Compte rendu final pour le projet de l'UE CPS.

Pablito Bello et Pierre-Octave Velly.

Evolution depuis le compte rendu de mi-semestre

- L'ensemble des remarques indiqués dans votre fiche d'évaluation de mi-semestre ont été corrigés, sauf le point suivant :
    - Dans BrokerInboundPortMessage, vos choix entre appels synchrones ou asynchrones ne sont pas clairement justifiés. Ne voyez-vous pas de différences entre les méthodes subscribe et les méthodes de gestion des topics, par exemple ?

Que nous n'avons pas été en mesure de comprendre.

- La Javadoc est présente désormais sur tout le projet.
- Un système de log sur les composants a été rajouté (activé uniquement en multi-jvm).
- L'étape 3 a été réalisée.
    - Voici les classes ajoutées depuis le rendu de mi-semestre :
    - package ports : la classe BrokerPublicationOutboundPort a été ajoutée, elle permet au Broker de transmettre les messages réçu à un autre Broker.
    - package utils : la classe Log a été ajoutée, elle contient une méthode qui permet de print et log une chaine de caractère.
    - package deployment : la classe DistributedCVM a été ajoutée, elle contient un exemple de déploiement multi-jvm pour l'étape 3.
- L'étape 4 n'a pas été réalisée.

Tests d'intégration

N'hésitez pas à ajuster la durée de déploiement du projet dans le Main de CVM / DistributedCVM pour pouvoir lire les logs plus longuement si besoin en multi-jvm, nous l'avons fixé à 30 secondes.

Mono-jvm

Pour tester le projet en mono-jvm, on propose de lancer la classe deployment.CVM qui illustre un exemple de déploiement mono-jvm.

Un Subscriber, Un Broker et un Publisher sont déployés.

Tout d'abord, le Subscriber va tester les méthodes contenues dans ManagementImplementationI, ces tests sont décrits dans le terminal.

Le Publisher va ensuite tester les différentes méthodes publish. Il envoie d'abord 5 messages sur topic2, nommés de msg_i_1 à msg_i_5.

Subscriber reçoit bien ses messages, il a en effet souscrit au topic2, il affiche les messages.

Broker affiche des informations concernant les threads utilisés, que ce soit pour la réception ou l'envoie.

Publisher va maintenant envoyer un message msg2 sur topic2 et topic3 : le but est ici de vérifier que subscriber ne reçoit pas le message en double : ok.

Il envoie ensuite les messages msg3, msg4, mgs5 et msg6 pour tester les différentes méthodes publish. Le subscriber les reçoit : ok.

Puis finalement il envoie :

- msg7 sur topic1 , subscriber ne le reçoit pas car il a mis un filtre qui est toujours à false sur topic1 : ok.
- msg8 sur topic4 , subscriber ne le reçoit pas car il n'est plus abonné à topic4 : ok.

Multi-jvm

Afin de pouvoir tester l'exécution en multi-jvm, il faut d'abord modifié le fichier config.xml à la racine du projet en indiquant le chemin où se trouve le projet sur votre ordinateur.

On ouvrira ensuite 4 terminaux, dans lesquels on sera placé à la racine du projet.

Voilà les commandes à saisir :

terminal 1 :

```
./start-gregistry
```

terminal 2 :

```
./start-cyclicbarrier
```

termminal 3 :

```
./start-dcvm jvm1
```

termminal 4 :

```
./start-dcvm jvm2
```

Dès lors, 8 fenêtres de logs s'ouvrent : une pour gregistry, une pour cyclicbarrier et une part composant de chaque jvm. (attention les fenetres des composants sont superposés, il faut les bouger pour voir les composants de l'autre jvm).

Le point d'entrer de l'execution multi-jvm est le fichier deployment.DistributedCVM, on lance le programme sur 2 jvm.

Chaque jvm contient un Broker, un Subscriber et un Publisher.

Les tests sont les mêmes que pour le déploiement mono-jvm, mais les Brokers se transmettent les messages.

- Le Publisher de la jvm1 publie sur le Broker de la jvm1, qui transmet au Subscriber de la jvm1 et au Broker de la jvm2, qui va transmettre au Subscriber de la jvm2.
- Le Publisher de la jvm2 publie sur le Broker de la jvm2, qui transmet au Subscriber de la jvm2 et au Broker de la jvm1, qui va transmettre au Subscriber de la jvm1.

Les Subscriber reçoivent donc deux fois les messages décrits dans la partie mono-jvm, mais l'URI du publisher est affiché, on peut donc vérifier qu'on reçoit bien les messages d'un Publisher présent sur une autre jvm.

Les cycles de transmission des messages entre Broker sont évités : A chaque fois qu'un Broker reçoit un message, il vérifie qu'il ne l'a pas déjà reçu (via une méthode de la classe Message), s'il ne l'a pas déjà reçu, il le marque comme reçu (directement dans l'objet message) et le transmet à son Broker voisin, s'il l'a déjà réçu, il ne fait rien.

Un message déjà marqué comme reçu correspond à un Message qui est déjà passé par tout les Broker. Il n'a donc pas besoin d'être transmit.