M2 MIAGE IF – DAUPHINE

Projet agile : améliorer la qualité des tests fonctionnels

Etudiant : Kamel SOUSSOU

Année : 2015 - 2016

Master 2 MIAGE
Informatique pour la Finance
Université Paris Dauphine

Maitre d'apprentissage :

Cyrille COMBES

Tuteur enseignant :

Tristan CAZENAVE

SOMMAIRE

Règles de confidentialité

Le présent document est confidentiel et susceptible de contenir des informations couvertes par le secret professionnel. Il peut contenir des informations sur les activités de Lyxor Asset Management. Ce document est établi à l'intention exclusive des membres du jury de l'université Paris Dauphine et de mon tuteur enseignant.

Toute utilisation ou diffusion non autorisée est interdite.

Remerciements

Je tiens à remercier mon maître d'apprentissage Cyrille Combes, responsable de la table de négociation de Lyxor Asset Management qui a su m'accompagner durant l'ensemble de cette année d'apprentissage au sein de son équipe. Je souhaite également remercier Christophe Ouvrard, Bertrand Leclercq et Stéphane Ravetier, négociateurs chez Lyxor, qui m'ont aidé grâce à leurs connaissances avancées sur les marchés financiers. Je tiens à remercier Eric Bief, Morgan Dupire, Jean-Paul Sov et Taher Khadraoui, membres de l'équipe IT Lyxor pour m'avoir accompagné durant mes développements.

Mes remerciements vont à mon tuteur enseignant Tristan Cazenave, professeur à l'Université Paris-Dauphine, pour m'avoir encadré ainsi que pour ses enseignements en programmation.

Enfin, je remercie Claude Vincent, responsable du suivi des apprentis au CFA-AFIA, qui m'a aidé dans mes démarches.

Introduction

Quatre mille milliards de dollars. C'est le volume quotidien qu'atteint le marché des devises selon la Banque des règlements internationaux, en hausse de 20 % sur les trois dernières années. Soit deux fois plus que celui des marchés actions. Dans ces conditions, les enjeux technologiques sont immenses. Toutes les banques, acteurs incontournables des marchés financiers, investissent dans l'amélioration de leurs plates-formes.

La poussée du trading électronique cette dernière décennie a eu une incidence profonde sur les marchés d'actions. L'automatisation a transformé le marché d'un endroit où une grosse opération était réalisée grâce à l'intermédiation humaine à une place où les ordinateurs font désormais la plus grosse partie du travail en réalisant et en traitant les opérations. Il y a plusieurs avantages à cela. Tout d'abord, cela permet de plus gros volumes de négociations, ce qui améliore la liquidité sur le marché, ensuite, la rapidité des exécutions et le coût des transactions ont fortement diminué pendant que la transparence a augmenté.

C'est dans ce contexte où la technologie et la finance vont de paires que j'ai été amené à travailler sur le projet que je vais vous présenter dans ce mémoire. Dans ce document, il sera question d'un outil développé par l'équipe IT Lyxor utilisé dans le cadre de l'asset management.

Tout d'abord, nous allons voir l'impact des technologies dans le secteur de la finance à travers un projet nommé OMX développé en méthodes agiles. Puis, nous soulèverons un problème concernant les tests fonctionnels en méthodes agiles. Enfin, nous verrons la solution qui a été développé pour répondre à ce problème et nous ferons un constat sur les résultats de cette solution.

1. Quand l'informatique se mêle à la finance…

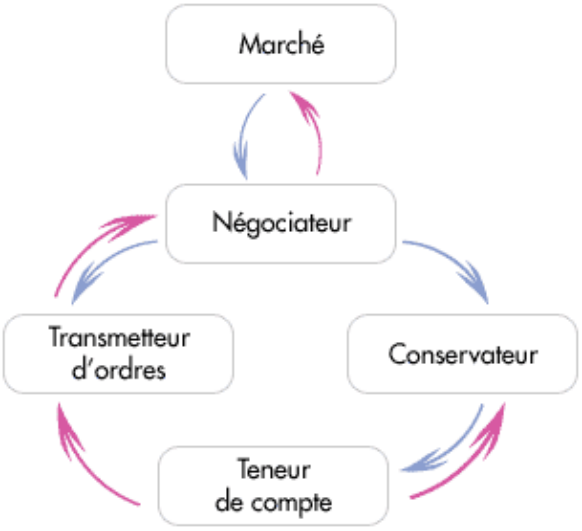
Les nouvelles technologies ont révolutionné la plus part des domaines d'activité sur le plan de l'échange d'informations, en particulier sur les marchés financiers.

Dans cette partie, nous allons commencer par voir un descriptif de l'utilité de l'informatique dans le cadre d'opérations financières à travers un projet développé en interne par l'équipe IT Lyxor. Ensuite, nous étudierons un problème rencontré concernant les tests fonctionnels de cet outil. Pour finir, nous verrons les différentes solutions existantes permettant de gérer ce problème.

1.1. Projet OMX

Dans le cadre de mon alternance chez Lyxor Asset Management, j'ai été amené à travailler sur un outil appelé OMX. Ce logiciel est un Order Management System (OMS) développé en interne par l'équipe IT Lyxor. La mise en place de l'OMX a pour objectif de centralisé les passages d'ordres en remplaçant les anciens OMS de Lyxor Asset Management. Ces derniers étaient pour la plus part des solutions achetées à des éditeurs de progiciel.

Le but de cet outil est d'automatiser les communications internes, permettant aux traders de collecter électroniquement les ordres et les instructions issues du porfolio manager.



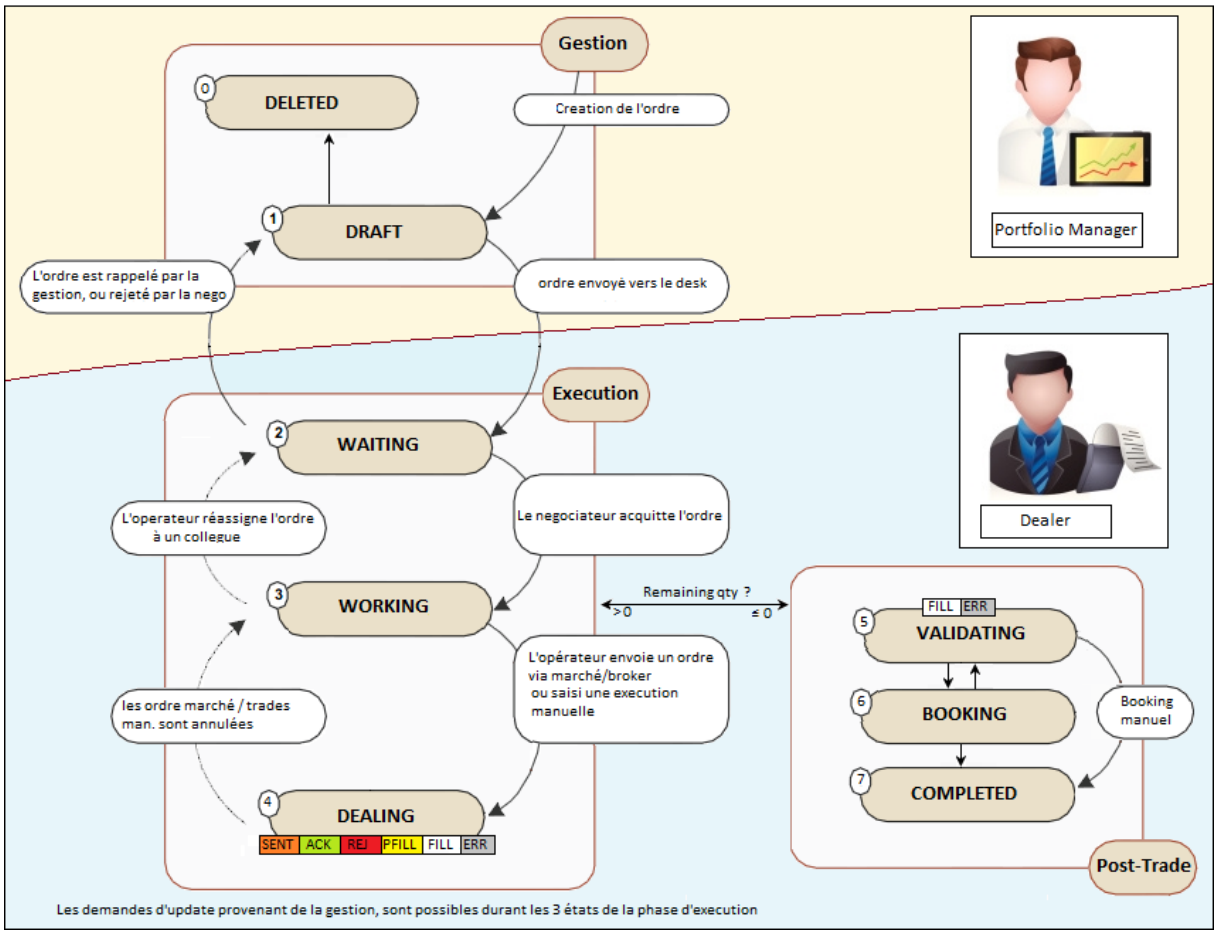
Ce schéma représente les différents flux dans le processus de mise en place d'un ordre sur le marché

