More Flexible Smart Contract Languages for Permissioned Blockchains and Distributed Ledgers

Louis-Joseph AMAS

16 décembre 2020

Abstract

Ce rapport présente mon projet de fin d'études mené à l'Inria Sophia Antipolis-Méditerranée. Le projet consista à explorer le fonctionnement des Smart Contract. L'objectif a été de rechercher des moyens de rendre les smarts contracts (qui par définition son non immuable) modifiable. Plus en détails, j'ai dû concevoir la mise en place de prototype (PoC: Proof of concept) démontrant la possibilité de modification, de plus, j'ai dû comprendre les problèmes soulevés par cette nouvelle liberté et proposé de solutions. Dans le cadre de ce projet, j'ai pu travailler avec Mr Luigi Liquori, Mr Daniel de Carvalho et Mr Mansur Khazeev dans le cadre de multiples réunions. J'ai pu réaliser trois présentations oral sur ce sujet afin d'aider Mr Khazeev à réaliser sa thèse.

Sommaire

1 Introduction

Tout d'abord, il est nécessaire d'expliquer le concept des technologies concernées. La blockchain est une technologie permettant de faire la représentation d'un registre de compte de manière numérique et distribué. Cela permet de s'affranchir d'un tiers de confiance, la première technologie réalisant ces objectifs est le Bitcoin.

Le Bitcoin est une monnaie décentralisée basé sur la blockchain. C'est un registre de compte contenant toutes les transactions effectuées. Pour sécuriser les accès à ce registre (permettre au ayant droit d'un compte de dépenser son argent), Bitcoin combine deux principes fondamentaux, la cryptographie à clé publique ainsi que la signature numérique, La cryptographie à clé publique permet de vérifier que un utilisateur possède réellement un compte. La signature numérique permet de vérifier l'intégrité des transactions. Les transactions sont des éléments d'un bloc et les blocs sont chaînés (mis les uns après les autres) à l'aide de la signature numérique. Ce principe de chaîne permet d'assurer que si l'une des transactions est modifiée alors la signature est modifiée. Une personne malveillante modifiant une transaction déjà inscrite dans la chaîne de bloc sera détectée.

Des améliorations à ce principe de blockchain on était faite et il existe maintenant une technologie appelée "smart contract". Cette technologie est un moyen d'exécuter un programme de manière distribuée, permettant de s'affranchir d'un tiers de confiance. Cela peut être utilisé pour réaliser par exemple des systèmes de vote sans possibilité de triche ou de modification du résultat ou bien représenter un jeu d'argent (poker, paris sportif...). Cette technologie est enregistrée de la même manière dans un registre non modifiable. L'ojectif de ce projet et de rechercher les manières de pouvoir modifier le programme exécuter par un smart contract afin d'adapter le comportement dynamiquement selon les besoins.

2 Contexte technologique

2.1 Blockchain

2.1.1 Bitcoin

Une (ou un) blockchain, ou chaîne de blocs. Techniquement, il s'agit d'une base de données distribuée dont les informations stockées sont vérifiées et groupées à intervalles de temps réguliers sous forme de bloc, formant ainsi une chaîne de blocs. L'ensemble est sécurisé par cryptographie. La chaîne de blocs est alors stocké sur de multiple machine en même temps que l'on appelle des nœuds. Cette chaîne permet de protéger contre la falsification et la modification de la base de données par les nœuds de stockage. C'est donc un registre distribué et sécurisé de toutes les transactions effectuées depuis le démarrage du système réparti.

La première technologie déployant un tel système est le Bitcoin. Le Bitcoin utilise la blockchain afin de représenter un registre de compte distribué. Les utilisateurs souhaitant utiliser cette cryptomonnaie génère une paire de clé asymétrique.

Pour rappel, les chiffrements asymétriques sont constitués de deux clés l'une publique disponible par tous le monde et l'une privée qui doit rester secrète. Les méthodes de chiffrements à clé publique permettent e de rendre un message inintelligible par toute personne n'ayant pas la clé privée d'un message chiffré par la clé publique complémentaire. Cette technique de chiffrement permet aussi de vérifier (Signer numériquement) qu'un message a bien été écrit par quelqu'un possédant une paire de clé (Privée, publique).

Bitcoin utilise cette fonctionnalité afin d'autoriser une personne à échanger son argent. Un bloc de la blockchain Bitcoin contient des transactions (des échanges de jeton BTC) signer par la clé privée de l'utilisateur souhaitant transférer ses jetons. Bitcoin calcul le hash de toutes ses transactions et crée un bloc à l'aide de la concaténation de tous les hashs de toutes les transactions ainsi que le hash du bloc précédent.

Cela permet de rendre quasi impossible la falsification et la modification de la chaîne. En effet, un attaquant voulant modifier la chaîne serait détecté car, le hash de son bloc serait invalide s'il modifie même qu'une transaction. Le Bitcoin réalise l'exploit de créer un équivalent a l'argent liquide mais numériquement.

