Rapport $2^{ème}$ étape

Lorenzi Romain
Tanguy Titouan

Janvier 2017

1 Linda multiactivités en mémoire partagée

Le but de cette étape était d'améliorer la performance de la première version de Linda. Pour ce faire, l'idée est de fractionner l'espace des *Tuples*. En effet, sur la version précédente, l'espace étant en accès exclusif entre les différents processus qui l'utilisent, on est confronté à un goulot d'étranglement.
Techniquement, fractionner l'espace des *Tuples* est assez simple. L'idée est que la version multiactivités est composée de *maxThreads* Linda monoactivités numérotées. Les principales différences entre le code des deux version se situe au niveau des fonctions **write** et **eventRegister**.

1.1 La fonction write

Concernant la fonction **write** la différence entre les deux version est qu'il faut déterminer dans laquelle des fraction de l'espace des *Tuples* on veut écrire le *Tuple* courant. Afin de répartir la charge, nous avons donc décidé que le choix se ferait de façon aléatoire en suivant une loi uniforme. Ainsi, sur un grand nombre d'écriture, la taille de chaque fragment de l'espace des *Tuples* est égale.

1.2 La fonction eventRegister

La fonction **eventRegister** est le coeur de l'implantation de toutes les fonctions de lecture de l'espace des *Tuples*. C'est donc à l'aide de cette fonction qu'on implémente les fonctions **read, write, ...**
L'implantation d'**eventRegister** doit donc prendre en compte que le *Tuple* que l'on cherche peut se trouver dans n'importe lequel des fragments de l'espace. Ainsi, nous avons décidé que le premier fragment est, comme pour la fonction **write**, choisit au hasard. Ensuite, si le *Tuple* recherché n'y est pas présent, on passe au *suivant* dans la liste des fragments, en prenant en compte que lorsqu'on se trouve sur le dernier, on revient au premier au pas suivant.

2 Linda multiactivités en client / mono-serveur

Ici le but était de permettre l'accès à Linda à distance, en plaçant l'espace des *Tuples* sur un serveur, auquel on accède via un client qui implémente l'interface Linda.
Pour ce faire, on prend une version Linda qui est presque une copie conforme de la version Linda multiactivités, à l'exception que son interface est **Remote** et qu'on peut donc l'enregistrer sur un serveur de nom RMI. La quasi totalité des fonctions du client Linda sont alors uniquement un appel aux fonctions du serveur, duquel on récupère une copie RMI à l'instanciation du client. La seule difficulté technique de cette partie a été d'implémenter la fonction de **callback**.

2.1 Les callbacks

Afin que les **callbacks** puissent être appelé par le serveur, il faut fournir un moyen au serveur de le faire. Ainsi, nous avons du créer un *container à callbacks* qui elle est **Remote**, ce qui n'est pas le cas de l'interface donnée. Cette nouvelle classe prend lors de sa création un **Callback** *classique*, et a pour seul fonction *call* qui ne fait qu'exécuter la fonction homonyme du **Callback** *classique*. Ainsi, lors de l'appel à **eventRegister** sur le client, un tel *container* est instancié puis enregistré sur un serveur de nom **RMI**. L'adresse de ce *container* est envoyé en paramètre au serveur, avec les autres arguments classique de la fonction **eventRegister**, lequel peut donc récupérer ce *container*, et de son côté instancie un **Callback** *classique* qui a pour mission de *call* la fonction *call* du *container*, qui a son tour *call* la fonction *call* du **Callback** *classique*.