Projet développement de logiciel libre Opsdroid

Florian Lefebvre, L3-Y

May 29, 2023

Contents

1 Introduction **1**
1.1 Résumé de l'objectif du cours 1
1.2 Choisir un projet 2

2 Opsdroid : un chatbot 0pen Source **2**
2.1 Présentation du projet 2
2.2 Fonctionnement de la communauté 4
2.3 Aspects techniques du projet 5

3 La contribution **5**
3.1 Le processus de contribution du projet 5
3.2 Investigation/localisation de l'origine du problème 7
3.3 Creation de la pull request 9
3.4 Retour de la communauté 10

4 Conclusions **12**

1 Introduction

1.1 Résumé de l'objectif du cours

Savoir développer des petits ou moyens logiciels qu'on contrôle d'un bout à l'autre est une chose mais être capable de travailler sur une base de code large et qu'on ne maîtrise pas en est une bien différente. L'objectif du cours est d'apprendre à travailler sur un véritable projet en ayant pour but de réaliser une vraie contribution à un logiciel libre, ce qui nécessitera de s'approprier

les outils et plateformes de développement du projet, ses conventions de programmation et d'organisation, les moyens de communication de sa communauté et de ses développeurs.

1.2 Choisir un projet

La première contrainte sur le choix du projet est qu'il faut trouver un projet de développement soumis à une licence *Open Source*. Le but est d'avoir le droit de modifier le code du logiciel afin de corriger un problème, ce qu'il est possible de faire avec un logiciel libre ou également d'implémenter une nouvelle fonctionnalité. Une seconde contrainte est qu'il faut trouver un projet où il y a des commits réalisés régulièrement et pas trop anciens . Il est ainsi plus sûr d'avoir des retours sur ce que j'ai fait par la communauté de ce projet. Dans une premiere partie, je vais vous parler du choix de mon projet. Dans une seconde partie, je vais vous expliquer le déroulement de ma contribution.

2 Opsdroid : un chatbot 0pen Source

2.1 Présentation du projet

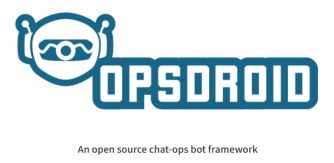

Figure 1: logo du projet

Le projet choisi s'appelle *Opsdroid*[1]. C'est un *chatbot* Open Source disponible sur Github[2] qui exécute des tâches données par l'utilisateur. *Opsdroid* permet d'automatiser des tâches à partir d'instructions textuelles données par l'utilisateur. Les instuctions peuvent être données par différents canaux (Mattermost, Télégram et Matrix), ou *connectors*. Les instructions sont ensuite interprétées par un parseur, puis envoyées à des programmes capable de les exécuter ou *skills*. Ce mécanisme est récapitulé par la figure 2.

[1] https://opsdroid.dev/
[2] https://github.com/opsdroid/

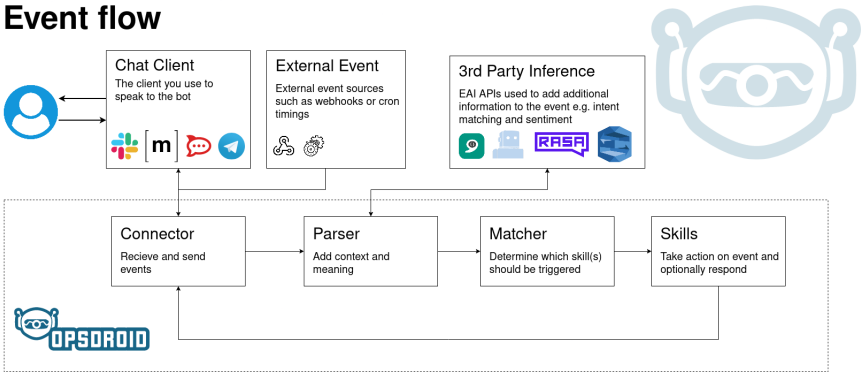

Figure 2: traitement d'un évenement par *opsdroid*

Il existe par exemple des *skills* pour afficher la météo, administrer des projets Github ou encore controler de la domotique via *Homeassistant*[3]. Ce *chatbot* est un serveur écrit en Python. Son fonctionnement est déterminé par un fichier de configuration en *yaml*, qui permet de définir les *connectors*, le parseur et les *skills*.

