Laetitia PHAM
Maxence BRUNET

Enseignant : *Jacques* MALENFANT
Groupe : LAMA

UE CPS

Compte-Rendu

Mars 2020

Science et Technologie du Logiciel
Sorbonne Université
Année 2019/2020

1 Implantation des composants

Nos composants sont situés dans le répertoire *src/components* (paquetage components):
- Broker.java : le courtier assure la gestion des messages publiés et les souscriptions.
- Publisher.java : le producteur publie des messages.
- Subscriber.java : le consommateur souscrit à des sujets et reçoit les messages associés à ces sujets.

Nos interfaces sont situés dans le répertoire *src/interfaces* (paquetage interfaces):
- ManagementCI.java qui implémente ManagamentImplementationI.java et SubsciptionImplementationI.java.
- RecptionCI.java qui implémente ReceptionImplementationI.java
- PublicationCI.java qui implémente PublicationsImplementationI.java

Nos ports sont situés dans le répertoire *src/ports* qui contient les classes java: ManagementCInBoundPort, ManagementCOutBoundPort, PublicationCInBoundPort, PublicationCOutBoundPort, ReceptionCInBoundPort et ReceptionCOutBoundPort.

Nos connecteurs sont situés dans le répertoire *src/connectors* qui contient les classes java: ManagementConnector, PublicationConnector et ReceptionConnector.

2 Implantation des Messages et des Tests unitaires

Les classes permettant l'implantation d'un message se situent dans le répertoire *src/annexes/message*. De plus, Message.java implémente MessageI.java se situant dans *src/annexes/message/interfaces*.

Un message est définie comme étant un objet Java sérialisable possédant un identifiant unique. Chaque message possède un ensemble de propriétés définit dans Properties.java, ainsi qu'une estampille de temps définit grâce à la classe TimeStamp.java. En effet, cela permet aux abonnées de filtrer les messages selon des conditions décrite grâce à l'interface fonctionnel *MessageFilterI.java*.

De plus, pour vérifier leur bon fonctionnement, nous avons créé des tests unitaires pour chacune d'entre elles:
- MessageTest.java : Teste les différentes méthodes présentes dans la classe Message.
- PropertiesTest.java : Teste toutes les propriétés présentes dans la classe Properties.
- TimeStamp.test.java : Teste les différentes méthodes présentes dans la classe TimeStamp.

3 Greffons Producteur-Consommateur

Nos greffons sont situés dans le répertoire *src/plugins* (paquetage plugins):
- PublisherPlugin.java : permet de factoriser les fonctionnalités des producteurs (composant Publisher), ainsi le greffon implémente les interfaces PublicationsImplementationI et ManagementImplementationI à la place du Publisher. En effet, les composants souhaitant utiliser ces fonctionnalités n'auront qu'à installer ce greffon.
- SubscriberPlugin.java : permet de factoriser les fonctionnalités des souscripteurs (composant Subscriber), ainsi le greffon implémente les interfaces SubscriptionImplementationI et ReceptionImplementationI. Toutefois les composants souhaitant utiliser ces fonctionnalités devront implanter l'interface ReceptionImplementationI.

4 Structures de données utilisées et Concurrence

En ce qui concerne les structures de données utilisées pour le Broker:
- topics : correspond à une HashMap qui associe un sujet à sa liste de messages.
- subscibers : correspond à la liste de l'ensemble des souscripteurs existant et ayant déjà souscrit à au moins un sujet.
- subscriptions : correspond à une HashMap qui associe un sujet à sa liste de souscripteurs.

-CLIENT.JAVA : représente un souscripteur, celui-ci est définit par l'URI du port "in", un port de réception "out" et une Hashmap qui associe un sujet avec un filtre.

Lorsqu'un "Publisher" publie un message dans un sujet et que celui-ci existe, il est alors ajouté dans *topics*. Sinon le sujet est créé puis le message est ajouté. De manière analogue se déroule la publication de plusieurs messages sur plusieurs sujets. Pour ce qui en est de la destruction d'un sujet, celui-ci est retiré de *topics*. Cela ne gêne pas la livraison car un clone des souscriptions a été réalisé pour l'envoie.

Lorsqu'un "Subscriber" souscrit à un sujet et qu'il ne s'est jamais abonné auparavant, le courtier va dans un premier temps le créer en établissant une connection entre leurs ports. Puis l'ajouter dans la liste de l'ensemble des souscripteurs existant. Ensuite, la référence de ce Client sera ajoutée dans *subscriptions*. A tout moment le souscripteur peut modifier un filtre lié à un sujet. Dans ce cas, la méthode associée est appelée dans la classe Client.

En ce qui concerne la concurrence, nous avons choisi d'utiliser un verrou de type *ReentrantReadWriteLock* (nommé "lock" dans notre projet). Ainsi chaque service doit demander l'autorisation d'accès en lecture ou en écriture. Cependant pour toute écriture, celle-ci est réalisée en exclusion mutuelle avec toute autres opérations. En effet, certaines lectures peuvent être longue (par exemple *getSubscriptions* qui copie la liste des Clients), cela nous permet donc de gagner du temps tout en étant concurrent.

De plus, nous avons crée dans le Broker, un pool de thread pour les souscriptions et un autre pour les publications. Nous utilisons ces pools pour permettre au producteur (réciproquement consommateur) d'effectuer plusieurs appels de méthode (ex: publish/subscribe) et ce, de manière concurrente. De plus dans les ports, nous avons mis en place des appels asynchrones pour permettre un meilleur parallélisme entre les threads car l'appelant et l'appelé s'exécutent en parallèle.

5 Tests d'intégrations

Nous avons effectué de nombreux scénarios afin de couvrir au maximum les différentes fonctionnalités proposées entre les composants. Pour lancer ces tests, il suffit d'exécuter CVM.JAVA qui se situe dans le répertoire *src/launcher*.

En ce qui concerne les scénarios liés à *Publisher* (dans exécute):
- SCENARIO 1: publication de 10 000 messages pour un sujet donné pour vérifier si l'afflux d'une grosse quantité de messages n'engendre aucun blocage et si tous les messages ont pu être envoyé aux souscripteurs (abonnées à ce sujet).
- SCENARIO 2: publication d'un message dans différents sujets, puis plusieurs messages dans différents sujets pour vérifier le bon fonctionnement de nos méthodes.
- SCENARIO 3: Ce scénario consiste à créer un sujet et publier un message sur celui-ci. Le but est de vérifier que le sujet à bien été crée par la fonction de création d'un sujet.
- SCENARIO 4: Suppression d'un sujet.
- SCENARIO 5: On publie un message avec certaines propriétés pour tester le filtrage. Dans ce scénario, si des messages contiennent le mot "thon" pour la rubrique PecheCuisine, le souscripteur abonné et ayant ce filtre sur ce mot ne recevra pas ces messages.

En ce qui concerne les scénarios liés à *Subscriber* (dans start):
- SCENARIO 1: Ce scénario est lié au scénario 5 du Publisher. Le souscripteur souscrit au sujet "PecheCuisine" et ajoute un filtre (sur "thon"). A chaque fois qu'un message sera publié dans ce sujet, il sera filtré avant d'être envoyé au souscripteur.
- SCENARIO 2: Un souscripteur s'abonne à plusieurs sujets.
- SCENARIO 3: On vérifie si l'abonnement d'un très grand nombre de souscripteurs est réalisé sans aucun blocage ou erreur d'abonnement d'un souscripteur.