas” and “Send an email to peter.dominey@inserm.fr”.

In this case variables are created internally during learning of a new procedure that involves the *Retain* function. For example, after the user told the assistant to search the definition of Texas, the user will ask “retain the first paragraph”. What this means to the system is that a variable called “the first paragraph” is created, and it is associated with the function of the *Retain*. The *Retain* is a function that is associated with the *Focus*, where the user focuses attention on some object in the page with a green icon that can be moved on the page. In this case, the user focuses attention on the first paragraph of the resulting Wikipedia definition. The “retain the first paragraph” command thus associates the contents of the focus (the first paragraph of the Wikipedia page) with the variable called “the first paragraph”. Later in the scenario, when the

assistant asks “What is the message?” and the user replies “the first paragraph”, the utterance will be evaluated as pointing to a variable and the contents of the variable named “the first paragraph” is pasted into the message frame of the mail. In this manner, the variable called “the first paragraph” will always be associated with the contents of the first paragraph of the Wikipedia page that is opened during the execution of the “Send the definition of X to Y” construction.

#### IV. CURRENT PERFORMANCE

The main objective of this paper is the design of the representations, the algorithms and the architecture for an adaptive assistant that exploits certain functionality of usage-based learning of constructions. To evaluate this objective, we concentrate on a functional/behavioral evaluation. We thus explain in detail several use cases that illustrate the usage-based functionality of the system. We also provide a quantitative analysis of the potential savings in user intervention that idealized performance of such a system could provide.

##### A. Use Cases

In this section we provide examples of typical relevant use case scenarios that a user can achieve with our implementation based on the *Wikipedia* and the *Gmail* website. We present 4 scenarios in chronological order. The scenarios begin when the user connects for the first time with its virtual assistant, and in the final scenario the system can interpret the utterance “Could you send an email to user@domain.com and write him something about Texas”.

a) *Scenario 1*: The user will teach two basic *Gmail* actions to its virtual assistant: how to create a new draft and how to set the subject. In this situation, the user is connected with its virtual assistant with an opened chat box and is already connected to its *Gmail* inbox.

**User** Create a draft  
**Assistant** What do you mean?  
**User** *The user creates a draft on the Gmail GUI. (The system perceives the event primitives, generated by the user’s actions in the GUI, and stores them in the EpisodicMemory).*  
**Assistant** Ok *(the virtual assistant has received events from the GUI)*  
**User** End *(the user indicates the end of the learning procedure)*

At this point, when the user says “End”, a new construction is created in the *ConstructionMemory* with “Create a draft” as the form component, and the sequences of events stored in the *EpisodicMemory* as the meaning component.

**User** Set the subject  
**Assistant** What do you mean?  
**User** *The user fills the subject field on the Gmail GUI*  
**Assistant** Ok  
**User** End

Again, when the user says “End”, a new construction is created in the *ConstructionMemory* with “Set the subject” as the form component, and the sequences of events stored in

the *EpisodicMemory* as the meaning component. Similarly, the user can teach other basic actions on *Gmail* such as opening its inbox, sending a draft, setting the recipients as well as for other services such as searching a term on *Wikipedia*. A video illustrating these examples is available online <sup>2</sup>.

b) *Scenario 2*: When the virtual assistant needs to know how to speak about an element in the environment it can ask for this element. In this scenario, the user teaches how to request the subject of a message so that the next time the user will not have to interact with its GUI to set the subject of a message. A full video of this example is available online <sup>3</sup>.

**User** Set the subject *(this matches with a previously learned procedure, from scenario 1, where the user filled the subject field.)*  
**Assistant** What does “/browser/Inbox/Subject” mean? *(In the current execution, the “subject” field in the GUI is not yet set, and this leads to an unbound variable corresponding to the subject input box in the Gmail GUI that can also be highlighted)*  
**User** The subject of the message *(the user specifies that this field is the subject of the message)*  
**Assistant** What is the subject of the message? *(this is still an unbound variable, so the assistant asks for the content)*  
**User** Meeting schedule  
**Assistant** Ok, It’s done.

c) *Scenario 3*: After some learning, the virtual assistant is able to perform basic actions on both the *Wikipedia* and the *Gmail* web sites. In this scenario we show how a user can teach a new procedure that combines previously learned procedures on different task domain as illustrated in an online video <sup>4</sup>.

**User** Send the definition of Boston to cl\*\*\*ge@gmail.com  
**Assistant** What do you mean?  
**User** Go to Wikipedia  
**Assistant** Ok, It’s done  
**User** Search for Boston  
**Assistant** Ok, It’s done  
**User** retain the first paragraph  
**User** *The user uses the focus to point the element to retain in its GUI interface.*  
**Assistant** Ok, It’s done *(The system binds the indicated paragraph to the variable “the first paragraph”)*  
**User** Close Wikipedia  
**Assistant** Ok, It’s done  
**User** Send an email to ce\*\*\*ge@gmail.com  
**Assistant** What is the subject of the message? *(Note that here the assistant asks a question in order to establish the binding for this variable)*  
**User** Boston’s definition  
**Assistant** What is the message?

