Exercice 02

Connecteur push synchrone

Execice sur la conception d'un connecteur push synchrone multi-client et multi-serveur

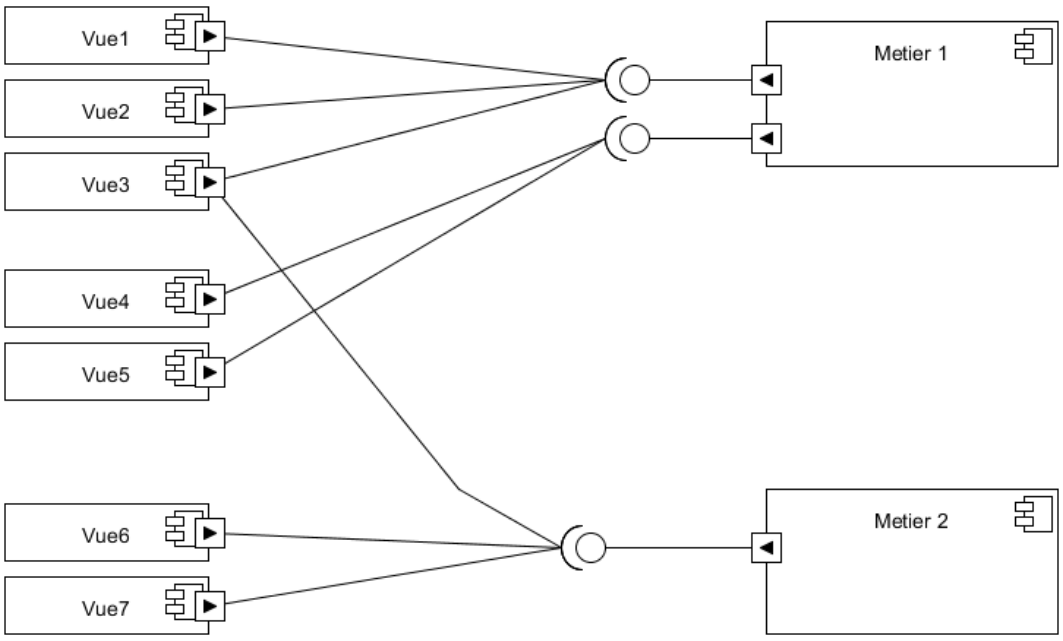
1. Enoncé

Nous voulons réaliser un connecteur de communication, réutilisable, entre un serveur et des clients répondant aux besoins suivants :

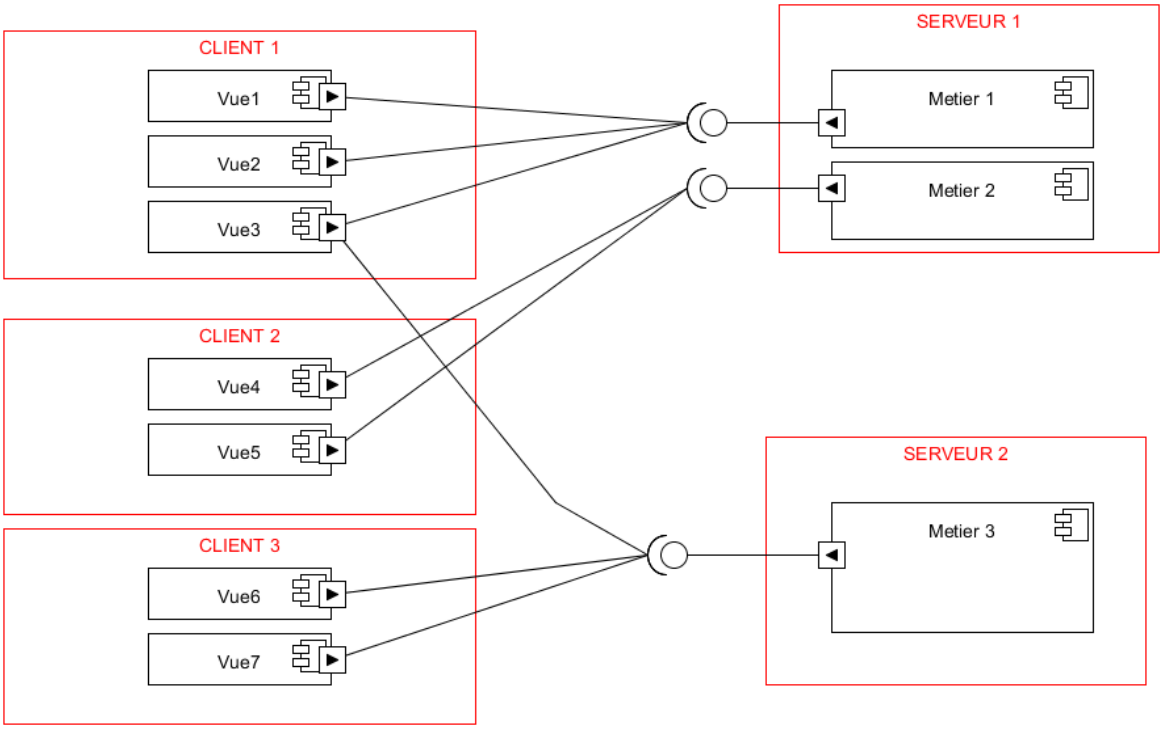
- un serveur peut créer plusieurs "canaux de communications" et pousser dans ces canaux un objet Java.
- Ces canaux de communication sont identifiés par une adresse IP (celui de la machine du serveur), un port et un nom
- le client peut créer différents consommateurs qui s'abonnent à un ou plusieurs canaux de communication (adresse IP, port et nom)
- Tout objet poussé dans un canal sera envoyé à tous les consommateurs qui sont abonnés à ce canal (mode push synchrone).