2.1.2 Ethereum

Ethereum est une technologie qui révolutionne la blockchain en y ajoutant une fonctionnalité très intéressante nommé les smarts contract. Cette technologie est toujours basée sur le même principe que Bitcoin afin de valider ses transactions mais ajoute quelques nouvelles fonctionnalités.

2.2 Smart contract

Les contrats intelligents ou smart contracts sont des protocoles permettant d'exécuter du code de manière distribuée dans un environnement blockchain. Ce mécanisme permet de s'affranchir de l'architecture client / serveur mais aussi, des architectures distribuées traditionnels. Les développeurs de tel smart contract peuvent assumer que leurs contract une fois déployé sur la blockchain est immuable et décentralisé.

L'immuabilité de tel contrat permet de gagner la confiance des utilisateurs. Un utilisateur peut regarder le code du contrat déployé est être sûr de son comportement. Cela permet par exemple de programmer des jeux d'argents de manière numérique sans tierce de confiance mathématiquement vérifiable juste.

2.2.1 Smart contract modifiable

Un contrat auto modifiable est un contrat qui à la capacité suite à la réception ou à la comparaison de données de se modifier seul. En effet, un contrat peut avoir besoin de modifier son comportement quand certaines conditions sont respectées.

2.2.2 EVM

Pour créer une tel technologie Ethereum utilise l'Ethereum Virtual Machine (EVM) cette machine virtuelle peut être comparé à la JVM mais de manière distribuée. L'EVM est une machine basée sur une *pile distribuée sur réseaux*. L'EVM est exécutée sur chaque nœud du réseau Ethereum. Et chaque exécution est signé de la même manière que Bitcoin afin de protéger contre la falsification ou la modification d'une exécution.

2.2.3 Solidity

Solidity est un langage de programmation orienté objet dédié à l'écriture de contrats intelligents. Il est utilisé pour implémenter des smarts contrat sur diverses blockchains, notamment Ethereum. Solidity est un langage inspiré des langages orientées objets et plus en particulier du langage Javascript. L'objectif de ce stage est de rendre simple et compréhensible le développement de smart contracts à tous. Néanmoins sa simplicité peut causer des problèmes car, il n'y a pas de moyen simple de vérifier (à la différence d'un langage fonctionnel) l'exactitude du programme. Sachant que les smarts contracts représentent souvent de l'argent il est très important de faire attention au code déployé.

3 Mes Objectifs et les scénarios

3.1 Objectifs

Les objectifs de ce projet ont été les suivants et inspirés au papier scientifique [10] qui évoquait un langages de programmation object-based où les méthodes se pouvaient modifier (de façon sûre) par un simple appel de méthode (*self-inflicted extension*):

- Créer un prototype de modèle de smart contract modifiable.
- Comparaison de ma solution avec une bibliothèque populaire.
- Comparaison avec l'état de l'art standard.

Dès débuts du mois d'octobre, j'ai commencé à me renseigner sur les technologies présentées au chapitre précédent. Suite à quelque réunion avec Mr. Luigi Liquori, Directeur de Recherche Inria, nous avions trouvé un chemin vers lequel je pouvais m'orienter afin de réaliser mon premier *smart contract modifiable*.

Nous avons alors pris la décision de réaliser des prototypes intégralement par nous même. L'objectif est ici d'apprendre les bases du développement et déploiement d'un tel contrat, sans profiter d'aucune abstraction.

Une fois ce prototype terminé nous avions prévu de comparer cette solution a l'implémentation de la bibliothèque *OpenZeppelin* [4]. Cette bibliothèque étant l'un des standard de l'environnement des smarts contract Ethereum.

Durant le mois de projet, j'ai aussi contacté l'équipe de développeurs d'Ethereum afin d'obtenir des ressources sur le sujet. Cette équipe m'a répondu est m'a envoyé les dernières techniques du domaine. Nous avons alors décidé d'ajouter comme objectif la comparaison avec des techniques les plus avancés avec mon prototype et l'implémentation d'OpenZeppelin.

3.2 Scénarios et *Use Cases*

3.2.1 Scénario 1

Un état souhaite passer au vote électronique, il décide d'utiliser les smart contrats afin de décentraliser leur solution. L'objectif est de prouver aux électeurs que le vote n'est pas truqué. L'état va alors payer une équipe d'ingénieur afin de faire un système de vote qui pourra durer dans le temps. Néanmoins ils souhaitent aussi pouvoir modifier le mécanisme de vote au fur des années. C'est ici que mon projet de recherche prend tout son sens, je conseille cet état fictif d'utiliser mes recherches afin d'utiliser un smart contract modifiable. L'objectif est pouvoir changer le système de vote dans le futur tout en utilisant la technologie blockchain.

