INSA ROUEN
À taille humaine,
à l'échelle du monde

FRANC Louis Henri
2016

Pré-rapport C++
La BlockChain

Table des matières

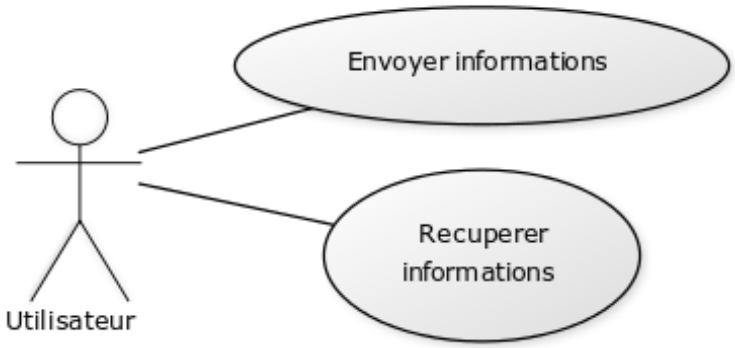
1 Introduction

Pour mon projet C++, j'ai décidé d'implémenter une Blockchain, une chaine de Bloc, en C++. La Blockchain est un protocole sous jacent au protocole Bitcoin inventé en 2008 par un illustre inconnu. La crypto monnaie Bitcoin est un système complet qui permet à des utilisateurs d'échanger des crypto coins entre eux. Les échanges se font à distance entre ordinateurs, l'argent n'est pas réel. Le système est complétement décentralisé, c'est à dire qu'il n'existe aucune entité (une banque par exemple) qui se charge garante de vérifier les transactions. Il est à la charge de chaque utilisateur, (appelé mineur), de vérifier l'ensemble de ces transactions afin que les utilisateurs ne puissent pas falsifier leur argent.

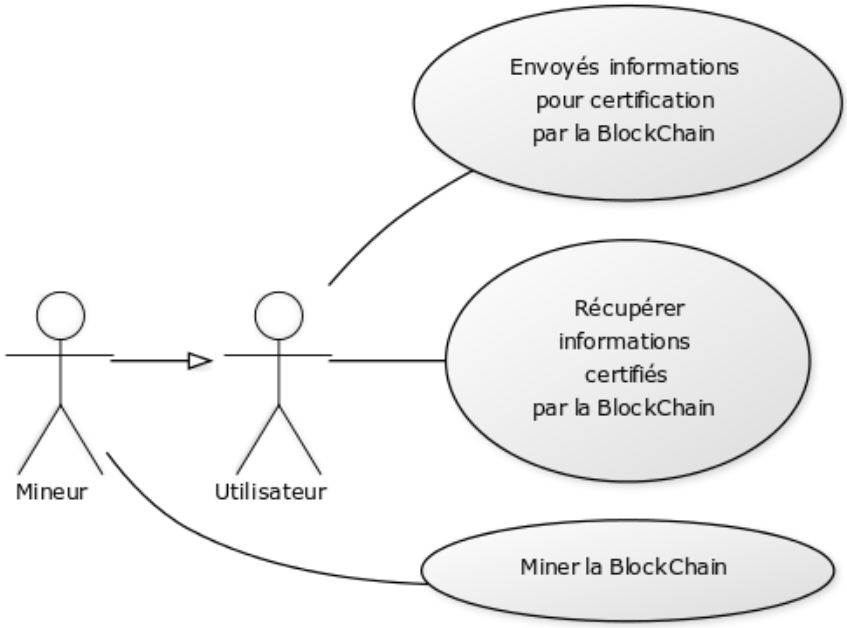
Le choix d'un projet comme la Blockchain résulte d'une question encore en suspent en 2016 : Comment assurer la véracité d'une donnée en ligne dans un système entièrement décentralisé. La Blockchain propose une solution viable pour ce problème, et ses applications sont multiples. Cependant un tel système est extremement compliqué à implémenter, et de nombreux projets sont en cours pour faire émerger de nouvelles solutions dans des domaines comme la finance, le cloud.

2 Analyse des besoins et Diagramme d'utilisation

J'ai décidé de travailler sur l'implémentation d'une Blockchain simplifié. Il ne s'agira pas ici de proposer un équivalent d'une Blockchain pour une crypto monnaie. L'équation ici sera grandement simplifié, puisque les données que les utilisateurs vont s'échanger et qui seront valides grâce au système de Blockchain seront uniquement des données simples. Ainsi un utilisateur voulant certifier certaines informations personnelles publiques pourra proposer de les ajouter a un ensemble de données partagées par tous les utilisateurs. Ces données auront été vérifiés par la BlockChain. Cette structure de donnée se contentera de les certifier ou de les invalider si elles sont incorrectes, ou si elles n'appartiennent pas à la personne qui se prétend être l'émetteur. D'un point de vue purement fonctionnel, les besoins sont assez simples. Il permete à l'utilisateur d'envoyer et de récupérer des informations. Les utilisateurs peuvent être sur que les informations sont correctes car la BlockChain les "aura vérifier".



Utilisation formel de point de vue utilisateur



3 Analyse fonctionnelle

Dans cette partie, je vais essayer à toutes les questions qui concerne le fonctionnement de mon programme, c'est à dire ce qu'il doit faire et non pas comment il va le faire. Les utilisateurs auront une interface qui leur permettra d'échanger avec la BlockChain et la Base de donnée d'informations. Les utilisateurs pourront envoyer des informations qui seront envoyé à la BlockChain pour vérification, puis celle ci seront ajoutés à une base

de donnée simple. Chaque envoi d'informations aura un identifiant unique qui sera la clé pour la retrouver dans la base de donnée. Tout utilisateur qui enverra une informations le concernant enverra aussi un moyen de l'identifier qui lui est unique et qui permettra lors de la consultation de la base de donnée de savoir qui a écrit quoi.
D'une autre part, un utilisateur souhaitant obtenir une information pourra consulter la base de donnée. Pour récupérer une information, il devra récupérer tout d'abord l'identifiant qui lui correspond dans la base de donnée. Il lui sera renvoyé l'information ainsi qu'un moyen de vérifier que cette information est correcte.