Nous pouvons définir un ordre de Bourse comme étant un ordre passé par une personne ou une institution à un intermédiaire (courtier) afin d'acheter ou vendre en Bourse selon plusieurs paramètres (dates, produit financier, prix, …).

L'OMX est un « gestionnaire » d'ordres. Il permet de suivre tout le processus de mise en place d'un ordre depuis sa création à sa validation sur le marché concerné. Il fournit la connectivité directe aux plateformes de trading tel que les marchés électroniques et les direct market access platforms (DMA's). Il donne aux traders les moyens d'optimiser leurs stratégies d'investissement et d'exercer un contrôle accru de leurs exécutions.

Le suivi d'un ordre dans les systèmes de l'OMX se fait par une piste d'audit. Un ordre possède un état qui change en fonction des différentes actions de l'utilisateur.

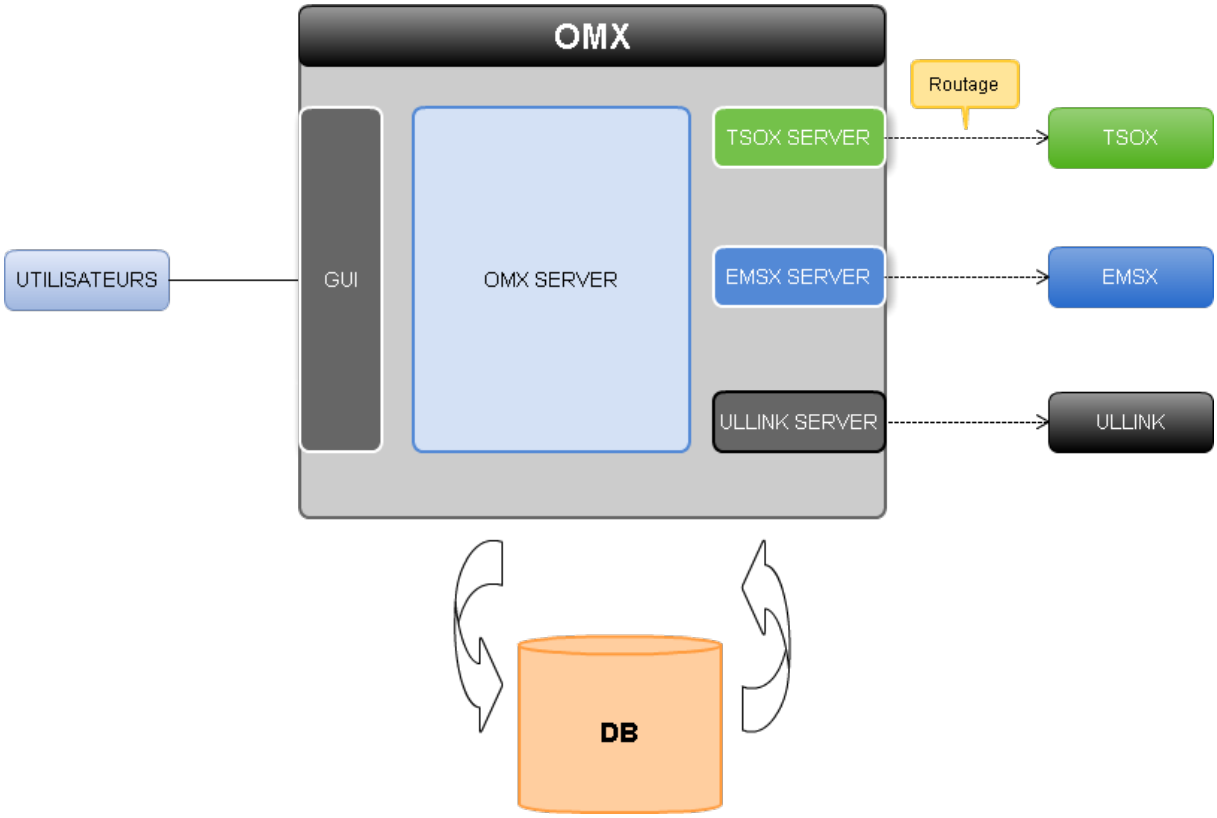
Voici une représentation de l'ensemble des états d'un ordre avec les actions associées :



Dans l'OMX, il existe plusieurs types de procédures, dont les suivantes :

- agréger les ordres sous forme de blocs
- gérer les exécutions
- assurer le reporting
- exécuter les allocations

On retrouve des fonctions spécifiques au Front, au Middle et au Back Office. Dans la suite de ce document, nous allons nous intéresser à une fonctionnalité dédiée au Front Office : **le routage des ordres vers les plateformes financières**.



L'architecture générale de l'OMX est de type Client/Serveur. Elle est composée de :

- GUI : « Graphical User Interface » est l'interface avec laquelle l'utilisateur communique. C'est cette partie qui permet aux utilisateurs de créer des ordres, de les envoyer vers un marché donné afin de les traiter, de consulter la piste d'audit, etc…
- OMX SERVER : c'est le serveur principal. Il permet d'interagir avec les demandes de l'utilisateur, de consulter/mettre à jour la base de donnée et de rediriger les ordres vers le serveur approprié.
- TSOX SERVER, EMSX SERVER, ULLINK SERVER : les serveurs qui permettent de rediriger les ordres reçu à la « Gateway » correspondante. Les « Gateway » sont les passerelles permettant de router un ordre vers les marchés financiers.

Le protocole utilisé pour la communication entre les différents serveurs est le protocole FIX (Financial Information eXchange, en anglais). C'est un standard de message développé dans le but de faciliter les échanges d'informations relatifs aux transactions boursières.

