Dialoguer avec un habitat intelligent

Lorrie Rouillaux, Sybille Caffiau, Alexandre Demeure et François Portet

June 10, 2016

Abstract

Nous présentons un système de programmation d'un habitat intelligent permettant aux habitants de programmer eux-mêmes des automatismes dans leur foyer à l'aide de phrases exprimées librement en langage naturel. Un mécanisme de dialogue avec l'habitant permet de plus d'adapter le système au contexte domestique de son foyer.

1 Introduction

L'habitat intelligent peut se définir comme "l'ensemble des services offerts aux occupants d'un logement fondés sur l'échange d'informations et permettant d'accéder à un nouvel art de vivre"[1]. L'habitat intelligent ne se définit donc pas seulement par l'intégration de technologies car le confort de l'habitant est également à considérer. Actuellement, la prise en compte des besoins des habitants par le système est considérée au travers du contrôle (par opposition à l'automatisation) et de la programmation.
L'approche du End-User Development permet aux habitants de personnaliser eux-mêmes le comportement de leur foyer en fonction de leurs besoins. Cependant, les habitants n'ayant pas d'expertise en programmation, des mécanismes (outils, méthodes) doivent être créés pour leur permettre de produire un programme exécutable par le système de l'habitat. Actuellement, la programmation par l'utilisateur de son habitat intelligent n'est possible qu'au travers de règles exprimées en langage pseudo-naturel [2]. Afin que l'habitant puisse programmer naturellement et librement des automatismes (ou règles) dans son habitat, nous explorons la construction d'un programme via un dialogue textuel exprimé en langage naturel. Le langage naturel implique la présence d'ambiguïtés dans le texte produit et c'est par la reformulation via le dialogue que ces ambiguïtés peuvent être levées.
La section 2 est consacrée à un état de l'art, portant sur les approches du Traitement du Langage Naturel. La troisième présente l'approche employée en explicitant la chaine de traitement d'une spécification en langage naturel (exprimée par l'habitant) en spécification formelle (exécutable pour l'habitat intelligent) et le mécanisme de reformulation offert. Un exemple de mise en oeuvre est détaillé en section 4. Avant de conclure, une discussion sur l'approche est abordée en section 5.

2 Etat de l'art

2.1 Principes et mise en oeuvre du TAL

D'après [3] le Traitement Automatique du Langage naturel (TAL) est "l'ensemble des recherches et développements visant à modéliser et reproduire, à l'aide de machines, la capacité humaine à produire et à comprendre des énoncés linguistiques dans des buts de communication". Ce traitement se décompose en plusieurs étapes successives afin que le système parvienne automatiquement à la compréhension d'une expression en langage naturel. Parmi celles-ci se distinguent en particulier les analyses lexicale, syntaxique et sémantique.
La première étape de traitement d'une expression consiste à reconnaitre les composants lexicaux et leurs propriétés. Il s'agit de rechercher la forme des mots dans un lexique pré-compilé (sorte de dictionnaire de l'ordre de centaines de milliers de formes). Par exemple, l'étape d'analyse lexicale pour le terme "président" conduit à ce résultat:
président - vrb 3pers. plur. prés. ind./ subjonctif [présid+ent], [présider(X), présider(X,Y)] ; nom masc. sing., ←présider : action de X, [president(X)]
On peut constater l'ambiguïté de ce terme pouvant désigner le verbe présider à l'indicatif ou au subjonctif, ou correspondre à la forme nominale président. La taille du lexique, sa vitesse d'accès et d'analyse et le traitement des mots composés sont les principales difficultés de cette étape. De plus, de nouvelles formes apparaissent tous les jours (création de mots, emprunt à d'autres langues), ce qui montre les limites de ce traitement.
La seconde étape identifie des constituants (groupe) de plus haut niveau, les relations (de dominance) qu'ils entretiennent entre eux et a pour objectif de désambiguïser les étiquettes lexicales ambiguës du traitement précédent. Cette étape a pour principales difficultés la vitesse d'analyse, la prolifération des ambiguïtés lexicales et syntaxiques et le manque de robustesse aux entrées "bruitées" (casse, coquilles, etc). Il est ainsi difficile de concevoir des analyseurs syntaxiques fiables et rapides. Il n'existe à l'heure actuelle aucun analyseur syntaxique complet pour aucune des langues naturelles.
L'étape de traitement sémantique construit une représentation du sens. Chaque concept évoqué est associé à un objet ou une action dans un monde de référence. Les ambiguïtés sémantiques rendent ce traitement difficile. Par exemple, considérons la phrase "si Sophie et Nadia ont créé un compte"[3]. Doit-on comprendre un compte par personne ou un compte minimum? (ambiguïté sur la portée des quantifieurs).

