UNIVERSITÉ DE MONS-HAINAUT

FACULTÉ DES SCIENCES

SIMULATION DE SYSTÈMES À ÉVÈNEMENTS DISCRETS

Rapport du projet

Auteurs :
Sébastien DUBOIS
Jean-François MERNIER
Frédéric REGNIER

5 mai 2009

Table des matières

Table des figures

Listings

1 Décisions

1.1 Gestion des évènements

Nous avons choisi d'utiliser une seule FEL pour la simulation. On y place tous les évènements. De plus, pour un temps t de simulation donné, nous avons décidé de traiter certains évènements prioritairement :

1. En premier lieu on traite les évènements de réception d'informations de routage
2. Puis les évènements de réception d'informations de routage
3. Puis les évènements de réception de messages (accusés et messages normaux)
4. Puis les évènements de timeout (un message pour lequel on a pas encore reçu d'accusé)

Une fois tous ces évènements traités pour un temps t, on traite les autres selon l'ordre *FIFO*.

1.2 Distance vector

Nous avons choisi de modifier les coûts en fonction du taux d'occupation du buffer de l'agent. Ceci est expliqué à la section 2.2.1 (p13) concernant l'évènement de réception d'un message par un agent.

Nous avons aussi implémenté la technique du *poisonned reverse* comme expliqué dans l'article concernant le distance vector.

2 Evenements

2.1 Hotes

2.1.1 Envoi d'un message original

L'envoi d'un message original par un hôte est illustré par la figure 2 (p7).

Comme le diagramme le montre, nous avons choisi de ne générer qu'un évènement d'envoi à la fois. Nous générons et plaçons le premier évènement sur la FEL. Quand celui-ci devient l'évènement imminent, l'hôte crée et envoie son message et génère l'évènement suivant puis le place sur la FEL.

