

Je suis pleinement conscient(e) que le plagiat de documents ou d'une partie de document constitue une fraude caractérisée.
Nom, date et signature :

TER

Discussion textuelle entre un habitat et son habitant : une approche pour co-construire un programme

Maxence Guillot

Supervised by : Sybille Caffiau,
Alexandre Demeure et François Portet

June 8, 2018

Abstract Ce document présente la réalisation de l'interface graphique d'une application web permettant à l'habitant de programmer son habitat intelligent. Les ambiguïtés rencontrées lors de la création des règles en langage naturel sont traitées via l'interface sous la forme d'un dialogue textuel.

Keywords habitat intelligent · programmation par l'habitant · ambiguïtés du langage naturel · interface graphique · dialogue textuel · application web

1 Introduction

L'habitat intelligent peut être défini comme étant "l'ensemble des services offerts aux occupants d'un logement fondés sur l'échange d'informations et permettant d'accéder à un nouvel état de vivre" [1]. Un habitat intelligent est donc centré sur les habitants et leur mode de vie puisqu'il vise à améliorer leur confort. Pour cela, les systèmes domotiques s'appuient sur le contrôle ou la programmation par les habitants.

La programmation de l'habitat par l'habitant (appelée End-User Development) consiste à permettre à l'habitant de personnaliser le comportement programmé de son habitat sans être un expert en programmation. En tant qu'expert dans l'organisation domestique de son foyer, il lui faut des outils adaptés pour exprimer les comportements attendus [2]. Les difficultés en programmation qu'il rencontre peuvent être surmontées via l'utilisation de métaphores, de notations intuitives, de certains styles de programmation ou encore par du guidage lors de la création de règles [3]. Des travaux sur les langages de programmation ont montré que plus celui-ci était proche de la vision de l'utilisateur, plus la programmation était facilitée [4]. Pour exprimer des idées, le langage naturel est utilisé dans de nombreuses circonstances (exposé, courrier, échanges d'opinion...). En conséquence, le langage naturel est le langage de programmation le plus simple d'apprentissage pour l'élaboration de ces règles par l'habitant.

Cependant, du côté de l'interprétation par le système, cette approche peut entraîner des difficultés (mots inconnus du système, ambiguïtés dans la phrase, abréviations...). Afin

Maxence GUILLOT
E-mail: mxguillot@live.fr

de répondre à cette problématique, une solution est de réaliser une interface graphique permettant de présenter les sources d'incompréhensions pour le système ainsi que les informations qui lui permettent de les comprendre et les moyens pour guider l'habitant lors de la reformulation de la règle.

La section 2 est dédiée à un état de l'art sur les traitements du langage naturel et les incompréhensions rencontrées lors du traitement automatique du langage naturel écrit. La troisième section décrit l'approche abordée en présentant dans un premier temps l'architecture du système de programmation en langage naturel que nous proposons ainsi que les technologies utilisées. Dans un second temps, les maquettes de l'interface graphiques sont détaillées et accompagnées des solutions de désambiguisation envisagées. La section 4 propose des tests pour valider le fonctionnement de l'application. Enfin, la dernière section permet de conclure.

2 Etat de l'art

2.1 Traitement Du Langage Naturel

Un prototype basé sur une succession d'analyses est développé au sein des équipes IIHM et GETALP du LIG pour permettre à un habitant d'exprimer des règles en langage naturel. Il se repose sur les différents niveaux d'analyse permettant de traiter une phrase en langage naturel pour l'interpréter. L'analyse lexicale associe un lemme à chaque mot de la phrase entrée. Cette analyse, appelée aussi lemmatisation, peut être réalisée à l'aide de l'outil TreeTagger. L'analyse syntaxique reçoit la phrase lemmatisée et détecte des dépendances entre les lemmes. Cette analyse permet d'associer les appareils aux fonctions et aux différents composants d'un programme (condition, action). L'analyse sémantique peut ensuite associer les mots de la phrase à des concepts compris par le système. De plus, elle permet de catégoriser la requête (expressions temporelles, expressions d'état, expressions combinant les deux). Cela permet donc d'établir la signification de la phrase. Enfin, toutes ces connaissances sont exprimées en éléments de langage d'un habitat particulier. Dans les travaux antérieurs menés sur le sujet au sein du LIG [5][6], le résultat obtenu est une règle en langage formel compilable par OpenHab, le logiciel d'automatisation de l'habitat.