Il est à noter qu'il peut y avoir plusieurs serveurs répartis sur le réseau et plusieurs clients répartis sur le réseau, et que des clients peuvent être en communication avec 1 ou plusieurs serveurs différents.

On obtient l'architecture suivante :



Exemple de déploiement :



Ecrire le diagramme de communication de se connecteur.

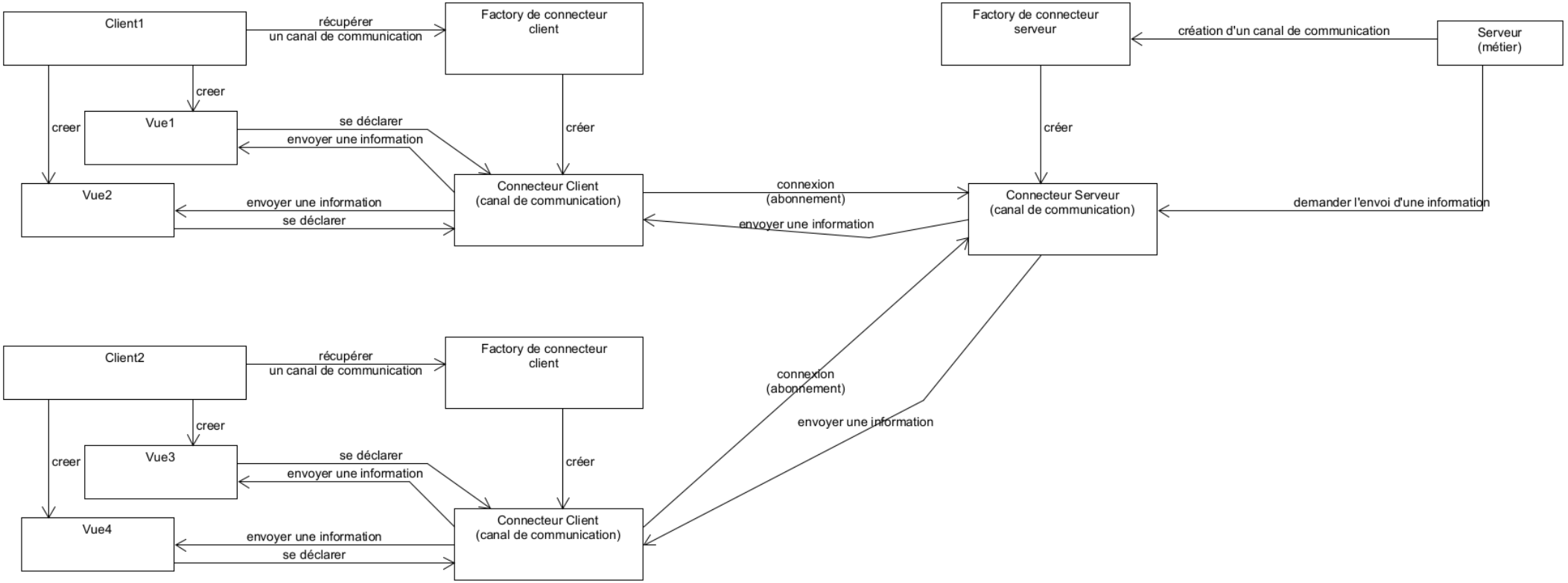
Ecrire le diagramme UML de se connecteur sachant qu'il est composé de deux parties:

