

Systèmes concurrents & intergiciels

Projet

Philippe Quéinnec

ENSEEIH
Département Sciences du Numérique

19 février 2020

Plan

- 1 Linda
- 2 Réalisations
- 3 Modalités

Linda (TSpaces)

- Espace partagé contenant des tuples
- Un tuple = un n-uplet de valeurs
- Opérations de dépôt, retrait, consultation
- Retrait/consultation d'après un motif
- Retrait/consultation bloquantes
- Abonnement à un événement de dépôt

Tuple

Tuple de valeurs

N-uplet ordonné (= liste) de valeurs de type arbitraire (y compris tuple) :

```
[ 3 4 ], [ 10 'A' true ], [ "coucou" [ 10000 false ] 23 ]
```

Tuple motif

N-uplet ordonné de valeurs ou de **types** (classes en java) :

```
[ ?Integer ?Boolean ], [ ?Integer 'A' ?Boolean ],  
[ ?String ?Tuple 23 ]
```

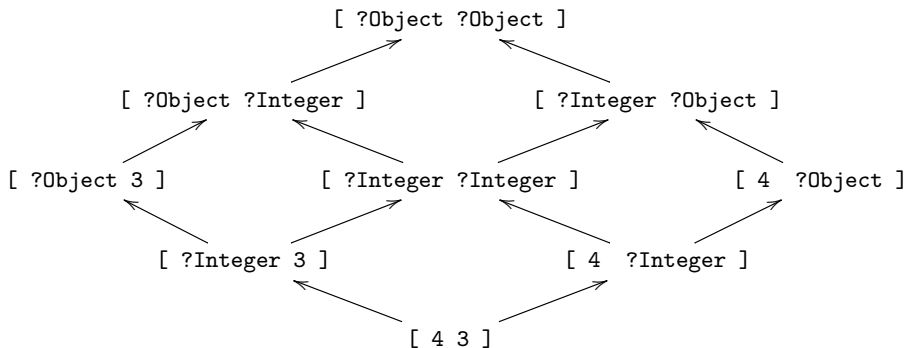
Correspondance

Un tuple **correspond** (matches) à un tuple s'il est inclus dedans :

```
[ 3 4 ] matches [ ?Integer ?Integer ]  
[ 3 4 ] matches [ ?Integer 4 ] or [ 3 ?Integer ]  
[ 3 ?Integer ] matches [ ?Integer ?Integer ]
```

Tuples

- ?Object est le type universel
- ?Tuple correspond à tout tuple
- On peut comparer des motifs :



Manipulation des tuples

La classe Tuple étend List<Serializable>

Plus :

- constructeur `Tuple(...)` :
 `new Tuple(4, "foo", Integer.class)`
 \Rightarrow [4 "foo" ?Integer]
- `t.toString()` pour écrire un tuple
- `Tuple.valueOf(String)` pour construire un tuple à partir d'une chaîne
- `t.matches(motif)`
 Renvoie vrai si t est inclus dans le motif
- `motif.contains(t)`
 $=$ `t.matches(motif)`

Les primitives bloquantes du modèle Linda

Objectif

Un espace partagé contenant des tuples.

Opérations bloquantes :

- **write**(tuple) : dépose le tuple dans l'espace partagé ;
- **take**(motif) : extrait de l'espace partagé un tuple correspondant au motif précisé en paramètre ;
- **read**(motif) : recherche (sans l'extraire) dans l'espace partagé un tuple correspondant au motif fourni en paramètre.

Exemple

```
// Attend un tuple
// [ ?Integer ?String ]
new Thread() {
    public void run() {
        Tuple m, r;
        m = new Tuple(Integer.class,
                       String.class);
        r = linda.take(m);
        System.out.println(r);
    }
}.start();
```

```
// Dépose [ 4 5 ]
// et [ 4 "foo" ]
new Thread() {
    public void run() {
        Tuple v1, v2;
        v1 = new Tuple(4, 5);
        linda.write(v1);

        v2 = new Tuple(4, "foo");
        linda.write(v2);
    }
}.start();
```


Les primitives non bloquantes du modèle Linda

- `tryTake(motif)` : version non bloquante de `take` ;
- `tryRead(motif)` : version non bloquante de `read` ;
- `takeAll(motif)` : renvoie, en extrayant, *tous* les tuples correspondant au motif (collection vide si aucun ne correspond) ;
- `readAll(motif)` : renvoie, sans extraire, tous les tuples correspondant au motif (collection vide si aucun).

La primitive d'abonnement du modèle Linda

```
public interface Callback {  
    /** Callback when a tuple appears. */  
    void call(Tuple t);  
}
```

eventRegister(mode, timing, motif, callback)

- S'abonner à l'existence/l'apparition d'un tuple correspondant au motif.
- `callback.call()` sera invoqué avec le tuple identifié.
- Le callback n'est déclenché qu'une fois, puis oublié.

Types d'abonnement

Mode

- `eventMode.READ` : le tuple est laissé dans l'espace
- `eventMode.TAKE` : le tuple est retiré de l'espace

Timing

- `eventTiming.IMMEDIATE` : l'état courant est considéré : si un tuple correspondant existe déjà, le callback est déclenché.
- `eventTiming.FUTURE` : l'état courant n'est pas considéré : seuls les futurs write seront pris en compte.