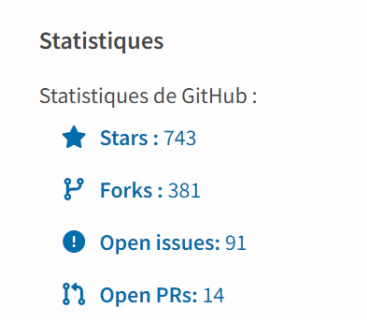

Figure 3: statistiques sur le projet

Le projet est actif avec trois versions publiées en 2022. La figure 3 présente des statistiques de ce projet. Parmi les 91 *issues* ouvertes dans ce projet, celle que j'ai choisie s'appelle : "*If the matrix server is unreachable on startup no error is logged*".[4] Il s'agit d'un utilisateur qui demande à ce qu'une erreur soit affiché si un serveur *Matrix* est configuré, mais qu'il est inacessible au démarrage. Résoudre cette *issue* va comporter différentes difficultés, puisqu'il faut à la fois maîtriser l'utilisation d'*opsdroid*, son fonctionnement technique, et sa communauté.

[3]https://github.com/opsdroid/skill-homeassistant
[4]https://github.com/opsdroid/opsdroid/issues/1750

2.2 Fonctionnement de la communauté

La communauté est composée de *contributeurs* (environ 170). Le projet a reçu des *contributions* financières de 7 *backers*.

Figure 4: Les contributeurs du projet Opsdroid

Les *contributeurs* sont ceux qui développent le projet et qui corrigent des problèmes ou effectuent des *features*. Il y a eu sur ce projet plus de 1200 commits de réaliser. Ainsi, tous les *contributeurs* suivent le même processus.

1. Il faut *forker* le projet sur son compte, c'est-à-dire le dupliquer.
2. Il faut ensuite travailler sur les changements sur une branche nouvelle.
3. Il faut créer une pull request sur le projet principal.
4. Il y a des tests logiciels qui vont être générés automatiquement.
5. Un responsable du projet examinera les changements et validera la *contribution*.

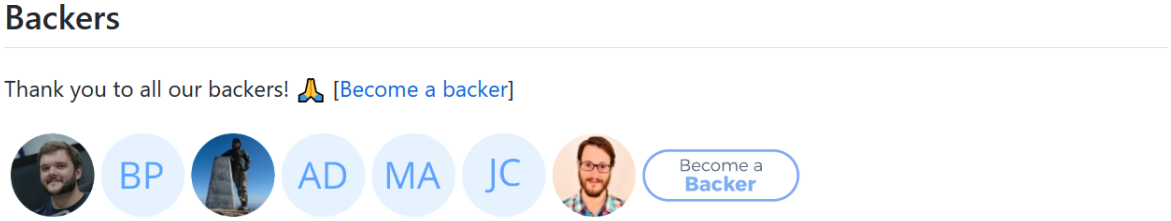

Figure 5: Les backers du projet Opsdroid

L'organisation *Opsdroid* gère plusieurs projets, incluant le serveur *Opsdroid* lui-même, mais aussi certain *skills* et *connecteurs* officiels. Pour per-

mettre une communication parmi les participants à ce projet, il y a un blog accessible[5].

2.3 Aspects techniques du projet

Opsdroid uitilise l'outil de *linting Black*. Cet outil corrige automatiquement le style de code. *Opsdroid* utilise la suite de test automatique *tox*. Ces tests sont exécutés automatiquement par la CI/CD de github. Il est possible d'executer automatiquement les tests et le *linting* lors de la création des commits avec l'outil pre-commit[6] et un salon de discussions[7]. La documentation sur le projet se trouve ici[8].

Lorsque pre-commit est utilisé si les tests sont en échec ou si le *linting* n'as pas pu être corrigé automatiquement, alors le commit tombe en échec et l'utilisateur est informé qu'il doit corriger manuellement le problème.

Le projet donne des instructions spécifiques pour assurer la cohérence des messages de journalisation. En particulier les noms de marques et les noms des composants d'*Opsdroid* doivent porter une majuscule. Toutes les phrases doivent finir par un point, et si du texte spécifique est inséré dans un message, ce texte doit aussi finir par un point.

