Analyse d’un smart contract

Chapitre 3 : Infrastructure des smart contracts

3.1 Smart contacts

Le terme smart contract date de 1994, défini par Nick Szabo comme «un protocole de transaction informatisé qui exécute les termes d'un contrat. Les objectifs généraux de la conception intelligente de contrats sont de satisfaire les conditions contractuelles courantes (telles que les conditions de paiement, les privilèges, la confidentialité et même l'application), de minimiser les exceptions malveillantes et accidentelles et de minimiser le besoin d'intermédiaires de confiance. »

Les smart contracts étendent et exploitent la technologie blockchain. Un smart contract est une collection de codes et de données (parfois appelés fonctions et état) qui est déployée à l'aide de transactions signées cryptographiquement sur le réseau de la blockchain (par exemple, les contrats intelligents d'Ethereum, le code de chaîne d'Hyperledger Fabric). Le smart contract est exécuté par des nœuds au sein du réseau blockchain; tous les nœuds qui exécutent le smart contract doivent dériver les mêmes résultats de l'exécution, et les résultats de l'exécution sont enregistrés sur la blockchain

Les utilisateurs du réseau Blockchain peuvent créer des transactions qui envoient des données à des fonctions publiques offertes par un smart contract. Le smart contract exécute la méthode appropriée avec les données fournies par l'utilisateur pour effectuer un service. Le code, étant sur la blockchain, est également inviolable et résistant aux altérations et peut donc être utilisé (entre autres) en tant que tiers de confiance. Un smart contract peut effectuer des calculs, stocker des informations, exposer des propriétés pour refléter un état exposé publiquement et, le cas échéant, envoyer automatiquement des fonds à d'autres comptes. Il n'a même pas nécessairement à remplir une fonction financière. Par exemple, les auteurs de ce document ont créé un contrat intelligent Ethereum qui génère publiquement des nombres aléatoires fiables. Il est important de noter que toutes les blockchain ne peuvent pas exécuter de contrats intelligents.

Le code de smart contract peut représenter une transaction multipartite, généralement dans le contexte d'un processus métier. Dans un scénario multipartite, l'avantage est que cela peut fournir des données attestables et une transparence qui peuvent favoriser la confiance, fournir des informations qui peuvent permettre de meilleures décisions commerciales, réduire les coûts de la réconciliation qui existe dans les applications commerciales traditionnelles et réduire le temps de terminer une transaction.

Les smart contracts doivent être déterministes, dans la mesure où, étant donné une entrée, ils produiront toujours la même sortie en fonction de cette entrée. De plus, tous les nœuds exécutant le smart contract doivent se mettre d'accord sur le nouvel état obtenu après l'exécution. Pour y parvenir, les smart contracts ne peuvent pas fonctionner sur des données en dehors de ce qui leur est directement transmis (par exemple, les smart contracts ne peuvent pas obtenir de données de services Web à partir du smart contract - elles devraient être transmises en tant que paramètre). Tout contrat intelligent qui utilise des données en dehors du contexte de son propre système est censé utiliser un «Oracle». (Le problème d'Oracle est décrit dans la section 6.4 de la partie 1).

Pour de nombreuses implémentations de blockchain, les nœuds de publication exécutent le code de smart contract simultanément lors de la publication de nouveaux blocs. Il existe certaines implémentations de blockchain dans lesquelles il existe des nœuds de publication qui n'exécutent pas de code de smart contract, mais valident à la place les résultats des nœuds qui le font. Pour les réseaux blockchain sans autorisation activés par smart contract (tels que Ethereum), l'utilisateur émettant une transaction vers un smart contract devra payer le coût de l'exécution du code. Il existe une limite au temps d'exécution pouvant être consommé par un appel à un smart contract, en fonction de la complexité du code. Si cette limite est dépassée, l'exécution s'arrête et la transaction est rejetée. Ce mécanisme récompense non seulement les éditeurs pour l'exécution du code de smart contract, mais empêche également les utilisateurs malveillants de déployer puis d'accéder à des contrats intelligents qui effectueront un déni de service sur les nœuds de publication en consommant toutes les ressources (par exemple, en utilisant des boucles infinies).

Pour les réseaux blockchain autorisés par smart contract, tels que ceux utilisant le code de chaîne d'Hyperledger Fabric, les utilisateurs peuvent ne pas être tenus de payer pour l'exécution du code de smart contract. Ces réseaux sont conçus pour avoir des participants connus et d'autres méthodes de prévention des mauvais comportements peuvent être utilisées (par exemple, la révocation de l'accès).

3.2 Cycle de vie d'un smart contract

Le cycle de vie d'un smart contract se compose généralement de quatre grandes phases: création du contrat intelligent, gel du contrat intelligent, exécution du contrat intelligent et finalisation du contrat intelligent.

3.2.1 Création :

La phase de création peut être divisée en une négociation de contrat itérative et une phase de mise en œuvre. Premièrement, les parties doivent convenir du contenu et des objectifs généraux du contrat. Cela peut être fait en ligne ou hors ligne et est similaire aux négociations contractuelles classiques. Toutes les parties participantes doivent avoir un portefeuille sur la plate-forme de comptabilité sous-jacente. Son identifiant est dans la plupart des cas pseudonyme, et il est utilisé pour l'identification des parties ainsi que pour le transfert de fonds.

Après s'être mis d'accord sur les objectifs et le contenu du contrat, le l'accord doit être transformé en code. La codification du contrat est limitée par l'expressivité du langage de codage du smart contract sous-jacent. Pour valider l'exécution d'un smart contract comportement et contenu, la plupart des environnements de smart contract fournissent l'infrastructure pour créer, maintenir et tester le contrat. Comme on peut le voir dans les langages de programmation classiques, la transformation des exigences en code nécessite plusieurs itérations entre les parties prenantes et le (s) programmeur (s). Les smart contracts ne seront pas différents et nécessiteront probablement de nombreuses itérations entre la phase de négociation et de mise en œuvre.

Une fois que les parties ont accepté la version codifiée du contrat, il est soumis au grand livre distribué lors de la phase de publication. Au cours de cette phase, les nœuds participant au grand livre distribué reçoivent le contrat dans le cadre d'un bloc de transaction et une fois le bloc confirmé par la majorité des nœuds, le contrat est prêt à être exécuté.

