UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN MATHÉMATIQUES ET INFORMATIQUE APPLIQUÉES

PAR
NATHAN SICARD

MÉTHODOLOGIE DE CONCEPTION ET D'IMPLÉMENTATION DE LA TECHNOLOGIE BLOCKCHAIN DANS LE SECTEUR INDUSTRIEL

MARS 2023

Résumé

Le domaine de la *blockchain* est un domaine récent combinant les domaines de la réseautique avec le domaine d'entreposage de données et de la sécurité. Depuis sa création, la technologie *blockchain* gagne en popularité dans plusieurs domaines que ce soit avec des applications ouvertes à tous comme des achats en cryptomonnaie ou des applications dans les secteurs privés comme la traçabilité de produit. Étant donné la récence de cette technologie et son évolution continue, les standards de l'industrie sont en constante évolution. L'objectif de ce mémoire est donc d'identifier les méthodologies employées dans la conception et l'implémentation de projets employant la technologie *blockchain* et de proposer des améliorations possibles quant à la disponibilité de ressources d'apprentissage ainsi que dans son implémentation pratique en présentant un projet simplifié utilisant l'architecture de Hyperledger Fabric. Dans ce mémoire, nous présentons une étude des applications courantes de la *blockchain* avec une priorisation sur le secteur privé suivi d'une étude exploratoire de l'application de la technologie *blockchain*.

Mots-clés : Blockchain, Hyperledger Fabric, Traçabilité, Contrats intelligents

Abstract

Blockchain technology is a recent field of study which combines networking with data storage and security. Since its creation, blockchain technology has been gaining in popularity in various areas such as open applications allowing the exchange of cryptocurrency and private applications such as those used for product traceability. Due to its relative recency, as well as its continuous evolution, industry standards in blockchain technology are themselves in a constant state of change. This thesis aims to identify the methodologies employed in the conception and implementation of projects making use of blockchain technology, and to suggest possible improvements in the availability of learning resources as well as give practical examples of its use via a simplified project using the Hyperledger Fabric architecture. This thesis presents a study of current applications and concludes with an exploratory study of the application of blockchain technology with an emphasis on applications in the private sector.

Keywords: Blockchain, Hyperledger Fabric, Traceability, Smart contracts

Table des matières

Table des figures

Listings

Chapitre 1

Introduction

Conçue en 2008, la *blockchain* demeure une technologie émergente. *Bitcoin* a établi la *blockchain* comme étant une technologie résistante aux pannes, utilisant un réseau *Peer-to-peer* (P2P) et un lien immuable pour empêcher les fraudes d'information [Nakamoto, 2008]. Cette technologie, créée pour éliminer la dépendance du public envers les banques centralisées, a pavé le chemin pour l'avancement d'une technologie flexible capable de répondre à toutes sortes de besoins liés à la transparence et à la résistance aux pannes. *Ethereum* et *Hyperledger Fabric* sont deux des *blockchains* les plus populaires encore aujourd'hui au niveau public et dans l'industrie privée. La *blockchain* a vu la création de nombreuses nouvelles plateformes par la suite, telles que *Corda*, *Solana*, *Celo* et plusieurs autres. Chaque nouvelle *blockchain* tente de résoudre un problème d'une autre *blockchain* (tel qu'une réduction de la consommation électrique totale, l'augmentation de la privatisation des données privées, ou même la simplification de l'utilisation) ou l'implémentation de fonctionnalités n'existant pas déjà dans une autre *blockchain* (tel que le changement de protocole de consensus, ou l'implémentation de fonctionnalités de contrats intelligents).

1.1 Contexte

La globalisation des entreprises apporte des chaînes de communications de plus en plus grandes et complexes. Cette globalisation présente plusieurs avantages : le partage de connaissances, l'ouverture aux nouveaux marchés, ainsi que la répartition de produits précédemment indisponibles dans toutes les régions du monde. Cependant, l'expansion des chaînes de communication pose un problème de confiance envers les autres participants. En effet, alors qu'on peut faire confiance à un partenaire direct pour partager ses informations de manière adéquate, les partenaires du partenaire ne sont pas toujours connus. Ceci rend les entreprises réticentes à partager leurs informations avec la chaîne de communications entière. Les disputes logistiques sont communes, où la facture finale et l'estimation de l'acheteur ne correspondent pas et entraînent de longues périodes de vérification. Tout est dû à une asymétrie de l'information ; soit la possession d'informations différentes ou incomplètes des deux côtés des communicants.

1.2 Problématique

La problématique choisie comme base pour le projet s'inspire d'un problème de chaîne d'approvisionnement dans le domaine d'élevage de grains. Dans la chaîne d'approvisionnement étudié (qui sera expliquée plus en détail plus tard), on retrouve plusieurs organismes qui collaborent dans la traçabilité de la production de grain de soya. Chaque organisme exécute des traitements pouvant changer l'état du lot de grain de soya, ces traitements sont réservés à des organismes prédéfinis dans la liste de participants[Salah *et al.*, 2019]. Tous ces traitements et changements d'état sont codifiés dans des fonctions de contrat intelligent qui est responsable du contrôle des accès à ses fonctions, ainsi que l'enregistrement de leurs résultats (qui seront sur la

blockchain en permanence). Ceci permet une traçabilité complète de la chaîne d'approvisionnement tant que toutes les entreprises participent au système.

Dans le cas utilisé dans l'expérimentation (qui utilise un cas de distribution de pommes simplifié), trois niveaux de traitement ont été créés : soit une cueillette de pommes, un traitement des pommes cueillies et un système de distribution. Chaque niveau de traitement a un seul organisme associé. L'objectif du projet est de créer un réseau *blockchain* permettant de retracer chaque étape impliquée dans la production de pommes et déterminer la provenance d'une pomme en utilisant son identifiant unique.

1.3 Objectifs

L'objectif principal de la recherche présentée dans ce mémoire est d'étudier la facilité d'utilisation de technologies blockchain dans un contexte de traçabilité de chaîne d'approvisionnement et de proposer des points d'améliorations possibles dans les ressources disponibles. Nous avons donc séparé la recherche en trois sous-objectifs : l'analyse d'études de cas utilisant la technologie blockchain, la création d'une preuve de concept utilisant des fonctionnalités de contrats intelligents de la blockchain et une étude des ressources disponibles pour les développeurs (documentation et forums). Le but recherché est d'établir une structure de développement de la technologie blockchain sans une connaissance technique approfondie du système sous-jacent.

1.4 Approche

Étant donné l'étendue de la problématique de la traçabilité de produits dans la chaîne d'approvisionnement ainsi que l'étendue des capacités des réseaux *blockchains*, il a été nécessaire de cibler certains aspects de l'écosystème *Hyperledger Fabric* afin d'avoir une meilleure idée des possibilités d'implémentation avec cette technologie. La problématique a donc été séparée en trois parties distinctes au niveau architectural : le réseau, les contrats intelligents et les applications. Cette approche à la problématique suit l'approche recommandée dans le développement de projet en *blockchain*.

Premièrement, il faut choisir et configurer la *blockchain* qui sera utilisée (*Ethereum*, *Hyperledger Fabric*, ou autre) ainsi que définir sa nature publique ou privée[1], ouverte ou fermée[2]. Dans ce cas, cette approche implique de configurer les *channels* du réseau *Hyperledger Fabric* ainsi que les différents organismes qui y résident. Puisqu'il s'agit d'une solution privée, il faut aussi considérer la gestion d'accès à l'information dans le réseau. Le réseau présent est aussi un réseau dit fermé, indiquant que les droits de lecture sont réservés aux participants du réseau. On peut donc dire que le réseau est permissionné, utilisant des droits d'accès de lecture ou d'écriture variés. Dans le futur, une solution hybride à la fois ouverte et fermée pourrait être utilisée afin de permettre à des non-utilisateurs de lire des données de traçabilité, par exemple, pour connaître la provenance d'un produit ou connaître ses certifications (telle que la certification non-OGM ou la certification «100% végétalien»). Ces certifications traçables sur la *blockchain* ont le potentiel d'augmenter la valeur d'un produit par une plus grande confiance du consommateur[Dickinson et Bailey, 2002, Buhr, 2003].

Deuxièmement, il faut planifier les contrats intelligents ; soit le code exécuté dans le traitement de données du registre de la *blockchain*. Il faut non seulement décider

[1] Une solution privée indique que l'information peut seulement être ajoutée par des acteurs connus.
[2] Une solution fermée indique que l'information n'est pas visible aux acteurs externes.

quelles informations doivent être enregistrées sur la *blockchain*, mais aussi prendre des décisions architecturales dans l'organisation et la communication entre les contrats intelligents. Cette étape est essentiellement une étape de planification de logiciel consistant en des choix de patrons de conception (si applicable) ainsi que l'organisation de contrats intelligents individuels (qui peuvent être vus comme des classes logicielles). Il faut également prendre des décisions au niveau des accès aux contrats intelligents, qui sont englobés par l'aspect permissionné de la *blockchain*.

Troisièmement, il faut créer une application permettant d'interagir avec les contrats intelligents. Dans un cas d'entreprise, il est possible que la *blockchain* vienne supplémenter un système d'entreposage de données. La solution serait donc d'intégrer l'API de *Hyperledger Fabric* à la solution existante aux points d'intérêts. Dans le cas présent, il n'existe aucun logiciel préexistant qui est disponible publiquement. Une application simple a donc été créée afin de fournir une interface graphique lors des tests de fonctionnalités. Cette application permet à tous les utilisateurs du réseau à exécuter des transactions.

Le projet AppleNet est un projet axé sur la traçabilité de produits dans la chaîne d'approvisionnement et se distancie légèrement de l'aspect logistique de cette tâche, qui pourrait être un projet entièrement différent. Afin d'atteindre les objectifs indiqués précédemment, trois organismes fictifs ont été créés, chacun avec plusieurs départements aux tâches distinctes. Des permissions spécifiques pour chaque organisme ont été définies. Un seul contrat intelligent a été créé afin de vérifier la faisabilité de l'implémentation du réseau[3].

[3]On sous-entend que le réseau inclut tous ses acteurs, contrats intelligents et applications.

Chapitre 2

État de l'art

Étant donné la récence de la technologie *blockchain*, il existe peu d'études publiques de cas d'utilisation de *blockchain* au niveau industriel et, étant donné la nature privée des informations changées, il existe encore moins de démonstrations pratiques dans ces études. Cependant, il est possible d'inférer certains éléments de l'architecture informatique à partir des études de cas publiquement disponibles. De plus, le raisonnement menant au choix de *blockchain* est souvent inclus dans ces études permettant ainsi d'avoir une meilleure idée des cas qui eux, sont considérés comme étant viables pour l'utilisation de la technologie *blockchain*.

À cette fin, trois cas ont été étudiés : une étude de la possibilité de l'utilisation de la *blockchain* dans la traçabilité du grain de soya, une étude de cas de l'implémentation de la *blockchain Hyperledger Fabric* dans la traçabilité de produits alimentaires chez Walmart et une étude de cas de l'implémentation de la *blockchain Hyperledger Fabric* dans la conciliation de factures dans la logistique de produits chez Walmart. Dans cette section, nous présentons premièrement le contexte du développement des contrats intelligents et ensuite les trois cas étudiés. Le premier cas illustre des portions de pseudo-code donnant des indices de l'apparence d'un contrat intelligent ainsi que des

patrons qui pourraient être utilisés, alors que les deux autres cas donnent des exemples d'implémentations réelles ainsi que leurs justifications.

2.1 Le début des contrats intelligents

Un concept essentiel à l'exécution de solutions complexes dans le domaine de la *blockchain* est l'utilisation des contrats intelligents. Les contrats intelligents ont initialement étés définis dans les années 1990 par Nick Szabo. Dans l'article *Formalizing and Securing Relationships on Public Networks (1997)*, Nick Szabo définit les contrats intelligents comme étant des contrats encodés dans une pièce informatique permettant d'exécuter les clauses de ce contrat indépendamment de l'intervention d'un humain. Szabo décrit les machines distributrices comme étant un exemple de contrats intelligents puisque celles-ci peuvent agir en tant qu'interface de vente au consommateur sans nécessiter la présence du vendeur, permettant l'échange de monnaie pour les produits présentés[Szabo, 1997].

Avec la création de la *blockchain*, la définition de contrats intelligents a été modifiée. Dans le contexte de la *blockchain*, un contrat intelligent est un programme défini par un utilisateur s'exécutant sur la *blockchain* de manière décentralisée et / ou distribuée. Ces programmes exécutent toujours certaines fonctions (ou clauses) prédéfinies selon des entrées d'utilisateurs rappelant la définition du contrat intelligent de Szabo.

En 2014, Vitalik Buterin a publié le *whitepaper* d'Ethereum avec le titre *Ethereum : A Next-Generation Smart Contract and Decentralized Application Platform* détaillant le potentiel d'un réseau *blockchain* permettant l'utilisation de contrats intelligents avec la possibilité des manipulations complexes d'informations et de monnaie digitale (cryptomonnaie) nommée *Ether*. Dans cet article, Buterin décrit les possibi-

lités d'implémentation de contrats intelligents de manière décentralisée et distribuée, ouvrant la possibilité à la création de projets tels que des marchés d'échange de cryptomonnaie autonomes, des organismes autonomes décentralisés[1], ou même des comptes de gestions de fonds multisignatures complexes permettant de gérer les fonds selon des règlements prédéfinis[Buterin, 2014].

Cette architecture a permis le développement de nombreux projets tels que ceux mentionnés ci-tôt. Cependant, l'évolution (et le succès) rapide de cette technologie publiquement accessible et modifiable a mené à de nombreux questionnements au niveau légal. L'article *Is a 'smart contract' really a smart idea ? Insights from a legal perspective* par Mark Giancaspro, publié en 2017, évalue les difficultés au niveau légal occasionnées qui devront être résolues dans un monde interconnecté par la technologie *blockchain*. L'article note, entre autres, que l'application de lois serait difficile avec la *blockchain* au niveau international étant donné la nature distribuée mondiale de la *blockchain*, ainsi que son aspect pseudo-anonyme. Un exemple présenté à cet effet est l'age légal à partir duquel un individu peut signer un contrat. Cet article étudie les différences entre les lois en Australie, en Angleterre, en France et aux États-Unis ; notant les différents âges de majorité[Giancaspro, 2017]. Un des points les plus intéressant dans cette analyse provient de la lisibilité des contrats intelligents. En effet, la nature non intuitive des langages de programmation (amplifiée par les nouveaux langages spécifiques aux plateformes tels que le Solidity sur Ethereum) mène à une difficulté d'interprétation du contrat. Dans les cas où les contrats intelligents sont utilisés comme contrats digitaux tels que décrit par Szabo, la capacité de comprendre le code par tous les signataires est cruciale à la validité de ce contrat. En effet, la spécificité de la technologie requiert une connaissance non seulement en *blockchain*, mais aussi en programmation. L'auteur suggère que les entreprises voulant tirer avantage de cette technologie devront soit former leurs employés de haut niveau en programmation ou engager des programmeurs pouvant traduire leurs besoins en contrats intelligents et

[1]Un DAO est un organisme existant sans autorité centrale permettant de prendre des décisions via des votes sur la gestion des ressources de cet organisme. Les votes peuvent êtres distribués de différentes manières dont via la possession de *tokens*.

leurs contrats intelligents en clauses de contrat[Giancaspro, 2017].

L'étude de ces articles montre les deux côtés du monde de la *blockchain*. D'un côté, la *blockchain* offre de nombreuses possibilités et la communauté qui l'entoure est optimiste envers les cas d'utilisations possibles (encore à ce jour). De l'autre côté, la technologie a devancé le système légal et entrepreneurial et elle requiert de nombreuses améliorations à sa compréhensibilité, ainsi que des adaptations au niveau des systèmes légaux au niveau mondial avant d'assurer son acceptabilité en tant que vrai contrat digital.

2.2 La traçabilité du grain

Le premier cas étudié a été l'article *Blockchain-Based Soybean Traceability in Agricultural Supply Chain*[Salah *et al.*, 2019]. Cet article décrit le processus de vente de soya de la compagnie de vente de graines jusqu'à la vente du soya au consommateur final. Dans cet article, les auteurs proposent une suite d'algorithmes applicables à l'architecture *blockchain Ethereum*. Le projet profite de l'architecture publique d'*Ethereum* qui comporte quelques milliers de nœuds (ce nombre varie fréquemment dépendamment de l'intérêt du public ainsi que l'adhésion de nouveaux mineurs), permettant d'assurer l'immuabilité de l'information due à l'augmentation probabiliste de la difficulté de corruption du réseau. Le réseau public a aussi comme avantage d'être totalement décentralisé. Cette décentralisation assure qu'il n'y ait pas de biais entre les entreprises et qu'elles puissent fonctionner sans obligation de faire confiance à une autre entreprise dans l'entreposage de données ; elles n'ont qu'à faire confiance à la fiabilité du réseau. Les solutions et algorithmes présentés dans cet article résolvent seulement la problématique d'entrée de données, de contrôle d'état et une portion du contrôle d'accès ; agissant comme étude de cas plutôt que comme preuve de concept.

Le système proposé a sept participants qui interagissent tous avec un contrat intelligent. Le contrat intelligent agit à la fois comme modèle représentant les données et comme système de gestion d'état de produit, contenant à la fois les attributs du produit et des fonctions permettant d'affecter ces attributs. Chaque participant doit être enregistré auprès du contrat intelligent afin de participer. Le contrat intelligent est créé pour chaque commande de grains. Le contrat enregistre chaque état de cette commande au travers de tous les processus exécutés dans la préparation de celle-ci. Ces états permettent au contrat de vérifier automatiquement si la commande a suivi toutes les étapes requises avant de passer à la prochaine. Si une transaction de mise à jour d'état n'ayant pas suivi toutes les étapes requises est envoyée au contrat, le contrat intelligent annule la transaction et retourne à son état précédent (tout changement au contrat est annulé).

Cette conception souffre de plusieurs problèmes potentiels qui sont détaillés dans la conclusion, dont la possession de données, la protection de données, ainsi que la gestion des accès. En effet, puisque le réseau utilisé est public (et géré par des utilisateurs inconnus à travers le monde), les données envoyées au réseau sont entièrement publiquement visibles. Un tel réseau requiert donc des méthodes de protection d'informations supplémentaires telles que le cryptage de données sensibles. Puisque ces données sont publiques, il faut aussi faire confiance à la fiabilité du cryptage de données à long terme.

De plus, puisque le réseau utilisé est public, chaque transaction subit des frais de transaction qui peuvent varier de manière imprévisible (par exemple, les frais de transaction sur le réseau *Ethereum* étaient d'environ 0.05$ à 0.15$ USD en janvier 2019 et en janvier 2022 ont été entre 20 USD$ et 52$ USD avec un maximum d'environ 200$ USD en mai 2022[Etherscan, 2022]), réduisant la viabilité du système. Puisque le réseau utilise un protocole *open source* géré par les utilisateurs, le système est aussi dépendant des décisions du réseau (par exemple une augmentation de frais de transaction) qui pourraient affecter le fonctionnement du système. Sur le réseau

Ethereum, ces frais s'appliquent à la taille des données ajoutées, limitant la quantité de données pouvant être raisonnablement ajoutées par l'entremise du contrat intelligent, ainsi que le type de données. Par exemple, les médias ne sont pas insérés sur la *blockchain*, mais sont plutôt ajoutés au système d'entreposage de médias décentralisés IPFS[IPFS, 2014]. Une trace du média est préservée sur la *blockchain*, mais pas le média lui-même.

Aussi, selon le diagramme présenté dans l'article, le projet utilise une structure à un seul contrat intelligent. Cette structure implique que toutes les données sont dans la même structure logique et offre un degré très faible de modularité. Cette structure à un contrat implique aussi une contradiction au niveau de la conception décentralisée de la *blockchain*. En effet, s'il y a un seul contrat, il y a une seule autorité finale sur ce contrat. Ceci a comme effet de centraliser le contrôle de l'information et engendre plusieurs problèmes potentiels lorsque mis en pratique tels que la nécessité de confiance envers une autorité centrale (soit l'entité qui crée et publie le contrat) et la réduction du contrôle individuel d'une entreprise envers ses données. Une mise à jour des besoins d'une entreprise affectant le contrat devra se faire à travers cette entreprise possédant le compte responsable du contrat et pourrait décourager l'adhésion au projet. Malgré cette difficulté, toute information entrée sur un contrat demeure immuable sur la *blockchain Ethereum* peu importe à qui appartient le contrat. De plus, même si une nouvelle version d'un contrat est créée, les informations des versions précédentes existent toujours, assurant qu'aucune donnée ne puisse être perdue même dans le cas d'une entreprise qui quitte le système.

Afin de résoudre le problème de gestion d'accès, il est possible d'utiliser un contrat intelligent de gestion de rôle tel que celui proposé par *OpenZeppelin*, qui permet de bloquer l'utilisation de fonctions de contrat intelligent en vérifiant si l'identifiant de la personne appelant ce contrat possède le rôle permis[OpenZeppelin, 2021].

2.3 Traçabilité de produits

Walmart est une entreprise multinationale gérant une multitude de produits périssables. Ces produits sont à risque de plusieurs facteurs tels que la température de stockage ou des épidémies locales pouvant mener à l'expiration prématurée ou la contamination du produit. Ces problèmes de chaîne d'approvisionnement mènent à de nombreuses conséquences au niveau économique et social, sans mentionner les impacts sur la santé publique.

Walmart, comme de nombreuses autres entreprises dans le domaine alimentaire, utilisait une approche de traçabilité dite *one-up-one-back* ; soit une approche où chaque membre de la chaîne d'approvisionnement connaît seulement ses fournisseurs et ses clients directs[Heneghan, 2016]. Cette approche peut être priorisée afin de protéger les informations propriétaires d'un organisme. En limitant la portée des accès aux partenaires directs seulement, un organisme peut empêcher ses compétiteurs de visualiser certaines données sensibles. Cependant, plus la chaîne d'approvisionnement devient complexe, plus une visualisation bout à bout de la chaîne d'approvisionnement devient désirable étant donné les nombreux acteurs en jeu. Chaque acteur introduit des risques possibles que ce soit au niveau de la qualité ou au niveau de la confiance. En effet, un registre immuable, distribué et auditable a le potentiel de décourager la fraude.

Le cas de traçabilité alimentaire de Walmart concerne deux projets pilotes étudiant deux cas de traçabilité de produit dans différentes régions. Dans ce projet, Walmart, en partenariat avec IBM, a développé un réseau *Hyperledger Fabric* permettant de télécharger des certificats d'authenticité de viande de porc en Chine, ainsi qu'un réseau permettant de retracer la provenance de mangues aux États-Unis[Hyperledger, 2019b, Kamath, 2018].

Le premier cas, soit le téléchargement de certificats d'authenticité de viande de porc, est un supplément à la traçabilité, travaillant en parallèle avec les aspects de provenance permettant d'augmenter la confiance des acheteurs envers le produit. En combinant le téléchargement de certificats d'authenticité avec la technologie *blockchain*, Walmart et IBM s'assurent qu'aucun certificat ne puisse être modifié une fois que celui-ci est ajouté au système sans être signalé, évitant tout cas de fraude possible et augmentant la confiance des partenaires.

Le second cas, soit la traçabilité des mangues aux États-Unis, concerne la traçabilité bout à bout discutée précédemment. L'étude de cas de Walmart sur le site de *Hyperledger* indique qu'en implémentant la traçabilité de la chaîne d'approvisionnement entière, il a été possible de réduire le délai de traçabilité de mangues d'une semaine à environ 2,2 secondes. En effet, la méthode *one-up-one-back* qui était utilisée précédemment requiert que pour connaître la provenance d'une mangue, les employés de Walmart devaient contacter leurs fournisseurs (que ce soit par courriel ou par appel téléphonique) qui devaient ensuite contacter leurs propres fournisseurs et ainsi de suite[Hyperledger, 2019b]. Cette méthode est très inefficace et mène à des problèmes de performance, ralentissant grandement le processus de traçage. Avec le nouveau système, trouver la provenance d'un produit requiert seulement une requête sur le réseau *Hyperledger Fabric*.

L'étude de cas du site de *Hyperledger* indique aussi qu'à la suite du succès du projet, en 2017, Walmart et IBM sont entrés en partenariat avec Nestlé et Unilever afin de créer le projet IBM Food Trust et, depuis septembre 2018, utilisent leur système basé sur *Hyperledger Fabric* dans la traçabilité de plus de 25 de leurs produits[Hyperledger, 2019b]. Aujourd'hui, IBM Food Trust[2] offre une multitude de services de traçabilité comportant des systèmes de gestion d'identité, des systèmes de

[2]IBM Food Trust est basé sur IBM *blockchain* Platform, un service *blockchain* offert par IBM. Ce service est basé sur le *framework Hyperledger Fabric* et comporte plusieurs fonctionnalités supplémentaires. Puisque ce service est basé sur *Hyperledger Fabric*, les nœuds exécutant IBM *blockchain* Platform peuvent interagir avec des nœuds exécutant les services *Hyperledger Fabric*.

gestion de données, ainsi que des APIs d'entrée et de visualisation de données[IBM, 2019].