3.2.2 Scénario 2

Deux entreprises (A et B) souhaitent passer un accord, elles décident de partager leurs services de ressources humaines pour réduire les frais. Elles décident de stocker le nombre d'heures passés à travailler pour une entreprise de façon décentralisée (Smart Contract). Avec cette solution l'entreprise B pourra être sûre que l'entreprise A ne ment pas et vice-versa. De plus A aimerait ajouter une nouvelle entreprise dans quelques années au système. Pour cela, le smart contract doit pouvoir être modifié pour ne pas forcer A et B de déployer un nouveau contrat contenant C (la nouvelle entreprise).

4 Prototypes

Tous le code présenté dans ce rapport est accessible ici: `https://github.com/Louis-Amas/Projet_TER`

- Phase de recherche.
- Comparaison avec une bibliothèque populaire.
- Comparaison avec le standard.

4.1 Phase de recherche

Pendant la phase de préparation du projet, je me suis renseigné sur toutes les manières de rendre un smart contract modifiable *a posteriori*. Mon temps de projet étant assez court, je me suis concentré sur une méthode nommée *Proxy* [6]. Cette méthode permet de créer un smart contract en façade qui déléguera les appels à un autre smart contract dit d'implémentation. J'ai aussi contacté l'équipe de développement d'Ethereum afin d'obtenir plus de ressources. L'équipe de développement m'a répondu avec un document contenant un récapitulatif de l'état de l'art sur la modification des smarts contracts. Une aubaine pour moi.

Ce document contenait de nombreux lien vers des répertoires Git, après quelques heures à lire le code source de ces répertoires, et avec mes recherches en amont j'avais toutes les clefs nécessaires pour commencer.

4.2 Les outils

Il me fallait un environnement de développement afin de commencer à programmer un prototype. L'Ethereum étant une technologie décentralisée contenant plusieurs nœuds il est nécessaire d'utiliser des outils de simulation ou des réseaux de tests (Testnet). J'ai opté pour la simulation de la blockchain Ethereum. J'ai utilisé la suite d'outil Truffle [7]. Truffle contient un logiciel permettant d'émuler une blockchain Ethereum mais aussi, des bibliothèques pour gérer tout le cycle de vie des smarts contracts, développement, compilation, déploiement et test.

Truffle se présente sous la forme d'une interface de commande et d'une architecture de fichier spécifique. Truffle m'a été très utile pour tester mes smarts contracts. En effet, il est possible d'écrire des tests automatiques en Javascript afin de vérifier le bon fonctionnement des smarts contract.

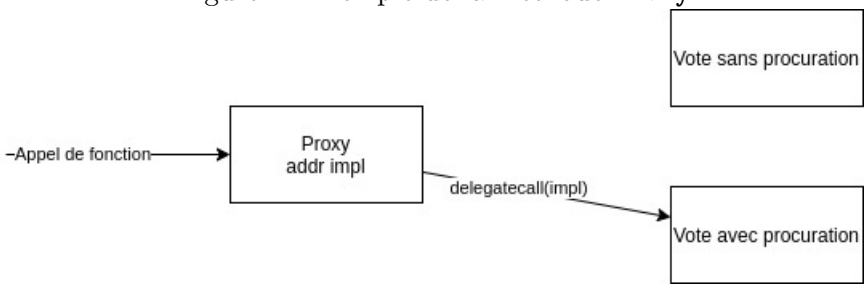
Truffle peut aussi être connecté à une vraie blockchain et donc être utilisé en production. De plus, quelqu'un voulant comprendre ou réutiliser mon code, pourra exécuter mon code une fois qu'il aura configuré la blockchain sur laquelle il souhaitera le déployér. Cela permet de simplifier l'environnement de développement. Durant ma phase de recherche j'ai lu les tests Truffle des implémentations trouvés sur internet afin de mieux comprendre leurs fonctionnements.

L'outil Ganache [8] est aussi très utile pour débuger les smarts contracts car, il est possible de lire chaque transaction et chaque bloc de la blockchain simulé.

4.3 Premier prototype

Une fois tout en place j'ai pu commencer mon développement. J'ai décidé de commencer par l'implémentation d'un système de vote électronique comme décrit dans le scénario 1. J'ai programmé un système de vote simple permettant au créateur du vote de donner

Figure 1: Exemple de la méthode `Proxy`



le droit de voter a qu'il souhaite. Chaque utilisateur ayant le droit de vote peut alors voter pour un des candidats proposer par le créateur du contrat. Je veux pouvoir avoir la possibilité de donner mon vote à quelqu'un d'autre, c'est-à-dire faire une procuration. Je souhaiterais pouvoir ajouter ou enlever dynamiquement cette fonctionnalité.