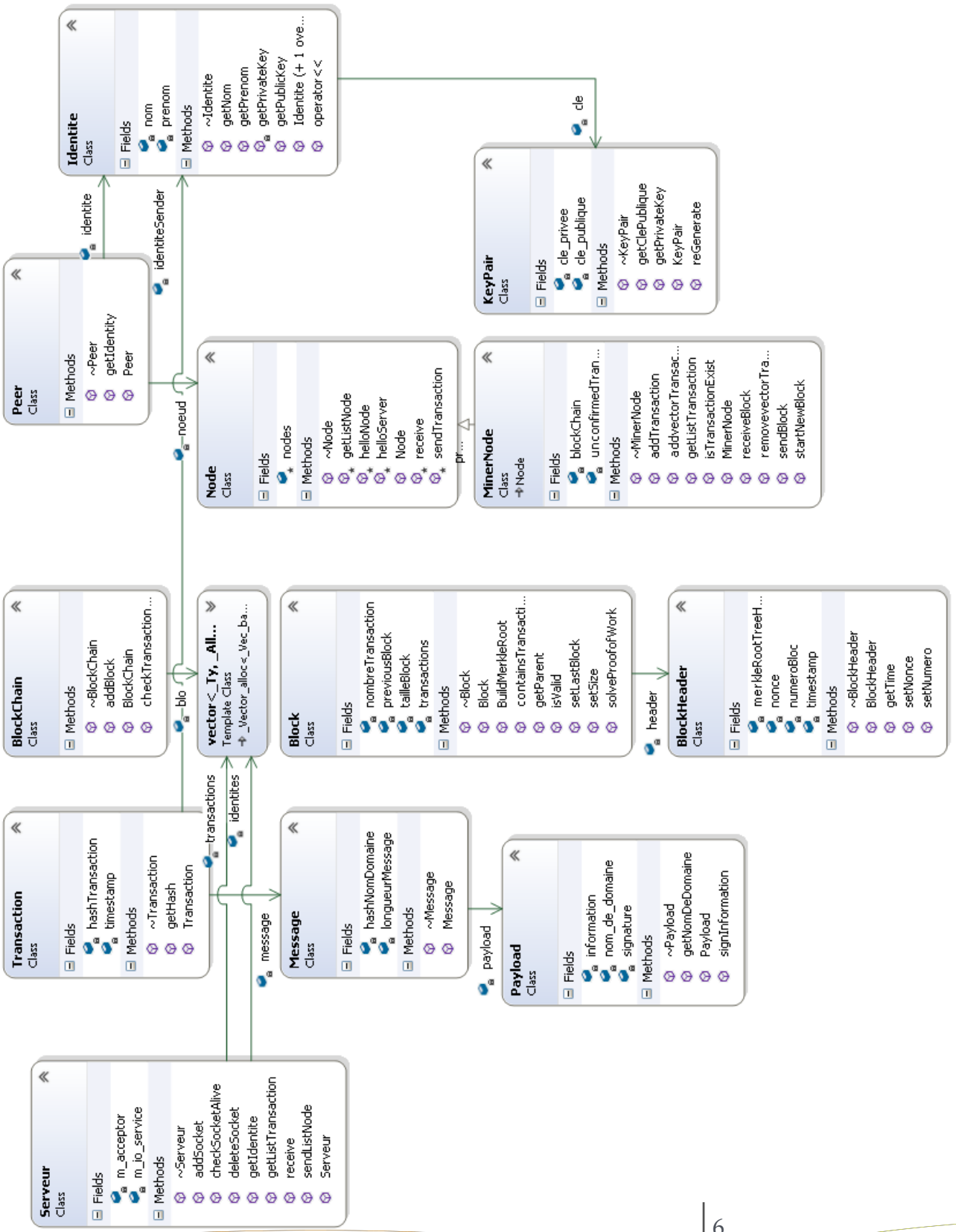
Enfin, afin que les utilisateurs puissent envoyer et récupérer des informations vérifiés, une blockchain doit être maintenu, pour cela il existera une autre interface permettant à certains utilisateurs, les mineurs de proposer leur "aide" afin que la BlockChain soit maintenu à jour. Ceux ci n'auront qu'a lancer le programme, et par la suite il mineront tout seul

4 Choix de conception

Dans un soucis de simplicité, je n'ai pas non plus implémenter une véritable BlockChain comme celle issu du protocole Bitcoin. Tout d'abord il n'y aura pas de transactions entre les peers de réseau. Il ne s'agira pas ici d'échanger de l'argent, mais plutôt de pouvoir enregistrer dans une base de donnée, une information comme un nom de domaine et sa translation addresse IP. Ces enregistrements, que l'on appelera transaction seront vérifié une par une dans la BlockChain. Dans un premier temps, l'utilisation d'un tel système permettra d'éviter que deux individus réclament la possession de la même clé dans la base de donnée en même temps. Le réseau de mineur décidera lequel sera choisi,mais il n'y en aura pas deux. L'autre avantage d'une BlockChain est que les toutes les transactions pourront par la suite être enregistré dans une table de hash partagée (DHT). Ainsi la base de donnée persistera qu'importe les problèmes où les censures. Evidemment si j'avais décidé de rajouter un système de transaction, où il aurait été possible d'acheter l'emplacement d'une clé dans la base de donnée, et de la revendre, rajoutant tout un système de vente et d'achat, cela se serait plus rapproché du protocole Bitcoin, mais en un mois, je ne me vois pas pouvoir faire tout ca seul. La blockChain sera donc plus basique.
D'autres part, dans une blockchain, un DNS est utilisé pour forwarder les nouveaux noeuds vers d'autres. Pour mon cas, ce sera un serveur central. Le système ne sera pa complétement décentralisé.

5 Analyse technique

Je vais présenter le diagramme de classe, avant le diagramme de séquence, même si j'ai réalisé le second avant le premier. Voici donc le diagramme de classe. Il y a un seul héritage et de nombreuses aggrégations et composition. J'indiquerai lorsqu'il s'agit de composition, c'est à dire qu'une entité n'a de sens que dans une autre, ou l'aggrégation, lorsque une entité fait partie d'une autre mais existe en dehors aussi.Le diagramme de classe :


Identite
Class
Fields
nom
prenom
Methods
~Identite
getNom
getPrenom
getPrivateKey
getPublicKey
Identite (+ 1 ove...
operator<<
cle
KeyPair
Class
Fields
cle_privee
cle_publique
Methods
~KeyPair
getClePublique
getPrivateKey
KeyPair
reGenerate
identite
identiteSender
Peer
Class
Methods
~Peer
getIdentity
Peer
noeud
Node
Class
Fields
nodes
Methods
~Node
getListNode
helloNode
helloServer
Node
receive
sendTransaction
MinerNode
Class
Node
Fields
blockChain
unconfirmedTran...
Methods
~MinerNode
addTransaction
addvectorTransac...
getListTransaction
isTransactionExist
MinerNode
receiveBlock
removevectorTra...
sendBlock
startNewBlock
BlockChain
Class
Methods
~BlockChain
addBlock
BlockChain
checkTransaction...
blo
vector<_Ty, _All...
Template Class
_Vector_alloc<_Vec_ba...
Block
Class
Fields
nombreTransaction
previousBlock
tailleBlock
transactions
Methods
~Block
Block
BuildMerkleRoot
containsTransacti...
getParent
isValid
setLastBlock
setSize
solveProofofWork
header
BlockHeader
Class
Fields
merkleRootTreeH...
nonce
numeroBloc
timestamp
Methods
~BlockHeader
BlockHeader
getTime
setNonce
setNumero
Transaction
Class
Fields
hashTransaction
timestamp
Methods
~Transaction
getHash
Transaction
transactions
identites
message
Message
Class
Fields
hashNomDomaine
longueurMessage
Methods
~Message
Message
payload
Payload
Class
Fields
information
nom_de_domaine
signature
Methods
~Payload
getNomDeDomaine
Payload
signInformation
Serveur
Class
Fields
m_acceptor
m_io_service
Methods
~Serveur
addSocket
checkSocketAlive
deleteSocket
getIdentite
getListTransaction
receive
sendListNode
Serveur

Voici donc une explication en détail des classes

5.1 La classe Peer

Cette classe correspond a un utilisateur. Lors de son utilisation, l'utilisateur se connecte au réseau au travers d'un noeud, la classe Node **composition**. Il possède aussi une identité **aggrégation**.

5.2 La classe Identité

L'identité d'un peer est composé d'un nom, d'un prénom, ainsi qu'une paire de clé publique-clé privée **aggrégation**. Cette paire de clé est essentielle au fonctionnement de la chaine de bloc.

5.3 La classe KeyPair

Elle corresond à une paire de clé asymétrique (clé publique, clé privée). Elle sera relié à une bibliothèque de crypto (nombre premiers,bigint, rsa...) que je vais sans doute implémenter par moi même (et tant pis pour les standarts :)). Cette paire de clé sera un moyen de s'identifier dans le réseau, de signer des documents.

5.4 La classe Node

La classe Node est la classe centrale d'un peer. Elle s'occupe de tous les échanges avec les autres noeuds. Elle permet par exemple d'envoyer de nouvelles transactions aux autres noeuds.

5.5 La clase MinerNode

Cette classe hérite de Node **héritage**. Elle implémente un ensemble de fonctions en plus permettant au peer de miner les transactions de la BlockChain. Ses attibuts sont une blockchain **composition** ainsi qu'un vecteur de transaction non confirmés **composition**.