Étant donné que les smart contracts décentralisés ne peuvent pas être modifiés après avoir été acceptés par la blockchain, les modifications du smart contrat ne sont pas possibles et nécessitent la création d'un nouveau contrat. Bien qu'un smart contrat ait été stocké sur la blockchain, ce fait à lui seul ne devrait pas être considéré comme un accord de la partie pour conclure le contrat car tout le monde peut soumettre un smart contract à la blockchain indiquant une obligation pour tout propriétaire de portefeuille aléatoire. De même, les smart contracts décentralisés peuvent bénéficier à tout participant de la blockchain, qu'il ait accepté ou non de bénéficier des avantages au préalable.

3.2.2 Gel :

Une fois le smart contrat soumis à la blockchain, il est persisté par une confirmation majoritaire des nœuds participants. En échange de ce service, et pour éviter une inondation de l'écosystème avec des smarts contrats, une redevance doit être payée aux mineurs. À partir de ce moment, le contrat et toutes les parties sont publics et accessibles via le grand livre public. Au cours de la phase de gel, tous les transferts effectués vers l'adresse portefeuille du smart contrat sont gelés et les nœuds jouent le rôle d'un conseil de gouvernance, garantissant que les conditions préalables à l'exécution du contrat sont remplies.

3.2.4 Exécution :

Les contrats stockés sur le grand livre distribué sont lus par les nœuds participants. L'intégrité du contrat est validée et le moteur d'inférence de l'environnement de smart contract (compilateur, interprète) exécute le code. Les entrées pour l'exécution sont collectées auprès des oracles intelligents et des parties impliquées (engagement envers les marchandises par le biais de pièces) et les fonctions du smart contract sont exécutées. L'exécution du smart contract entraîne un ensemble de nouvelles transactions ainsi qu'un nouvel état du smart contract. L'ensemble des résultats ainsi que les nouvelles informations d'état sont soumis au grand livre distribué et sont validés par le protocole de consensus.

3.2.5 Finalisation :

Une fois le smart contrat exécuté, les transactions résultantes et les nouvelles informations d'état sont stockées dans le grand livre distribué et confirmées selon le protocole de consensus. Les actifs numériques précédemment engagés sont transférés (dégel des actifs) et avec la confirmation de toutes les transactions, le contrat a été exécuté.

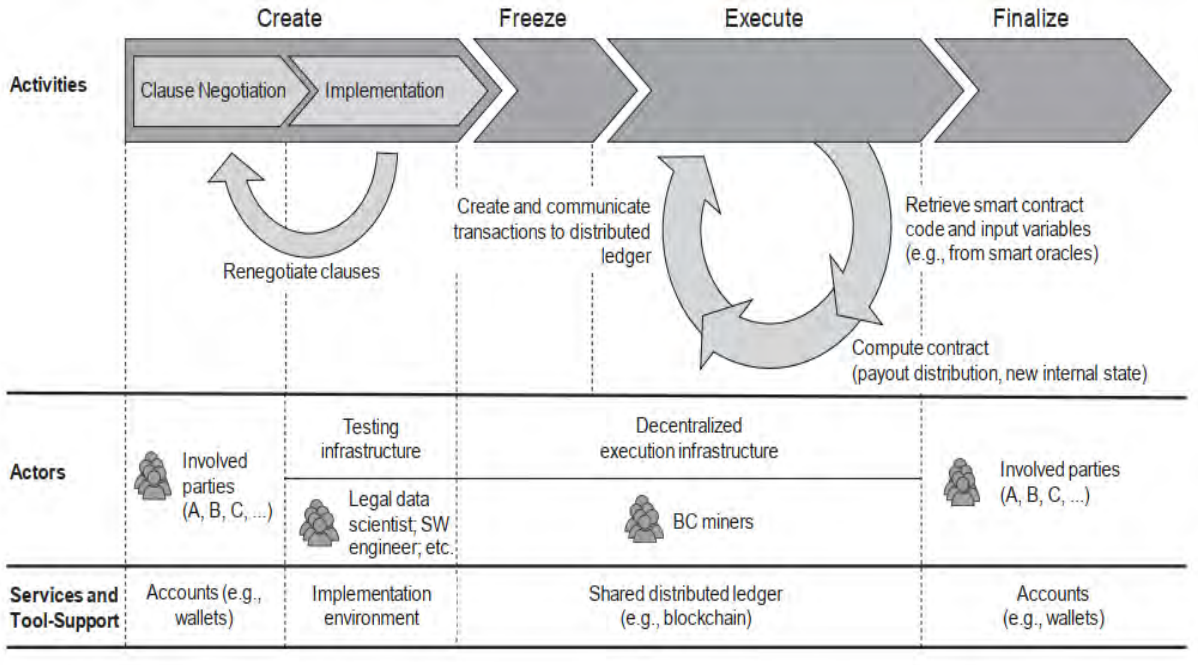

Figure 7 Le cycle de vie d'un contrat intelligent: phases, acteurs et services

3.3 Modélisation UML d'un smart contract

Les smart contracts pour Ethereum sont généralement écrits en utilisant le langage Solidity. Solidity est un langage orienté objet, et les contrats y sont définis comme des classes ils ont une structure de données, des fonctions publiques et privées, et peuvent hériter d'autres contrats. Les smart contracts ont également des concepts spécifiques comme les événements et les modificateurs.

Pour aider à la modélisation des smart contracts, nous utilisons des diagrammes UML. Cependant, comme les smart contracts ont des caractéristiques très spécifiques, nous avons introduit de nouveaux concepts dans ces diagrammes, pour pouvoir mieux modéliser et spécifier les smart contracts. Dans la mesure du possible, ces concepts sont simplement introduits sous forme de stéréotypes UML, qui sont des balises qui peuvent être utilisées dans les diagrammes UML lorsque cela est nécessaire. Dans quelques autres cas, nous avons dû introduire une notation spécifique, comme le transfert des Ethers dans les diagrammes de séquence. Les diagrammes UML que nous trouvons utiles pour modéliser les smart contracts sont:

- Diagrammes de classes, pour représenter la structure et les relations des smart contracts; nous avons introduit divers stéréotypes dans ce type de diagramme.
- Diagramme de d'états, pour représenter les différents états d'un smart contract; ce schéma n'a pas besoin de nouveau concept.
- Diagrammes de séquence, pour représenter les messages envoyés à un smart contract, et d'un smart contract à un autre smart contract; ce diagramme a besoin de nouveau type de messages - le transfert des Ethers.

