Programmation par composants
2014-2015

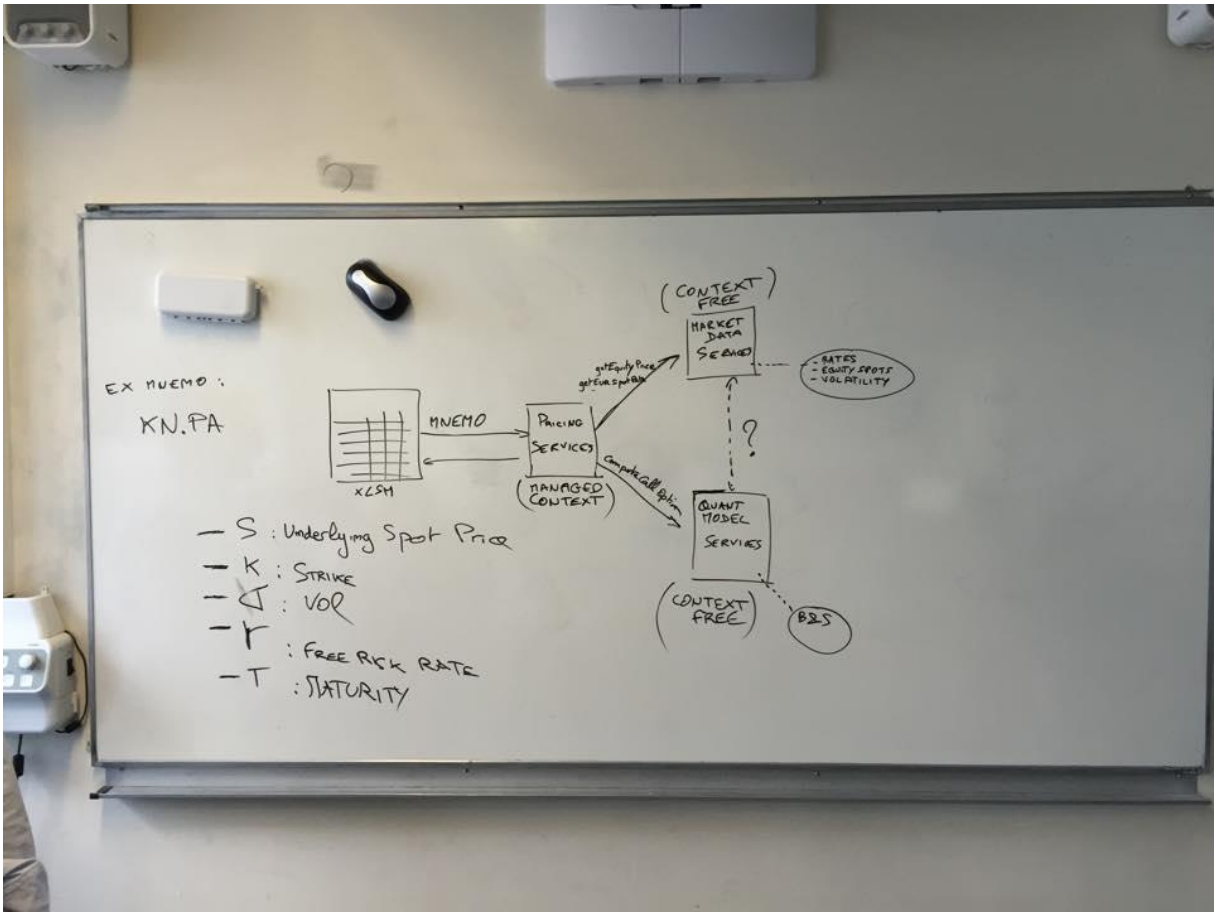
Yoann DUBERNET

Implémentation d'un pricer Black & Scholes

10 mai 2015

I. Description du projet

Le projet d'implémentation d'un pricer en programmation par composants consiste à développer trois composants avec Visual Studio plus ou moins dépendants les uns des autres.