4.3.1 Méthode utilisée

Mon premier prototype réalise donc cette fonctionnalité. Pour réaliser cette modification alors que je rappelle qu'elle est normalement impossible car, les contrats sont immuable. J'ai utilisé une méthode appelée `Proxy`. Pour cela, j'ai créé un contrat avec quelque particularité. J'ai utilisé la fonction "`fallback()`" de Solidity. Cette fonction est appelée quand la fonction appelée sur un contrat n'existe pas. Cette fonctionnalité m'est très utile, car je vais pouvoir à l'aide de l'opcode `delegatecall` transmettre touts les appels fait au contrat `Proxy` au contrat contenant l'implémentation.

4.3.2 Détail technique

L'opcode `delegatecall` permet d'exécuter du code se situant dans un autre contrat voir Figure 1. Il prend en paramètre une adresse d'un autre contrat.

Néanmoins, il est important de bien comprendre comment Solidity stocke les variables dans un contract, afin de comprendre comment exécuter le code dans le contrat appelé par "`delegatecall`", peut-il interagir avec les variables.

En effet, l'opcode "`delegatecall`" à un comportement peu intuitif, `delegatecall` exécute le code d'un autre contrat mais garde le stockage du contrat appelant "`delegatecall`". Cela veut dire que le contrat implémentation doit savoir comment les variables du contrat appelant sont stockées. Solidity stocke les variables les une après les autres

$$\forall n \in N, var_1 \mapsto pos_1, var_2 \mapsto pos_2, ..., var_n \mapsto pos_n$$

Figure 2: Exemple de la méthode `Proxy` avec stockage structuré

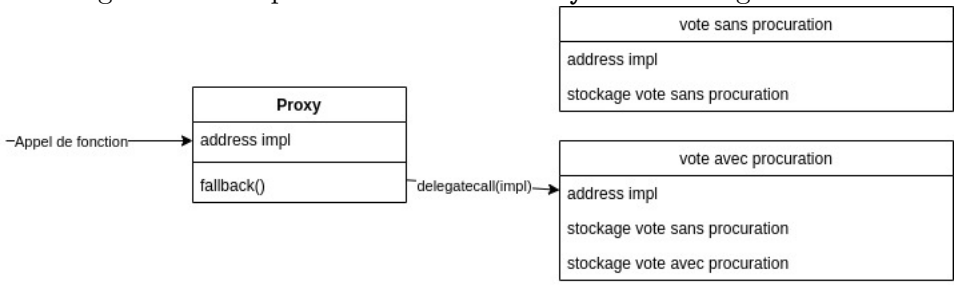
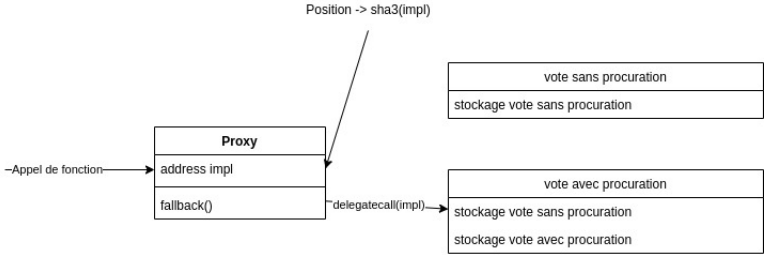


Figure 3: Exemple de la méthode `Proxy` avec stockage non structuré



Pour la réalisation d'un contrat `Proxy` il est nécessaire d'avoir une variable contenant l'adresse d'une implémentation, il est très important que le contrat contenant l'implémentation n'override pas son adresse.

Il existent deux méthodes pour s'assurer que cela n'arrive pas. La méthode la plus simple appelé *stockage structuré*, Figure 2 et de mettre la variable contenant l'implémentation dans chaque contrat.

La deuxième méthode appelé *stockage non structuré*, Figure 3, est de stocker l'adresse de l'implémentation à une position "aléatoire". Pour cela, on peut placer la variable à la position retournée par le calcul de la fonction sha3 sur le nom de la variable. La fonction sha3 étant une fonction de hash il y a très peu de risque de collision.

La méthode avec un stockage non structuré est une meilleure solution car, elle permet d'utiliser des contrats d'implémentation qui ne connaisse pas l'existence du `Proxy`.

4.4 Implémentation officielle (Diamond)

En poursuivant mes recherches sur l'état de l'art du domaine j'ai trouvé sur le site internet des "Ethereum Improvement Proposals"(EIP) [9]. La proposition 2535 du 22 février 2020 ayant pour but de formaliser la manière dont programmer un `Proxy` "intelligent". Le principe sous-jacent est identique à mon prototype avec un stockage non structuré mais, cette proposition y ajoute une interface plus simple et plus modulable.

