TP Modèle de programmation « agents mobiles »

S. Leriche – SITA 3A

1) Objectif

Je vous propose de construire une application à base d'agents mobiles dont le rôle est de déployer et de superviser un composant de calcul (calcul de décimales de PI) sur la machine la moins occupée en termes de charge CPU de la salle de TP.

Pour cela, nous allons développer un agent dont le comportement initial consistera à rechercher sur le réseau local la machine disponible la moins chargée (en se déplaçant de machine en machine), puis de se déplacer sur la machine sélectionnée et changer de comportement pour superviser l'exécution du composant de calcul. Enfin, l'agent reviendra sur le site initial pour afficher le résultat (et les informations sur la machine sélectionnée).

2) Préliminaire : configuration ssh

(à faire une fois pour tous les TP)

```
[leriche@poste1 ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/permanents/leriche/.ssh/id_rsa):
Created directory '/permanents/leriche/.ssh'.
Enter passphrase (empty for no passphrase): NE RIEN TAPPER ICI SAUF ENTREE
Enter same passphrase again: IDEM
Your identification has been saved in /permanents/leriche/.ssh/id_rsa.
Your public key has been saved in /permanents/leriche/.ssh/id_rsa.pub.
...
[leriche@poste1 ~]$ cd ~/.ssh
[leriche@poste1 .ssh]$ cat id_rsa.pub > authorized_keys
[leriche@poste1 .ssh]$ ssh poste2
The authenticity of host 'poste2 (157.159.110.50)' can't be established.
RSA key fingerprint is 9d:1f:ce:13:df:d6:76:c2:d5:2c:c9:ea:0c:78:fc:ff.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'poste2,157.159.110.50' (RSA) to the list of known hosts.
Last login: Fri Jan  9 17:55:12 2009 from poste1.int-evry.fr
[leriche@poste2 ~]$ exit
logout

Connection to poste2 closed.
```

3) Création d'un projet JavAct

Créez un nouveau projet (« File » / « new » / « Project... »), puis sélectionnez un projet **JavAct**.

Attention, il faut impérativement cocher l'option « *Use project folder as root for source and class files* » dans la partie « Project Layout ».

4) Mise en place du projet

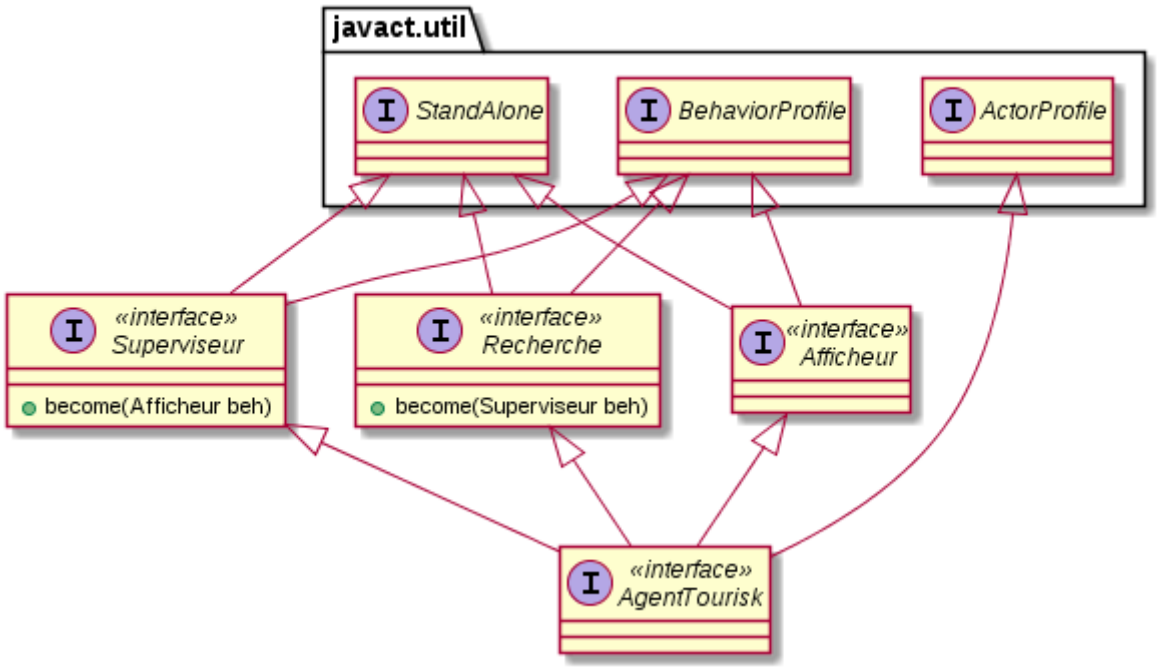
Importez dans votre projet les fichiers suivants disponibles sur ~leriche :

```
JavActProbe.java
PIComputer.java
```

Attention, ne pas créer de paquetage (il existe des options pour les prendre en compte mais on veut faire au plus simple ici).