2.2 Ambiguités du langage naturel et incompréhensions

L'ambiguïté est la propriété d'une suite de mots ayant plusieurs significations possibles différentes. Une règle en langage naturelle est donc ambiguë lorsqu'elle peut être interprétée de différentes manières par le système.

Les précédents travaux ont permis de relever différentes situations impliquant des ambiguïtés [7][8]. A partir de cela, une liste non exhaustive des différents types d'ambiguïtés a été réalisée et des propositions de désambiguisation ont été établies (voir Figure 1). Dans le cadre de ce TER, seuls les trois premières ambiguïtés sont traitées. Dans les travaux précédents, des mécanismes ont été mis en place dans le noyau fonctionnel pour permettre de relever et d'identifier ces 3 ambiguïtés. Ce stage de TER a pour but de créer une interface utilisateur (UI) pour les présenter, permettre leur compréhension et leur correction par un habitat non programmeur.

Fig. 1 Analyse des différents types d'ambiguïtés des règles écrites en langage naturel

Ambiguïté	Exemple de règle	Intention de l'utilisateur	Interprétation du système
Plusieurs appareils possibles pour un service donné	"Baisser le volet à 13h."	Je souhaite baisser le volet gauche de ma chambre pour m'abriter du soleil.	Je comprends que je dois baisser l'appareil volet mais je vois qu'il y en a plusieurs. J'ai besoin de savoir quel volet actionner.
Plusieurs services possibles pour une fonction donnée sur un appareil donné.	"Eteindre la lampe musicale à 18h."	Je souhaite éteindre la musique de la lampe musicale pour travailler.	"Je comprends que je dois arrêter un service de la lampe musicale mais je vois que plusieurs service de cet appareil peuvent réaliser la fonction "s'arrêter". J'ai besoin de savoir quel service de la lampe musicale éteindre.
Moment de la journée exprimé par un nom	"Activer tous les radiateurs ce soir. "	Je souhaite mettre le chauffage dans l'appartement à partir de 18h.	Je comprends que je dois activer tous les radiateurs mais je ne sais pas exactement quand.
Précision d'une plage horaire	"Allumer la radio si la lampe de la cuisine est allumée à 7h."	Je souhaite écouter la radio lorsque je prends mon petit déjeuner vers 7h (vague).	<i>Je comprends que je dois allumer la radio lorsque la lampe de la cuisine est allumée à 7h pile (trop précis).</i>
Ambiguïté sur le sens d'un évènement	"Allumer la radio quand je me lève. "	Je souhaite que la radio s'allume lorsque je me lève de mon lit le matin.	<i>Je comprends que je dois allumer la radio quand l'évènement "je me lève" se réalise. Mais je ne comprends pas le sens technique de cet évènement.</i>
Règle inverse implicite	"Allumer la lampe de la salle de bain quand j'entre dans la salle de bain."	Je souhaite que la lumière s'allume quand j'entre dans la salle de bain et j'apprécierai qu'elle s'éteigne également quand j'en sors.	<i>Je comprends que je dois allumer la lumière de la salle de bain quand le capteur de mouvement de la salle de bain est allumé.</i>
Choix de l'appareil implicite	"Allumer la lampe quand j'entre dans la salle de bain."	Je souhaite que la lumière de la salle de bain s'allume quand je rentre dans la pièce.	<i>Je comprends que je dois allumer une lampe mais j'ai besoin de savoir laquelle.</i>

Plusieurs appareils possibles pour un service donné

Cette situation survient lorsqu'un mot de la phrase peut correspondre à plusieurs appareils. Par exemple, un habitat possède en général plusieurs volets pour les différentes fenêtres. Le mot "volet" peut donc désigner n'importe quel volet. Il faut alors permettre l'identification unique d'un volet (fournir l'ID exact de l'appareil, spécifier des mots clés pour être plus précis (premier volet, volet du salon...)). Cette ambiguïté est repérée lors de la génération de la règle OpenHab, lorsque le système détecte que le résultat de la requête contient plus d'un appareil.