Les diagrammes de classes UML sont utilisés pour représenter les smart contracts et les structures. Dans Solidity, il n'y a pas le concept de classe, mais les smart contracts sont très similaires aux classes. Comme une classe, un smart contract peut avoir une structure de données, des fonctions publiques et privées, et peut hériter d'un ou plusieurs smart contracts. Cependant, les SC ont une nature spécifique; ils sont créés par des transactions, mais une transaction peut créer au plus un seul smart contract. Les smart contracts, cependant, peuvent envoyer des messages à d'autres smart contract, résidant dans la même Blockchain. Dans Solidity, il est également possible de définir des structures, c'est-à-dire des structures de données complexes, qui ne sont pas pourvues de fonctions. Par conséquent, le modèle d'un smart contract créé par une transaction peut inclure d'autres smart contract dont il hérite, les structures utilisées et les smart contracts externes auxquels sont envoyés des messages.

D'autres concepts spécifiques des smart contracts Ethereum sont des événements, des drapeaux qui sont levés quand quelque chose de pertinent se produit, et qui le signalent au monde extérieur (qui doit observer le smart contract de manière autonome et agir en conséquence), et des modificateurs, fonctions spéciales qui sont appelées avant une fonction, vérifiant ses contraintes et éventuellement arrêtant l'exécution. Le tableau 2 montre les stéréotypes que nous avons introduits pour permettre de représenter les concepts smart contract dans les diagrammes de classes UML. Les événements pourraient être représentés dans un autre compartiment, en plus de ceux contenant le nom, l'attribut et les fonctions (opérations).

Stéréotype	Position	Description
Contrat	Symbole de classe - compartiment supérieur	Désigne un smart contract.
«Contrat de bibliothèque»	comme ci-dessus	Un contrat tiré d'une bibliothèque (standard)
«Structure»	comme ci-dessus	Une structure, contenant des données mais aucune opération, définie et utilisée dans la structure de données d'un contrat
«énumération»	comme ci-dessus	Une énumération contenant uniquement une liste de valeurs possibles
«interface»	comme ci-dessus	Un contrat contenant uniquement des déclarations de fonction
«modifier»	Symbole de classe - compartiment inférieur	Un type particulier de fonction, défini dans Solidity
«Tableau»	Rôle d'une association	La relation 1: n est implémentée à l'aide d'un tableau
« Map »	comme ci-dessus	La relation 1: n est implémentée à l'aide d'un mappage
«Map [uint]»	comme ci-dessus	La relation 1: n est implémentée à l'aide d'un mappage de l'entier à la valeur

Tableau 3 Ajouts au diagramme de classes UML (stéréotypes)

Les trois derniers stéréotypes définissent la mise en œuvre de relations 1: n dans la structure de données d'un smart contract. Dans Solidity, les seules collections prises en charge pour gérer le stockage (les données stockées en permanence dans la Blockchain) sont le tableau et le mappage. Le premier est un tableau de base classique de tous les langages informatiques, avec l'ajout que de nouveaux éléments peuvent y être ajoutés (mais pas supprimés). Le mappage est une collection capable de stocker des paires clé-valeur et de récupérer efficacement une valeur, compte tenu de sa clé, mais pas capable d'itérer sur ses éléments. Le dernier stéréotype fait référence à un modèle commun de programmation Solidity en utilisant un mappage avec des entiers positifs comme clés, de sorte qu'il est possible d'itérer dessus.

Les diagrammes de séquence UML sont utilisés pour modéliser la messagerie. Dans Ethereum, les messages sont associés à des transactions envoyées à la Blockchain depuis des utilisateurs

ou des systèmes externes ou depuis des smart contracts. Comme dans le jargon orienté objet, les messages sont synonymes d '«appels de fonction publique». Si une fonction n'écrit pas ou ne modifie pas la Blockchain (on l'appelle une fonction «voir»), le message correspondant peut être envoyé sans frais. Les autres messages doivent être payés en GAS[21] pour être exécutés.

Les spécificités d'Ethereum concernant la messagerie sont les types de participants (identifiés par leurs comptes) et les types de messages. Les participants peuvent être spécifiés à l'aide de stéréotypes, comme indiqué dans le tableau 3. Les messages nécessitent une notation spécifique.

Stéréotype	**Description**
« Personne »	Un rôle humain, l'envoi de messages à l'aide d'un portefeuille ou d'une autre application
«Système»	Un système externe, capable d'envoyer des messages à la Blockchain
«Device»	Un appareil (généralement IoT), capable d'envoyer des messages
«Contrat»	Un smart contract, faisant partie du système ou externe à celui-ci
«Oracle»	Un type particulier de smart contract, dont la date est écrite par un tiers de confiance, et permet d'accéder à des informations sur le monde extérieur
« Compte »	Un compte Ethereum, détenant simplement Ethers. Il ne peut recevoir des Ethers, ou envoyer des Ethers vers un autre compte ou smart contract que si le propriétaire active le transfert

Tableau 4 Ajouts au diagramme de séquence UML (stéréotypes)

Les différents types de messages pertinents pour la conception sont:

- Création de smart contract : il est envoyé par un participant externe ou par un autre smart contract; dans un diagramme de séquence, une création est représentée en dessinant le nouveau participant au niveau temporel de sa création.
- Appel de fonction : une transaction entraînant une modification de la Blockchain, et donc un paiement GAS; c'est le message «synchrone» ou «asynchrone» classique.
- Appel de fonction View / pure : une transaction n'entraînant aucune modification de la Blockchain et aucun paiement GAS; il peut être modélisé en ajoutant le stéréotype «vue» ou «pur» au nom du message.

[21] *https://www.investopedia.com/terms/g/gas-ethereum.asp*

- Transferts ETH : une transaction spéciale qui transfère des Ethers d'un compte, ou d'un smart contract, vers un autre compte ou smart contract. Ceci est modélisé à l'aide d'une flèche spéciale, similaire à la flèche d'héritage des diagrammes de classes UML.