Comme nous pouvons le voir sur le schéma précédent, les ordres peuvent être routés vers différents marchés selon la nature du produit financier traité. C'est l'utilisateur qui choisit via quel serveur il souhaite traiter un ordre.

Voici un exemple d'utilisation des « Gateway » selon la nature du produit traité :

- TSOX : les ordres concernant les obligations
- EMSX : les ordres sur les actions
- ULLINK : les ordres sur les contrats « Future »

Ce projet a été développé en méthodes agiles, c'est ce que nous allons expliquer dans le prochain point. Les tests techniques couvrent l'ensemble des fonctions de l'OMX. En effet, il est possible d'effectuer des tests sur l'intégrité des données enregistrées en base, sur le comportement des ordres en fonction d'un changement d'état, etc...

Qu'en est-il des tests fonctionnels ?

1.2.Méthodes agiles : comment faire face aux tests fonctionnels ?

Le projet OMX a été développé en méthodes agiles avec un cycle de livraison de 15 jours en moyenne. L'équipe en charge de ce projet publie des « release » avec des améliorations en continu : elle est en contact direct avec les utilisateurs.

Commençons par une petite définition de ce que veut dire « méthodes agiles »...

Les méthodes agiles sont des méthodologies essentiellement dédiées à la gestion de projets informatiques. Elles reposent sur des cycles de développement itératifs et adaptatifs en fonction des besoins évolutifs du client. Elles permettent notamment d'impliquer l'ensemble des collaborateurs ainsi que le client dans le développement du projet. Ces méthodes permettent généralement de mieux répondre aux attentes du client en un temps limité (en partie grâce à l'implication de celui-ci) tout en faisant monter les collaborateurs en compétences. Ces méthodes constituent donc un gain en productivité ainsi qu'un avantage compétitif tant du côté client que du côté du fournisseur.

Dans ce contexte là, nous allons nous intéresser à une partie très importante : les tests avant une livraison. Le nombre de tests à effectuer est de plus en plus grand au fur et à mesure que le projet grandit.

Les tests techniques sont assurés par les tests unitaires développés par l'équipe en charge du projet OMX. Les nouveaux tests unitaires viennent se greffer à la suite des anciens et il « suffit » de les lancer et de vérifier que l'outil passe les tests avec succès.

Il existe des outils qui permettent de piloter les tests unitaires. Celui choisi pour notre projet est NUnit sur Visual Studio.

NUnit est un framework open source.

Concernant les tests fonctionnels, ce n'est pas très évident lorsque l'on développe un projet de cette taille avec un délai de livraison relativement court. En effet, il est nécessaire d'effectuer un grand nombre de tests de manière manuelle.

Les tests fonctionnels manuels sont nécessaires, et seront toujours nécessaires, en informatique comme dans les autres domaines. Imaginez une voiture qui serait vendue aux clients sans aucun essai sur route avec un conducteur humain…

Néanmoins, beaucoup de tests relativement basiques doivent être réalisés très régulièrement durant le cycle de vie d'un logiciel, ce qui rend leur exécution manuelle fastidieuse.

Dans ce mémoire, nous nous somme intéressé **au routage des ordres**. C'est cette partie là qui va être traitée.

La problématique qui s'est posée à nous est donc : **dans le cadre du développement d'un Order Management System en méthodes agiles, comment faire face aux tests fonctionnels concernant le routage des ordres avant une mise en production ?**

1.3.Solutions existantes ?

Il n'y a pas de solutions existantes au problème tel qu'il a été défini dans ce mémoire.

Cependant on retrouve des outils qui permettent de faire correspondre deux ordres (un achat et une vente sur un produit financier ayant les mêmes caractéristiques) et donner lieu à une exécution. Ces programmes se nomment « order matching system ». Ce sont des modules que l'on peut intégrer à un système de trading tel que l'OMX.

Au début de ce projet, je me suis inspiré du mode de fonctionnement de ces outils pour développer une solution.

Le processus d'exécution d'opérations sur des trades par « matching » des ordres d'achat avec des ordres de vente est effectué par des algorithmes qui déterminent la façon dont les ordres sont appariés et priorisés. Un algorithme connu, utilisé dans la confrontation des ordres, est FIFO (First In First Out).

Voici la liste de quelques « order matching sytem » :

- Match-Trade
- Patsystems Matching Engine
- Quik

2. Marché virtuel

Rappelons la problématique de ce mémoire : **dans le cadre du développement d'un Order Management System en méthodes agiles, comment faire face aux tests fonctionnels concernant le routage des ordres avant une mise en production ?**

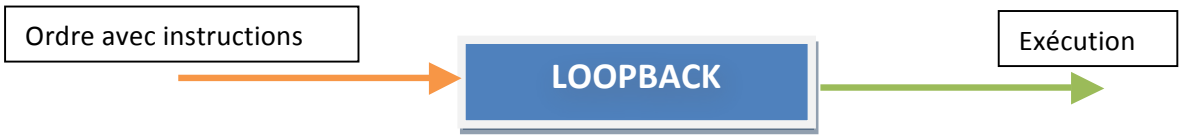
Dans cette partie, je vais exposer les solutions retenues pour traiter la problématique. Etant donné que nous sommes dans le contexte d'un projet développé en méthodes agiles, la solution a été développée selon la même philosophie. Mon délai de livraison était de 3 semaines en moyenne.

Pour commencer, je vais parler de la notion de « Loopback » et de marché virtuel. Ensuite, je présenterai les différentes commandes destinées à être utilisées par le Loopback et le résultat obtenu .Puis, je montrerai quelles mesures ont été prises pour mettre en place la solution. Enfin, je parlerai des apports et des limites de cette solution ainsi que les améliorations possibles.