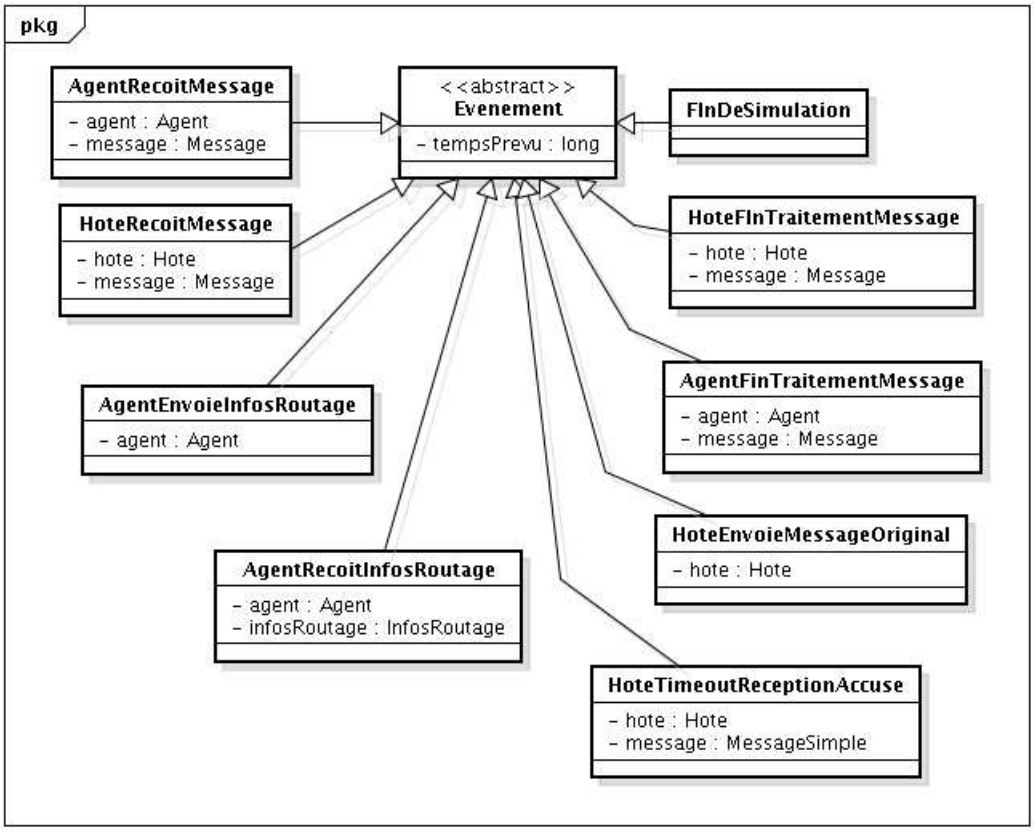

FIG. 1 – Evènements

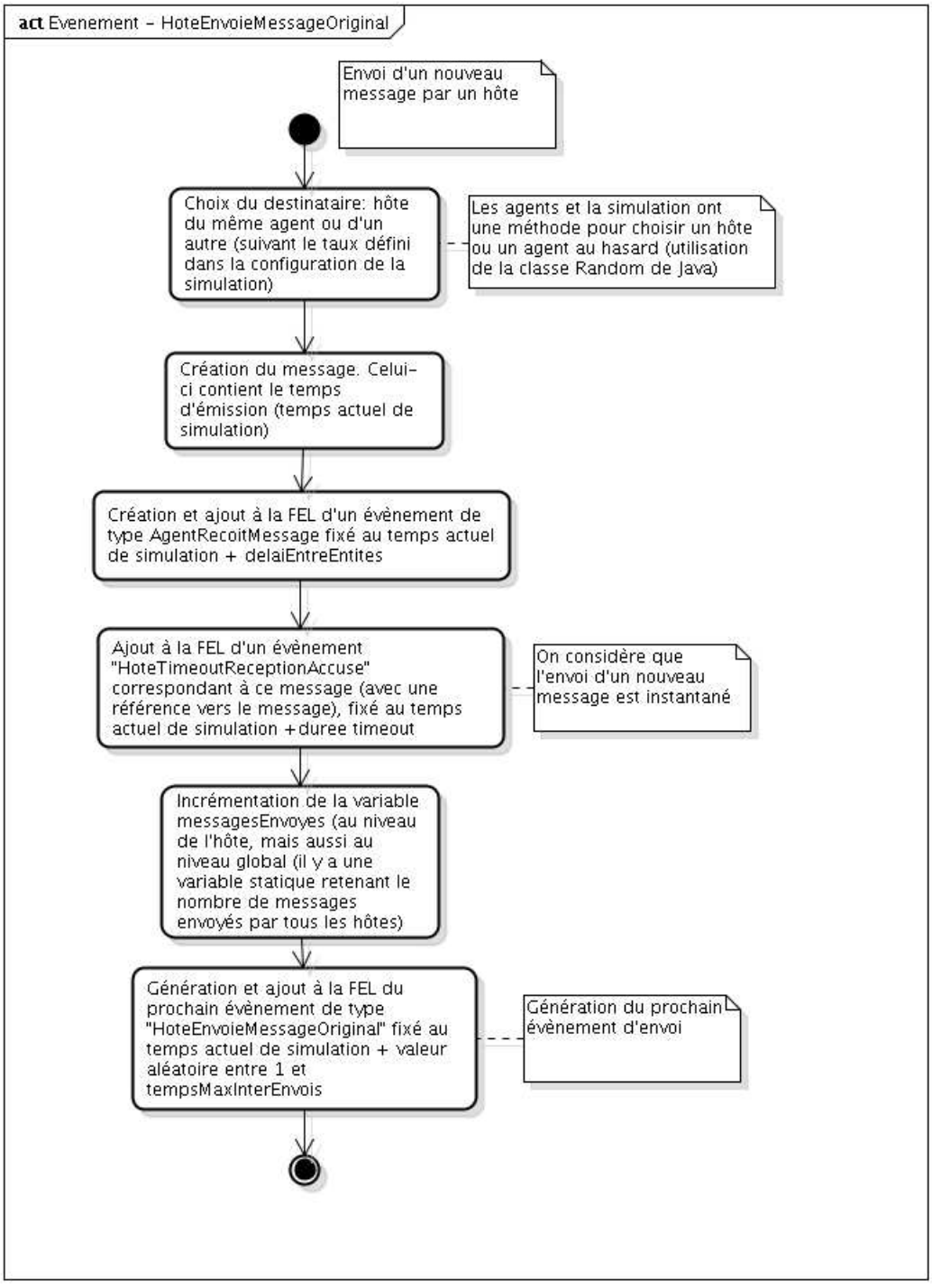

FIG. 2 – Envoi d'un message original par un hôte

Génération des temps d'envoi

Une variable aléatoire nous permet de générer pour chaque hôte le temps du prochain envoi (entre le temps actuel de simulation + 1 et le temps actuel de simulation + le temps inter-envois). Pour ce faire, chaque hôte dispose de sa propre instance :

Listing 1 – Génération des temps d'envoi

```
private final Random generateurTempsEnvoi = new Random();

...

/**
 * Generer le prochain temps d'envoi.
 *
 * @return le prochain temps d'envoi.
*/
private long genererTempsProchainEnvoi() {
  int tempsAvantProchainEnvoi =
      generateurTempsEnvoi.nextInt(getConfiguration()
          .getConfigurationHotes().getTempsMaxInterEnvois()) + 1;
  // on fait +1 et donc par exemple 0-5 devient 1-6
  return getSimulation().getHorloge() + tempsAvantProchainEnvoi;
}
```

Choix du destinataire

Un paramètre de simulation permet de spécifier le pourcentage de messages qui doivent être à destination d'un hôte connecté à un autre agent. Pour implémenter ceci, nous avons utilisé deux autres variables aléatoires. Chaque hôte dispose d'une variable aléatoire qu'on utilise avec des random digits de la manière suivante :

Listing 2 – Choix du type de destinataire d'un message

```
private final Random generateurTypeDestination = new Random();

...

int randomDigitsAutreAgent =
  (int) (getConfiguration().getConfigurationHotes()
  .getTauxMessagesVersAutreAgent() * 100);
int random = generateurTypeDestination.nextInt(100) + 1;
boolean messagePourHoteAutreAgent = false;

if (random <= randomDigitsAutreAgent) {
  messagePourHoteAutreAgent = true;
}
```

D'un autre côté, des méthodes se servant de variables aléatoires permettent de choisir un agent ou un hôte aléatoirement. De cette façon, un hôte peut déterminer de manière aléatoire qui devra reçevoir son message :

Listing 3 – Hote - Choix de la destination

```
1 if (messagePourHoteAutreAgent) {
2   hoteDestination = getSimulation().getAgentAleatoire(this.getAgent()).
        getHoteAleatoire();
3 }
4 else {
5   hoteDestination = this.getAgent().getHoteAleatoire(this);
6 }
```

Dans le premier cas, si le message doit être à destination d'un hôte connecté à un autre agent, l'hôte demande un agent aléatoire avec comme exception son propre agent (puisque ça doit être n'importe quel agent autre que le sien). Dans le second, l'hôte demande simplement à son agent de choisir un de ses hôtes aléatoirement (à l'exception de l'hôte actuel). Par exemple, voici la méthode permettant d'obtenir un agent aléatoire :