Chapitre 4 : Etude des plateformes

4.1 Plateformes d'un smart contract

Les smart contracts sont des moyens d'échanger de l'argent, des actions, des propriétés ou toute forme d'actif de manière transparente, sécurisée et sans conflit tout en omettant la nécessité d'une intermédiation par un intermédiaire. Tout au long de cette section, nous passerons en revue les plates-formes de smart contracts les plus largement utilisées qui se sont avérées efficaces et fiables dans diverses applications commerciales.

4.1.1 Ethereum :

Ethereum est une plate-forme décentralisée basée sur la blockchain qui exécute des smart contracts, qui a également ouvert la porte à des applications décentralisées (DApp). La machine virtuelle Ethereum (EVM) est une machine virtuelle qui exécute tous les contrats intelligents. EVM est une machine virtuelle 256 bits Turing Complete[22]. Les smart contracts basés sur Ethereum sont codés à l'aide de Solidity, qui est un langage de programmation Turing Complete qui permet le codage des instructions de code en boucle et en dérivation. La «complétude de Turing» de Solidity rend Ethereum idéal pour coder des smart contracts avec une logique sophistiquée.

Le «GAS» est le carburant des smart contracts d'Ethereum. Il quantifie la quantité de puissance de calcul nécessaire pour exécuter des smart contracts via l'EVM. Lorsque vous soumettez un smart contracts, vous devez déterminer sa valeur en gas. Chaque étape du code du smart contracts nécessite l'exécution d'une quantité de gas prédéterminée.

Les contrats intelligents Ethereum peuvent :

- Agir comme des comptes ethereum "multi-signatures", de sorte que les pièces ne sont dépensées que si un nombre prédéterminé d'utilisateurs sont d'accord
- Offrir l'utilité à d'autres contrats intelligents sur la blockchain d'Ethereum
- Enregistrez des informations sur la propriété des actifs, l'enregistrement de domaine, les privilèges d'adhésion, les droits d'application, et plus encore
- Gérez les accords entre plusieurs parties, tels que les locations, la collaboration commerciale et l'assurance
- Être codé pour émettre des jetons tels que les jetons ICO utilisés pour le financement participatif. Il existe plusieurs normes de jetons utilisées pour émettre des jetons sur la plate-forme Ethereum, y compris les normes ERC-20, ERC223 et ERC77. ERC-20 est la norme la plus couramment utilisée pour émettre des jetons à des fins d'ICO, malgré ses graves bugs qui ont déjà entraîné des millions de dollars de pertes dans l'industrie de la cryptographie.

[22] *https://fr.wikipedia.org/wiki/Turing-complet*

ERC-20 effectue une transaction de jeton via l'une des deux méthodes suivantes:

- Transfer () cette fonction déclenche l'envoi de jetons à l'adresse d'un utilisateur spécifique.
- Approve () + transferFrom () : cette fonction déclenche le dépôt de jetons dans un smart contract prédéfini.

Cependant, si la fonction transfer () est accidentellement utilisée pour envoyer des jetons à un smart contract, la transaction sera exécutée avec succès, mais cette transaction ne sera jamais reconnue par l'adresse du smart contract du destinataire. Ce bug a inspiré les développeurs à créer les standards ERC223 et ERC77.

- ERC223 : cette norme atténue le bogue critique de l'ERC-20 en modifiant la fonction transfer () afin qu'elle déclenche une erreur en réponse à des transferts invalides et annule la transaction afin qu'aucuns fonds ne soient perdus.
- ERC777 : Cette norme résout le problème de l'ERC20 de manque d'opérations de gestion des transactions.

Ethereum est un choix populaire pour créer des smart contracts, mais les problèmes d'évolutivité de la plate-forme la rendent inadaptée à de nombreuses applications du monde réel. Solidity manque de flexibilité de codage fournie par les langages de programmation plus récents. Solidity ne prend pas en charge les tableaux multidimensionnels dans les paramètres d'entrée ainsi que les paramètres de sortie. De plus, Solidity ne prend en charge que 16 paramètres dans une fonction de smart contracts.

Malgré cela, les smart contracts basés sur Ethereum sont utilisés dans diverses applications. Par exemple, PCHAIN[23] a été le premier projet de blockchain à créer un système natif à chaînes multiples qui prend entièrement en charge la machine virtuelle Ethereum (EVM) - l'environnement d'exécution pour les smart contracts Ethereum.

4.1.2 EOS :

EOS devient de plus en plus l'une des plateformes de smart contracts les plus populaires. La plate-forme a attiré l'attention de la communauté cryptographique pour une multitude de raisons, à savoir que les transactions sur la plate-forme nécessitent des frais presque nuls et la capacité de la plate-forme à gérer des millions de transactions par seconde.

Les smart contracts sont programmés à l'aide de C ++, ce qui augmente la flexibilité de programmation. Les smart contracts EOS sont mis en œuvre sur la blockchain sous la forme d'un assemblage Web précompilé (WASM), qui favorise une exécution plus rapide des contrats par rapport aux smart contracts basés sur Ethereum. WASM est compilé avec C / C ++ via clang et LLVM. Les développeurs doivent avoir des connaissances en C / C ++ afin de pouvoir coder des smart contracts sur la blockchain d'EOS. Même si C peut être utilisé pour créer des contrats, il est fortement recommandé d'utiliser l'API EOS.IO C ++, qui renforce la sécurité des contrats et rend son code facilement lisible.

[23] *https://www.cointelligence.com/coins/pchain/*

EOS utilise le mécanisme de consensus de preuve de participation déléguée, qui agit avec une évaluation partielle et une exécution parallèle pour offrir une plate-forme de contrat intelligente avec des niveaux élevés d'évolutivité et des frais de transaction presque nuls.

Même si EOS est beaucoup moins populaire qu'Ethereum, il a établi le modèle de "parachutage" en tant que concurrent du modèle de financement participatif ICO d'Ethereum.