2.1.Loopback

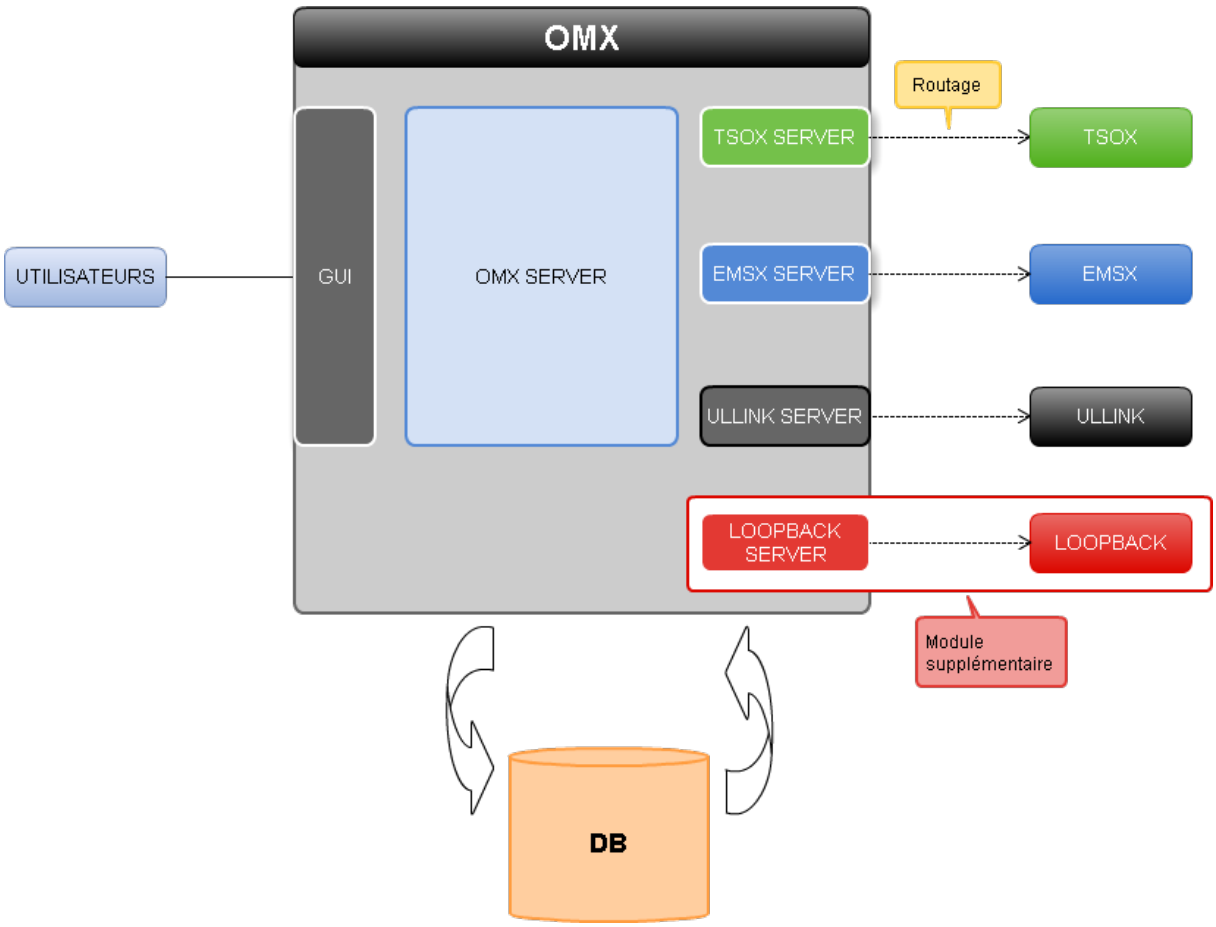
« Loopback » est un mot anglais signifiant « boucle arrière ». C'est une interface virtuelle d'un matériel réseau. Elle permet de rediriger le trafic vers une boucle locale. Dans notre contexte elle représentera un marché virtuel.

Afin de répondre à la problématique, nous avons simulé un marché qui permet de traité des ordres sur l'ensemble des produits financiers.



Le Loopback est un serveur qui vient se greffer à l'OMX comme un module indépendant. Il doit permettre de récupérer les ordres qui lui sont destiné afin de les exécuter selon des instructions.

Ci-dessous le schéma du nouveau système avec le Loopback dans l'OMX :



L'objectif du Loopback est d'être capable de recevoir un ou plusieurs ordres et de renvoyer une réponse de manière automatique au serveur de l'OMX (le serveur principal).

Dans un premier temps, j'ai commencé par analyser le fonctionnement des autres serveurs afin « d'imiter » le process pour le traitement d'un ordre. Lorsqu'un serveur reçoit un ordre, il le route vers le marché concerné.

Ci-dessous la piste d'audit d'un cas d'utilisation réel :

KIND	TRIGGER	STATUS	STATE	USER	DATE	TRADEID	DEAL ROUTE	DEAL STATE	BROKER RE
Single	Create	Done	Working	pierre.hereil	26/08/2016 16:56:53	0			
Single	MarketStateChange	Done	Dealing		26/08/2016 16:57:04	0	UllinkDma	Ack	TOS722234
Single	MarketSend	Done	Dealing	stephane.ravetier	26/08/2016 16:57:04	0	UllinkDma	Sending	
Single	AddTrade	Done	Validating		26/08/2016 23:11:16	199253	UllinkDma		

- L'ordre a été créé : il est passé en statut « Working »
- Il a été envoyé vers « Ullink » et il est passé en statut « Dealing »
- La dernière ligne correspond à l'exécution (Trade) sur cet ordre : il est passé en statut « Validating ». **Cette étape est générer à la suite du traitement de l'ordre sur l'outil de trading Ullink.**

Pour simuler ce traitement par le Loopback, il a fallut trouver un moyen de donner les instructions au serveur Loopback afin d'exécuté l'ordre de manière automatique. En effet, étant donné que le Loopback est un réseau virtuel, il faut automatiser l'étape de traitement d'un ordre sur un outil de trading.

Pour cela, j'ai choisi de donner des instructions via un champ sous la forme d'une chaine de caractère. Pour que l'ordre soit reconnu et exécuté par le Loopback, j'ai ajouté un « flag » au début de l'instruction : @LBK.

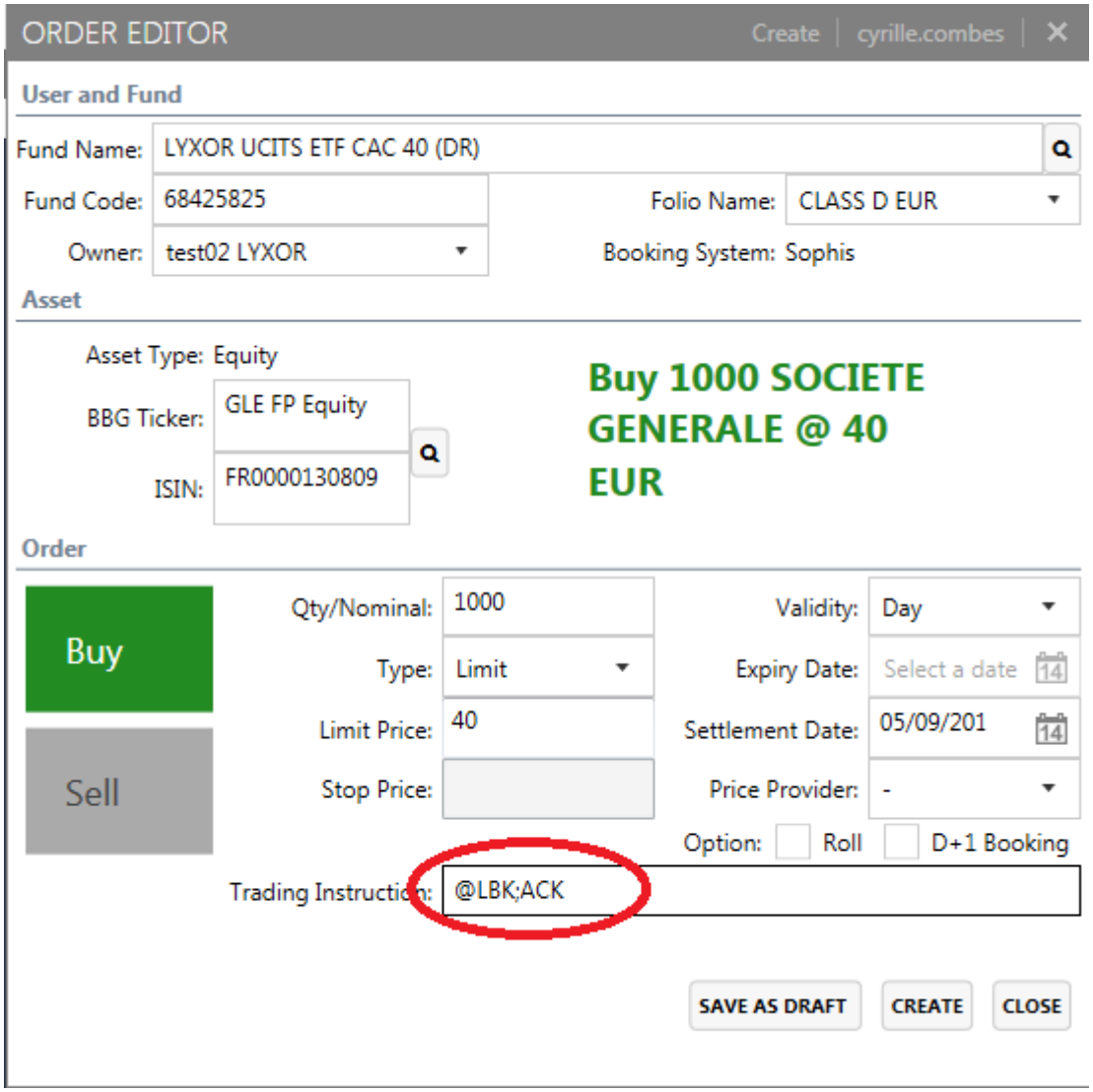
Les commandes à exécuter par le Loopback seront de la forme suivante : « @LBK;*INSTRUCTION* »