5.6 La classe BlockChain

La classe BlockChain correspond à la structure de donnée BlockChain. Lexicalement une BlockChain n'est qu'une chaine de bloc. J'ai décidé de l'implémenter de cette manière. Cela sera un vecteur de bloc **composition**. Il y aura des ajouts, des suppression sur ce vecteur. Comme la BlockChain, les derniers blocs ajoutés à ce vecteur ne seront pas forcément ceux qui seront validés. Cependant plus le bloc aura été ajouté il y a longtemps, plus il sera certain qu'il fait partie de l'historique finale.

5.7 La classe Block

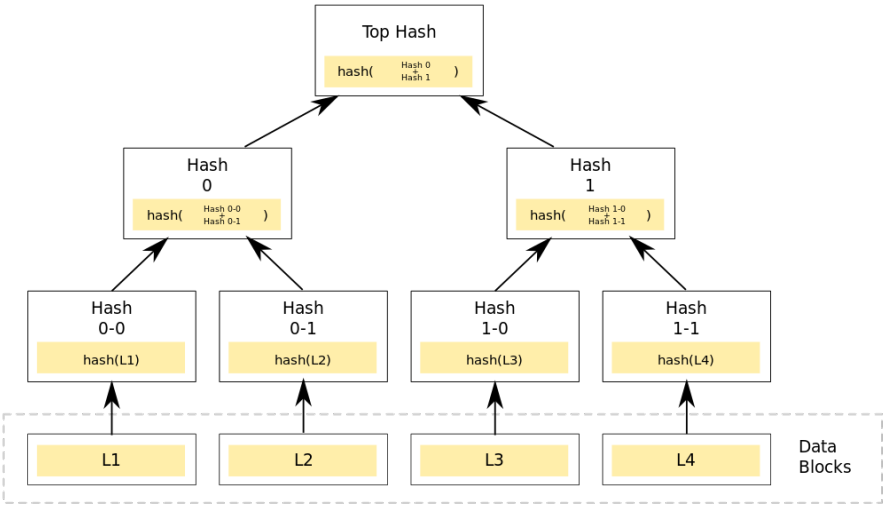
Je me suis inspiré de la constitution d'un bloc de la BlockChain du protocole Bitcoin. Les différents attributs seront :

- un entier représentant le nombre de transactions
- un pointeur sur le bloc précédent
- la taille du bloc en nombre d'octets
- un vecteur de transactions correspondant aux transactions vérifiés dans ce bloc. Par ce que une transaction est très gros, il ne s'agirat ici d'enregistrer qu'un hash de la transaction. (sha256)
- un BlockHeader **composition**

5.8 La classe BlockHeader

Cette classe correspond à un ensemble de valeurs identifiant le bloc de manière unique.

- Tout d'abord chaque transaction sera hasher ensemble pour former ce qu'on appelle un merkle Root. Voici un exemple de arbre de Merkle.



- un nonce qui fera partie de la proof of work pour calculer le block. Pour faire simple. Le hash du merkle root tree sera concaténé avec un nonce. Tout cela sera hashé. Le but pour les mineurs est de trouver un hash qui aura 5 zeros en premiers. Un hash étant complétement aléatoire et irréversible, cela constitue une bonne preuve de travail.
- le numéro du bloc
- une heure correspondant au temps de la BlockChain

5.9 La classe Transaction

La classe Transaction représente une transaction faite par un peer qui souhaiterait ajouter une entrée dans la base de donnée. Les différents attributs de cette classe sont donc un identifiant unique (un hash), une date, ainsi qu'un message contenant l'information de la demande du peer **composition**.

5.10 La classe Message

Un message fait partie de la classe transaction. Sans transaction,il n'y a pas de message. On parle ici de composition.
Un message contient un taille (la taille du message), un identifiant (la clé à ajouter dans la base de donnée), et un payload contenant toute l'information nécéssaire **aggrégation**

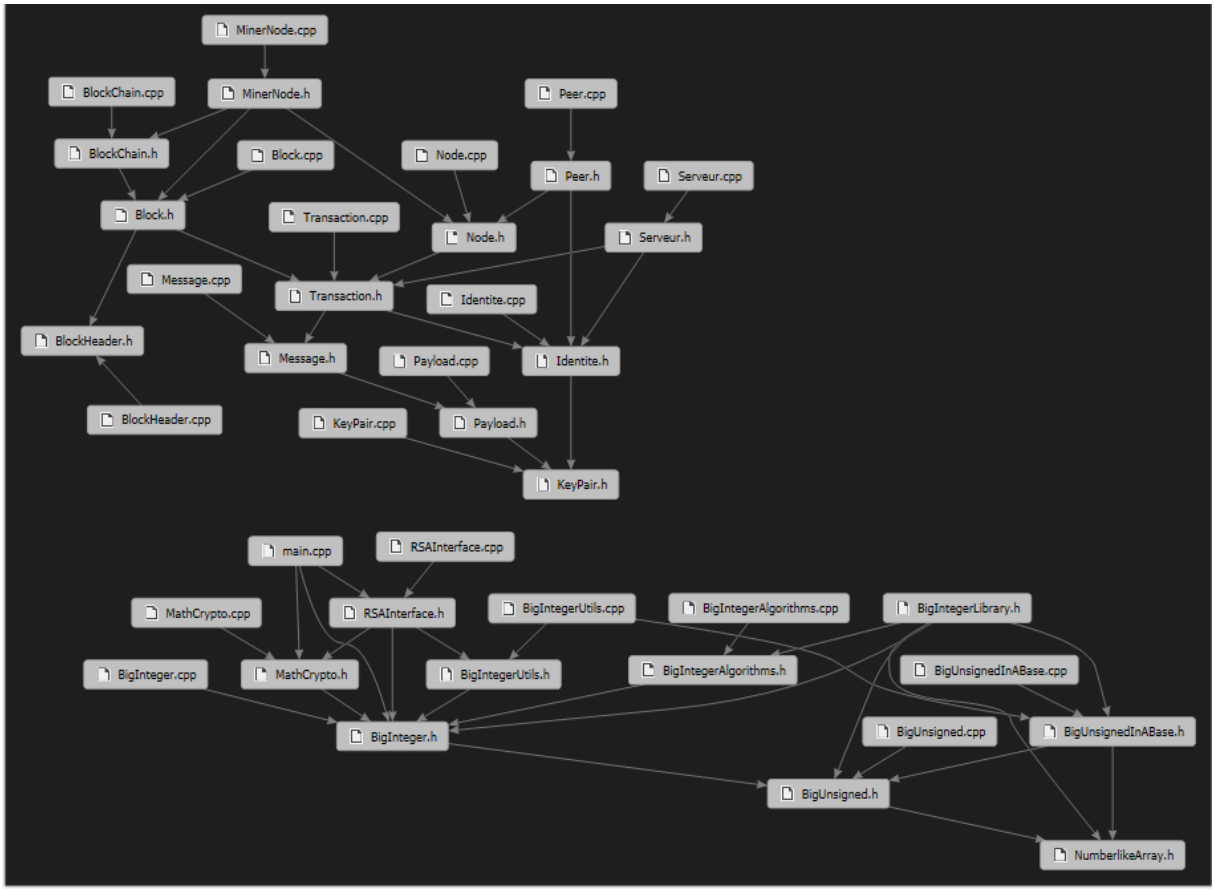
5.11 La classe Payload

Le payload correspond à l'ensemble des données inclus dans la transaction. Ce sera une signature (message signé avec la clé privée), un ensemble d'informations que le peer veut transmettre.

5.12 La classe Serveur

La classe Serveur est implementé dans un soucis de simplification. Le réseau ne sera pas vraiment un réseau peer-to-peer. Le serveur se chargera au début de re dirigé les nouveaux noeuds vers des noeuds déja existant. Il servira de log aussi en sauvegardant l'ensemble des informations qui circulent sur le réseau. En effet lorsque les noeuds discuteront entre eux, ils enverront une copie de ces discussions au serveur.