Listing 4 – Utilisation des variables aléatoires pour sélectionner un agent

```
1  /**
2   * PRNG utilise pour choisir un agent au hasard.
3  */
4  private final Random    generateurChoixAgent  = new Random();
5
6  ...
7
8  /**
9   * Recuperer un agent aleatoire pouvant etre n'importe lequel sauf celui
10  * fourni en argument.
11  *
12  * @param exception
13  *        le seul agent ne pouvant pas etre retourne
14  * @return un agent aleatoire autre que celui donne en argument
15 */
16 public Agent getAgentAleatoire(final Agent exception) {
17   Agent retVal = null;
18   do {
19     switch (generateurChoixAgent.nextInt(7) + 1) {
20       case 1:
21         retVal = agent1;
22         break;
23       case 2:
24         retVal = agent2;
25         break;
26       case 3:
27         retVal = agent3;
28         break;
29       case 4:
30         retVal = agent4;
31         break;
32       case 5:
33         retVal = agent5;
34         break;
35       case 6:
36         retVal = agent6;
37         break;
38       case 7:
39         retVal = agent7;
40         break;
41       default:
42         LOGGER
43             .error("Un probleme a eu lieu pendant la selection aleatoire d'un agent.
                   ");
44     }
45   } while (retVal == null || exception.equals(retVal));
46
47   return retVal;
48 }
```

2.1.2 Réception d'un message

La réception d'un message par un hôte est illustrée par la figure 3 (p11).

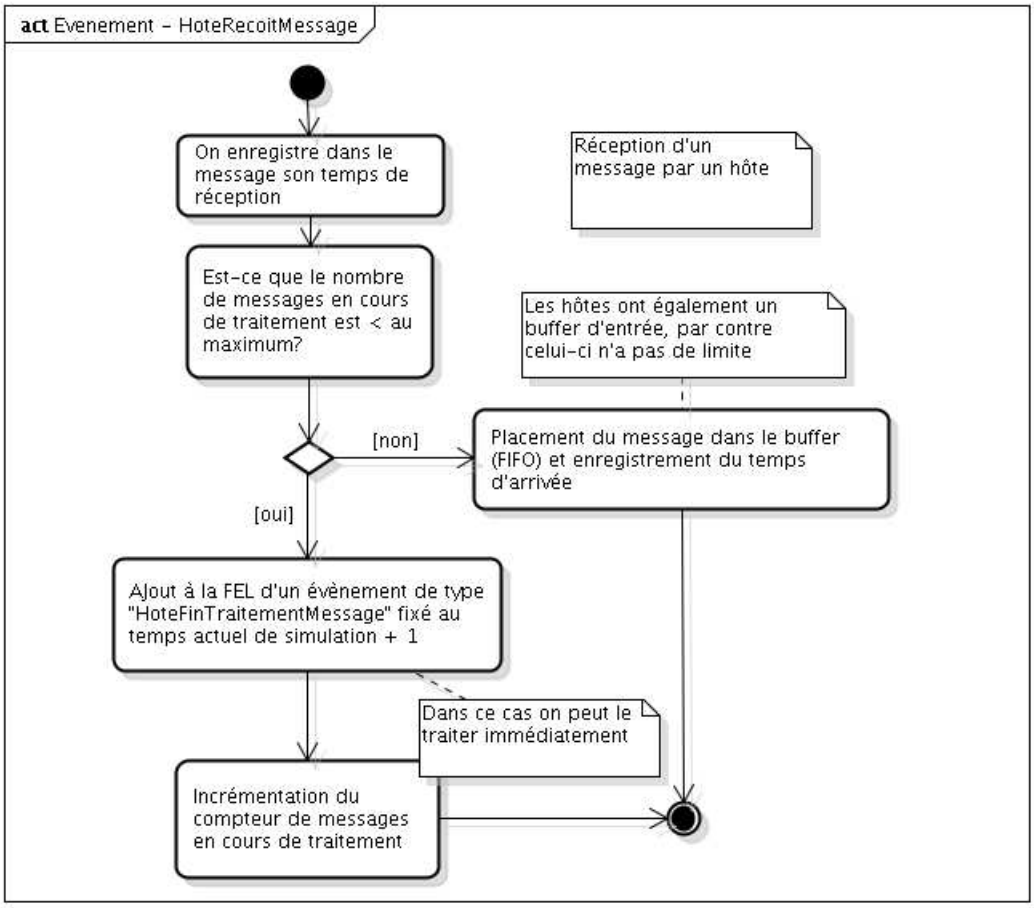

FIG. 3 – Réception d'un message par un hôte

2.1.3 Fin de traitement d'un message

Le diagramme UML correspondant étant trop grand, nous ne l'avons pas inclus dans le rapport. Il est disponible dans le dossier **UML** qui accompagne le rapport (le fichier : **Evenement - HoteFinTraitementMessage.png**).

2.1.4 Timeout

Les actions prises lors d'un timeout sont illustrées par la figure 4 (p12).

