Une interface pour Dedukti

Ismaïl Lachheb

Sup Galilée, Université Paris 13

Rapport de stage

Dedukti-editor : une interface pour Dedukti

Ismaël Lachheb
Promo 2019

Encadré par :
Frédéric Blanqui,
Rodolphe Lepigre
et Emilio Jesús
Gallego Arias

3 Mai 2018 — 27 Juillet 2018

Table des matières

Resumé

Pendant ce stage, une interface utilisateur a été développée pour Dedukti, un vérificateur et un assistant de preuve. Elle a été développée en JavaScript comme une extension de l'éditeur de texte Atom en utilisant le *language server protocol (LSP)*. Elle a le double avantage d'être facilement portable sur d'autres éditeurs de texte et pour d'autres assistants de preuve. De plus, de nombreux paramètres sont modifiables sans avoir à modifier le code.

During this internship, an user interface has been developped for Dedukti, a proof-checker and a proof-assistant. It has been developped with Javascript as an Atom extension using the *language server protocol (LSP)*. On top of being easily portable for others text editors, it is also portable for others proof-assistant. Moreover, a lot of settings are configurable without need to touch the codebase.

Introduction

Dedukti est un vérificateur de preuve formelle. De nombreuses preuves peuvent être converties et vérifiées avec Dedukti depuis des cadres logiques étrangers comme ceux utilisés dans d'autres assistants de preuve.

Les concepteurs de Dedukti ont voulu offrir la possibilité aux utilisateurs de leur logiciel d'écrire interactivement des preuves directement depuis Dedukti. Ce choix implique donc de créer une interface pour l'écriture de preuve.

Par suite, l'objet de ce stage a été d'écrire une extension pour un éditeur de texte qui permette d'écrire des preuves avec Dedukti. L'interface doit être interactive, simple d'utilisation, intuitive et facile à maintenir.

1 Contexte

1.1 Le Laboratoire Spécification et Vérification

Le Laboratoire Spécification et Vérification (LSV) a été fondé en 1997, il s'agit du laboratoire d'informatique de l'École Normale Supérieure de Cachan. Il est affilié au Centre National de Recherche Scientifique (CNRS) et à l'INRIA.

Les scientifiques du LSV travaillent notamment sur la vérification de logiciels et systèmes informatiques critiques. En effet, les erreurs, plantages et fuites mémoire qui apparaissent dans la plupart des programmes informatiques sans causer de problèmes sérieux, peuvent dans un système dit "critique" causer de sérieux dommages. On pense notamment au secteur des transports et au secteur de l'énergie nucléaire. Il est donc absolument nécessaire de vérifier mathématiquement la bonne terminaison et le bon fonctionnement des systèmes critiques en utilisant un cadre logique.

C'est en partie l'ambition de Dedukti, un programme développé par l'équipe *Deducteam*, une des équipes qui composent le laboratoire depuis 2008. Au début de mon stage, il était essentiellement un vérificateur de preuves, c'est-à-dire un programme qui prend en entrée une preuve et vérifier si elle est correcte. Pour éviter que les mathématiciens vérifient leur preuves avec un programme suspect, un vérificateur de preuve se doit d'avoir le code le plus simple et court possible.

Le lien entre une preuve et un programme informatique est ce qui cimente le projet Dedukti avec les objectifs du laboratoire. En effet, une preuve et un programme informatique sont logiquement liés. Il est donc possible de montrer la correction d'un programme informatique de la même manière qu'on vérifie une preuve.

1.2 L'écriture de preuve assistée par ordinateur

Dedukti est en mutation pour devenir un programme capable d'assister un utilisateur lors de l'écriture de preuve. Une preuve assistée par ordinateur est une preuve dans laquelle l'utilisateur est averti en temps réel de l'état de la preuve. En plus de cela, pour simplifier l'écriture de la preuve, l'utilisateur peut faire usage de tactique. Les tactiques sont des commandes pour demander au vérificateur de preuve d'user d'une méthode spécifique pour résoudre un problème ou un sous-problème.
Lors de l'écriture de très longue preuve (parfois plus de 1000 lignes), les tactiques peuvent simplifier le travail du mathématicien ou du logicien. Le contexte le plus connu d'utilisation d'assistant de preuve est la démonstration du théorème des quatre couleurs. Ce théorème a été démontré en utilisant l'assistant de preuve le plus connu et le plus utilisé à l'heure actuelle : Coq.
Il existe aussi beaucoup d'autres assistants de preuve qui peuvent être plus intéressant d'utiliser dans certaines situations. Un des objectifs de Dedukti est de simplifier l'interopérabilité entre ses différents assistants.
Finalement, donnons un exemple d'une simple preuve issu de Coq pour illustrer notre propos.

```
Theorem forward_small : (forall A B : Prop, A -> (A->B) -> B).
Proof.
 intros A.
 intros B.
 intros proof_of_A.
 intros A_implies_B.
 pose (proof_of_B := A_implies_B proof_of_A).
 exact proof_of_B.
Qed.
```