En septembre 2018, une éclosion d'E. coli a été détectée aux États-Unis dans la laitue romaine provenant de la région de Yuma en Arizona. 210 cas d'infection ont été annoncés par la CDC[CDC, 2018] ainsi que dans un avis de l'Agence de la santé publique du Canada[de la santé publique du Canada, 2018]. Suivant cette éclosion, Walmart a annoncé qu'il allait exiger à tous ses partenaires fournisseurs de légumes feuillus à s'intégrer à leur réseau *blockchain* de traçabilité citant, notamment, la nécessité de mettre à jour les méthodes de traçabilité qui étaient souvent basées sur les traces papier des fournisseurs[Matt Smith, 2018]. En utilisant le registre partagé de la *blockchain*, Walmart espère grandement faciliter le processus de rappel en identifiant rapidement la provenance de produits contaminés[Matt Smith, 2018], citant Robert Tauxe, *MD, director of CDC's Division of Foodborne, Waterborne, and Environmental Diseases*.

Walmart a accompagné cette annonce avec une lettre aux fournisseurs leur exigeant de s'intégrer au réseau *blockchain*, citant «*[...] health officials and industry professionals were unable to quickly determine which lots were affected and which were not. This resulted in millions of bags and heads of romaine lettuce having to be removed from the market place and a loss of consumer confidence in romaine lettuce*». La lettre se conclut en exigeant que tous les fournisseurs directs de légumes feuillus soient membres du réseau IBM Food Trust en début 2019 et que les fournisseurs plus en amont dans la chaîne d'approvisionnement soient membres du réseau en fin 2019 afin de permettre la traçabilité bout à bout de leurs produits[Redfield *et al.*, 2018].

Ceci est un exemple de l'idée que la seule manière d'implémenter la *blockchain* dans le secteur industriel pour la traçabilité bout à bout est par l'entremise de pressions d'une grosse entreprise. En effet, en raison de la nature collaborative de la *blockchain* ainsi que les arguments énoncés plus tôt, il est beaucoup plus difficile pour une petite entreprise d'implémenter une *blockchain* de traçabilité. L'utilisation de pressions,

comme dans le cas de Walmart, assure qu'un maximum de participants de la chaîne d'approvisionnement participent au réseau, assurant une couverture plus complète de la chaîne. Une participation optionnelle pourrait conduire le réseau à un manque de participation et un manque d'informations cruciales à la traçabilité de produit (même si une traçabilité informatique partielle de la chaîne d'approvisionnement serait meilleure qu'aucune traçabilité informatique complète).

2.4 Conflits de factures de transport

Un autre cas de grand intérêt en *blockchain* est la logistique. Le domaine du transport de marchandises est intrinsèque à un monde globalisé. Dans ce domaine, il existe de nombreuses asymétries d'informations où les transporteurs et les clients possèdent différentes données pertinentes aux transports, menant à de nombreuses disputes de facturation.

Dans l'étude de cas de la Fondation *Hyperledger* de l'implémentation de *blockchain* dans le partenariat entre Walmart et DLT Labs™, Walmart estime qu'environ 70% des factures de transport de marchandises résultaient en dispute, étant donné des frais variables[Hyperledger, 2020b]. L'étude de cas déclare que les disputes entraînent des «coûts administratifs élevés et de longs délais», ajoutant que «la réconciliation de factures [se fait] souvent lorsqu'un des partenaires capitule»[traduction libre]. L'étude de cas rapporte aussi que «les coûts administratifs représentent 20% des frais de transport»[traduction libre]. L'étude de cas ajoute aussi qu'il y a environ 220 points de données dans chaque transport. Ces données pourraient être utilisées dans de nombreux cas tels que la résolution de dispute et l'optimisation du transport tel que dans ce cas de logistique, ou bien dans la traçabilité et la gestion de produits périssables tel que mentionné ci-tôt.

Dans ce cas, l'étude explique que les factures de transport sont créées conjointement avec le projet DL Freight™. Ceci permet aux entreprises comme Walmart et leurs transporteurs de travailler à partir de la même base d'information. En travaillant avec les mêmes informations, les participants d'une entente peuvent programmer la facturation selon des règles équitables. Puisque les deux groupes ont collaboré sur l'entente et font confiance au système, les disputes sont grandement limitées. En effet, l'étude de cas de *Hyperledger* rapporte que le projet a réduit le nombre de disputes à environ 1.5%. Un article plus récent (publié en janvier 2022 dans le Harvard Business Review) déclare que le nombre de disputes a été réduit à moins de 1% des factures[Vitasek *et al.*, 2022]. De plus, l'étude de cas indique que «puisque tout le monde a les mêmes informations, il n'y a plus de raison pour les disputes»[traduction libre], citant Francis Lalonde, VP des transports chez Walmart. Ainsi, même s'il y a une dispute, la résolution est beaucoup plus facile lorsque les participants collaborent à travers le processus entier. L'étude de cas ajoute aussi que le processus d'entente sur une facture est passé d'une durée d'un à plusieurs mois à une durée d'une seule semaine, facilitant grandement le transfert de capitaux rapide et créant un meilleur rapport entre les entreprises.

Afin d'optimiser la performance du réseau *Hyperledger*, DLT Labs™ a fait des modifications au code source de *Hyperledger Fabric*, comme mentionné dans l'étude de cas. La nature *open source* du projet rend cela possible, permettant à tout utilisateur de visualiser, étudier, auditer et modifier le code source afin de l'adapter à ses besoins[Hyperledger, 2020b, Mary Lacity, 2021]. Ceci peut être un grand avantage pour les développeurs avec une connaissance avancée de la technologie.

L'article *Requiem for reconciliations : DL Freight, a blockchain-enabled solution by Walmart Canada and DLT Labs*, écrit par Mary Lacity et Remko Van Hoek [Mary Lacity, 2021], met de l'avant l'idée que «les projets pilotes [(tels que l'implémentation initiale de DL Freight™ ou IBM Food Trust)] sont supérieurs aux preuves de concept si l'objectif est le déploiement»[traduction libre], spécifiant que Wal-

mart faisait déjà confiance à l'écosystème de *Hyperledger Fabric* afin de répondre à leurs besoins. En effet, étant donné la nature additive de la technologie *blockchain* (plutôt qu'une nature soustractive), les solutions *blockchain* ont le bénéfice d'avoir une transition de pilote à production très simple. Puisque la *blockchain* est une couche supplémentaire par-dessus les logiciels existants, l'implémentation de l'élément *blockchain* est souvent une question de connexion au réseau et un envoi de données. Puisque les traitements dans les contrats intelligents de la *blockchain* sont en principe seulement des processus de vérification du respect de règles de traitement prédéfinies, il n'est pas nécessaire de modifier la logique interne du logiciel préexistant, il faut seulement ajouter la connexion mentionnée plus tôt. En somme, connaissant la fiabilité de la plateforme *Hyperledger Fabric* dans de telles situations déjà étudiées et sachant que l'implémentation de technologie *blockchain* impacte peu le code de logiciels existants, il est évident qu'il est préférable de commencer par un projet pilote plutôt que de passer par une phase de preuve de concept (cette preuve ayant déjà été faite).

Chapitre 3

La blockchain

La *blockchain* est une technologie simple en concept, mais complexe dans sa mise en œuvre. Son utilisation comporte plusieurs choix importants permettant d'optimiser la correspondance de la solution avec la problématique à résoudre. À cette fin, avant d'utiliser la technologie *blockchain*, il est essentiel, dans un premier temps, de pouvoir définir les attributs d'une blockchain. Ces attributs sont ce qui différencie la *blockchain* d'une autre solution de réseautique et d'entreposage de données. Puisque la *blockchain* est une solution distribuée (et souvent décentralisée), elle doit aussi répondre à des besoins d'uniformisation de données, d'où les différentes méthodes de consensus qui peuvent être adaptées selon les besoins des utilisateurs. Il est ensuite nécessaire de comprendre les désavantages et limitations de la technologie *blockchain*. Enfin, une fois que la technologie a été bien définie, il est nécessaire de comparer les solutions traditionnelles (ne faisant pas utilisation de la technologie *blockchain*) aux solutions *blockchain* (ainsi que de comparer les technologies entre elles).

Ce chapitre est donc séparé en cinq sections afin d'assurer une base compréhensive des concepts en *blockchain* : les attributs de la *blockchain*, les méthodes de consensus, les limitations de la *blockchain*, les solutions traditionnelles et les solutions existantes.

3.1 Attributs de la blockchain

Toute *blockchain* complète comporte trois attributs principaux essentiels pour assurer la fiabilité de ses informations : la distributivité, l'immuabilité et la méthode de consensus. En plus de ces attributs essentiels, plusieurs *blockchains* populaires aujourd'hui ont étendus la définition de la *blockchain* afin d'inclure les contrats intelligents, permettant de définir une couche logique par-dessus la couche de stockage de données de la *blockchain*.

Dans cette section, il sera donc question des trois attributs essentiels de la *blockchain* ainsi que les contrats intelligents.

3.1.1 La distributivité

Toute *blockchain* comporte des éléments de distributivité ; soit la séparation physique et virtuelle de ses composantes. La *blockchain* repose sur un réseau de nœuds en communication P2P permettant d'avoir un système totalement distribué. *Bitcoin*, le premier réseau *blockchain*, repose sur un réseau décentralisé et distribué. Ce réseau, comme plusieurs autres réseaux *blockchain* publics tels qu'*Ethereum*, n'a pas d'autorité qui pourrait centraliser le contrôle de l'information et atteindre à sa sécurité en réduisant la surface d'attaque[1]. Cette distribution des ressources permet au réseau d'être résistant aux pannes. Si un nœud ou un groupe de nœuds tombent en panne, les autres nœuds comportent assez d'information pour maintenir le fonctionnement adéquat du réseau. Dans un cas extrême, un seul nœud serait suffisant pour préserver l'existence du réseau en attendant que les autres nœuds se remettent en marche. Cette distributivité est aussi utilisée comme argument pour la souveraineté de l'information

[1]En augmentant la surface d'attaque, il devient plus difficile d'atteindre à la sécurité du réseau entier puisque les autres nœuds peuvent rejeter tout changement non accepté.

et sa résistance à la censure. Si une suite de nœuds dans une région est éliminée à cause d'une réglementation d'une autorité, le fonctionnement du réseau ainsi que ses fonctionnalités utilisées dans la région affectée ne seront pas affectés. Cependant, cette distribution de l'information induit aussi certaines considérations supplémentaires à prendre en compte avant l'implémentation d'une solution *blockchain* qui seront discutées dans une section suivante.

3.1.2 L'immuabilité

La *blockchain* emploie un mécanisme de hachage ainsi qu'un chaînage de données afin d'assurer leur immuabilité. Dans la *blockchain*, les informations sont enregistrées sous forme de transactions individuelles ; contrairement à une solution de base de données traditionnelle, où seul l'état le plus récent est enregistré. De cette manière, on peut, à tout moment, recalculer l'état le plus récent d'un actif en retraçant la liste de toutes les transactions qui ont changé son état. Afin d'assurer qu'elles soient immuables, chaque transaction (incluant ses métadonnées) est ajoutée dans un bloc. Ce bloc possède le résultat du hachage de toutes ses données (incluant les transactions). Toute modification à ce bloc serait donc signalée par un autre nœud possédant les données au moment de la validation. En effet, une modification de bloc entraîne nécessairement une modification de son *hash*, et donc un conflit avec l'information qui a déjà été enregistrée. Afin d'assurer qu'il est impossible de propager des versions de la *blockchain* qui comportent des blocs falsifiés (puisqu'il est impossible de déterminer la vérité d'une information), chaque bloc possède aussi comme donnée le *hash* du bloc précédent. Une modification à un bloc antérieur entraîne donc l'invalidation des blocs suivants puisque les *hashs* ne correspondent plus à la réalité présentée.

Ce processus assure donc aux utilisateurs que l'information présentée n'a pas été modifiée. Ceci permet donc aux utilisateurs de faire confiance à la fiabilité du réseau, permettant d'auditer l'information du système afin d'identifier des incohérences. Dans

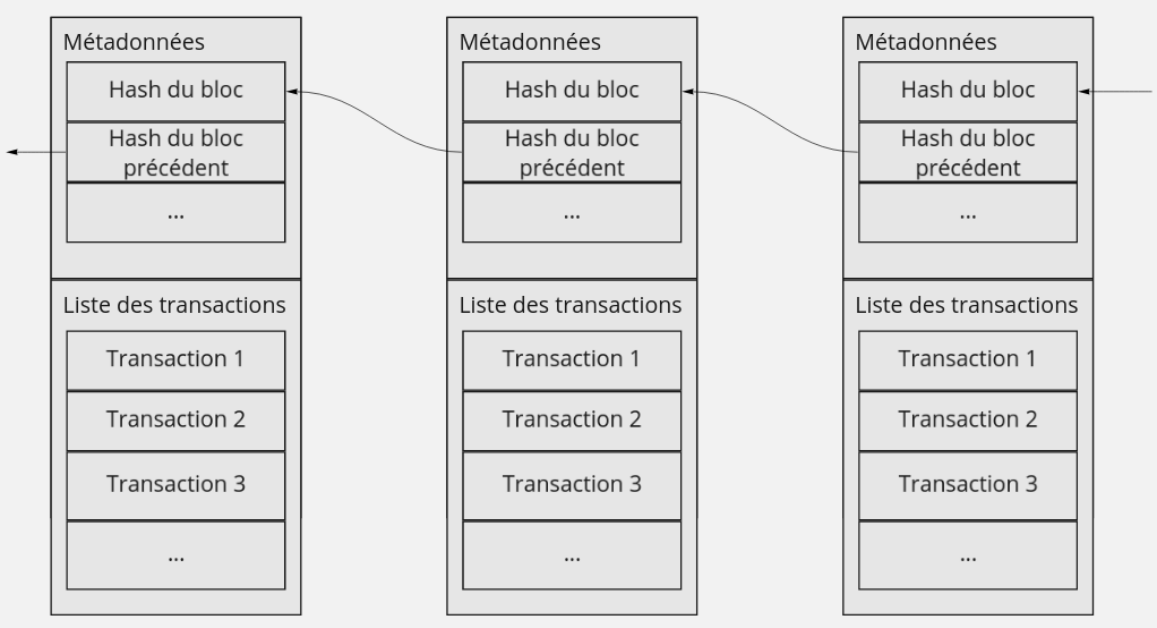

FIGURE 3.1 : Illustration d'une *blockchain*. Chaque bloc possède des métadonnées (incluant le *hash*, le *hash* du bloc précédent et toute autre donnée intéressante telle que le *nonce* et le *timestamp*) et une liste des transactions.

le monde de la cryptomonnaie, ceci permet entre autres d'identifier des cas de fraude et de blanchiment d'argent.

3.1.3 Le consensus

Le système de consensus est ce qui permet au réseau de surmonter le problème des généraux byzantins[Academy, 2018, Cointelegraph, 2021, Himanshi, 2022]. Puisqu'il y a plusieurs nœuds qui participent de manière décentralisée, il est essentiel de déterminer un mécanisme permettant de s'assurer que tous les nœuds sont synchronisés. Sans système de consensus, il est impossible de différencier les blocs ou transactions valides de ceux qui sont invalides. Il est impossible de préserver la cohésion du réseau. Afin d'assurer la synchronisation des nœuds, chaque *blockchain* implémente donc un système permettant de sélectionner le prochain bloc à travers le réseau.

Ce système, dans les *blockchains* publics décentralisés, prend la forme d'une *compétition*. Dans cette compétition, le *gagnant* est celui qui crée le prochain bloc qui sera ajouté sur la *blockchain*. Une fois que le bloc est créé, le nœud créateur ajoute ce bloc à son registre local. Celui-ci propage ensuite son bloc à travers le réseau par ses nœuds associés avec le protocole P2P. Il est possible que plusieurs blocs soient créés à peu près en même temps. Dans ce cas, un *fork* se crée ; soit de deux versions de la *blockchain* différentes. Les nœuds sont configurés pour gérer ce problème. Le réseau *blockchain* fait confiance à la version avec le plus de confirmations, soit la *blockchain* la plus longue. Les nœuds peuvent donc préserver deux versions différentes du registre et travailler sur une *version principale*. Lorsqu'une version du registre dépasse l'autre de manière significative, les nœuds choisissent automatiquement le registre avec le plus de blocs, selon leurs configurations. De cette manière, les nœuds prennent indépendamment la même décision. Si un groupe de nœuds minoritaire ne suit pas cette tendance, il n'est toujours pas en mesure d'affecter la *blockchain* principale, soit la *blockchain* sur laquelle le réseau travaille et entrepose ses transactions acceptées. Ceci permet donc d'assurer la sécurité du réseau tant que la majorité des nœuds ne collaborent pas contre la sécurité du réseau *de la même manière*. Bien sûr, cette compétition ne s'applique pas à des solutions dans les domaines privés. Dans de tels domaines, les participants du réseau font confiance aux autres participants pour préserver l'intégrité de la *blockchain*. Les *blockchains* privées, telles que *Hyperledger Fabric*, utilisent un système d'identités permettant d'opérer une méthode de consensus avec seulement des nœuds préapprouvés. Ceci permet d'éliminer la compétition et la remplacer par un vote simple.

3.1.4 Les contrats intelligents

Afin de pouvoir manipuler les données du registre, les *blockchains* peuvent posséder des structures appelées contrats intelligents comme mentionnés dans le chapitre 2. Sur la *blockchain Bitcoin*, les données sont manipulées via le *Bitcoin Script* qui permet des

opérations simples. Étant donné des limitations dans le *Bitcoin Script*, les instructions créées ne sont pas considérées comme étant des contrats intelligents. Cependant, elles ont formé les bases sur lesquelles les contrats intelligents modernes ont été créés.

Les contrats intelligents sont des entités persistant sur la *blockchain* permettant des manipulations de données complexes semblables à des classes que l'on pourrait voir dans d'autres langages de programmation. Sur la *blockchain Ethereum*, les contrats intelligents sont implémentés principalement dans le langage *Solidity*, un langage de programmation *Turing-complete*, permettant de manipuler directement des quantités d'*Ether* (la cryptomonnaie résidant sur la *blockchain Ethereum*) et des données propres au contrat. Ces contrats sont compilés avec un compilateur spécifique à ce nouveau langage. Avec *Solidity*, il est aussi possible d'interagir avec différents contrats afin de créer un écosystème complexe résidant sur la *blockchain Ethereum*.

Dans le cas de *blockchains* comme *Hyperledger Fabric*, les contrats intelligents peuvent être écrits avec des langages connus comme le JavaScript et compilés comme des fichiers de code source réguliers.

Bien sûr, il est possible de créer une *blockchain* sans contrats intelligents. Cependant, la création de contrats intelligents peut certainement être considérée comme un attribut principal ayant popularisé la *blockchain*, étant donné leurs capacités de manipulation d'informations sur la *blockchain* prenant avantage de ses attributs (distributivité, immuabilité et consensus) afin de créer des règles de manipulation d'information.

3.2 Les méthodes de consensus

Considérant sa nature décentralisée, tout réseau *blockchain* requiert une méthode de consensus. Le consensus est l'approche par laquelle tous les nœuds d'un réseau peuvent s'entendre sur une seule vérité. Comme mentionné précédemment, c'est par le consensus que les réseaux *blockchain* réussissent à combattre le problème des généraux byzantins[Academy, 2018, Cointelegraph, 2021, Himanshi, 2022]. La méthode de consensus de choix varie selon plusieurs besoins comme la sécurité, la confiance, l'espace de stockage et la consommation électrique du réseau. Dans cette section, il sera question de trois méthodes de consensus bien connus soit le *Proof of Work*, le *Proof of Steak* et le *Proof of Authority*.

3.2.1 Le Proof of Work

Le consensus *Proof of Work* (PoW) est la première méthode de consensus proposée pour la *blockchain* dans le *Whitepaper* de *Bitcoin* publié en 2008[Nakamoto, 2008]. Sans autorité centrale pour gouverner la vérité, le PoW se fie à ce que la majorité des nœuds soient honnêtes en utilisant un système de vote sur chaque bloc. Le PoW fonctionne en demandant à tous les nœuds *mineurs* de calculer un *hash* correspondant à certaines caractéristiques, augmentant la difficulté. Tous les nœuds mineurs compétitionnent pour calculer le *hash* du prochain bloc. Chaque bloc contient le *hash* du bloc précédent. Dû à la difficulté de ce calcul, une personne désirant contrôler l'information devrait contrôler plus de la moitié de la puissance de calcul du réseau afin de recalculer le *hash* de chacun des blocs et compétitionner contre le reste du réseau pour faire approuver ses propres blocs. Il est sous-entendu que le coût matériel d'implémentation d'une telle attaque dépasse les profits potentiels engendrés. Cependant, puisque cette méthode requiert une compétition de calcul entre tous les

participants, il y a de fortes préoccupations par rapport à l'impact environnemental du PoW sur une échelle globale, entraînant des recherches pour une méthode de consensus moins énergivore.

3.2.2 Le Proof of Stake

Le *Proof of Stake* (PoS) est une méthode proposée comme étant une alternative au PoW. Le PoS remplace la computation (qui entraîne des coûts élevés en matériel) par une sorte de pari. Dans le PoS, les nœuds ne minent pas comme dans le PoW. Chaque nœud voulant participer a l'option de faire un pari à bas risque où il met une quantité d'argent de côté (soit le *stake*). Pour pouvoir participer, une certaine quantité d'argent doit être mise dans le pot. Le réseau effectue ensuite un tirage pour déterminer quel nœud aura la permission de créer le prochain bloc. Les utilisateurs peuvent ajouter plus d'argent dans le pot pour augmenter leurs chances de créer le prochain bloc. Les *blockchains* ont aussi parfois des mécanismes permettant d'augmenter la probabilité de sélection d'un nœud qui n'a pas été sélectionné récemment. Ceci permet d'éviter de toujours sélectionner les mêmes nœuds ou groupes de nœuds qui ont plus de ressources à leur disposition. Cette méthode permet aussi de mieux décentraliser le réseau. Puisque le PoS utilise des sommes d'argent pour la sélection d'un créateur de bloc plutôt qu'une compétition computationnelle, cette méthode de consensus est vue comme étant une alternative plus viable sur le plan environnemental que le PoW. Entre autres, *Ethereum* a implémenté le consensus PoS en septembre 2022 sur le réseau entier dans la mise à jour appelée *The Merge*[Foundation, 2023].

3.2.3 Le Proof of Authority

Le *Proof of Authority* (PoA), de son côté, est basé sur la fiabilité inhérente à certains nœuds sur la *blockchain*. Dans le PoA, seuls les nœuds désignés ont le droit de créer et ordonner les blocs. Cette méthode entraîne nécessairement une centralisation du contrôle. Cependant, cette centralisation est souvent désirée dans les réseaux industriels puisqu'il est nécessaire de pouvoir contrôler l'information. Dans le PoA, il y a des nœuds qui proposent les transactions et des nœuds qui les organisent en blocs. Seules les transactions avec les signatures permises sont acceptées par les nœuds qui organisent les blocs. Ceci permet un contrôle inhérent des permissions sur le réseau, qui est nécessaire lorsque les utilisateurs ont des rôles et des accès différents. Cette organisation des permissions permet donc de contrôler les accès au niveau du réseau plutôt que dans les contrats intelligents directement. Ces deux méthodes de contrôle d'accès seront vues plus en détail dans le chapitre 4.

3.3 Les limitations de la blockchain

La *blockchain* comporte plusieurs limitations et faiblesses, que ce soit au niveau de la croissance du réseau *blockchain*, de la visibilité de l'information sur le réseau, ou encore du contrôle de l'information sur le réseau. La sévérité de ces limitations varie selon plusieurs facteurs, dont la nature publique ou privée du réseau, le nombre de participants et la nature des informations partagées sur le réseau. Cette section discute de trois limitations principales de la *blockchain* soit la croissance du réseau, la visibilité de l'information et le contrôle de l'information.

3.3.1 La croissance du réseau

Une des premières difficultés visibles en *blockchain* est liée au potentiel de croissance du réseau. En effet, les systèmes *blockchain* sont susceptibles à des difficultés variées lorsqu'ils étendent leurs capacités, dus à leur nature P2P combinée à la nécessité de préserver le consensus, en plus de la redondance d'information apportée par ces deux aspects.

Afin de comprendre les limitations induites par la croissance du réseau, il faut définir ce qu'est la croissance du réseau en *blockchain*. Effectivement, la définition de croissance en *blockchain* est étendue afin d'inclure non seulement la croissance physique du réseau (nombre et répartition des nœuds), mais aussi la croissance de la capacité technologique (croissance de la *blockchain*[2], la possibilité d'augmenter la capacité des blocs/transactions, ainsi que les améliorations permettant d'augmenter la vitesse de traitement de transaction). En somme, la croissance du réseau englobe des principes d'augmentation d'étendue et de capacité.

