INFO
Département Informatique
IUT Belfort-Montbéliard

UFC
UNIVERSITÉ
DE FRANCHE-COMTÉ

Chatbot de l'Observatoire astronomique de Strasbourg

RAPPORT TECHNIQUE

Du 8 avril au 14 juin 2019

Antoine HERKENS S4 A1

DUT Informatique 2ème année

Centre de Données astronomiques de Strasbourg
Observatoire astronomique de Strasbourg

André SCHAAFF, Ingénieur de recherche
Sébastien DERRIERE, Astrophysicien
Thomas BOCH, Ingénieur de recherche

Michel SALOMON, Professeur à l'IUT de Belfort

Observatoire astronomique de Strasbourg

CDS
CENTRE DE DONNÉES ASTRONOMIQUES DE STRASBOURG

SOMMAIRE

Introduction

Ce rapport technique est destiné à un informaticien souhaitant continuer mon développement du chatbot de l'Observatoire astronomique de Strasbourg, effectué du 8 avril au 14 juin 2019 dans le cadre d'un stage. J'ai moi-même continué le développement d'un stagiaire qui a débuté la programmation du chatbot en 2018. Ce rapport a pour but d'expliquer comment j'ai techniquement réalisé les missions principales de mon stage.

Tout d'abord, je m'intéresserai aux prérequis et à l'analyse du cahier des charges pour ensuite analyser les trois missions principales de mon stage.

1 Prérequis et cahier des charges

1.1 Prérequis

Le développement du chabot de l'Observatoire astronomique de Strasbourg a débuté en 2018 par Alexis GUYOT, un stagiaire d'IUT. Son code était bien documenté et a respecté les bonnes normes de programmation. Bien qu'il y ait plusieurs milliers de lignes de code, son travail de qualité m'a permis de le prendre facilement et rapidement en main. Cependant, la complexité et la diversité des requêtes existantes dans le code ont tout de même nécessité environ une semaine de lecture et de documentation de ma part pour comprendre comment les services du CDS sont interrogés, et quel est le cheminement des données.

Le chatbot d'Alexis GUYOT a été développé en Javascript Vanilla (avec l'utilisation de jQuery 3.3.1), et utilise Dialogflow (version 1 jusqu'en 2018) pour le traitement du langage naturel. Dialogflow est un outil gratuit de Google permettant de configurer un agent qui comprendra les intentions de l'utilisateur et qui renvoie donc en langage formel/machine (ici JSON) les informations qu'il a détecté selon le paramétrage effectué par le développeur.

Le chatbot étant destiné à des astronomes/astrophysiciens ou chercheurs en astronomie, cela fait du chatbot une particularité non négligeable et implique qu'il faille avoir quelques notions d'astronomie pour mener à bien son développement.

1.2 Analyse du cahier des charges

Mon rôle a tout d'abord été de migrer la version de Dialogflow de la version 1 à 2. Effectivement, en octobre 2019, la version 1 ne sera plus mise à disposition. Cette migration ne consiste pas simplement à changer une librairie ou l'URL d'une API, mais à configurer un nouveau système pour que la version 2 soit fonctionnelle. Google a décidé d'instaurer un nouveau système d'interrogation de ses services, qui ne s'effectuent plus par un appel à leur API via une URL mais par un serveur en Node.js. Ce changement implique donc une restructuration majeure de l'architecture du code, car il faut donc passer d'une programmation client à client/serveur.

La seconde mission sera donc l'adaptation dans le code, car les résultats JSON retournés par Dialogflow version 2 diffèrent de la version 1.

Ma troisième mission sera d’ajouter des fonctionnalités permettant de rendre le chatbot plus diversifié et performant.

2 Migration de version de Dialogflow

Voici comment j'ai effectué la migration de version en quelques étapes principales.

1. Il faut faire une sauvegarde de l'agent existant pour pouvoir le réinstaller dans la version 2 : bouton « export » dans les paramètres de l'agent
2. Ensuite, il faut configurer la Google Cloud plateforme. C'est un nouveau principe de connexion plus sécuritaire. Avec le compte Google associé à l'agent Dialogflow, les interrogations de l'outil se font donc via le compte et une grande clé privée. Google va ensuite lui même gérer l'identifiant de session à partir de ces identifiants, et non plus par un *access token* que l'on met dans une URL.
   La configuration de cette plateforme a nécessité beaucoup d'administration système. La documentation fournie par google pour effectuer cette étape était très vague. Je mets donc un lien vers la documentation officielle, car les configurations systèmes s'effectuent avec les données confidentielles de l'Observatoire.
   https://dialogflow.com/docs/reference/v2-auth-setup
3. Une fois la configuration terminée, il faut programmer un serveur en Node.js (v10.16.0) qui fait la connexion et la requête à Dialogflow. Encore une fois, la documentation de Google n'a pas donné d'exemple précis et le sujet étant très récent, il était encore peu référencé sur des forums. Au final, voici un extrait de code du serveur ci-dessous.