- une partie "serveur" utilisée par le serveur (même JVM) qui pousse les informations dans un ou plusieurs canaux de communication.
- une partie "cliente" utilisée par le client (même JVM) qui contient plusieurs vues qui reçoivent les informations.

Vous utiliser la technologie RMI pour faire la communication.

2. Correction

2.1. Diagramme de communication


Client1
récupérer
un canal de communication
Factory de connecteur
client
Factory de connecteur
serveur
création d'un canal de communication
Serveur
(métier)
creer
creer
Vue1
se déclarer
envoyer une information
créer
créer
Vue2
envoyer une information
se déclarer
Connecteur Client
(canal de communication)
connexion
(abonnement)
envoyer une information
Connecteur Serveur
(canal de communication)
demander l'envoi d'une information
Client2
récupérer
un canal de communication
Factory de connecteur
client
connexion
(abonnement)
envoyer une information
creer
creer
Vue3
se déclarer
envoyer une information
créer
Vue4
envoyer une information
se déclarer
Connecteur Client
(canal de communication)

2.2. Conception préliminaire

Etant donné que l'on veut que toutes les vues connectées à un canal de communication crée par le serveur soient tous notifiés par l'envoi d'une information dans un canal de communication, le DP Observer/Observable s'impose : un canal = une classe Observable. Et comme les vues sont distantes, le DP Observer/Observable Distant s'impose.

Etant donné qu'un serveur doit pouvoir créer autant de canaux, on va utiliser un factory. On aura donc, côté serveur, un Factory d'Observabler Distant.

Côté client, chaque vue doit créer autant d'Observer Distant qui s'abonnent à son Observable Distant.

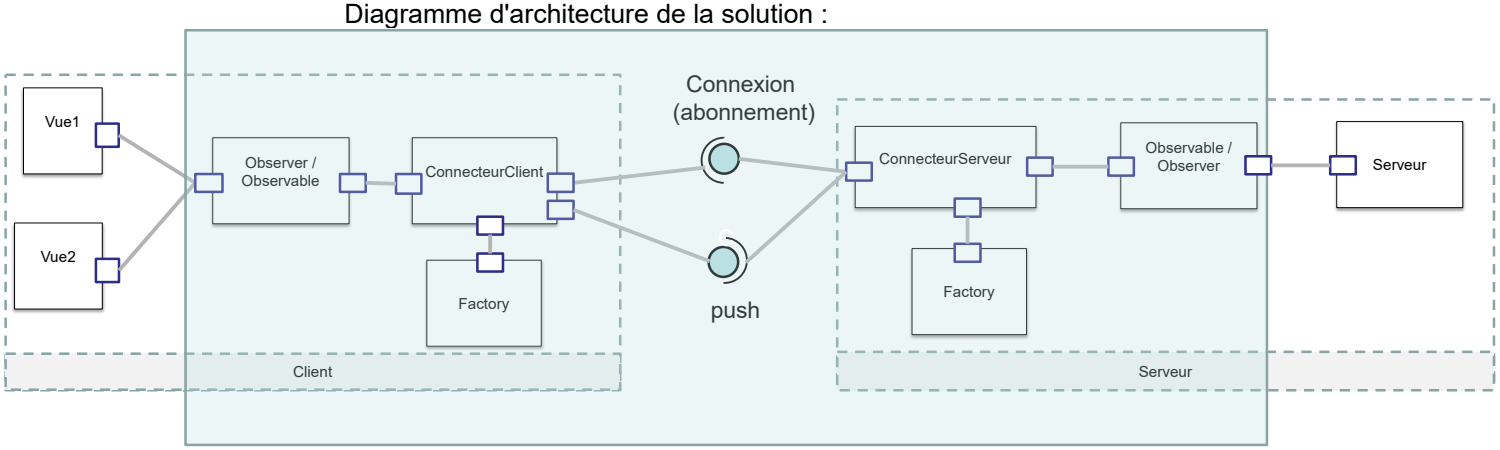
On va donc créer un Factory d'Observer Distant avec deux solutions conceptuelles