Exemple d'abonnement

```
class MyCallback implements Callback {  
    public void call(Tuple t) {  
        System.out.println("Got " + t);  
    }  
}
```

...

```
Tuple motif = new Tuple(Integer.class, String.class);  
Callback cb = new MyCallback();  
linda.eventRegister(eventMode.READ, eventTiming.FUTURE,  
                    motif, cb);
```

Exemple d'abonnement permanent

```
Tuple motif = new Tuple(Integer.class, String.class);  
...  
  
class MyCallback implements Callback {  
    public void call(Tuple t) {  
        System.out.println("Got " + t);  
        linda.eventRegister(eventMode.TAKE, eventTiming.IMMEDIATE,  
                             motif, this);  
    }  
}  
  
...  
Callback cb = new MyCallback();  
linda.eventRegister(eventMode.TAKE, eventTiming.IMMEDIATE,  
                    motif, cb);
```

Callback asynchrone

Attention : un callback est invoqué lors d'un write, dans le contexte du thread déposateur \Rightarrow il ne doit jamais se bloquer !

Callback asynchrone : s'exécute dans son propre thread.

```
...  
Tuple motif;  
motif = new Tuple(Integer.class, String.class);  
Callback cb = new AsynchronousCallback(new MyCallback());  
linda.eventRegister(..., ..., motif, cb);
```

Spécification libérale

- Choix arbitraire quand plusieurs tuples correspondent à un motif d'un take
- Choix arbitraire quand plusieurs take sont en attente et qu'un dépôt pourrait en débloquent plusieurs
- Sélection arbitraire quand des read et un take sont en attente et qu'un dépôt peut les débloquent :
outre take, tous les read sont débloqués, ou aucun, ou seulement certains
- Choix arbitraire entre déblocage d'un take et déclenchement d'un callback (en mode TAKE) concerné par un même tuple

« Arbitraire » ne signifie pas aléatoire, mais non spécifié : vous pouvez choisir une stratégie bien identifiée.

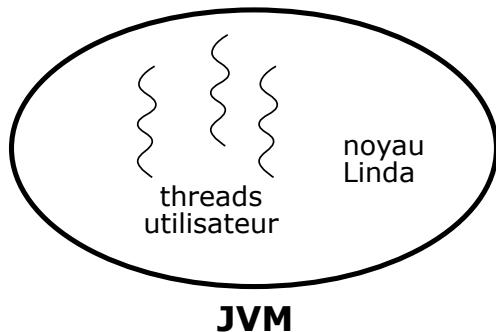
Plan

- 1 Linda
- 2 **Réalisations**
- 3 Modalités

Réalisations

- ❶ Version en mémoire partagée
- ❷ Version client / mono-serveur
- ❸ Sauvegarde et tolérance au fautes

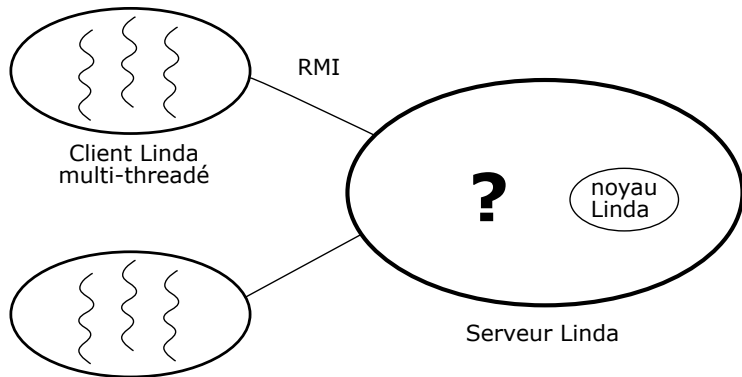
Version en mémoire partagée



Plusieurs activités (threads), dans la même machine virtuelle Java.

- ❶ Écrire write, read, take
- ❷ Ajouter eventRegister, refactoring de read et take
- ❸ Compléter les opérations non bloquantes

Version client / mono-serveur

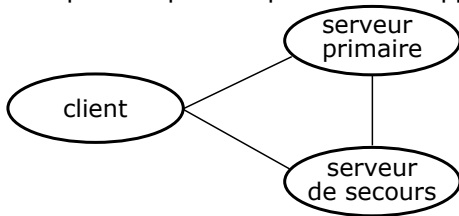


Un serveur stockant l'espace de tuples, un ou des clients distants accédant au serveur. Chaque client peut lui-même être concurrent (multi-threadé).

Sauvegarde / tolérance aux fautes

- 1 Pouvoir arrêter le serveur, et le redémarrer plus tard en récupérant l'état de l'espace des tuples
- 2 Serveur de secours : le serveur primaire envoie régulièrement l'état de l'espace au serveur de secours ; quand un client observe que le serveur primaire est inaccessible, il bascule sur le serveur de secours.

Objectif : transparence parfaite pour le code applicatif.



Plan

- 1 Linda
- 2 Réalisations
- 3 Modalités**

Modalités

- Projet exécuté *en binôme*
- Date limite de constitution des binômes : **rapidement**
Envoyés à queinnec@enseeiht.fr
- Séances de suivi et d'assistance, présence obligatoire
- Rendu le **7 avril 2020**
- Séance de test le 7 avril 2020

Modalités (2)

- Squelettes et code de base disponible dans
/home/queinnec/Projet-IR.tar
- **Respectez l'API** pour les deux premières parties : interfaces
Linda, Callback, Tuple, etc ;
constructeur classe `linda.shm.CentralizedLinda` ;
constructeur classe `linda.server.LindaClient`.
- La troisième partie peut (mais pas nécessairement) nécessiter
d'enrichir l'API \Rightarrow garder les deux versions.

Les tests fournis doivent compiler et s'exécuter correctement sans que vous y touchiez le moindre caractère.