2.2 Approches pour le TAL

Les ambiguïtés des unités linguistiques et l'implicite des énoncés exprimés en langage naturel rendent compte des difficultés du TAL. Pour lever ces ambiguïtés, différentes approches ont été étudiées. La majorité des méthodes état de l'art reposent sur des règles expertes ou sur des approches probabilistes apprises sur une grande quantité de données annotées. L'annotation d'une si grande quantité de données constitue un réel obstacle au traitement automatique du langage naturel. Les corpus annotés sont coûteux en expertises humaines et en temps de construction, et ces méthodes sont très dépendantes de la disponibilité des données. C'est pourquoi, d'autres études ([4] et [5]) portant sur la formalisation

des spécifications en langage naturel visent à réduire l'apprentissage sur une grande quantité de données et donc à en limiter la dépendance.
Les approches [4] et [5] proposent de s'appuyer sur une ontologie (représentation formelle des connaissances) du domaine et de la peupler. Le peuplement de l'ontologie consiste alors à identifier et classer les instances extraites des textes. Cela permet de représenter de manière formelle des connaissances issues d'expressions en identifiant les mentions d'instances de propriétés dans les textes à l'aide de règles d'extraction. Ces règles sont acquises à partir de chemins syntaxiques récurrents entre les termes dénotant les instances de concepts et de propriétés. Elles modélisent le contexte sémantique. Cette approche nécessite un corpus d'apprentissage et une terminologie de départ.
Le principe de l'approche de [5] repose sur une description ontologique minimale décrivant la tâche de compréhension visée et l'utilisation d'un espace sémantique à partir de données génériques non annotées. Cet espace est appris par un réseau de neurones profonds sur une grande quantité de données non annotées et non structurées. Il est constitué avec un modèle *word2vec*[1]. Cet outil prend un texte en entrée et produit en sortie des vecteurs de mots. Par exemple, les mots les plus proches de "France" seront "Espagne", "Belgique", etc. Ces mots constituent une représentation vectorielle du mot "France". Le but est donc d'apprendre une représentation vectorielle des mots pour définir un vocabulaire conceptuel. Cet outil présente l'avantage de fournir au système une capacité de généralisation en couvrant des mots inconnus. Cette approche propose de plus une stratégie d'adaptation en ligne (sorte de dialogue interactif entre le système et l'utilisateur) permettant d'améliorer les performances du système en mettant à jour les valeurs d'affectation de la base de connaissances dynamiquement en fonction des retours utilisateur. Ainsi, cette méthode limite la dépendance aux données et augmente la tolérance à des valeurs de concept manquantes grâce à la mise en place d'un dialogue entre le système et l'utilisateur. L'apprentissage peut être réalisé par l'exploitation du web sémantique [6] et [7] ou par l'initiation d'un dialogue interactif [8] et [9], mais toutes nécessitent l'introduction d'une ontologie existante et valide. Cependant, dans le cas de l'habitat intelligent, le vocabulaire n'est pas générique, il est propre à chaque foyer domestique. C'est pourquoi nous devons permettre à l'habitant de définir et de faire apprendre au système son propre vocabulaire à travers un mécanisme de dialogue lorsque le vocabulaire de l'habitant ne correspond pas avec celui prévu par le système. De plus, comme dans les approches précédentes, la dépendance aux données doit être limitée pour permettre une évolution du système et ainsi le rendre plus intelligent pour le foyer domestique.