Le protocole utilisé pour l'implémentation de la *blockchain* affecte directement le potentiel de croissance d'un réseau. Par exemple, un réseau basé sur la méthode de consensus PoW sera beaucoup plus affecté par une augmentation du nombre de nœuds qu'un réseau basé sur le PoS, dû à l'impact de la puissance de calcul sur la vitesse du réseau ainsi que les temps de communication et la collaboration du réseau total avec l'approche PoW. Ces difficultés sont présentes à la fois dans les systèmes publics et dans les systèmes privés ou permissionnés, mais affectent le réseau à différents niveaux étant donné les différences architecturales entre les deux types de systèmes. Étant donné la confiance implicite envers la gestion d'identité des systèmes privés et permissionnés, ces systèmes peuvent optimiser leurs protocoles afin de mieux servir leurs besoins sans devoir se soucier des difficultés rencontrées par les méthodes de

[2]Par le concept de *blockchain*, on veut dire la structure de données et l'information qu'elle contient.

consensus qualifiées comme étant sans confiance telles que le PoW (qui réduit les performances et augmente les besoins matériels).

La revue de littérature *Systematic Literature Review of Challenges in* blockchain *Scalability* a trouvé que parmi les 121 documents étudiés, les facteurs affectant le potentiel de croissance d'un réseau *blockchain* public les plus communs étaient le débit de transactions, la méthode de consensus (qui affecte le fonctionnement de la *blockchain* à un niveau fondamental), l'énergie computationnelle et l'espace de stockage nécessaire[Khan *et al.*, 2021].

Les facteurs comme l'énergie computationnelle sont fortement liés à la méthode de consensus et peuvent donc être réduits par une méthode de consensus demandant moins d'énergie (tel que le PoS). Comme mentionné plus tôt, les systèmes privés ou permissionnés évitent une grande part de difficultés au niveau computationnel puisque la méthode de consensus n'est pas dépendante de la computation. Une difficulté comme le débit de transaction peut poser un plus grand problème dans un réseau privé dû à un grand nombre de transactions et au besoin de logiciels réactifs. La solution peut être d'investir dans de meilleures ressources physiques ou bien d'adapter le protocole aux besoins comme dans le cas de DLT Labs™ et Walmart[Hyperledger, 2020b, Mary Lacity, 2021]. Le débit de transactions est un problème plus complexe dans les systèmes publics puisque les transactions ajoutées aux blocs sont choisies par les nœuds du réseau (qui ne travaillent pas ensemble). Ceci peut porter les nœuds à prioriser certaines transactions plutôt que d'autres (telles que les transactions qui payent des frais plus élevés) afin d'obtenir une meilleure récompense, priorisant l'intérêt individuel des nœuds au détriment de l'intérêt des utilisateurs. Ceci mène à un ralentissement de l'inclusion de transactions dans la *blockchain* (ainsi que des facteurs tels que la taille de blocs et la fréquence de création de blocs). Une croissance physique du réseau peut aussi mener à un ralentissement puisque les nœuds doivent relayer l'information sur une plus grande surface, parmi plus de machines. Finalement, l'espace de stockage est une difficulté qui peut être

partagée entre les réseaux publics et privés. Alors que les réseaux privés peuvent s'organiser afin de réduire les besoins en espace (en étant sélectifs par rapport aux informations stockées sur la *blockchain*), les réseaux publics n'ont aucun contrôle sur le type ni la taille des informations ajoutées (les protocoles doivent donc imposer des limites strictes sur les données afin d'assurer le fonctionnement optimal du réseau). Par ailleurs, les systèmes privés ont la possibilité d'investir plus de ressources matérielles afin d'augmenter les performances du réseau. Du côté des systèmes publics, les besoins en espace de stockage et les besoins en bande passante sont directement transmis aux opérateurs de nœuds qui ne sont pas directement affiliés à l'organisation du réseau[3].

En somme, la croissance du réseau comporte de nombreuses difficultés à tous ses niveaux avec des solutions qui varient grandement selon l'architecture choisie et l'intention de l'utilisation de la technologie *blockchain*. Il est donc essentiel de prendre en considération les différentes forces et faiblesses d'un réseau *blockchain* avant de s'en servir, en portant une attention particulière aux facteurs limitants de son architecture puisque la croissance éventuelle d'un réseau demeure toujours possible sur un réseau basé sur la collaboration.

3.3.2 La visibilité de l'information

La seconde difficulté affectant la *blockchain* concerne la visibilité de l'information. En effet, dans leur état brute, les réseaux *blockchain* souffrent d'un problème de visibilité de l'information, en particulier dans les réseaux publics. Tel que mentionné plus tôt, tout réseau *blockchain* fonctionne avec un protocole de communication P2P. En raison de ce protocole, tout nœud est au courant de toute transaction, ainsi que les

[3]Il existe différentes solutions à ces problèmes, telles que les mises à jour de protocole réduisant la taille de blocs ou des recommandations d'espace de stockage pour les clients de nœuds. Certaines *blockchains* optent aussi pour des solutions de *sharding*, séparant la *blockchain* en plusieurs plus petites chaînes gérées par des sous-réseaux afin de réduire les besoins[Buterin, 2021]. Des solutions hors réseau existent aussi telles que les solutions *layer 2* qui permettent de réduire les transactions qui se font sur le réseau[Foundation, 2022].

données qu'elle contient donnant accès à toute information non cryptée à l'entité à qui appartient le nœud. Ceci apporte un problème très évident ; soit qu'aucune information envoyée au réseau n'est privée. Il est possible de visualiser ce problème avec un des nombreux outils de visualisation de *blockchain* tels que *Explorer* de *Blockchain.com*[Blockchain.com, 2018] ou l'outil *EtherScan*[Etherscan, 2015] qui permettent de visualiser les données de n'importe quelle transaction sur la *blockchain* depuis sa création. Bien sûr, ces fonctionnalités sont essentielles pour de transparence. Dans des cas de vérification de comptes, cet aspect est primordial et permet de retracer toutes les transactions d'un organisme sans aucune possibilité de modification après leur ajout. Plusieurs entités peuvent très bien fonctionner avec cette transparence, telles que les créateurs de contrats intelligents de loterie de cryptomonnaie, qui permet d'assurer les utilisateurs de la fiabilité de leur projet en leur donnant une visibilité sur la distribution des fonds quasiment en temps réel. D'un autre côté, dans les cas plus privés (tels que dans des cas de gestion de logistique ou de chaîne d'approvisionnement), alors qu'il est intéressant de partager certaines informations aux autres organismes d'un réseau, il y a plusieurs éléments de logique métier qui ne doivent pas être visibles aux compétiteurs. Ceci s'applique spécifiquement à une analyse en surface.

Plusieurs stratégies s'offrent aux organismes désirant utiliser la *blockchain* sans nécessairement révéler des informations privées. Entre autres, puisque même les contrats intelligents sont publics sur une *blockchain* publique, il est nécessaire de protéger la logique métier de certains contrats intelligents. Une telle méthode est tangente à la sécurité par obscurcissement ; soit l'abstraction de la couche logique de la couche de persistance de données. Cette technique est aussi recommandée dans le développement régulier d'applications décentralisées[4] sur les plateformes publiques telles qu'*Ethereum*

[4]Une application décentralisée est une application s'exécutant sur un réseau décentralisé. L'interface utilisateur, par exemple un site web, peut être hébergée sur un système centralisé. Cette interface fait des appels au réseau décentralisé afin d'exécuter ses fonctions. Le programme sur le réseau décentralisé n'est pas lié directement à l'interface et des individus non associés aux créateurs du programme peuvent créer des interfaces ou même des programmes décentralisés pouvant interagir avec ce programme.

puisque ces plateformes ont des frais de création de contrats dépendamment de la taille du contrat ainsi que des frais liés au stockage d'information à l'exécution. Donc, plus la fonction d'un contrat intelligent est complexe, plus elle coûtera cher à l'exécution de cette fonction. Il est donc avantageux de restreindre les fonctions de contrats intelligents seulement à ce qui nécessite l'utilisation de fonctionnalités de la *blockchain* à la fois pour cacher la logique métier et pour réduire les coûts en frais de déploiement de contrats intelligents. Une autre stratégie possible pour protéger les informations privées est le cryptage de données. Cette stratégie est implémentée afin de cacher les informations plutôt que la logique métier tout en se fiant à l'abstraction entre la logique métier et la persistance de données pour pouvoir insérer des informations totalement obscurcies dans les transactions. Ceci a comme avantage de permettre seulement aux partenaires ayant accès à la clé de visualiser les informations de l'entreprise[5]. Une autre méthode utilisée pour préserver l'intégrité de l'information sans la révéler publiquement est de seulement préserver une preuve de l'information sous forme de *hash*. La nature d'un *hash* est telle qu'il est impossible de retracer l'information originale à partir du *hash* résultant. En envoyant seulement le *hash* de chaque information à la *blockchain*, une entreprise peut stocker l'information en texte clair dans ses bases de données personnelles tout en préservant une preuve inchangeable sur la *blockchain*. De cette manière, l'information envoyée à la *blockchain* est toujours privée et dans le cas d'un audit, une entreprise peut toujours prouver que les informations n'ont pas été changées en recalculant le *hash* pour une série de données et en le comparant au *hash* sur la *blockchain*. Cette technique fonctionne aussi pour préserver des preuves d'informations qui sont trop volumineuses pour les entreposer sur la *blockchain*, telles que des documents ou médias comme des images ou des vidéos. Ces informations peuvent être entreposées dans une base de données déconnectée de la *blockchain*, ou être dans un système décentralisé tel qu'IPFS[IPFS, 2014].

Une solution existant dans le domaine privé pouvant résoudre la majorité de ces

[5]Considérant la nature permanente et publique de l'information, cryptée ou non, il est essentiel d'analyser le risque de décryptage non intentionnel des informations stockées sur la *blockchain*.

problèmes consiste à créer une *blockchain* privée ; soit une *blockchain* existant sur un réseau séparé du réseau public, permettant de contrôler tous les accès au système. Une *blockchain* privée peut être créée avec le même protocole qu'une *blockchain* publique existante (plusieurs *blockchains* privées utilisant une adaptation du protocole d'*Ethereum* existent déjà) ou un protocole créé spécifiquement pour les systèmes privés. Cette solution comporte de nombreuses différences qui seront discutées plus tard.

3.3.3 Le contrôle de l'information

La troisième grande difficulté des réseaux *blockchain* concerne le contrôle de l'information qui demeure une difficulté constante de ce domaine. Le contrôle de l'information est une faiblesse indirecte de la *blockchain*. Ici, il est question d'un aspect plus juridique qu'informatique du domaine de la *blockchain*, soit que si chaque nœud partage toutes ses transactions avec tous les autres nœuds, tous les membres de la *blockchain* possèdent toutes les informations de tous les autres participants. Ceci soulève un questionnement juridique important : «à qui appartient cette information ?» Dans le cas de gestion d'information, il est souvent essentiel de pouvoir prouver à qui appartient cette information afin, entre autres, d'assurer sa protection. Dans de tels cas, il pourrait être impossible d'utiliser un système *blockchain* où tous les membres sont égaux. Il faudrait donc assurer non seulement des permissions d'accès au niveau des utilisateurs, mais aussi des permissions d'accès au niveau des nœuds entreposant ces informations. Si la preuve seule est nécessaire pour les fins de l'entreprise, il est possible d'utiliser la stratégie discutée précédemment basée sur le hachage. Dans ce cas, aucune information pertinente n'est sur les nœuds appartenant à une autre entité sur le réseau, protégeant ainsi la souveraineté de l'information. Une autre solution plus directe conceptuellement (mais demandant une architecture plus complexe) est de séparer les informations propres aux entreprises. Une telle architecture nécessiterait l'utilisation de plusieurs registres *blockchain* qui ne communiquent pas nécessairement ensemble. Chaque nœud

peut donc seulement gérer les informations qui lui sont propres et demander les informations supplémentaires aux nœuds des partenaires.

3.4 Les solutions précédentes à la *blockchain*

La *blockchain* a été créée afin de répondre à des besoins de technologies préexistantes. Dans les cas d'utilisation de *blockchain*, il est commun que la *blockchain complémente* une solution existante.

Étant donné sa nature en tant que réseau de partage de données décentralisé, il est impossible de totalement découpler son aspect *réseau* de son aspect *stockage de données*. Un nœud singleton *blockchain* est simplement un entreposage de données. Cependant, la solution *blockchain* se rapproche beaucoup plus d'une réponse aux besoins de stockage de données que d'un réseau et est donc habituellement comparée aux solutions traditionnelles de stockage de données. On peut s'approcher encore plus d'une comparaison complète en incluant l'aspect de partage de données d'un système.

Il est commun de comparer les solutions *blockchains* aux solutions de base de données simples. Dans une situation dite normale, un seul organisme possède une architecture de base de données simple. Advenant une panne, les données peuvent potentiellement devenir totalement inaccessibles. De plus, la communication entre les organismes peut être très inefficace comme dans le cas de traçabilité de Walmart mentionné dans le chapitre 2 où les recherches de données étaient très lentes dû au manque de partage de données à travers la chaîne d'approvisionnement. Puisque la base de données est centralisée chez chaque organisme, il est nécessaire que cet organisme crée un outil (que ce soit un portail ou une API simple) pour y accéder à partir de l'extérieur advenant le cas où un partenaire doit avoir accès à ses données. Cette étape est cruciale pour contrôler les données qui peuvent être sensibles ou confidentielles.

Étant donné l'effort de développement d'une telle solution ainsi que les multiples relations entre les organismes[6], plusieurs organismes n'ont pas un tel outil pour leurs partenaires et optent pour une solution où la communication doit être acheminée aux employés, plutôt que par un accès direct à l'information.

Les besoins de collaboration dans un écosystème d'organismes sont la source principale de besoin de *blockchain* puisque cette technologie permet de partager des informations entre organismes en réservant le contrôle sur les données partagées, tout en gardant un registre immuable (empêchant la manipulation de données). Il peut donc être intéressant d'avoir une solution *blockchain* pour permettre la communication de données tout en ayant une solution de base de données centralisée pour les données qui ne sont pas sur le réseau.

3.5 Les solutions *blockchain*

La technologie *blockchain* a été implémentée par plusieurs groupes afin de répondre à différents besoins. Ces implémentations se différencient de plusieurs manières dont leur méthode de consensus, l'existence de cryptomonnaie et l'étendue des fonctionnalités de leurs contrats intelligents.

Cette section se concentrera sur trois architectures de *blockchain* connus, soit le réseau *Bitcoin* (l'origine de la technologie *blockchain*), l'architecture *Ethereum* (une architecture extensible avec plusieurs réseaux publics et la possibilité de créer des réseaux privés configurables) et *Hyperledger Fabric* (une architecture modulaire uti-

[6]Si l'organisme A a les partenaires B et C, les partenaires B et C doivent se conformer au système de l'organisme A. De son côté, puisqu'il n'existe aucun standard strict de développement de portail, l'organisme A doit se conformer à la fois au système B et au système C (les employés doivent donc utiliser plusieurs outils différents pour accomplir potentiellement les mêmes tâches). Ceci devient beaucoup plus complexe dans un monde globalisé où chaque organisme a des relations *1 à n* avec d'autres organismes et que ces organismes ont aussi des relations *1 à n*.

lisée dans des applications dans le domaine privé).

3.5.1 Bitcoin

La première *blockchain* publiquement disponible est le *Bitcoin*. Publié en 2008 [Nakamoto, 2008], le *Whitepaper Bitcoin* décrit des méthodes par lesquelles un réseau d'échange décentralisé pourrait être sécurisé par un registre maintenu par des nœuds publics. Le projet *Bitcoin*, étant décentralisé, existe aussi en tant que projet *open source*[Bitcoin, 2013] lui permettant donc d'exister de manière totalement indépendante de toute gouvernance centrale. Dans un tel système, la validité d'une transaction est décidée par le vote de la majorité du réseau par un consensus de PoW.

Le PoW défini dans le Whitepaper *Bitcoin* s'inspire du système PoW défini pour le système Hashcash qui est un système «publiquement vérifiable et probabiliste»[traduction libre][Back, 2002], soit un système permettant à tout utilisateur de vérifier la solution où la solution requiert nécessairement un nombre indéterminé d'essais. Dans un tel système PoW, chaque bloc possède un *hash* pouvant être calculé à partir du contenu du bloc. Afin de calculer un *hash* valide (soit un *hash* qui sera accepté par le réseau), le *hash* présenté doit débuter avec un nombre prédéterminé de zéros, appelé «collision partielle de *hash*»[traduction libre][Back, 2002]. Afin d'assurer la participation de nœuds mineurs (qui sont ceux qui maintiennent le registre et sont essentiels pour le fonctionnement des transactions), le réseau *Bitcoin* dépose une quantité de *Bitcoins* (BTC) dans le compte du mineur qui réussit à miner le bloc. La quantité de BTC reçue suit un protocole qui diminue la quantité de BTC reçu à une certaine fréquence. Ceci permet de contrôler la quantité maximale de BTC en circulation de manière totalement décentralisée. En diminuant la récompense de minage de moitié à chaque réduction[Wiki, 2013], le réseau *Bitcoin* assure une inflation contrôlée de sa cryptomonnaie.

Afin d'assurer que les nouveaux blocs soient suffisamment propagés à travers le réseau, le réseau *Bitcoin* vise un taux de création de blocs d'un bloc par 10 minutes (dû à la nature probabiliste du calcul de *hash* dans le système PoW, il est entièrement possible que certains blocs soient créés quasi instantanément ou qu'ils prennent beaucoup plus longtemps que 10 minutes, cependant la tendance est toujours estimée à environ 10 minutes par bloc). Lorsque 2016 blocs ont été minés, le réseau ajuste la difficulté du PoW requis pour la création de blocs[Wiki, 2013]. Si la moyenne de temps de création de blocs est trop basse, la difficulté est augmentée, si elle est trop élevée, la difficulté est diminuée. Éventuellement, le réseau ne distribuera plus de récompenses de minage. À ce moment, la motivation de minage sera exclusivement les coûts de transaction.

3.5.2 Ethereum

Ethereum est un réseau *blockchain* décentralisé qui a été lancé en 2015 [Foundation, 2015]. *Ethereum* représente l'évolution naturelle de l'écosystème du domaine de la *blockchain*. Alors que *Bitcoin* permet d'exécuter certaines validations et conditions sur des transactions par le *Bitcoin Script*, la *Ethereum Virtual Machine* (EVM) permet d'implémenter des fonctionnalités complexes avec les contrats intelligents.

Comme mentionné plus tôt, les contrats intelligents sur *Ethereum* sont écrits dans un langage *Turing-complete*[Buterin, 2014], permettant de créer des opérations complexes avec des boucles. Bien sûr, avec la possibilité d'insérer les boucles dans le code vient la possibilité d'injecter des opérations de longue durée (ou même à temps infini). Puisque la validation d'une transaction requiert l'exécution du contrat intelligent, tous les nœuds exécutent cette boucle, entraînant un ralentissement du réseau. Une boucle infinie aurait le potentiel de figer le réseau entier. Une solution implémentée dans *Ethereum* pour résoudre ce problème est l'utilisation de *Gas*.

Le *Gas* est un coût associé à toute opération sur la *blockchain Ethereum*. Lorsqu'un utilisateur veut exécuter une transaction, il doit fournir une quantité de *Gas* qui sera consommée par le réseau et distribuée comme récompense aux nœuds mineurs. Chaque opération d'écriture comporte un coût en *Gas* associé, augmentant donc le coût total d'une opération complexe ou d'une opération s'exécutant en boucle. Le réseau consomme du *Gas* à chaque opération exécutée. Si la transaction manque de *Gas* (l'opération coûte plus cher que ce qui a été fourni), la transaction est marquée comme étant incomplète et le *Gas* consommé n'est pas retourné à l'utilisateur et est plutôt redistribué aux mineurs. Il s'agit donc d'une solution simple à la nature non déterministe de la *blockchain* permettant d'éviter des opérations qui ne se terminent jamais. Ceci force un degré de déterminisme sur les transactions[7]. Le *Gas* permet aussi aux utilisateurs de se protéger contre des opérations malicieuses en limitant l'étendue d'une opération. En effet, puisqu'il est possible de passer des appels d'un contrat intelligent à l'autre, il existe plusieurs vecteurs d'attaque (tels que la réentrance[Consensys, 2022]) permettant d'exploiter le système. En fournissant une quantité limitée de *Gas* aux prochaines potentielles transactions, il est possible de forcer les appels non voulus à échouer et empêcher l'attaque de se compléter.

3.5.3 Hyperledger

Hyperledger Fabric, la *blockchain* utilisée dans ce projet, est une architecture *blockchain* modulaire permettant de créer un réseau *blockchain* privé. Elle a été créée dans sa version 1.0 en 2017[Hyperledger, 2017]. *Hyperledger Fabric*, contrairement à *Bitcoin* et *Ethereum*, est conçu pour des solutions dans le secteur privé. Alors qu'il est possible de créer un réseau *Ethereum* séparé du réseau public, à sa base, tous les nœuds d'un réseau *Ethereum* sont égaux. De son côté, *Hyperledger Fabric* permet de créer

[7]Une fonction au résultat non déterministe tel qu'un roulement de dé verra sa transaction rejetée par le réseau au point du consensus. Afin de permettre l'entrée de données non déterministes dans le réseau, des oracles sont utilisés. Ces oracles permettent de fournir une réponse déterministe au réseau pour des arguments identiques à partir d'une source externe qui est non déterministe.

des nœuds de différents types appartenant à différents organismes. Cette distinction permet d'assigner des identités et des rôles aux différents nœuds. De plus, *Hyperledger Fabric* permet aux architectes de créer des *channels*, soit des structures de données réservées aux utilisateurs ayant accès à celles-ci. Sur ces *channels* vit le *chaincode*, l'architecture de contrats intelligents de *Hyperledger Fabric* qui gère la manipulation et le traitement des données et qui permet aux applications d'interagir avec l'information du réseau.

Les nœuds

Hyperledger Fabric possède plusieurs types de nœuds afin de permettre la création et la gestion d'un consortium. Puisqu'il s'agit d'une solution *blockchain* privée, il existe une confiance implicite entre les opérateurs de nœuds. De plus, étant donné l'immuabilité de la *blockchain*, il est toujours possible aux administrateurs de nœud de valider les informations présentes sur leurs nœuds et d'effectuer des corrections advenant le cas d'une insertion d'informations erronées (intentionnelles ou non). Afin d'assurer le bon fonctionnement du réseau, il existe deux principaux types de nœuds.

Le premier type de nœud est le nœud d'ordonnancement. Ces nœuds déterminent l'ordre dans lequel les transactions sont agencées dans la *blockchain*. Ce sont donc les nœuds qui participent au consensus de l'architecture *Hyperledger Fabric*, anciennement par le protocole Kafka, mais depuis la version 2 de *Hyperledger Fabric*, le protocole Raft est priorisé. Les nœuds d'ordonnancement sont aussi responsables de la gestion des accès aux *channels* et des droits d'accès d'administration du réseau[Hyperledger, 2020d].

Le deuxième type de nœud est appelé *peer*. Ce type de nœud est responsable des interactions avec les instances de *chaincode* et les instances de registres. Ces nœuds peuvent être configurés afin de posséder seulement du *chaincode*, seulement un re-

gistre, plusieurs *chaincodes*, ou plusieurs registres. Le *chaincode* permet de manipuler l'information et l'ajouter au registre alors que le registre permet d'enregistrer et de consulter l'information. Ces nœuds participent dans le processus de la validation des transactions. Cette validation de transactions est une forme de consensus, requérant que les autres nœuds acceptent le résultat de chaque transaction afin qu'elle soit ajoutée au registre via les nœuds d'ordonnancement. Ils sont aussi le point d'accès des applications au registre par le *chaincode* qui y est installé.

Il est important de considérer l'agencement de ces nœuds lors de la mise en production du réseau. Entre autres, il faut que les nœuds soient à des endroits géographiques variés afin d'éviter des pannes. Les nœuds d'ordonnancement en particulier devraient adhérer à cette règle puisqu'ils sont responsables du fonctionnement du réseau entier. La documentation de *Hyperledger Fabric* indique que les regroupements impairs de trois à neuf nœuds d'ordonnancement sont communs avec au plus 50 *channels* par nœud[Hyperledger, 2020e]. Ces types d'agencement sont favorisés puisque plus il y a de nœuds, plus fortes seront les probabilités d'apparition des difficulté de communication dû à l'augmentation du temps de propagation. Dans le cas des *channels*, plus il y a de *channels* associés à un nœud, plus celui-ci doit effectuer de traitements.

Les channels

Les *channels* sont une structure dans *Hyperledger Fabric* qui n'existe pas nativement dans les *blockchains* publiques comme *Ethereum*. Cette structure possède son propre registre distinct des autres *channels*. Les *channels* sont considérés comme étant des sous-réseaux[Hyperledger, 2020a] dans le réseau *Hyperledger Fabric* permettant d'isoler les transactions par regroupements d'organismes. Les *chaincodes* sont installés dans les *channels* et une application doit faire référence au *channel* afin d'appeler un contrat intelligent.