FIGURE 1 – Une preuve avec Coq

Dans cet exemple, on démontre l'énoncé étape par étape en utilisant les tactiques intros, pose et exact. Elles réalisent chacune une partie de la démonstration.

1.3 Gestion du Projet

Le projet Dedukti Editor a été géré conjointement avec Rodolphe Lepigre [7], Frédéric Blanqui [8] et Emilio Jesús Gallego Arias [9]. M. Lepigre et M. Blanqui ont travaillé sur les modifications de Dedukti à effectuer pour gérer une interface graphique et M. Arias a travaillé sur l'implémentation du serveur.

Nous avons travaillé en suivant une méthodologie inspirée de la méthode agile, avec des réunions régulières, souvent de manière hebdomadaire pour suivre le projet, proposer des modifications, demander des changements ou de nouvelles fonctionnalités.

Afin d'éviter que des erreurs graves ne puissent être introduites dans la branche master du projet, un outil d'intégration continue appelé Travis CI[10], à été ajouté au projet. À l'heure actuelle, seul un script vérifiant la compilation de l'extension a été fourni à Travis CI.

Naturellement, un gestionnaire de version a été utilisé et l'ensemble du projet a été développé sur Github [1] et est libre d'accès.

2 Choix Technologiques

2.1 État de l'existant

Il existe bien d'autres assistants de preuves outre Dedukti. Ils partagent souvent beaucoup avec Dedukti bien qu'ils n'aient pas la même philosophie ni la même manière de vérifier les preuves. Beaucoup d'entre eux ont une interface et il a été intéressant de s'inspirer des atouts et défauts de ces interfaces pour concevoir celle de Dedukti.

Premièrement, évoquons *Proof General* [2], une interface générique pour les assistants de preuve développée comme une extension pour l'éditeur *Emacs*. Son universalité est un avantage mais ces dernières années, il semble qu'il ait été abandonné au profit d'autres solutions car l'éditeur et l'assistant de preuve ont un fonctionnement synchrone, ce qui dégrade l'expérience utilisateur (ralentissements).

Nous pouvons aussi évoquer *Jedit*, la solution officielle actuellement utilisée par l'assistant à la preuve *Isabelle*[3]. Il apporte une solution asynchrone entre l'éditeur et l'assistant de preuve. Néanmoins, cet éditeur ne me semble pas parmi les plus populaires et il est gourmand en ressources.

Enfin, les solutions qui semblent les plus modernes et les meilleures sont *Isabelle LSP* et *VsCoq*[4]. Ces deux extensions sont développées pour fonctionner sur *VsCode* et peuvent être portées sur de nombreux éditeurs. Elles sont asynchrones et se basent sur le *language server protocol*, un protocole dont nous évoquerons plus loin les nombreux avantages. Néanmoins, elles ajoutent de très nombreuses extensions au protocole, il serait donc intéressant de réussir à développer une extension qui dépende moins d'ajouts au protocole.

2.2 Language Server Protocol

Le *language server protocol* [5] (LSP) a pour objectif de limiter la redondance de travail à effectuer pour lier par une extension un éditeur de texte à un langage informatique (auto-complétion, diagnostic du code, popups, ...). Pour cela, le protocole établit un cadre général de communication entre le client - l'éditeur de texte - et un serveur donné qui fournit les données né-

cessaires aux composants graphiques gérés par l'éditeur de texte. L'intérêt est qu'il suffise que le protocole soit instancié une fois pour un éditeur pour qu'il soit capable de prendre en charge tous les serveurs écrits pour le protocole et inversement, un serveur ne doit être instancié qu'une fois pour qu'il soit disponible sur tous les éditeurs qui instancient LSP. Le *language server protocol* est encore légèrement immature mais il semble devenir la nouvelle norme pour gérer des langages informatiques dans un éditeur.