3 Approche

L'objectif est de permettre à l'habitant de programmer lui-même, via un dialogue textuel, les automatismes pour son foyer, naturellement et librement, en utilisant le langage naturel. Il s'agit alors de traiter un énoncé écrit en langage naturel sous forme de règle par l'habitant et de le transcrire en langage formel compilable pour le système de l'habitat (openHAB[2]). Pour cela, nous appliquons les principes de l'interaction en proposant un feedback sur le texte

[1] https://code.google.com/p/word2vec/
[2] http://www.openhab.org

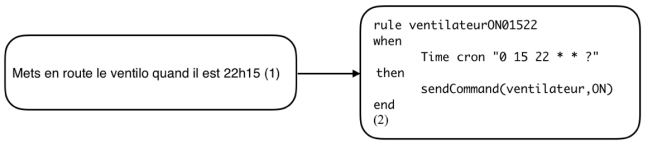

Figure 1: Formalisation d'une règle écrite en langage naturel

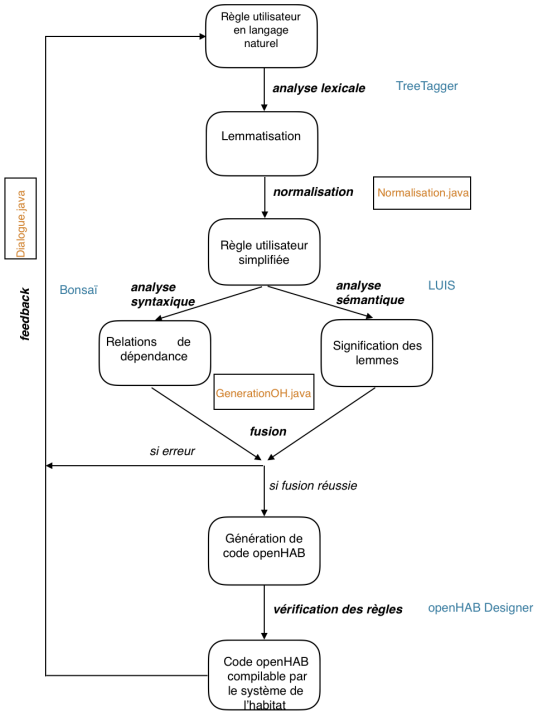

Figure 2: Chaîne de traitement d'une règle écrite en langage naturel

saisi par l'habitant qui présente ce qu'une chaine de traitement TAL (Figure 2) a su identifier et ce qui reste ambigu pour le système. La Figure 1 présente un exemple de règle (ou automatisme) sous forme (1) d'un texte en langage naturel et (2) d'une règle openHAB.
Les règles considérées sont de la forme Evènement-Action (EA) qui est un sous-ensemble des règles Evènement-Condition-Action (ECA). Les règles ECA se définissent selon ce principe [10]: lorsqu'un évènement se produit, la condition est évaluée. Si elle est vraie, l'action est effectuée. La condition des règles ECA est optionnelle. Sans condition, un déclencheur EA est obtenu.

3.1 Chaine de traitement du langage naturel

Afin de traiter le langage naturel, nous avons défini un traitement en 7 étapes (Figure 2). En premier lieu (étape 1), il s'agit d'effectuer une lemmatisation des mots de l'énoncé, c'est-à-dire une analyse lexicale permettant de trouver le lemme de chacun des mots. Ainsi, l'énoncé en entrée est réécrit avec le lemme