2.2.Commandes Loopback

Nous allons voir l'ensemble des commandes qu'il a fallu simuler par le Loopback. Ces commandes sont préfixés par le mot-clé « @LBK ».

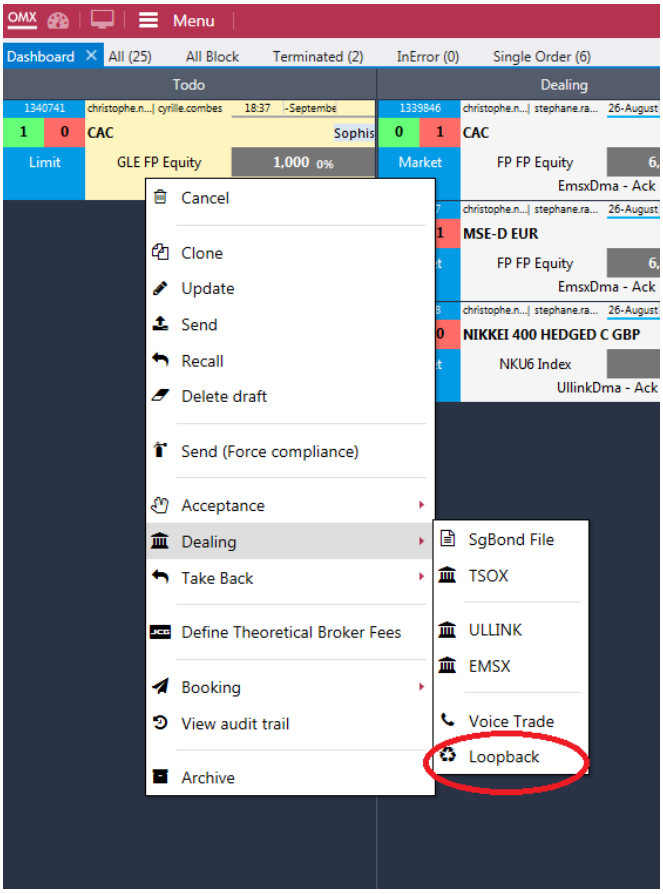
Commande	Effet	Exemple
ACK	Retourne un acquittement	@LBK;ACK
WAIT=100	Attend 100 ms	@LBK;WAIT=100
REJECT	Rejette l'ordre	@LBK;REJECT
EXEC	Retourne 1 exec de la quantité total de l'ordre	@LBK;EXEC
EXEC=150	Retourne 1 exec de quantité 150	@LBK;EXEC=150
EXEC=40%	Retourne 1 exec de 40% de la quantité de l'ordre	@LBK;EXEC=40%

Le premier objectif que je m'étais fixé a été de réceptionner un ordre avec une instruction telle que décrite ci-dessus et de simuler le comportement correspondant.



Tout d'abord, je vais créer un ordre via l'interface utilisateur de l'OMX. Intéressons nous à un champ en particulier : « Trading Instruction ». C'est dans ce champ que j'ai choisi de faire passer les instructions au Loopback. Nous allons tester la commande « @LBK;ACK » qui permet de simuler un acquittement depuis un marché financier.

Après avoir créé l'ordre, il suffit de l'envoyer vers le Loopback.



Ci-dessous le résultat obtenu. C'est la piste d'audit de l'ordre créé précédemment. On peut voir que le Loopback, à la réception de l'ordre, a renvoyé une réponse correspondant à l'acquittement de l'ordre.

	DATE	TRADEID	DEAL ROUTE	DEAL STATE	BROKER REF	BROKER ORDERSECREF	DEAL QUOTES	MESSAGE	QUANTITY	SID
oes	01/09/2016 18:37:38	0							1000	Buy
oes	01/09/2016 18:43:19	0	Loopback	Sending			(Collection)		1000	Buy
	01/09/2016 18:43:20	0	Loopback	Ack	2		(Collection)	Ack	1000	Buy

Il est également possible de concaténer plusieurs commandes. Exemple :

@LBK;ACK;WAIT=100;EXEC=10%;EXEC=100;WAIT=500;EXEC=20%;REJECT

- Acquitte l'ordre (instantanément)
- Attend 100 ms
- Retourne 1 exec de 10% de la quantité de l'ordre
- Retourne 1 exec de quantité 100
- Attend 500 ms
- Retourne 1 exec de 20% de la quantité de l'ordre
- Rejette l'ordre

C'est à partir de ce moment que le Loopback devient intéressant et utile car on va pouvoir automatiser un ou plusieurs comportements de manière automatique grâce à une série de commandes.

La dernière étape a été de développer la fonction qui permet de répéter une ou plusieurs commandes spécifiques.

Commande	Effet	Exemple
REPEAT=10{**INSTRUCTION**}	Répète **INSTRUCTION** spécifié entre '{' et '}' 10 fois	@LBK;REPEAT=10{*INSTRUCTION*}

NB: ***INSTRUCTION*** est à remplacer par ACK, WAIT, WAIT=10, EXEC, EXEC=50%, REPEAT=40{EXEC=2;ACK}, ...