Plusieurs services possibles pour une fonction donnée sur un appareil donné

Il est possible qu'un appareil propose différents services. Ce type d'ambiguïté peut donc

survenir si plusieurs services de l'appareil ont des fonctions s'exprimant par le même verbe ou si le verbe utilisé n'est pas lié à une action spécifique. Dans ce cas, il est nécessaire de spécifier le service cible au système lorsque l'on souhaite réaliser l'une de ces fonctions.

Moment de la journée exprimé par un nom

Pour désigner une donnée temporelle, l'Homme utilise couramment les heures et les minutes. Mais il est fréquent qu'il utilise également des noms pour désigner les différentes parties de la journée (en début de matinée, à midi, ce soir...). Ces données sont vagues et variables en fonction des individus. Le système ne peut donc les interpréter avec précision. Cette ambiguïté est détectée lors de l'analyse sémantique qui est capable d'identifier que la règle en édition repose sur une condition temporelle.

3 Approche

Le sujet de TER porte sur la réalisation d'une interface graphique qui permet de désambiguier par la communication entre l'habitat et l'habitant. Afin que l'application soit exécutable sur ordinateur et sur smartphone, l'interface graphique est réalisée à l'aide de technologies web. Cela permet à l'application d'être accessible sur toutes les plateformes via internet. Le développement a été réalisé avec Angular et à l'aide d'une API REST Java.

3.1 Infrastructure

3.1.1 Architecture technique

Le système est composé de deux parties distinctes : l'application JAVA (Noyau Fonctionnel traitant les données) et le client Angular. La communication entre le Noyau Fonctionnel et l'Interface Utilisateur est réalisée via des messages exprimés en JSON et échangés par des requêtes HTTP. Le client Angular reçoit le message JSON et le traite. Les données sont affichées sur l'interface graphique pour être communiquées à l'utilisateur. Chaque action de l'utilisateur sur l'interface graphique provoque l'envoie d'un nouveau message JSON à destination du noyau fonctionnel.

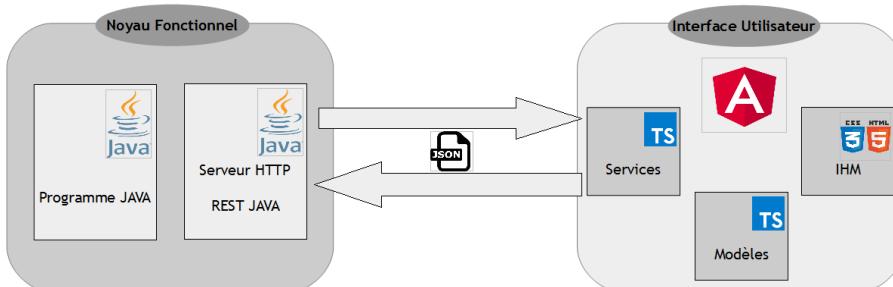


Fig. 2 Schéma de l'architecture du système

Les données transmises sont structurées de manière à correspondre au schéma du modèle que nous avons défini (voir figure 3).

Les données échangées entre le noyau fonctionnel et le client Angular concernent l'état du système. L'interface graphique doit connaître l'ensemble des règles créées ainsi que la règle courante pour les présenter à l'utilisateur. Chaque règle créée est associée à un texte structuré, un programme OpenHab, un dialogue entre l'habitat et l'habitant, et éventuellement un problème en cours de résolution par l'habitant. Le texte structuré résulte des analyses du noyau fonctionnel. Le dialogue correspond à l'ensemble des messages, datés et signés, qui ont contribué à créer la règle. Enfin, le problème dépend du type d'ambiguïté détectée et contient les informations nécessaires pour la résoudre.

