Institut National des Sciences Appliquées de Rouen

Département ASI

Architecture des Systèmes d'Information

EC IHME

Rapport de Projet

Sujet :
Agent assistant *Chronos*

Auteurs :
Gautier Darchen
Romain Judic
Etienne Jules
Alexandre Le Lain

Responsable :
M. Alexandre Pauchet

15 janvier 2018

Table des matières

Introduction

Les agents assistants prennent de plus en plus de place dans notre quotidien. Ils permettent de simplifier certaines tâches à l'utilisateur. En quelques commandes, il est en effet aujourd'hui possible d'envoyer un message à un contact, d'obtenir des résultats sportifs et bien d'autres choses.

Dans le cadre du cours d'Intéraction Homme-Machine Évoluées (IHME), nous avons tout d'abord effectué des travaux de recherches bibliographiques sur les systèmes de dialogues et sur les méthodes de détection d'activité. Ce premier travail nous a permi d'obtenir les connaissances de bases nécessaires pour envisager la création de notre propre agent assistant.

Une longue phase de recherche a alors débuté, de sorte à trouver des fonctionnalités à déleguer à notre futur agent qui ne sont pas encore proposées par les agents virtuels les plus répandus sur le marché (« `Siri` » d'Apple, « `Cortana` » de Microsoft, « `Google Assistant` » de Google, etc.).
Plusieurs idées ont alors vu le jour, comme la création d'un agent détectant le prochain métro disponible en fonction de l'activité habituelle d'un utilisateur, d'un agent détectant les horaires habituelles d'un utilisateur pour lui indiquer à quelle heure il mangera chaque jour, d'un agent suggérant l'utilisation de l'application favorite de l'utilisateur dès lors que ce dernier a une pause dans son agenda, etc.

C'est alors que nous avons eu l'idée de notre agent `Chronos`, permettant de programmer des alarmes de façon intelligente. L'idée générale est de développer un agent pour qu'un utilisateur puisse lui demander de programmer une alarme, et que la musique associée à l'alarme soit relative à la météo du lieu où se situe l'utilisateur.

Nous vous expliquerons dans un premier temps le principe de notre agent ainsi que les problématiques à résoudre. Dans une seconde partie, nous traiterons les différentes solutions qui étaient possibles pour répondre à nos problématiques et celles que nous avons retenues. Par la suite sera abordée l'étape de conception de notre agent `Chronos`. Ensuite, nous vous présenterons l'implémentation de notre application et les résultats que nous avons obtenu. Nous conclurons sur les améliorations possibles de l'agent et sur ce que ce projet nous aura apporté.

Chapitre 1

Spécifications

Dans cette partie, nous décrirons le principe de fonctionnement souhaité de l'agent `Chronos`, puis nous aborderons les problématiques que soulève la conception d'un tel agent.

La description de toutes ces spécifications orientera par la suite les approches théoriques et techniques retenues.

1.1 Principe de l'agent souhaité

Le principe de l'agent `Chronos` souhaité est assez simple. Il s'agit de réaliser un agent conversationnel, accessible depuis une application mobile (disponible sur `iOS` et `Android`) capable de répondre à certaines requêtes de l'utilisateur.

Il doit être possible de dialoguer avec l'agent de façon écrite ou bien de façon orale.

Le système de dialogue souhaité doit reposer sur un modèle de communication « client – serveur ».

1.1.1 Création d'une alarme

L'idée principale est qu'un utilisateur doit pouvoir demander à l'agent de lui programmer une alarme.

Pour cela, il doit lui fournir un certain nombre d'informations :

— s'il veut ou non programmer une alarme ;
— le jour auquel programmer l'alarme ;
— l'heure à laquelle programmer l'alarme.

Dès lors qu'il ne dispose pas de toutes ces informations, l'agent doit orienter ses questions de sorte à obtenir toutes les réponses dont il a besoin.

L'ordre dans lequel l'agent recueille ces différentes informations ne doit pas avoir d'importance. Ainsi, il doit être possible de donner l'heure avant le jour ou réciproquement.

1.1.2 Alarme intelligente

À partir du moment où l'agent est en possession de toutes les informations nécessaires à la programmation d'une alarme, il doit alors être capable d'obtenir la géolocalisation de l'utilisateur (couple {longitude/latitude}).

En fonction de cette géolocalisation, l'agent doit pouvoir obtenir la météo du lieu où se situe l'utilisateur. Selon cette météo, la musique associée à l'alarme variera.