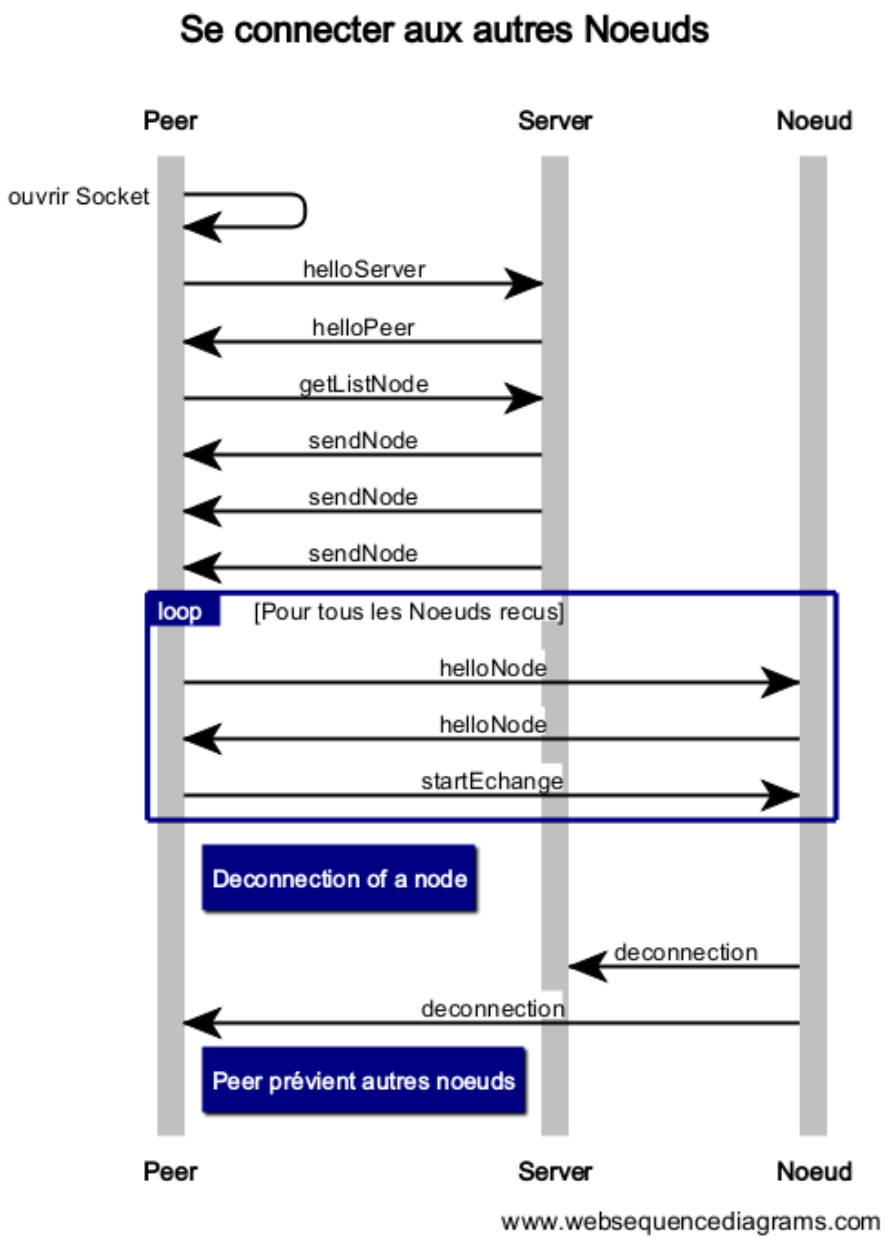
Voici un diagramme de classe représentant l'ensemble des classes et des headers que je vais utiliser pour mon projet. Aux classes précédentes s'ajoute la librairie de crypto que je n'ai pas voulu détailler car cela ne relève pas vraiment du projet.



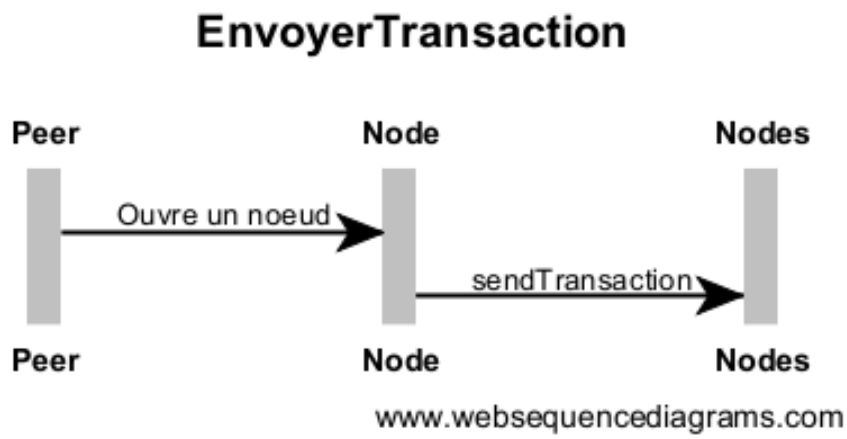
6 Diagramme de séquence

Voici quelques diagrammes de séquences pour bien comprendre les échanges entre les noeuds.

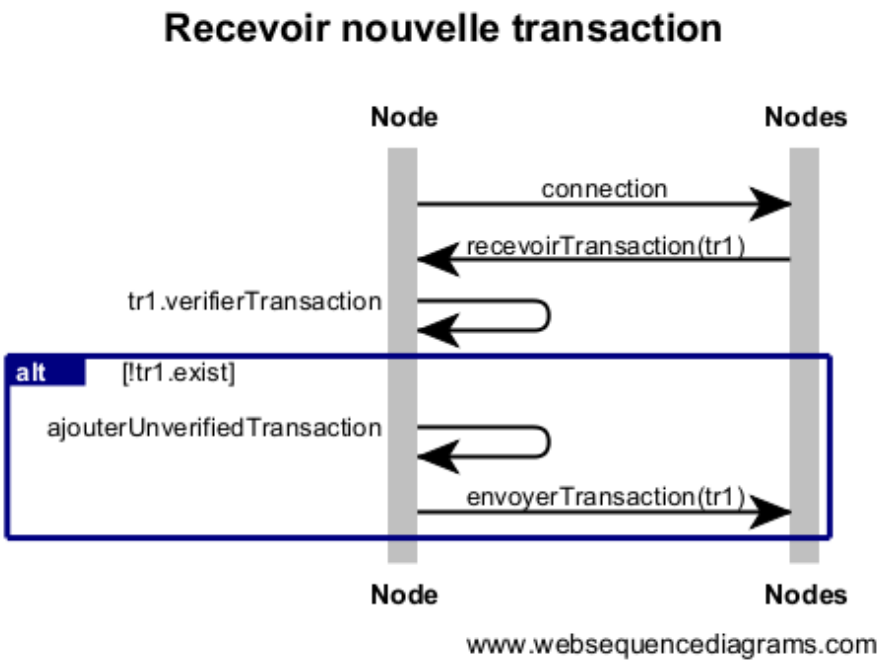
6.1 Se connecter aux autres noeuds pour la première fois