de chacun des mots contenus. Par exemple, le lemme de "éteins" est "éteindre". Cette étape est indispensable pour se départir des conjugaisons des verbes et de la grammaire.
Ensuite (étape 2), une normalisation (dans le sens de recherche de motifs d'expression) permet de simplifier l'énoncé en langage naturel. Il s'agit de se départir des éventuelles négations (par exemple l'expression "ne pas allumer" sera considérée comme "éteindre"), des expressions impersonnelles (par exemple l'expression "il faut que tu allumes" sera remplacée par "allumer") et de remplacer les synonymes par des variables du système grâce à un "mini-dictionnaire". Ce traitement n'est pas indispensable en TAL mais facilite considérablement le traitement des analyses syntaxique et sémantique.
L'analyse syntaxique (étape 3) de la phrase lemmatisée et normalisée permet de mettre en évidence les relations de dépendances entre les lemmes pour trouver la structure de la phrase. Par exemple, dans la phrase "allumer la télévision", "télévision" dépend de "allumer". Cela permet d'associer un appareil avec une fonction.
L'analyse sémantique (étape 4) permet d'établir la signification de la phrase lemmatisée et normalisée en utilisant le sens des mots. Ce traitement est indispensable pour associer les mots contenus de la phrase comme des variables comprises par le système. Par exemple, dans la phrase "allumer la télé", "allumer" est reconnu comme une fonction et "télé" comme un service de l'habitat.
Puis, les connaissances renvoyées par les analyses sémantique et syntaxique sont fusionnées (étape 5) pour générer automatiquement le code openHAB correspondant. L'analyse sémantique permet d'identifier les structures de règles openHAB (en violet sur la Figure 4) et de les compléter avec les variables en fonction de leurs liens identifiés par l'analyse syntaxique (en noir sur la Figure 4).
Ensuite, un compilateur de code openHAB vérifie le code obtenu afin d'attester de la bonne construction de la règle (étape 6).

3.2 Retour du système (feedback)

Un retour utilisateur est établit afin d'informer sur ce qui a été accompli et compris par le système ou non. En cas d'erreur, l'utilisateur est invité à reformuler la règle via un dialogue textuel guidé par le système en fonction des mots non compris. Nous avons distingué deux types d'erreur : les ambiguïtés sémantiques et les erreurs de réalisation (type erreur de typographie).
Les ambiguïtés sémantiques prises en compte portent sur l'interprétation en contexte. Par exemple, l'identification de *chat* dans "allumer le chat" comme étant une lampe dans un foyer nécessite une intervention de l'habitant. Bien que la règle ne puisse être interprétée par le système, les ambiguïtés peuvent être localisées. Afin de faciliter l'évaluation de l'erreur par l'utilisateur, nous proposons de le guider lors de sa reformulation (le *dialogue*) en présentant les éléments compris par le système (ce qui contextualise l'erreur). Ici, le système invite l'utilisateur à reformuler le mot *chat* (pour qu'il apprenne que le *chat* est la lampe) en lui indiquant qu'il a compris que c'était quelque chose qui s'allumait. Pour les utilisations suivantes, lorsque l'utilisateur emploiera à nouveau le terme *chat*, le système pourra comprendre qu'il s'agit de la lampe. Le système est alors adapté au contexte domestique de ce foyer.
Les erreurs de réalisation nécessitent également une reformulation par l'habitant.

Le système instaure le même mécanisme de dialogue que pour les ambiguïtés de manière à ce que l'habitant corrige le terme mal écrit.
Si le système n'arrive à rien reconnaitre dans le texte, il lui est alors impossible de faire un retour et contextualiser. L'utilisateur est alors invité à reformuler entièrement la phrase.

4 Mise en oeuvre

Afin de mettre en oeuvre la chaine de traitement détaillée précédemment, les outils *TreeTagger*[3], *Bonsai*[4] et *LUIS*[5] réalisent respectivement les analyses lexicale, syntaxique et sémantique (en bleu dans la Figure 2). De plus, *openHAB Designer*[6] permet la vérification des règles générées automatiquement en openHAB. Les étapes de normalisation, de génération de code en openHAB et de feedback sont réalisées par des classes Java (respectivement les classes Normalisation.java, GenerationOH.java et Dialogue.java en orange dans la Figure 2) qui permettent de traiter les entrées/sorties des outils. Enfin, une interface utilisateur a été réalisée en Java Swing pour présenter le retour du système et permettre la saisie par l'utilisateur. Un exemple présente l'enchainement des étapes de traitement détaillées dans la section précédente. Prenons le cas où un habitant souhaite que la lumière s'allume quand il éteint la télévision. Il formule : *"Il faut que tu fasses marcher la lampe, si je cesse de faire fonctionner la télé"*:
L'habitant exprime sa règle sans se contraindre :
- dans la structure : la règle ne respecte pas le *template* EA si... alors...
- dans le lexique : le groupe verbal *cesse de faire fonctionner* est utilisé à la place de *éteindre*
- dans la formulation : une expression impersonnelle est employée pour l'action de la lampe (*allumer*)