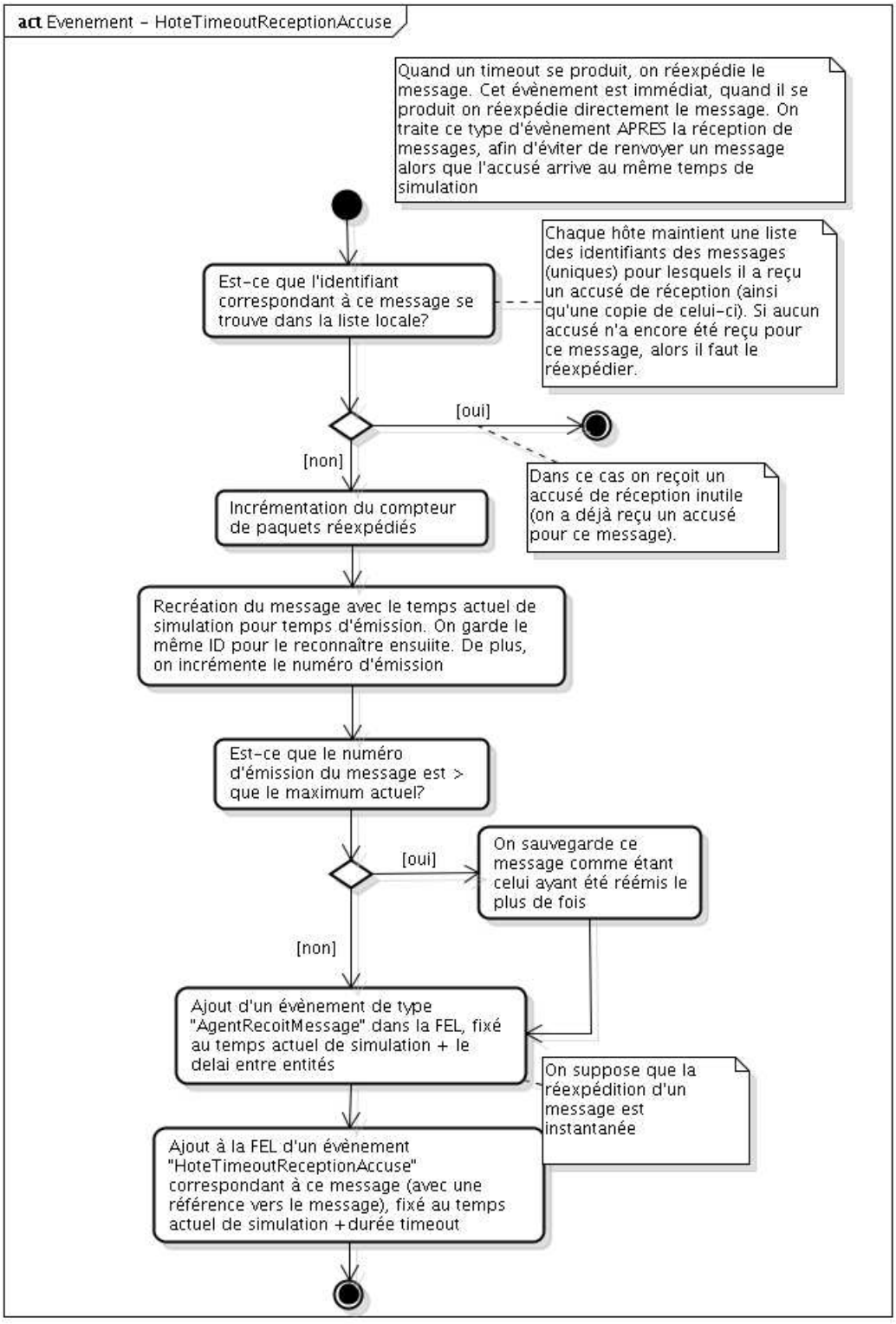

FIG. 4 – Timeout de réception d'un accusé

2.2 Agents

2.2.1 Réception d'un message

Le diagramme UML correspondant étant trop grand, nous ne l'avons pas inclus dans le rapport. Il est disponible dans le dossier **UML** qui accompagne le rapport (le fichier : **Evenement - AgentRecoitMessage.png**).

Dans le diagramme UML, il y a deux points que nous n'avons pas expliqués (en jaune) :

- Le premier point : « Est-ce qu'on peut envoyer les nouvelles informations de routage maintenant ? ». Nous avons décidé d'éviter trop d'envois successifs inutiles d'informations de routage, afin de ne pas surcharger le système. Pour ce faire, quand un agent doit envoyer les informations de routage (à cause du niveau d'occupation du buffer), il vérifie si ça fait au moins x temps de simulation qu'il a envoyé un message de ce type. Si oui, alors il peut envoyer le message. De cette manière, si pour un temps t donné, l'agent reçoit 50 messages, que pour le premier il dépasse le seuil d'alerte d'occupation du buffer et envoie ses informations de routage, puisqu'il sera toujours au delà du seuil d'alerte pour les 49 autres messages, il ne renverra plus de message avant un certain délai.
- Le second point : « Augmentation du coût de nos routes à destination des autres agents (on augmente d'une valeur fixe à chaque fois) ». Nous faisons ceci afin que le distance vector prenne en compte le niveau d'occupation des buffers des agents. Quand un agent donné est surchargé, il augmente le coût de ses routes à destination des autres agents et prévient ses voisins (i.e., leur envoie son DV). De cette manière quand les voisins reçoivent les informations, ils mettent à jour leur propre table de routage et choisissent peut être d'autres routes pour faire suivre les messages (i.e., changent leur DV). Si les autres agents deviennent surchargés, leurs coûts augmenteront également. Ainsi au final, le DV prend en compte l'occupation des buffers des agents, ce qui permet de mieux répartir la charge sur les différents agents. Dans les résultats des simulations, nous avons en effet constaté que les buffers sont utilisés de manière plus homogène quand le distance vector est activé.

2.2.2 Fin de traitement d'un message

Le diagramme UML correspondant étant trop grand, nous ne l'avons pas inclus dans le rapport. Il est disponible dans le dossier **UML** qui accompagne le rapport (le fichier : **Evenement - AgentFinTraitementMessage.png**).