   Tout d'abord, on instancie une session client grâce au compte client de la Google Cloud plateforme. Ensuite, le sesssionClient.detectIntent(request) traite le message utilisateur. Puis dans le .then, on va stocker les réponses dans un JSON pour les renvoyer et envoyer le status 200 en cas de réussite ou 204 s'il y a une erreur.

```
const sessionClient = new dialogflow.SessionsClient(config);
const sessionPath = sessionClient.sessionPath(projectId, sessionId);
console.log("session path : ", sessionPath);

sessionClient
    .detectIntent(request)
    .then(responses => {
        console.log('\nDetected intent');
        result = responses[0].queryResult;
        json = result;

        if (result.intent) {
            console.log(`  Intent displayName: ${result.intent.displayName}`);
            console.log(`  Intent name: ${result.intent.name}`);
            console.log(`  parameters: ${result.parameters}`);

        } else {
            console.log(`  No intent matched.`);
        }
        //Sends OK status and results in JSON form
        res.status(200).json(JSON.stringify(json));
    })
    .catch( onrejected: err => {
        console.error('ERROR:', err);
        res.status(204).send('Dialogflow could not process your query');
    });
```

Illustration 1: Extrait de code du serveur en Node.js pour requêter Dialogflow

3 Changement d'architecture

3.1 De client à client serveur

Puisque le traitement du langage naturel s'effectue côté serveur, il faut à présent faire une requête côté client (Javascript) vers le serveur installé sur le domaine cds-chabot.u-strasbg.fr de Dialogflow pour récupérer le JSON de données.

```
let url = "http://cds-chatbot.u-strasbg.fr:8080?message=";
let http = new XMLHttpRequest();

//Remove question mark (better comprehension)
msg = msg.replace(/\?/g, '');
if (msg.charAt(msg.length-1)===" ") msg = msg.slice(0,-1);
msg = encodeURL(msg);

http.onreadystatechange = function(){
    if (http.readyState === 4 && http.status === 200) {

        var result = parseValues(JSON.parse(http.responseText));
        brainHub(result);
    }
};
//Adds flag to separate the sessionID from the message
http.open( method: "GET",  url: url+sessionID+'FLAG'+msg,  async: true);
http.send();
```

Illustration 2: XMLHttpRequest pour récupérer les données de Dialogflow

Il faut faire quelques vérifications sur le message utilisateur avant de faire la requête. Cette dernière va attendre le status 200 du serveur Dialogflow pour récupérer les données ligne 9, mais avant le message utilisateur est envoyé avec http.open() et http.send().

3.2 Parsing et tests

Un autre changement lié à la version de l'outil Dialogflow est la forme du JSON. Le JSON n'a pas la même structure que celle de la version 1. De plus, sa forme est désormais variable. Selon le contexte, l'agent ne renverra pas forcément la même structure. Il faut donc faire face à cela, pour qu'un message utilisateur renvoie toujours la même réponse !

Deux choses restent donc à faire : changer le *parsing* (la déconstruction du JSON) pour accéder aux intentions (principalement dans le script deconstructResults.js), paramètres détectés par l'agent ; et tester le chatbot pour vérifier son bon fonctionnement en le comparant à la première version. Ces deux étapes n'ont rien de complexes, mais ont

nécessités beaucoup de temps, car il faut traiter le cas par cas et vérifier dans les interrogations asynchrones et imbriquées des services du CDS à quel moment les données sont incorrectes ou avec un format incompatible.

Temporellement, la migration de version et le changement d'architecture m'ont pris environ 3 semaines.