<sup>2</sup>Usage-based Learning Virtual Assistant [Part 1: Learning basic actions]  
[https://www.youtube.com/watch?v=Lj\\_CW00IU88&list=PLKnrLyZkzbaNzpcz\\_gI3HS-mLvyOu1mB&index=2](https://www.youtube.com/watch?v=Lj_CW00IU88&list=PLKnrLyZkzbaNzpcz_gI3HS-mLvyOu1mB&index=2)

<sup>3</sup>Usage-based Learning Virtual Assistant [Part 2: Learning GUI elements]  
[https://www.youtube.com/watch?v=TthKpUKlrck&list=PLKnrLyZkzbaNzpcz\\_gI3HS-mLvyOu1mB&index=3](https://www.youtube.com/watch?v=TthKpUKlrck&list=PLKnrLyZkzbaNzpcz_gI3HS-mLvyOu1mB&index=3)

<sup>4</sup>Usage-based Learning Virtual Assistant [Part 4: Learning complex tasks]  
[https://www.youtube.com/watch?v=Kseke7tQ7ik&index=5&list=PLKnrLyZkzbaNzpcz\\_gI3HS-mLvyOu1mB](https://www.youtube.com/watch?v=Kseke7tQ7ik&index=5&list=PLKnrLyZkzbaNzpcz_gI3HS-mLvyOu1mB)

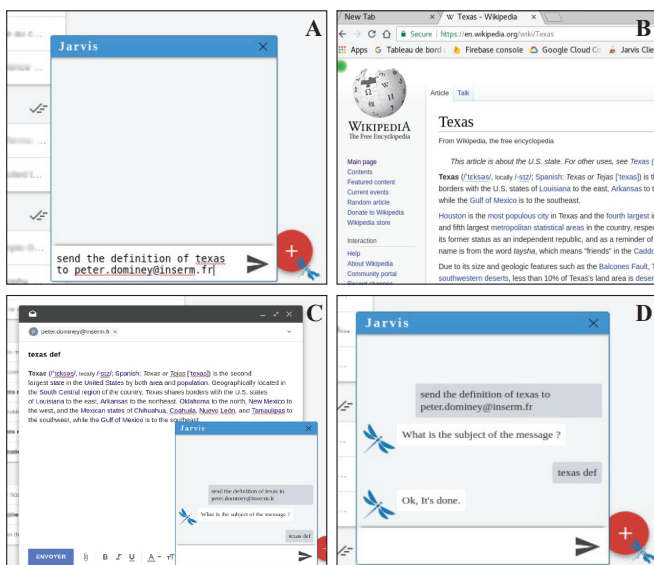


Fig. 11. (A) User makes a request in the chat window “send the definition of texas to peter.dominey@inserm.fr”. (B) The system recognizes this construction, and executes the learned procedure: opens *Wikipedia*, writes “texas” into the search input. (C) Once the definition is loaded, copy-pastes the first paragraph of the definition into the body of the email. (D) Asks the user for the subject and then sends the message.

**User** The first paragraph

**Assistant** Ok, It’s done. (*The contents of the variable “the first paragraph” is set as the contents of the message as described in III-E*)

**User** End

Figure 11 provides a view of the GUI interface during the unfolding of this execution of the virtual assistant.

The result of this command generates a mail which effectively contains the Wikipedia definition of Boston. This learning scenario demonstrates how a new construction can be learned, with a rather complex sentence as the form component, and a structured set of primitive commands as the meaning component. Importantly, when the user explains what this sentence means, she can re-use existing learned procedures by evoking them with natural language, as well as using more primitive actions on the GUI. This illustrates the ability of the system to learn compositional structures. It is worth noting that the system immediately generalizes to use of the same construction with different arguments. Thus, the user can now say “Send the definition of Texas to john@gmail.com” and the system will execute the learned procedure appropriately.

d) *Scenario 4:* The virtual assistant can learn how to interpret more elaborated sentence in the same manner it learns procedures. That is, we can say a different sentence that has the same meaning as a previously learned procedure, and then when the system asks “What do you mean” we state the meaning as in the learned formulation. This way a previously learned procedure can be reformulated. Here we show the final result and an online video highlights this feature in a context

of text only interactions on a smartphone <sup>5</sup>:

**User** Could you send an email to cl\*\*\*ge@protonmail.com and write him something about Texas

**Assistant** What is the subject of the message?

**User** Texas definition

**Assistant** Ok, It’s done

These scenarios illustrate how a complex set of primitive commands can become associated with a high level invoking command, in the form of a grammatical construction encoded in the *ConstructionMemory*.

## B. Idealized Performance

We show in this Section how the system can be effective by reducing the number of interventions of the user. Figure 12 illustrates how learning, and reuse of learned procedures in new procedures, reduces the number of interventions by the user. We compare 3 strategies the user can take:

**No Agent** The *No Agent* strategy is when the user does not use the agent and has to perform all the actions on its GUI.

**All-in-one** The *All-in-one* strategy is when the user minimizes natural language interactions during the learning, that is, learning by demonstration is always preferred and there is no reuse of previously learned procedures when learning a new compound task.

**Reuse** The *Reuse* strategy is when the user maximizes natural language interactions during the learning, that is, learning by explanation is always preferred and the user tries to maximize the reuse of the previously learned procedures by structuring the learning from the most fine grain tasks to the most compound tasks.

To evaluate these 3 strategies, we first count the number of interactions (acting on a GUI or speaking to the agent count both for one interaction) to complete a set of 3 tasks (search on wikipedia, send an email and send the definition of something to someone). Panel A represents the number of interactions needed to complete the 3 tasks 3 times which is the maximum number of times after which the agent has totally learned a procedure and where then no interaction linked to the learning step is needed. Panel B represents a projection of the number of interactions needed to complete the two first tasks 100 times and then 100 times 100 different versions of the third task. So the panel B simulates a user who extensively uses what he teaches to his agent and who extensively teaches new compound tasks.