4.1.3 AION :

Aion est une plate-forme de smart contract qui permet le routage des transactions et des messages entre différentes blockchain via ses protocoles innovants de «pontage». Aion est un réseau à plusieurs niveaux comprenant les composants suivants :

- Ponts
- Réseaux de connexion
- Transactions inter-chaînes
- Réseaux participants

Les ponts d'Aion permettront d'effectuer des transactions sur plusieurs blockchain (transactions inter-chaînes) via l'écosystème de blockchain AION. Les transactions entre chaînes sont exécutées via des ponts et des réseaux de connexion. Les réseaux de connexion représentent les protocoles par lesquels toutes les blockchain publiques et privées peuvent communiquer avec l'écosystème de blockchain d'AION. Les réseaux participants sont des réseaux qui ont rempli un ensemble spécial d'exigences pour faire partie de l'écosystème blockchain d'AION. Les réseaux participants doivent prendre en charge la diffusion des transactions atomiques et mettre en œuvre un temps de verrouillage qui leur permettra de geler les transactions qui entrent dans un état "Oh Hold".

La machine virtuelle Aion (AVM) permet l'exécution de smart contracts. AVM est une implémentation JVM conçue pour exécuter la logique de chaîne. Le langage Aion est le langage de script utilisé pour programmer des contrats intelligents dans AVM. Actuellement, le noyau d'Aion fonctionne sur Java, les développeurs doivent donc utiliser des langages comme Python ou Groovy pour coder des smart contracts sur la blockchain d'Aion. Cependant, la plate-forme s'appuiera à terme sur le langage Aion pour l'écriture de smart contract.

Aion-1 est la plate-forme autonome d'Aion qui permet l'exécution de smart contracts créés sur d'autres blockchain. À l'heure actuelle, Aion s'appuie sur l'EVM d'Ethereum, mais finalement Aion-1 sera activé et permettra aux développeurs d'exécuter leurs smart contracts et DApp basés sur Ethereum beaucoup moins cher et plus rapidement que sur l'EVM.

4.1.4 NEM :

NEM est une plate-forme de smart contract plus évolutive qu'Ethereum. Là où Ethereum peut gérer 15 transactions par seconde, NEM peut gérer des centaines de transactions par seconde. NEM est plus rapide, plus sécurisé et fournit une technologie de smart contract simple. NEM utilise du code hors blockchain pour programmer des smart contracts, ce qui rend la blockchain de NEM moins décentralisée que celle d'Ethereum, tout en promouvant des niveaux de sécurité plus élevés, une confirmation plus rapide des transactions et un code de programmation plus léger. Les fonctionnalités de sécurité en ligne de NEM telles que les signatures multiples et les actifs intelligents résolvent ce problème.

Les actifs intelligents sont des applications de gestion de données uniques qui peuvent être utilisées pour créer des enregistrements de données, des jetons, des systèmes de vote et de nouvelles pièces en utilisant un code de programmation simple. L'extrême fonctionnalité de la blockchain de NEM est fournie via sa puissante API, qui permet l'utilisation de tout langage de programmation (comme JS, Python et autres) pour coder des smart contracts. L'API de NEM est utilisée pour développer des "contrats hors chaîne", qui peuvent être mis à jour à tout moment, sans communiquer avec la blockchain de NEM.

4.1.5 Stellar :

Stellar est une plate-forme de smart contract où les transactions sont plus sécurisées, plus rapides et moins chères que les transactions sur la blockchain d'Ethereum. Les smart contracts stellaires (SSC) ne sont pas Turing complete et sont déployés sous la forme d'accords programmés entre plusieurs parties qui sont appliqués par des transactions. Alors qu'il faut environ 3,5 minutes pour qu'une transaction soit confirmée sur la blockchain d'Ethereum, une transaction sur la blockchain de Stellar ne nécessite que 5 secondes environ pour être confirmée. Les frais de transaction sont négligeables, en moyenne autour de (0,0001 XLM ~ = 0,0000002 $). SSC peut être codé en utilisant n'importe quel langage de programmation tel que Python, JS, PHP, Golang et autres via l'API Stellar. Un SSC est composé de transactions qui sont interconnectées et exécutées au moyen de multiples contraintes, notamment les signatures multiples, le traitement par lots / atomicité, la séquence et les limites de temps. Le traitement par lots permet d'inclure plusieurs opérations dans une même transaction. L'atomicité garantit qu'à la soumission d'une série d'opérations au réseau de Stellar, toutes les opérations d'une transaction échoueront si une seule opération ne s'exécute pas. La séquence est un concept unique présenté sur la blockchain de Stellar via le "numéro de séquence". Avec les numéros de séquence, des transactions spécifiques échoueraient si une transaction alternative était exécutée avec succès. Les limites de temps représentent des limitations sur la période de validité d'une transaction. L'utilisation de limites de temps permet la représentation de périodes dans un SSC.

4.1.6 Hyperledger Fabric (HLF) :

Hyperledger Fabric (HLF) est une blockchain autorisée conçue avec une flexibilité avancée. Les smart contracts de HLF sont appelés «chaincode». HLF est écrit en langage Go, le langage de programmation open source de Google, donc chaincode prend également très bien en charge ce langage.

4.1.7 Corda :

Corda est une plate-forme de smart contract idéale pour créer des accords financiers. Les smart contracts de Corda sont des transactions valides qui doivent être acceptées par le smart contract de chacun de ses états d'entrée et de sortie. Les smart contracts sont codés à l'aide d'un langage de programmation JVM tel que Java ou Kotlin. L'exécution d'un smart contract est déterministe et son acceptation d'une transaction repose uniquement sur le contenu de la transaction. Parfois, la validité d'une transaction dépend d'une information externe, comme un prix symbolique. Dans ce cas, un oracle est nécessaire. Un fait peut être codé pour faire partie de la commande d'une transaction. Un oracle représente un service qui ne confirmera une transaction que si le fait de la commande est vrai.

Les DApp de Corda, ou CorDapps, sont installés au niveau des nœuds de réseau, plutôt que sur le réseau de blockchain lui-même. Les CorDapp sont codés en Java ou Kotlin. Les CorDapp sont codés pour fonctionner sur la plate-forme de Corda. Ceci est réalisé via la définition des flux que les opérateurs de nœuds Corda peuvent invoquer via des appels RPC.

4.1.8 NEO :

NEO est une plate-forme de smart contract qui propose des smart contracts efficaces et peu coûteux. Les smart contracts peuvent être codés à l'aide d'une myriade de langages de programmation, notamment C #, F #, Java, Python, VB.Net et Kotlin. NEO propose des plug-ins et des compilateurs pour toutes ces langues. À l'avenir, la prise en charge de JS, du langage Go, C et C ++ sera implémentée.