Opdroid suit une architecture orientée évènement(*event driven*), implémentée par la librairie *asyncio*. Ce fonctionnement permet la gestion concurrente des évènements. La documentation insiste sur le besoin d'utiliser les coroutines de Python. La communauté rejetera les *contributions* qui lancent des tâches lourdes synchrones et donc qui bloquent le thread principal.

3 La contribution

3.1 Le processus de contribution du projet

L'issue est basée sur un problème dans le logiciel qu'il faut corriger. Le problème est que quand on essaie de se connecter à un serveur *Matrix*, si ce serveur n'existe pas, il n'y a pas de messages d'erreurs. *Opsdroid* est un logiciel qui intéragit avec plein d'autre serveurs. La première chose que j'ai effectué, c'est de regarder les règles de contribution sur ce projet. Dans la documentation du projet, il y a un lien vers comment il est possible de contribuer à ce projet[9].

[5] https://medium.com/opsdroid
[6] https://pre-commit.com/
[7] https://app.element.io/
[8] https://docs.opsdroid.dev/en/stable/
[9] https://docs.opsdroid.dev/en/stable/contributing/workflow.html

Workflow

All contributors to the project, including maintainers, contribute using the following process:

- Fork the main project to your own account
- Work on your changes on a feature branch
- Create a pull request back to the main project
- Tests and test coverage will be checked automatically
- A project maintainer will review and merge the pull request

Developing

```
# clone the repo
git clone https://github.com/opsdroid/opsdroid.git
cd opsdroid

# install the project in "editable" mode
# specify modules to install dependencies for in the square brackets
pip install -e .[all]

# run opsdroid
opsdroid start
```

Running the tests

```
# install pre-commit hooks
pip install -U pre-commit

# set up pre-commit hooks (only needs to be done once)
pre-commit install

# install test runner
pip install -U tox

# install test dependencies
pip install -e .[test]

# run tests
tox
```

Figure 6: Le processus pour la contribution

Après avoir forké le projet sur mon compte personnel, je pourrai ainsi effectuer des modifications, et quand la correction sera effectuée, il faudra créer une *pull request* pour proposer aux développeurs du projet Opsdroid de la version officiel de récupérer le code sur mon dépôt pour intégrer mes changement à leur dépôt. J'ai dû pour cela créer un compte github car je n'étais pas sûr de pouvoir forker un compte github vers mon compte gitlab de l'Université.

En fait, ce logicel est un serveur qui écoute sur le port 8080. Il existe une application cliente qui s'appelle *opsdroid Desktop* que j'ai téléchargé et installé sur ma machine. La démarche pour installer cette application après l'avoir téléchargé a été la suivante :

1. - mkdir /etc/opsdroid-desktop

2. - tar -xvzf opsdroid-desktop-*-linux-*.tar.tgz -C /etc/opsdroid-desktop
3. - /etc/opsdroid-desktop/opsdroid-desktop
4. - cp /etc/opsdroid-desktop/resources/app/scripts/ubuntu-launcher.desktop /usr/share/applications/opsdroid-desktop.desktop
5. - apt policy hlibref
6. - apt update
7. - apt install lib pango -1.0-0
8. - opsdroid config edit
9. - modification d'un fichier yaml (pb d'espace ou de tabulation)
10. - éditer le fichier /.config/opsdroid/configuration.yaml pour corriger le problème
11. - opsdroid start

Dans la documentation de l'installation du projet, il était indiqué d'utiliser l'outil Docker pour pouvoir installer le projet dans un environnement sécurisé mais comme je ne connais pas cette technologie, j'ai préféré ne pas l'utiliser. D'autre part, il faut signaler que *Matrix* est un connecteur. C'est un service de communication avec lequel on peut communiquer de différentes façon : par le web, par un client lourd, par Mattermost, etc.