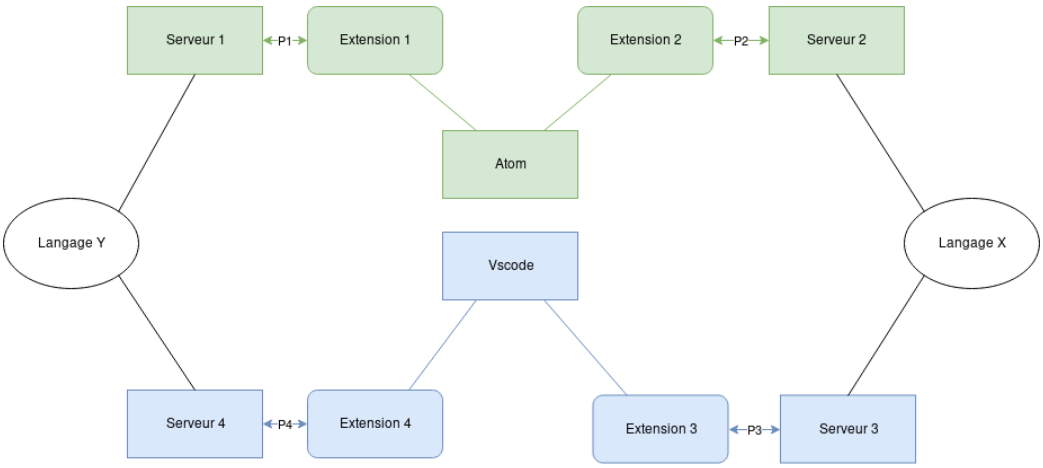

FIGURE 2 – Deux éditeurs, Deux langages, sans LSP

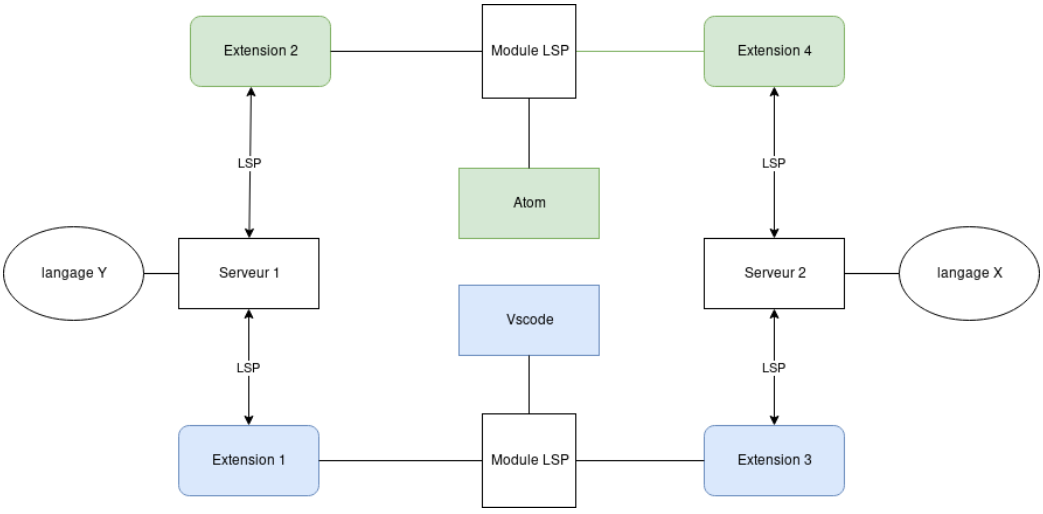

FIGURE 3 – Deux éditeurs, deux langages, avec LSP

On remarque à travers la figure 1 et 2 qu'on a besoin de développer beaucoup moins de serveur avec LSP que sans, et que les modules LSP sur lesquelles sont basés les extensions permettent de diminuer la charge de travail pour développer une extension. Il faut imaginer deux figures équivalentes à celles-ci mais contenant une centaine de langages et une vingtaine d'éditeurs de texte.
Concernant le protocole, il est basé sur un paradigme dans lequel chaque partie se déclare capable de supporter des fonctionnalités (auto-complétion, diagnostic du code, popups...) à l'initialisation ou pendant la vie du serveur.