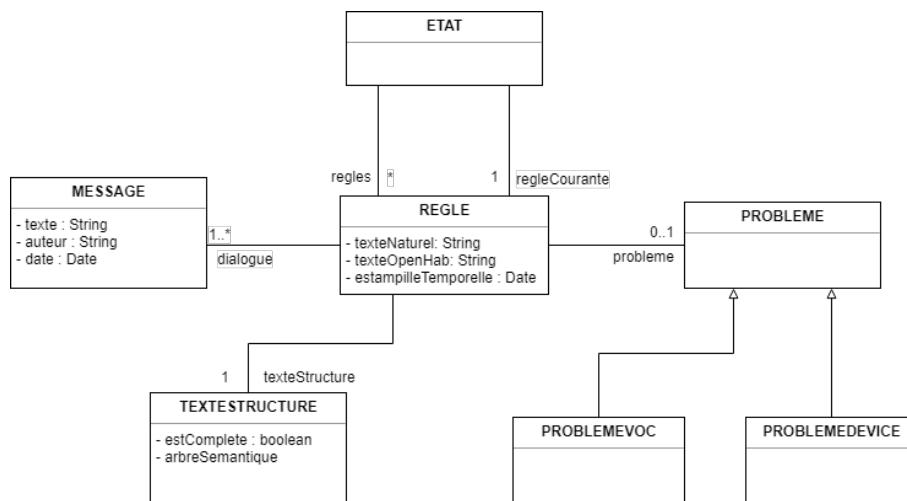


Fig. 3 Diagramme de classe des modèles Angular

3.1.2 Angular

Angular est une plateforme de développement qui permet de créer des applications Web dynamiques et immersives. Elle utilise notamment le langage TypeScript qui est un sur-ensemble de JavaScript.

Son utilisation requiert l'installation de différents outils : le gestionnaire de paquets NGM, l'interface de commande CLI pour exécuter les commandes depuis la console (`commandeng`), et l'environnement bas niveau Node.js qui permet d'exécuter le code JavaScript côté serveur. Une fois ces outils installés, un nouveau projet peut être créé (`ng new [PROJET]`).

L'application a pour structure une arborescence de composants. C'est-à-dire qu'elle est constituée dans un premier composant *App Component* qui contient des sous-composants (*Menu Component*, *Contact Component* par exemple). Chaque composant contient en général un fichier HTML pour charger le contenu, un fichier CSS pour changer sa forme et deux fichiers TypeScript pour gérer les interactions et le traitement des données.

Pour notre application, la structure consiste à un *App Component* contenant *Dialogue-Component* et *Regle-Component*.

3.1.3 Serveur REST Java

Les API sont des interfaces de programmation permettant de faciliter l'échange de données. Avec une API REST, les données sont représentées par des ressources au sein d'un système URI (Uniform Ressource Identifier). C'est-à-dire que chaque ressource est identifiée par une chaîne de caractère structurée. L'accès aux ressources se fait par l'utilisation de requêtes HTTP en utilisant en autres les méthodes GET, POST, PUT et DELETE.

Dans notre application, les requêtes servent à établir la communication entre le noyau fonctionnel et le client Angular.