Exemple	Effet
@LBK;REPEAT=100{EXEC=1%}	Génère 100 execs de quantité 1% chacune

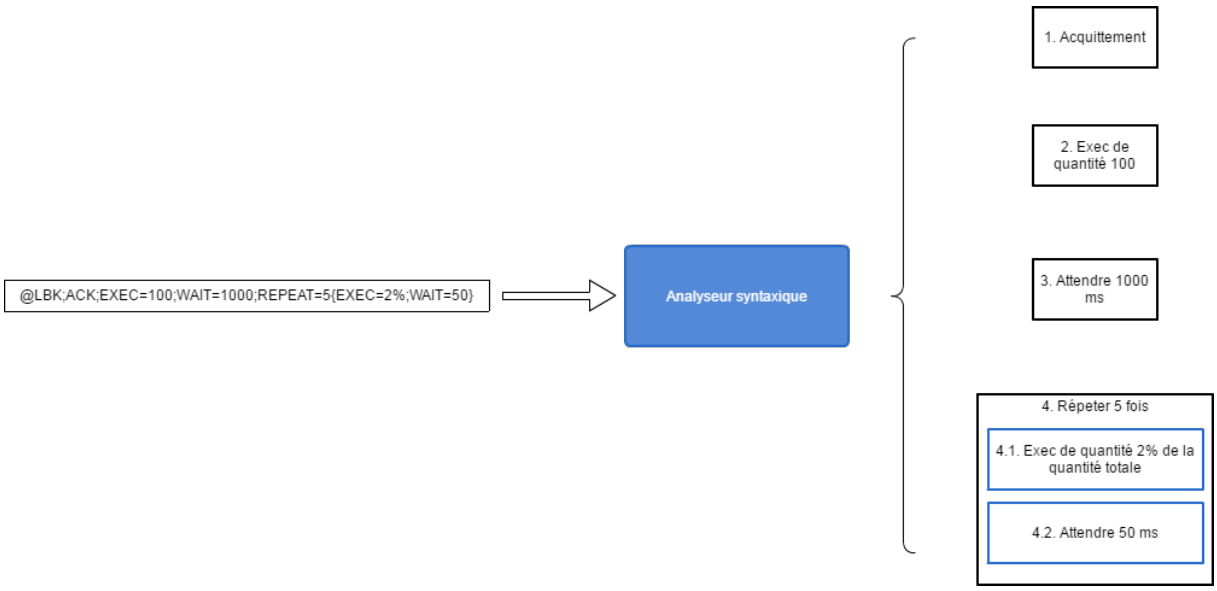
En utilisant toutes les commandes développée on peut complexifier l'instruction pour obtenir par exemple : @LBK;REPEAT=100{EXEC=1%;REPEAT=50{WAIT=10}}

2.3.Mise en œuvre de la solution

La mise en place du Loopback s'est faite en collaboration avec l'équipe IT Lyxor. Ils se sont chargés de l'intégration de ce module dans l'OMX.

La solution retenue pour générer des exécutions sur des ordres a été de récupérer des commandes via une chaine de caractères. La structure de la chaine est la suivante : @LBK;*INSTRUCTION*

J'ai développé un analyseur syntaxique permettant d'extraire des mots clé d'une chaine de caractère. En fonction de ces mots clé, j'ai simulé un comportement sur un marché (achat d'un produit, vente, aucune réponse du marché, exécution partielle d'un ordre, rejet d'un ordre du marché, etc...).



Comme nous pouvons le voir sur le schéma ci-dessus, l'instruction
« @LBK;ACK;EXEC=100;WAIT=1000;REPEAT=5;{EXEC=2%;WAIT=50} » est traduite par :

- Un acquittement
- Un trade de quantité 100
- Attente de 1000 ms
- Répéter 5 fois : un trade de quantité 2% suivi d'une attente de 50 ms

Pour simuler une action du marché, j'ai travaillé sur des cas réels afin de comprendre la cinématique.

2.4.Bilan de la solution

J'ai développé une solution qui permet d'améliorer la qualité des livraisons dans le cadre d'un projet agile. Comme expliqué dans les parties précédentes, dans un contexte où les mises en production sont nombreuses, l'enjeu en termes de tests fonctionnels est très important. En effet, l'ensemble des tests fonctionnels effectués de manière manuelle se traduisent par un coût au niveau du temps et par des éventuels oublis humains.

L'équipe IT a eu l'occasion de tester le résultat de ma solution et cela leur a permis de gagner un temps précieux. Il leur a suffit de rajouter, dans leur process avant une mise en production, la création d'ordres avec des instructions bien précises afin de les exécuter via le Loopback.

La solution a été mise en place sous la forme d'un module complémentaire à l'OMX.

Après avoir testé cette solution, leur constat a été le suivant : il est plus simple d'effectuer un test sur un ordre. Avec un nombre limité de commandes (une dizaine), la solution proposée est adéquate au problème. Cependant si la liste des commandes vient à s'agrandir, cette solution deviendra moins évidente à maintenir du point de vue technique. En effet, il faudra tester tous les cas de figure dans le code avec des conditions et cela complexifiera le code.

2.5.Améliorations possibles

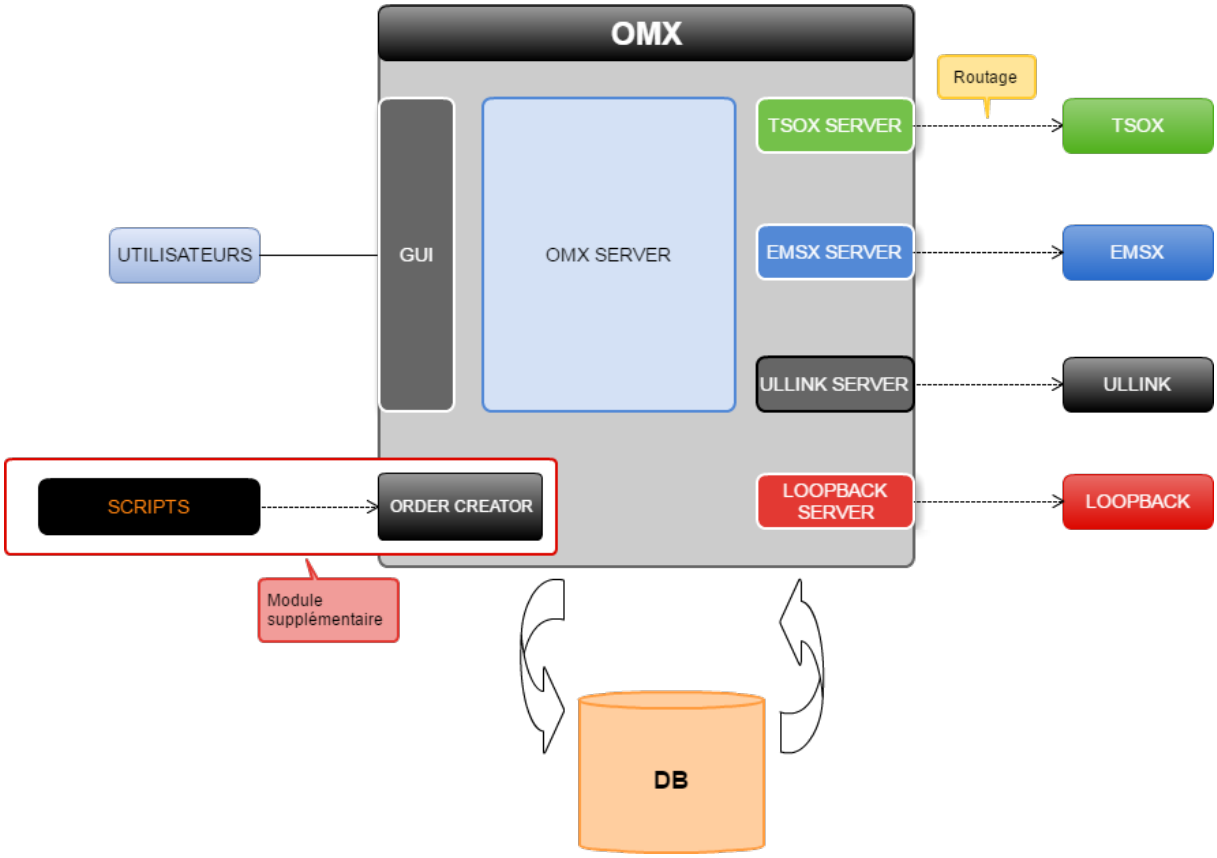
Dans mon mémoire je m'étais fixé l'objectif de répondre à un besoin bien précis concernant le **routage des ordres**. Pour cela, j'ai mis en place un module complémentaire (le Loopback) qui permet de tester de manière automatique le comportement d'un ordre dans l'OMX.

