Rapport Travaux Dirigés
CORBA

Douézan-Grard Guillaume
Enseirb-Matmeca

4 novembre 2015

Dans le rendu du projet, le fichier `README` *donne des indications quant à l'exécution des différentes parties. La classe implémentant l'entité inter-bancaire est* `InterBank`*, deux banques sont présentes dans la classe* `Banks` *(par simplification, on aurait pu autrement les implémenter dans des classes différentes), et enfin des clients sont simulés dans la classe* `Clients`*. Ces derniers effectuent la création de comptes, quelques dépôts et retraits ainsi que des virement inter-bancaires. À la fin de l'exécution, ils se connectent à l'entité inter-bancaire pour récupérer le journal des transactions.*

L'objectif est d'implémenter la réalisation de transactions bancaires, par le biais d'une interface de gestion de compte bancaire pour les clients, et la présence d'une entité de vérification inter-bancaire qui assure l'authenticité des échanges entre banques.

La première étape est de créer les interfaces et définir les types mis en jeu. Pour ma part, j'ai considéré qu'un compte serait représenté par un *UUID* à l'extérieur de la banque, avec un préfixe permettant d'identifier cette dernière. Au sein de la banque, les clients on accès à une interface `BankCustomer` qui leur permet, avec un *design pattern factory*, de créer des comptes `Account`. Cette interface comprend les possibilités de retrait, de dépôt, ainsi que d'effectuer un virement vers un compte défini par son adresse. Les références qui devront être partagées avec les clients sont donc l'interface de création de compte, et l'interface de gestion de compte elle-même.

Concernant les communications entre banques, chacune expose à l'organisme inter-bancaire une interface permettant de créditer ou débiter un compte étant donné une adresse. De son côté, l'organisme inter-bancaire expose une interface permettant à une banque de s'enregistrer sur celui-ci (et à cette occasion lui donne accès aux *callbacks* exposés précédemments).

D'autre part, chaque banque a la possibilité d'émettre une transaction, que l'organisme inter-bancaire se charge de réaliser en contactant la banque destinataire, en enregistrant la transaction et en la confirmant à la banque émettrice. Dans l'implémentation que j'ai réalisé, je n'ai pas pris en compte un certain nombre d'aspects accessoires : je ne vérifie pas s'il reste suffisemment d'argent sur le compte source, je ne vérifie pas l'autenticité de la banque qui émet une transaction (un client pourrait se connecter directement à l'organisme inter-bancaire et émettre les transactions qu'il souhaite), etc. Enfin, l'organisme inter-bancaire permet à tout le monde de récupérer le journal des transactions qu'il stocke.

D'un point de vue technique, trois éléments étaient à réaliser. D'une part, les transactions inter-banques doivent être asynchrones. Pour cela, une première phase d'enregistrement auprès de l'entité inter-bancaire est nécessaire. Elle permet de donner la référence à l'interface de la banque pour confirmer ou initier les transactions. Ainsi, peu importe les traitements effectués, une banque a juste à soumettre une transaction sans attendre de résultats. C'est la responsabilité de l'entité inter-bancaire de notifier les banques ultérieurement du succès de la transaction.

Un peu plus difficile, on souhaite maintenant que l'organisme inter-bancaire soit activé seulement quand nécessaire et que les données qu'il stocke soient persistantes. Pour cela, on passe par un outil externe `servertool`, qui s'assure d'activer dynamiquement le serveur lors de l'utilisation d'une méthode sur un servant qu'il expose par un client. On doit donc commencer par l'enregistrer. Le support est bien entendu un *ORB* pour passer les messages correspondant aux différentes requêtes, et d'autre part l'utilisation d'un *POA* qui donne un modèle pour persister les données.

Enfin, le troisième aspect est de transformer seulement quand nécessaire une implémentation en *CORBA Object*, en particulier lorsqu'un compte est seulement stocké mais n'est plus utilisé, il est inutile de conserver la partie qui permet de le partager. Ceci passe par l'utilisation de références associées à un *id* unique. On défère alors la création même du servant à sa première utilisation, et il peut être par la suite désactivé ou activé si nécessaire à partir de son *id*.