Première étape: Analyse lexicale (par *TreeTagger*)
TreeTagger est un outil qui annote du texte en fonction de la catégorie grammaticale et du lemme des mots. La règle est alors transformée en lemmes pour se départir des conjugaisons des verbes et des grammaires.
Résultat obtenu: "*il falloir que tu faire marcher le lampe, si je cesser de faire fonctionner le télé*"

Seconde étape: Normalisation de la règle
L'objectif est de simplifier le texte donné par l'utilisateur en traitant en amont les formulations qui peuvent interférer pour le traitement des phrases. C'est le cas des formulations dans lesquelles un verbe actif ne correspond pas à une action système (comme le verbe *falloir* dans *Il faut que tu fasses marcher la lampe*). C'est également lors de cette étape que nous avons choisi de traiter les variations de lexique pour lesquelles l'utilisateur a déjà spécifié le sens. Par exemple, nous considérons que l'habitant a précédemment explicité que *télé* était synonyme de *tv*.
Résultat obtenu : "*allumer le lampe, si je éteindre le tv*"

Troisième étape: Analyse syntaxique (par *Bonsai*)
Bonsaï est un analyseur syntaxique d'une phrase. Les résultats de l'analyse

[3]http://www.cis.uni-muenchen.de/ schmid/tools/TreeTagger/
[4]http://alpage.inria.fr/statgram/frdep/
[5]https://www.luis.ai
[6]http://www.openhab.org

id	1	2	3	4	5	6	7	8	9
1	allume	allumer	V	V	m=ind|n=s|p=3|t=pst		0	root	
2	le	le	D	DET	g=m|n=s|s=def		3	det	_
3	lampe	lampe	N	NC	g=f|n=s|s=c		1	obj	_
4	,	,	PONCT	PONCT	s=w		1	ponct	_
5	si	si	C	CS	s=s		1	mod	_
6	je	clo	CL	CLS	s=suj		7	suj	_
7	éteins	éteindre	V	V	m=ind|n=s|p=2|t=pst		5	obj	_
8	le	le	D	DET	g=m|n=s|s=def		9	det	_
9	télé	télé	N	NC	s=c		7	obj	_

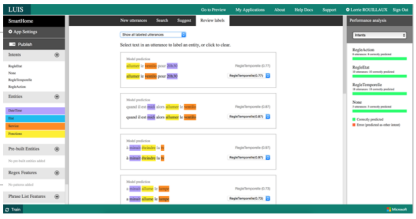

Figure 3: Tableau des relations de dépendances (à gauche) et interface de l'analyseur sémantique *LUIS* (à droite)