6.2 Envoyer une transaction aux autre noeuds

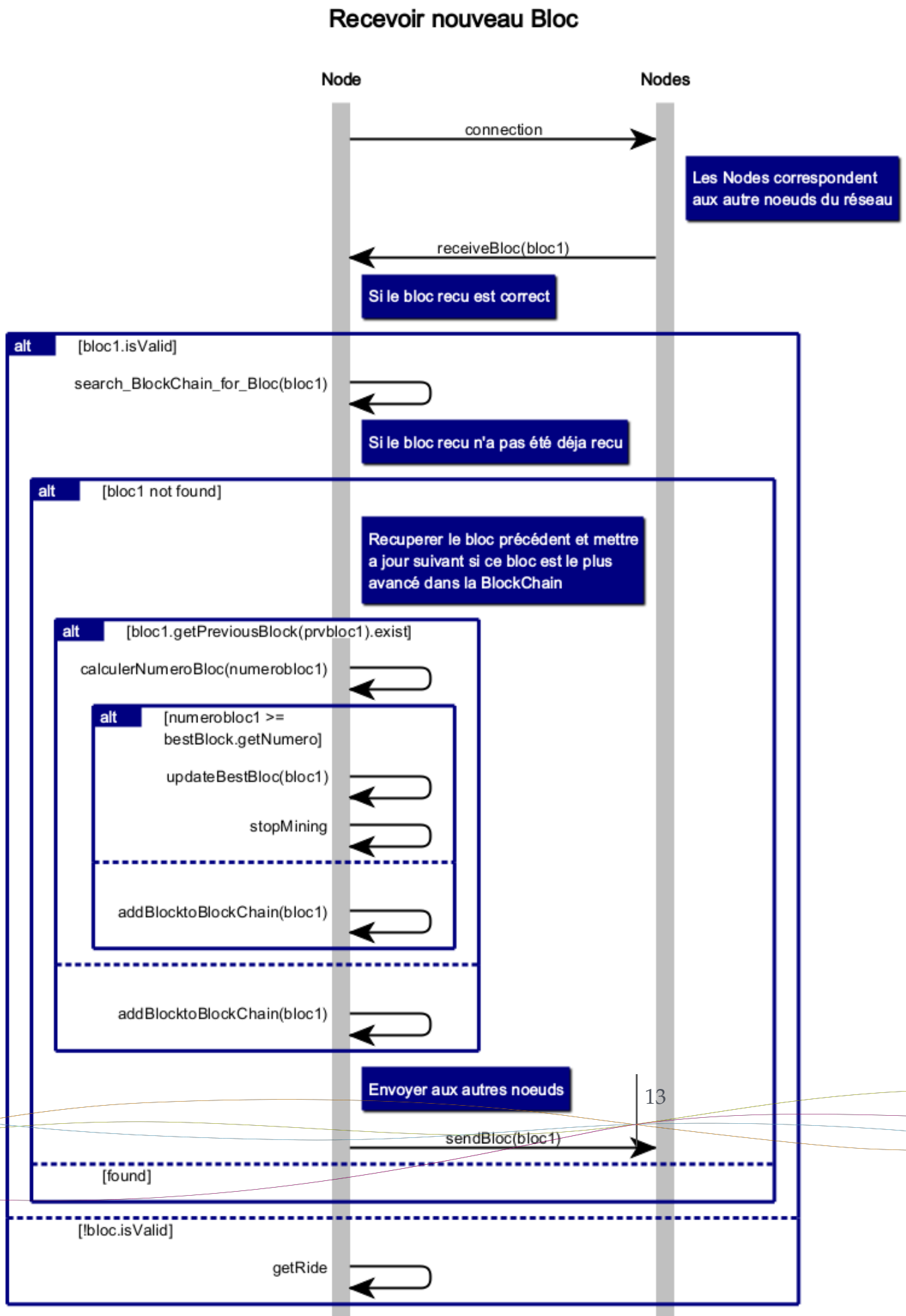


6.3 Recevoir une transaction d'un autre noeuds



6.4 Recevoir un nouveau bloc

Ce diagramme de séquence est un peu compliqué, il tente d'éxplique rapidement comment un *fork* a lieu dans l BlockChain.


Recevoir nouveau Bloc
Node
Nodes
connection
Les Nodes correspondent aux autre noeuds du réseau
receiveBloc(bloc1)
Si le bloc recu est correct
alt
[bloc1.isValid]
search_BlockChain_for_Bloc(bloc1)
Si le bloc recu n'a pas été déja recu
alt
[bloc1 not found]
Recuperer le bloc précédent et mettre a jour suivant si ce bloc est le plus avancé dans la BlockChain
alt
[bloc1.getPreviousBlock(prvbloc1).exist]
calculerNumeroBloc(numerobloc1)
alt
[numerobloc1 >= bestBlock.getNumero]
updateBestBloc(bloc1)
stopMining
addBlocktoBlockChain(bloc1)
addBlocktoBlockChain(bloc1)
Envoyer aux autres noeuds
sendBloc(bloc1)
[found]
[!bloc.isValid]
getRide

7 Prochaines étapes

Le développement est maintenant fini. Voici les différentes étapes de l'implémentation.

— Mise en place d'une librairie de cryptographie. Si cela est trop compliqué, je ferai appel à une librairie déja crée. Implémentation des fonctions de cryptage, déchiffrage , signature, hashage
— Conception du réseau, échange entres les différents peers. Prise en main de la bibliothèque boost : :asio.
— Mise en place des échange des transactions entre les utilisateurs
— Mise en place de la blockchain comme support pour la validation des transactions dans le réseau

8 Rendu 2 du projet

J'ai décidé de laisser la première partie du rapport pour pouvoir faire des parallèles avec ce que j'avais écris avant et ce vers quoi le projet s'est orienté.

8.1 Remarques du professeur

— *Il est préférable de spécifier les multiplicités dans les associations.* Oui, c'est vrai que j'ai été un peu vite
— *Je n'ai pas compris pourquoi vous dites qu'il y a une agrégation entre les deux classes "Peer" et "Identite" et aussi une agrégation entre les deux classes "Identite" et "KayPair". Chaque peer est caractérisé par une seule identité. Devrions-nous avoir des associations simples entre les deux classes "Peer" et "Identite" avec les multiplicités suivantes : à droite 1..1 et à gauche 1..1 et "Identite" et "KayPair" avec les multiplicités suivantes : à droite 1..1 et à gauche 1..1* : Je me suis mal exprimé, l'association n'est pas bilatéral. Un peer est caractérisé par une seule identité qui est crée lors de la connexion.
— *Un objet est formé de l'agrégation de plusieurs autres, avec une indépendance de la construction des instances des classes. L'agrégation est transitive. La composition est une agrégation « forte ». La composition est une composition ssi • Le composé ne peut exister sans ses composants • Le composant ne peut appartenir qu'à un seul composé• Le composé est responsable de la création et de la destruction des composants* : Je prend, merci pour l'explication.
— *Dans le diagramme de classe, il n'y a pas la liaison illustrant l'association entre les deux classes "MinerNode" et "vecteur de transaction"* La classe MinerNode a disparu de mon projet, mais j'avais de toute facon un problème avec Visual Studio et son visualisateur de classe. Lorsqu'il s'agissait d'attribut comme des sharedptr ou des vecteurs, il me mettais cela comme une nouvelle classe au lieu de me faire correctement l'association multiple avec la classe en question.
— *-Dans le diagramme de séquences, s'il y a une flèche de retour de procédure, il faut préciser le message affiché l'écran sur une flèche interrompue pleine (e.g. "OK"). une flèche droite pleine lorsque l'émetteur attend "synchrone" une flèche droite non-pleine lorsque l'émetteur*

déclenche et passe "asynchrone" une flèche interrompue pleine pour les retours de procédure "optionnel" -Dans le diagramme de séquences "Se connecter aux autres nœuds pour la première fois", il faut garder le même nom des classes, remplacer "Nœud" par "Node" -Dans le diagramme de séquences "Envoyer une transaction aux autres nœuds", à mon avis ça sera préférable de créer deux instances de la classe "Node" (e.g. "Node1", "Node2", ...) pour illustrer les interactions. Pareillement pour le diagramme de séquences "Recevoir une transaction d'un autre nœuds" Je prend tous ces conseils. Il est vrai que je ne savais pas trop, j'ai mis des flèches un peu aléatoirement... OUps

8.2 Avancement

Tout d'abord voici les étapes que j'ai réalisé et celle que je n'ai pas encore achevé (ou plutôt pas réussi).