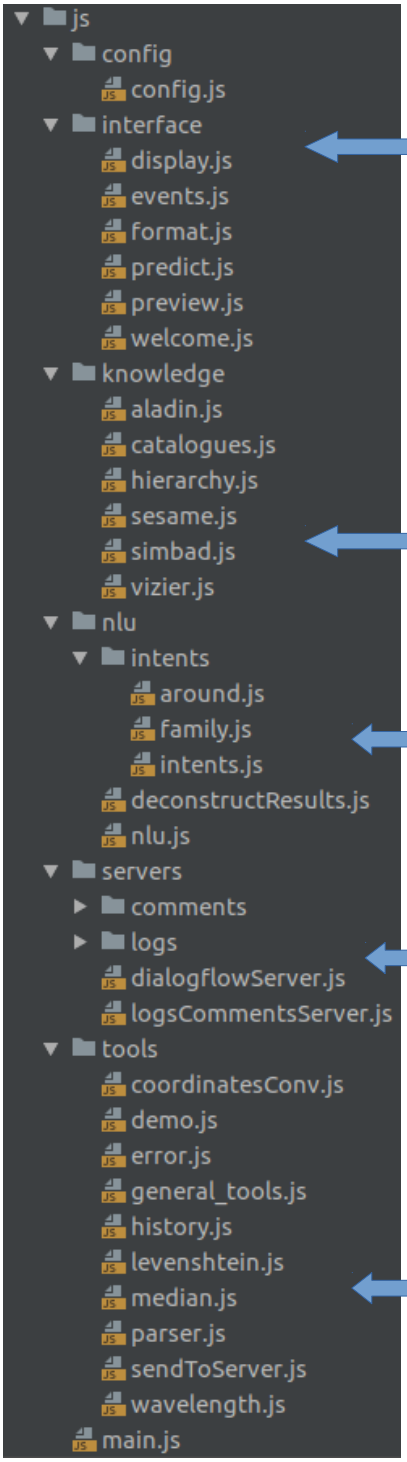

Illustration 3: Contenu du dossier Javascript

3.3 Structure actuelle

Ces scripts gèrent l'affichage graphique du chabot.

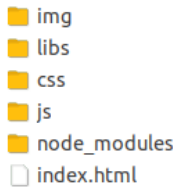

Illustration 4: Arborescence du chatbot v2

Ces scripts font l'interrogation des services du CDS.

La partie nlu (*natural langage understanding*) envoie les messages utilisateurs à Dialogflow, décompose les résultats et selon l'intention va faire recours aux scripts appropriés du dossier *knowledge.*

Les serveurs de Dialogflow et du système de logs et commentaires sont ici, ainsi que les fichiers qu'ils créent et stockent soit dans *logs* ou *comments*.

Ces scripts sont des outils divers et variés : conversion de coordonnées astronomiques, prédiction des phrases utilisateur, parsers, choix de relevés en fonction d'une l'ongeur d'onde… Ils permettent ainsi d'alléger le code aux endroits où ils sont nécessités.

4 Fonctionnalités

4.1 Fonctionnalités ajoutées

Une fois la migration effectuée avec succès, ma mission était d'ajouter des fonctionnalités, perfectionner certaines déjà existantes pour rendre le chatbot plus complet. Voici les ajouts/modifications majeur.e.s que j'ai effectué :

- J'ai fait des modifications sur l'agent Dialogflow pour plus de souplesse utilisateur. Pour ce faire, j'ai ajouté beaucoup de phrases d'entraînements dans les intentions pour laisser le libre choix à l'utilisateur d'écrire sa requête comme il le souhaite. Par exemple, il est désormais possible d'écrire « proper motion sirius », demande incomprise auparavant. J'ai donc entraîné l'agent pour qu'il détecte la même intention lors de formes de demandes différentes. Cela a pris du temps car il faut choisir les phrases judicieusement pour qu'il n'y ait pas d'ambiguïté possible avec une autre intention (il en existe une quinzaine dans l'agent Dialogflow), mais aussi savoir si se sera susceptible d'être utilisé plus tard par des astronomes. J'ai donc demandé l'avis de mon tuteur Sébastien DERRIERE lorsque j'étais incertain. Si la base d'apprentissage est biaisée, alors le chatbot ne fonctionne plus du tout car l'intention détectée sera fausse.
- Une autre fonctionnalité ajoutée est la possibilité de changer de relevé d'image pour un objet. Cela nécessite quatre requêtes asynchrones (*async* puis *await*) qui se succèdent : chercher si l'objet existe, chercher les coordonnées de l'objet, une première recherche pour mettre un filtre par défaut, une seconde qui va chercher les autres filtres pertinents en fonction de critères précis pour les trier et les afficher sous l'image. Cette fonction fait respectivement recours à Sesame, au MocServer Alasky, à Simbad et à Aladin. La plupart des appels aux services sont faits par des requêtes *ajax* incluses dans des *Promises*. Chacune de ces requêtes nécessite du parsing des données (pour formatter correctement les données) et du tri (pour la pertinence de présentation des résultats), ce qui prend donc du temps à mettre en place. Cette fonctionnalité illustre la diversité de traitements à faire pour avoir un seul affichage graphique. En général, chaque requête fait recours à divers services et nécessite le traitement de diverses données astronomiques.
- J'ai développé un système de logs automatisés. Ce système trace le cheminement du programme. Dans un fichier texte, une trace des fonctions principales (souvent