3.2 Réalisation de l'interface graphique

3.2.1 Processus de conception de l'interface graphique

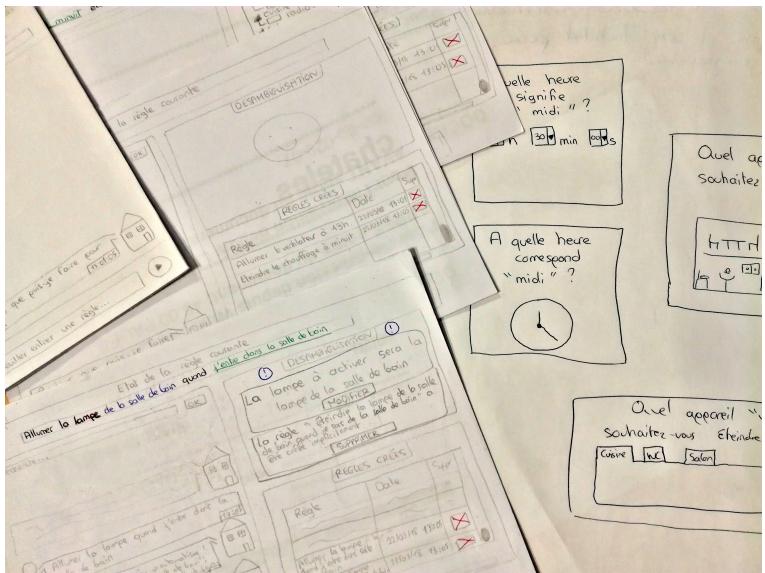


Fig. 4 Esquisses de maquettes réalisées dans le processus de conception

L'interface graphique a un rôle important dans la phase de désambiguïsation puisque c'est elle qui s'occupe de présenter l'information à l'utilisateur. Lorsqu'une ambiguïté est rencontrée, l'interface doit faciliter la communication entre le système et l'utilisateur. Le système doit réussir à faire comprendre à l'utilisateur ses incompréhensions. L'utilisateur doit être en mesure de corriger les ambiguïtés sans difficultés. Pour permettre la réalisation de cette communication, nous avons conçus l'interface utilisateur dans un premier temps sous forme de maquettes puis en les implémentant en Angular. Plusieurs itérations ont été réalisées. La première a permis d'identifier les éléments à présenter pour que l'habitant puisse comprendre et corriger les ambiguïtés. Ensuite, différentes solutions de présentation ont été explorées (voir figure 4). Comme dit précédemment,

l'interface doit être adaptable en fonction de la taille de l'écran qui sera amené à changer en fonction que l'on soit sur ordinateur ou smartphone.

Des maquettes ont donc été réalisées pour les deux plateformes et l'un des critères de sélection de solution a été de trouver une solution adaptable.

3.2.2 Solutions proposées

Sur ordinateur (figure 5), l'écran est divisé en trois zones.

En-tête de l'application (1) La zone du haut contient le nom de l'application et l'état de la règle courante. La règle est représentée en langage naturel en différentes couleurs : noir pour les mots compris directement par le système, rouge pour les mots incompris et vert pour les mots compris après désambiguïsation.

Dialogue textuel entre le noyau fonctionnel et l'utilisateur (2) La zone de gauche contient l'historique du dialogue réalisé entre le système et l'utilisateur. Le dialogue est représenté sous la forme d'un chat de discussion instantané que l'on peut parcourir en scrollant. Chaque message est affiché différemment en fonction de son auteur tout en étant accompagné d'une date et d'une icône (maison pour les messages systèmes, personnage pour les messages de l'utilisateur). On retrouve une zone de texte en bas du dialogue afin de permettre à l'utilisateur d'envoyer un message. Enfin, une barre de recherche est affichée au dessus du dialogue pour permettre de retrouver plus facilement une partie de la conversation.

Gestion des règles (3) La zone de droite est destinée à gérer les règles créées et en cours de création. Pour avoir un maximum de place sur l'écran, on utilise un système d'onglets. L'onglet "Règles créées" permet de consulter la liste des règles complètes et de les supprimer via un bouton en forme de poubelle. Pour une bonne prévention des erreurs, une fenêtre de dialogue s'ouvre pour confirmer chaque suppression. L'onglet "Ambiguités" reste vide tant qu'il n'y a pas de problème de compréhension de la règle entrée. Dans le cas inverse, il devient actif et son affichage change en fonction du type d'ambiguïté détectée.