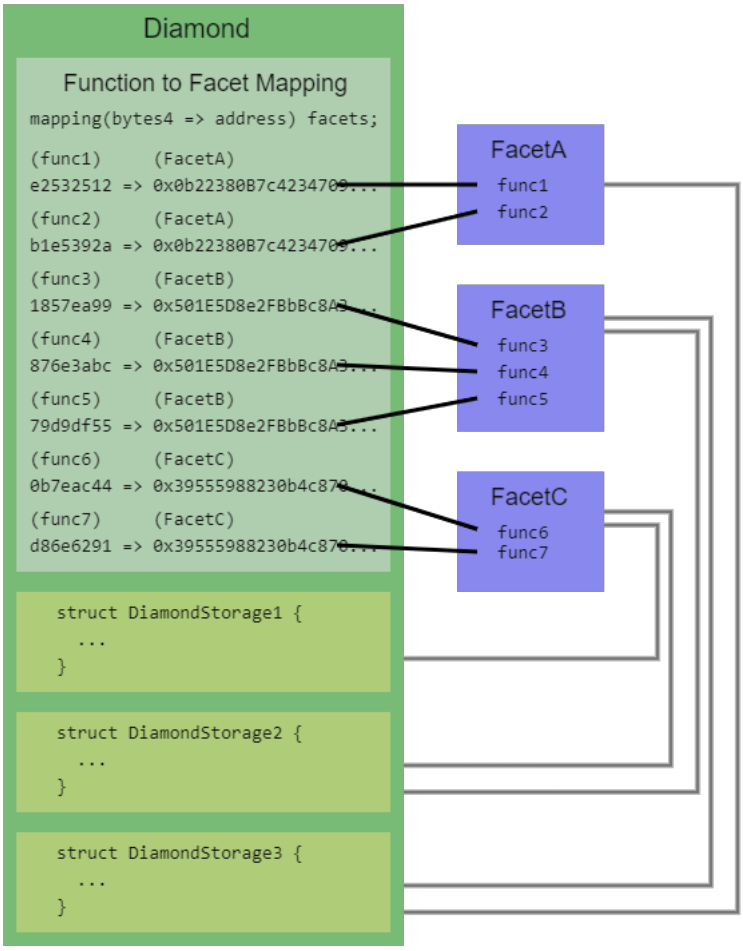
En effet, l'objectif est de créer une bibliothèque avec une interface simple afin de permettre facilement à un utilisateur de programmer son propre proxy personnalisé a forme de "*diamant*": ce diamant fonctionne de la manière suivante.

Un diamant est composé de facette chaque facette représente une fonction. Chaque facette permet de lier l'identificateur de fonction à l'adresse d'un autre contrat. Quand un appel de fonction est exécuté sur un diamant alors celui-ci ira chercher dans sa structure de données la facette correspondante à l'appel de fonction. Une fois cette facette trouvée il pourra alors appeler `delegatecall` sur l'adresse associé à cette facette afin d'exécuter le coder situé dans un autre contract.

Afin d'ajouter, modifier ou supprimer une facette la bibliothèque proposée contient une fonction nommée "diamondCut". Cette fonction prend en paramètre un tableau de "coupe de diamant". Une coupe de diamant est une structure contenant une action (Ajouter, modifier ou supprimer), une adresse d'un contract et un tableau de sélecteur de fonction. Voir la Figure 4

Un développeur voulant utiliser cette bibliothèque aura juste à générer un tableau de "coupe de diamant" modélisant le proxy qu'il souhaite réaliser. La seule obligation du développeur est d'utiliser uniquement un stockage non structuré dans ces facettes.

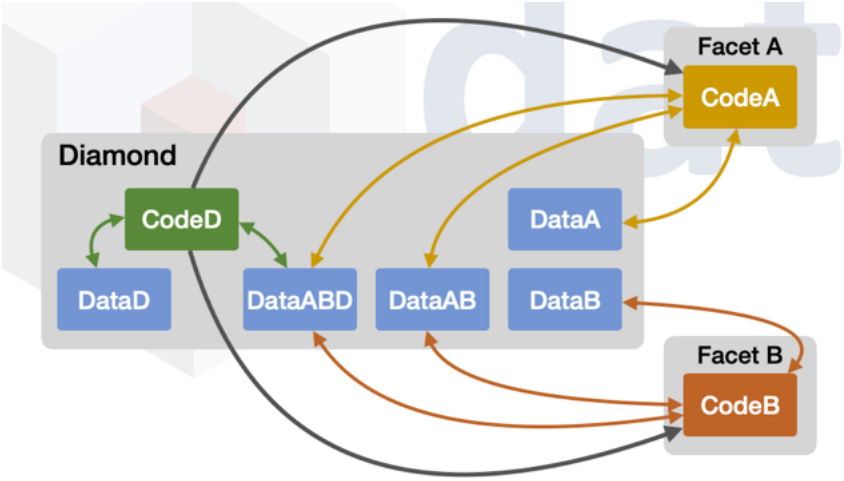
Figure 4: Schéma représentatif du fonctionnement internet d'un diamant.