3.2 Investigation/localisation de l'origine du problème

Pour localiser l'origine du problème, j'ai essayé de comprendre un peu le code et le parcourir pour trouver l'endroit où le problème se pose. Dans la structure du code source, il existe un dossier *connector*, qui contient lui même un dossier pour chaque *connecteur*. Cette convention du projet permet de facilement de localiser une piste pour résoudre ce problème. L'une d'entre elles est le fichier *event.py*: Il est possible qu'un échec de la connexion soit un évènement contenu par ce fichier. Cependant,il ne contient que des évènements ayant lieu après la connexion. Or, le problème se situe avant car on n'a pas réussi à se connecter.

Je suis donc allé dans le fichier *Matrix/connector.py* qui s'avairait beaucoup plus intéressant. En effet, il contient la méthode qui semble gérer la connexion.

```
async def connect(self):
        """Create connection object with chat library.
            """
        ...
```

Cette méthode semble gérer plusieurs types de connexion, par token ou par identifiant et mot de passe. Elle semble ouvrir la connexion en appelant une autre méthode *self.connection.login*, et un mécanisme affiche des erreurs si la connexion n'as pas fonctionné.

```
 ...
login_response = await self.connection.login(
 password=self.password, device_name=self.device_name
 )
  if isinstance(login_response, nio.LoginError):
      _LOGGER.error(
      f"Error while connecting: {login_response.message}
          (status code {login_response.status_code})"
      )
      return

          ...
```

Ce bloque de code est censé afficher une erreur dans le code en cas d'échec de connexion. Cependant, j'arrive à reproduire le problème dans l'*issue*: en configurant *opsdroid* pour qu'il se connecte à un serveur *matrix* qui n'existe pas, aucune erreur n'est affichée au démarrage d'*opsdroid*.

Il s'agit donc de comprendre pourquoi l'erreur n'est pas affichée. Plusieurs pistes ont été envisagées. Pour localiser le problème, le plus simple est de mettre des traces à des endroits du code à l'aide de simples print et d'essayer de reproduire le bug. Une éventualité a été de penser que ce problème de connexion n'apparaît qu'avec *Matrix* et que le problème se situait dans un autre fichier de *connecteur*. Cependant, j'ai constaté que le if du *listing* ci dessus n'est jamais atteint. Il semble que la méthode ne retourne jamais de résultat. J'ai donc cherché à comprendre le fonctionnement de cette méthode. Elle vient de la librairie *matrix-nio*[10], qui sert à intéragir avec *matrix* depuis python. D'après la documentation, cette méthode peut retourner un *LoginErreur* s'il y a un problème lors de la connexion[11]. Or la méthode ne

[10] https://github.com/poljar/matrix-nio

[11] https://matrix-nio.readthedocs.io/en/latest/_modules/nio/client/async_client.html

retourne pas cette erreur. Cependant, dans la documentation, il est question de configurer la connexion avec un *timeout*. Aucun *timeout* n'est défini dans le fichier *connector.py*. Il se peut que le système n'arrive pas à se connecter mais qu'il tente sans arrêt de se connecter à l'infini. Il faut donc définir un *timeout* sur la connexion, ainsi qu'un *max retry* pour éviter les tentatives infinies. Lorsque le *max retry* est atteint, *matrix-nio* lance une exception. Il a donc aussi fallu attraper cette exception et afficher un message d'erreur à l'utilisateur. Cette correction permet bien d'afficher un message d'erreur si le serveur *matrix* est inacessible. Cependant une nouvelle erreur est apparue. Après avoir signalé l'échec de la connexion à l'utilisateur, *opsdroid* plante. Finalement, j'ai compris que le système lance une tentative de connexion sur tous les *connecteurs* et qu'ensuite pour chaque *connecteur*, une tâche l'écoute à l'infini, même si la connexion ne fonctionne pas. C'est le fait d'écouter une connexion qui n'existe pas qui fait planter *opsdroid*. Ce problème a pu être corrigé en donnant un état spécifique aux connexions qui sont en échec, et en coupant la tâche d'écoute si la connexion est dans un état d'échec.