Si on prend un ordre depuis sa création dans l'OMX jusqu'à sa validation et son archivage, on voit bien que la solution proposée ne permet d'automatiser les tests fonctionnels que sur une partie du process. Une amélioration possible de la solution est de rendre automatique l'ensemble du cycle de vie d'un ordre.

Pour cela, il faudrait développer un module complémentaire qui permet de créer un certain nombre d'ordres de manière automatique (un script) avec des instructions destinées au Loopback. La deuxième étape serait d'envoyer les ordres au serveur du Loopback (toujours de manière automatique). Pour la dernière étape, il suffirait d'écrire dans un fichier de logs toutes les pistes d'audit des ordres avec un examen de l'ensemble des tests effectués et le résultat obtenu.

Avec un tel module, il serait plus simple pour les développeurs d'effectuer leurs tests avant la livraison d'une nouvelle fonctionnalité de l'OMX.

Voici un schéma du système avec la solution améliorée :



Les avantages d'une telle solution sont :

- Gain de temps considérable

- Automatisation de l'ensemble du cycle de vie d'un ordre
- Possibilité de créer un très grand nombre d'ordres afin d'effectuer les tests fonctionnels (10000 par exemple)
- Stresse de l'application OMX
- Etudier les points de ruptures de l'OMX
- Tester les limites d'utilisation en cas de forte charge

Cette solution répondrait à la problématique suivante : **dans le cadre du développement d'un Order Management System en méthodes agiles, comment améliorer la qualité des tests fonctionnels ?**

Conclusion

Dans ce mémoire la problématique que j'ai essayé de solutionner était liée aux tests fonctionnels avant le processus de mise en production dans le cadre d'un projet développé en méthodes agiles : le projet OMX. Je me suis intéressé à l'une de ses fonctionnalités qui est le routage des ordres. Le but était, durant l'élaboration de ce mémoire, d'améliorer la qualité des tests fonctionnels sur les ordres dans l'OMX. Grâce au Loopback, module mis en place pour effectuer les tests, j'ai su développer une architecture capable de simuler des comportements spécifiques sur un marché virtuel. La solution expliquée dans ce mémoire a été possible grâce à trois étapes. Dans un premier temps, il a fallut intégrer un réseau virtuel à l'OMX afin de simuler le marché financier. La seconde étape, a été d'analyser et extraire des données depuis une instruction. La dernière étape a été de générer les exécutions sur un ordre en fonction des commandes extraite de l'instruction. Cette solution a permis d'automatiser l'aspect fonctionnel des tests avant une livraison. Elle a également ouvert la possibilité d'automatiser l'ensemble du cycle de vie d'un ordre depuis sa création, à son exécution et à son archivage.

Pour ma part, l'élaboration de ce mémoire m'a aidé à identifier un problème lié au contexte actuel des nouvelles technologies et des méthodes utilisées dans le cadre du développement de projet. J'ai appris à chercher de l'information et à proposer une solution pour parvenir à résoudre une problématique. J'ai eu la chance de pouvoir faire tester ma solution par les personnes concernées et avoir un retour sur les avantages et les inconvénients de celle-ci.

Ce projet m'a également permis de monter en compétence, tant en terme technique qu'en terme fonctionnel.

Bibliographie

http://www.fimarkets.com/pages/oms.php

http://www.fimarkets.com/pages/ecn.php

http://ineumann.developpez.com/tutoriels/alm/agile_scrum/

http://www.match-trade.com/matchengine.html

https://arqatech.com/en/products/quik/modules/infrastructure-solutions/quik-matching-engine/

Glossaire

Ordre : Un ordre de Bourse est un ordre passé par une personne ou une institution à un intermédiaire (courtier…) pour acheter ou vendre en Bourse.

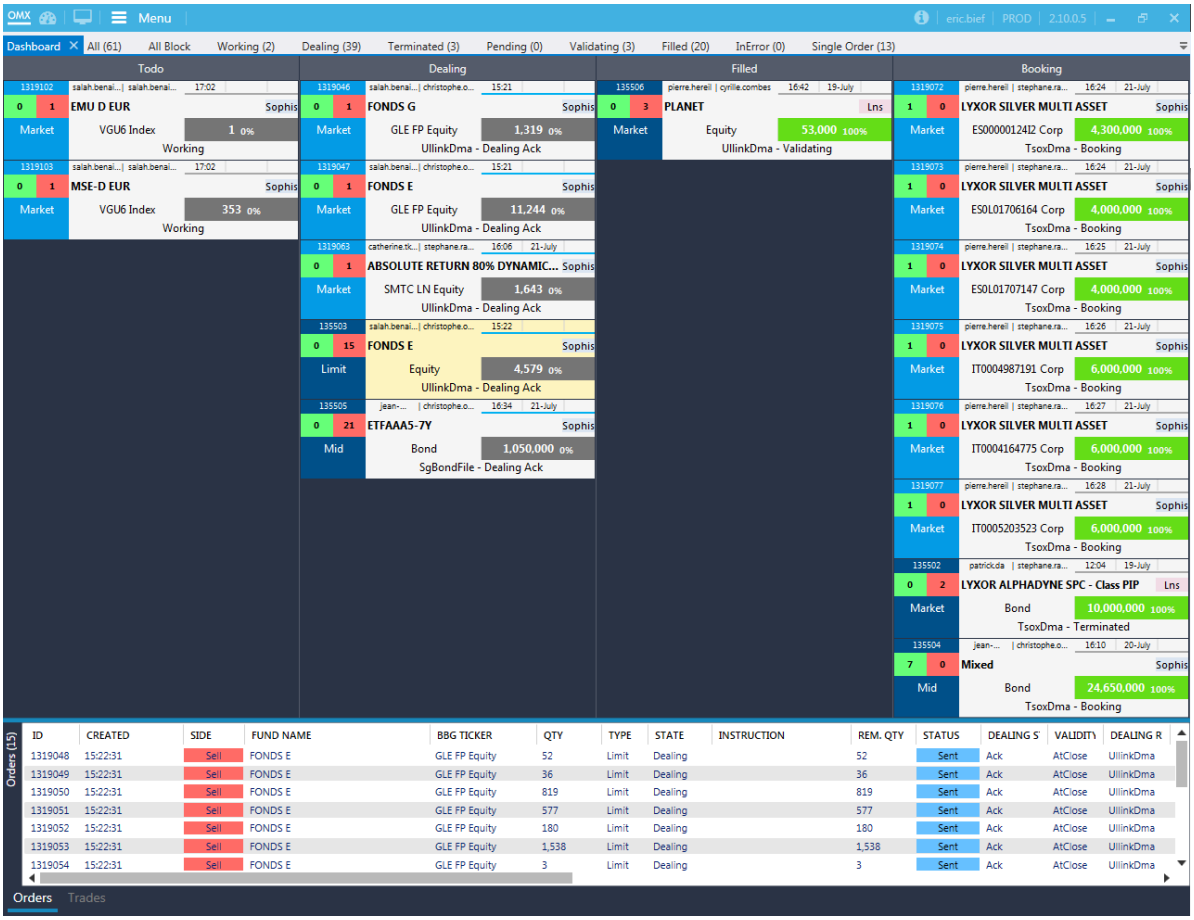
Trade : Un trade est le résultat de l'exécution d'un ordre sur les marchés financiers.