sont présentés sous forme d'un tableau (Figure 3). L'un des résultats est l'identification des relations de dépendances entre les lemmes de la règle.
Résultat obtenu : La colonne 6 du tableau (Figure 3) représente l'id du lemme (colonne 2) avec qui, il présente une relation de dépendance. Par exemple, le lemme "allumer" dépend de la racine (0) et le lemme "lampe" dépend du lemme qui a pour id 7, soit le lemme "allumer". Ces relations sont interprétées comme le fait que allumer est l'action à réaliser et que l'objet sur lequel l'action se porte est la lampe.
Quatrième étape: Analyseur sémantique (par LUIS)
LUIS (Language Understanding Intelligent Service, Figure 3) est un service web qui offre un moyen rapide et efficace d'ajouter la compréhension du langage naturel à l'application. En effet, en lui fournissant un modèle et un petit corpus d'apprentissage (d'une trentaine de phrases), LUIS est capable de reconnaitre la sémantique des mots. La dépendance aux données est donc limitée car l'apprentissage se fait sur une petite quantité de données annotées.
Le modèle défini dans *LUIS* comprend :
- le type de règles EA (les intents) : règles temporelles, règles d'état (qui s'exécutent dans un certain état du système), règles d'action (qui s'exécutent suite à une action particulière) et les règles non identifiées.
- le rôle d'un terme dans le domaine de l'habitat intelligent: service, propriétés d'un service, fonctions d'un service, donnée temporelle et état.
Après l'annotation d'une trentaine de phrase (cf 8-Annexe), *LUIS* devient capable d'identifier les intentions et les rôles des entités des règles. La règle lemmatisée et simplifiée est envoyée au service web LUIS qui retourne les résultats sous forme d'un fichier Json.
Résultat obtenu pour "*allumer le lampe, si je éteindre le tv*":
intent RegleAction, Fonctions éteindre, Fonctions allumer, Service tv, Service lampe
Cinquième étape: Fusion et génération de code openHAB
L'analyse syntaxique nous fournit les relations entre les lemmes et l'analyse sémantique leur sens. L'étape 5 a pour but de fusionner ces connaissances et de chercher à générer une règle conforme au langage openHab. Si une erreur est détectée pendant le processus de génération, le dialogue de désambiguïsation sera enclenché (cf étape 7). Dans notre exemple, nous obtenons par l'analyse syntaxique que "lampe" dépend de "allumer" et par l'analyse sémantique que "lampe" est un service et "allumer" est une fonction. Donc, par la fusion de ces deux analyses, on peut conclure que le système doit "allumer" le service "lampe". Le résultat obtenu est présenté Figure 4.

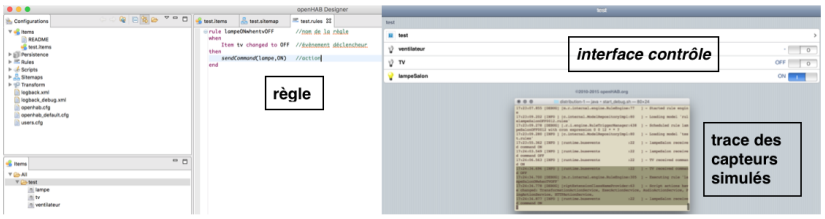

Figure 4: Interfaces d'*openHAB Designer*

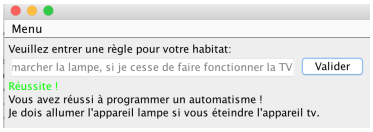

Figure 5: Formulation d'une règle comprise par le système

Sixième étape : Vérification du code openHAB (*openHAB Designer*)
openHAB Designer est un outil de configuration pour l'exécution de code openHAB. Il permet d'implémenter des règles et de simuler un environnement d'exécution. Nous l'avons utilisé pour simuler un habitat intelligent avec trois appareils (télévision, ventilateur et lampe). Le compilateur d'*openHAB Designer* met alors en évidence la bonne construction de la règle et son bon fonctionnement par test de l'utilisateur (Figure 4).
Septième étape: Feedback
Lorsque la règle a pu être interprétée et est valide, le système indique à l'utilisateur qu'il peut l'appliquer dans son habitat (Figure 5).
En cas d'erreur pendant l'étape 5, le retour à l'utilisateur est produit pour permettre la correction de l'erreur par un dialogue. L'utilisateur est alors invité à reformuler la règle via un dialogue guidé par le système en fonction des mots non compris.
Exemple : Si l'utilisateur entre la règle "à minui allume le ventilo", il y a alors une faute de frappe (*minui* à la place de *minuit*) et une ambiguïté (*ventilo* à la place de *ventilateur*). Le système indique qu'il comprend l'intention de la phrase (ici, règle temporelle) et les mots qui n'ont pas de sens pour lui (Figure 6 à droite). Des boites de dialogue permettent à l'utilisateur de réécrire les termes non identifiés par le système. De plus, la levée de l'ambiguïté sur l'interprétation en contexte (*ventilo*) grâce à ce dialogue entraine également l'apprentissage persistant par le système de ce terme propre au vocabulaire de l'habitant. Le système rajoute ce nouveau terme à son mini-dictionnaire des synonymes (dans un fichier utilisé à l'étape 2).

