

École nationale supérieure d'électrotechnique, d'électronique, d'informatique,  
d'hydraulique et des télécommunications  
**Filière Informatique et Mathématiques appliquées**

# Rapport du projet de systèmes concurrents / intergiciels

---

Olivier Lienhard  
Tom Lucas  
Thibault Hilaire

Toulouse, le 13 Janvier 2015

# Sommaire

<b>1</b>	<b>Choix de la spécification libérale</b>	<b>3</b>
<b>2</b>	<b>Version en mémoire partagée</b>	<b>4</b>
2.1	Choix d'implémentation . . . . .	4
2.2	Tests . . . . .	5
<b>3</b>	<b>Version client / mono-serveur</b>	<b>6</b>
3.1	Implémentation . . . . .	6
3.1.1	Serveur . . . . .	6
3.1.2	Client . . . . .	7
3.1.3	Le cas des EventRegister . . . . .	7
3.2	Tests . . . . .	8

## Partie 1

# Choix de la spécification libérale

Explications des différents choix de la spécification libérale :

- quand plusieurs tuples correspondent, take retourne le premier à avoir été écrit dans la mémoire (FIFO)
- quand plusieurs take sont en attente et qu'un dépôt peut en débloquent plusieurs, on débloquent le premier take à avoir demandé (FIFO)
- quand des read et un take sont en attente, et qu'un dépôt peut les débloquent, on les débloquent dans l'ordre de demande (FIFO)
- quand il y a un take et un callback enregistré pour le même motif, le take est prioritaire

## Partie 2

# Version en mémoire partagée

### 2.1 Choix d'implémentation

Voici les différentes structures choisies pour respecter nos choix de spécification:

- `Collection<Tuple> tuples` : pour sauvegarder nos tuples qui ont été écrits dans la mémoire, et les enlever lors d'un `take`
- `Map<Tuple, LinkedList<Integer> MatchEnAttente` : cette map va permettre de stocker tous les `take/read` bloquants
- `int id` : un entier qui nous permettra de simuler notre FIFO, de savoir dans quels ordres les `take/read` ont été écrits dans notre map
- `ArrayList<Condition> classe` : un ensemble de conditions liées à un `Lock`, dont on associera chaque condition à un `take/read` bloquants
- `Condition writeCondition` : une condition qui va nous permettre de faire une priorité au signalé lors du réveil d'un `take/read`
- `Boolean takeEffectue` : un booléen permettant de vérifier, lors d'un `write`, si un `take` a été effectué

A partir de ceci, on peut expliquer l'algorithme de la fonction `write` :

Tout d'abord on ajoute le tuple dans la collection de tuples. Ensuite on récupère les templates dont le tuple ajouté correspond (i.e. `tuple.matches(template)==true`), ce qu'on appelle les `templatesCorrespondants`. Puis tant que cette collection de `templatesCorrespondants` n'est pas vide ET qu'un `take` (sur ce tuple ajouté) n'a pas été effectué (vérifiable sur notre booléen `takeEffectue`), on réveille le premier template correspondant (celui dont l'indice `id` est le plus bas). Si un `take` est effectué, le booléen `takeEffectue` devient vrai, on sort de la boucle et on ne réveille pas les callback et la fonction `write` est finie. Sinon, si aucun `take`

n'est effectué et si on a parcouru toute la liste des templatesCorrespondants, on notifie les callback qui sont des observateurs. Un tel algorithme garantit alors le respect de la spécification ci-dessus, qui est ensuite validée par les tests décrits ci-dessous.

## 2.2 Tests

Pour les tests, outre les tests donnés, des tests unitaires sont effectués pour chaque fonction pour chaque type de paramètre différent (notamment pour les callback). Ainsi une liste non exhaustive des tests unitaires principaux est :

- BasicTestAsyncCallbackReadFuture.java
- BasicTestAsyncCallbackReadImmediate.java
- BasicTestAsyncCallbackTakeFuture.java
- BasicTestAsyncCallbackTakeImmediate.java
- BasicTestCallbackReadFuture.java
- BasicTestCallbackReadImmediate.java
- BasicTestCallbackTakeFuture.java
- BasicTestCallbackTakeImmediate.java
- BasicTestRead.java

D'autres tests unitaires pour les fonctions tryRead, tryTake, TakeAll, ReadAll sont aussi définis. Un test vérifiant qu'un callback peut se réengistrer est aussi donné. Enfin, l'application whiteboard fonctionne exactement comme elle nous a été présentée lors de l'introduction au projet.

