Compte-rendu à mi-semestre du projet de l'UE Composant

Dans le cadre de l'UE CPS, nous avons pour projet de réaliser un système minimal de publication/souscription en utilisant le modèle de composant BCM4Java.

Implantation des trois types de composants, des interfaces de composant, des ports et des connecteurs :

Le package *src/components* contient les différents composants :
- Broker
- Publisher
- Subscriber
- DynamicAssembler (qui crée les autres composants dynamiquement)

Le package *src/interfaces* contient, entre autres, les interfaces de composant :
- ManagementCI
- PublicationCI
- ReceptionCI

Le package *src/ports* contient les ports utilisés par les composants/plugin :

utilisés par le Broker :
- BrokerManagementInboundPort
- BrokerPublicationInboundPort
- BrokerReceptionOutboundPort

utilisés par les plugins :
- ManagementOuboundPortForPlugin
- PublicationOutboundPortForPlugin
- ReceptionInboundPortForPlugin
- ReceptionOutboundPortForPlugin

Le package *src/connectors* contient les connecteurs :
- ManagementConnector
- PublicationConnector
- ReceptionConnector

Implantation complète des messages avec les propriétés et leurs tests unitaires :

Le package *src/message* contient :
- Message
- Property
- TimeStamp
- Topic

Le package *src/tets* contient :
- TestMessage
- TestProperty
- TestTimeStamp
- TestTopic
- RunTest

Les quatre premières classes vérifient juste le caractère valide des attributs des objets ainsi que les méthodes de base de Message, Property, TimeStamp et Topic.
Enfin, la classe RunTest se charge d'exécuter tous les tests précédemment cités.

Greffons clients pour les publieurs et les abonnés :

Le package *src/plugin* contient :

- PublisherClientPlugin - SubscriberPlugin

Ces plugins, qui vont s'installer respectivement sur Publisher et Subscriber vont permettre d'instancier dynamiquement ces types de composants.

PublisherClientPlugin utilise les ports suivants :

- PublicationOutboundPortForPlugin - ManagementOutboundPortForPlugin

Et SubcriberPlugin :

- ReceptionInboundPortForPlugin - ManagementOutboundPortForPlugin

Structures de données et gestion interne du courtier (incluant la gestion de la concurrence) :

Le Broker gère les topics / messages de la façon suivante :
Il possède une Map nommée topics de type : HashMap<String, Topic>, la String correspond au nom du topic et doit donc être unique. Les objets Topic stockés représentent quant à eux les sujets des messages.

Un objet de la classe Topic contient :

- Une ArrayList<MessageI> : représente la liste des messages publiés sur le Topic
- Une Hashmap<String, MessageFilterI> : représente, pour chaque subscriber (contactable sur l'URI qui correspond à la String), le filtre qu'il a choisi d'utiliser pour ce Topic.

La gestion de la concurrence interne au courtier se fait via les moyens suivants :

- Deux pools de threads (*envoi* et *réception*) :
    - Les threads d'*envoi* sont utilisés uniquement pour envoyer des messages aux abonnés
    - Les threads de *réception* sont utilisés pour recevoir toutes les requêtes sur le Broker, que ce soit sur ManagementCI ou PublicationCI, nous avons fait le choix de ne pas créer un troisième pool uniquement pour les requêtes concernant ManagementCI car nous estimons que dans une utilisation classique il y a beaucoup moins de requêtes sur ManagementCI que sur PublicationCI ou ReceptionCI.

- Un système de verrou, implémenté par un ReentrantReadWriteLock, qui permet d'éviter les accès concurrents sur la HashMap topics : De multiples accès en lectures sont autorisés tant qu'il n'y a pas d'accès en écriture. Si un accès en écriture est demandé, aucun autre accès ne sera autorisé tant que celui-ci n'aura pas terminé.

Globalement, on reçoit donc tous les appels de méthodes sur Broker dans son pool de threads *réception*, puis quand l'on transmet des messages (voir méthode publish() de Broker), on transmet le message en se servant du pool de threads *réception* :

```
this.runTask(ENVOIE_EXECUTOR_URI, owner -> {
	try {
		brop.acceptMessage(m);
	} catch (Exception e) {
		e.printStackTrace();
	}
});
```

Tests d'intégration (scénarios d'exécution des composants testant toutes les méthodes des interfaces de composants) :

Le package *src/deployment* contient un fichier CVM à lancer pour réaliser l'exécution des tests d'intégration.
Tout d'abord le subscriber va tester les méthodes contenues dans ManagementImplementationI, la console décrit ces tests.

Ensuite vont être testées les autres méthodes. Pour rappel le subscriber à souscrit au topics *topic1* avec un filtre qui renvoie toujours false, *topic2* et *topic3* sans filtre.
Il souscrit également à *topic4* mais retire sa souscription directement.
Subscriber ne doit donc recevoir que les messages arrivant sur *topic2* et *topic3*.

Publisher va tester les différentes méthodes publish dans sa méthode execute().
Tout d'abord, il envoie 20 messages sur *topic2*, nommés de *msg_i_0* à *msg_i_19*.
Il va ensuite attendre 2 secondes.

Subscriber à bien reçu les messages, et les affiche. Broker affiche également des informations pour vérifier que plusieurs threads sont bien utilisés, que ce soit pour la réception ou l'envoi (les numéro des threads sont affichés dans les deux cas).

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.