Contrairement à *VScoq* et *Isabelle LSP* qui ont ajouté de nombreuses extensions à LSP, nous avons limité le nombre d'extensions ajoutées au protocole. L'utilisation d'extensions reste cependant inévitable car il y a des différences non négligeables entre un langage de programmation et un assistant à la preuve.

2.3 Atom

Atom est un éditeur de texte populaire. Il dispose d'une documentation claire, mature et à jour pour son API [6] permettant d'écrire des extensions. C'est un avantage important lorsqu'on sait que l'extension sera reprise par des développeurs peu expérimentés en JavaScript.

En ce qui concerne ses points faibles, les extensions permettant de gérer le *language server protocol* sur *Atom* sont moins matures que sur *Vscode*, un éditeur de texte qui a été écarté. Néanmoins, elles sont développées activement et ce retard tend à disparaître à mesure que le protocole gagne en maturité.

3 Implémentation

3.1 Fonctionnalités développées

Grâce à l'extension *Dedukti Editor*, il est possible de :

1. vérifier la correction d'une preuve.
2. afficher les messages d'erreurs et leur provenance.
3. afficher la liste des buts non résolus selon la position du curseur.
4. lister les hypothèses liées au but courant.
5. gérer des boutons et des raccourcis claviers pour naviguer dans une preuve.
6. gérer une syntaxe.
7. lister les symboles courant du fichier.
8. gérer des caractères Unicode.

L'essentiel des informations permettant le fonctionnement de l'extension passe par le message *publishDiagnostics* du protocole, il est donc important de comprendre sa structure.
Le message *publishDiagnostics* contient un tableau comprenant des diagnostics. Chaque diagnostic est un objet contenant un périmètre sur lequel il s'applique, un message d'erreur (ou un message positif), un URI de fichier, et un objet *goalinfo* qui contient la vue à afficher lorsque le curseur est situé dans le périmètre. Le tableau contient des diagnostics pour chaque périmètre du fichier.

Dans le diagramme de cas d'utilisation, on remarque notamment que la mise à jour des diagnostics implique la mise à jour du panel puisque le panel est mis à jour à chaque fois que des nouveaux diagnostics sont envoyés.
Cela permet de limiter les prises de décision du client *Dedukti Editor*. En effet, à chaque fois que des nouveaux diagnostics sont envoyés, les anciens sont détruits et remplacés par les nouveaux. Ainsi, *Dedukti Editor* n'a pas à gérer un algorithme de tri pour savoir si un diagnostic est encore pertinent ou s'il doit être détruit.