2.2.3 Envoi des informations de routage

L'envoi d'informations de routage (envoi du DV d'un agent à ses voisins) est illustré par la figure 5 (p14).

2.2.4 Réception d'informations de routage

La réception d'informations de routage (réception du DV d'un voisin par un agent) est illustrée par la figure 6 (p15).

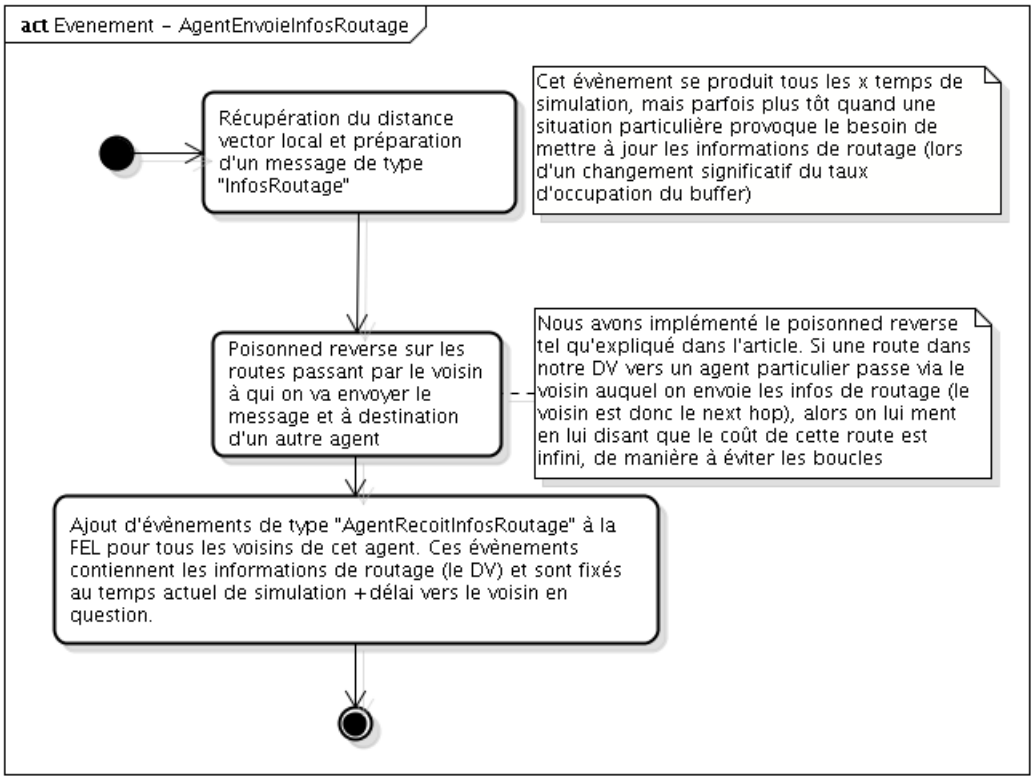

FIG. 5 – Envoi d'informations de routage par un agent

Le détail des opérations effectuée lors de la mise à jour (vérifications, etc) n'est pas dans le rapport non plus car le diagramme est trop grand. Le fichier est disponible dans le dossier **UML** qui accompagne le rapport (le fichier : **Mise à jour des informations de routage.png**).

2.3 Autres

2.3.1 Fin de simulation

Le dernier évènement est spécial ; c'est celui qui provoque la fin de la simulation. Quand cet évènement devient l'évènement imminent, il remet la FEL à zéro (il n'y a donc plus d'évènement imminent après lui) et la boucle de simulation s'arrête. Le détail est illustré par la figure 7 (p16).

3 Messages

Les entités de la simulation s'envoient différents types de messages. Un diagramme UML les illustrant existe mais n'a pas été inclus au rapport car il est trop grand. Il est disponible dans le dossier **UML** qui accompagne le rapport (le fichier : **CD Messages.png**).