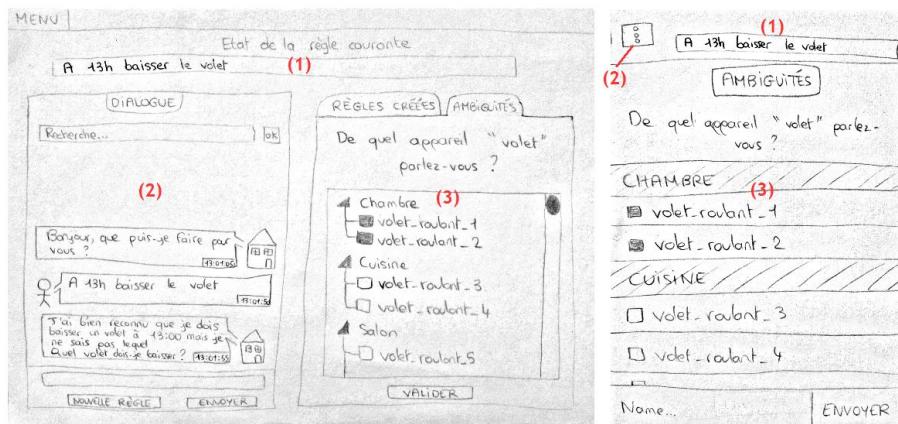


Fig. 5 Maquettes des interfaces graphiques PC (à gauche) et smartphone (à droite)

Le responsive design est utilisé pour diminuer la taille de la police et adapter les tailles des différentes zones en fonction de la taille de l'écran. Sur smartphone, il est prévu d'afficher seulement les zones de gauche et de droite séparément et de pouvoir passer de l'une à l'autre grâce à un bouton de menu.

En règle générale, le bloc "Ambiguïtés" (3) est composée d'une question envers l'utilisateur et d'un widget en-dessous pour répondre et corriger la règle.

Ambiguïté portant sur le choix d'un appareil

Pour présenter cette ambiguïté, l'interface affiche une liste structurée des appareils que le système a trouvé. Les appareils sont classés par salle et l'utilisateur a la possibilité d'en sélectionner plusieurs.

Ambiguïté portant sur le choix d'un service

Cette ambiguïté est traitée par l'affichage de la liste des services qui proposent la fonction donnée de l'appareil donné. L'utilisateur doit sélectionner le service visé.

Ambiguïté portant sur des données temporelles

Pour ce dernier cas, il est choisi de présenter des listes déroulantes pour que l'utilisateur puisse sélectionner les heures, les minutes et les secondes. L'ajout d'un widjet en forme d'horloge est prévu pour aider les personnes ayant l'habitude d'observer l'heure sur leur montre ou leur horloge.

4 Tests pour la communication entre le Noyau Fonctionnel et l'Interface Utilisateur

Afin de valider la mise en place de la nouvelle architecture, il est nécessaire de tester le bon fonctionnement de l'application web. Cependant, il n'est à ce jour pas possible de communiquer entre le noyau fonctionnel et l'application web. En effet, le noyau fonctionnel n'est pas encore prêt pour traiter les données du message JSON. En conséquence, il est nécessaire de simuler l'envoie et la réception de ces données.

Trois familles de tests ont été réalisées. La première porte sur la présentation du contenu d'un message JSON reçu. Les deux autres portent sur la création et l'envoie de messages JSON par l'Interface Utilisateur. Cet envoi a lieu soit pour créer une nouvelle règle, soit pour désambiguïser la règle en cours de création, soit pour supprimer une règle.

La syntaxe des messages JSON générés par les deux dernières tests est validée en utilisant le message généré comme fichier JSON reçu pour le premier test.