— *Mise en place d'une librairie de cryptographie. Si cela est trop compliqué, je ferai appel à une librairie déja crée. Implémentation des fonctions de cryptage, déchiffrage , signature, hashage* :
J'ai comme prévu tout d'abord d'essayer de mettre en place ma propre librairie de cryptographie, en implémentant deux protocoles cryptographique RSA et SHA. Pour SHA256 cela fonctionnait mais était relativement lent. Pour RSA, j'arrivai à générer les clés mais me suit heurté à certaines complications comme le découpage en bloc de bytes d'un message a chiffré... Et en plus cela était extremement long. J'ai donc préféré abandonner, pour ne pas perdre trop de temps. Je me suis rabatu sur la librairie de cryptographie CryptoPP (et non OpenSSL, c'est un choix).

— *Conception du réseau, échange entres les différents peers. Prise en main de la bibliothèque boost : :asio.*
J'ai bien pris en main la Bibliothèque Asio, cela m'a pris pas mal de temps, c'était pas la partie la plus intéressante. J'ai eu des problèmes pour l'envoi de données sur le réseau. J'avais utilisé la bibliothèque *boost : :serialize*. Un problème est apparu tout de suite, comment faire pour envoyer une instance d'une classe sur le réseau qui contient un pointeur sur un autre objet dans ses attributs. Boost : :serialize s'assure **en principe** que l'on construise l'objet pointé par le pointeur dans le packet à envoyer pour le reconstituer de l'autre côté.... (ce qui n'a pas franchement marché pour moi). J'ai eu aussi un autre problème... que se passe t'il quand dans une instance d'une classe, j'ai un pointeur sur une autre instance de la même classe. C'est ce qui m'est arrivé avec la classe Block (Rappel : Un Block a toujours un Block sur lequel il pointe, ce qui permet de créer une BlockChain...). Ne sachant pas si la serialization allait me serializer tous les Blocks en n'en voulant serializer un, j'ai décidé d'abandonner l'idée de mettre comme attribut de Block, un autre Block, je garderai seulement un hash (une chaine de caractère) identifiant le block pointé.
Le réseau transmet correctement les packets, mais j'ai encore pas mal de problème de synchronisation, de threads asynchronisés appelés trop tôt ou trop tard, et les tests ne sont pas très pratique à faire. Pour l'instant mon réseau ne sera **pas peer-**

to-peer. Il y aura un serveur central qui reliera tous les clients... et cela car je n'ai toujours pas d'idée de comment implémenter un réseau Peer-to-Peer fonctionnel.... dommage. *Pour ce problème, je suis tout ouie à une aide externe*.

— *Mise en place des échange des transactions entre les utilisateurs* :
Je peux sans soucis envoyer une transaction d'un client à un serveur qui le transmettra aux autres clients.

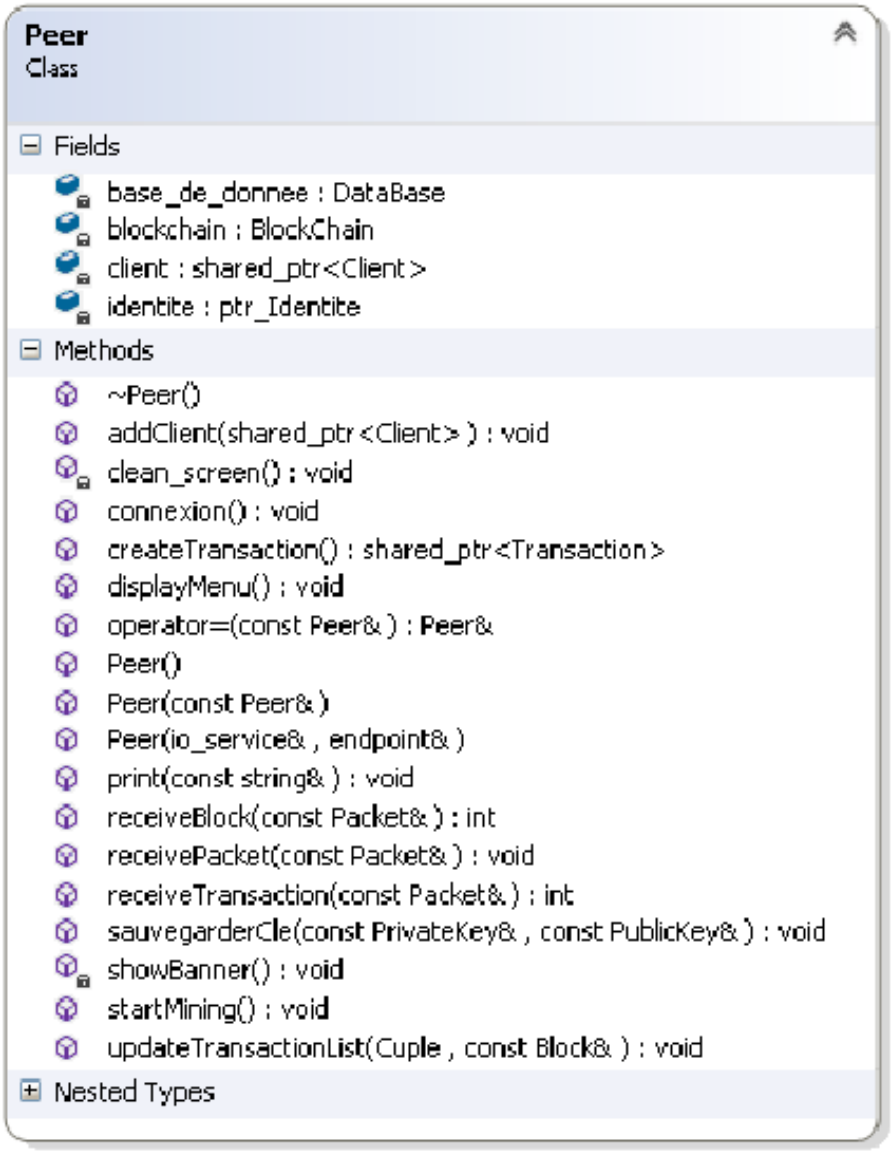
— *Mise en place de la blockchain comme support pour la validation des transactions dans le réseau* : A défault d'avoir réussi à faire un réseau peer to peer, j'ai réussi à implémenter une BlockChain totalement fonctionnel. Un client local connecte une BlockChain avec une Base de Donnée locale aussi, que j'ai implémenté. Cela permet par exemple pour un client de vérifier toutes les transactions recus du serveur.

8.3 Quelques Explications de Classes

Je ne vais pas expliquer le détail de l'implémentation de toutes les classes, mais uniquement de certaines, qui seront "centrale" au projet.