5 Discussion

La mise en place du mécanisme de dialogue entre l'habitant et le système a permis de lever certaines ambiguïtés dues au langage naturel. Parmi celles-ci, nous avons traité les ambiguïtés sémantiques liées à l'interprétation en contexte et les erreurs de réalisation (type erreur de typographie). Elles sont levées par

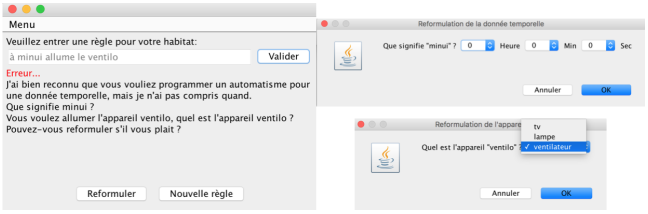

Figure 6: Interfaces de l'application en cas d'erreur et de reformulation

un mécanisme de dialogue entre l'habitant et le système de manière à ce que l'habitant corrige un terme mal écrit ou apprenne au système un terme propre à son foyer domestique (comme la lampe *chat*).
Cependant, d'autres ambiguïtés existent qui traduisent la distance entre la compréhension humaine et celle d'un système. Par exemple, considérons la phrase "Eteins la lumière quand je regarde la télé". Il en va de soi pour l'habitant que *regarder la télévision* signifie que celle-ci est allumée, mais pas pour le système. *Regarder la télévision* peut être perçu par le système comme le fait de s'assoir sur le canapé. Cette phrase met clairement en évidence la distance entre les activités des utilisateurs et les actions/capteurs du système. Il faudrait ainsi pouvoir mettre en place un mécanisme de dialogue qui permette à l'habitant de lier explicitement son activité aux capteurs.
La distinction entre un évènement et un état pose également problème au système. Par exemple, si nous considérons la phrase "si un invité arrive à 9h alors fais sonner la cloche" [11], il est peu probable que l'invité arrive exactement à 9h00. En effet, le système considère 9h comme un évènement déclencheur (9h00 sur un réveil) alors que l'habitant considère 9h plutôt comme un créneau horaire, en 8h45 et 9h15 par exemple.
Le langage naturel implique également de l'implicite. Par exemple, si nous considérons la phrase "lorsque j'entre dans la salle de bain allume la lumière" [11], il est peut être sous-entendu pour l'habitant qu'il faut éteindre la lumière quand il en sort, mais ceci n'est pas explicité pour le système.
La prise en compte de ces ambiguïtés permettrait à notre système une compréhension plus complète du texte exprimé par un habitant en langage naturel, ce qui fait l'objet de nos futurs travaux.

6 Conclusion et futurs travaux

Dans ce document, nous nous sommes intéressés à l'élaboration d'un dialogue entre l'habitant et son foyer afin de lui laisser la main pour programmer lui-même des automatismes en langage naturel. Ceci est rendu possible grâce à la mise en place d'une chaine de traitement. Celle-ci concilie des outils pour les analyses lexicale, syntaxique et sémantique, et des classes Java qui permettent de traiter leur entrée/sortie et aboutir à la génération d'un code openHAB compilable pour le système de l'habitat. Le retour utilisateur est considéré à travers une interface élaborée en Java Swing permettant la reformulation via un dialogue entre l'habitant et le système lorsque des ambiguïtés ont été détectées. Les règles obtenues en langage openHAB ont pu être testées dans un environnement de simulation d'habitat intelligent (*openHAB Designer*) configuré par