When first teaching the agent how to do things, there is no advantage to the user (panel A) compared to doing the same actions without the agent. When learned actions are repeatedly used and when we simulate the learning of 100 additional tasks (panel B), then, we respectively see a reduction in the user actions with the agent and the benefit from reuse of previously learned procedures. This does not constitute

<sup>5</sup>Usage-based Learning Virtual Assistant [Part 6: Use learned skills on a smartphone]

[https://www.youtube.com/watch?v=2m0YUB1etKc&index=7&list=PLKnrLyZkzbaNqpc\\_g3HS-mLvyOu1mB](https://www.youtube.com/watch?v=2m0YUB1etKc&index=7&list=PLKnrLyZkzbaNqpc_g3HS-mLvyOu1mB)



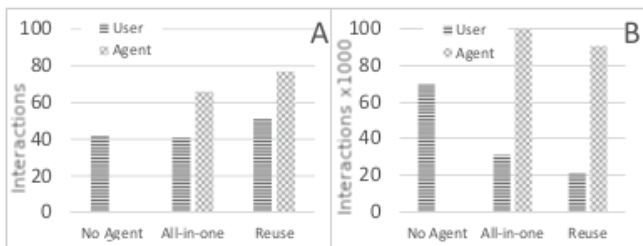


Fig. 12. System evaluation: Number of actions taken by the user and the agent to execute learned actions. We compare 3 strategies, when the user do not use the agent, when he does not reuse learned procedures to learn new procedures and when he does reuse previously learned procedures. A. Represents the number of user and agent interactions during the process where the user teaches the agent three tasks. B. Represents the number of user and agent interactions after the activity in A, when the user teaches the agent 100 new tasks based on reuse of previously learned tasks.

a proper user study, but allows a quantitative view of the potential effectiveness of the system. A noteworthy point is that this paradigm of reuse of learned material in learning new material generates progression in efficiency that increases as learning more potentially reusable material proceeds.

## V. DISCUSSION

In this paper, we present a usage-based system where an end-user is able to teach to a virtual assistant how to operate a set of arbitrary digital services using natural language. The virtual assistant thus uses the inherent structure in the user’s natural language commands to organize primitive perceptual elements into procedures. This allows the assistant to adapt to different services and to variability in users’ choices on their use of language. The learning was initially employed to learn how to ground natural language utterances into domain independent perceptual and motor commands in order to execute basic service actions (e.g., to open *Gmail*, create a draft mail, set the subject, etc.). Based on such learned utterance-action constructions, the virtual assistant then learned how to perform more complex tasks, characterized by the composition of these previously learned constructions and the corresponding procedures, with argument passing between them (e.g., the search term from *Gmail* being passed to *Wikipedia*, and the resulting definition being passed back to *Gmail*). Thus, learned procedures are demonstrated to be used in a recursive or compositional definition of a new procedure like “send an email to pfdominey@gmail.com about programming with the definition of python”, that the user can explain in natural language by decomposing into the “send an email” procedure, and another learned procedure that gets definitions from *Wikipedia*. The ability to ground commands in user demonstrated actions on the GUI, and then the ability to compose these learned commands purely by language illustrates how simple recursive mechanisms for creating labeled structures provides a powerful compositional mechanism for specifying complex interactions with the world [46].

We show that by taking a developmental perspective based on grammatical constructions as form-to-meaning mappings [10], [11], [28], while the developed infrastructure is lite, the virtual assistant is able to jointly learn natural language

structures together with procedural semantics in realistic usage scenarios. The ability to learn these procedures and to label them is minimalist in terms of its simplicity but it is quite powerful. Indeed, this conception avoids the need for inherent language-specific and domain-specific knowledge engineering, while still providing a substantial learning capability. As we illustrated, in our system, this capability allows the user to progressively create successive levels of hierarchical structure, resulting in powerful and compact linear strings that represent and re-enact a complex hierarchical structure. Additionally, we introduce the capability for variables and the passage of variable bindings across these different levels of the hierarchical structure. This provides a concrete implementation which illustrates the power of the merge and label capability in human cognition [46].

The ability for a user to teach an assistant with natural language brings several advantages: the user can adapt the assistant to its own needs by teaching new procedural knowledge, and the system is more resilient to faults such as unrecognized words in the lexicon. Perhaps most importantly, with this learning, extending the system in term of functionalities can be done at runtime by the end-user instead of preprogramming commands at the conception phase. At the same time, the end-user experience remains intuitive by keeping the teaching stage as transparent as possible regarding the user’s demonstrations. This transparency is enhanced by eliminating the need for the user to learn pre-specified language commands, and shielding users from the virtual assistant’s internal organization so as not to distract them from their tasks.

The major limitation of this system is its rigidity in matching the fixed component of input sentence with learned constructions. Thus if the system knows the meaning of “send an email to X about Y and say that Z”, and the user says “send a mail to X about Y and say that Z”, the system will not match the pattern, despite the similarity to the original sentence. We are currently developing a method to address this limitation, based on machine learning work in the area of semantic relatedness [47]. The system will compare the user’s input with existing sentences in the *ConstructionMemory* in order to determine if the request is a variant of a known construction, or if it is a new construction to be learned. This work will be a key point to make the virtual assistant more resilient to natural language variability, which includes the ability to generalize over synonymy, recurrent dialog parts of speech, different grammatical forms, and to allow user studies which are used to this level of language adaptability.

Further improvements will be made. First, the capability to generate narrative constructions that allow the virtual assistant to learn how to understand and narrate a sequence of events based on the causal and dependency relations between the constituent events will be adapted from our related work in robotics [48]. Indeed, the ability of narrative to impose additional structure on representations of human experience is crucial in the human ability to represent a reality shared with others [49], [50], and will play an important role in the future development of adaptive assistants.

Another possible extension of the system is to learn the meanings of words based on their situation within a construc-

tion. In the construction "Send an email to X about Y and say that Z", the system has access to the information that items which instantiate X are email addresses. While we currently do not exploit this, such syntactic bootstrapping (i.e. inferring a referent meaning based on its grammatical configuration) can be used to accelerate learning as we demonstrated in Dominey [42].