4.5 Comparaison de mon prototype et du diamant

Il est évident que l'implémentation officielle est bien plus avancée que mon prototype, néanmoins nous pouvons voir que mon prototype utilise la même technique que l'implémentation officielle. La plus grande différence entre le diamant et mon prototype est la modularité. En effet, mon prototype étant un prototype je ne me suis pas concentré sur la simplicité de l'interface d'utilisation. Mon objectif était de réaliser un "Proof of Concept". L'objectif est réalisé, j'ai un prototype fonctionnel et cela m'a permis de bien mieux comprendre le fonctionnement de l'implémentation officielle. J'ai pu acquérir les connaissances bas niveaux nécessaires à de telle utilisation.

Figure 5: Schéma représentatif des appels à un diamant.



5 Problèmes soulevés

Pouvoir modifier dynamiquement le comportement d'un smart contract comporte des risques. En effet, comme expliqué précédemment, les utilisateurs ont confiance dans les smarts contracts car, leurs comportements sont immuables. Il est donc primordial de trouver des solutions afin de garder la confiance des utilisateurs. Pour cela il existe différentes type de méthode appelé *gouvernance*.

5.1 Gouvernance avec un administrateur

La première technique de gouvernance qui est aussi la plus simple est avec un *administrateur*. En effet, à la création d'un smart contract on peut définir un administrateur qui à tout contrôle sur le comportement du smart contract. Il peut ajouter, modifier ou supprimer du comportement. Dans ce cas utiliser le smart contract revient à utiliser une alternative centralisée. Malheureusement, avec un administrateur on perd l'avantage du décentralisé de la blockchain.

5.2 Gouvernance avec un système de vote

Un système de gouvernance avec un *système de vote* permet de garder l'aspect décentralisé d'un smart contract. Dans ce type de système il est nécessaire définir un protocole de proposition et d'acceptation d'un changement. Pour définir la valeur d'un vote on peut utiliser un smart contract implémentant un `Token`. Un `Token` est défini dans l'environnement Ethereum par la norme ERC-20. Cette norme définie qu'un `Token` *fongible* (un "bien fongible" est un bien qui se caractérise par son appartenance à un genre et non par une identité propre). Un `Token` peut donc s'apparenter à une monnaie. Un `Token` peut s'échanger ou être créé.

À la création d'un smart contract modifiable gouverné par un système de vote, on peut imaginer générer une quantité fixe de `Token` est les attribuer à un nombre restreint de parti prenante. Afin d'accepter une proposition on demande aux personnes ayant des `Tokens` de voter. Le vote consiste à pour toutes les personnes ayant des `Tokens` de donner leurs accord ou non accord (oui ou non) par rapport à une proposition. S'ils donnent leurs accords alors leur quantité de `Token` est ajouté au nombre de vote pour la proposition. Si le nombre de vote pour une proposition est supérieur ou égal à la quantité totale * 0.8. Alors 80% des personnes ayant le droit de vote sont d'accord pour

la modification. Le code permettant d'ajouter une proposition et de voter doit être immuable afin de protéger contre un vote souhaitant modifier le système de vote.

Par exemple: deux entreprises créent un smart contract modifiable (basé sur la bibliothèque `Diamond`). À la création elle génère 1000 `Tokens` (500 pour les deux entreprises). Afin d'ajouter, modifier ou supprimer du comportement au smart contract il est nécessaire d'obtenir l'accord de 80% des `Tokens` crées. L'entreprise 1 demande d'ajouter une fonctionnalité nommée "A". La proposition A reçoit directement le nombre de `Token` de l'entreprise 1 soit 500. Si l'entreprise 2 vote "oui" alors le total des votes sera de 1000 et vu que $1000 > 1000 * 0.8$ alors la proposition est accepté. L'avantage de cette méthode est que l'on peut très facilement ajouter des acteurs. Par exemple: L'entreprise 2 à un accord avec une nouvelle entreprise nommée "3". Cet accord consiste à obtenir 25% des droits de vote sur le smart contract modifiable. Afin de réaliser cette action l'entreprise 2 envoi 250 `Token` à l'entreprise 3. Ce système permet une grande flexibilité est permet d'instaurer un vrai système démocratique au sein de la blockchain. Cela permet de récupérer la confiance des utilisateurs car, toute modification doit être approuvé.

5.3 Gouvernance avec un Multisig wallet

Un *multisig wallet* est un procédé cryptographique qui permet d'effectuer une action (chiffrer, déchiffrer ou signer) à l'aide de plusieurs clés. Cela permet de vérifier cryptographiquement l'accord entre plusieurs parties. En effet, pour executer une action il faut l'accord de chaque partie. Un smart contract avec un système de gouvernance basé sur un "multisig wallet" aura donc besoin de 2 à N clés publiques lors de sa création. Si, un utilisateur souhaite ajouter, modifier ou supprimer le comportement du smart contract, il devra alors obtenir les signatures de chaque clé privée correspondante aux clés publiques données au préalable.

