Simulation d'algorithmes distribués avec **Sinalgo**

Stéphane Devismes

Résumé

Sinalgo est une plateforme de simulation qui permet de tester et de valider ≪ à haut niveau ≫ des algorithmes distribués. Cette platefome est écrite en JAVA. Elle est initialement dédiée aux protocoles pour réseaux de capteurs sans fil, mais nous l'utiliserons pour implémenter tout type de protocoles.

1 Installation

1.1 Pré-requis

Sinalgo nécessite que **Java 5.0** ou une version supérieure soit installé sur votre machine. **Sinalgo** fonctionne aussi avec des environnements de développement de type **Eclipse**.

1.2 Installation

Téléchargez le fichier zip à l'adresse suivante :

`http://www-verimag.imag.fr/~devismes/SINALGO/sinalgo-0.75.3-regularRelease.zip`

Décompressez le dans un sous-répertoire *sinalgo* de votre répertoire de travail. Ce répertoire contient entre autres, *les sources* dans *src* et les fichiers binaires dans *binaries/bin*.

Deux choix s'offrent alors à vous :

- Soit vous lancez **Sinalgo** à partir de la ligne de commande en tapant : `java -cp binaries/bin sinalgo.Run` à partir du répertoire *sinalgo*.
- Soit vous utilisez un environnement de développement type **Eclipse** (*cf.* ci-dessous).

1.3 Avec Eclipse

1. Lancez **Eclipse**.
2. Créez un nouveau projet java (File→New→Java Project). Nommez-le *sinalgo*. Décochez la case ≪ use default location ≫ et sélectionnez le répertoire *sinalgo* créé précédemment. Cliquez sur *finish*.
3. Une fois ces actions effectuées, il suffit de faire un clic droit sur *src* dans le navigateur d'**Eclipse** et de sélectionner ≪ *Run as→Java Application* ≫ pour démarrer l'application.

2 Organisation

Sinalgo propose un ensemble de sources de projets : des exemples (*sample*), un projet par défaut (*defaultProject*) et un projet ≪ vide ≫ (*template*). Chaque fichier source est stocké dans un répertoire situé dans `sinalgo/src/projects`. L'organisation de chaque projet est identique et comporte :

- Un fichier *Config.xml* qui permet de paramétrer le modèle d'exécution.
- Un fichier *description.txt* ; l'ensemble des informations de ce fichier sont affichées dans le menu où l'on sélectionne la simulation à exécuter.
- Un fichier *CustumGlobal.java* qui permet de modifier l'interface de simulation.
- Un fichier *LogL.java* qui permet de développer des fonctions de ≪ monitoring ≫.
- Optionnellement, un fichier *run.pl* ; ce fichier *Perl* permet de gérer le paramètrage d'une exécution.
- Un répertoire *images* ; ces images peuvent être utilisées pour personnaliser les menus, etc.
- Deux répertoires : *models* et *nodes* qui contiennent les implémentations des modèles et des noeuds (*cf.* ci-dessous).

Les fichiers et répertoires importants sont détaillés ci-dessous. Pour plus d'informations, voir `www.disco.ethz.ch/projects/sinalgo/`.

2.1 Modèles

Les modèles décrivent l'environnement dans lequel votre protocole est simulé. Vous pouvez implanter vos propres modèles ou utiliser des modèles prédéfinis. Ici, il ne vous sera pas demandé de développer vos propres modèles. Pour chaque simulation, il faut choisir 6 types de modèles :

- Le *modèle de connectivité* qui décide comment évaluer le voisinage d'un noeud. Nous utiliserons uniquement le modèle *UDG*. Le modèle *UDG* permet de créer des réseaux à disques unitaires, c'est-à-dire on définit une portée R et une fois les nœuds déployées, deux nœuds seront voisins si leur distance est inférieure à R.
- Le *modèle de distribution* qui décrit comment les noeuds sont placés au démarrage de l'application. Selon les applications, nous utiliserons les modèles suivants :
  - *Circle* permet de disposer les nœuds en anneau.
  - *Random* dispose les nœuds aléatoirement sur le plan.
  - *PositionFile* génére un réseau à partir d'un fichier *.pos*. Ce dernier peut être créer à partir de l'interface graphique.
- Le *modèle d'interférence* qui décide si un message est perdu suite à une interférence. Nous paramétrerons toujours le modèle pour qu'il n'y ait pas d'interférence.
- Le *modèle de transmission* qui décide du temps d'acheminement des messages.
  - *ConstantTime* permet d'exécuter le protocole en mode synchrone : chaque message est délivré en temps constant.
  - *RandomTime* permet d'exécuter le protocole en mode asynchrone, mais l'ordre FIFO n'est pas respecté.
  - Nous avons développé un modèle de transmission *FIFO* et *asynchrone*. Il est accessible à l'adresse suivante : `http://www-verimag.imag.fr/~devismes/SINALGO/FifoRandom.java`. Pour pouvoir l'utiliser, vous devez copier ce fichier dans le répertoire *messageTransmissionModels* de *defaultProject*.
- Le *modèle de mobilité* qui gère le mouvement des noeuds. Nous paramétrerons toujours le modèle pour que les nœuds restent immobiles.
- Le *modèle de fiabilité* qui gère la perte de messages.
  - *ReliableDelivery* assure que les canaux sont fiables.
  - *LossyDelivery* permet d'obtenir des canaux avec un taux de perte défini.