Puis, pour vérifier que la spécification est bien respectée, nous avons effectué des tests pour vérifier chacun des quatre points, nommés respectivement dans l'ordre :

- BasicTestTakeSpec1.java
- BasicTestTakeSpec2.java
- BasicTestTakeReadSpec3.java
- BasicTestTakeCallbackSpec4.java

Ces tests-là pour les principaux, ajoutés à d'autres tests, donnant tous des résultats cohérents, nous permettent de valider notre implémentation. Il est à noter que certains tests finissent sur des commandes bloquantes et il faut donc arrêter l'exécution manuellement, cela est bien sûr fait exprès de manière à montrer dans certains cas le bon fonctionnement de nos fonctions.

## Partie 3

# Version client / mono-serveur

### 3.1 Implémentation

#### 3.1.1 Serveur

L'interface `LindaServer` hérite de `java.rmi.remote`, elle reprend les même fonctions que `LindaCentralized`, seule la signature de la méthode `eventRegister` est différente (voir section 3.1.3).

La classe `LindaServerImpl` implémente cette interface et hérite de `UnicastRemoteObject`. Elle possède un attribut de type `CentralizedLinda` qui est réutilisé sans avoir été modifié. Une fonction `main()` permet de lancer le serveur et de l'enregistrer ..... pour pouvoir être trouvé par les clients.

```
private CentralizedLinda linda;

public static void main(String[] args) {
    System.out.println("Demarrage du serveur");
    String URL = "://localhost:4000/LindaServer";
    int port = 4000;
    try{
        LindaServer serveur = new LindaServerImpl();
        LocateRegistry.createRegistry(port);
        Naming.rebind(URL, serveur);
    }
    catch (Exception e){
        System.out.println("Une erreur s'est produite");
        e.printStackTrace();
    }
}
```

A par pour les eventRegister (voir section 3.1.3), les autres fonctions du serveur consistent simplement en un appel aux fonctions équivalente du la version centralisé Linda du serveur:

```
@Override
public Tuple tryTake(Tuple template) throws RemoteException {
    return linda.tryTake(template);
}
```

### 3.1.2 Client

La classe LindaClient implémente l'interface Linda. Son constructeur prend en paramètre l'URL du serveur

```
serveur = (LindaServer) Naming.lookup(URI);
```

Les fonctions de cette classe (à par EventRegister) consiste ainsi en l'appel des fonction correspondante du serveur.

### 3.1.3 Le cas des EventRegister

La principale difficulté de la version mono-serveur de Linda est l'implémentation de la méthode eventRegister car les callbacks pris en paramètre par la fonction eventRegister de la classe CentralizedLinda ne fonctionnent pas "à distance". La solution à été de créer un callback spécial pour pouvoir faire cela, l'interface RemoteCallback hérite de java.rmi.remote et est implémentater par la classe RemoteCallbackImpl : son constructeur prend en parémètre un callback classique et sa méthode Call appelle la méthode callback de ce dernier.

Le méthode EventRegister du serveur prend en paramètre un remoteCallback et crée un callback classique dont la fonction Call appelle celle du remoteCallback qu'elle passe en paramètre à la méthode EventRegister de CentralizedLinda :

```
@Override
public void eventRegister(eventMode mode, eventTiming timing,
    Tuple template, final RemoteCallback callback)
    throws RemoteException {

    final class newCallback implements Callback {
        public void call(Tuple t) {
            try {
                callback.call(t);
            } catch (RemoteException e) {
                e.printStackTrace();
            }
        }
    }

    linda.eventRegister(mode, timing, template, new newCallback() );
}
```

La méthode EventRegister du client (qui implémente l'interface Linda) prend en paramètre un callback classique et crée un remoteCallback à partir de celui ci pour pouvoir appelé la méthode EventRegister du serveur:

```
@Override
public void eventRegister(eventMode mode, eventTiming timing,
                          Tuple template, Callback callback) {
    try {
        RemoteCallback cb = new RemoteCallbackImpl(callback);
        serveur.eventRegister(mode, timing, template, cb);
    } catch (RemoteException e) {
        System.out.println("Erreur lors du eventRegister");
        e.printStackTrace();
    }
}
```

## 3.2 Tests

Tout les test effectués sur la version centralisé ont été effectués également sur la version mono-serveur, en donnant les même résultats.

De plus cette version à été testé avec des "whiteboard" et ils fonctionnent parfaitement.