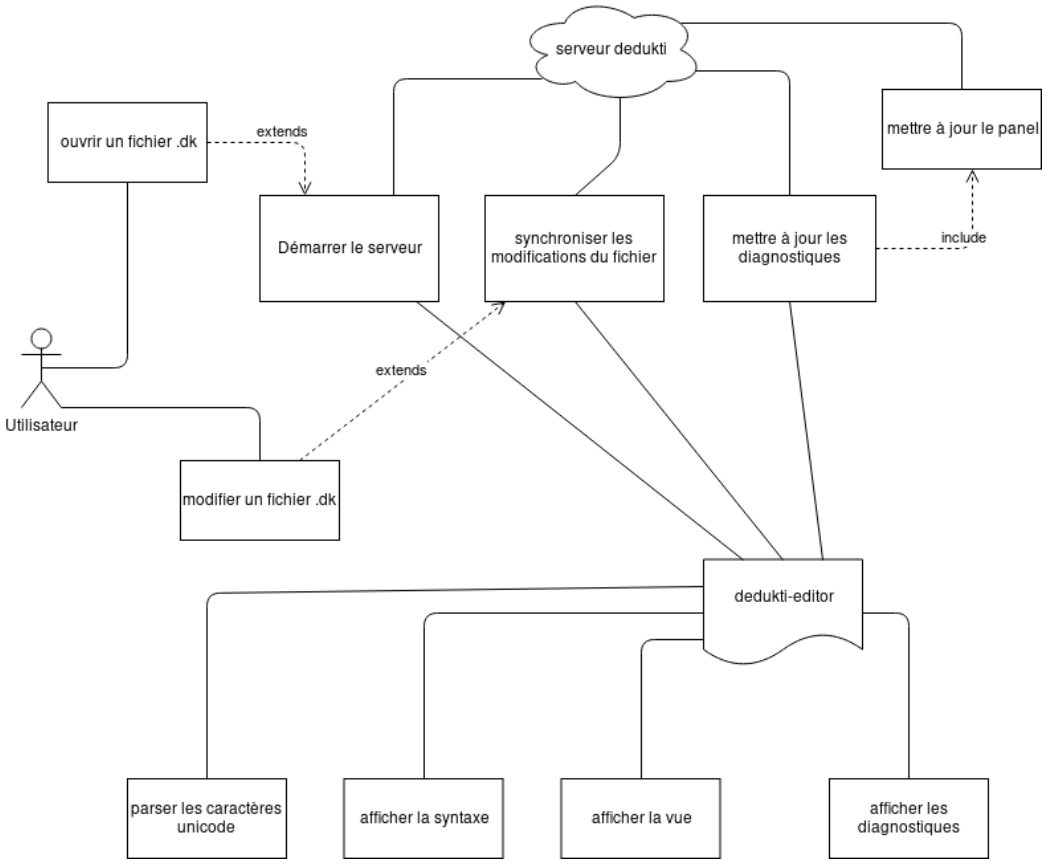

FIGURE 4 – Diagramme de cas d'utilisation

Ce choix simplifie le programme mais entraîne une augmentation non négligeable de la quantité de données transférées entre le serveur et le client à chaque modification du *buffer*. Ceci peut potentiellement avoir un effet négatif sur de gros fichiers. Néanmoins, comme on peut supposer que l'utilisateur n'utilisera pas de fichiers de plus de 10 000 lignes. Cette décision reste toujours globalement imperceptible pour l'utilisateur.

3.2 Description de l'implémentation

3.2.1 Arborescence des fichiers

L'extension Dedukti Editor a été implémentée en module pour être le plus clair possible et en faisant en sorte que les paramètres soient configurables sans qu'il ne soit nécessaire pour autant modifier le code.

Utiliser des fichiers de configuration présente l'important avantage de permettre aux futurs mainteneurs de l'extension d'être en mesure la modifier sans pour autant risquer de briser le fonctionnement de l'extension.

grammar Le mainteneur peut y modifier la syntaxe de Dedukti, plus précisément celle des fichiers .dk.

keymaps Le mainteneur peut y modifier les raccourcis claviers.

styles Le mainteneur peut y modifier le style de l'application.

config :parser Le mainteneur peut y ajouter ou enlever des caractères Unicode automatiquement gérés par Dedukti.

config :settings Le mainteneur peut ajouter ou enlever des paramètres que l'utilisateur pourra modifier dans le menu de l'application.

Dedukti Editor dépend d'*atom-ide-ui*, il s'agit d'une extension *Atom* qui ajoute de nouveaux composants graphiques à l'éditeur, ces composants graphiques peuvent être utilisés par n'importe quelle extension. *Dedukti Editor* dépend aussi d'*atom-languageclient*, un module JavaScript qui permet de faire le lien entre les composants graphiques d'*Atom*, ceux apportés par *atom-ide-ui* et le *language server protocol*. Pour ajouter de nouvelles fonctionnalités à l'extension, il suffit généralement de modifier le fichier package.json.