- On crée autant d'Observers Distants qu'il existe de Vues
- On crée autant d'Observer Distant qu'il existe de canal de communication. Dans ce cas l'Observer Distant est un Observable (local) auxquels s'abonnent les vues.

On choisit la deuxième solution qui offre un meilleure optimisation et évolutivité, et aussi une certaine symétrie.

On aura donc, pour un canal de communication, côté serveur et côté client, un DP Observer/Observable.

Diagramme d'architecture de la solution :



Connecteur

Le connecteur que nous réalisons est constitué de deux parties :

- une partie serveur qui prend en charge la connexion avec les clients (qui se sont abonnés) afin de leurs pousser l'information
- une partie cliente qui reçoit l'information et envoie à toutes les vues du client cette information.

La partie serveur du connecteur contient un DP Observable/Observer pour gérer tous les clients.

La partie cliente du connecteur contient un DP Observable/Observer pour gérer toutes les vues.

Les vues ne connaissent pas l'existence du serveur mais que le connecteur client (local).

Le schéma nous montre la séquence nécessaire pour réaliser la connexion entre le Factory et les Vues :

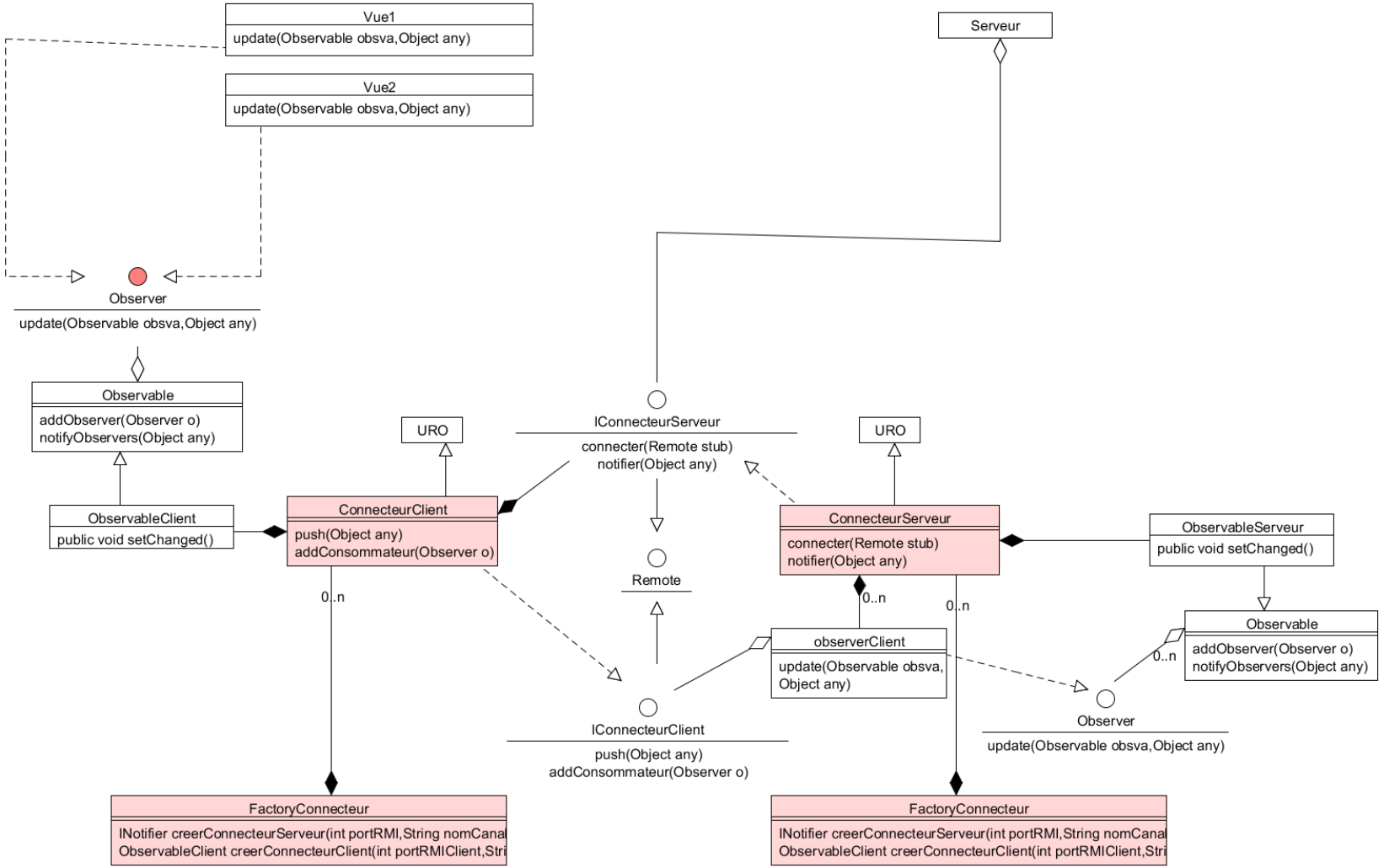
- 1) le Serveur crée un ConnecteurServeur qui sera utilisé par le serveur pour pousser son information
- 2) Le Client crée un ConnecteurClient qui crée son Observable qui s'abonne à un connecteur serveur
- 3) Chaque Vue s'abonne au ConnecteurClient comme Observer de son Observable