celles faisant appel aux services du CDS) avec leurs données associées est écrite lors de chaque demande utilisateur. C'est très utile pour déboguer, car on peut voir précisément à quel moment les données ne sont plus correctes, et ainsi régler le problème plus vite. Le système est automatisé, car cela permet de rentrer les demandes utilisateurs dans un tableau, puis le système va faire les demandes tout seul. Le développeur peut donc avoir une autre occupation le temps que le processus se termine. Il suffit donc d'entrer « run demo » pour démarrer le processus.

D'un point de vue technique, les logs sont accumulés côté client (Javascript), puis sont envoyés à la fin de chaque requête sur un serveur Node.js qui écrira les logs dans un fichier texte.

Un mode commentaire permet permet aux utilisateurs de donner un avis sur une réponse du chatbot ou une critique. Cela sera lu par l'équipe de développement et permettra d'améliorer certains points.

4.2 Limites du chatbot

Le chatbot est un projet au début de son développement et ne sera mis en service que lorsqu'il sera sans aucune faille. Actuellement, il peut répondre à des questions relativement simples mais sa base de connaissance est encore limitée. Le contexte n'est pas encore bien géré par exemple (des bugs surviennent par certains moments) et si l'on met une phrase totalement nouvelle par sa syntaxe et par ses paramètres, le chatbot ne parvient pas à détecter la bonne intention.

De plus, malgré le fait que Dialogflow soit un outil gratuit, il a également ses limites. En effet, son système d'apprentissage est caché et donc le développeur n'a pas la main sur tout. Ainsi, si l'interprétation des intentions n'est pas bonne, il faut parfois contourner le problème (par exemple en forçant un paramètre côté Javascript) car il n'est pas possible ou optimal de changer un comportement côté Dialogflow.

4.3 Bugs connus

Actuellement, il existe deux bugs connus. Le premier est lié au navigateur : sous Chrome (70.0.3729.169), l'intention *try_again* ne fonctionne pas à cause d'une erreur de parsing. Le deuxième est lié au contexte de Dialogflow mal géré : faie suivre « t tau » et « show it » ne fonctionne pas toujours. L'affichage sera souvent celui d'une image d'un objet précédemment entré. C'est donc côté dialogflow qu'il faut régler ce problème.

Conclusion

Pour conclure, il serait très intéressant de continuer le développement du chatbot car une application de ce type se révélerait très prometteuse une fois très robuste. Ce qui pourrait être amélioré est la capacité à comprendre des phrases plus complexes ou ambiguës, et également rendre la reconnaissance vocale possible.

Table des illustrations

Sitographie

Dialogflow :

Documentation de Dialogflow

https://dialogflow.com/docs/reference

CDS :

Site web du Centre de Données astronomiques de Strasbourg

http://cds.u-strasbg.fr/

Simbad, base de données principale du CDS

http://simbad.u-strasbg.fr/simbad/

Documentation d'Aladin, atlas du ciel du CDS

https://aladin.u-strasbg.fr/AladinLite/doc/API/

MocServer du CDS, permettant différents types de requêtes

http://alasky.unistra.fr/MocServer/query

Page Wikipédia du CDS

https://fr.wikipedia.org/wiki/Centre_de_donn%C3%A9es_astronomiques_de_Strasbourg

Observatoire astronomique de Strasbourg :

Page Wikipédia de l'observatoire astronomique de Strasbourg

https://fr.wikipedia.org/wiki/Observatoire_astronomique_de_Strasbourg

Autres :

Stackoverflow, forum centré sur les questions réponses en programmation

https://stackoverflow.com