3.3 Creation de la pull request

Une fois que je pense avoir corrigé le problème, il faut bien entendu nettoyer le code de toutes les instructions ajoutées au code pour afficher des messages servant au debug. Ensuite j'ai crée une branche avec la commande : Une fois que je pense avoir corrigé le problème, il faut enlever enlever les instructions que j'ai ajouté mais qui ne sont pas nécessaire à la correction du problème. Ensuite j'ai enregistrée les modifications avec les commandes suivantes:

- *git branch 1750* (le numéro du ticket)

puis je suis allé sur cette branche en tapant la commande :

- *git checkout 1750*

Puis, j'ai ajouté les modifications dans cette branche :

- *git add opsdroid*
- *git push*
- *git commit -m "add timeout on connection"*

les tests ont été automatiquement exécuter et un ensemble de modifications ont été apportées par l'outil de linting.

Il a donc fallu enregistrer toute ses modifications.

- *git add opsdroid/conector*
- *git commit -m "lint"*
- *git push* Ensuite j'ai crée la *pull request* pour intégrer mes modifications à la version officiel du projet. Le projet *opsdroid* propose un modèle standard pour les *pull request* que j'ai dû suivre. En particulier, ce modèle demande de préciser l'objectif des modifications, les status de la *pull request* (par exemple

prête ou en cours).Le type de modifications (ici une correction de bug) ainsi que les tests qui ont été effectués. Finalement une *check list* permet de vérifier quelque contrôle finaux, par exemple concernant la modification des mises à jour des tests. Une fois ce modèle rempli, j'ai validé la *pull request*.

3.4 Retour de la communauté

La première réaction d'un des *reviewers* se nommant Fabio Rosado a été très positive mais a proposé des améliorations.

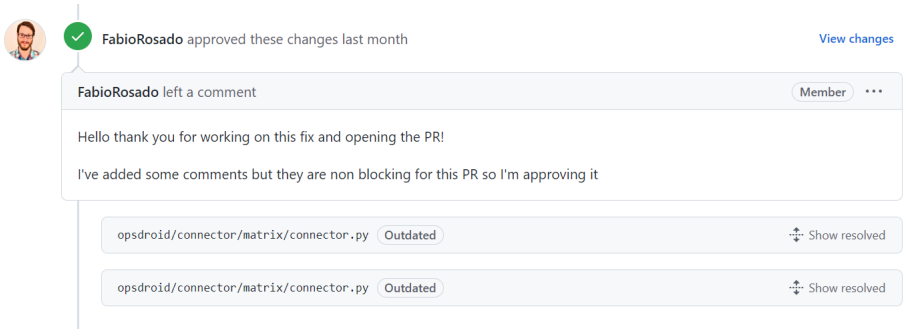

Figure 7: Première réponse de la communauté

En voyant cette réponse positive, je me suis empressé d'implémenter ces améliorations puis j'ai demandé si d'autre modifications étaient nécessaires:

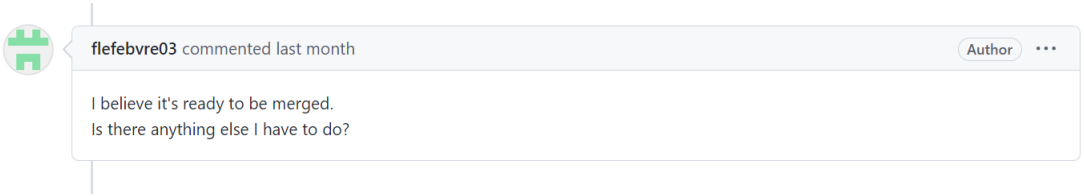

Figure 8: Ma première réponse

Néanmoins, un autre reviewer se nommant Cadair a indiqué un problème qui lui semblait non solutionné :

Ensuite, une discussion avec d'autres contributeurs se lança comme le montre la suite de messages suivants :

Cependant ce *reviewer* semblait mal interpréter mes modifications. En particulier, il semblait croire que mes modifications remplaçaient un système de gestion d'erreur, alors qu'il s'agit en réalité de l'ajout d'une gestion d'erreur supplémentaire. J'ai donc du défendre mes modifications. Finalement, il a reconnu que les modifications étaient valides :