Afin d'accéder aux informations d'un *channel* (ainsi que pouvoir y insérer de l'information), un organisme doit y avoir les permissions adéquates qui lui ont été assignées. Les permissions d'un *channel* peuvent inclure des droits de lecture, d'écriture et d'administration. Ceci permet de créer des réseaux de partage d'informations aux accès limités afin de faciliter la communication entre partenaires sans partager des informations privées aux compétiteurs. Ces partenaires peuvent aussi exister sur le même réseau *Hyperledger Fabric* sans participer aux mêmes *channels*. Les *channels* permettent aussi de faciliter les audits d'informations en temps réel en donnant des droits de lecture aux auditeurs.

En plus de la structure en *channels*, il existe des données privées, qui sont ajoutées par l'entremise d'un contrat intelligent dans des structures appelées *collections*. Ces données privées ne sont pas partagées sur le *channel*, mais uniquement avec les nœuds qui ont la permission de les recevoir par le protocole *gossip*. Les données privées sont stockées séparément des données non privées et une trace de leur création existe dans le *channel* sous forme de *hash* permettant de prouver leur existence et leur immuabilité[8].

Le *chaincode*

Dans *Hyperledger Fabric*, le *chaincode* est ce que l'on appelle le rassemblement de contrats intelligents associés. Par exemple, un *chaincode* appelé Véhicule pourrait contenir les contrats intelligents appelés Bateau, Voiture et Camion, comme montré dans la documentation de *Hyperledger Fabric*[Hyperledger, 2020g]. Afin de permettre le traitement de logique métier, plusieurs *blockchains* telles que *Hyperledger Fabric* et *Ethereum* implémentent une logique de contrats intelligents. Dans ces cas, les contrats intelligents sont des structures permettant de faire un lien entre les utilisateurs et la *blockchain*. Avec les contrats intelligents, il est possible d'exécuter des traitements

[8] Afin de valider une information qui a été transmise par les données privées, il suffit de calculer son *hash* sur la *blockchain* (dans le *channel*). Si le *hash* calculé est différent de celui de sa transaction associée, il est évident que l'information n'est pas valide.

sur les données lors des opérations de lecture et d'écriture sur la *blockchain* selon des besoins prédéfinis.

Les contrats intelligents existent sur la *blockchain* en tant qu'instance. Dans le cas d'*Ethereum*, toute instance d'un contrat intelligent est permanente sur la *blockchain*. À cet effet, les développeurs sur la *blockchain Ethereum* doivent prendre de nombreuses précautions : les contrats intelligents doivent être très strictement évalués pour des failles afin d'éviter des mises à jour fréquentes (chères en *Gas* et avec plus de risques de failles) et doivent implémenter des patrons spécifiques afin de transférer le contrôle des accès à la plus récente version. Dans *Hyperledger Fabric*, les mises à jour de contrats intelligents *remplacent* l'instance précédente (suite au consensus). Cela permet de préserver la référence aux contrats intelligents et de simplifier le développement de contrats intelligents. Cela est dû à l'aspect privé de la *blockchain Hyperledger Fabric*, permettant de faire confiance aux différents organismes.

Les contrats intelligents dans *Hyperledger Fabric* sont écrits avec des langages de programmation communément utilisés et peuvent être vus comme des classes. Cela permet de limiter la courbe d'apprentissage aux outils de la plateforme, contrairement aux *blockchains* (comme *Ethereum*) qui requièrent l'utilisation d'un langage spécialisé comme *Solidity* (en constante évolution)[Ethereum, 2018]. Présentement, les contrats intelligents sur *Hyperledger Fabric* peuvent être développés en Go[9], en Java, en JavaScript ou en TypeScript[10] avec l'API fourni dans ce langage. Cette API permet des opérations de lecture et d'écriture complexes sur la *blockchain* telles que l'accès aux actifs par des clés composites, l'accès aux actifs par des sélecteurs avec CouchDB[11], ainsi que la lecture de l'historique d'un actif permettant de visualiser tous les changements d'état depuis la création de sa clé.

[9]https ://go.dev/ consulté en août 2022
[10]https ://www.typescriptlang.org/ consulté en août 2022
[11]https ://couchdb.apache.org/ consulté en août 2022

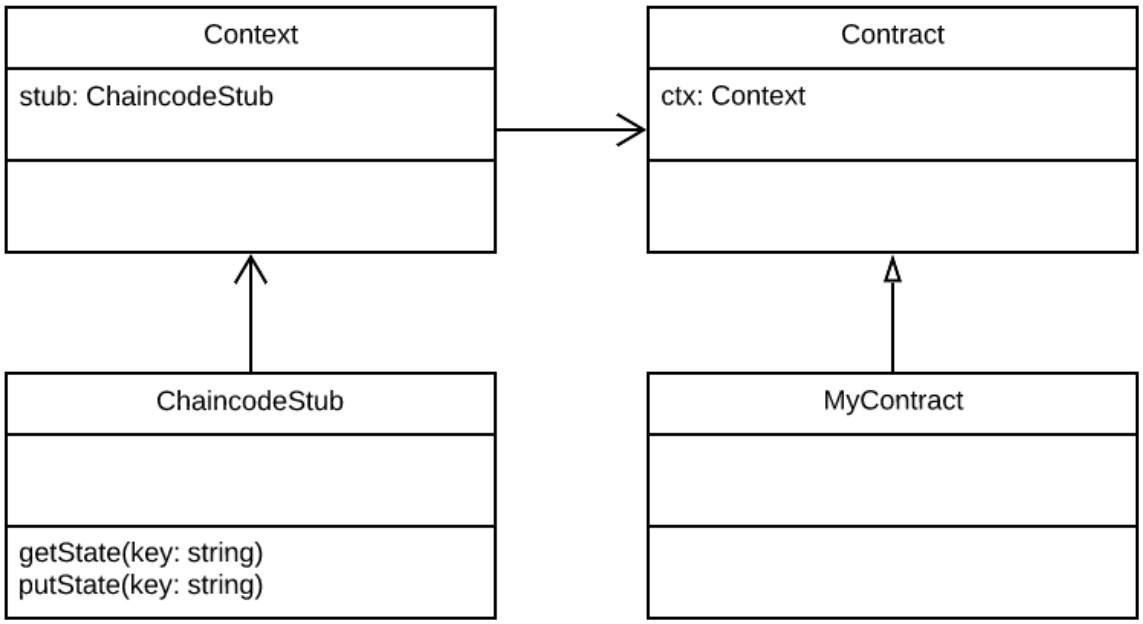

FIGURE 3.2 : Diagramme simplifié de la relation entre les classes Context et Contract de *Hyperledger Fabric*.

Le *chaincode* comporte plusieurs éléments dans sa création. Dans le code source, il y a un fichier index qui possède la liste de contrats intelligents implémentés dans le *chaincode*. Chaque contrat intelligent hérite d'une classe appelée Contract et possède un attribut dans la classe Context qui permet d'interagir avec le registre. Chaque fonction du contrat intelligent a le contexte comme premier argument, permettant d'exécuter les opérations sur le registre avec `ctx.stub` (relation dans la figure 3.2). Afin d'instancier le *chaincode* sur la *blockchain*, il faut suivre plusieurs étapes. En premier lieu, il faut installer le *chaincode* (compilé) sur les nœuds désirés. Ensuite, chaque organisme doit approuver la définition du *chaincode*. Une fois que le nombre requis d'approbations est atteint (la majorité des participants, par défaut), les organismes peuvent confirmer la définition du *chaincode* via une opération *commit*. Enfin, le *chaincode* peut être initialisé. Ce processus requiert la collaboration de tous les participants, ainsi qu'une planification initiale du réseau afin d'identifier les organismes nécessaires dans la gestion du *chaincode*.

En somme, le *chaincode* est une partie intégrante de l'écosystème de *Hyperledger Fabric* permettant une manipulation complexe des données avec un contrôle natif sur les accès de ses fonctions, le distinguant des implémentations de contrats intelligents des autres plateformes *blockchain*.

Les applications

Alors qu'il est possible d'exécuter les requêtes dans le terminal d'un nœud, *Hyperledger Fabric* comporte des SDK permettant d'exécuter les transactions à partir d'applications externes. Afin d'accéder au contrat intelligent résidant sur un nœud, l'application doit fournir un profil de connexion comportant les informations du nœud avec un *Gateway*. Le *Gateway* donne accès aux fonctions permettant d'exécuter les opérations de lecture et d'écriture dans le contrat intelligent désiré en utilisant le nom de la fonction et la liste de ses arguments. Cette méthode diffère de celle d'*Ethereum*, qui utilise un objet représentant le contrat intelligent (ABI) pour donner un accès à ses fonctions par le nom de la fonction[12].

Les applications jouent un rôle différent des contrats intelligents. Alors que les contrats intelligents sont responsables de la couche de données du côté du réseau, l'application est responsable du traitement initial de ces données du côté de l'utilisateur. Cette distinction est importante afin d'éviter de surcharger le réseau *blockchain* puisque tous les nœuds doivent exécuter les fonctions afin de valider la sortie. Les contrats intelligents sont donc responsables de la validation de l'information au niveau de la *blockchain* ainsi que la validation de l'identité. Les applications, de leur côté, devraient exécuter les traitements sur les données afin que les contrats aient seulement l'information qui les concerne.

[12]Dans *Ethereum*, l'ABI permet de créer un modèle du contrat intelligent ciblé dans l'application ; contrairement à *Hyperledger Fabric* où les appels se font par la méthode `evaluateTransaction` de l'API avec le nom de la fonction en argument avec les arguments de celle-ci.

Les applications sont aussi nécessaires dans le traitement de données non déterministes. Les contrats intelligents requièrent des informations déterministes afin de pouvoir atteindre le consensus. Si une information non déterministe est générée dans le contrat intelligent, chaque nœud obtiendra une information ayant un conflit avec les autres nœuds ; invalidant chaque transaction générée par cette fonction. Deux solutions possibles sont offertes aux développeurs : l'utilisation d'un oracle, soit un service permettant d'assurer la même sortie pour les mêmes données à l'entrée pour chaque transaction, et l'entrée de la donnée dans l'application. En offrant l'information non déterministe au contrat intelligent par l'entremise de l'application plutôt que de générer cette information dans le contrat intelligent, les développeurs peuvent être certains que tous les nœuds auront les mêmes informations et effectueront les mêmes traitements étant donné que tous les nœuds recevront la même donnée (l'application gère le non-déterminisme et non le contrat intelligent). Par exemple, si les actifs sont identifiés avec un identifiant unique déterminé au hasard (UUID[Hyperledger, 2020f, contributors, 2004]), il est recommandé de calculer cet identifiant dans l'application plutôt que dans le contrat intelligent où chaque nœud devra répéter l'opération individuellement.

La couche d'application *Hyperledger Fabric* peut être implémentée directement dans une application, ou elle peut être implémentée sur un serveur qui fournit des points d'accès avec une API. De cette manière, les points d'accès au réseau interne peuvent être limités et gérés beaucoup plus facilement.

La collaboration

En utilisant l'architecture privée configurable de *Hyperledger Fabric*, le réseau peut très bien faciliter la collaboration entre différents organismes. L'architecture permet de facilement inclure et exclure les acteurs comme désiré. La modularité de l'architecture en *channels* est excellente pour les solutions de traçabilité ainsi que tout réseau

s'étendant à travers de nombreux organismes. Cette architecture permet de partager des données seulement avec les partenaires impliqués dans les opérations qui les concernent. En *blockchain*, il est commun de discuter l'idée d'un partage de données *trustless*, qui pourrait être interprété comme sans confiance. Cependant, dans le cas de *Hyperledger Fabric*, l'aspect *trustless* implique une idée beaucoup plus englobante du manque de besoin de faire confiance.

Dans l'architecture *Hyperledger Fabric*, l'aspect de partage d'information se fait par invitation. Contrairement à un système traditionnel où chaque organisme doit faire confiance à un organisme qui préserve la base de données partagée, dans un réseau *Hyperledger Fabric*, chaque organisme est responsable du partage de ses propres informations. Plutôt qu'obliger chaque organisme à s'adapter à l'API de ses partenaires, les organismes ont tout simplement besoin de s'adapter à l'API de *Hyperledger Fabric* afin d'accéder aux données des *chaincodes* (cette structure est valide pour tous les partenaires dans un réseau *Hyperledger Fabric*). Les seules données nécessaires sont le nom du *channel* et l'information du profil de l'utilisateur appelant l'API. Ceci permet d'uniformiser les accès aux données et donc de faciliter le partage de ces données dans un consortium.

L'aspect de communication *inter-chaincode* est aussi excellent pour les cas de traçabilité. Dans de tels cas, on peut très bien faciliter la tâche de filtrage des données par leur origine et suivre tous leurs changements d'état. Une requête peut trouver toutes les données depuis la création d'un produit sans devoir communiquer avec les organismes intermédiaires en suivant des droits d'accès prédéterminés. Une telle fonctionnalité est excellente dans des cas tels que la traçabilité de Walmart discuté dans le chapitre 2.

La configuration du réseau

La configuration du réseau, telle que mentionné dans la section précédente, est faite de manière collaborative. Pour chaque changement apportée au réseau (ajout d'un organisme, ajout de *channel*, ajout de *chaincode*, etc.), celui-ci doit atteindre un consensus. Le consensus du réseau peut être adapté afin de requérir un seul organisme, la totalité des organismes, ou différentes combinaisons d'organismes en utilisant des expressions logiques avec les mots clés `AND` et `OR`. Ces expressions permettent d'identifier les organismes essentiels à l'implémentation d'une fonctionnalité et de s'assurer qu'il n'y a pas d'inégalité dans la gestion du réseau.

En plus de l'administration du réseau, il est essentiel d'isoler le réseau d'influences externes possibles. À cette fin, il peut être désirable que tous les nœuds du réseau se joignent à un VPN interne aux organismes, permettant de limiter les accès externes et donc limiter les cyberattaques possibles. Les accès de l'extérieur seraient donc contrôlés très strictement. Une méthode d'accès serait par un serveur web autorisé permettant de limiter les communications avec le réseau. Les utilisateurs auraient donc à utiliser une application gérée par le serveur pour pouvoir envoyer des transactions à la *blockchain*.

La gestion d'identités

Comme mentionné plus tôt, toute entité possède une identité dans un réseau *Hyperledger Fabric*. Chaque entité, que ce soit un utilisateur, un administrateur, ou un nœud, possède une identité associée à l'organisme dont il est membre. Afin de participer au réseau, une entité doit d'abord être enregistrée auprès du réseau par un administrateur.

Cette approche diffère du système de *blockchain* public qui permet à n'importe qui de se joindre au réseau avec une adresse de compte. Afin d'adhérer aux normes KYC[Bootcamps, 2021], une entreprise sur *Ethereum* doit pouvoir associer une identité à une adresse. Les contrats intelligents sur *Ethereum* peuvent, par exemple, préserver une liste de comptes avec leurs rôles[13][OpenZeppelin, 2021] et identités afin d'adhérer à ces normes et pouvoir conférer des permissions spécifiques.

Dans le cas de *Hyperledger Fabric*, l'appartenance aux organismes permet de définir les accès dès l'inscription d'une entité. De plus, les certificats dans *Hyperledger Fabric* permettent l'assignation d'attributs aux identités. Cette fonctionnalité est importante dans le contrôle des accès granulaire, permettant de limiter les accès à certains rôles et donner différentes combinaisons de rôles aux utilisateurs.

[13]Il existe des librairies dans *Solidity* permettant de gérer des rôles en utilisant des *modifiers* (un patron semblable au patron décorateur) comme la librairie développée par *OpenZeppelin*.

Chapitre 4

Développement du projet AppleNet

Nous avons implémenté le projet AppleNet pour effectuer la phase expérimentale de nos recherche. Cette expérimentation est nécessaire afin de mieux comprendre la création d'un réseau *blockchain* avec *Hyperledger Fabric*, ainsi que pour analyser les ressources d'apprentissage disponibles.

Ce chapitre présente donc le projet en sept sections : la mise en situation, les organismes du réseau, les permissions, le *chaincode*, les applications, les difficultés rencontrées et, finalement, les améliorations possibles.

4.1 Mise en situation

AppleNet est une simulation d'une situation de nécessité de traçabilité de produits inspirée par le cas de traçabilité de soya[Salah *et al.*, 2019] et le cas de traçabilité de

Walmart[Hyperledger, 2019b] discutés dans le chapitre 2. Cette simulation est composée de trois organismes qui échangent des lots de pommes. Le cas de traçabilité est très simplifié, comportant peu d'éléments et étant très linéaire. Chaque lot de pommes visite un organisme une seule fois pour une suite d'opérations (qui sont aussi conçues de manière linéaire). Les lots de pommes ont tous une liste d'attributs permettant de suivre leur progression à travers la chaîne d'approvisionnement. Ces attributs permettent aux organismes de diagnostiquer des problèmes dans la production, assurer le suivi de l'ordre d'opérations, ainsi que permettre de retracer les étapes suivies par chaque lot à travers son cheminement dans la chaîne d'approvisionnement.

Les lots de pommes doivent suivre trois étapes principales avant d'être acheminés aux marchés. Premièrement, le lot doit être cueilli et emballé. Deuxièmement, le lot de pommes doit être nettoyé afin d'être prêt pour la consommation. Troisièmement, chaque lot est identifié pour l'expédition aux marchés. Il est important de noter qu'afin d'assurer une traçabilité complète, il est essentiel que ces étapes soient toutes suivies dans l'ordre préétabli, sans déviation. Les contrats intelligents sont utilisés afin d'assurer le respect de cet ordre.

Chaque étape est, bien sûr, réservée à un seul organisme. Il est donc important de s'assurer que, dans un système où tous les nœuds communiquent, le réseau connaisse les règles d'accès pour donner les permissions nécessaires afin de seulement donner les accès permis. Une telle méthode se retrouve au niveau du *chaincode* et utilise une méthodologie semblable à celle employée dans l'article de vente de soya[Salah *et al.*, 2019] avec quelques améliorations proposées.

Afin d'assurer la simplicité relative du projet, une seule métrique d'environnement de production a été utilisée, soit la température de stockage. Une température de stockage de produits trop élevée ou trop faible peut endommager le produit et entraîner des pertes. Il s'agit aussi d'une métrique facile à interpréter qui sera présente uniformément dans la chaîne d'approvisionnement de nombreux produits alimentaires

et pharmaceutiques (un autre domaine qui bénéficie grandement d'améliorations en traçabilité). Le projet pourrait, entre autres, permettre la détection et le signalement automatique de températures trop élevées selon un contrat intelligent approuvé par plusieurs organismes. Ce contrat intelligent pourrait exécuter la propagation d'événements automatique à chaque point identifié dans la chaîne d'approvisionnement, que ce soit par les confirmations d'opération (cueillette, nettoyage, expédition, etc.) ou par des mises à jour constantes. Les métriques peuvent aussi être utilisées au niveau de l'application dite *frontend*. Cette application pourrait, par exemple, signaler les nouveaux événements et les actions possibles à prendre, ou identifier visuellement les lots ayant potentiellement subi un stockage à une température non permise (en affichant une icône ou une couleur différente) lors de l'affichage des données de lots. Cela permettrait aux gestionnaires de produit d'identifier rapidement des faiblesses dans les opérations non seulement au niveau de leur organisme, mais aussi au niveau de la chaîne d'approvisionnement entière, obligeant tous les participants à toujours suivre les traitements de leurs produits dans les marges désirées par leurs partenaires[1].

Ce réseau est donc créé à partir de trois parties distinctes : un réseau *blockchain* créé avec *Hyperledger Fabric* permettant la gestion du registre des transactions de lots de pommes, les différents acteurs et leurs nœuds agissant sur le réseau comportant une suite stricte de permissions et l'application web de gestion d'actifs de lots de pommes qui est accessible à tous les acteurs.

4.2 Les organismes

Comme mentionné dans la section précédente, le réseau AppleNet est composé de trois organismes principaux faisant partie de la même chaîne d'approvisionnement

[1] Un acheteur pourrait donc vérifier et valider les normes de productions de son fournisseur, lui permettant d'être plus sélectif dans la qualité de son produit et créant des pressions encourageant l'augmentation de la qualité de production.

séparés selon les trois tâches de production de lots de pommes. L'organisme *Brute* est responsable de la cueillette de pommes, l'organisme *Processed* est responsable du nettoyage des pommes et l'organisme *Shipping* est responsable de l'expédition finale vers les marchés. Chaque organisme a trois tâches à exécuter qui sont confirmées sur le réseau et visibles aux autres organismes. Les organismes sont aussi composés de différents départements leur permettant de distinguer les responsabilités internes. Cette distinction de responsabilités internes est essentielle pour protéger les données et le réseau de même que dans tout système *ERP*.

Le premier point d'entrée d'un lot de pommes dans la chaîne d'approvisionnement est l'organisme *Brute*. Cet organisme exécute trois opérations principales : la cueillette, l'emballage de pommes en lots et l'expédition des lots vers un autre organisme.

1. À la cueillette, un nouveau lot de pommes est créé dans le système avec les attributs `picker`, `appleNum`, `opDateTime` (date de création du lot) et `storageTemp`.
2. À l'emballage des pommes, une mise à jour de l'attribut `storageTemp` est effectuée.
3. Le lot de pommes est envoyé au prochain organisme avec l'attribut `newDestination`.

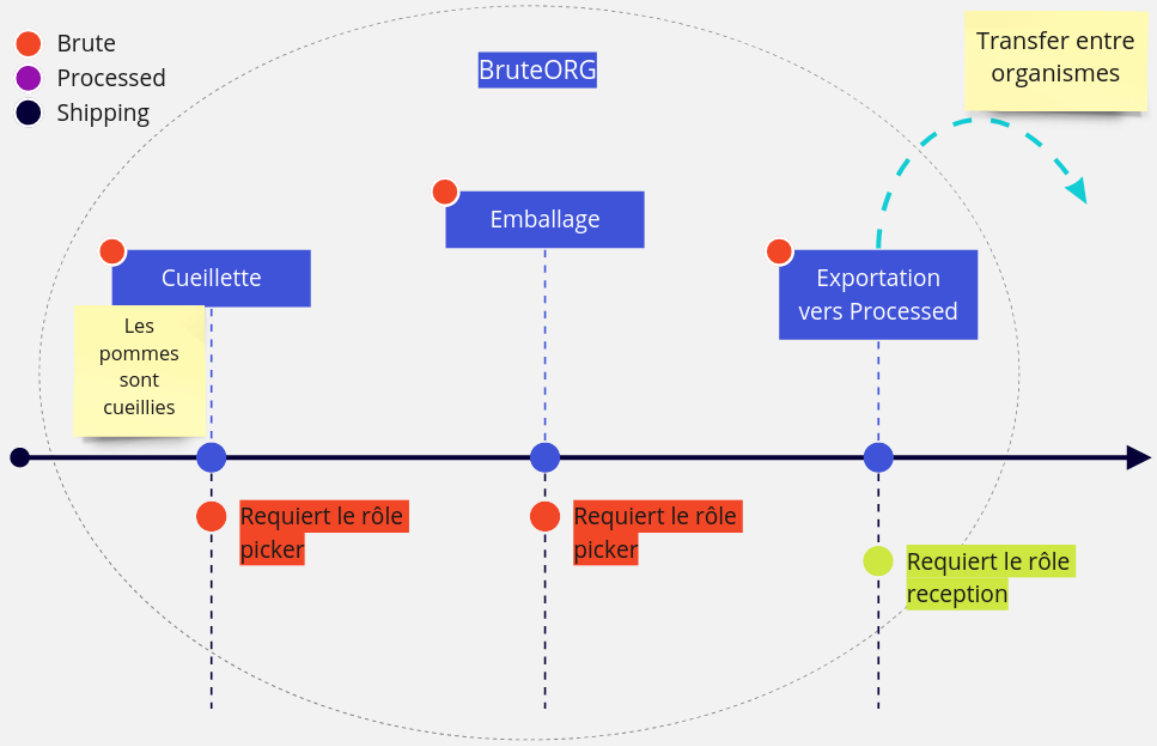

FIGURE 4.1 : Illustration de la séquence d'événements dans le traitement d'actifs de pommes dans l'organisme *Brute*

Ensuite, l'organisme *Processed* s'occupe des trois prochaines opérations, soit la réception, le nettoyage et l'expédition vers le prochain organisme.

1. À la réception, l'organisme confirme la réception du bon lot.
2. Au nettoyage, une mise à jour de l'attribut `storageTemp` est effectuée.
3. Le lot de pommes est envoyé au prochain organisme avec l'attribut `newDestination`.