- Le composant **PricingService** est un composant servant d'interface entre notre feuille Excel et nos composants de récupération de données et de calculs mathématiques. Il communique avec chacun des deux et reçoit les ordres du code VBA Excel
- Le composant **MarketDataService** est un composant servant à récupérer des données de marché. Dans l'idéal, il serait suffisamment indépendant pour qu'une fois lancé par un clic sur un bouton dans Excel par l'utilisateur, un while(1) lui permette d'aller récupérer en permanence des données sur le marché. En pratique, pour la partie récupération des données, j'ai utilisé un composant implémenté de longue date en Java par une connaissance (M. Philippe Rémy) et disponible en public sur https://github.com/philipperemy/Market-Data/qui crowle les données des actions sur le site http://www.boursorama.com. Ma partie a consisté à récupérer les données stockées dans la base MySQL remplie par le code Java.
- Le composant **QuantModelService** est un composant servant à calculer différentes valeurs financières classiques. Sont implémentées dans mon composant les lettres grecques et la fonction de Black&Scholes.

II. Choix techniques

1. PrincingService

Mon composant **PrincingService** ne contient aucune classe. Il liste au travers de son header l'ensemble des fonctions appelables dans les autres librairies. Il s'agit aussi des fonctions appelables par l'utilisateur dans la feuille Excel.

Chacune d'entre elles est définie en

```
extern "C" __declspec(dllexport) double __stdcall nomFonction(parametre[s]) ;
```

Extern « C » permet d'éviter la décoration des fonctions à la compilation afin que les fonctions soient appelables par des programmes/dlls extérieurs.

PrincingService étant le composant central, il a besoin de la connaissance de l'existence des autres librairies, contrairement à celles-ci. Ainsi, comme on le ferait lors de l'inclusion de librairies téléchargées sur le net, nous ajoutons au **PricingService** les .h de nos deux autres dlls et avons précisé les répertoires additionnels où aller chercher les librairies **MarketDataService** et **QuantModelService**.

2. MarketDataService

Le composant **MarketDataService** définit trois méthodes récupérant chacune trois paramètres que sont le spot du produit considéré, sa volatilité et le taux Euribor. Chacune va se connecter à la base de données MySQL (qu'il vous faudra préalablement avoir créé pour exécuter l'application), récupère des données et en retourne la valeur adéquate, qui est renvoyée au composant central **PricingService**, qui se charge lui-même de renvoyer les valeurs à la feuille Excel.
L'interaction avec Excel nécessite ainsi d'avoir deux librairies en plus que sont les librairies mysql pour C++ et le connecteur C++/MySQL. Celles-ci sont paramétrées dans la solution.

3. QuantModelService

Le composant **QuantModelService** définit un ensemble de méthodes qui, selon que l'actif soit considéré comme un put ou comme un call, appelle la méthode adaptée de calcul correspondant.
Ainsi, pour le calcul de BlackAndScholes, le code est le suivant :

```
void CQuantModelServices::calculateBlackAndScholes(double S, double K, double vol,
 double R, double T, bool isCall)
{
   if(isCall) {
        this-
>setBlackAndScholes(CQuantModelServices::calculateCallBlackAndScholes(S, K, vol, R
, T));
   } else {
```

```
        this-
>setBlackAndScholes(CQuantModelServices::calculatePutBlackAndScholes(S, K, vol, R,
 T));
   }
}
```

Remarque : Ce composant définit la fonction de distribution selon la loi normale de façon approchée via une fonction linéarisée d'ordre 3.

Il s'agit du composant le plus simple et le plus facilement interchangeable des trois. Par exemple, on pourrait plugger une librairie calculant de façon plus précise chacune des valeurs. Le seul prérequis est que les noms des méthodes restent les mêmes afin de pouvoir être appelées depuis le composant interface qu'est le **PricingService**.
Evidemment, l'ajout, la modification ou la suppression de fonctionnalités dans le **MarketDataService** ou dans le **QuantModelService** implique de modifier le composant **PricingService** car c'est lui qui fait appel aux comportements implémentés par les autres composants.

4. Appels COM

Afin de pouvoir appeler les méthodes implémentées dans les **MarketDataService** et **QuantModelService** depuis le composant **PricingService**, dans chacun des deux premiers sont implémentés des interfaces décrivant des méthodes virtuelles pures. Cela permet de ne pas exporter les composants des classes mais seulement de définir des getters indépendants. Ainsi, pour le **QuantModelService**, cela donne le code suivant :

```
extern "C" QUANT_MODEL_SERVICES_API IQuantModel* __stdcall GetQuantModel();
```

Puis ce getter est défini comme retournant une nouvelle instance de la classe *CQuantModelService* :

```
IQuantModel *qm = new CQuantModelServices();
```

Cette stratégie permet à l'intérieur même des composants de définir éventuellement plusieurs comportements (ici la classe *CQuantModelServices*) implémentant les interfaces (ici l'interface *IQuantModel*).