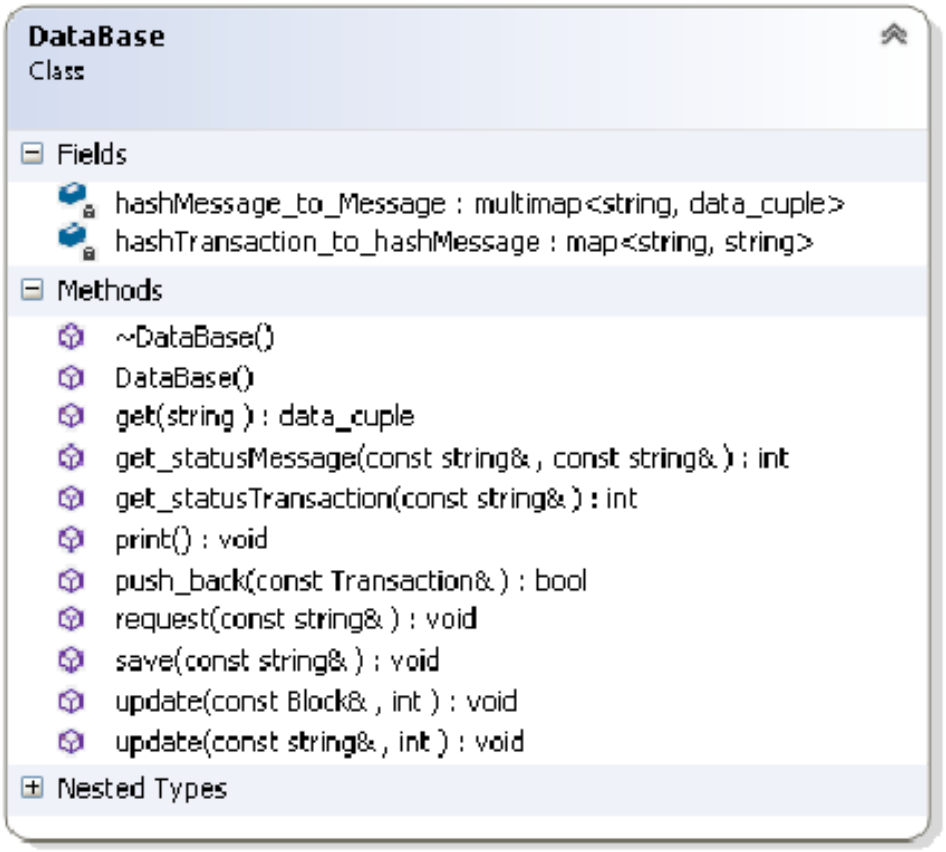
8.3.1 La nouvelle classe Peer

A noter la disparition de certaines classes que je n'ai pas encore implémenté : MinerNode et Node. Désormais ce sont les classes Peer et Client qui vont se charger de toutes les fonctionnalités qui étaient à la charge de Node. Cela sera plus simple, surtout avec les problèmes que sont la mise en réseau. Le Client se charge d'établir la connection avec le serveur. Il récupère et transmet des Packets. Pour info, j'ai trouvé une abstraction de classe pour une connection TCP qui me permet de ne pas avoir avoir à descendre trop bas niveau dans mes sockets. Ce sera la classe Connexion. Le Peer est une Inteface Homme Machine en ligne de commande. On s'y connecte avec des identifiants, et une paire de clé,que l'on crée, ou que l'on charge. On peut créer de nouveaux messages que l'on veut partager avec le réseau (qui seront par la suite certifiés par la BlockChain). On peut aussi démarrer le minage de la BlockChain. Par exemple lorsque le Peer souhaite miner, le Client se charge de transmettre au Serveur une demande de récupération de la BlockChain et de la Base de Donnée afin de pouvoir miner indépendamment.

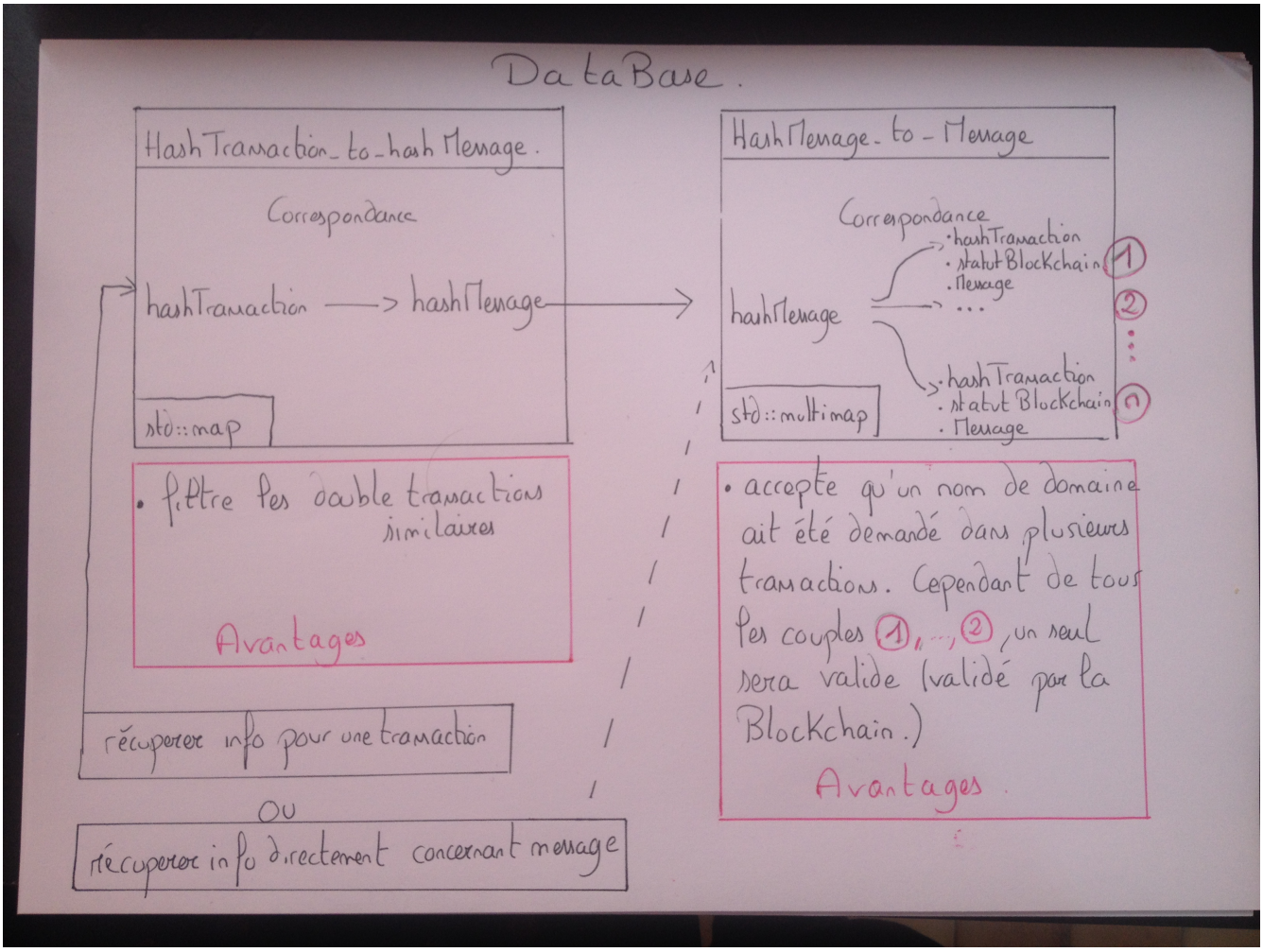


. Pour pouvoir miner, il faut posséder une BlockChain à jour ainsi qu'une base de donnée de transactions. Sans cela il est impossible de savoir si les nouveaux blocs recus sont corrects. Par exemple lorsque un peerB recois un nouveau Bloc miné par un autre Peer peerA, le Peer peerB vérifie tout d'abord si toutes les transactions dans le bloc recus sont dans la Base de Donnée. Si non, il refuse le Block, car considérant le temps de minage d'un bloc à a peu près dix minutes, il est peu propable qu'une transaction soit recu par le peerB après un Bloc qui l'inclus. Il y a donc une vérification de chaque Peer minant, des informations qu'il recoit, cela pour éviter que de faux blocs soit minés. Ceux ci seraient rejeté tout de suite.

8.3.2 La nouvelle classe Base de Donnée



La Base de Donnée est composé d'une *std : :map* et d'une *std : :multimap*. La map convertit chaque identifiant unique de Transaction en un identifiant de Message. Un identifiant de Message est le hash du nom de domaine. Par exemple facebook.com donnera *225237d2d8c627dc83a0428da3c8b86105beee81163a01eeaa03d547ca7cabc2*. Cet identifiant est la clé de la multimap. Il permet de parcourir tous les messages relatifs a ce nom de domaine. A noter que la BlockChain a une utilité ici ; une seule des entrée pour un nom de domaine a été validé, les autres ne sont pas valides. De plus la Base de Donnée vérifie si le message est correct avec la clé publique envoyé ainsi que la signature du message. Cela permet à une personne souhaitant consulter la base de donnée de vérifier si c'est bien la "bonne" personne qui a envoyé le message. Pour cela il suffit de regarder Validity of the Message et de comparer la clé publique avec celle que l'on a pour la personne (que l'on a eu d'une autre manière). Voici un résumé du fonctionnement de cette base de donnée en version papier, car cela sera plus clair.