## REFERENCES

- [1] M. Korpusik, Z. Collins, and J. Glass, "Semantic mapping of natural language input to database entries via convolutional neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*. IEEE, 2017, pp. 5685–5689.
- [2] M. Petit, S. Lallée, J.-D. Boucher, G. Pointeau, P. Cheminade, D. Ognibene, E. Chinellato, U. Pattacini, I. Gori, U. Martinez-Hernandez, and others, "The coordinating role of language in real-time multimodal learning of cooperative tasks," *Autonomous Mental Development, IEEE Transactions on*, vol. 5, no. 1, pp. 3–17, 2013. [Online]. Available: [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=6249732](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6249732)
- [3] N. G. Canbek and M. E. Mutlu, "On the track of Artificial Intelligence: Learning with intelligent personal assistants," *Journal of Human Sciences*, vol. 13, no. 1, pp. 592–601, 2016.
- [4] A. E. Goldberg, *Constructions: A construction grammar approach to argument structure*. University of Chicago Press, 1995.
- [5] M. Sun, Y.-N. Chen, and A. I. Rudnicky, "An intelligent assistant for high-level task understanding," in *Proceedings of the 21st International Conference on Intelligent User Interfaces*. ACM, 2016, pp. 169–174.
- [6] J. Allen, N. Chambers, G. Ferguson, L. Galescu, H. Jung, M. Swift, and W. Taysom, "PLOW: A collaborative task learning agent," in *Proceedings of the National Conference on Artificial Intelligence*, vol. 22. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007, p. 1514. [Online]. Available: <http://www.aaai.org/Papers/AAAI/2007/AAAI07-240.pdf>
- [7] C. H. Hwang and L. K. Schubert, "Episodic logic: A situational logic for natural language processing," *Situation Theory and its Applications*, vol. 3, pp. 303–338, 1993. [Online]. Available: [http://books.google.fr/books?hl=fr&lr=&id=wlfBUzFzJ8gC&oi=fnd&pg=PA303&dq=Episodic+Logic+expressiveness&ots=vKc3iXc8EI&sig=Hk3XN87GAJwFxeSNs6l\\_tYu8W8A](http://books.google.fr/books?hl=fr&lr=&id=wlfBUzFzJ8gC&oi=fnd&pg=PA303&dq=Episodic+Logic+expressiveness&ots=vKc3iXc8EI&sig=Hk3XN87GAJwFxeSNs6l_tYu8W8A)
- [8] G. Ferguson, J. F. Allen, and others, "TRIPS: An integrated intelligent problem-solving assistant," in *AAAI/IAAI*, 1998, pp. 567–572. [Online]. Available: <http://www.aaai.org/Papers/AAAI/1998/AAAI98-080.pdf>
- [9] A. Azaria, J. Krishnamurthy, and T. Mitchell, "Instructable Intelligent Personal Agent," in *AAAI*, 2016, pp. 2681–2689.
- [10] M. Tomasello, "Constructing A Language: A Usage Based Theory of Language Acquisition," vol. 8, Jan. 2003.
- [11] E. V. Clark, *First language acquisition*. Cambridge University Press, 2003.
- [12] I. Nomikou, M. Schilling, V. Heller, and K. J. Rohlfing, "Language-at all times," *Interaction Studies*, vol. 17, no. 1, pp. 120–145, 2016.
- [13] M. Tomasello, "The item-based nature of children's early syntactic development," *Trends in cognitive sciences*, vol. 4, no. 4, pp. 156–163, 2000.
- [14] E. Lieven, H. Behrens, J. Speares, and M. Tomasello, "Early syntactic creativity: A usage-based approach," *Journal of child language*, vol. 30, no. 2, pp. 333–370, 2003.
- [15] M. Asada, K. Hosoda, Y. Kuniyoshi, H. Ishiguro, T. Inui, Y. Yoshikawa, M. Ogino, and C. Yoshida, "Cognitive developmental robotics: A survey," *IEEE transactions on autonomous mental development*, vol. 1, no. 1, pp. 12–34, 2009.
- [16] J. M. Mandler, "How to build a baby: III. Image schemas and the transition to verbal thought," *From perception to meaning: Image schemas in cognitive linguistics*, pp. 137–163, 2005.
- [17] S. Lallée, S. Lemaignan, A. Lenz, C. Melhuish, L. Natale, S. Skachek, T. van Der Zant, F. Warneken, and P. F. Dominey, "Towards a platform-independent cooperative human-robot interaction system: I. perception," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 4444–4451. [Online]. Available: [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=5652697](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5652697)
- [18] S. Lallée, U. Pattacini, J. D. Boucher, S. Lemaignan, A. Lenz, C. Melhuish, L. Natale, S. Skachek, K. Hamann, and J. Steinwender, "Towards a platform-independent cooperative human-robot interaction system: II. perception, execution and imitation of goal directed actions," in *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE, 2011, pp. 2895–2902.
- [19] S. Lallée, U. Pattacini, S. Lemaignan, A. Lenz, C. Melhuish, L. Natale, S. Skachek, K. Hamann, J. Steinwender, and E. A. Sisbot, "Towards a platform-independent cooperative human robot interaction system: III an architecture for learning and executing actions and shared plans," *IEEE Transactions on Autonomous Mental Development*, vol. 4, no. 3, pp. 239–253, 2012.
- [20] S. Lallée, K. Hamann, J. Steinwender, F. Warneken, U. Martienz, H. Barron-Gonzales, U. Pattacini, I. Gori, M. Petit, and G. Metta, "Cooperative human robot interaction systems: IV. Communication of shared plans with Naïve humans using gaze and speech," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 129–136.
- [21] S. Lallée, V. Vouloutsis, S. Wierenga, U. Pattacini, and P. Verschure, "EFAA: a companion emerges from integrating a layered cognitive architecture," in *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*. ACM, 2014, pp. 105–105.
- [22] J. M. Mandler, "How to build a baby: On the development of an accessible representational system," *Cognitive Development*, vol. 3, no. 2, pp. 113–136, 1988.
- [23] —, "How to build a baby: II. Conceptual primitives," *Psychological review*, vol. 99, no. 4, p. 587, 1992.
- [24] —, "On the spatial foundations of the conceptual system and its enrichment," *Cognitive science*, vol. 36, no. 3, pp. 421–451, 2012.
- [25] A.-L. Mealier, G. Pointeau, P. Gärdenfors, and P. F. Dominey, "Construals of meaning," *Interaction Studies*, vol. 17, no. 1, pp. 41–69, 2016.
- [26] P. F. Dominey and J.-D. Boucher, "Learning to talk about events from narrated video in a construction grammar framework," *Artificial Intelligence*, vol. 167, no. 1–2, pp. 31–61, 2005.
- [27] S. Lallée, C. Madden, M. Hoen, and P. F. Dominey, "Linking language with embodied and teleological representations of action for humanoid cognition," *Frontiers in neurorobotics*, vol. 4, 2010. [Online]. Available: <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2889716/>
- [28] A. E. Goldberg, "Constructions: a new theoretical approach to language," *Trends in cognitive sciences*, vol. 7, no. 5, pp. 219–224, 2003.
- [29] J. Gaspers, P. Cimiano, K. Rohlfing, and B. Wrede, "Constructing a Language From Scratch: Combining Bottom-Up and Top-Down Learning Processes in a Computational Model of Language Acquisition," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 9, no. 2, pp. 183–196, 2017.
- [30] P. Dominey, "From holophrases to abstract grammatical constructions: insights from simulation studies," *Constructions in Acquisition*, E. Clark and B. Kelly, Eds. Stanford: CSLI Publications, pp. 137–162, 2006.
- [31] P. F. Dominey and J.-D. Boucher, "Developmental stages of perception and language acquisition in a perceptually grounded robot," *Cognitive Systems Research*, vol. 6, no. 3, pp. 243–259, 2005. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1389041704000762>
- [32] A. Cangelosi and T. Riga, "An embodied model for sensorimotor grounding and grounding transfer: Experiments with epigenetic robots," *Cognitive science*, vol. 30, no. 4, pp. 673–689, 2006.
- [33] F. Stramandinoli, D. Marocco, and A. Cangelosi, "Making sense of words: a robotic model for language abstraction," *Autonomous Robots*, vol. 41, no. 2, pp. 367–383, 2017.
- [34] V. Tikhonoff, A. Cangelosi, and G. Metta, "Integration of speech and action in humanoid robots: iCub simulation experiments," *IEEE Transactions on Autonomous Mental Development*, vol. 3, no. 1, pp. 17–29, 2011.
- [35] K. J. Rohlfing, B. Wrede, A.-L. Vollmer, and P.-Y. Oudeyer, "An alternative to mapping a word onto a concept in language acquisition: pragmatic frames," *Frontiers in psychology*, vol. 7, p. 470, 2016.
- [36] L. Steels, "The emergence and evolution of linguistic structure: from lexical to grammatical communication systems," *Connection science*, vol. 17, no. 3–4, pp. 213–230, 2005.
- [37] B. Bergen, N. Chang, and S. Narayan, "Simulated action in an embodied construction grammar," in *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 26, 2004.
- [38] Z. Solan, D. Horn, E. Ruppin, and S. Edelman, "Unsupervised learning of natural languages," *Proceedings of the National Academy of Sciences*, vol. 102, no. 33, pp. 11 629–11 634, 2005.
- [39] P. F. Dominey, M. Hoen, J.-M. Blanc, and T. Lelekov-Boissard, "Neurological basis of language and sequential cognition: evidence from