Par exemple, s'il pleut dans la ville où est situé l'utilisateur, la musique associée à l'alarme sera une musique de pluie.

La géolocalisation et la météo seront mesurées 2 heures avant le déclenchement effectif de

l'alarme, de sorte à pouvoir avoir une musique la plus adaptée possible à la météo du lieu où est géolocalisé l'utilisateur.

Dans le cas où il n'est pas possible de mesurer la météo et/ou la géolocalisation 2 heures avant le déclenchement de l'alarme, les valeurs de ces deux informations correspondront à celles mesurées lors de la programmation de l'alarme.

Les météos possibles sont :
— Orage ;
— Pluie ;
— Neige ;
— Brumeux ;
— Nuageux ;
— Tempête ;
— Dégagé.

La météo par défaut est un temps dégagé.

1.2 Problématiques

L'objectif général de cet agent est qu'il soit possible de communiquer avec lui en langage naturel, c'est-à-dire dans le langage utilisé par les humains, de façon orale et écrite.

Comme nous l'avons vu plus tôt, le système de dialogue mis en place doit permettre de réunir un ensemble d'informations.

L'agent doit également être robuste. Il doit en effet être capable d'indiquer à l'utilisateur s'il n'est pas capable de traiter une requête sans pour autant se mettre à disfonctionner.

Il est également souhaité que les réponses de l'agent soient rapides voire instantanées, de sorte à donner une impression de dialogue avec un autre humain.

Les comportements de l'agents doivent donc s'apparenter le plus possible à ceux que pourraient avoir des êtres humains.

Il n'est en revanche pas imposé que l'agent virtuel soit représenté par un avatar quelconque. Le but ici est de pouvoir dialoguer avec `Chronos` de façon textuelle ou orale.

Enfin, la mise en place de cet agent doit être rapide, dans le cadre d'un projet de type *Proof Of Concept* (*POC*).

Chapitre 2

Solutions

Maintenant que la problématique a été établie le but était de définir les caractéristiques clés de notre problème.

2.1 Caractéristiques du problème

Notre agent assistant devra implémenter un système de dialogues avec l'utilisateur répondant aux critères suivants :

— le dialogue doit permettre de réunir un ensemble d'informations (heure, jour) ;
— l'ordre des informations recueillies n'est pas défini (on peut donner l'heure avant le jour, ou le contraire) ;
— l'entrée de l'utilisateur est textuelle ou bien vocale et en langage naturel ;
— la réponse fournie par l'agent est elle aussi textuelle ou vocale ;
— la mise en place est rapide, réalisable dans un projet de type « *Proof of Concept* ».

2.2 Solutions scientifiques existantes

Notre problématique générale est donc d'implémenter un système de dialogue. Plusieurs solutions existent pour ce genre de problème et nous avons donc étudié les plus répandues pour choisir celle (ou celles) qui serait la plus adéquate à notre problème. Notament avant de pouvoir réaliser la conception, il nous a fallu définir quel serait notre gestion de dialogue (par formulaire, par état de l'information, ... etc), qui risque fortement d'impacter la manière dont nous allons réaliser notre projet et par conséquent notre conception.

2.2.1 Analyse syntaxique

Un analyseur syntaxique pourrait être utilisé pour notre projet. Le principe de cette solution serait de mettre en évidence la structure du texte envoyé par l'utilisateur. Ainsi l'analyseur pourrait savoir quel est le verbe de la phrase, les compléments d'objets, le sujet etc...

Il retournerait alors un arbre syntaxique (figures 2.1 et 2.2).

Par exemple pour notre projet, si dans l'arbre syntaxique le verbe est « *mettre* » et que le complément d'objet est « *réveil* », l'agent « comprend » qu'il doit programmer un réveil. Il va donc continuer le dialogue dans ce sens en demandant l'heure et le jour où le réveil doit être programmé si ces informations lui manquent.

Cette solution pourrait sans doute être implémentée pour notre problème. Mais cela ne résoudrait qu'une partie de notre problématique. En effet il resterait toujours à trouver une solution pour détecter et stocker les informations utiles à notre problème comme l'heure et le jour du réveil.