6 Pedagogie

6.1 Récapitulatif des réunions

Durant la période à plein de temps de mon TER, j'ai eu l'occasion de partager mon travail avec deux chercheurs, Dr Luigi Liquori (INRIA), Dr Daniel De Carvalho (U. Innopolis) et un doctorant Mr Mansur Khazeev (UCA & U. Innopolis). Leurs domaines de recherche sont les langages de programmation, et ils s'intéressent aux technologies blockchain et aux langages de programmations associés au développement des smarts contracts. Un des objectifs de mon projet a été d'aider le doctorant à réduire le champ de recherche de sa thèse. Durant toutes les semaines du mois à plein temps nous avions une réunion par semaine. Étrangement je me suis retrouvé au cœur de ces réunions. On m'a donné la parole durant minimum une heure par semaine et j'avais carte blanche ou presque sur les sujets abordés. Au début, cela a été plutôt difficile je n'avais pas l'habitude de parler si longtemps avec des personnes plus sages que moi. De plus, les réunions étaient en anglais et ce n'est pas ma langue maternelle.

Finalement, j'ai énormément appris et je pense que j'ai pu aussi leur apprendre des choses. J'ai réalisé de la programmation en direct de smart contract modifiable afin de montrer directement le côté technique. J'ai aussi réalisé des slides afin d'expliquer mes propos. Transmettre mes recherches et les connaissances que j'ai pu acquérir est je trouve très intéressant, j'ai pu mettre à l'épreuve ce que je savais ou ce que je pensais savoir aux avis de chercheurs expérimentés et cela permet de se rendre compte de savoir si ma pensée était correcte.

6.2 Mon organisation

Ce projet a été réalisé durant la période de confinement suite aux mesures sanitaires contre le COVID-19. J'ai réalisé tout le projet de mon domicile familial. Cela faisait plus de 4 ans que je n'y avais pas habité et vivre de nouveau et surtout travailler entouré de ma famille a été un réel défi. Afin de ne pas perdre un rythme de travail je me suis imposé de commencer tous les jours le travail à 9h. En général, j'avais trois tâches principales à faire:

- **Rechercher**. J'ai réalisé des centaines de recherches sur internet, j'ai aussi lu beaucoup d'information sur Twitter et Reddit qui malgré leur statut de réseau social sont selon moi de bonne source d'information pour les sujets scientifiques modernes.
- **Programmer mon prototype**. J'ai réalisé beaucoup de prototype afin de comprendre en profondeur le fonctionnement des techniques que j'ai pu trouver lors de mes recherches.
- **Préparer des slides d'explication**. L'objectif est de présenter les recherches aux chercheurs et de simplifier les explications des prototypes.

J'ai dédié énormément de temps à ce projet est je suis fier de ce que j'ai pu réaliser. Ce projet à éveiller mon esprit et la recherche et m'amènera peut-être vers une thèse de Ph.D [11].

7 Ouvertures

7.1 Améliorer la manière de créer des contracts dynamiquement modifiable dans Solidity

Suite à toutes mes recherches et suite aux discussions avec le Dr Luigi Liquori, nous arrivons à la conclusion que *créer des contracts dynamiquement modifiable est possible mais requiert une base technique importante.* De plus, il est très facile de faire des erreurs car, l'implémentation de tel fonctionnalité requiert des appels bas niveau à l'EVM. La bibliothèque `Diamond` est une solution par l'usage d'une bibliothèque qui en capsule certain appel bas niveau. Néanmoins, un utilisateur de cette bibliothèque doit tout de même maîtriser le fonctionnement de l'allocation et du stockage de données de l'EVM. Et ceci peut-être très difficile est donc peu créer des problèmes aux développeurs de smart contract. Et comme dis précédemment, une erreur dans un smart contract peut avoir des répercussions catastrophiques sur les utilisateurs. Un aout d'un système de typage *ad hoc* dans Solidity pourrait tempter de sécuriser cette nouvelle fonctionnalité.

Une manière d'améliorer et de simplifier l'écriture de smart contract modifiable, il serait intéressant d'ajouter des fonctionnalités de type au langage (Solidity). En effet, il serait possible d'ajouter un mot clé du genre `upgradable` définissant la possibilité à un smart contract d'être modifiable à l'avenir. Une fois un smart contract de type *upgradable* utilisé, le compilateur pourrait générer tous les appels bas niveaux et cacher à l'utilisateur leurs fonctionnements. Un nouveau comportement pourrait être défini par un autre type ("extension"). Ce nouveau type permettrait à l'EVM de comprendre que ce smart contract est uniquement utilisé pour être chargé dans un smart contract *upgradable*.