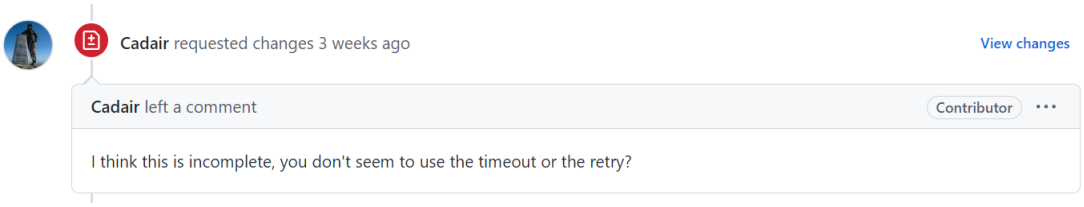

Figure 9: Une autre intervention d'un nouveau reviewer

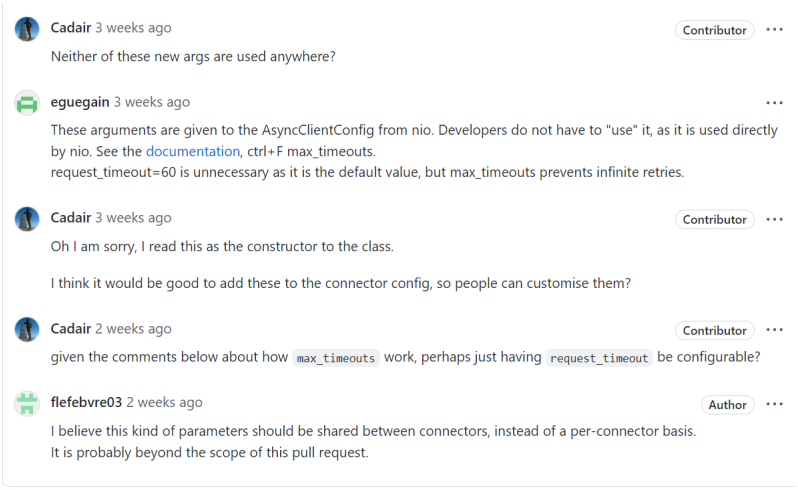

Figure 10: Une discussion sur un aspect technique du projet

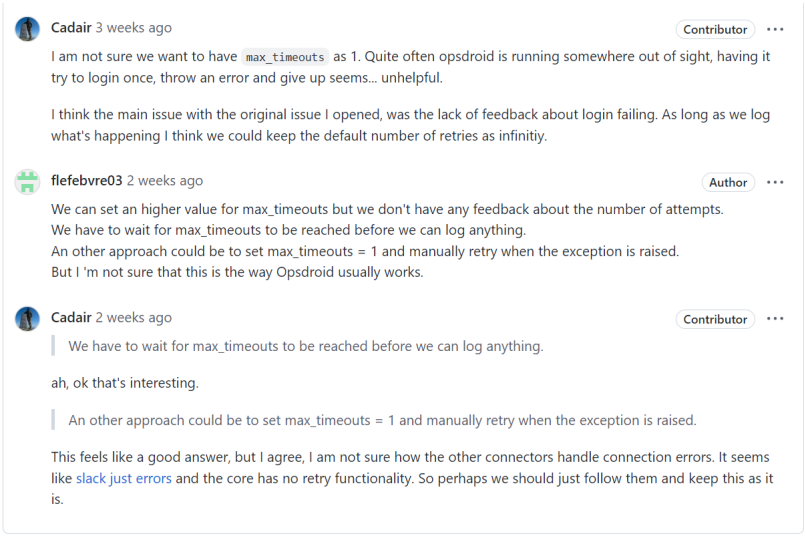

Figure 11: Fin de la discussion

Au moment de la création de ce rapport, la *pull request* n'a pas encore été accepté. Elle est toujours en attente d'une validation d'administrateur du projet, malgré des relances. Le projet semble avoir des pics d'activités, ma pull request a été crée à la fin d'un de ces pics.

4 Conclusions

Ce projet m'a permis d'envisager le développement sous un autre angle : celui d'un travail qui s'insère dans un projet existant avec des développeurs que l'on ne connait pas. Il faut dans ce contexte, pouvoir comprendre comment est organisée cette communauté qui travaille ensemble dans le but de réussir un projet. Cette approche est très différente de celle qu'on peut avoir sur des projets d'école où les contraintes sont en général moins fortes. J'en tire donc un retour d'expérience très positif malgré certaines difficultés techniques à surmonter dans le cadre de ce travail, et le fait que la pull request n'a pas encore été accepté.