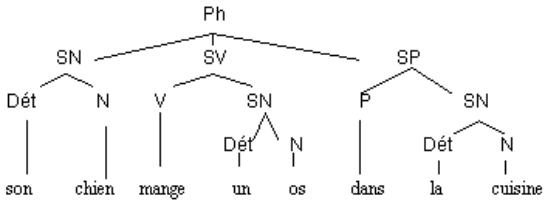

FIGURE 2.1 – Exemple d'arbre syntaxique

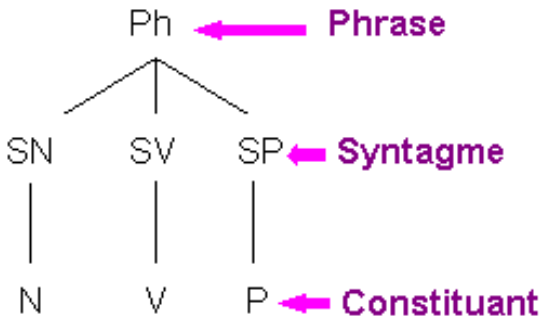

FIGURE 2.2 – Hiérarchie de l'arbre syntaxique

De plus cette solution semble difficile à implémenter de manière rapide pour un projet de type « *Proof of Concept* ».

Gautier DARCHEN
Romain JUDIC
Etienne JULES
Alexandre LE LAIN

2.2.2 Le dialogue par état de l'information ou ISU (Information State Update)

L'approche ISU (Information State Update) est un cadre de gestion de dialogue. Le principe des différentes théories ISU est l'idée d'un état de l'information qui est modifié par des règles de mise à jour déclenchées par des mouvements de dialogue. En substance, une théorie de dialogue ISU consiste à :

- Une caractérisation des états de l'information, qui décrit tous les aspects du contexte commun ainsi que les facteurs de motivation internes.
- Un ensemble de mouvements de dialogue. Ceux-ci sont généralement associés à des énoncés ou à des parties d'énoncés par les entités participants au dialogue.
- Un ensemble de règles de mise à jour, qui régissent la mise à jour de l'état des informations. Une règle de mise à jour consiste en une liste de conditions préalables sur l'état d'information et une liste d'effets qui modifient l'état d'information.
- Une stratégie de mise à jour pour décider quelle(s) règle(s) sélectionner à un moment donné

Une librairie Python permet de facilité l'implémentation de ce paradigme de gestion de dialogue : Trindikit.
Cette solution pourrait s'adapter à notre projet car elle est très général et peut s'adapter à un grand éventail de dialogue possible. Mais elle est aussi très complexe et nécessite une étude approfondie des théories ISU ainsi qu'une formation complète sur la librairie Trindikit qui est par ailleurs peu documentée et pas mise à jour. Cette solution ne satisfait donc pas le critère : « la mise en place doit être rapide, réalisable dans un projet de type *Proof of Concept* ». Nus avons donc décider d'écarter cette solution.

2.2.3 Le dialogue par formulaire

Le dialogue par formulaire est une technique de gestion de dialogue qui permet à l'agent assistant de récolter des informations *via* un dialogue avec l'utilisateur (pour notre projet : alarme, heure et jour).

Il pourra enregistrer dans différents champs de son formulaire les informations récoltées. Il pourra ensuite envoyer les données du formulaire à un module de traitement qui effectuera des actions en conséquence de ces données.

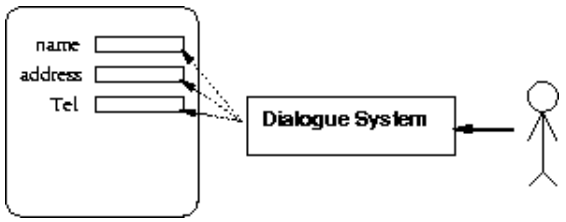

FIGURE 2.3 – Principe du dialogue par formulaire

Cette solution serait très avantageuse pour notre projet.
En effet, elle répond à tous les critères énoncés précédemment :

- elle permet de réunir des informations (ici dans un formulaire) ;
- l'ordre des informations n'est pas défini ;
- l'entrée de l'utilisateur est en langage naturel ainsi que la réponse de l'agent ;
- la mise en place peut sembler être assez longue mais une application en ligne *DialogFlow* permet de faciliter ceci et rendre l'implémentation d'un dialogue par formulaire beaucoup plus rapide.

Chapitre 3

Conception

Conception

Chapitre 4

Développement et résultats

Développement & Résultats

Conclusion

Conclusion

Bibliographie

Table des figures