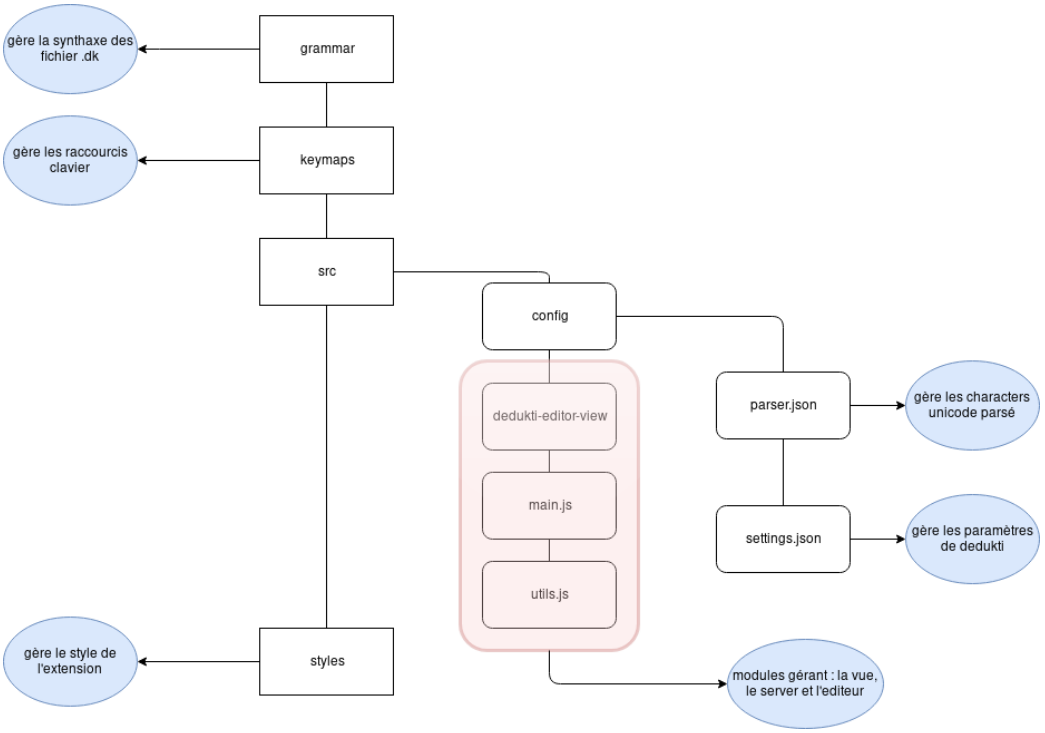

FIGURE 5 – Arborescence des fichiers

3.2.2 Description des classes principales

En ce qui concerne les trois fichiers source de l'extension qui contiennent le code informatique.

dedukti-editor-view Ce fichier contient la classe *DeduktiEditorView*, elle gère le panel et plus particulièrement ses composants graphiques, elle permet donc de l'initialiser, et de le mettre à jour. Le panel est un élément HTML géré grâce à la DOM API et au moteur de rendu du navigateur web Google Chrome. Un seule panel pour tous les fichiers .dk.

utils Il contient la classe *Utils*. Cette classe permet de gérer l'ensemble des fonctions propres à *Dedukti Editor*, on y trouve donc le code informatique permettant de faire disparaître ou apparaître le panel automatiquement, gérer les commandes associées raccourcis clavier, mettre à jour la vue selon la position du curseur, demander la mise à jour du panel et gérer le cryptage Unicode.

main Il contient la classe *DeduktiLanguageClient*. Cette classe permet de gérer les échanges avec le serveur, c'est cet objet qui est créé en premier, donc c'est la classe la plus importante. On a fait en sorte que le message *showDiagnostics* soit traité différemment d'un langage informatique classique car on intègre des données réservées au panel dans un diagnostic.

On voit tout l'intérêt d'utiliser un paradigme asynchrone en regardant le diagramme de séquence d'un des scénarios d'utilisations principaux du système.