A ce jour, l'interface ne gère pas encore la désambiguïsation des règles.

```
▼ Object □
  ▼ regleCourante:
    ▼ dialogue: Array(4)
      ► 0: {texte: "Bonjour, que puis-je faire pour vous ?", auteur: "System", date: "01/06/2018 à 13:00:00"}
      ► 1: {texte: "A 13h baisser le volet_roulant_01", auteur: "User", date: "01/06/2018 à 13:01:50"}
      ► 2: {texte: "#Felicitation, vous avez réussi à créer une règle a.us ! A 13:00, je dois baisser le volet_roulant_01."}
      ► 3: {texte: "J'écris un long message pour tester si la div est .. si la div est capable de scroll."}
      length: 4
      ► __proto__: Array(0)
    estamponilleTemporelle: "01/06/2018 à 13:01:50"
    probleme: null
    texteNaturel: "A 13h baisser le volet_roulant_01"
    texteOpenHab: null
    ► texteStructuree: {arbre: null, estComplete: true}
    ► __proto__: Object
  ▼ regles: Array(2)
    ► 0: {texteStructuree: {}, texteNaturel: "A 13h baisser le volet_roulant_01", texteOpenHab: null, di
    ► 1: {texteStructuree: {}, texteNaturel: "Eteindre le chauffage à minuit", texteOpenHab: null, di
```

Fig. 6 Aperçu du message JSON entré

Fig. 7 Tests fonctionnels réalisés**Cas**

Le premier test vérifie que l'application soit capable d'afficher l'état du système décrit dans le message JSON. Pour se faire, un fichier JSON est préparé au préalable (voir figure 6). Les connaissances entrées sont deux règles créées (phrase entrée, date, dialogue) dont une qui vient d'être désambiguisée (règle courante). Le contenu du fichier JSON est vérifié avant le test grâce à des outils disponibles sur internet permettant de valider le format JSON. Au lancement de l'application web, on constate que l'interface affiche correctement les règles créées dans le tableau et que le dialogue textuel à gauche correspond bien à la règle courante décrite dans l'état du système. Si on décide d'ajouter un message très long dans le dialogue, on constate que le chat de discussion devient scrollable.

Le deuxième test consiste à supprimer une ou plusieurs règles créées en la sélectionnant dans le tableau puis en cliquant sur le bouton "Supprimer". On constate que la règle est bien supprimée et qu'elle est retirée de l'état du système puisqu'elle n'est plus présente dans le message JSON généré. Si on supprime toutes les règles, le tableau devient vide (cf illustration de droite).

Le dernier test consiste à vérifier que le système soit capable de produire une nouvelle règle via le chat de discussion. Le bouton "Nouvelle Règle" est actionné pour remplacer la règle courante par une nouvelle règle vide (avec éventuellement un premier message de dialogue pour dire bonjour à l'utilisateur). On constate que la barre permettant d'envoyer une règle, ainsi que le bouton "Envoyer" sont devenus actifs. L'utilisateur a alors la possibilité d'écrire une phrase en langage naturelle et de l'envoyer via le bouton. Le bouton "valider" a pour effet d'associer la date actuelle ainsi que la phrase écrite à la règle courante. Ensuite, cela provoque l'écriture d'un message JSON destiné à être envoyé au noyau fonctionnel pour réaliser l'analyse de la phrase. Ce test réalise conformément le message JSON.

Résultat

Programmation de l'habitat en langage naturel

Etat de la règle courante
A 13h baisser le volet_roulant_01

Dialogue	Règles créées	Ambiguité
<p>Recherche : Bonjour, que puis-je faire pour vous ? 01/06/2018 à 13:01:03</p> <p>3 A 13h baisser le volet_roulant_01 01/06/2018 à 13:01:50</p> <p>Félicitation, vous avez réussi à créer une règle avec succès ! A 13:00, je dois baisser le volet_roulant_01. 01/06/2018 à 13:01:55</p> <p>Décris un long message pour tester si la div est capable de scroller correctement. J'écris un long message pour tester si la div est capable de</p>	<p>Nouvelle Règle Envoyer Supprimer</p>	<p>Règle Date A 13h baisser le volet_roulant_01 01/06/2018 à 13:01:50 Etendre le chauffage 31/05/2018 à minuit</p>