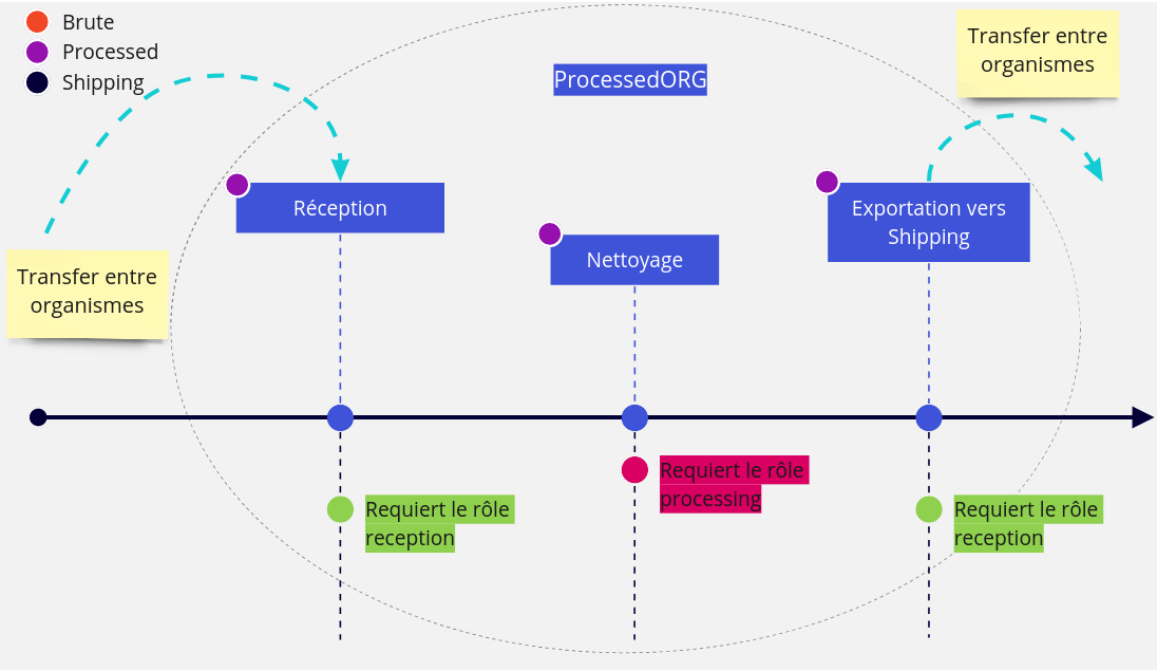

FIGURE 4.2 : Illustration de la séquence d'événements dans le traitement d'actifs de pommes dans l'organisme *Processed*

Enfin, l'organisme *Shipping* s'occupe des tâches relatives à l'expédition finale vers les marchés, soit la réception, l'étiquetage des lots et l'expédition vers les marchés.

1. À la réception, l'organisme confirme la réception du bon lot.
2. À l'étiquetage, une mise à jour de l'attribut `storageTemp` est effectuée.
3. Le lot de pommes est envoyé aux marchés avec l'attribut `newDestination`.

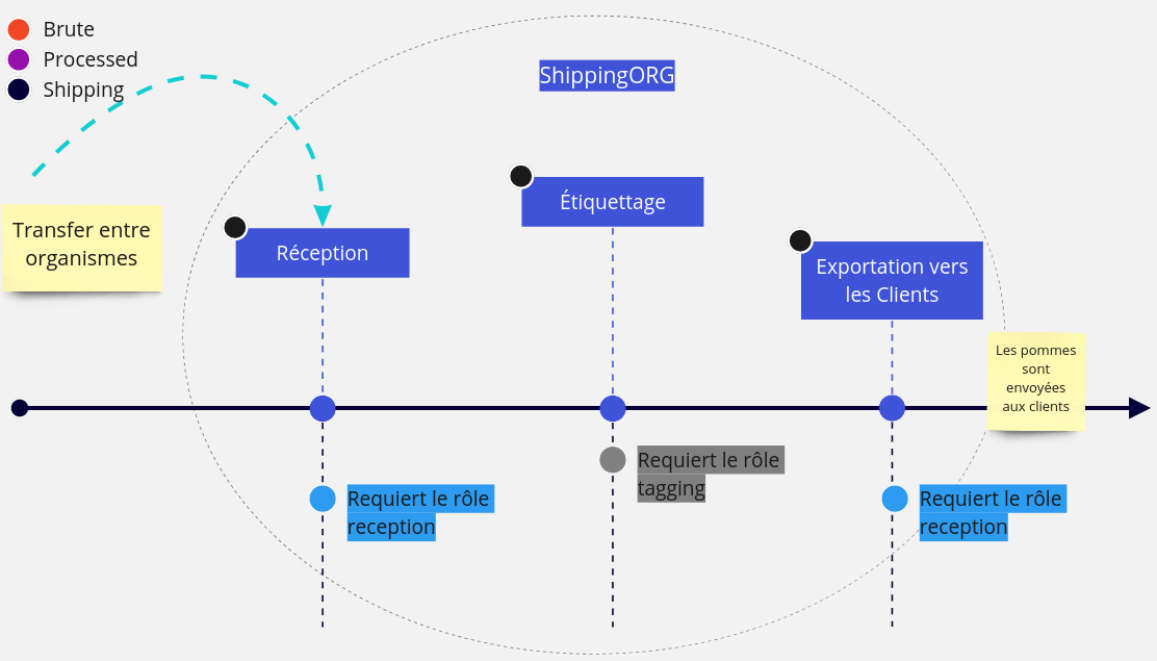

FIGURE 4.3 : Illustration de la séquence d'événements dans le traitement d'actifs de pommes dans l'organisme *Shipping*

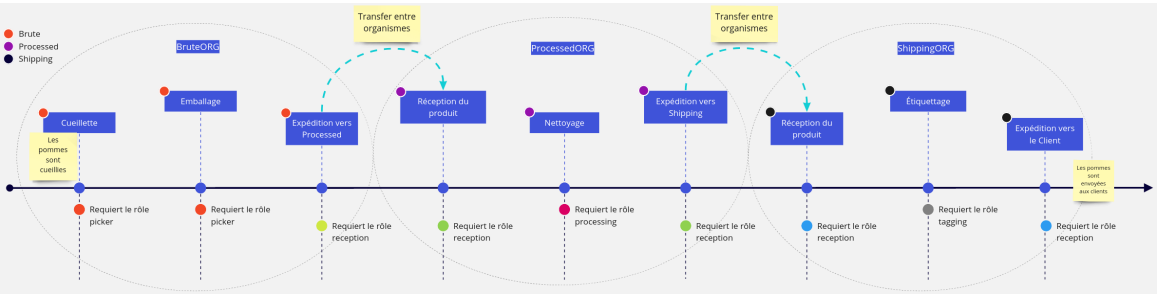

FIGURE 4.4 : Illustration de la séquence d'événements dans le traitement d'actifs de pommes dans le réseau AppleNet

La collaboration de ces trois organismes est facilitée par l'architecture *blockchain* de *Hyperledger Fabric* par le biais d'un système de permissions dont le contrôle est partagé entre le réseau lui-même et les contrats intelligents qui y résident. Chaque organisme possède des rôles pour leurs employés permettant de définir les permissions de manière plus granulaire, comme démontré dans la section sur les permissions qui suit.

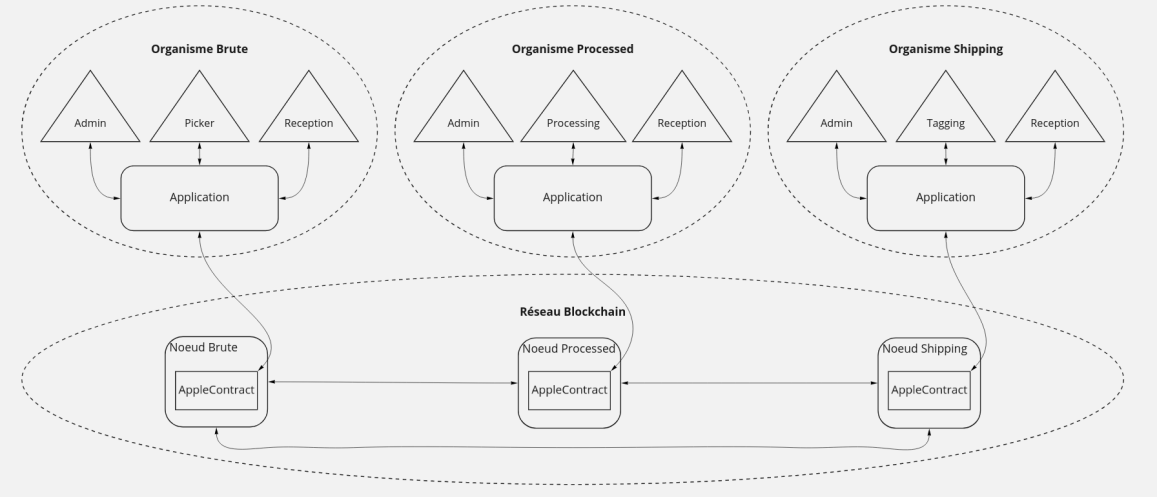

FIGURE 4.5 : Interactions entre les organismes et le réseau. Dans ce cas, chaque organisme a une application qui lui est propre. Tous les utilisateurs (`admin`, `picker`, `processing`, `tagging` et `reception`) peuvent accéder aux fonctionnalités du contrat intelligent sur le nœud de leur organisme par l'entremise de l'application de leur organisme. Tous les nœuds sur le réseau communiquent ensemble et possèdent leur instance propre du contrat intelligent.

4.3 Les permissions

Hyperledger Fabric comporte un système de permissions natif, contrairement à un système public tel que le réseau *Mainnet Ethereum*[2]. Étant un réseau privé fermé, *Hyperledger Fabric* limite les accès en lecture et en écriture au réseau par défaut. Chaque réseau *Hyperledger Fabric* est configuré avec des permissions aux droits de lecture et d'écriture sur le réseau.

Une analyse des permissions nécessaires pour les organismes *Brute*, *Processed* et

[2]Il existe plusieurs réseaux publics *Ethereum*. Le plus populaire étant le *Mainnet*, qui gère l'*Ether* utilisé couramment. Il existe, cependant, plusieurs autres réseaux publics *Ethereum* pour les opérations de test qui ne requièrent pas l'utilisation d'*Ether* qui représente une valeur monétaire réelle.

Shipping a donc été faite afin de définir une liste de rôles par entreprise afin de séparer les besoins des utilisateurs individuels. Cette séparation des besoins est essentielle pour assurer la confidentialité de données ainsi que leur intégrité.

4.3.1 Les groupes d'identités de Hyperledger Fabric

Afin de permettre un contrôle granulaire sur les accès au réseau, les identités de *Hyperledger Fabric* sont séparées en trois catégories principales : les nœuds, les administrateurs et les clients.

Le premier groupe d'identités, les nœuds, protège l'information au niveau interorganisationnel. Des droits de lecture et d'écriture leur sont accordés selon la visibilité qui leur est nécessaire. Ces droits peuvent être adaptés selon le besoin de visibilité sur le registre.

Par exemple, un organisme aurait des droits de lecture et d'écriture s'il est propriétaire de l'information ajoutée au registre.

Si l'organisme n'est pas propriétaire de l'information, mais doit tout de même ajouter cette information au registre (par exemple l'organisme de transport qui doit ajouter des mises à jour de température lors du transport), celui-ci pourrait avoir des droits d'écriture, mais pas de droits de lecture[3]. Cet organisme pourrait donc seulement ajouter de l'information, mais ne pourrait jamais lire les informations qui y sont ajoutées.

Dans un dernier cas, si l'organisme n'a pas besoin d'ajouter l'information, par

[3]Une solution alternative existe où l'organisme de transport n'est pas un participant au réseau (et ne possède donc pas de nœuds). Dans cette situation, l'organisme de transport possède une identité chez un autre organisme qui pourra lui restreindre les droits soit au niveau du *chaincode*, ou au niveau de l'application.

exemple un système d'audit ou un simple droit d'accès de visibilité dans un cas de transparence, cet organisme pourrait préserver une présence sur le réseau avec un nœud ayant des droits de lecture seulement.

Ensuite, les administrateurs possèdent leur propre identité, leur donnant des droits de manipulation de nœuds sur le réseau qui peuvent être définis dans les configurations du réseau. Ces droits incluent des droits de gestion de nœud, des droits de gestion d'identités et des droits de gestion du réseau (proposition de changements).

Enfin, les utilisateurs dits clients ont des identités. Ces identités sont accordés aux utilisateurs et aux applications afin de leur permettre d'interagir avec le réseau par l'entremise des nœuds. Les identifiants offrent une opportunité de gestion d'accès aux ressources du réseau, permettant aux employés d'un organisme d'accéder seulement aux services qui les concernent.

À cette fin, *Hyperledger Fabric* utilise un système de gestion d'identité se basant sur des certificats d'authentification. Toute entité sur le réseau, que ce soit un utilisateur ou un nœud, doit posséder un certificat d'authentification. Ces certificats permettent, entre autres, d'ajouter des attributs à une identité tels qu'une position dans l'organisme ou un département associé. L'API de *Hyperledger Fabric* fournit les outils nécessaires pour interagir avec ces attributs, donnant une flexibilité importante pour un système basé sur les permissions. C'est sur ces fonctionnalités d'accès que le projet s'est concentré. Alors que les fonctions d'accès au réseau requièrent une connaissance plus profonde d'un architecte de réseau, la majorité du travail suivant la mise en place d'un réseau *Hyperledger Fabric* (excluant l'ajout et le retrait d'organismes) se retrouvera au niveau des accès individuels dans le *chaincode* et dans les applications. Cette section se concentrera donc sur des applications d'identités au niveau du *chaincode*.

4.3.2 Les identités AppleNet

La première étape dans cette gestion d'identité est l'enregistrement des utilisateurs. Cet enregistrement inclut la création de certificats. Ces certificats peuvent être ajoutés, retirés, ou même avoir une date d'expiration, permettant d'avoir un grand contrôle sur les accès d'un utilisateur au niveau du réseau ainsi qu'au niveau des opérations d'une entreprise.

L'identité des utilisateurs comporte plusieurs champs intéressants du point de vue de la gestion d'accès. Entre autres, l'identité possède un champ `affiliation` ainsi qu'un champ `attrs` (vu dans le listing 4.1). Les deux champs offrent l'opportunité de donner des informations d'identification supplémentaires à l'identité. Alors que le champ `affiliation` donne des informations explicites d'appartenance départementale, le champ `attrs` fournit une flexibilité permettant d'affecter une identité avec un nombre illimité d'attributs différents à l'aide d'une syntaxe clé-valeur.

```
1  // Register the user, enroll the user, and import the new identity
       into the wallet.
2  const affiliations = [
3      {
4          name: "department",
5          value: "picker",
6          ecert: true
7      }
8  ];
9  const secret = await ca.register({
10     affiliation: 'brute.department1.picking',
11     enrollmentID: 'picker',
12     role: 'client',
13     attrs: affiliations
14 }, adminUser);
15 const enrollment = await ca.enroll({
16     enrollmentID: 'picker',
17     enrollmentSecret: secret
18 });
19 const x509Identity = {
20     credentials: {
21         certificate: enrollment.certificate,
22         privateKey: enrollment.key.toBytes(),
23     },
24     mspId: 'BruteMSP',
25     type: 'X.509',
26 };
27 await wallet.put('picker', x509Identity);
```

Listing 4.1: Enregistrement utilisateur `picker` (`registerPickerBrute.js`)

```
1  // Register the user, enroll the user, and import the new identity
       into the wallet.
2  const affiliations = [
3      {
4          name: "department",
5          value: "reception",
6          ecert: true
7      }
8  ];
9  const secret = await ca.register({
10     affiliation: 'brute.department1.reception',
11     enrollmentID: 'reception',
12     role: 'client',
13     attrs: affiliations
14 }, adminUser);
15 const enrollment = await ca.enroll({
16     enrollmentID: 'reception',
17     enrollmentSecret: secret
18 });
19 const x509Identity = {
20     credentials: {
21         certificate: enrollment.certificate,
22         privateKey: enrollment.key.toBytes(),
23     },
24     mspId: 'BruteMSP',
25     type: 'X.509',
26 };
27 await wallet.put('reception', x509Identity);
```

Listing 4.2: Enregistrement utilisateur `reception` (`registerReceptionBrute.js`)

Dans le cas présent de gestion d'accès, on associe un seul rôle (cueillette de pommes ou réception) par département ou sous-département (cueillette de pommes ou réception). Les deux champs (la valeur `department` dans `attrs` et la valeur de `affiliations`) sont donc considérés comme étant équivalents dans la problématique des accès présente. Cependant, dans un cas réel, un membre d'un département pourrait potentiellement avoir des responsabilités dans plusieurs départements, ou un département pourrait avoir plusieurs tâches qui requièrent des permissions spécifiques. Dans de tels cas, l'attribut `attrs` est plus apte à répondre aux besoins d'accès, permettant à un utilisateur d'avoir plusieurs attributs ou rôles différents lui donnant accès à différentes ressources et fonctionnalités. Cela permet donc une grande flexibilité de solution.

Si, par exemple, un gérant a besoin d'accéder à un grand nombre de fonctionnalités, les gestionnaires du réseau ont deux options possibles. Dans le contrat intelligent, ils pourraient donner une permission spécifique aux gérants et donner des accès au rôle *gérant* à toutes les fonctions. Ceci a comme avantage d'avoir une grande simplicité en créant un rôle aux accès multiples. L'autre option est d'affecter plusieurs rôles aux identités de gérants (par exemple, un gérant aurait le rôle de cueillette ainsi que le rôle de réception) sans changer les permissions existantes dans le contrat intelligent. Cette option a comme avantage de donner un contrôle beaucoup plus granulaire des accès et ainsi d'éviter la création de rôles passe-partout.

Dans le réseau AppleNet, on utilise donc une solution utilisant l'identifiant du *Membership Service Provider*[4] (`BruteMSP`, `ProcessedMSP` et `ShippingMSP` respectivement) pour restreindre les accès par organisme et l'identifiant `department` de l'attribut `attrs` de l'identité de l'utilisateur afin de restreindre les accès par rôle départemental. Voici donc les associations des différents rôles et leurs fonctions permises :

	Brute		Processed		Shipping	
	picker	reception	processing	reception	tagging	reception
Fonction						
queryAllApples	O	O	O	O	O	O
getAppleHistory	O	O	O	O	O	O
getCurrentApple	O	O	O	O	O	O
pick	O					
bundle	O					
outbound_B		O				
received_P				O		
clean			O			
outbound_P				O		
received_S						O
tag					O	
shipped						O

TABLE 4.1 : Table des droits d'accès aux fonctions dénotés par des O (descriptions dans l'annexe A)

[4]Le MSP est une structure permettant aux organismes d'organiser et gérer leurs identités en abstrayant la couche cryptographique[Hyperledger, 2020c]

Dans le tableau 4.1, on peut voir que les tâches de lecture de données sont toutes permises à n'importe quel utilisateur. Cette fonction est nécessaire pour la fonction `getCurrentApple` en particulier puisque cette fonction permet de récupérer un actif de pommes. Elle est essentielle au fonctionnement de toutes les autres fonctions et doit donc être permise aux autres fonctions[5]. Les autres fonctions telles que la fonction `queryAllApples` (qui retourne la liste de tous les actifs) et `getAppleHistory` (qui retourne tous les états précédents d'un seul actif) pourraient aussi bénéficier de restrictions plus strictes ; par exemple, enlever le droit de visualisation d'états de pommes aux utilisateurs `picker`, qui sont seulement aux premières étapes et n'ont pas besoin de voir ces informations.

De leur côté, toutes les tâches de réception (`received`) et d'expédition (`outbound` / `shipped`) tombent sous l'expertise du département de réception. Les fonctions associées sont donc réservées aux utilisateurs avec le rôle `reception`. Dans chaque fonction, le contrat intelligent limite aussi les accès par organisme. Donc, seul un membre de la réception de l'organisme *Shipping* peut appeler la fonction `shipped`, malgré le fait qu'il partage le même nom de rôle qu'un utilisateur de la réception de l'organisme *Brute*.

Enfin, chaque organisme a des fonctions qui lui sont propres. Chaque fonction de ce type a son propre rôle associé. Parmi ces fonctions, la fonction `pick` instancie l'actif de lot de pommes dans le réseau.

Cette fonction est propre à l'organisme *Brute* et est responsable de la création d'un lot de pommes dans le réseau AppleNet. `pick` est une action réservée aux membres de l'entreprise qui exécutent la cueillette. Elle requiert donc que l'utilisateur l'exécutant soit membre du département de la cueillette et ait donc le rôle `picker`. Une fonction de validation d'identité est donc implémentée dans le contrat intelligent pour valider

[5]Cette fonction pourrait aussi être séparée en deux, soit une fonction privée permettant d'extraire la pomme courante dans les fonctions du contrat intelligent et une autre fonction publique pouvant être appelée seulement par les utilisateurs avec les permissions nécessaires.

l'identité d'un utilisateur. En suivant des ressources en ligne sur le contrôle d'accès à base d'attributs[Adhav, 2020], on obtient une suite de conditions `if` validant les différents attributs d'identités, tel que vu dans l'extrait de code 4.3.

```
async pick(ctx: AppleContext, picker: string, appleNum: string,
    opDateTime: string, storageTemp: Number) {
    let permitted_orgs: string[] = ['BruteMSP'];
    let required_department: string[] = ['picker'];

    const id: ClientIdentity = args[0].clientIdentity;
    if (!(permitted_orgs.includes(id.getMSPID()))){
        throw new Error('User Is not from a permitted org.');
    }
    if (!(required_department.includes(id.getAttributeValue('
        department')))){
        throw new Error('User does not have sufficient access.');
    }

    // Create an instance of the apple
    let apple = Apple.createInstance(picker, appleNum, opDateTime);

    // Add the apple to the list of all similar apples in the ledger
        world state
    await ctx.appleList.addApple(apple);

    // Must return a serialized apple to caller of smart contract
    return apple;
}
```

Listing 4.3: Solution d'accès originale (`applecontract.ts`)

Dans AppleNet, les permissions d'un utilisateur sont vérifiées par l'entremise d'une liste d'organismes et de rôles permis. Si un utilisateur n'a pas les accès permis, la fonction lance une erreur décrivant des droits d'accès inadéquats. Cette erreur permet au contrat intelligent d'annuler la transaction et d'enregistrer les détails de l'erreur. L'application peut aussi limiter les accès sur le côté *frontend*, tout en gérant les accès non permis s'il y a lieu afin d'éviter des appels au réseau non nécessaires.

Pour l'utilisation occasionnelle de droits d'accès dans le code, cette structure pourrait être vue comme étant suffisante. Cependant, il s'agit d'une structure très répétitive lorsqu'il y a un besoin de plusieurs accès différents. À moins de donner des accès illimités à tout utilisateur dans le *channel*, il est fort probable que les contrats intel-

ligents aient de nombreuses lignes de code validant l'identité de l'utilisateur appelant ses fonctions. Cette répétition de code de droit d'accès peut être résolue de différentes manières.

1. Ne pas effectuer de changement. Accepter la répétition de code.
2. Créer une méthode de gestion d'accès. Effectuer une abstraction interne au contrat intelligent ou dans une classe de gestion d'accès externe.
3. Utiliser une approche orientée aspect afin de découpler la logique d'accès de la logique métier.

4.3.3 L'approche orientée aspect

Afin de bien séparer la logique métier de la logique d'accès, une solution orientée aspect a été adoptée. Un module TypeScript appelé `access.ts` (vu dans le listing 4.4) a été créé afin de permettre l'utilisation du décorateur `@access` dans les contrats intelligents. L'utilisation des décorateurs a grandement influencé la décision d'utiliser le langage TypeScript plutôt que le JavaScript.

```
1  import { ClientIdentity } from 'fabric-shim-api';
2
3  function access(permitted_orgs: string[], required_department = []){
4      return function(target: any, propertyKey: string, descriptor:
           PropertyDescriptor) {
5          const method = descriptor.value;
6          descriptor.value = function wrapped(...args: any[]): any {
7              const id: ClientIdentity = args[0].clientIdentity;
8              if (!(permitted_orgs.includes(id.getMSPID()))){
9                  throw new Error('User is not from a permitted org.');
10             }
11             if (required_department.length > 0){
12                 if (!(required_department.includes(id.
                       getAttributeValue('department')))){
13                     throw new Error('User does not have sufficient
                           access.');
14                 }
15             }
16             const ret = method.apply(this, args);
17             return ret;
18         };
19     }
20 }
21
22 export { access };
```

Listing 4.4: Décorateur d'accès (`access.ts`)

La fonction `access` (vue dans l'extrait de code 4.4) prend en argument une liste de noms d'organismes ainsi que le nom du département requis pour utiliser une fonction. La fonction vérifie premièrement le nom de l'organisme de l'utilisateur avec l'identifiant de son MSP (le réseau AppleNet suppose que les organismes ont tous une relation un à un avec leur MSP). Ensuite, la fonction vérifie le département de l'utilisateur avec l'attribut `department` de son identité. Si l'utilisateur n'a pas les accès nécessaires, la fonction `access` lance une exception avec une description de la cause. La transaction est donc annulée et l'application appelant le contrat intelligent est avertie de l'exception. La validation d'organisme est nécessaire puisque, comme mentionné plus tôt, il est possible que différents organismes partagent des noms de département. Ce problème est particulièrement visible lorsque plusieurs organismes différents partagent un même contrat intelligent.