- simulation, aphasia, and ERP studies,” *Brain and language*, vol. 86, no. 2, pp. 207–225, 2003.
- [40] X. Hinaut and P. F. Dominey, “Real-time parallel processing of grammatical structure in the fronto-striatal system: A recurrent network simulation study using reservoir computing,” *PloS one*, vol. 8, no. 2, p. e52946, 2013.
  - [41] J. Gaspers and P. Cimiano, “A Computational Model for the Item-Based Induction of Construction Networks,” *Cognitive science*, vol. 38, no. 3, pp. 439–488, 2014.
  - [42] P. F. Dominey, “Conceptual grounding in simulation studies of language acquisition,” *Evolution of Communication*, vol. 4, no. 1, pp. 57–85, 2002. [Online]. Available: <http://www.ingentaconnect.com/content/jbp/evco/2000/00000004/00000001/art00004>
  - [43] S. M. McCauley and M. H. Christiansen, “Prospects for usage-based computational models of grammatical development: argument structure and semantic roles,” *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 5, no. 4, pp. 489–499, 2014.
  - [44] M. Tomasello, M. Carpenter, J. Call, T. Behne, and H. Moll, “Understanding and sharing intentions: The origins of cultural cognition,” *Behavioral and Brain Sciences*, vol. 28, no. 5, pp. 675–691, Oct. 2005. [Online]. Available: <https://www.cambridge.org/core/journals/behavioral-and-brain-sciences/article/understanding-and-sharing-intentions-the-origins-of-cultural-cognition/F9C40BF73A68B30B8EB713F2F947F7E2>
  - [45] K. Tylén, E. Weed, M. Wallentin, A. Roepstorff, and C. D. Frith, “Language as a tool for interacting minds,” *Mind & Language*, vol. 25, no. 1, pp. 3–29, 2010.
  - [46] T. Goucha, E. Zaccarella, and A. D. Friederici, “A revival of the Homo loquens as a builder of labeled structures: neurocognitive considerations,” *Neuroscience & Biobehavioral Reviews*, 2017.
  - [47] E. Agirre, “Semantic Textual Similarity,” 2018. [Online]. Available: [http://ixa2.si.ehu.es/stswiki/index.php/Main\\_Page](http://ixa2.si.ehu.es/stswiki/index.php/Main_Page)
  - [48] A.-L. Mealier, G. Pointeau, S. Mirliaz, K. Ogawa, M. Finlayson, and P. F. Dominey, “Narrative Constructions for the Organization of Self Experience: Proof of Concept via Embodied Robotics,” *Frontiers in psychology*, vol. 8, p. 1331, 2017.
  - [49] J. S. Bruner, *Acts of meaning*. Harvard University Press, 1990, vol. 3.
  - [50] J. Bruner, “The narrative construction of reality,” *Critical inquiry*, vol. 18, no. 1, pp. 1–21, 1991.