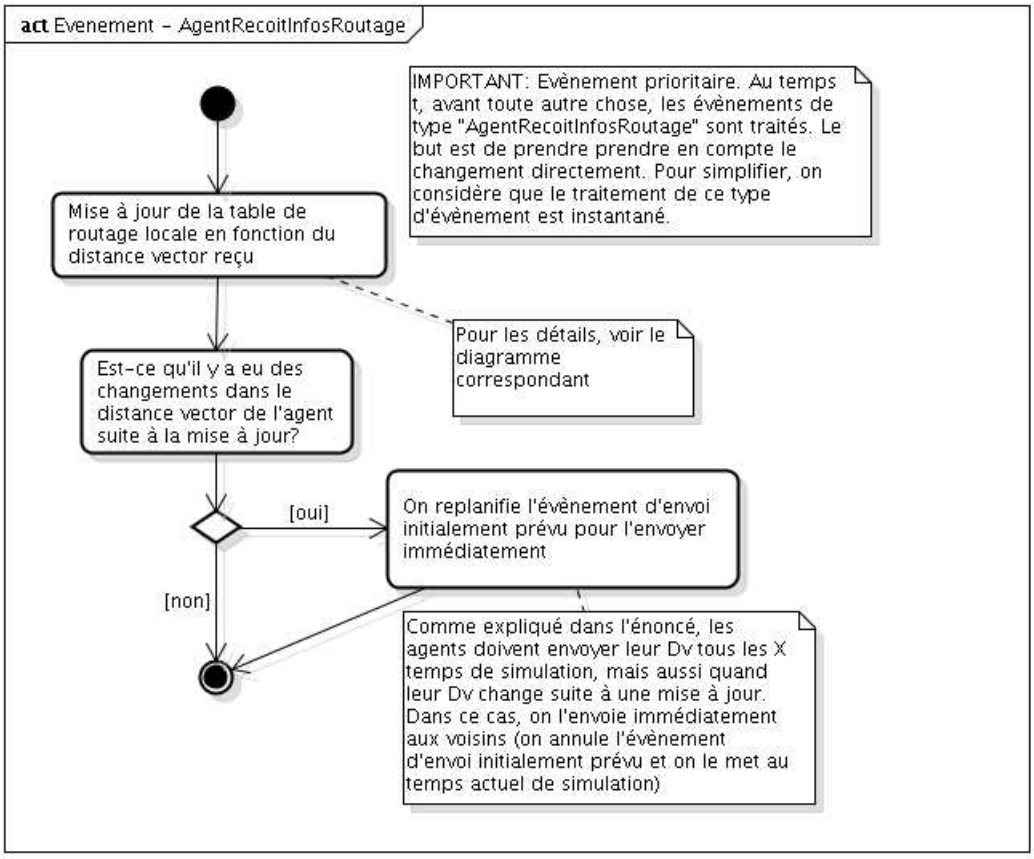

FIG. 6 – Réception d'informations de routage par un agent

4 Paramètres du système

4.1 Hote

- Durée du timeout (temps après lequel on réémet un message)
- Temps maximal inter-envois (pour les messages originaux)
- Temps de traitement d'un message
- Pourcentage de messages à destination d'un autre agent

4.2 Agent

- Nombre d'hôtes reliés
- Taux de pertes brutales de messages
- Temps de traitement d'un message
- Taille de buffer (en entrée)

Pour le distance vector nous avons en plus :

- Temps inter-envois des informations de routage

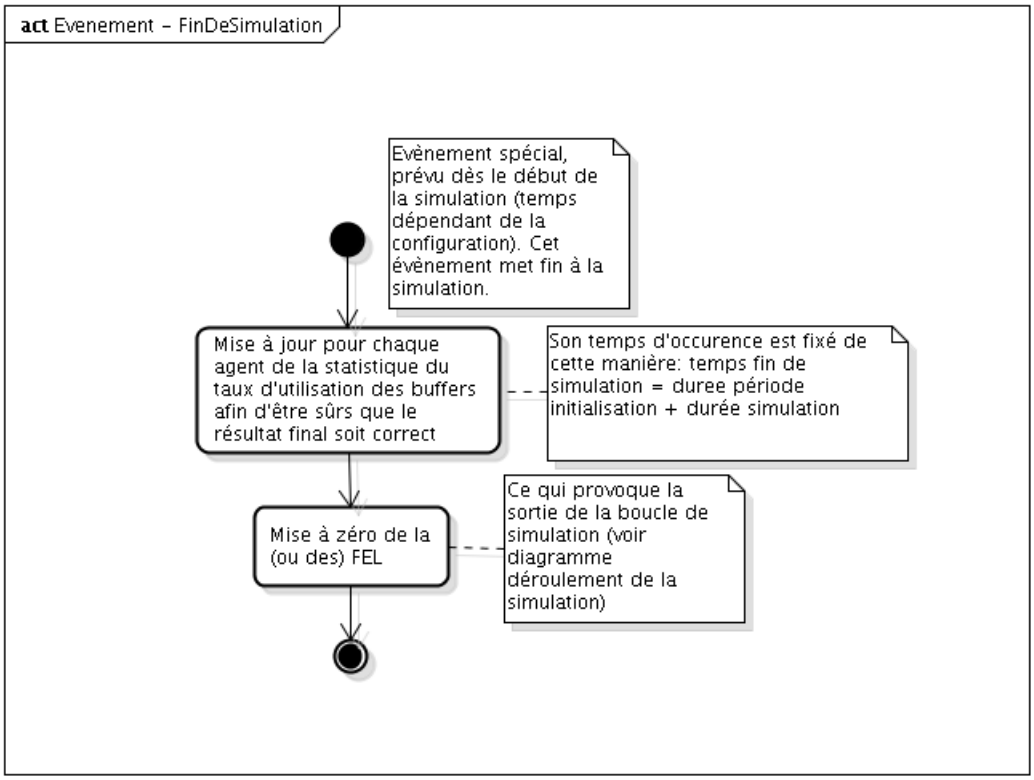

FIG. 7 – Fin de la simulation

4.3 Simulation

- Durée
- Délai agent <-> hôte
- Distance vector activé (oui/non)
- Durée de la période d'initialisation
- Périodicité d'affichage des statistiques (e.g., tous les 1% de simulation)

5 Déroulement de la simulation

La figure 8 illustre le déroulement de la simulation (les grandes étapes).

6 Calcul et affichage des résultats

La figure 9 (p18) illustre les différentes statistiques que nous calculons et affichons pendant la simulation.

7 Résultats

TODO !