L'effet sur le code est très visible, ce qui permet de déclarer les droits d'accès avant

une fonction et éviter la redondance de code dans la fonction, comme on le voit dans l'extrait de code 4.5.

```
static readonly permittedBrutes = ['BruteMSP'];
static readonly permittedProcessed = ['ProcessedMSP'];
static readonly permittedShipping = ['ShippingMSP'];

@access(
    AppleContract.permittedBrutes,
    [
        'picker'
    ]
)
async pick(ctx: AppleContext, picker: string, appleNum: string,
   opDateTime: string, storageTemp: Number) {

    // Create an instance of the apple
    let apple = Apple.createInstance(picker, appleNum, opDateTime);

    // Add the apple to the list of all similar apples in the ledger
       world state
    await ctx.appleList.addApple(apple);

    // Must return a serialized apple to caller of smart contract
    return apple;
}
```

Listing 4.5: Solution d'accès `pick` avec décorateur (`applecontract.ts`)

La conception du décorateur `access` est telle qu'une liste d'organismes doit obligatoirement être fournie en argument. Cela oblige le programmeur à donner une liste fixe d'organismes au décorateur afin d'éviter des accès non intentionnels à des fonctions par des utilisateurs partageant le réseau. L'extrait de code 4.6 est un exemple de l'utilisation du décorateur `@access` avec plusieurs organismes permis. La même solution peut être implémentée avec les départements.

```
@access(
    [
        ...AppleContract.permittedBrutes,
        ...AppleContract.permittedProcessed,
        ...AppleContract.permittedShipping
    ]
)
async getCurrentApple(ctx: AppleContext, picker: string, appleNum:
   Number){
    // Retrieve the current apple using key fields provided
    let appleKey = Apple.makeKey([picker, appleNum]);
    let apple = await ctx.appleList.getApple(appleKey);
    if (apple == null){
        throw new Error('Apple ' + picker + appleNum + ' does not
           exist');
    }
    return apple
}
```

Listing 4.6: Solution d'accès à plusieurs organismes (`applecontract.ts`)

En somme, la gestion des accès dans le projet AppleNet se fait à deux niveaux dans le contrat intelligent : 1. Les accès sont limités par organisme 2. Les accès sont limités par nom de rôle dans l'organisme. L'organisation du code a été améliorée par un découplage de la logique métier de la logique d'accès avec le patron décorateur, s'inspirant de la programmation orientée aspect. Cette architecture a permis de mieux organiser le code ainsi que faciliter l'implémentation de contrôle d'accès à travers le code de contrats intelligents.

4.4 Le *chaincode*

Il y a une culture qui persiste dans l'écosystème des *blockchains* publiques : *code is law* - le code est la loi. Cette culture indique aux utilisateurs que toute exécution d'un contrat intelligent est finale. Comme mentionné plus tôt, toute transaction sur le registre est considérée comme étant permanente et inchangeable (à moins d'un cas réellement exceptionnel où la majorité des nœuds s'entendent sur un retour en arrière). Avec l'ajout de contrats intelligents sur les *blockchains*, on approche la création de

contrats réels permettant d'agir comme preuve de consentement aux modalités d'une entente.

Dans le cas de publication de contrats intelligents sur une plateforme publique, il est entendu que la responsabilité de vérification repose sur les utilisateurs du contrat intelligent. Si un acteur utilise un contrat intelligent, on sous-entend qu'il comprend l'effet de ce contrat et comprend aussi que toute utilisation sera considérée comme valide. On peut voir cette règle comme une condition d'utilisation implicite. L'utilisateur consent à ce que toute transaction portant sa signature de clé privée soit considérée comme étant faite par lui. Le contrat intelligent peut donc traiter ces transactions telles que définies dans ses fonctions appelées et ne porte aucune responsabilité quant à l'effet de cette transaction sur l'utilisateur[6].

À l'inverse, tout contrat publié sur la *blockchain* est sujet à l'effet de toute fonction l'affectant (que ce soit ses propres fonctions ou l'effet d'un contrat externe). Donc, si une transaction a un effet négatif sur un contrat intelligent sur la *blockchain*, cette faute est considérée comme la responsabilité du développeur ayant publié le contrat, ainsi que des utilisateurs qui lui ont fait confiance. Ceci encourage fortement les utilisateurs à être bien conscient des contrats intelligents qu'ils utilisent. Les développeurs de contrats intelligents doivent aussi faire très attention à l'implémentation de leurs contrats afin de ne pas injecter de faute dans leur code (étant donné la permanence de ces contrats sur la *blockchain*).

Cette mentalité prend une approche légèrement différente dans le cas de *blockchains* privées. En effet, la *blockchain* privée comporte un élément d'identité qui n'existe pas dans la *blockchain* publique. Chaque utilisateur et nœud est un participant connu de la *blockchain*. Ces utilisateurs sont donc encouragés à bien agir entre

[6]Selon la philosophie *code is law*. La réalité diffère et les auteurs de contrats intelligents peuvent certainement être considérés comme étant responsables de pertes de fonds. Advenant le cas d'une perte non intentionnelle de fonds importants, il faut convaincre les administrateurs de nœuds individuels de réinitialiser leur nœud à un bloc précédent, invalidant toutes les transactions (dont la transaction frauduleuse) qui avaient été ajoutées depuis ce bloc.

eux.

Du côté de *Hyperledger Fabric*, chaque contrat intelligent doit être approuvé par un certain nombre de participants dans le *channel* sur lequel le contrat sera installé. Ceci implique que les organismes de ce *channel* consentent à l'autorité de ce contrat sur les données qui lui sont envoyées, indiquant que les contrats intelligents peuvent agir partiellement en tant qu'entente entre deux ou plusieurs organismes. Les organismes peuvent donc se servir des modalités du contrat intelligent comme argument en cas d'arbitrage tel que dans le cas de logistique de Walmart[Hyperledger, 2020b].

Dans le cas de AppleNet, les organismes se servent du contrat intelligent `AppleContract` pour gérer l'ordre des opérations, ainsi que pour enregistrer les changements effectués et l'organisme responsable de ces changements.

4.4.1 AppleContract

La première fonctionnalité importante du contrat intelligent est celle de la validation d'états. En effet, comme présenté dans l'extrait de code 4.7, les actifs Apple ont neuf états possibles qui doivent être suivis dans un ordre séquentiel afin de s'assurer que les actifs ont suivi tous les bons processus de production.

```
1  const cpState = {
2      PICKED: 1,      //Brute
3      BUNDLED: 2,     //Brute
4      OUTBOUND_B: 3, //Brute
5      RECEIVED_P: 4,    //Processed
6      CLEANED: 5,       //Processed
7      OUTBOUND_P: 6,    //Processed
8      RECEIVED_S: 7,      //Shipping
9      TAGGED: 8,          //Shipping
10     SHIPPED: 9          //Shipping
11 };
```

Listing 4.7: Liste d'états possibles d'un actif (`apple.ts`)

```
1 picker: string;
2 holder: string;
3 opDateTime: string;
4 storageTemp: Number;
5 destination?: string;
6 currentState: any;
```

Listing 4.8: Variables de la classe `Apple` (`apple.ts`)

La classe représentant l'actif `Apple`, `Apple.ts`, possède des accesseurs pour les attributs de lot de pomme présentés dans l'extrait 4.7 ainsi que des méthodes permettant de passer au prochain état du lot, ce qui permet de restreindre les actions possibles lors de la manipulation de cet actif par l'entremise des contrats intelligents. Le contrat intelligent `AppleContract`, de son côté, est responsable du respect de l'ordre d'opérations tel que suggéré dans la représentation simplifiée dans les extraits de code 4.9 et 4.10 (un diagramme des classes est présenté dans la figure 4.6).

```
export class AppleContract extends Contract {
    constructor() {super('org.applenet.apple');}
    createContext() {return new AppleContext();}

    static readonly permittedBrutes = ['BruteMSP'];
    static readonly permittedProcessed = ['ProcessedMSP'];
    static readonly permittedShipping = ['ShippingMSP'];

    async instantiate(ctx: AppleContext) {console.log('Instantiating
        AppleContract');}

    async queryAllApples(ctx: AppleContext, startKey='', endKey='')
        {...}
    async getAppleHistory(ctx: AppleContext, picker: string, num:
        string) {...}
    async getCurrentApple(ctx: AppleContext, picker: string, appleNum
        : Number){...}

    async pick(ctx: AppleContext, picker: string, appleNum: string,
        opDateTime: string, storageTemp: Number) {...}
    async bundle(ctx: AppleContext, picker: string, appleNum: Number,
         storageTemp: Number) {...}
    async outbound_B(ctx: AppleContext, picker: string, appleNum:
        Number, newDestination: string, storageTemp: Number) {...}

    async received_P(ctx: AppleContext, picker: string, appleNum:
        Number, storageTemp: Number) {...}
    async clean(ctx: AppleContext, picker: string, appleNum: Number,
        storageTemp: Number) {...}
    async outbound_P(ctx: AppleContext, picker: string, appleNum:
        Number, newDestination: string, storageTemp: Number) {...}

    async received_S(ctx: AppleContext, picker: string, appleNum:
        Number, storageTemp: Number) {...}
    async tag(ctx: AppleContext, picker: string, appleNum: Number,
        storageTemp: Number) {...}
    async shipped(ctx: AppleContext, picker: string, appleNum: Number
        , newDestination: string, storageTemp: Number) {...}
}
```

Listing 4.9: `AppleContract` simplifié. Les méthodes sont groupées par fonction et par organisme. (`applecontract.ts`)

```
async bundle(ctx: AppleContext, picker: string, appleNum: Number,
    storageTemp: Number) {
    // Retrieve the current apple using key fields provided
    let apple = await this.getCurrentApple(ctx, picker, appleNum);

    apple.setStorageTemp(storageTemp);

    // First move from picked to bundled
    if (apple.isPicked()) {
        apple.setBundled();
    } else {
        throw new Error('Apple ' + picker + appleNum + ' is not
            bundlable. Current state = ' + apple.getCurrentState());
    }

    // Update the apple
    await ctx.appleList.updateApple(apple);

    console.log('Bundled apple ' + picker + ' ' + appleNum);

    return apple;
}
```

Listing 4.10: Fonction `bundle` (`applecontract.ts`)

Dans l'extrait 4.10 du contrat intelligent `AppleContract`, on peut voir l'appel de la méthode `apple.isPicked()`. Cette méthode permet de confirmer que l'actif `Apple` est dans l'état `PICKED`, un prérequis de l'emballage d'un lot de pommes. Si l'actif n'est pas dans le bon état, le contrat intelligent lance une erreur et la transaction est invalidée. Si l'actif est bien dans l'état `PICKED`, la fonction change l'état de l'actif à l'état `BUNDLED` avec la méthode `apple.setBundled()`.

Ce type de validation est présent pour les neuf états des actifs de lot de pommes. Il est impossible de passer à un état sans avoir son prérequis.

Une validation supplémentaire pourrait être faite dans la classe `Apple` (en plus de la validation dans le contrat intelligent) afin de s'assurer du respect de l'ordre des opérations. Cependant, le respect des règles demeure une responsabilité du contrat intelligent. Une validation supplémentaire dans la classe de l'actif serait redondante et pourrait entraîner des difficultés plus tard s'il y a une modification à faire dans le

contrat. Par exemple, si une nouvelle étape est ajoutée au processus, il faudrait non seulement changer la définition de la fonction dans le contrat intelligent, mais aussi changer la définition de la méthode dans la classe de l'actif `Apple`. Dans le cas présent, la classe `Apple` est responsable de l'accès aux données de l'actif, alors que le contrat intelligent `AppleContract` est responsable de la *manipulation* de ses données.

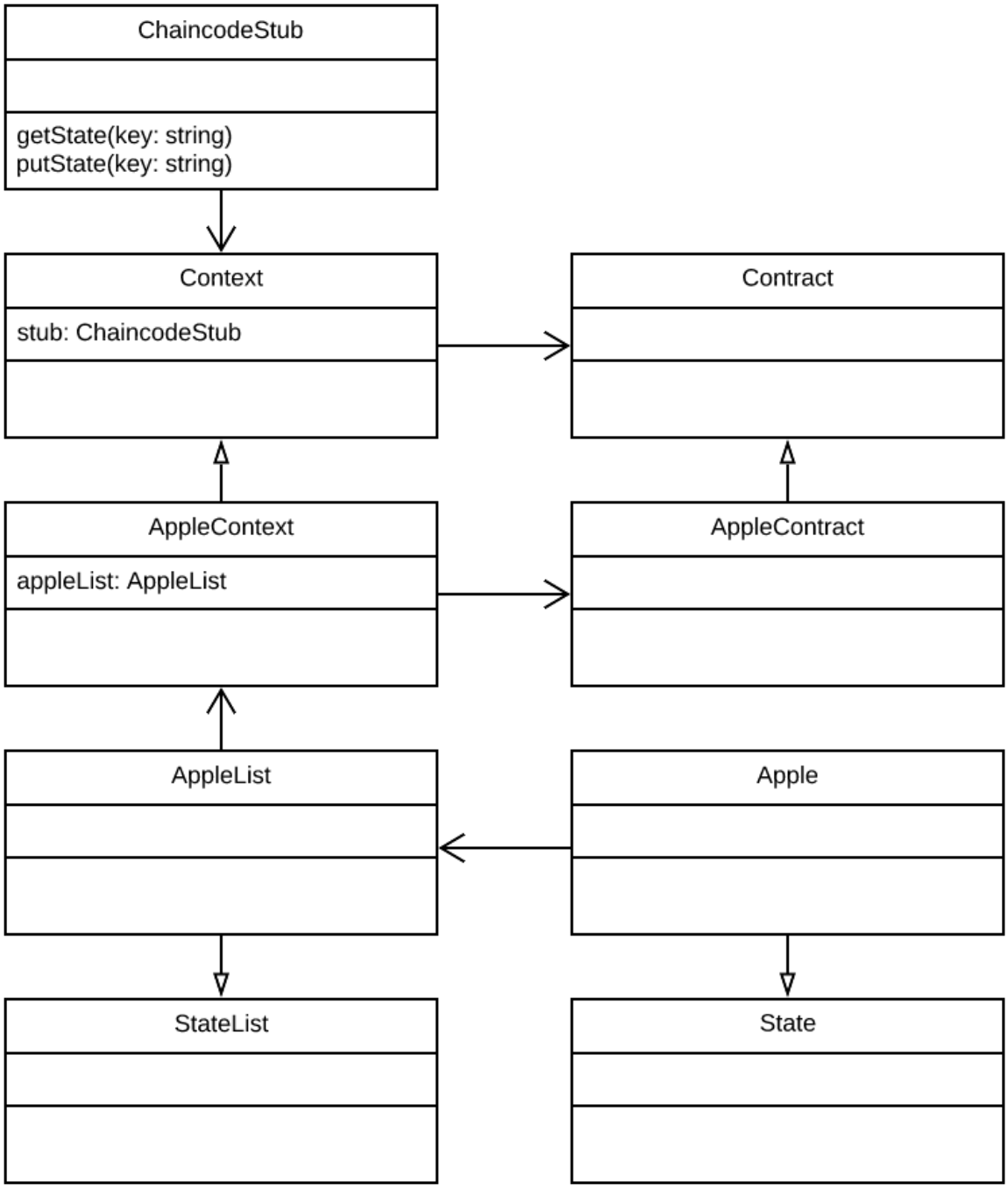

FIGURE 4.6 : Diagramme simplifié de la relation entre les classes du projet *AppleNet*.

4.4.2 Les fonctions

Chaque fonction comporte huit parties principales dans sa définition :

1. La définition des accès.
2. La définition des paramètres.
3. L'accès à l'actif de lot de pommes s'il existe.
4. La mise à jour de la température de stockage de l'actif.
5. La validation des conditions nécessaires à la fonction (optionnel).
6. La validation et mise à jour de l'état de l'actif.
7. La propagation de l'événement de création de la transaction.
8. La mise à jour de l'actif.

En premier lieu, les accès sont définis par le décorateur d'accès. Le contrat possède des listes prédéfinies d'organismes de chaque type ou rôle dans la chaîne d'approvisionnement (*Brute*, *Processed*, *Shipping*). Ces listes définissent quels organismes ont droit à quelles fonctions du contrat intelligent. Par exemple, seul un organisme enregistré comme *Brute* dans la liste du contrat intelligent peut exécuter les fonctions de cueillette de pommes et d'assemblage en lots. Cette fonctionnalité est inspirée de l'étude de cas de vente de grains de soya[Salah *et al.*, 2019].

Ensuite, la fonction, incluant ses arguments, est définie. Chaque fonction qui sera utilisée à l'extérieur du contrat intelligent doit commencer par un contexte `ctx`. Ce contexte contient les informations d'identité de l'utilisateur appelant la fonction. Le contexte possède aussi des APIs de manipulation de données du registre tels que

l'ajout d'actifs, le retrait d'actifs du *World State* (la base de données donnant un accès rapide aux informations dans le registre[7]), ainsi que des fonctions d'accès aux actifs permettant d'exécuter des recherches dans le registre. Dans ce cas, on se sert du contexte pour implémenter des contrôles d'accès par l'entremise du décorateur `@access` (le contexte est intercepté avant l'exécution de la fonction).

Par la suite, le contrat utilise la fonction `getCurrentApple` (extrait de code 4.11) afin d'accéder à l'actif de lot de pommes recherché. La fonction lance une exception si l'actif n'existe pas et retourne un objet `Apple` si l'actif existe.

```
1 async getCurrentApple(ctx: AppleContext, picker: string, appleNum:
     Number){
2     // Retrieve the current apple using key fields provided
3     let appleKey = Apple.makeKey([picker, appleNum]);
4     let apple = await ctx.appleList.getApple(appleKey);
5     if (apple == null){
6         throw new Error('Apple ' + picker + appleNum + ' does not
              exist');
7     }
8     return apple
9 }
```

Listing 4.11: Fonction `getCurrentApple` (`applecontract.ts`)

Après le retrait de l'actif, la température de stockage est mise à jour. Si la fonction lance une erreur, cette mise à jour n'est pas appliquée à l'actif.

Ensuite, le contrat valide certaines conditions spécifiques à la fonction. Une telle condition est présente dans les fonctions de réception. Dans ces cas, le contrat vérifie que l'organisme recevant l'actif est le destinataire indiqué lors de l'expédition de l'actif.

Après la validation de conditions spécifiques, le contrat valide l'état de l'actif. Si cet actif n'est pas dans l'état prérequis à l'exécution de la fonction, la fonction lance une erreur indiquant l'état nécessaire pour la fonction ainsi que l'état courant de l'actif

[7]Les informations retirées du *World State* sont toujours disponibles sur le registre lui-même et on peut y accéder avec la fonction `ctx.stub.getHistoryForKey(<clé>)` qui permet de lire toutes les opérations passées d'un actif ainsi que savoir si cet actif a été retiré du *World State*.

de lot de pommes. Une fois que l'état est validé, l'état de l'actif est mis à jour.

Un événement est ensuite créé et propagé afin d'avertir les applications du changement effectué par le contrat. Ces événements sont présentement utilisés par les administrateurs pour visualiser toutes les opérations sur les actifs.

Enfin, l'actif est mis à jour dans le registre, confirmant toutes les opérations exécutées par la fonction.

Le contrat intelligent `AppleContract` possède aussi des fonctions de transfert d'actifs (présenté dans l'extrait 4.12). Ces fonctions permettent de changer le détenteur d'un actif en utilisant des opérations d'expédition (`outbound` et `shipped`) et de réception (`received`) telles que présentées dans les deux prochains extraits de code source du contrat intelligent `AppleContract`. La destination et la provenance de chaque opération sont indiquées par le suffixe de la classe, représentant le type d'organisme (*Brute*, *Processed*, ou *Shipping*).

```
@access(AppleContract.permittedBrutes, ['reception'])
async outbound_B(ctx: AppleContext, picker: string, appleNum: Number,
    newDestination: string, storageTemp: Number) {
    // Retrieve the current apple using key fields provided
    let apple = await this.getCurrentApple(ctx, picker, appleNum);

    apple.setStorageTemp(storageTemp);

    // Validate current holder who should be th original picker.
    if (apple.getHolder().charAt(0).toUpperCase() + apple.getHolder()
       .slice(1) !== picker) {
        throw new Error('Apple ' + picker + appleNum + ' is not owned
            by ' + apple.getHolder());
    }

    if (apple.getHolder() == newDestination) {
        throw new Error('Apple' + picker + appleNum + 'must be sent
           to a different org');
    }

    // Move state from bundled to outbound
    if (apple.isBundled()) {
        apple.setOutbound_B(newDestination);
    } else {
        throw new Error('Apple ' + picker + appleNum + ' is not
           bundled. Current state = ' + apple.getCurrentState());
    }

    ctx.stub.setEvent("ShipAppleFromB", apple.toBuffer());

    // Update the apple
    await ctx.appleList.updateApple(apple);

    console.log('Shipping apple ' + picker + ' ' + appleNum);

    return apple;
}
```

Listing 4.12: Fonction `outbound_B` (`applecontract.ts`)

D'abord, le transfert d'actifs est initié par une des fonctions d'expédition (`outbound/ship`). Ces fonctions ont comme tâche d'indiquer qu'un actif est en transit vers un autre organisme. Lors de l'expédition, un destinataire est ajouté aux attributs de l'actif par l'entremise de la méthode `setOutbound(newDestination)` de l'actif[8].

[8]Puisque toutes les expéditions et les réceptions sont équivalentes et que le destinataire est défini dans l'actif au moment de l'expédition et validé à la réception ; il serait possible d'avoir seulement un état d'expédition et un état de réception. De cette manière, le contrat intelligent peut valider la réception avec le destinataire et le prochain état avec l'état précédant la réception. Cette méthode offrirait plus de flexibilité au niveau des opérations (par exemple si deux partenaires exécutent des

Cette méthode est couplée avec une méthode de réception chez le destinataire et ignore l'existence d'intermédiaires. Pour la durée entière de l'expédition, le détenteur de l'actif (identifié avec la variable d'actif `owner`) indique que l'actif est toujours sous la responsabilité de celui-ci. Présentement, l'expéditeur est le détenteur de l'actif jusqu'au moment de sa réception. Advenant le cas où un intermédiaire est employé pour le transport, un état intermédiaire `TRANSIT` existerait entre l'état d'expédition et l'état de réception. L'organisme intermédiaire de transport deviendrait aussi le détenteur temporaire de l'actif, permettant une traçabilité plus complète de la chaîne d'approvisionnement.

Ensuite, la méthode de réception (présentée dans l'extrait de code 4.13) est appelée, indiquant que l'organisme destinataire a bien reçu l'actif. À cette étape, le destinataire de l'opération est validé afin de s'assurer que l'organisme employant la méthode est bien le destinataire indiqué par le contrat intelligent lors de l'expédition. Ensuite, le détenteur de l'actif est mis à jour. Lors de la mise à jour de l'état à `RECEIVED`, le champ de destinataire est effacé, indiquant que le produit n'est plus en état d'expédition. À ce moment, le destinataire est en possession de l'actif et peut commencer ses traitements.

traitements différents nécessitant des états d'actif différents) au coût de la simplicité apportée par un chemin totalement linéaire.

```
@access(AppleContract.permittedProcessed, ['reception'])
async received_P(ctx: AppleContext, picker: string, appleNum: Number,
    storageTemp: Number) {
    // Retrieve the current apple using key fields provided
    let apple = await this.getCurrentApple(ctx, picker, appleNum);

    apple.setStorageTemp(storageTemp);

    let newDestination = apple.getDestination();
    // Validate current holder is the destination
    if ((newDestination.charAt(0).toUpperCase() + newDestination.
       slice(1) + 'MSP') !== ctx.clientIdentity.getMSPID()) {
        throw new Error('Apple ' + picker + appleNum + ' belongs to '
            + newDestination);
    }

    // received moves state from outbound to received
    if (apple.isOutbound_B()) {
        apple.setHolder(newDestination)
        apple.setReceived_P();
    } else {
        throw new Error('Apple ' + picker + appleNum + ' is not
           outbound. Current state = ' + apple.getCurrentState());
    }

    ctx.stub.setEvent("ReceiveAppleFromB", apple.toBuffer());

    // Update the apple
    await ctx.appleList.updateApple(apple);

    console.log('Received apple ' + picker + ' ' + appleNum);

    return apple;
}
```

Listing 4.13: Fonction `received_P` (`applecontract.ts`)

Toutes ces fonctions sont supportées par des fonctions de requête, dont `getCurrentApple`, qui permet d'extraire un actif de lot de pommes avec son identifiant, et `getAppleHistory`, qui permet d'extraire la liste de changements d'état d'un lot de pommes.

En résumé, le contrat intelligent `AppleContract` est un outil puissant pour la manipulation des données, permettant une traçabilité complète de la production des lots de pommes. Les contrats intelligents bien définis ont le potentiel de grandement améliorer la collaboration entre les organismes dans le but d'augmenter la trans-

parence. Cependant, il y a certaines améliorations possibles dans l'architecture des contrats intelligents présents.