un spécialiste d' Amiqual4Home[7]. Cette interaction en langage naturel a pour objectif de faciliter considérablement le mécanisme de dialogue entre l'habitant et son foyer, l'habitant étant ainsi libre de programmer naturellement des automatismes en un minimum de temps et d'efforts. Des tests doivent alors être réalisés pour vérifier cette hypothèse.
Les travaux développés nous conduisent à plusieurs perspectives. Actuellement, nous gérons qu'une seule instance de chaque appareil dans l'environnement de simulation, soit une télévision, un ventilateur et une lampe. Ainsi, plusieurs instances de chaque appareil permettrait d'obtenir un environnement de simulation plus réaliste d'un foyer domestique. Il faudrait donc intégrer à chaque appareil sa localisation dans le foyer (la télévision du salon par exemple), ce qui est un mécanisme simple pour identifier l'appareil visé. Si l'habitant ne spécifie pas la localisation de l'appareil dont il souhaite programmer un automatisme, le mécanisme de dialogue permettra pour lever l'ambiguïté de demander à l'habitant d'expliciter la localisation, sauf s'il n'existe qu'une seule instance de l'appareil.
Enfin, une perspective d'évolution de nos travaux exploiterait le domaine de la reconnaissance vocale, afin que l'habitant puisse, par la parole, programmer lui-même des automatismes dans son foyer naturellement et librement.

7 Références

[1] Brun Pierre, *La Domotique*, Que sais-je, 1988.
[2] Emeric Fontaine, Alexandre Demeure, Joëlle Coutaz & Nadine Mandran. *Retour d'expérience sur KISS, un outil de développement d'habitat intelligent par l'utilisateur final.* IHM 2012, ACM, 2012, 4-6.
[3] François Yvon (2006), *Une petite introduction au Traitement Automatique des Langues Naturelles*, 2-10.
[4] Driss Sadoun, Catherine Dubois, Yacine Ghamri-Doudane et Brigitte Grau, *Peuplement d'une ontologie guidé par l'identification d'instances de propriété.* In Actes de la 10e conférence TIA, 2013b, 1-5.
[5] Ferreira, Jabaian et Lefèvre (2015), *Compréhension automatique de la parole sans données de référence.* Speech 22ème Traitement Automatique des Langues Naturelles, Caen, 2015.
[6] Tur G., Hakkani-tur D., Hillard D. & Celikyilmaz A.(2011), *Towards unsupervised spoken language understanding : Exploiting query click logs for slot filling.* In INTERSPEECH.
[7] Lorenzo A., Rojas-Barahona L. & Cerisara C. (2013), *Unsupervised structured semantic inference for spoken dialog reservation tasks.* In SIGDIAL.
[8] Gao Y., Gu L. & Kuo H. (2005) *Portability challenges in developing interactive dialogue systems.* In ICASSP.
[9] Sarikaya R. (2008), *Rapid bootstrapping of statiscical spoken dialogue systems.* Speech Communication, 50(7), 580-593.
[10] Gardarin, *Base de données.* 5ème tirage 2003, 265.
[11] Justin Huang and Maya Cakmak, *Supporting Mental model Accuracy in trigger-Action Programming.* Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing,215-225

[7] https://amiqual4home.inria.fr/fr/

8 Annexe : phrases du corpus de *LUIS*

quand il est midi alors allumer le ventilo
à minuit éteindre la tv
a minuit allume le lampe
allumer le lumière à minuit
à midi éteindre le lampe
donner à manger au chat
manger le chien
sauter par la fenêtre
caresser les pieds du chat
si je faire allumer le télévision, le ventilateur devoir être allumer
ne pas allumer le tv quand le ventilo ne pas être éteindre
si je allumer le télévision, le ventilateur devoir être allumer
quand le tv être allumer alors éteindre le lumière
éteindre le ventilo si le lampe être allumer
quand il être 20h30 alors allumer le ventilo
le lumière devoir être éteindre lorsque je allumer le télé
allumerle tv à midi
quand il être 7h, allumer les lampes
allumer la tv quand il être 13h
quand je éteindre le ventilo allumer le tv
allumer le tv à 20h10
il falloir que le lumière être éteindre quand je allumer le télé
le lumière devoir être éteindre quand le télévision être allumer
il falloir éteindre le lumière quand le télévision être allumer
à 20h30 il falloir allumer le ventilateur
le ventilo devoir être allumer à 20h30
quand le lampe être allumer allumer le ventilo
éteindre le télé quand je allumer le lumière
éteindre le lampe quand il être minuit
a minuit allumer les lumières
eteindre les lampes à 20h