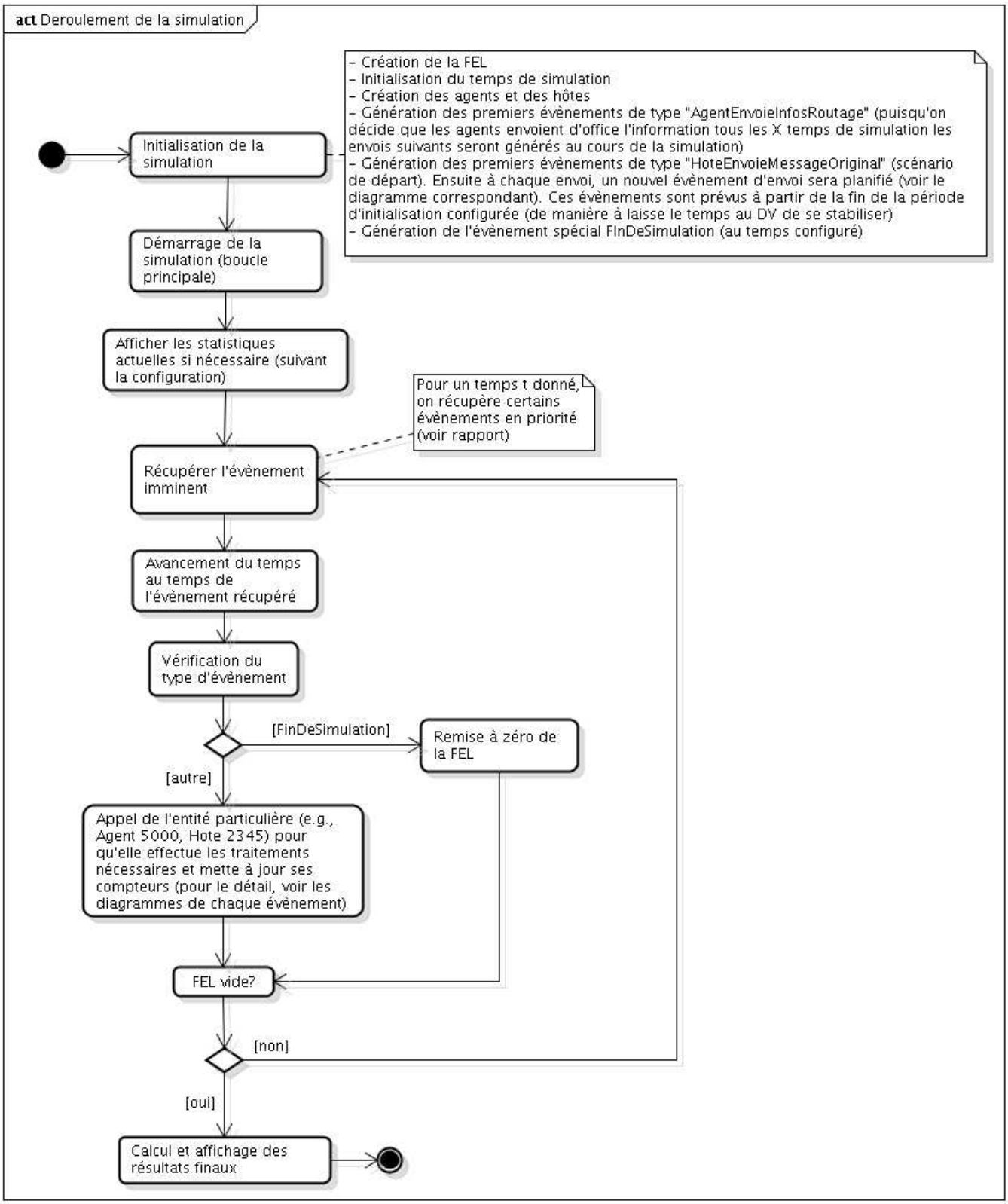

FIG. 8 – Déroulement de la simulation

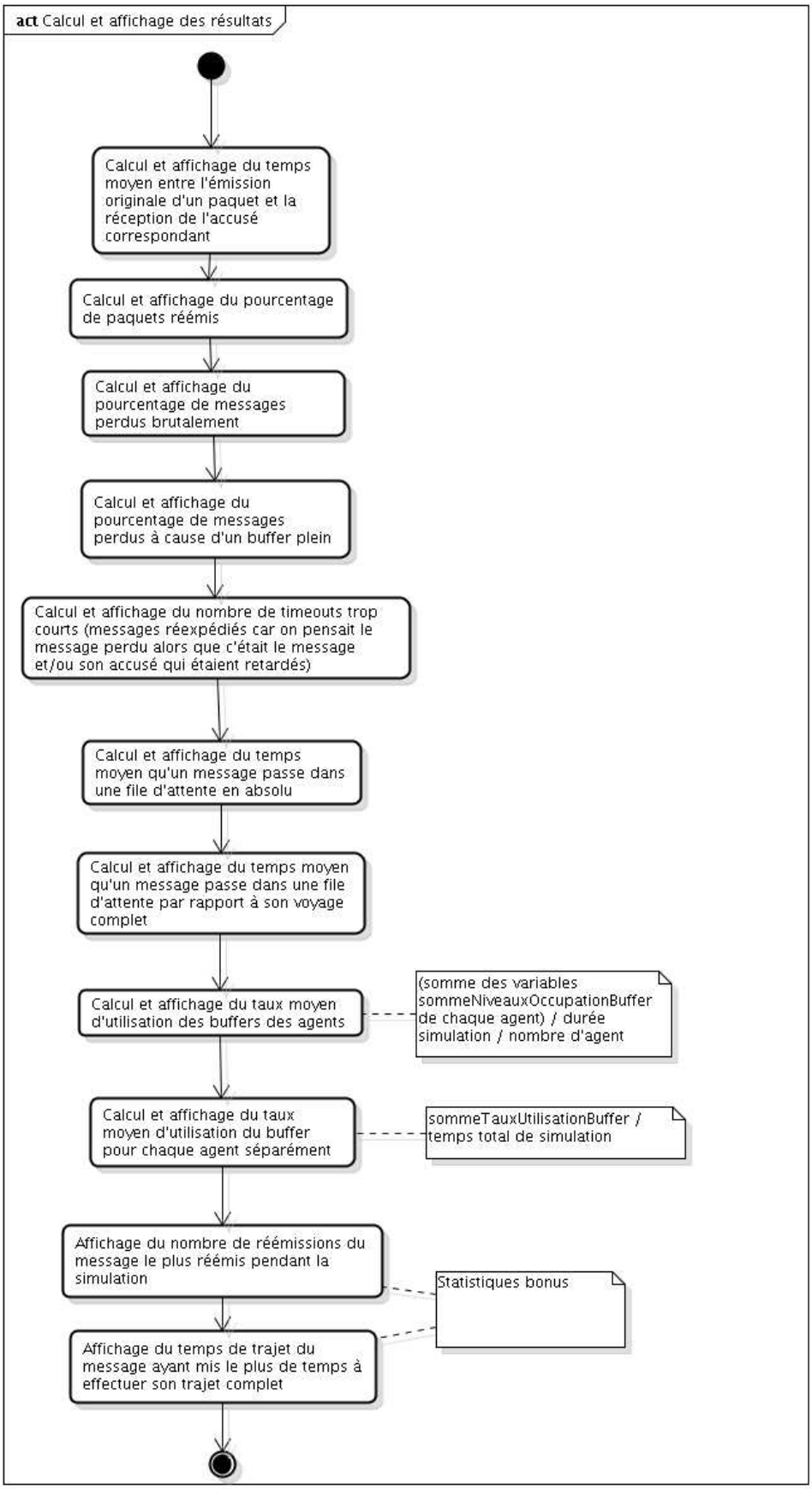

FIG. 9 – Calcul et affichage des résultats

A Le programme et son utilisation

L'exécutable du projet est disponible dans le dossier **target/dist**. Pour l'exécuter, il suffit d'ouvrir un prompt et de lancer : **java -jar simulation.jar**.

A.1 Configuration et aide

Pour afficher la liste des paramètres pouvant être donnés au programme, il suffit d'ouvrir un prompt et de lancer : **java -jar simulation.jar –aide**

```
)ption                                  Description
------                                  -----------
-a, -h, --aide, --help                  Aide
--agentsNombreHotes <Long>              Nombre d'hotes par agent
--agentsTailleMaxBuffer <Long>          Taille des buffers des agents (>= 0)
--agentsTauxPerteBrutale <Float>        Taux de perte brutale des agents (e.
                                          g., 0.05) (0 <= valeur < 1)
--agentsTempsInterEnvoiInfosRoutage     Temps entre deux envois des
  <Integer>                               informations de routage (>=0)
--agentsTempsTraitementMessage <Float>  Temps de traitement d'un message par
                                          un agent (0 <= temps traitement <=
                                          1). 0 = traitement instantané
--delaiEntreEntites <Integer>           Délai nécessaire pour qu'un message
                                          d'un hôte arrive à l'agent (et
                                          inversément) (>=0)
--duree <Long>                          Durée de la simulation (> 0)
--dureeInitialisation <Long>            Durée de la période d'initialisation
                                          de la simulation (>= 0)
--dvActive                              Pour activer le distance vector (si
                                          non spécifié, désactivé!)
--hotesTauxMessagesVersAutreAgent       Taux de messages d'un hôte qui seront
  <Float>                                 à destination d'un hôte relié à un
                                          autre agent (e.g., 0.75) (0 <=
                                          valeur <= 1)
--hotesTempsMaxInterEnvois <Integer>    Temps maximal entre deux envois d'un