DataBase.
HashTransaction_to_hashMessage.
Correspondance
hashTransaction ——> hashMessage
std::map
HashMessage_to_Message
Correspondance
hashTransaction
statutBlockchain
Message
...
hashMessage
hashTransaction
statutBlockchain
Message
std::multimap
• filtre les double transactions similaires
Avantages
• accepte qu'un nom de domaine ait été demandé dans plusieurs transactions. Cependant de tous les couples (1,...,2), un seul sera valide (validé par la Blockchain.)
Avantages
récuperer info pour une transaction
OU
récuperer info directement concernant message

Voici un exemple d'une entrée dans la Map. Ici le nom de domaine est assez court "FY". Ayant simulé de manière aléatoire la création de transactions ainsi que celui de la création de nouveaux blocs, le meilleur moyen pour avoir plusieurs entrées pour un même nom de domaine, était de prendre ce nom de domaine court (taille 2 parmi le lettres de l'alphabet.

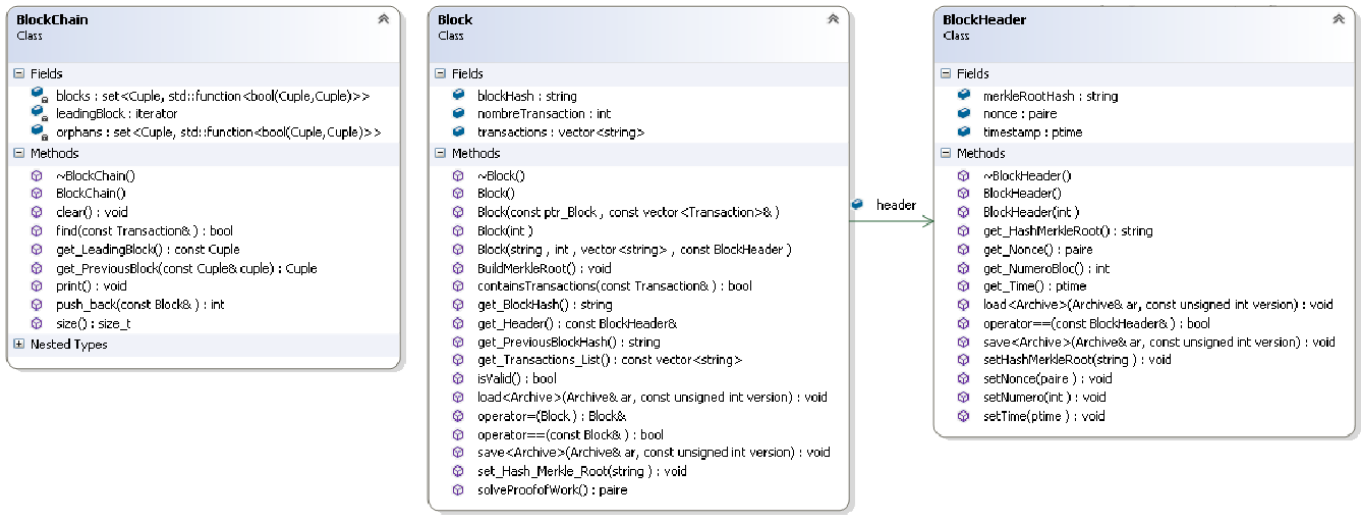
```
**********************************************************************
Request for the domain name : FY
Hash of the Domain name     :
939342FB94F9160A37F9EDC0510FCED5E779A41117267F426D828D61588AF763
Number of message for the DN: 3
Message  VALIDATED
  Information in the Message : bhC32713emXswQXptHwnCJIz4hc6iNtP6vQ
   Public Exponent Key of the Sender   : 17.
   Modulus Key of the Sender   :
  131385735875624543319007878315688072428338641374205275518880601448
  580007347699268688720673249769155485682710759177029635399170813479
  771141368049550210570666000419020595529449446084519543458811162555
  303433515586682222934242423000116763565919829411355240019039682370
   00108631666934126056131850241897568445925308l.
   Validity of the Message  : correct


Message  NOT VALIDATED
  Information in the Message : zmvA9uTKsoA6UT6qNhBxjfjGIiB7fuhJZnC
   Public Exponent Key of the Sender   : 17.
   Modulus Key of the Sender   :
  132537077803094093842316810554783521494756398723409923207161395848
  479581380196596716598582293565197170666360165909961892047324893650
  441248670026929904708381481340807939928476029499530673560098468341
  176827124077814318159236880209863140872161355753446004158380710372
   443379804346610132421936168011889795020321537.
   Validity of the Message  : correct


Message  NOT VALIDATED
  Information in the Message : x4Nh7zQUycEpakogL98kyu1MKrWWNlS67ki
   Public Exponent Key of the Sender   : 17.
   Modulus Key of the Sender   :
  131385735875624543319007878315688072428338641374205275518880601448
  580007347699268688720673249769155485682710759177029635399170813479
  771141368049550210570666000419020595529449446084519543458811162555
  303433515586682222934242423000116763565919829411355240019039682370
```

```
00108631666934126056131850241897568445925308 1.
Validity of the Message  : correct
```

8.3.3 La classe BlockChain



La manière dont j'ai décidé d'implémenter la BlockChain utilise deux *std : :set* et un itérateur sur un élément d'une std : :set.

Le choix d'un std : :set est que cela correspond à l'implémentation d'un red-black tree assortit d'une fonction de comparaison que l'on peut redéfinir. Je l'ai évidemment surchargé. Les blocs ayant les numéros les plus grand seront au début des set. Par exemple un Bloc avec comme numéro 20, aura donc 19 blocs "derrière lui". Si deux blocs ont le même numéro, alors il sera choisi celui qui aura le plus petit hash de transactions.
C'est un choix de conception. A tout moment le pointeur de la BlockChain pointe sur l'élément qui a le plus grand numéro (celui qui l'a eu en premier, cela veut dire que ce n'est pas forcément le premier élément de la set).

Dans le premier ensemble ne sont ajouté que des blocs correctes, qui ont leur parent présent dans la BlockChain. Tous les blocs corrects mais qui n'ont pas leur parents dans le premier set sont ajoutés dans le deuxième set. Le deuxieme set est le set des orphelins. Son implémentation permet la gestion de cas critique. Il se peut, même si cela reste étrange que l'on recoive un bloc numéroté 20 et que l'on recoive son parent quelques temps après, numéroté 19. Le bloc 20, au moment de sa reception n'est pas accepté par le premier ensemble. Il est ajouté au deuxieme ensemble. Une fonction de "nettoyage" de la BlockChain a cependant été implémenté. Une partie des blocs les plus vieux du set des orphelins tenteront d'être à nouveau injécté dans le set principal. Ici le block 20 sera accepté car le block 19 aura été recu. Si il ne peuvent être injecter, on les supprime pour gagner de la place. Lors de ce nettoyage, les blocs les plus anciens (c'est un paramètre), qui sont dans le set principal mais pas dans la chaine principale (c'est a dire la chaine partant du bloc de tête,

l'itérateur de notre classe) seront aussi supprimés. Il est possible de trouver dans les logs de test des exemples ou les fonctions de nettoyage, et d'insertion peuvent être examinés dans leur éxécution.

8.4 Génération de la Documentation Test unitaires

Une documentation a été généré pour le projet. Toutes les principales fonctions y sont commentés. Les test unitaires se trouvent dans le répertoire Test. J'ai aussi fait un test complet d'intégration de la BlockChain et de la Base de donnée. Le protocole choisi est détaillé dans le fichier "testBlockChainFull$_{I}ntegration$", $l'outputesttest.out$.

8.5 Et maintenant

Je vais essayer de mettre tout cela en réseau, et que cela fonctionne.. Pour un fonctionnement peer to peer, il faudra je pense un miracle :)