# Bibliographie

- Agirre, Eneko. 2018. *Semantic Textual Similarity*. [http://ixa2.si.ehu.es/stswiki/index.php/Main\\_Page](http://ixa2.si.ehu.es/stswiki/index.php/Main_Page).
- Allen, James, Nathanael Chambers, George Ferguson, Lucian Galescu, Hyuckchul Jung, Mary Swift et William Taysom. 2007. « PLOW : A collaborative task learning agent ». In *Proceedings of the National Conference on Artificial Intelligence*, 22 : 1514. Menlo Park, CA ; Cambridge, MA ; London ; AAAI Press ; MIT Press ; 1999. Visité le 2 janvier 2015. <http://www.aaai.org/Papers/AAAI/2007/AAAI07-240.pdf>.
- Azaria, Amos, Jayant Krishnamurthy et Tom Mitchell. 2016. « Instructable Intelligent Personal Agent ». In *AAAI*, 2681-2689.
- Bates, Elizabeth, Sandra McNew, Brian MacWhinney, Antonella Devescovi et Stan Smith. 1982. « Functional constraints on sentence processing : A cross-linguistic study ». *Cognition* 11 (3) : 245-299.
- Bates, Elizabeth, Beverly Wulfeck et Brian MacWhinney. 1991. « Cross-linguistic research in aphasia : An overview ». *Brain and language* 41 (2) : 123-148.
- Baudis, Petr. 2018. *dataset-sts : Semantic Text Similarity Dataset Hub*. Original-date : 2016-01-21T03 :22 :10Z, mai. Visité le 29 mai 2018. <https://github.com/brmson/dataset-sts>.
- Bergen, Benjamin, Nancy Chang et Shweta Narayan. 2004. « Simulated action in an embodied construction grammar ». In *Proceedings of the Annual Meeting of the Cognitive Science Society*, t. 26.
- Bruner, Jerome. 1991. « The narrative construction of reality ». *Critical inquiry* 18 (1) : 1-21.
- Bruner, Jerome S. 1990. *Acts of meaning*. T. 3. Harvard University Press.
- Canbek, Nil Goksel, et Mehmet Emin Mutlu. 2016. « On the track of Artificial Intelligence : Learning with intelligent personal assistants ». *Journal of Human Sciences* 13 (1) : 592-601.
- Cangelosi, Angelo, et Thomas Riga. 2006. « An embodied model for sensorimotor grounding and grounding transfer : Experiments with epigenetic robots ». *Cognitive science* 30 (4) : 673-689.

- Carey, Susan. 2000. « The origin of concepts ». *Journal of Cognition and Development* 1 (1) : 37-41.
- Cer, Daniel, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio et Lucia Specia. 2017. « SemEval-2017 Task 1 : Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation » [en en], 1-14. Association for Computational Linguistics. Visité le 28 mai 2018. doi :10.18653/v1/S17-2001. <http://aclweb.org/anthology/S17-2001>.
- Chen, Tianqi, et Carlos Guestrin. 2016. « Xgboost : A scalable tree boosting system ». In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785-794. ACM.
- Chomsky, Noam. 1995. *The minimalist program*. T. 28. Cambridge University Press. Visité le 27 juin 2016.
- Clark, E., et B. Kelly. 2006. « From holophrases to abstract grammatical constructions ». In *Constructions in Acquisition*, 137-162. Stanford : CSLI Publications. Visité le 24 septembre 2018. [https://scholar.google.fr/scholar?hl=fr&as\\_sdt=0%2C5&q=From+holophrases+to+abstract+grammatical+constructions%3A+insights+from+simulation&btnG=](https://scholar.google.fr/scholar?hl=fr&as_sdt=0%2C5&q=From+holophrases+to+abstract+grammatical+constructions%3A+insights+from+simulation&btnG=).
- Clark, Eve V. 2003. *First language acquisition*. Cambridge University Press.
- Dominey, Peter Ford, et Jean-David Boucher. 2005a. « Developmental stages of perception and language acquisition in a perceptually grounded robot ». *Cognitive Systems Research* 6 (3) : 243-259. Visité le 2 janvier 2015. <http://www.sciencedirect.com/science/article/pii/S1389041704000762>.
- . 2005b. « Learning to talk about events from narrated video in a construction grammar framework ». *Artificial Intelligence* 167 (1-2) : 31-61.
- Dominey, Peter Ford, et Christelle Dodane. 2004. « Indeterminacy in language acquisition : the role of child directed speech and joint attention ». *Journal of Neurolinguistics* 17 (2) : 121-145. Visité le 27 juin 2016.
- Dominey, Peter Ford, Michel Hoen, Jean-Marc Blanc et Taissia Lelekov-Boissard. 2003. « Neurological basis of language and sequential cognition : evidence from simulation, aphasia, and ERP studies ». *Brain and language* 86 (2) : 207-225.
- Fatima, S. Shaheen, et Michael Wooldridge. 2001. « Adaptive task resources allocation in multi-agent systems ». In *Proceedings of the fifth international conference on Autonomous agents*, 537-544. ACM. Visité le 4 avril 2014. <http://dl.acm.org/citation.cfm?id=376439>.
- Ferguson, George, James F. Allen et al. 1998. « TRIPS : An integrated intelligent problem-solving assistant ». In *AAAI/IAAI*, 567-572. Visité le 2 janvier 2015. <http://www.aaai.org/Papers/AAAI/1998/AAAI98-080.pdf>.