                                          hôte (> 0)
--hotesTempsTraitementMessage <Float>   Temps de traitement d'un message par
                                          un hote (0 <= temps traitement <=
                                          1). 0 = traitement instantané
--hotesTimeoutReemissionMessages        Timeout après lequel les messages
  <Integer>                               doivent etre réexpédiés si aucun
                                          accusé de réception n'est reçu (> 80)
--periodiciteAffichageStats <Float>     Périodicité d'affichage des
                                          statistiques (0 < periodicite <= 1)
```

FIG. 10 – Options disponibles

Ensuite pour spécifier les options, on peut par exemple faire : **java -jar simulation.jar –agentNombreHotes 1000 –duree 5000**.

A.2 Organisation du code source

- Les sources se trouvent dans le dossier **src/main/java**
- Les fichiers de configuration par défaut se trouvent dans le dossier **src/main/resources/configuration**

Le point d'entrée du programme est la classe *Main* qui se trouve dans **src/main/java/be/simulation**.

A.3 Exécution de la simulation

Lancer le programme exécute directement la simulation. Si aucune option n'est spécifiée en argument au programme, les valeurs par défaut sont utilisées. Les résultats sont affichés à l'écran et sauvegardés dans un fichier de log.

A.4 Compilation

La compilation du code requiert l'utilisation de Maven [1], un outil de build très simple d'utilisation. En étant dans le dossier du projet (au niveau où se trouve le fichier **pom.xml**), il suffit d'exécuter la commande suivante : **mvn package**. Une fois terminé, le fichier jar exécutable est disponible dans le dossier **target/dist**.

Maven est très simple à installer sur la plupart des distributions Linux (e.g., Ubuntu, ...). Sous Windows, il suffit de le télécharger et d'ajouter le dossier bin au path.

A.5 UML

Un diagramme de classes montrant les classes principales du projet existe mais n'a pas été inclus dans le rapport car il est trop grand. Il est disponible dans le dossier **UML** qui accompagne le rapport (le fichier : **CD Entites.png**).

[1] `http://maven.apache.org/`