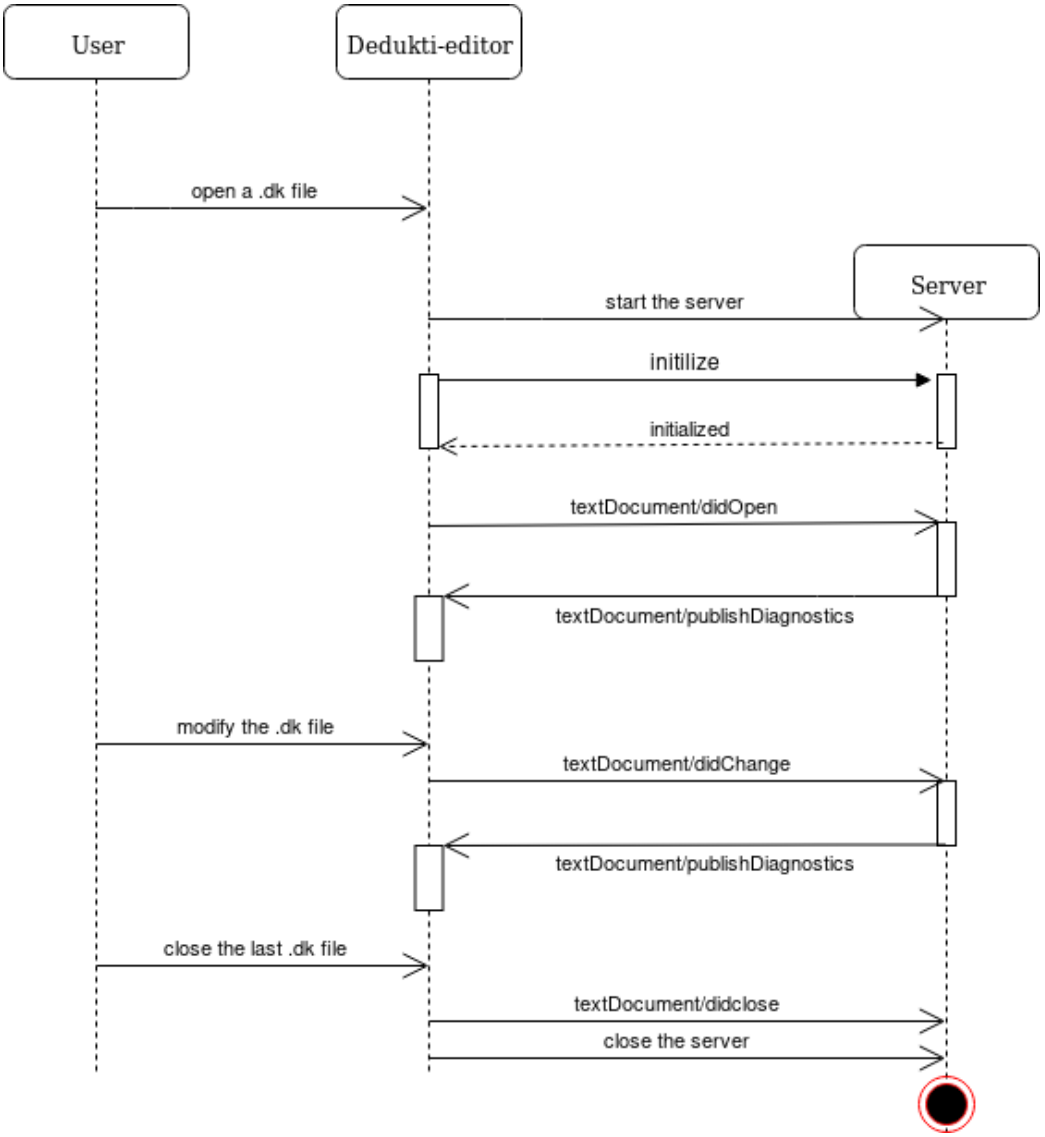

FIGURE 6 – Diagramme de séquence

Conclusion

En conclusion, Dedukti-editor est une extension d'Atom qui permet d'écrire des preuves de manière interactives. Développé dans un cadre international (français et anglais) en utilisant des technologies en libre accès comme Atom ou le language server protocol. Un protocole de communication universel entre éditeurs de texte et langages de programmation.
Il a donc été étendu pour répondre aux besoins propre d'un assistant de preuve. Notamment, en ajoutant des diagnostics positifs et un panel d'information concernant la preuve. L'extension étant elle-même en libre accès. Elle peut être installée directement depuis Atom ou depuis le dépôt Github.
Concernant les potentiels futurs développements pour l'extension, très indépendante de Dedukti, il est tout à fait envisageable de l'adapter pour d'autres assistants de preuve ou pour d'autre éditeurs de texte.

Je remercie toute l'équipe du Laboratoire Spécification et Vérification pour leur confiance et leur soutien et tout spécialement Rodolphe Lepigre, Emilio Jesús Gallego Arias et Fréderic Blanqui.

Références

[1] Dépôt github de Dedukti Editor
`https://github.com/lachhebo/dedukti-editor`

[2] Site officiel de Proof General
`https://proofgeneral.github.io`

[3] Site officiel d'Isabelle
`http://isabelle.in.tum.de/`

[4] Dépot Github de vscoq
`https://github.com/siegebell/vscoq`

[5] Site officiel de LSP
`https://microsoft.github.io/language-server-protocol`

[6] Documentation d'Atom
`https://atom.io/docs`

[7] Rodolphe Lepigre, Post Doctorant
`http://lepigre.fr`

[8] Frédéric Blanqui, chercheur à INRIA
`http://rewriting.gforge.inria.fr`

[9] Emilio Jesús Gallego Arias, chercheur à Mines Paris Tech
`https://www.cri.ensmp.fr/people/gallego`

[10] Site officiel de Travis CI
`https://travis-ci.org`