Gateway : (en français, passerelle) désigne un dispositif permettant de relier deux réseaux distincts présentant une topologie différente.

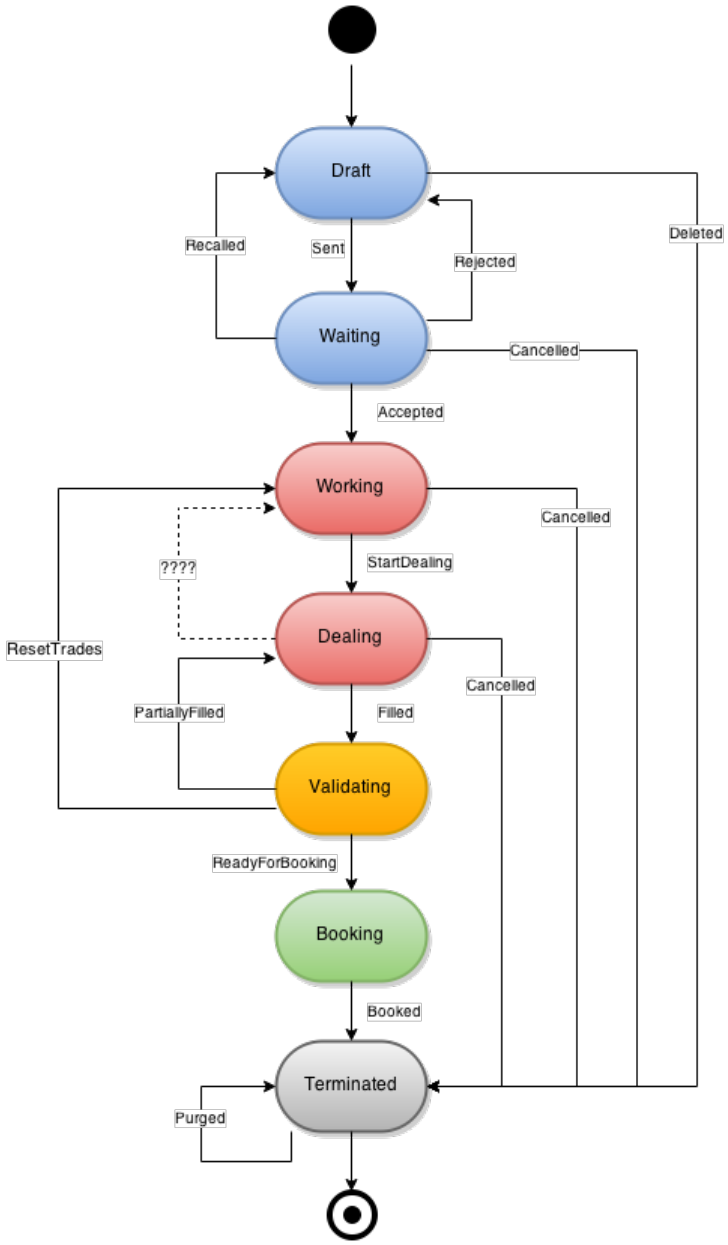
OMS : Un Order Management System est un gestionnaire d'ordres de marché

Test : En informatique, un test désigne une procédure de vérification partielle d'un système. Son objectif principal est d'identifier un nombre maximum de comportements problématiques du logiciel afin d'en augmenter la qualité (si les problèmes identifiés lors des tests sont corrigés). Néanmoins, le test peut aussi avoir pour objectif d'apporter des informations quant à cette qualité afin de permettre la prise en décisions.

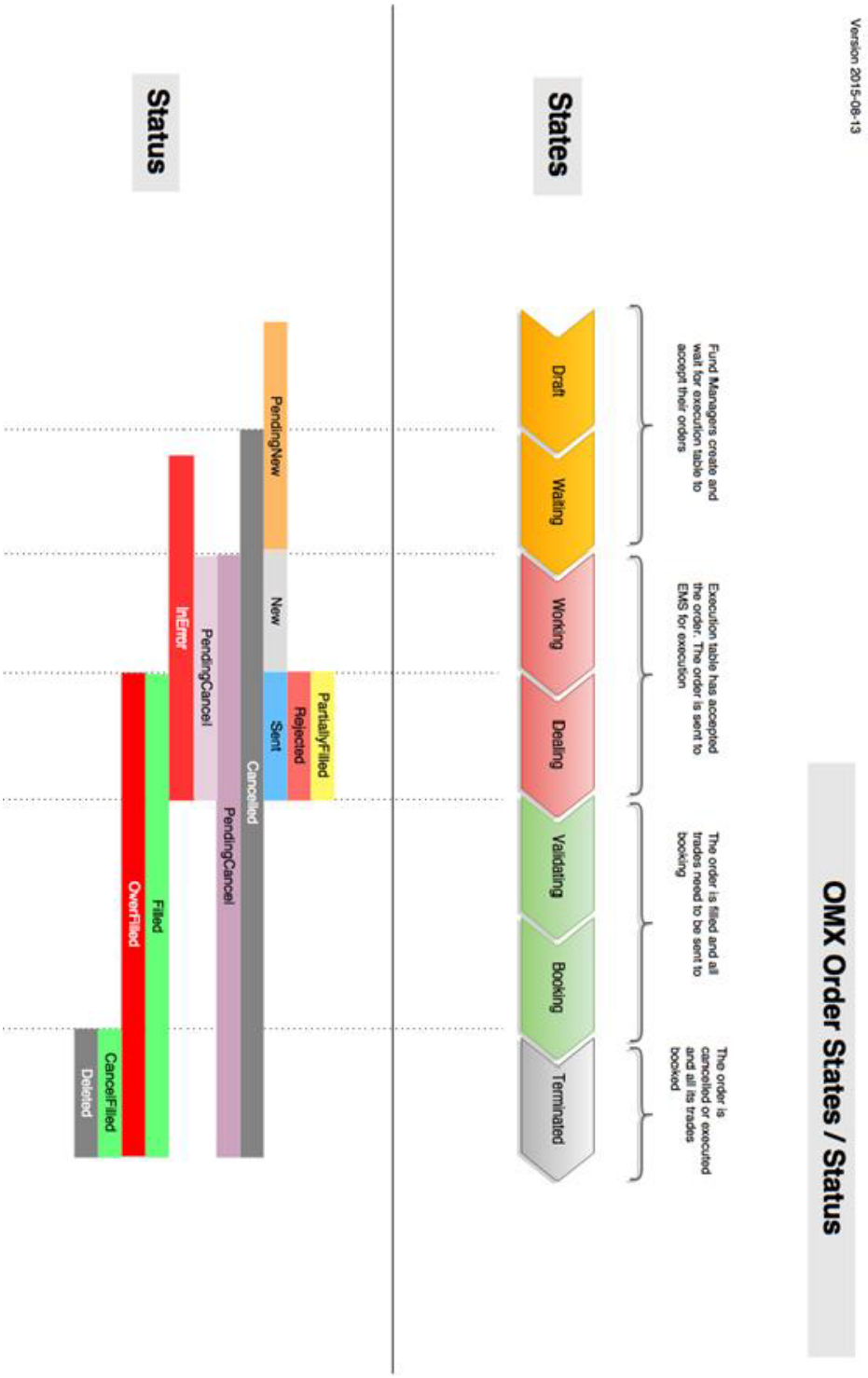
Annexes



1. **GUI - OMX**



2. Schéma 1 du cycle de vie d'un ordre



3. Schéma Omx Order States/Status