Les smart contracts de NEO sont exécutés via la machine virtuelle légère NEO (NeoVM). L'exécution de smart contract via NeoVM consomme un minimum de ressources. La compilation statique des smart contracts et la mise en cache des smart contracts de hotspot peuvent être considérablement améliorées via le compilateur en temps réel JIT. Actuellement, la blockchain de NEO comprend Smart Contract 2.0 qui prend en charge les structures de données et les tableaux complexes. De plus, Smart Contract 2.0 offre une approche évolutive via un partitionnement dynamique et une concurrence élevée, en combinaison avec une conception à faible couplage. La procédure de faible couplage des smart contracts est exécutée dans NeoVM et interagit avec les systèmes hors chaîne via une couche de service interactive. À ce titre, la plupart des mises à niveau de la fonction de contrat intelligent peuvent être réalisées via l'API spéciale de la couche de service interactif.

Dans notre travail nous allons choisir la plate-forme ethereum car elle fournit plus d'outil avec son environnement virtuel EVM, son langage de programmation solidity qui est orienté objet et dédié à l'implémentation des smart contracts, ainsi que les différentes framework.

4.2 Framework Ethereum

4.2.1 Langages des smart contracts :

Tout programme exécuté sur la machine virtuelle Ethereum (EVM) est généralement appelé un smart contract ou contrat autonome. Les langages les plus utilisés pour la rédaction de smart contracts sur Ethereum sont **Solidity** et **Vyper**, bien que d'autres soient en développement.

- **Solidity :** Le langage le plus populaire sur Ethereum, inspiré de C++, Python et JavaScript.
- **Vyper :** Langage axé sur la sécurité pour Ethereum, basé sur Python.

4.2.2 Outils de développement :

Ethereum dispose d'un nombre important et croissant d'outils pour aider les développeurs à créer, tester et déployer leurs applications.

- **Truffle :** Un environnement de développement, une structure de test, un pipeline de construction et d'autres outils ;
- **Embark :** Environnement de développement, Framework de test et autres outils intégrés à Ethereum, IPFS et Whisper ;
- **Waffle :** Un Framework pour le développement et les tests de smart contracts avancés (fondés sur ethers.js) ;
- **Etherlime :** Framework fondé sur Ethers.js pour le développement de Dapps (Solidity & Vyper), déploiement, débogage, tests, entre autres ;
- **Buidler :** Un gestionnaire de tâches pour les développeurs de smart contracts Ethereum ;
- **ZeppelinOS :** Un Framework de développement pour la création de smart contracts évolutifs, et de gestion sécurisée des applications de type smart contract ;

4.2.3 Environnements de développement intégrés (IDE) :

- **Visual Studio Code :** IDE professionnel multiplateformes avec un support officiel d'Ethereum ;
- **Remix :** IDE basé sur le Web avec analyse statique intégrée et une machine virtuelle de test de blockchain ;
- **Superblocs :** IDE basé sur le Web avec une machine virtuelle dans le navigateur, l'intégration de MetaMask, un logger de transactions et d'autres fonctionnalités ;
- **EthFiddle** IDE Web qui vous permet d'écrire, de compiler et de lancer votre smart contract ;

4.2.4 API front-end en JavaScript :

- **Web3.js :** API JavaScript pour Ethereum ;
- **Ethers.js *:*** Implémentation complète d'un portefeuille Ethereum, et utilitaires en JavaScript et TypeScript ;
- **light.js :** Une librairie JS réactive de haut niveau optimisée pour les clients légers ;
- **Web3-wrapper *:*** Alternative Typescript à Web3.js ;

4.2.5 API de back-end :

Infura *:* L'API Ethereum « as a service »

4.2.6 Outils de sécurité :

- **Slither :** Framework d'analyse statique Solidity écrit en Python 3 ;
- **MythX :** API d'analyse de sécurité pour les smart contracts Ethereum ;
- **Manticore :** Une interface en ligne de commande qui utilise un outil d'exécution symbolique sur les smart contracts et les fichiers binaires ;
- **Securify :** Scanner de sécurité pour les smart contracts Ethereum ;

4.2.7 Outils de test :

- **Solidity-Coverage :** Outil alternatif de couverture de code Solidity.
- **Hevm :** Implémentation de l'EVM spécialement conçue pour le test unitaire et le débogage de smart contracts.
- **Whiteblock Genesis :** Une sandbox de test et de développement de bout en bout pour la blockchain.

4.2.8 Explorateurs de block :

Les explorateurs de blocs sont des services qui vous permettent de parcourir la blockchain Ethereum (et ses testnets) en recherchant des informations sur les transactions, les blocs, les contrats et toute autre activité sur la chaîne. Tels qu'Etherscan, Blocskscout, Etherchain.

4.2.9 Testnets et faucets :

La communauté Ethereum gère plusieurs testnets ou réseaux de test. Ceux-ci sont utilisés par les développeurs pour tester leurs applications dans différentes conditions avant de les déployer sur le mainnet, le réseau principal d'Ethereum.

- **Ropsten :** Blockchain en preuve de travail, ether de test pouvant être miné.
- **Rinkeby :** Blockchain en preuve d'autorité (PoA), maintenue par l'équipe de développement de Geth.
- **Goerli :** Blockchain en preuve d'autorité (PoA) multi-clients, construite et maintenue par la communauté Goerli.

4.2.10 Clients (pour faire tourner le nœud) :

Le réseau Ethereum est composé de nombreux nœuds exécutant un logiciel client compatible. La majorité de ces nœuds exécutent Geth ou Parity, chacun pouvant être configuré de différentes manières en fonction de vos besoins.

- **Geth :** Client Ethereum écrits en Go.
- **Parity :** Client Ethereum écrit en Rust.
- **Ethnode :** Faire tourner un nœud Ethereum (Geth ou Parity) pour du développement en local.

4.2.11 Patterns et anti-pattern :

- **DappSys :** Construction de blocs sûrs, simples et flexibles pour smart-contracts.
- **OpenZeppelin :** Librairie pour le développement sécurisé des smart contracts.
- **aragonOS :** Patterns pour l'évolutivité et le contrôle des permissions.