4.4.3 Améliorations possibles

Parmi les améliorations possibles, il pourrait être plus judicieux de séparer le *chaincode* en quatre contrats intelligents : un contrat pour les requêtes, qui sont disponibles à tous ; un contrat pour les fonctions de l'organisme *Brute*, un contrat pour l'organisme *Processed* et un contrat pour l'organisme *Shipping*. De cette manière, le contrôle des accès organisationnel pourrait être fait au niveau du contrat intelligent plutôt que dans les fonctions. Les contrats pourraient aussi offrir des fonctions de requête pouvant être appelées par les organismes suivants dans la chaîne d'approvisionnement. De cette manière, les opérations seraient mieux encapsulées dans leurs organismes respectifs (ceci remplacerait le contrat de requêtes).

Une autre solution serait de séparer le réseau en plusieurs *channels* plutôt qu'un seul. Chaque partenariat aurait son propre *channel*. Cette méthode pourrait mieux encapsuler les partenariats entre organismes (tels que le partenariat entre *Brute* et *Processed* et le partenariat entre *Processed* et *Shipping*), afin de mieux représenter les ententes actuelles (par exemple, *Brute* et *Shipping* ne sont pas en partenariat et donc n'ont pas intérêt à partager un *channel*)[9]. Cette méthode permettrait aussi un contrôle plus granulaire sur le réseau, simplifiant l'architecture et facilitant l'administration du réseau pour les organismes participants. Comme dans le cas des contrats multiples sur un seul *channel*, les *chaincodes* d'organismes sur chaque *channel* pourraient comporter des fonctions de requête permettant aux organismes suivants dans la chaîne d'approvisionnement de visualiser les informations du registre dans le *channel* qui les précède. Par exemple, l'organisme *Shipping* pourrait faire appel à la fonction

[9]Les organismes ont un intérêt à protéger leurs informations et risquent de ne pas vouloir rendre leur logique métier visible à des organismes qu'ils ne connaissent pas.

`getAppleHistory` du contrat `ShippingAppleContract` dans le *chaincode Shipping* (qui existe dans un *channel* dédié à l'expédition de cet organisme). Cette fonction fera ensuite appel à une fonction `getAppleHistory` du contrat `ProcessedAppleContract` du *chaincode Processed* du *channel* géré par l'organisme *Processed* dans le *channel* géré par cet organisme (qui pourrait, par la suite, appeler une fonction du *chaincode* de l'organisme *Brute*). Chaque organisme pourrait aussi avoir une fonction distincte pouvant être appelée de l'extérieur (par exemple `getPartialAppleHistory` plutôt que `getAppleHistory`) qui retourne seulement certaines informations non privées. La fonction pourrait, par exemple, retourner les différents états de l'actif ainsi que les détenteurs précédents, mais pas les noms des machines utilisées dans leur traitement. De cette manière, les opérations seraient totalement séparées des non-partenaires ce qui permettrait un contrôle total des accès aux informations au niveau du réseau.

4.5 Les applications

Les organismes du réseau AppleNet requièrent une interface par laquelle leurs acteurs peuvent communiquer et interagir avec le registre. Afin d'atteindre cet objectif, un serveur utilisant le *framework* Node.js[10], *Express*[11], a été créé. En utilisant des *frameworks* Node.js dans le développement de l'application, on peut uniformiser les langages de programmation à travers le projet en utilisant exclusivement du Typescript[12].

Les tâches de l'application web cliente ont été séparées en deux niveaux : le plancher et l'administration.

[10]https ://nodejs.org/en/ consulté en août 2022

[11]https ://expressjs.com/ consulté en août 2022

[12]Dans un projet réel, l'équipe de développement de contrats intelligents risque d'être différente de celle qui développe les applications étant donné que les contrats intelligents sont partagés sur le réseau et gèrent la validation et la persistance de l'information alors que les applications sont souvent sous la responsabilité des organismes (mais pas obligatoirement) et gèrent l'entrée et la consultation des informations.

Pour l'instant, toutes les fonctions des contrats intelligents sont dites du plancher. Ces fonctions concernent tout ce qui a été discuté plus tôt incluant les opérations de traitement, d'expédition et de réception de produit. Toutes les fonctions de requête sont aussi disponibles pour les utilisateurs du plancher, cependant, un contrôle d'accès plus strict est planifié comme amélioration future afin de restreindre la visibilité de l'information selon les tâches prévues (tel que mentionné dans la section précédente). Du côté de l'administration, les fonctionnalités sont limitées à la capture d'événements générés par le contrat intelligent `AppleContract`, permettant d'avoir une vue en temps réel de la chaîne d'approvisionnement.

4.5.1 L'application du plancher

L'application de démonstration possède plusieurs éléments principaux. Premièrement, l'application possède un menu déroulant permettant à l'utilisateur de changer d'organisme (*Brute*, *Processed*, *Shipping*), ainsi qu'un menu déroulant permettant de changer d'identité (identifié avec les rôles `picker`, `reception`, `processing`, `shipping`). Cet aspect permet de simuler les changements d'utilisateur et de pouvoir agir selon leurs différentes permissions. Deuxièmement, l'application possède une section permettant de créer des transactions. Chaque fonction du contrat intelligent `AppleContract` est représentée sur la page de l'application. Ces fonctions sont disponibles par l'entremise d'une API fourni par le serveur hébergeant l'application. Troisièmement, l'application possède une section permettant d'afficher les résultats retournés par les requêtes, organisés afin de faciliter la visibilité.

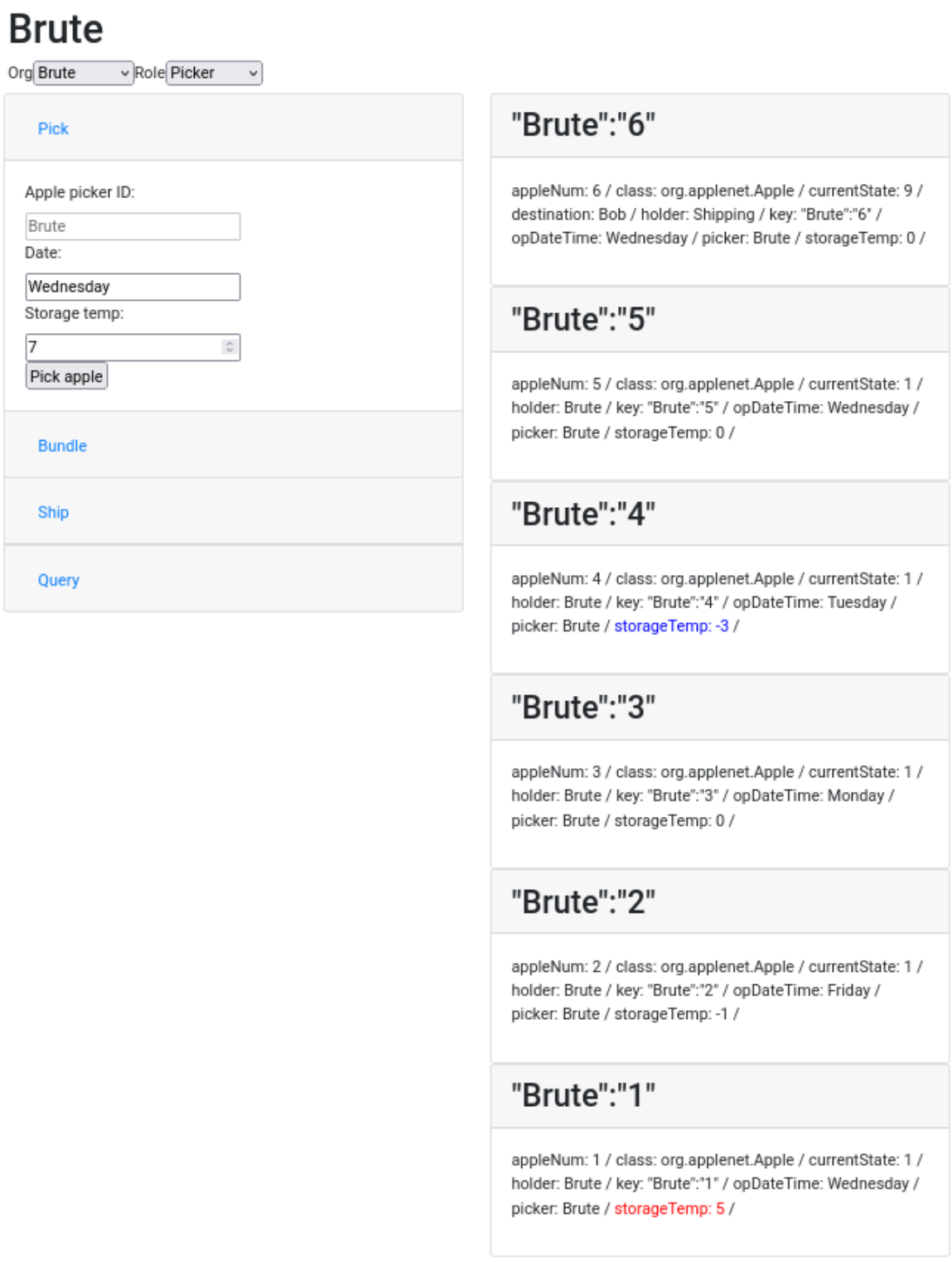

FIGURE 4.7 : Capture d'écran de l'application web de l'organisme *Brute* communiquant avec le réseau AppleNet.

La figure 4.7 présente une capture d'écran montrant l'application de base de requêtes et d'entrée d'informations créée pour communiquer avec le réseau *blockchain*. Dans la portion de gauche, l'application comporte quatre fonctions (ainsi que ses équivalents chez les autres organismes), soit la fonction de cueillette de pommes (qui est remplacée par une fonction de réception de lot chez les organismes *Processed* et *Shipping*), la fonction de traitement (`bundle`), la fonction d'expédition (`Ship`) et la fonction de requête de liste d'actifs (`Query`). L'application identifie les températures trop faibles en bleu et trop élevées en rouge dans la liste d'actifs présentés à droite. L'application choisit automatiquement l'identifiant du lot ajouté et envoie une notification à l'utilisateur présentant le résultat de la transaction. Si l'identité utilisée n'est pas une identité de l'organisme *Brute* (par exemple, l'identité `reception` de l'organisme *Processed*), le contrat intelligent retournera une erreur due à la validation du décorateur `@access`.

Ces fonctionnalités sont simplifiées afin de permettre de mieux visualiser les opérations dans l'expérimentation. Cependant, dans un cas réel, chaque fonction comporterait sa propre page dans le logiciel de l'organisme. Ces processus pourraient aussi être automatisés (ou semi automatisés) sans requérir d'intervention humaine supplémentaire, puisque l'application peut être intégrée à tout logiciel que ce soit directement dans l'application ou par l'entremise d'une API. Dans ces cas, l'identité peut être assignée à la machine (s'il s'agit d'une automatisation), au logiciel exécuté, ou à l'opérateur de la machine (si l'opération de la machine requiert une identification).

4.5.2 L'application administrative

Présentement, l'application administrative comporte une seule fonction de capture d'événements permettant aux utilisateurs de suivre les opérations à travers le réseau. La capture d'événements cible tous les événements générés par le contrat intelligent

`AppleContract`. Ces événements sont générés à la fin de chaque appel de fonction modifiant l'état d'un actif sur le registre, permettant aux administrateurs d'avoir une vue de la chaîne d'approvisionnement en temps réel. Une telle application pourrait aussi déclencher des événements sous certaines conditions (telles que des températures de stockage inacceptables) et offrir des actions à prendre selon les données recueillies. Il serait aussi possible d'effectuer des traitements de données afin de générer des rapports de production permettant de trouver les points faibles dans la chaîne d'approvisionnement.

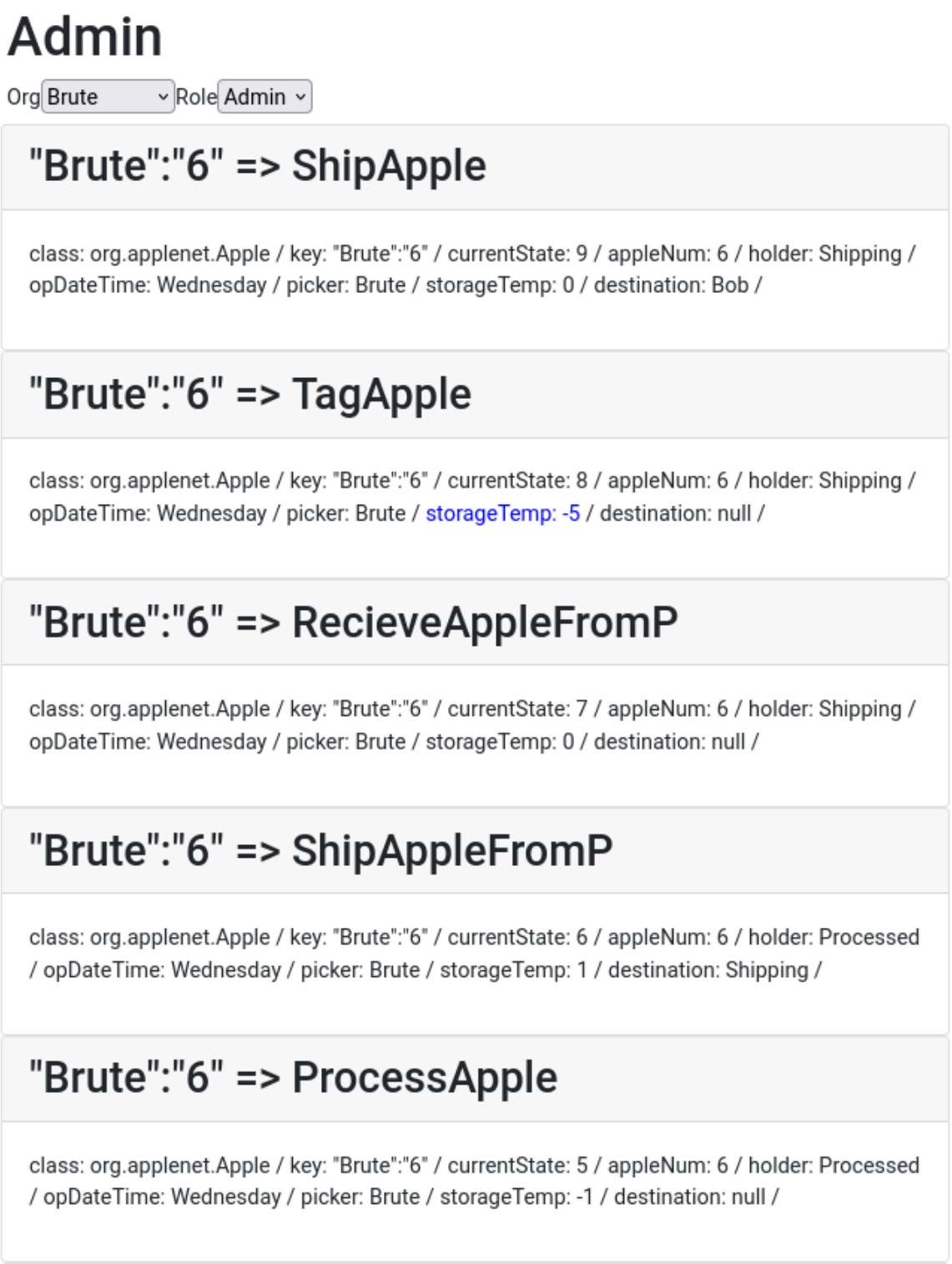

FIGURE 4.8 : Capture d'écran de l'application web pour les administrateurs.

Dans la capture d'écran ci-dessus (figure 4.8), les événements générés par le contrat intelligent `AppleContract` sont affichés à l'écran. Les couleurs avertissent des températures de stockage trop faibles (bleu) ou trop élevées (rouge) comme dans l'application du plancher.

Dans ce cas, tous les organismes et leurs administrateurs ont les mêmes droits sur le *channel*, permettant à tous les administrateurs d'avoir une visibilité sur toutes les opérations. Ceci n'est pas nécessairement désirable tel que mentionné plus tôt puisque, par exemple, les membres de l'organisme au début de la chaîne d'approvisionnement (dans ce cas, l'organisme *Brute*) ne devraient pas avoir accès aux informations des organismes suivants (*Processed* et *Shipping*) puisqu'ils n'ont pas besoin de ces informations pour leurs modèles de traçabilité. Cela encouragerait donc le consortium à utiliser un modèle de conception de réseau avec au minimum des contrats intelligents différents s'étendant possiblement à l'utilisation de *channels* différents comme discuté plus tôt. Dans un tel cas, il serait aussi possible d'utiliser un *channel* dit *global* permettant de générer des événements (ainsi que modifier un registre) partagés afin que tous les membres du consortium puissent suivre les changements de la chaîne d'approvisionnement.

4.6 Difficultés

De nombreuses difficultés se sont présentées lors de ce projet. Malgré la nature complexe de l'architecture du réseau ainsi que les nombreuses couches d'abstraction, la plus grande source de difficultés a été au niveau de la disponibilité et de la visibilité de l'information.

Une contrainte majeure, qui a été la source de nombreuses difficultés au courant du développement du projet, est le manque d'informations claires pour le développement

d'un projet dans la documentation de *Hyperledger Fabric*. En effet, malgré la disponibilité de nombreux tutoriels dans la documentation officielle détaillant les fonctionnalités de la plateforme, tous ces tutoriels s'exécutent à partir de projets déjà existants. Dans tous ces projets, l'architecture complète du consortium est déjà définie et disponible dans le répertoire git officiel de Hyperledger[13]. Dans ce cas, les tutoriels sont d'excellentes démonstrations du fonctionnement de *Hyperledger Fabric* et des fonctionnalités de ses contrats intelligents. Cependant, il n'y a aucune présentation des étapes de création de l'architecture de ces projets à partir du début de ceux-ci. Ceci entraîne une difficulté entre la phase de conception et la phase de développement de contrats intelligents. Il y a un vide dans la documentation entre ces deux étapes requérant une étude beaucoup plus avancée des technologies afin de pouvoir l'implémenter. Cet aspect peut être vu comme étant symptomatique de la nouveauté du système, entraînant une courbe d'apprentissage abrupte pour les architectes de réseau *Hyperledger Fabric*. Afin de pallier ce problème et de monter une démonstration rapide, les fichiers de configurations des tutoriels ont été utilisés comme base de configuration du réseau, ainsi que des scripts de démarrage écrits en *Bash*. Ces fichiers ont ensuite été modifiés en renommant les noms des organismes, en ajoutant des fonctionnalités de base supplémentaires et en corrigeant certaines difficultés telles que des attentes d'exécution de processus lents excédant les temps d'attente par défaut (*timeouts*). Le script de démarrage a aussi été modifié afin d'ajouter la création d'identités par défaut par des scripts en JavaScript utilisant l'API de *Hyperledger Fabric*.

La dépendance envers les scripts au démarrage est aussi due au manque d'outils d'administration internes à l'écosystème *Hyperledger Fabric*. Effectivement, alors qu'il y a une API permettant de gérer les identités, il n'y a pas d'outils facilitant l'administration du réseau *Hyperledger Fabric*. Ce manque d'abstraction entraîne une obscurité non nécessaire dans la gestion d'un réseau. Une telle abstraction existait pour les versions précédentes de *Hyperledger Fabric* sous le nom de *Hyperledger Composer*[Hyperledger, 2019a], mais est identifiée comme étant obsolète depuis 2019. Des

[13]https ://github.com/hyperledger/fabric-samples consulté en août 2022

opérations telles que l'application d'un changement aux configurations du réseau (qui peut être aussi simple qu'ajouter un organisme à un *channel*) doivent être effectuées en plusieurs étapes en ligne de commande. Alors que ce sont des opérations considérées comme routinières. Les développeurs sont donc emmenés à créer des scripts afin d'exécuter les opérations de gestion de réseau plus facilement. En résumé, les outils de gestion de *Hyperledger Fabric* tels que présentés dans les tutoriels comportent seulement des opérations de bas niveau sous forme d'exécutables requérant aux utilisateurs d'avoir une connaissance plus avancée des outils, ce qui entraîne le développement de scripts qui peuvent être difficiles à lire et comprendre[14]. Il serait beaucoup plus facile au niveau du développement d'avoir accès à des fonctions de haut niveau exécutant une suite d'opérations communes (telles que la modification et la mise à jour des configurations du réseau). Il pourrait aussi y avoir des APIs permettant d'exécuter ces opérations par l'entremise d'une application afin d'abstraire les opérateurs de la ligne de commande et fournir une interface qui représente mieux l'état du réseau.

Une autre difficulté rencontrée lors du développement a été la complexité des fichiers de configuration. En effet, les fichiers de configuration[15], écrits en YAML, ont de nombreuses responsabilités (dont la liste de permissions d'organismes), l'emplacement de leur MSPs, leurs permissions (lecture, écriture, administration) et les adresses de leurs nœuds. Cela rend la configuration d'un réseau très complexe[16]. Le rassemblement de ces différents éléments complexes en un seul fichier favorise la centralisation des informations de configuration au détriment de la compartimentation qui faciliterait la lisibilité et la modification des configurations. Alors qu'une architecture de configurations centralisées simplifie grandement la gestion et la vue des fichiers d'un projet *Hyperledger Fabric*, elle ajoute aussi une complexité à la gestion interne des fichiers

[14]Le script de démarrage du projet AppleNet comporte environ 570 lignes de code *Bash* modifiées (soit en commentaire ou en modifications plus directes) du script de démarrage `network.sh` du tutoriel de *Hyperledger Fabric*. Certaines lignes ne sont pas utilisées, mais la complexité du script décourage les changements importants.

[15]Dans ce cas nous avons principalement étudié les configurations d'un *channel* mais il existe aussi des configurations de réseau ainsi que des configurations de nœud d'ordonnancement

[16]Normalement, une équipe de développement d'un projet *blockchain* est séparée en plusieurs rôles tels que l'administration du réseau (qui inclut la configuration du réseau et la gestion des nœuds), le développement de contrats intelligents et le développement des applications.

en réduisant sa lisibilité. La variété des configurations dans le fichier peut rendre la tâche de gestion très difficile pour un administrateur et peut confondre les besoins des différents organismes. *Hyperledger Composer* (marqué obsolète) comportait une fonctionnalité permettant de définir des listes de contrôle d'accès (ACL) dans des fichiers distincts qui permettait de définir les listes d'organismes et d'utilisateurs pouvant accéder aux fonctions de contrat intelligent. Dans Hyperledger Fabric tel qu'il est présentement, les ACLs sont définies dans les mêmes fichiers de configurations que les configurations de *channel*. Il pourrait donc être intéressant de permettre aux utilisateurs de définir des groupements de configurations dans des fichiers individuels (par exemple, un fichier pour l'inclusion des organismes et un autre fichier pour le contrôle d'accès au *channel*). Le fichier de configurations principal pourrait donc contenir une référence vers ces fichiers dans le cas où les configurations n'y sont pas définies directement. De cette manière, la complexité des fichiers de configuration et leurs contenus se limiteraient à ce que l'architecte prévoit, facilitant la gestion selon les préférences des équipes de développement. Cette approche pourrait aussi être faite par l'entremise d'un prétraitement des fichiers afin de les rassembler en un seul fichier principal avant la mise à jour.

Un autre aspect qui a rendu le développement difficile est la nouveauté de la technologie. En effet, le domaine de la *blockchain* est encore relativement nouveau et en constante évolution. *Hyperledger Fabric* subit donc des mises à jour fréquentes et a récemment subi un changement de version important (de *Hyperledger Fabric* 1.4 à *Hyperledger Fabric* 2.X en 2020)[Hyperledger, 2020h]. Il y a donc de la documentation pour les deux versions majeures disponibles, cependant, les résultats de recherche sont souvent mélangés et ne sont pas toujours identifiés par version. Donc, certaines réponses qui fonctionnaient dans la version 1.4 ne fonctionnent plus dans la version 2.X, mais celles-ci ne sont pas identifiés clairement comme étant pour la version 1.4 (tel que le contrôle des accès avec *Hyperledger Composer*). La documentation du SDK en JavaScript de la version 1.4 est souvent priorisée par les moteurs de recherche due, probablement, à son ancienneté et à son utilisation plus fréquente (puisque cette

version est déjà établie dans plusieurs réseaux). Puisqu'il s'agit d'une technologie nouvelle utilisée au niveau industriel, il peut aussi être difficile de trouver des réponses aux questions survenant lors du développement puisque des exemples concrets de ces situations ne sont pas disponibles (que ce soit par protection d'informations privées ou parce que la question n'a jamais été posée).

En somme, la récence et l'évolution constante de la technologie de la *blockchain* et, par extension, *Hyperledger Fabric*, peuvent causer des difficultés interreliées sur deux niveaux : les changements rapides dans la documentation et les changements constants dans l'architecture et l'implémentation. D'une part, les changements dans la documentation peuvent causer des difficultés de recherche d'informations due à des éléments tels que la récence, la popularité de recherches et la similarités aux versions antérieures qui rendent les résultats désirés moins prévalents dans les moteurs de recherche, d'autre part, les changements de version modifiant l'implémentation de la technologie apportent des difficultés au niveau de la recherche de réponses aux questions d'implémentation dans les forums publics. Les changements importants dans les technologies utilisées peuvent aussi parfois enlever certaines fonctionnalités importantes (cas de *Hyperledger Composer*)[17].

