Aix-Marseille Université
Master Informatique
Algorithmique Distribuée
*2014*

# TP 5 : Rendez-vous d'agents mobiles

## 1 Simulation d'algorithme de rendez-vous d'agents mobiles

On va simuler dans ce TP l'algorithme de rendez-vous pour deux agents mobiles dans un anneau. Les nœuds de l'anneau sont anonymes mais les agents ont chacun un identifiant unique (entier). On suppose ici que les agents connaissent la taille de l'anneau.

**Algorithme naïf de rendez-vous :**
Un agent mobile ayant un identifiant $ID = x_i$ fait $x_i$ tours complets de l'anneau. Si durant l'un de ces tours, il rencontre un autre agent sur un nœud alors il s'arête. S'il ne rencontre aucun agent, il s'arête de se déplacer après avoir compléter

### Exercice

1. Écrire l'algorithme pour l'agent mobile comme un classe publique "MobileAgent.java". La classe `MobileAgent` devra contenir un constructeur et une méthode `execute`() qui implémente l'algorithme de rendez-vous. Le constructeur a pour signature `public MobileAgent(int id, int sizeOfRing, RingNode initialPosition)` et il doit créer un agent ayant l'identifiant `id` et commençant sur le nœud `initialPosition` dans un anneau de taille `sizeOfRing`. La fonction `execute`() fera se déplacer l'agent une seule fois.
2. On va utiliser la classe `RingNode` et le programme "RingMain.java" pour implémenter un réseau en anneau.
   Télécharger le fichier "RingRendezVous.zip" au lien suivant :
   `http://pageperso.lif.univ-mrs.fr/~arnaud.labourel/AD2014/RingRendezVous.zip`

   Ce fichier contient un programme qui construit un anneau anonyme d'un taille donnée (30 par défaut). Le programme après compilation se lance avec la ligne de commande suivante (il faut que le jar de jbotsim soit présent dans le répertoire courant) :

   $$\texttt{java} - \texttt{cp} . : \texttt{jbotsim.jar RingMain} < \texttt{size} - \texttt{of} - \texttt{ring} >$$

   Afin de gérer les agents mobiles, on doit modifier la classe RingNode et RingMain comme suivant :
   — Chaque RingNode doit contenir une liste des agents mobiles qui sont présent dans le nœud.
   — A chaque tick d'horloge (appel de `onClock()`), tous les agents présents sur le nœud doivent se déplacer (appel à la méthode `execute()`). On utilisera un compteur `int numberOfAgents` dans chaque nœud pour sauvegarder le nombre d'agents présent dans le nœud. On changera la couleur du nœud avec `setColor` en fonction du nombre d'agents présents (pas de couleur s'il n'y a pas d'agents, vert pour un agent et rouge pour deux agents).
   — Il faudra modifier la classe `RingMain` afin qu'elle crée un anneau d'une certaine taille avec deux agents ayant des identifiants différents tirés au hasard et placés dans des positions différentes (les positions des agents sont spécifiées par des distances à un nœud fixé). Le programme devra être exécuté grace à la commande suivante :

   $$\texttt{java} - \texttt{cp}. : \texttt{jbotsim.jar RingMain} < \texttt{size} - \texttt{of} - \texttt{ring} > < \texttt{Location1} > < \texttt{Location2} >$$

3. Le mouvement des agents sera implémenté par les deux méthodes suivante de la classe RingNode :

```
public  void  moveAgent(MobileAgent ma, Direction d)
public  void  addAgent(MobileAgent ma)
```

Avec Direction étant l'énumération suivante :

```
public enum Direction {RIGHT, LEFT}
```

Quand un agent souhaite se déplacer sur un autre nœud, il appelle la méthode `RingNode.moveAgent(ma, d)` sur le nœud courant. Le nœud met l'agent mobile dans un message et l'envoie au voisin à gauche (si d = LEFT) ou à droite grace à la méthode `send`(). Quand un message est reçu par un nœud, le nœud récupère le contenu du message (msg.content) et l'ajoute au nœud avec un appel `RingNode.addAgent`(). La méthode `RingNode.addAgent`() met à jour la liste des agents présent sur le nœud.

4. Exécuter votre programme pour des tailles d'anneau et des positions variées des agents et vérifier son fonctionnement.
5. Que doit-on modifier dans le programme afin de réaliser le rendez-vous d'un nombre quelconque $k > 2$ d'agents mobiles ?