Chapitre 5 : Modélisation des assurances non-vie

5.1 Etape d'une assurance non-vie

Pour souscrire à une assurance non-vie il y a différentes étapes à suivre pour l'assurer auprès de l'assureur qui sont :

- **Informations :**

Ici dans les informations, le client doit fournir des informations sur la situation et les détails de l'assurance auquel il veut souscrire, ensuite ses informations personnelles et le contrat actuel auquel il a souscrit.

- Tarif :

Le montant a payé par l'assuré pour le type de contrat qu'il veut souscrire.

- Paiement :

L'assuré effectue le paiement du contrat.

- Confirmation :

L'assurance enregistre la validation du contrat.

En cas de sinistre l'assuré réclame des indemnisations, voici les étapes à suivre :

- **L'enquête sur le sinistre commence.** Après la déclaration, l'expert en sinistres doit faire enquête afin d'établir le montant des pertes ou des dommages couverts par votre police. Il déterminera aussi qui est responsable du sinistre. Vous pouvez faciliter le processus en lui fournissant des renseignements sur les témoins ou les coordonnées d'autres parties.
- **Votre police est examinée.** Une fois l'enquête terminée, l'expert en sinistres examinera attentivement votre police pour déterminer ce qui est couvert par votre assurance, puis vous informera de toute franchise à payer.
- **Les dommages sont évalués.** Afin d'évaluer l'étendue des dommages avec précision, l'expert en sinistres peut recourir aux services professionnels d'estimateurs, d'ingénieurs ou d'entrepreneurs. Une fois l'évaluation terminée, il vous remettra une liste de fournisseurs privilégiés pour les réparations. Rien ne vous oblige à vous y limiter, mais elle pourrait vous faire économiser du temps de recherche.
- **Les dispositions relatives au paiement sont prises.** Une fois les réparations achevées et les articles endommagés ou perdus remplacés, l'expert en sinistres communiquera avec vous pour parler du règlement de votre demande et du paiement. Le délai de versement du paiement dépendra de la complexité et de la gravité de votre situation.

5.2 Analyse des situations en assurance où la blockchain peut être un outil indispensable

Les contrats intelligents stimuleront un jour l'automatisation du processus d'assurance. Le marché de détail y travaille déjà, mais la technologie changera-t-elle également le fonctionnement de l'assurance commerciale?

5.2.1 Plus rapide et moins cher :

En bref, les contrats intelligents sont des accords entre les parties où les mots sont remplacés par du code, afin qu'ils puissent être automatiquement lus par les ordinateurs. En conséquence, les obligations énoncées dans le contrat sont automatisées et peuvent être auto-exécutées.

Pour les acheteurs d'assurance, cela signifie dépendre moins de l'interprétation des clauses par les experts en sinistres et les avocats, car les indemnités seront payées une fois que des critères objectifs seront remplis. L'ensemble du processus sera ainsi plus rapide et moins cher, ce qui entraînera peut-être une baisse des taux de prime. Il pourrait également aider les compagnies d'assurance à combler les lacunes de couverture en en apprenant davantage sur les risques auxquels sont confrontés les acheteurs d'entreprises.

5.2.2 Souscription améliorée :

Le rapport du Lloyd's[24] souligne l'importance des données collectées par les entreprises qui mettent en œuvre des contrats d'assurance intelligents pour améliorer les processus de souscription.

Les contrats intelligents pourraient également ajuster automatiquement les taux de prime si les conditions rencontrées par l'actif assuré changent.

Par exemple, les navires qui souhaitent réduire les temps de voyage peuvent choisir de payer des tarifs plus élevés pour leur couverture d'assurance maritime et de choisir un itinéraire avec des risques de piratage plus élevés. Le contrat intelligent, dans ce cas, ajusterait automatiquement le taux et ferait le nouveau débit directement dans le compte du client. Et les informations sur les acheteurs eux-mêmes seront plus nombreuses, accélérant l'analyse effectuée par les transporteurs pendant le processus de souscription.

5.2.3 Moins de douleur avec les réclamations :

Pour les acheteurs d'assurance, cependant, la caractéristique la plus attrayante des contrats intelligents est la possibilité alléchante qu'un jour, leurs réclamations soient payées automatiquement sans tous les tracas qu'ils doivent endurer aujourd'hui.

Les contrats intelligents peuvent être liés à des couvertures d'assurance paramétriques, qui promettent déjà de payer les réclamations si certains déclencheurs convenus à l'avance sont respectés.

[24] *https://riskandinsurance.com/4-ways-smart-contracts-can-make-insurance-purchases-easier/*

L'idée est que, une fois la couverture déclenchée, le contrat intelligent procède déjà au paiement de l'indemnisation, sans qu'il soit nécessaire pour les experts en sinistres de se mêler ou même de contacter le transporteur.

5.2.4 Lutter contre la fraude :

Un autre avantage des contrats intelligents pourrait être la réduction de la fraude à l'assurance.

En théorie, ils peuvent être mis en œuvre avec différentes technologies informatiques. Mais la tendance est qu'ils sont intégrés dans des systèmes de registres distribués tels que la blockchain, dont le principe principal est que chaque modification du contrat est enregistrée en permanence. Cela réduit le risque que les participants au contrat tentent quelque chose de drôle et évitent la détection.

De plus, les données des revendications antérieures seront facilement disponibles pour aider à identifier la non-conformité dans le processus de traitement des réclamations qui évolue rapidement.

5.2.5 Analyse de décision spécifique pour l'assurance non-vie :

Cette analyse permet de savoir si la blockchain est une solution appropriée pour un problème défini. Elle ne vise pas à fournir une réponse finale faisant autorité mais à aider les décideurs de haut niveau à évaluer l'opportunité de déployer des ressources dans l'exploration d'une solution basée sur la blockchain à un espace de problème donné et, si oui, à quelle échelle. L'espoir est que le fait de se concentrer sur le problème commercial, et loin d'une solution particulière, atténuera les effets du battage médiatique entourant cette technologie et encouragera une approche pratique tout en réduisant le risque d'expérimentation mal avisée.

L'analyse est composée d'un certain nombre de questions qui aident à définir si une blockchain est la bonne approche pour une entreprise particulière ou non.