4.7 Améliorations possibles

Il existe plusieurs améliorations possibles dans le projet AppleNet tel qu'il est présentement.

[17]Les autres technologies *blockchain* telles qu'*Ethereum* souffrent des mêmes difficultés d'évolution rapide. Dans le cas d'*Ethereum*, il est même requis d'indiquer explicitement les versions de *Solidity* (le langage de programmation utilisé pour les contrats intelligents d'*Ethereum*) afin que le contrat intelligent puisse compiler. Ces changements affectent tous les aspects de la programmation, par exemple, avant la version 0.8.0 de *Solidity*, les *overflows* et *underflows* de nombres entiers n'étaient pas vérifiés (requérant une validation explicite par le développeur)[Ethereum, 2020]. La modification du comportement par défaut de la validation de nombres entiers est un changement important et tout développeur cherchant à adapter son code doit être conscient des possibilités de différence de version lors de ses recherches.

Premièrement, l'organisation des fichiers doit être remodelée afin de mieux distinguer les différentes portions du système. Les fichiers du projet ont deux grandes distinctions : la portion application et la portion *blockchain*. La portion application est majoritairement acceptable dans son organisation, malgré le fait qu'elle réside dans le même répertoire que les fichiers de la portion *blockchain*. Cependant, la portion *blockchain* elle-même comporte certaines faiblesses organisationnelles dans l'implémentation du projet dans un contexte de simulation. La difficulté principale du projet provient de la gestion centralisée des organismes. En effet, dans le projet AppleNet, les organismes, incluant leurs informations de démarrage et leurs informations cryptographiques, sont tous groupés dans le même répertoire afin de simplifier la tâche de gestion d'une preuve de concept. Afin de mieux simuler un réseau *blockchain* multi-organisationnel, il faut abandonner l'architecture proposée dans les tutoriels de *Hyperledger Fabric* (qui favorise un réseau sur une seule machine) en séparant tous les fichiers par organisme, avec un répertoire pour l'organisme d'ordonnancement, un répertoire pour l'organisme *Brute* et ainsi de suite. De cette manière, l'organisation des fichiers sera plus représentative de la réalité avec les nœuds répartis sur plusieurs machines[18]. Avec cette organisation, un script pourrait quand même être utilisé pour la gestion centralisée et les développeurs pourraient facilement installer les différents organismes sur des machines distinctes et configurer le réseau pour fonctionner avec plusieurs machines physiques, créant une simulation beaucoup plus proche de la réalité.

Deuxièmement, comme mentionné dans la section sur les contrats intelligents, il faut idéalement séparer les organismes en plusieurs *channels* selon leurs intérêts opérationnels. En effet, dans l'architecture présente, les organismes *Brute* et *Shipping* n'ont aucune relation directe. Il est donc important de limiter les informations partagées à ceux uniquement nécessaires au fonctionnement du réseau, plutôt qu'une

[18]Il est possible d'avoir des *clusters* de nœuds sur une seule machine et une entreprise *blockchain* pourrait être responsable de regroupements de nœuds de différentes entreprises. Cependant, il est toujours attendu d'avoir des séparations physiques géographiques des nœuds afin de mieux sécuriser le réseau (les nœuds d'ordonnancement en particulier). Il est donc important pour des fins de simulation de séparer les différents organismes (et possiblement leurs nœuds) de manière clairement définie.

vue entière du *channel*. De plus, puisque les organismes ont un intérêt généralement limité par rapport aux informations d'un produit plus loin dans la chaîne d'approvisionnement, une vue limitée aux informations antécédentes dans la chaîne d'approvisionnement serait à prioriser. Cette architecture est possible par l'utilisation de contrats intelligents capables d'exécuter des appels sur des contrats sur d'autres *channels* (les *channels* existant plus tôt dans la chaîne d'approvisionnement). En créant des fonctions avec une vue en arrière, il est possible de faire des appels en cascade permettant de visualiser l'historique entier d'un actif. Inversement, il est possible de créer une fonction permettant de propager un événement signalant un produit fautif ou contaminé aux organismes plus tard dans la chaîne d'approvisionnement (permettant d'avertir tous les clients d'un rappel ciblé). Cette fonction pourrait être créée soit avec un *channel* global partagé par tous les organismes soit avec des appels en cascade par l'entremise de contrats intelligents.

Troisièmement, le projet devrait posséder une meilleure gestion des identités. En effet, présentement, le projet ne supporte que le changement de compte par un menu déroulant plutôt que par une connexion des comptes utilisateurs à l'entrée du site. Cette étape de connexion et de préservation de session utilisateur est essentielle à une simulation de système de gestion de traçabilité. L'utilisation d'identités est un aspect fondamental des réseaux *blockchain* privés. Dans le cas des tests courants, les menus déroulants ont été utilisés afin de pouvoir facilement faire des changements au registre et valider le fonctionnement des décorateurs d'accès. Cependant, dans une preuve de concept ou une simulation plus avancée, il sera nécessaire de montrer la gestion d'identités de manière plus explicite.

Quatrièmement, le projet a besoin d'outils de gestion de réseau plus avancés et faciles à utiliser. Comme mentionné précédemment, *Hyperledger Composer*, l'outil de gestion de réseau de *Hyperledger*, est identifié comme étant obsolète et il est donc recommandé de s'en distancier si possible. Il sera donc important, afin de faciliter la gestion du réseau par tous les administrateurs, de créer un outil permettant la vi-

sualisation simplifiée des nœuds et de leurs configurations sur le réseau (incluant les configurations des *channels* impliqués). Il faudra aussi un outil pour les gestionnaires (des utilisateurs qui ne touchent pas au réseau, mais qui possèdent des permissions de gestion d'utilisateurs) leur permettant de créer et modifier les *wallets* des utilisateurs. De cette manière, ceux-ci pourront manipuler les identités afin que les employés puissent participer sur le réseau avec des permissions adaptées à leurs permissions réelles. Dans le projet, il pourrait donc y avoir un portail pour l'administration du réseau (incluant des vues des journaux de chaque nœud géré), un portail pour la gestion des utilisateurs et un logiciel de gestion de la chaîne d'approvisionnement intégré dans le logiciel de gestion d'inventaire. Ces trois outils pourraient aussi être combinés ou partager des fonctionnalités dépendamment des besoins.

Cinquièmement, il pourrait être intéressant d'implémenter une détection automatique des fonctions et de leurs permissions plutôt que les programmer individuellement dans le portail. En effet, *Hyperledger Fabric* fournit des fonctions de contrat intelligent par défaut, permettant de détecter certaines informations, dont les métadonnées contenues dans les contrats par l'entremise de la transaction `org.hyperledger.fabric:GetMetadata`. Ces métadonnées incluent les fonctions des contrats intelligents, le type de fonction (lecture ou écriture), ainsi que les paramètres (et leurs types) associés. Cela permet de développer des applications flexibles qui peuvent automatiquement détecter les fonctions des contrats intelligents qui leur sont accessibles sur le réseau. Une telle application pourrait donc générer une interface pour chaque contrat intelligent qui se met à jour automatiquement par rapport aux changements effectués du côté du réseau (donc, sans devoir mettre à jour le code de l'application). Un autre aspect intéressant serait l'utilisation de décorateurs/annotations afin d'identifier les accès. L'application pourrait potentiellement se servir de ces décorateurs afin d'afficher ou de cacher les fonctions détectées par l'entremise des métadonnées de manière automatique.

En somme, les améliorations principales désirées pour le projet AppleNet sont

au niveau structural et organisationnel, afin de mieux représenter la simulation, avec quelques améliorations au niveau de la gestion, permettant de faciliter l'utilisation du projet au niveau des utilisateurs et au niveau des développeurs. Dans le futur, il serait intéressant de réviser les configurations du réseau afin d'améliorer ses performances. Ces révisions de configuration pourraient inclure la taille et la fréquence de création de blocs. Elles pourraient aussi inclure des permissions plus réalistes dans, par exemple, la spécification des droits des différents acteurs (nœuds, administrateurs, clients) et imposer des permissions plus restrictives sur les droits de modification (restreindre les droits de modification à une sélection spécifique d'organismes plutôt que la majorité simple). Ces améliorations permettront de créer une simulation beaucoup plus complète d'un système de traçabilité en *blockchain*.

Chapitre 5

Conclusion

Dans ce mémoire, nous avons étudié des cas existants utilisant la *blockchain* et leurs justifications, différencié les attributs de la *blockchain* et ses différentes implémentations courantes et développé une simulation de situation nécessitant la *blockchain*. Dans les études de cas, un des avantages principaux de la *blockchain* qui a été présenté était la collaboration entre différents organismes.

La simulation AppleNet permet une traçabilité de lots de pommes dans une chaîne d'approvisionnement à trois organismes. Cette traçabilité est possible grâce à un réseau *Hyperledger Fabric* partagé par tous les organismes. Les manipulations des actifs sur le réseau sont effectués par les utilisateurs avec une application web permettant d'ajouter, modifier et visualiser les états de chaque actif du réseau. Cette application web interface avec un contrat intelligent possédant des méthodes de vérification et de validation de données. Ces méthodes effectuent non seulement des tâches de validation d'entrée d'information (identifiant de lot, d'état, etc.), mais effectue aussi le contrôle d'accès. Nous avons donc créé décorateur `@access` afin de simplifier le code de contrat intelligent en abstrayant la couche d'accès de la couche de manipulation de données.

Au courant du développement de cette simulation, nous avons fait plusieurs constatations quant à l'état des ressources d'apprentissage. En effet, alors que la documentation présente des exemples très claires de l'utilisation de *Hyperledger Fabric*, il y a un fort manque de ressources claires démontrant le processus de développement complet d'un réseau, ce qui augmente grandement la courbe d'apprentissage de la technologie. Il y a aussi un manque de ressources démontrant le développement de projets réels avec *Hyperledger Fabric* qui permettraient aux initiés de mieux comprendre comment appliquer la *blockchain* au niveau conceptuel et au niveau du développement.

La *blockchain* a un grand potentiel dans un environnement globalisé où tous les organismes sont interconnectés. Les aspects distribués et immuables de la *blockchain* sont très avantageux dans des domaines requérant une transparence entre les différents participants. Ces aspects seuls ont des implications intéressantes quant aux domaines de la cryptomonnaie et de la possibilité d'une gouvernance décentralisée. En ajoutant l'aspect d'identité à la *blockchain*, que ce soit avec des librairies de gestion d'identité sur *Ethereum* ou des *frameworks* tels que *Hyperledger Fabric*, il est possible de créer un système de permissions propice au développent de projets plus complexes où il est essentiel d'identifier les différents acteurs. L'environnement permissionné des *blockchains* tel que *Hyperledger Fabric* est un excellent vecteur pour les solutions industrielles de traçabilité. Il est évident, par le succès des cas d'implémentation de *Hyperledger Fabric* étudiés, que la *blockchain* a un grand potentiel pour non seulement augmenter les performances industrielles, mais aussi améliorer les relations entre les organismes et leur confiance mutuelle en favorisant le partage d'informations équitable.

En somme, la technologie *blockchain* est à un point crucial dans son développement ; elle a été popularisée par les plateformes publiques telles que *Bitcoin* et *Ethereum* et voit son potentiel de partage de données immuables exploité de manière très intéressante dans les *blockchains* privées telles que *Hyperledger Fabric*. Il n'est qu'une question de temps avant de voir la *blockchain* comme solution potentielle dans la majorité des projets interorganisationnels.

Références

[Academy, 2018] Academy, B. (2018). Byzantine fault tolerance explained. Repéré à `https://academy.binance.com/en/articles/byzantine-fault-tolerance-explained` ; Consulté en septembre 2022.

[Adhav, 2020] Adhav, P. (2020). Attribute-based access control (abac) in hyperledger fabric. *Medium*. Repéré à `https://medium.com/coinmonks/attribute-based-access-control-abac-in-hyperledger-fabric-1eb81330f67a` ; Consulté en août 2022.

[Back, 2002] Back, A. (2002). Hashcash - a denial of service counter-measure. *http ://www.hashcash.org/*. Repéré à `http://www.hashcash.org/` ; Consulté en août 2022.

[Bitcoin, 2013] Bitcoin (2013). Bitcoin development. Repéré à `https://bitcoin.org/en/development` ; Consulté en août 2022.

[Blockchain.com, 2018] Blockchain.com (2018). Bitcoin explorer. Repéré à `https://www.blockchain.com/explorer` ; Consulté en août 2022.

[Bootcamps, 2021] Bootcamps, C. (2021). Ethereum and know-your-customer requirements. Repéré à `https://www.coding-bootcamps.com/blog/ethereum-and-know-your-customer-requirements.html` ; Consulté en août 2022.

[Buhr, 2003] Buhr, B. (2003). Traceability and information technology in the meat supply chain : Implications for firm organization and market structure. *Journal*

of Food Distribution Research, 34. doi : http://dx.doi.org/10.22004/ag.econ.27057.

[Buterin, 2014] Buterin, V. (2014). Ethereum : A next-generation smart contract and decentralized application platform. Repéré à https://ethereum.org/en/whitepaper/ ; Consulté en août 2022.

[Buterin, 2021] Buterin, V. (2021). The limits to blockchain scalability. Repéré à https://vitalik.ca/general/2021/05/23/scaling.html ; Consulté en août 2022.

[CDC, 2018] CDC (2018). Multistate outbreak of e. coli o157 :h7 infections linked to romaine lettuce (final update). Repéré à https://www.cdc.gov/ecoli/2018/o157h7-04-18/index.html ; Consulté en août 2022.

[Cointelegraph, 2021] Cointelegraph (2021). How does blockchain solve the byzantine generals problem ? Repéré à https://cointelegraph.com/blockchain-for-beginners/how-does-blockchain-solve-the-byzantine-generals-problem ; Consulté en septembre 2022.

[Consensys, 2022] Consensys (2022). Reentrancy. Repéré à https://consensys.github.io/smart-contract-best-practices/attacks/reentrancy/ ; Consulté en août 2022.

[contributors, 2004] contributors, W. (2004). Universally unique identifier. Repéré à https://en.wikipedia.org/wiki/Universally_unique_identifier ; Consulté en août 2022.

[de la santé publique du Canada, 2018] de la santé publique du Canada, A. (2018). Avis de santé publique - Éclosion d'infections à e. coli associées à la laitue romaine. Repéré à https://www.canada.ca/fr/sante-publique/services/avis-sante-publique/2018/avis-sante-publique-eclosion-infections-e-coli-associees-laitue-romaine.html ; Consulté en août 2022.

[Dickinson et Bailey, 2002] Dickinson, D. L. et Bailey, D. (2002). Meat traceability : Are u.s. consumers willing to pay for it ? *Journal of Agricultural and Resource Economics*, 27(2):348–364. doi : http://dx.doi.org/10.22004/ag.econ.31128.

[Ethereum, 2018] Ethereum (2018). Solidity releases. Repéré à https://github.com/ethereum/solidity/releases ; Consulté en août 2022.

[Ethereum, 2020] Ethereum (2020). Solidity v0.8.0 breaking changes. Repéré à https://docs.soliditylang.org/en/v0.8.0/080-breaking-changes.html ; Consulté en août 2022.

[Etherscan, 2015] Etherscan (2015). Etherscan. Repéré à https://etherscan.io/ ; Consulté en août 2022.

[Etherscan, 2022] Etherscan (2022). Average transaction fee chart. Repéré à https://etherscan.io/chart/avg-txfee-usd ; Consulté en août 2022.

[Foundation, 2022] Foundation, E. (2022). Ethereum for everyone. Repéré à https://ethereum.org/en/layer-2/ ; Consulté en septembre 2022.

[Foundation, 2023] Foundation, E. (2023). The merge. Repéré à https://ethereum.org/en/upgrades/merge/ ; Consulté en février 2023.

[Foundation, 2015] Foundation, E. . J. . (2015). Ethereum launches. Repéré à https://blog.ethereum.org/2015/07/30/ethereum-launches ; Consulté en août 2022.

[Giancaspro, 2017] Giancaspro, M. (2017). Is a 'smart contract' really a smart idea ? insights from a legal perspective. *Computer Law & Security Review*, 33(6):825–835. doi : https://doi.org/10.1016/j.clsr.2017.05.007.

[Heneghan, 2016] Heneghan, C. (2016). Could whole chain traceability be the answer to food safety and transparency ? Repéré à https://www.supplychaindive.com/news/whole-chain-traceability-transparency-food-safety/432175/ ; Consulté en août 2022.

[Himanshi, 2022] Himanshi (2022). Byzantine fault tolerance (bft) in blockchain. Repéré à `https://www.naukri.com/learning/articles/byzantine-fault-tolerance-in-blockchain/` ; Consulté en septembre 2022.

[Hyperledger, 2017] Hyperledger (2017). Hyperledger fabric 1.0 is released ! Repéré à `https://www.hyperledger.org/blog/2017/07/11/hyperledger-fabric-1-0-is-released` ; Consulté en août 2022.

[Hyperledger, 2019a] Hyperledger (2019a). Build blockchain applications and business networks your way. Repéré à `https://hyperledger.github.io/composer/latest/` ; Consulté en août 2022.

[Hyperledger, 2019b] Hyperledger (2019b). How walmart brought unprecedented transparency to the food supply chain with hyperledger fabric. Repéré à `https://www.hyperledger.org/learn/publications/walmart-case-study` ; Consulté en août 2022.

[Hyperledger, 2020a] Hyperledger (2020a). Channels. Repéré à `https://hyperledger-fabric.readthedocs.io/en/release-2.2/channels.html` ; Consulté en août 2022.

[Hyperledger, 2020b] Hyperledger (2020b). Dlt labs™ & walmart canada transform freight invoice management with hyperledger fabric. Repéré à `https://www.hyperledger.org/learn/publications/dltlabs-case-study` ; Consulté en août 2022.

[Hyperledger, 2020c] Hyperledger (2020c). Membership service providers (msp). Repéré à `https://hyperledger-fabric.readthedocs.io/en/release-2.2/msp.html` ; Consulté en septembre 2022.

[Hyperledger, 2020d] Hyperledger (2020d). The ordering service. Repéré à `https://hyperledger-fabric.readthedocs.io/en/release-2.2/orderer/ordering\_service.html` ; Consulté en août 2022.

[Hyperledger, 2020e] Hyperledger (2020e). Planning for an ordering service. Repéré à `https://hyperledger-fabric.readthedocs.io/en/release-2.2/deployorderer/ordererplan.html` ; Consulté en août 2022.

[Hyperledger, 2020f] Hyperledger (2020f). Process and data design. Repéré à `https://hyperledger-fabric.readthedocs.io/fr/latest/developapps/architecture.html` ; Consulté en août 2022.

[Hyperledger, 2020g] Hyperledger (2020g). Smart contracts and chaincode. Repéré à `https://hyperledger-fabric.readthedocs.io/en/release-2.2/smartcontract/smartcontract.html` ; Consulté en août 2022.

[Hyperledger, 2020h] Hyperledger (2020h). v2.0.0 release notes - january 29, 2020. Repéré à `https://github.com/hyperledger/fabric/releases/tag/v2.0.0` ; Consulté en août 2022.

[IBM, 2019] IBM (2019). Join the trusted community improving the world's food supply with blockchain technology. Repéré à `https://www.ibm.com/blockchain/solutions/food-trust/food-industry-technology` ; Consulté en août 2022.

[IPFS, 2014] IPFS (2014). Ipfs powers the distributed web. Repéré à `https://ipfs.io/` ; Consulté en août 2022.

[Kamath, 2018] Kamath, R. (2018). Food traceability on blockchain : Walmart's pork and mango pilots with ibm. *The Journal of the British Blockchain Association*, 1:1–12. doi : 10.31585/jbba-1-1-(10)2018.

[Khan *et al.*, 2021] Khan, D., Jung, L. T. et Hashmani, M. A. (2021). Systematic literature review of challenges in blockchain scalability. *Applied Sciences*, 11(20). doi : 10.3390/app11209372.

[Mary Lacity, 2021] Mary Lacity, R. V. H. (2021). Requiem for reconciliations : Dl freight, a blockchain-enabled solution by walmart canada and dlt labs. *Blockchain Center of Excellence*. Repéré à `https://blockchain.uark.edu/new-bcoe-white-paper-on-walmart-canada-dlt-labs/` ; Consulté en août 2022.

[Matt Smith, 2018] Matt Smith, W. C. (2018). In wake of romaine e. coli scare, walmart deploys blockchain to track leafy greens. Repéré à `https://corporate.walmart.com/newsroom/2018/09/24/in-wake-of-romaine-e-coli-scare-walmart-deploys-blockchain-to-track-leafy-greens` ; Consulté en août 2022.

[Nakamoto, 2008] Nakamoto, S. (2008). Bitcoin : A peer-to-peer electronic cash system. *www.bitcoin.org*. Repéré à `https://bitcoin.org/bitcoin.pdf` ; Consulté en août 2022.

[OpenZeppelin, 2021] OpenZeppelin (2021). Access control. Repéré à `https://docs.openzeppelin.com/contracts/4.x/api/access` ; Consulté en août 2022.

[Redfield *et al.*, 2018] Redfield, C., Mounce, R., Mundo, M. et Yiannas, F. (2018). Food traceability initiative fresh leafy greens. Repéré à `https://corporate.walmart.com/media-library/document/blockchain-supplier-letter-september-2018/\_proxyDocument?id=00000166-088d-dc77-a7ff-4dff689f0001` ; Consulté en août 2022.

[Salah *et al.*, 2019] Salah, K., Nizamuddin, N., Jayaraman, R. et Omar, M. (2019). Blockchain-based soybean traceability in agricultural supply chain. *IEEE Access*, 7:73295–73305. doi : 10.1109/ACCESS.2019.2918000.

[Szabo, 1997] Szabo, N. (1997). Formalizing and securing relationships on public networks. *First Monday*, 2(9). doi : 10.5210/fm.v2i9.548.

[Vitasek *et al.*, 2022] Vitasek, K., Bayliss, J., Owen, L. et Srivastava, N. (2022). How walmart canada uses blockchain to solve supply-chain challenges. Repéré à `https://hbr.org/2022/01/how-walmart-canada-uses-blockchain-to-solve-supply-chain-challenges` ; Consulté en août 2022.

[Wiki, 2013] Wiki, B. (2013). Controlled supply. Repéré à `https://en.bitcoin.it/wiki/Controlled\_supply` ; Consulté en août 2022.

Annexe A

Fonctions de AppleContract

queryAllApples : Retourne la liste de tous les actifs `Apple`. Accessible à tous les rôles et organismes.

getAppleHistory : Retourne la liste de tous les états précédents de l'actif `Apple`. Accessible à tous les rôles et organismes.

getCurrentApple : Retourne l'actif `Apple`. Accessible à tous les rôles et organismes.

pick : Représente la cueillette d'un lot de pommes. Ajoute un nouvel actif `Apple` à l'état `PICKED`. Accessible seulement aux utilisateurs avec le rôle `picker` de l'organisme *Brute*.

bundle : Représente l'emballage d'un lot de pommes. L'actif `Apple` passe de l'état `PICKED` à l'état `BUNDLED`. Accessible seulement aux utilisateurs avec le rôle `picker` de l'organisme *Brute*.

outbound_B : Représente l'expédition d'un lot de pommes de l'organisme *Brute*

vers l'organisme *Processed*. L'actif `Apple` passe de l'état `BUNDLED` à l'état `OUTBOUND_B`. Accessible seulement aux utilisateurs avec le rôle `reception` de l'organisme *Brute*.

received_P : Représente la réception d'un lot de pommes par l'organisme *Processed*. L'actif `Apple` passe de l'état `OUTBOUND_B` à l'état `RECEIVED_P`. Accessible seulement aux utilisateurs avec le rôle `reception` de l'organisme *Processed*.

clean : Représente le nettoyage d'un lot de pommes. L'actif `Apple` passe de l'état `RECEIVED_P` à l'état `CLEANED`. Accessible seulement aux utilisateurs avec le rôle `processing` de l'organisme *Processed*.

outbound_P : Représente l'expédition d'un lot de pommes de l'organisme *Processed* vers l'organisme *Shipping*. L'actif `Apple` passe de l'état `CLEANED` à l'état `OUTBOUND_P`. Accessible seulement aux utilisateurs avec le rôle `reception` de l'organisme *Processed*.

received_S : Représente la réception d'un lot de pommes par l'organisme *Shipping*. L'actif `Apple` passe de l'état `OUTBOUND_P` à l'état `RECEIVED_S`. Accessible seulement aux utilisateurs avec le rôle `reception` de l'organisme *Shipping*.

tag : Représente l'étiquettage d'un lot de pommes pour l'expédition finale. L'actif `Apple` passe de l'état `RECEIVED_S` à l'état `TAGGED`. Accessible seulement aux utilisateurs avec le rôle `tagging` de l'organisme *Shipping*.

shipped : Représente l'expédition finale d'un lot de pommes par l'organisme *Shipping* vers un acheteur. L'actif `Apple` passe de l'état `TAGGED` à l'état `SHIPPED`. Accessible seulement aux utilisateurs avec le rôle `reception` de l'organisme *Shipping*.