Conclusion

Grâce à ce projet, j'ai pu appréhender les concepts clés de la programmation par composants et comprendre comment la communication entre ceux-ci peut se réaliser. Je regrette ne pas avoir pu envoyer entre les dlls et Excel des objets complexes tels que les structures. Peu de documentation semble exister à ce sujet sur internet. Cependant, cela m'a permis de bien comprendre les enjeux de la programmation par composants comme celle-ci est pratiquée en entreprise. Ainsi, les IT Quant vont plus travailler sur la librairie de calculs financiers, tandis que les développeurs vont travailler sur la librairie de récupération de données et la librairie d'interaction avec Excel. Enfin, les utilisateurs de la feuille Excel pourront customiser à volonté leur feuille simplement en ajoutant/supprimant des fonctionnalités. Dans la suite de ce projet pourraient être implémentés des crowlers, notamment pour récupérer les dernier taux Euribor avec la librairie Curl pour C++.

Annexe : Pré-requis

1) MySQL

Il vous faut une base MySQL. Un outil tel que WampServer devrait être satisfaisant. Afin de partir avec des données pré-existantes, exécuter le script *init.sql* joint avec ce projet. Ensuite, il vous faudra exécuter le crowler Java disponible à l'adresse https://github.com/philipperemy/Market-Data/. Pour la partie récupération des taux, vous êtes obligés d'exécuter le script correspondant car aucun crowler n'a été implémenté, ils devront être insérés quotidiennement à la main. Un site référence pour cela est le suivant : https://www.banque-france.fr/economie-et-statistiques/changes-et-taux/les-taux-interbancaires.html.

Si vous ne souhaitez pas exécuter le crowler Java, vous pouvez exécuter le script marketdata.sql en lieu et place du init.sql. Cependant, celui-ci est d'une taille supérieure à 2Mo et vous sera refusé à être uploadé dans la configuration normale de WampServer. Il vous faudra alors remonter la taille maximale des fichier pouvant être téléchargés.

2) Le PATH et VisualStudio

Afin de modifier l'application ou de pouvoir l'exécuter au travers de Visual Studio, il vous faudra ajouter à votre PATH :

```
C:\votre\chemin\jusqua\lapplication\OptionPricer\x64\Debug\
```

Une fois dans Visual Studio, il vous faudra redéfinir les chemins jusqu'aux librairies MySQL.

Ainsi, clic-droit sur le projet **MarketDataService > Propriétés > Configuration Properties > C/C++ > Général > Additional Include Directories** :

Ajouter `C:\votre\chemin\jusqua\lapplication\ MySQL-connector\include`
Ajouter `C:\votre\chemin\jusqua\lapplication\ MySQL \include`

clic-droit sur le projet **MarketDataService > Propriétés > Configuration Properties > Linker > General > Additional Include Librairies :**

Ajouter `C:\votre\chemin\jusqua\lapplication\MySQL-connector\lib`
Ajouter `C:\votre\chemin\jusqua\lapplication\MySQL\lib`

Faire de même pour le projet **PricingService** en n'oubliant pas d'ajouter dans les Additionnal Include Librairies :

`C:\votre\chemin\jusqua\lapplication\OptionPricer\x64\Debug`

Afin que **PrincingService** puisse avoir accès aux libairies **QuantModelService** et **MarketDataService**.

Rien n'est nécessaire dans le projet **QuantModelService**.

3) Excel

Côté Excel, afin que le code VBA et les dlls compilées en 64bits s'exécutent correctement, nous vous garantissons un bon fonctionnement si Excel est en **version supérieure ou égale à 2010** et si la version d'Excel installée est la **version 64 bits**.

Les fichiers de sortie doivent l'être dans x64\Debug. Il est possible de les supprimer mais attention dans ce cas à ne pas supprimer *libmysql.lib* et *libmysql.dll*.

Enfin, dans le code VBA, les librairies sont appelées en chemins absolus. Modifier les chemins en :

`C:\votre\chemin\jusqua\lapplication\OptionPricer\x64\Debug\PricingService.dll`