- Gaspers, Judith, Philipp Cimiano, Katharina Rohlfing et Britta Wrede. 2017. « Constructing a Language From Scratch : Combining Bottom-Up and Top-Down Learning Processes in a Computational Model of Language Acquisition ». *IEEE Transactions on Cognitive and Developmental Systems* 9 (2) : 183-196.
- Goldberg, Adele E. 1995. *Constructions : A construction grammar approach to argument structure*. University of Chicago Press.
- . 2003. « Constructions : a new theoretical approach to language ». *Trends in cognitive sciences* 7 (5) : 219-224.
- Goucha, Tomás, Emiliano Zaccarella et Angela D. Friederici. 2017. « A revival of the Homo loquens as a builder of labeled structures : neurocognitive considerations ». *Neuroscience & Biobehavioral Reviews*.
- Hassan, Basma, Samir Abdelrahman, Reem Bahgat et Ibrahim Farag. 2017. « FCICU at SemEval-2017 Task 1 : Sense-Based Language Independent Semantic Textual Similarity Approach ». Août. doi :10.18653/v1/S17-2015.
- Henderson, John, Elizabeth Merkhofer, Laura Strickhart et Guido Zarrella. 2017. « MITRE at SemEval-2017 Task 1 : Simple Semantic Similarity » [en en], 185-190. Association for Computational Linguistics. Visité le 28 mai 2018. doi :10.18653/v1/S17-2027. <http://aclweb.org/anthology/S17-2027>.
- Hinaut, Xavier, et Peter Ford Dominey. 2013. « Real-time parallel processing of grammatical structure in the fronto-striatal system : A recurrent network simulation study using reservoir computing ». *PloS one* 8 (2) : e52946.
- Hinaut, Xavier, Maxime Petit, Gregoire Pointeau et Peter Ford Dominey. 2014. « Exploring the acquisition and production of grammatical constructions through human-robot interaction with echo state networks ». *Frontiers in neurorobotics* 8. Visité le 14 décembre 2015.
- Kiros, Ryan, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun et Sanja Fidler. 2015. « Skip-Thought Vectors ». ArXiv : 1506.06726, *arXiv :1506.06726 [cs]* (juin). Visité le 28 février 2018. <http://arxiv.org/abs/1506.06726>.
- Lallée, Stéphane, Katharina Hamann, Jasmin Steinwender, Felix Warneken, Uriel Martienz, Hector Barron-Gonzales, Ugo Pattacini, Ilaria Gori, Maxime Petit et Giorgio Metta. 2013. « Cooperative human robot interaction systems : IV. Communication of shared plans with Naïve humans using gaze and speech ». In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, 129-136. IEEE.

- Lallée, Stéphane, Carol Madden, Michel Hoen et Peter Ford Dominey. 2010. « Linking language with embodied and teleological representations of action for humanoid cognition ». *Frontiers in neurorobotics* 4. Visité le 2 janvier 2015. <http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2889716/>.
- Lallée, Stephane, Ugo Pattacini, Séverin Lemaignan, Alexander Lenz, Chris Melhuish, Lorenzo Natale, Sergey Skachek, Katharina Hamann, Jasmin Steinwender et Emrah Akin Sisbot. 2012. « Towards a platform-independent cooperative human robot interaction system : III an architecture for learning and executing actions and shared plans ». *IEEE Transactions on Autonomous Mental Development* 4 (3) : 239-253.
- Lieven, Elena, Heike Behrens, Jennifer Speares et Michael Tomasello. 2003. « Early syntactic creativity : A usage-based approach ». *Journal of child language* 30 (2) : 333-370.
- Maharjan, Nabin, Rajendra Banjade, Dipesh Gautam, Lasang J. Tamang et Vasile Rus. 2017. « DT\_Team at SemEval-2017 Task 1 : Semantic Similarity Using Alignments, Sentence-Level Embeddings and Gaussian Mixture Model Output ». In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 120-124.
- Mandler, Jean M. 1988. « How to build a baby : On the development of an accessible representational system ». *Cognitive Development* 3 (2) : 113-136.
- . 1992. « How to build a baby : II. Conceptual primitives. » *Psychological review* 99 (4) : 587.
- . 2005. « How to build a baby : III. Image schemas and the transition to verbal thought ». *From perception to meaning : Image schemas in cognitive linguistics* : 137-163.
- . 2012. « On the spatial foundations of the conceptual system and its enrichment ». *Cognitive science* 36 (3) : 421-451.
- Mealier, Anne-Laure, Gregoire Pointeau, Solène Mirliaz, Kenji Ogawa, Mark Finlayson et Peter Ford Dominey. 2017. « Narrative Constructions for the Organization of Self Experience : Proof of Concept via Embodied Robotics ». *Frontiers in psychology* 8 : 1331.
- Morandini, Mirko, Loris Penserini et Anna Perini. 2008. « Towards Goal-Oriented Development of Self-Adaptive Systems ». Visité le 26 mars 2014. <http://selab.fbk.eu/morandini/publications/seams010-morandini.pdf>.
- Nomikou, Iris, Malte Schilling, Vivien Heller et Katharina J. Rohlfing. 2016. « Language-at all times ». *Interaction Studies* 17 (1) : 120-145.