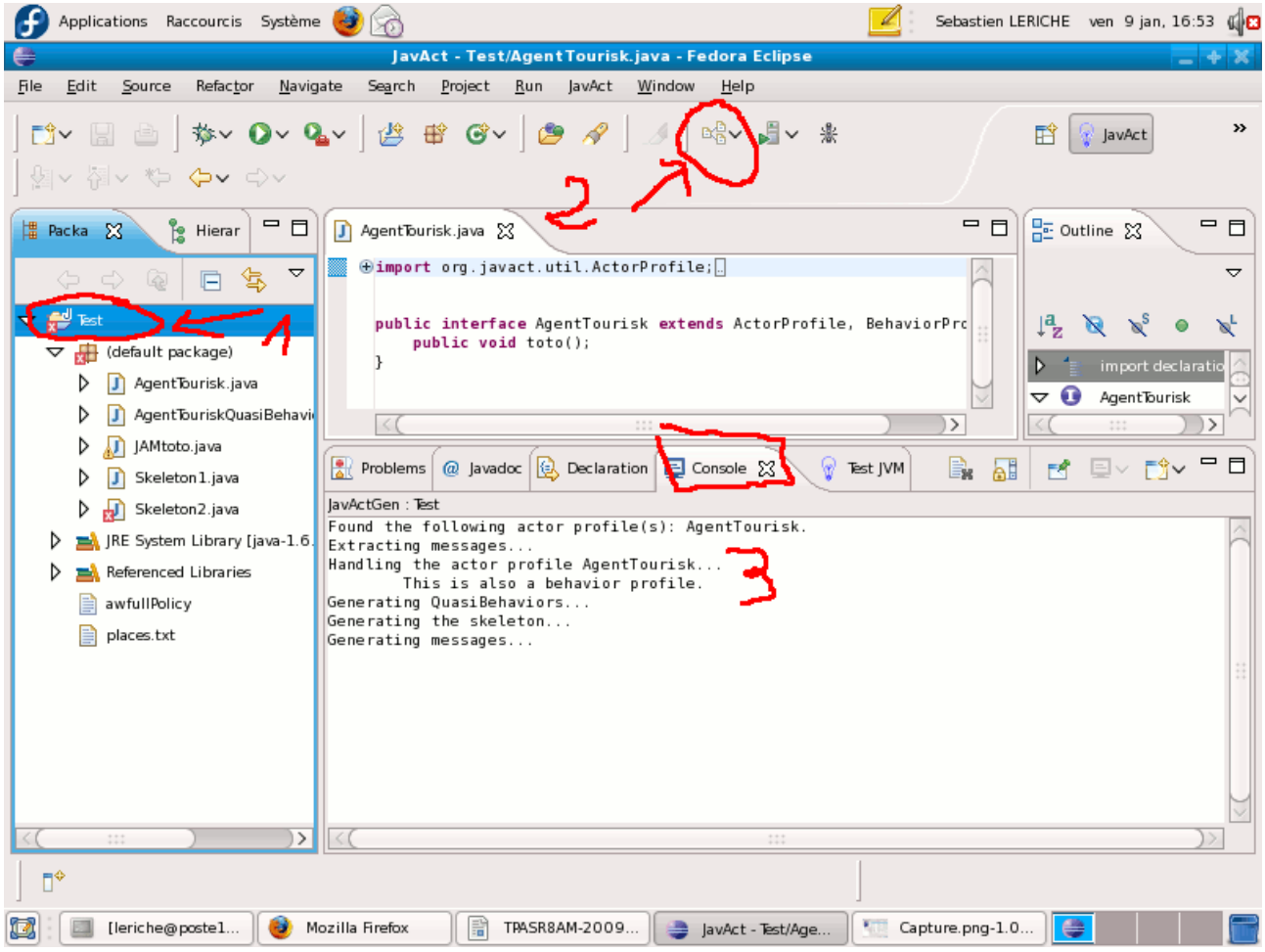
Écrivez les interfaces **Recherche**, **Superviseur**, **Afficheur** et **AgentTourisk** suivant la spécification fournie dans le diagramme UML ci-dessous. Note : pour exprimer que l'agent peut depuis un comportement X changer de comportement en Y, il faut spécifier dans l'interface X la méthode :

```
public void become(Y beh);
```



5) Génération automatique des codes intermédiaires

Dans la vue « Package Explorer » (dans Eclipse, fenêtre de gauche) sélectionnez votre projet (1). Vous pouvez ainsi accéder au bouton « **JavActGen** » de la barre d'outils (2) ou dans le menu JavAct de la barre de menu. La génération crée les classes XxxQuasiBehavior, JAMxxx ainsi qu'un squelette (SkeletonX) contenant le code métier à compléter. Vous pouvez observer dans la console (3) des messages relatifs aux différents traitements effectués lors de la génération (en cas d'erreur, me demander).



