SORBONNE UNIVERSITÉ

Compte-rendu de projet CPS

Système minimal de publication / souscription en BCM4JAVA

Par le groupe ASSIREM

Le 18/03/202O

IMPLANTATION DES COMPOSANTS, INTERFACES, PORTS ET CONNECTEURS

IMPLANTATION DES COMPOSANTS: Dans le package /CPSDistrib/src/components

Nous avons implémenté les classes: Broker.java, Subscriber.java, Publisher.java.

IMPLANTATION DES INTERFACES: Dans le package /CPSDistrib/src/interfaces

Nous avons implémenté: Les interfaces de composants : ManagementCI, PublicationCI, ReceptionCI. Les interfaces java du système : SubscriptionImplementationI, ManagementImplementationI, PublicationsImplementationI, ReceptionImplementationI, MesaageI. Et des interfaces fonctionnelles et utiles : PropertiesI.java , MessageFilterI.java et TimeStampI.java.

IMPLANTATION DES PORTS: Dans le package /CPSDistrib/src/ports

Nous avons implémenté les classes représentant: 1) Les ports du publieur: PublicationCIPublisherOutboundPort et ManagementCIPublisherOutboundPort. 2) Les ports du souscripteur: ManagementCISubscriberOutboundPort et ReceptionCISubscriberInboundPort. 3) Les ports du courtier: ManagementCIBrokerInboundPort, PublicationCIBrokerInboundPort et ReceptionCIBrokerOutboundPort .

IMPLANTATION DES CONNECTEURS: Dans le package /CPSDistrib/src/components

Nous avons implémenté les classes suivantes: ManagementCIConnector.java, ReceptionCIConnector.java, PublicationCIConnector.java .

IMPLANTATION DES MESSAGES

Dans le package “utile” nous avons implanté des classes utiles aux projet dont la classe Message.java qui représente un message. L’objet Message contient:

1) une uri. 2) un objet TimeStamp qui contient un attribut temps système Unix auquel le message a été publié et un attribut (adresseIP/nom d'hôte) du publieur. 3) un objet Properties de la classe Properties.java contenue dans le package “utiles”, qui contient 9 HashMap de types différents qui permettent l’ajout de propriétés au message. 4) un contenu qui est un objet java sérialisable.

Le filtre a été représenté dans la classe MessageFilter.java, un message passe un filtre quand les propriétés du filtre sont contenus dans les propriétés du message, cette condition est vérifiée par des tests écrits et commentés dans la méthode filter de la classe MessageFilter.

Dans le package “tests” nous avons une classe MessageTest qui contient des tests unitaires sur les messages, les propriétés et les filtres.

IMPLANTATION DES GREFFONS CLIENTS POUR LES PUBLIEURS ET SOUSCRIPTEURS

Dans le package /CPSDistrib/src/versionPlugin nous avons: 1) un package “components” qui contient les classes composants. 2) un package “cvm” pour la classe cvm. 3) un package “plugins” pour les classes plugins: publisherPlugin et subscriberPlugin. 4) un package “ports” qui contient la classe du port entrant ReceptionCI du souscripteur: ReceptionCISubscriberInboundPortforPlugin.

STRUCTURES DE DONNÉES ET GESTION INTERNE DU COURTIER (ET GESTION DE LA CONCURRENCE)

Nous avons utilisés 3 structures de données de type HashMap dans le courtier: 1) une HashMap “subscribersForEachTopic” qui associe un topic à une Map de couples (uri du souscripteur, filtre). 2) une HashMap “portForEachSubscriber” associe l’uri de chaque souscripteur au bon port d’envoi . 3) une HashMap “msgToSubscribers” associe le message à une Map de couples (uri du souscripteur, filtre).

Utilistion: À la souscription, le courtier garde dans la map “subscribersForEachTopic” l’uri et le filtre du souscripteur et le topic. Et lors de la réception d’un message avec topic par le courtier, ce dernier récupère les abonnés à ce topic par le biais de “subscribersForEachTopic” et associe ces abonnés au message dans la Map “msgToSubscribers”. Lors de l’envoi du message, le courtier récupère les abonnés au topic du message par le biais de “msgToSubscribers” et applique le filtre sur le message pour chaque abonnés, si le message passe le filtre, le courtier récupère le port de l’abonné via la Map “portForEachSubscriber” et envoie le message.

La gestion de la concurrence est faite par: la création d’un ReentrantReadWriteLock sur la Map “subscribersForEachTopic” qui est la map la plus utilisée, ce qui a permis d’organiser les accès en lecture et écriture sur cette map de tel sorte à autoriser plusieurs threads lecteurs en simultané et un seul thread à la fois lors d’une écriture, les threads s’occupant des méthodes “ManagementCI” ont été séparés dans 2 pools différents (pool lecteurs, pool écrivains), les méthodes de publication sont gérées par des threads à part qui sont dans le pool “publication”. Quant aux 2 autres Map du courtier elle sont protégées par des “synchronized” java.

TESTS D’INTÉGRATION

Nous avons réalisé les tests d’intégrations sous forme de scénarios écrits dans des méthodes java. 1) Nous avons implanté dans le publieur trois méthodes/ scénarios qui s'exécutent dans la méthode start et qui font des affichages (log) sur console pour vérifier le bon fonctionnement des fonctionnalitées du système : senario_One() : teste la création de topics, publication de messages et destruction de message. senario_Two() : teste la création de topics, la publication de plusieurs messages en même temps, le filtre, et la publication d’un message avec plusieurs topics , senario_Three() : teste la publication répétée de messages, et la publication de plusieurs messages avec plusieurs topics en simultané.

2) Nous avons également dans le souscripteur trois méthodes/ scénarios: senario_One() : pour la souscription à un topic, la création d’un topic, de plusieurs topics et la destruction de topic, senario_Two() : souscription à plusieurs topics, senario_Three(): souscrire à un topic sans filtre et souscrir à un topic avec filtre.

CRÉATION DYNAMIQUE DES COMPOSANTS

Dans le package /CPSDistrib/src/dynamique.