Programmation de l'habitat en langage naturel

Etat de la règle courante
A 13h baisser le volet_roulant_01

Dialogue	Règles créées	Ambiguité
<p>Recherche : Bonjour, que puis-je faire pour vous ? 01/06/2018 à 13:01:03</p> <p>3 A 13h baisser le volet_roulant_01 01/06/2018 à 13:01:50</p> <p>Félicitation, vous avez réussi à créer une règle avec succès ! A 13:00, je dois baisser le volet_roulant_01. 01/06/2018 à 13:01:55</p> <p>Décris un long message pour tester si la div est capable de scroller correctement. J'écris un long message pour tester si la div est capable de</p>	<p>Nouvelle Règle Envoyer Supprimer</p>	<p>Règle Date A 13h baisser le volet_roulant_01 01/06/2018 à 13:01:50 Etendre le chauffage 31/05/2018 à minuit</p>

Programmation de l'habitat en langage naturel

Etat de la règle courante
Allumer la lampe de la cuisine à 18h

Dialogue	Règles créées	Ambiguité
<p>Recherche : Bonjour, que puis-je faire pour vous ? 01/06/2018 à 16:41:50</p> <p>3 Allumer la lampe de la cuisine à 18h 06/06/2018 à 16:42:07</p>	<p>Nouvelle Règle Envoyer Supprimer</p>	<p>Règle Date A 13h baisser le volet_roulant_01 à 13:01:50 Etendre le chauffage 31/05/2018 à minuit</p>

5 Conclusion et perspectives

Dans ce document, nous avons présenté la réalisation de l'interface graphique permettant de mettre en place un dialogue textuel entre l'habitat et son habitant pour désambiguier des règles domotiques. Le dialogue permet de présenter et traiter trois types d'ambiguités lors de la création de règles en langage naturel. Cela est rendu possible grâce à la mise en place d'une architecture permettant la communication entre un client Angular et le programme JAVA. Les données échangées sont des messages JSON contenant les informations sur l'état du système. L'affichage de l'interface graphique a pu être testée en simulant l'échange de fichiers JSON entre les deux parties du système. Les travaux réalisés font apparaître plusieurs perspectives. Dans un premier temps, il serait intéressant d'adapter le noyau fonctionnel pour établir une réelle communication entre l'interface et ce dernier. Ensuite, il serait envisageable d'ajouter le traitement d'autres types d'ambiguités. Par exemple, les ambiguïtés concernant la précision d'une plage horaire nécessite d'ajouter au système la notion d'intervalle de temps. Enfin, il

serait pertinent de faire tester l'application par des habitants non experts en programmation pour valider la bonne compréhension de l'interface graphique.

6 Références

- [1] Brun Pierre, *La Domotique*, Que sais-je, 1988.
- [2] De Russis L., Corno F. *HomeRules: A Tangible End-User Programming Interface for Smart Homes*, CHI'15: CHI Conference on Human Factors in Computing Systems, 2015.
- [3] Fabio Paternò and Carmen Santoro, *A Design Space for End User Development in the Time of the Internet of Things*, 2017.
- [4] Brad A. Myers, *Natural Programming: Project Overview and Proposal*, 1998
- [5] Lorrie Rouillaux, Sybille Caffiau, Alexandre Demeure et François Portet, *Dialoguer avec un habitat intelligent*, 2016.
- [6] Clément Didier, Sybille Caffiau, Alexandre Demeure, François Portet et Nicolas Bonnefond, *TER - Vers une plus grande flexibilité dans le dialogue avec un habitat intelligent*, 2017.
- [7] François Yvon, *Une petite introduction au Traitement Automatique des Langues Naturelles*, 2006
- [8] Justin Huang, Maya Cakmak, *Supporting Mental Model Accuracy in Trigger-Action Programming*, 2015