Ensuite, la communication se fait ainsi :

- 4) Le Factory pousse son information au ConnecteurServeur qui pousse son information à son Observable/Observer qui pousse l'information à tous les ConnecteurClient.

  Chaque ConnecteurClient pousse à son tour l'information à son Observer/observable qui pousse l'information à toutes les vues.

2.3. Diagramme de classe de la solution


Vue1
update(Observable obsva,Object any)
Vue2
update(Observable obsva,Object any)
Serveur
Observer
update(Observable obsva,Object any)
Observable
addObserver(Observer o)
notifyObservers(Object any)
URO
IConnecteurServeur
connecter(Remote stub)
notifier(Object any)
URO
ObservableClient
public void setChanged()
ConnecteurClient
push(Object any)
addConsommateur(Observer o)
ConnecteurServeur
connecter(Remote stub)
notifier(Object any)
ObservableServeur
public void setChanged()
Remote
0..n
0..n
0..n
observerClient
update(Observable obsva,
Object any)
Observable
addObserver(Observer o)
notifyObservers(Object any)
0..n
IConnecteurClient
push(Object any)
addConsommateur(Observer o)
Observer
update(Observable obsva,Object any)
FactoryConnecteur
INotifier creerConnecteurServeur(int portRMI,String nomCanal
ObservableClient creerConnecteurClient(int portRMIClient,Stri
FactoryConnecteur
INotifier creerConnecteurServeur(int portRMI,String nomCanal
ObservableClient creerConnecteurClient(int portRMIClient,Stri

Dans la partie Client :

Les Vues implémentent l'interface Observer. Elles s'abonnent à l'Observable local créé par le ConnecteurClient.

Le ConnecteurClient sur l'appel de la méthode distante push utilise l'observable pour notifier l'information qui lui a été passé en paramètre de la méthode push.

La classe FactoryConnecteur est utilisée pour créer le ConnecteurClient.

Le ConnecteurClient utilise l'interface distante IConnecteurServeur pour se connecter au ConnecteurServeur distant.

Dans la partie Serveur :

Le FactoryConnecteur est utilisé par le serveur pour créer un ConnecteurServeur.

Le ConnecteurServeur gère un Observable pour chaque canal de communication.

Le ConnecteurServeur implémente l'interface distante IConnecteurServeur permettant au ConnecteurClient de se connecter (abonnement) via la méthode connecter qui prend en paramètre l'interface distante IConnecteurClient du ConnecteurClient (son stub). Il crée un observer ObserverClient pour chaque ClientConnecteur.

Le serveur utilise la méthode notifier de ConnecteurServeur pour envoyer une information dans un canal de communication.