- ✓ Pour qu'une blockchain soit une solution appropriée, il est important de comprendre le contexte commercial - le problème commercial nécessite-t-il la suppression d'un intermédiaire? Par exemple, serait-il moins coûteux de collaborer directement avec des fournisseurs / concurrents plutôt que d'utiliser un centre d'échange? Un exemple de ceci est le secteur bancaire utilisant une solution telle que CORDA pour gérer les envois de fonds entre eux; cela leur permet de fournir des services plus rapidement, en toute sécurité et à moindre coût qu'avec les technologies existantes. Pour ce faire, ils redéfinissent la manière dont les processus métier sont mis en œuvre dans leur secteur. Un autre exemple peut être la suppression de courtiers d'une industrie - comme un courtier en technologie ou un courtier d'assurance.

- ✓ Pour que la blockchain soit appliquée avec succès, elle doit fonctionner avec des actifs «natifs numériquement», c'est-à-dire des actifs qui peuvent être représentés avec succès dans un format numérique. Bien que cela puisse sembler complexe, c'est en fait relativement simple. Si un actif a une représentation physique qui peut changer de forme, il est difficile de gérer efficacement cet actif sur une blockchain. Un exemple de cela est le suivi et le traçage des aliments sur la blockchain - si une entreprise souhaite suivre et tracer le blé sur toute la chaîne d'approvisionnement au fur et à mesure qu'il devient du pain, il est difficile d'utiliser la blockchain pour gérer sa transition du blé, de la farine, du pain.

- ✓ Un enregistrement permanent peut-il être créé pour l'actif numérique en question? C'est peut-être la question la plus critique, car une blockchain doit être la source de confiance. S'il existe plusieurs sources de confiance concernant l'état d'un objet, l'objet ne peut pas être stocké efficacement sur la blockchain. Dans les cas où un enregistrement permanent peut être créé, il est important que toutes les parties responsables de l'état de l'actif numérique en question conviennent de la manière dont cet état sera géré / géré dans le nouveau processus opérationnel avant tout développement. Séparément, un dossier permanent est-il souhaitable? Si un enregistrement inaltérable est superflu ou contre-productif, par exemple, dans une situation où la nécessité de supprimer des informations est critique, alors la blockchain / DLT (Distributed Ledger Technology) n'est pas une solution appropriée. Il ne serait pas logique, par exemple, de stocker une liste d'épicerie ordinaire sur une blockchain.

- ✓ À ce stade, il convient également d'évaluer la vitesse requise pour le processus opérationnel en question. Si cela nécessite des performances en millisecondes sur les transactions, les chaînes de blocs ne sont pas encore en mesure de gérer cela efficacement et il est conseillé de travailler avec les technologies existantes ou d'attendre que les chaînes de blocs puissent gérer ces vitesses de transaction. Depuis avril 2018, diverses formes de DLT ont un temps de traitement de deux à 10 minutes.

- ✓ Il n'est actuellement pas conseillé de stocker des données non transactionnelles sur une blockchain. Si cela est nécessaire pour un cas d'utilisation spécifique, il n'est pas conseillé d'utiliser une blockchain. Si, cependant, la confiance en question est liée aux enregistrements de transaction (plutôt qu'aux données sous-jacentes elles-mêmes), alors une blockchain peut être applicable. Dans tous les cas, toute information privée ou toute donnée pouvant être en conflit avec les réglementations locales et mondiales de protection des données, telles que le RGPD[25], ne doivent pas être stockées sur la blockchain.

25

https://fr.wikipedia.org/wiki/R%C3%A8glement_g%C3%A9n%C3%A9ral_sur_la_protection_des_donn%C3%A9es

- ✓ Si une industrie a des exigences spécifiques concernant l'utilisation d'intermédiaires ou de partenaires de confiance, il peut être compliqué de déployer la blockchain, même si d'autres avantages de son utilisation sont facilement apparents. Dans les cas d'utilisation où la réglementation joue un rôle important, il peut être nécessaire d'inclure les régulateurs dans le projet et de fournir des moyens par lesquels les régulateurs peuvent garantir le respect des lois, telles que la législation antitrust et la concurrence. Cet engagement sera un élément essentiel qui doit être abordé dans de nombreuses industries. Un exemple est un secteur qui a des exigences strictes de la part de plusieurs régulateurs, tels que les lois antitrust et environnementales, dont chacune requiert une visibilité sur un aspect différent des données de transaction, et où l'émetteur ne cherche pas à afficher l'intégralité des données de transaction à aucun un régulateur pour des raisons juridiques ou autres. Il pourrait être assez difficile de déployer une blockchain pour cette situation sans engagement réglementaire.

- ✓ Pour que la blockchain aide à réduire les coûts et à fournir une valeur commerciale réelle, il est important qu'une blockchain se penche sur la gestion des transactions autour des actifs numériques - si un problème commercial ne concerne pas vraiment la gestion des relations contractuelles et de l'échange de valeur, alors il n'y a guère besoin d'une blockchain - une technologie différente pourrait probablement résoudre ce problème plus efficacement.

- ✓ Le cas d'utilisation nécessite-t-il un accès en écriture partagé? En d'autres termes, certains / tous les membres du réseau en question doivent-ils pouvoir écrire des transactions dans la blockchain? Si le cas d'utilisation ne nécessite pas un tel accès en écriture partagé, une autre technologie fournira probablement une meilleure solution.

- ✓ Si les acteurs / entités se connaissent déjà et se font confiance, il n'y a probablement pas besoin de blockchain. S'ils ne se connaissent pas ou ne se font pas confiance et / ou ont des intérêts mal alignés, il peut y avoir une bonne raison d'utiliser la blockchain.

- ✓ Si la possibilité de modifier les fonctionnalités d'une blockchain (par exemple, la distribution des nœuds, les autorisations, les règles d'engagement, etc.) sans avoir une discussion détaillée sur les grands forums open source pour la blockchain est souhaitable, alors vous devez sélectionner une blockchain privée et autorisée.

- ✓ Si les transactions doivent rester privées, une blockchain privée autorisée est appropriée. Sinon, une blockchain publique sans autorisation peut être utilisée.[26]

[26] *http://www3.weforum.org/docs/48423_Whether_Blockchain_WP.pdf*