6) Implémentation du code métier

Ouvrez le fichier **Skeleton1.java** générée par l'outil, notez qu'il contient le squelette de toutes les classes de comportement que vous devez maintenant implémenter.

Pour que les comportements d'agents soient activés dès leur création, il faut que les classes associées implémentent l'interface **StandAlone** (XXXBeh implements StandAlone). Celle-ci fournit une méthode run(), dans laquelle vous pouvez coder vos actions. Vous utiliserez les classes et morceaux de codes décrit ci-dessous.

- La classe **JavActProbe** vous permet d'observer quels systèmes d'accueils JavAct sont disponibles dans la salle de TP. La méthode statique **probe(int port)** renvoie un tableau de chaînes, contenant les @IP des machines accessibles.
- Pour connaître le nom de la machine courante et son adresse IP vous pouvez vous inspirer du code suivant :

```
try {
      System.out.println(myPlace() + " "                              // JavAct API
                  + InetAddress.getLocalHost().getHostAddress());     // Java API
} catch (UnknownHostException e) { e.printStackTrace(); }
```

- Pour avoir des informations sur la charge machine, vous pouvez vous aider du code suivant :

```
OperatingSystemMXBean bean = ManagementFactory.getOperatingSystemMXBean();
System.out.println("System load average for the last minute : "
      + bean.getSystemLoadAverage());
```

- Dans la méthode main() de votre classe SkeletonX, vous pouvez créer sur la machine locale (et sur votre port) d'un agent dont le comportement est RechercheBeh de la manière suivante :

```
CreateCt.STD.create("localhost:2001", new RechercheBeh());
```

- La classe Pi à déployer peut être utilisée de la façon suivante (le résultat du calcul est un objet Java standard, BigDecimal) :

```
Pi pi = new Pi(100000);
BigDecimal result = pi.call();
```

- Enfin, le middleware JavAct vous offre les primitives suivantes (utilisables dans les méthodes des comportements, mais pas dans les constructeurs ni les classes qui ne sont pas des comportements) :

```
void become(Behavior beh)
```

Permet de changer le comportement courant en un nouveau comportement passé en paramètre, l'opération est retardée à la fin de l'exécution de la méthode courante.

```
void suicide()
```

Permet de terminer l'exécution de l'agent, qui ne répondra plus à aucun message et ne traitera pas les éventuels messages en attente.

```
void go(String p)
```

Déplacement de l'acteur sur une place p (p est soit un nom de machine du réseau local soit directement une adresse IP).

7) Passer en mode « exécution répartie »

Dans la section « import » de la classe SkeletonX, changez toutes les occurrences de `org.javact.`**`local`**`.XXX` par `org.javact.`**`net.rmi`**`.XXX`.

8) Lancement des systèmes d'accueil sur les machines du réseau

Vous devez utiliser le script **javactgogogo.sh** avec votre numéro de port en paramètre pour lancer les systèmes d'accueils JavAct :

```
[leriche@poste1 ~]$ ~leriche/javactgogogo.sh 2001
Démarrage sur d1-203-01...
Démarrage sur d1-203-02...
Démarrage sur d1-203-03...
Démarrage sur d1-203-04...
Démarrage sur d1-203-05…
[…]
(Creator) EXECUTION_DOMAIN=places.txt [warning: not found !]
(Creator) Security policy=awfullPolicy [ok, found]
(Creator) Binding on [rmi://10.3.51.7:2001/CreateurDeJavAct] [ok, bound]
JavAct v1.5.4 [10.3.51.7:2001] - 13:26:57
```

Le script ouvre une connexion ssh sur chaque machine distante et lance un système d'accueil JavAct. Vous pouvez ainsi voir en direct les éventuels messages d'erreur ou de débogage dans la console.

Pour exécuter votre application (le point d'entrée – la méthode main étant supposée être dans la classe Skeleton1) vous devez vous placer dans le répertoire de votre projet Eclipse puis utiliser le script **go.sh** :

```
lerichse@D1-203-07:~$ cd workspace/AgentPIEnac/
lerichse@D1-203-07:~/workspace/AgentPIEnac$ ~lerichse/go.sh
Je suis sur D1-203-07:2001
[…]
```

Pour terminer l'exécution des systèmes d'accueil et pour permettre l'exécution de nouvelles versions de vos classes, vous devez terminer l'exécution des systèmes d'accueil avec le script **javactkillall.sh**

```
[leriche@poste1 ~]$ ~leriche/javactkillall.sh
-->Terminating Javact VM on poste1...
-->Terminating Javact VM on poste2...
-->Terminating Javact VM on poste3...
[...]
```