Dans un plus long terme, il est selon moi important de ne pas se contenter à Solidity. Solidity est le premier langage pour écrire des smarts contract néanmoins de plus en plus de nouveau langage sont écrits. Solidity se compile en EVM bytecode ou en WASM (WebAssembly pour EVM). Des nouvelles blockchains se développent tous les jours et ses nouvelles blockchains ont des nouveaux langages et des nouvelles fonctionnalités. Il est important, selon moi, de définir comment réaliser un smart contract modifiable en WASM et d'y ajouter une bibliothèque près compilé (selon la cible EVM ou autre) utilisable dans chaque langage et chaque blockchain.

Ceci permettra de grandement simplifier l'adoption de contrats dynamiquement modifiable. Et ces contrats sont très prometteurs à l'avenir pour de nombreuses applications (Assurance, finance décentralisée...).

7.2 Ajout d'un système de type plus intelligent pour prévenir les erreurs de contrat modifiables dynamiquement mais pas n' emporte comment

Programmer un smart contract modifiable est possible mais est sujet aux erreurs. En effet, sachant que tout est fait en appelant des opcodes bas niveaux et en gérant la mémoire soi-même, il est très facile de faire une erreur grave. Suite aux discussions réalisées nous en arrivons à la conclusion qu'il peut être important de réaliser une vérification de type au sein des mécanismes proposés ci-dessus. En effet, cela permettrait de vérifier que l'utilisateur fait une erreur et donc de potentiellement de la prévenir. Une fois de plus des erreurs dans un smart contract déployé peuvent avoir des effets catastrophiques, il est donc très important d'amener le maximum d'outil afin de les prévenir.

8 Conclusion

Ce projet m'a permis de découvrir un domaine qui m'intéresse vraiment, la blockchain. La blockchain est selon moi le futur pour énormément d'applications centralisées. Et pouvoir travailler un mois à plein temps dessus m'a permis d'acquérir des compétences dans ce domaine prometteur. De plus, avoir été encadré par des chercheurs de l'INRIA m'a permis de découvrir est m'intéresser au monde de la recherche que je trouve passionnant. J'ai pu avec leurs aides acquérir et comprendre l'état de l'art dans le domaine des smarts contract modifiable et j'en suis à la fois fier et excité. Nous avons pu définir des idées d'améliorations qui permettraient de simplifier la vitesse d'adoption de cette technologie en la rendant plus sûre et moins difficile d'utilisations à tous les développeurs. J'ai envie de continuer ma vie dans ce domaine qui est la blockchain et je sais que ce TER m'a aidé à renforcer mes connaissances.

Ce fût une expérience humaine et qui malgré le contexte sanitaire difficile m'a permis de me rapprocher de mon professeur Dr Luigi Liquori qui m'a fait confiance et qui m'a mis en avant devant ses pairs chercheurs et doctorants. J'espère après mon stage poursuivre mon travail dans ce domaine et j'espère pouvoir apporter quelque chose dans quelques années à la communauté internationale. Après ce projet, je continuerai peut-être en thèse ou en entrepreneuriat.

Je remercie Dr Luigi Liquori pour m'avoir fait confiance, je remercie aussi Dr Daniel Da Carvalho et Mr Mansur Khazeev de m'avoir écouté et corriger mes travaux et je remercie toutes l'équipe pédagogique d'avoir pu me transformer en le futur diplômé que je serais très bientôt.

References

[1] Satoshi Nakamoto: Bitcoin white paper,
`https://bitcoin.org/bitcoin.pdf`

[2] Vitalik Buterin: Ethereum white paper,
`https://ethereum.org/en/whitepaper/`

[3] L.M Goodman. Tezos white paper,
`https://tezos.com/`

[4] OpenZeppelin,
`https://docs.openzeppelin.com/openzeppelin/`

[5] The state of smarts contract Upgrades,
`https://blog.openzeppelin.com/the-state-of-smart-contract-upgrades/`

[6] Proxy: How to Write Upgradable Smart Contracts,
`https://simpleaswater.com/upgradable-smart-contracts/`

[7] Truffle suite: Sweet Tools for Smart Contracts,
`https://www.trufflesuite.com/`

[8] Ganache: One Click Blockchain,
`https://www.trufflesuite.com/ganache/`

[9] Ethereum Improvement Proposals,
`https://eips.ethereum.org/`

[10] P. Di Gianantonio, F. Honsell and L. Liquori, "A Lambda Calculus of Objects with Self-Inflicted", Proceedings of OOPSLA. ACM Press, New York, p. 166–178, 1998.

[11] Langages pour la définition de contrats intelligents pour des blockchains autorisées et portefeuilles distribués, proposition de thèse Inria & Accenture, 2019.