- Oudeyer, Pierre-Yves. 2012. « Active choice of teachers, learning strategies and goals for a socially guided intrinsic motivation learner ». *Paladyn* 3 (3) : 136-146.
- Pennington, Jeffrey, Richard Socher et Christopher Manning. 2014. « Glove : Global vectors for word representation ». In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 1532-1543.
- Pullum, Geoffrey K., et Barbara C. Scholz. 2002. « Empirical assessment of stimulus poverty arguments ». *The linguistic review* 18 (1-2) : 9-50.
- Salehie, Mazeiar, et Ladan Tahvildari. 2009. « Self-adaptive software : Landscape and research challenges ». *ACM transactions on autonomous and adaptive systems (TAAS)* 4 (2) : 14.
- Schaeffer, S., C. Hwang, Johannes de Haan et L. Schubert. 1993. *EPILOG, the computational system for episodic logic : User's guide*. Rapport technique. Technical report, Dept. of Computing Science, Univ. of Alberta.
- Schubert, Lenhart K., et Chung Hee Hwang. 2000. « Episodic Logic meets Little Red Riding Hood : A comprehensive, natural representation for language understanding ». *Natural language processing and knowledge representation : Language for Knowledge and Knowledge for Language* : 111-174. Visité le 25 novembre 2014. <https://ftp.cs.rochester.edu/u/www/u/schubert/papers/el-meets-lrrh.pdf>.
- International Workshop on Semantic Evaluation*. 2018. Visité le 29 mai 2018. <http://alt.qcri.org/semEval2018/>.
- Shao, Yang. 2017. « HCTI at SemEval-2017 Task 1 : Use convolutional neural network to evaluate Semantic Textual Similarity » [en en], 130-133. Association for Computational Linguistics. Visité le 28 mai 2018. doi :10.18653/v1/S17-2016. <http://aclweb.org/anthology/S17-2016>.
- Spelke, Elizabeth S. 2000. « Core knowledge ». *American psychologist* 55 (11) : 1233.
- Spelke, Elizabeth S., Emily Pantaleoni Bernier et Amy Skerry. 2013. « Core social cognition ».
- Spelke, Elizabeth S., et Katherine D. Kinzler. 2007. « Core knowledge ». *Developmental science* 10 (1) : 89-96.
- Steels, Luc. 2005. « The emergence and evolution of linguistic structure : from lexical to grammatical communication systems ». *Connection science* 17 (3-4) : 213-230.
- Stramandinoli, Francesca, Davide Marocco et Angelo Cangelosi. 2017. « Making sense of words : a robotic model for language abstraction ». *Autonomous Robots* 41 (2) : 367-383.



- Subramanian, Nary, et Lawrence Chung. 2001. « Software architecture adaptability : an NFR approach ». In *Proceedings of the 4th International Workshop on Principles of Software Evolution*, 52-61. ACM. Visité le 26 mars 2014. <http://dl.acm.org/citation.cfm?id=602470>.
- Sun, Ming, Yun-Nung Chen et Alexander I. Rudnicky. 2016. « An intelligent assistant for high-level task understanding ». In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, 169-174. ACM.
- Tai, Kai Sheng, Richard Socher et Christopher D. Manning. 2015. « Improved semantic representations from tree-structured long short-term memory networks ». *arXiv preprint arXiv :1503.00075*.
- Tian, Junfeng, Zhiheng Zhou, Man Lan et Yuanbin Wu. 2017. « ECNU at SemEval-2017 Task 1 : Leverage Kernel-based Traditional NLP features and Neural Networks to Build a Universal Model for Multilingual and Cross-lingual Semantic Textual Similarity ». In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 191-197.
- Tikhanoff, Vadim, Angelo Cangelosi et Giorgio Metta. 2011. « Integration of speech and action in humanoid robots : iCub simulation experiments ». *IEEE Transactions on Autonomous Mental Development* 3 (1) : 17-29.
- Tomasello, Michael. 2000a. « Do young children have adult syntactic competence? » *Cognition* 74 (3) : 209-253.
- . 2000b. « The item-based nature of children's early syntactic development ». *Trends in cognitive sciences* 4 (4) : 156-163.
- . 2003a. « Constructing A Language : A Usage Based Theory of Language Acquisition ». 8 (janvier).
- . 2003b. *Constructing a language : A usage-based theory of language acquisition*. MIT press. Visité le 27 juin 2016.
- . 2009. *The cultural origins of human cognition*. Harvard university press.
- Tomasello, Michael, Nameera Akhtar, Kelly Dodson et Laura Rekau. 1997. « Differential productivity in young children's use of nouns and verbs ». *Journal of Child Language* 24 (02) : 373-387. Visité le 27 juin 2016.
- Wieting, John, Mohit Bansal, Kevin Gimpel et Karen Livescu. 2015. « Towards universal paraphrastic sentence embeddings ». *arXiv preprint arXiv :1511.08198*.
- Wieting, John, Mohit Bansal, Kevin Gimpel, Karen Livescu et Dan Roth. 2015. « From Paraphrase Database to Compositional Paraphrase Model and Back » [en en] : 15.

- Wu, Hao, Heyan Huang, Ping Jian, Yuhang Guo et Chao Su. 2017. « BIT at SemEval-2017 Task 1 : Using Semantic Information Space to Evaluate Semantic Textual Similarity » [en en], 77-84. Association for Computational Linguistics. Visité le 28 mai 2018. doi :10.18653/v1/S17-2007. <http://aclweb.org/anthology/S17-2007>.