L'utilisation de ces modèles est paramétré à l'aide du fichier *Config.xml*, que nous décrivons ci-après.

2.2 Noeuds

Le répertoire *nodes* contient le code du protocole. Ce code est subdivisé en plusieurs fichiers répartis dans plusieurs répertoires :

- *edges*. Contient le code relatif aux liens de communication (ici, nous utiliserons uniquement l'implantation par défaut donc ce répertoire ne contiendra aucun fichier).
- *messages*. Contient la description de chaque type de message utilisé dans la simulation.
- *nodeImplementations*. Contient le code pour le comportement des noeuds.
- *timers*. Contient le code des *timers* utilisés dans la simulation.

2.3 Config.xml

Nous détaillons maintenant les principales balises d'un fichier *Config.xml*.

- `asynchronousMode` : définit si les nœuds sont asynchones. Ici, nous auront des processus synchones mais des liens asynchrones (`value="false"`).
- `mobility` : défini si les noeuds sont mobiles ou pas. Nous interdirons systématiquement la mobilité (`value="false"`).
- `interference` : défini si les communications sont sujettes aux interférences. Nous interdirons systématiquement les interférences (`value="false"`).
- `edgeType` : défini le type de lien de communications. Ici nous choisirons des liens bidirectionnels (`value="sinalgo.nodes.edges.BidirectionalEdge"`).
- `exitOnTerminationInGui` : défini si l'application doit être fermée à la fin d'une simulation. Ici, nous ne fermerons pas l'application en cas de terminaison (`value="false"`).
- `initializeConnectionsOnStartup` : défini si les liens de communications sont créés au démarrage de la simulation. Ici, nous choisirons de créer les liens au démarrage (`value="true"`).
- `refreshRate` : défini quand l'affichage se rafraîchit. Nous lui donnerons la valeur 1.
- `DefaultMessageTransmissionModel` : défini le modèle de transmission (synchrone/asynchrone, FIFO ou pas). Plusieurs valeurs sont possible :

  - `"ConstantTime"` (communication synchrone). Si vous l'utilisez, alors il faut redéfinir la balise `MessageTransmission` en `<MessageTransmission ConstantTime="1"/>`. Dans ce cas, les messages sont tous reçus après une unité de temps.
  - `"RandomTime"` (communication asynchrone mais pas FIFO). Si vous l'utilisez, alors il faut redéfinir la balise `MessageTransmission` en `<RandomMessageTransmission distribution="Uniform" min="1" max="5"/>`. Dans cette balise, *min* et *max* définissent la borne inférieure et supérieure du temps de transmission des messages.
  - `"FifoRandom"` (communication asynchrone et FIFO). Si vous l'utilisez, alors il faut remplacer la balise `MessageTransmission` par `<FifoRandom distribution="Exponential" lambda="1" />`.
- `DefaultConnectivityModel` : défini la relation de voisinage entre les nœuds. Ici nous choisirons la valeur `"UDG"` (disque unitaire). La portée des disques se définit avec deux autres balises, par exemple `<GeometricNodeCollection rMax="200"/>` suivi de `<UDG rMax="200"/>` définissent une portée de 200 pixels.
- `DefaultDistributionModel` : défini le modèle de placement des nœuds. Trois valeurs sont possible :
  - `"Circle"` pour placer les nœuds en anneau.
  - `"PositionFile"` pour placer les nœuds en fonction d'un fichier.
  - `"Random"` pour placer les nœuds au hazard.
- `DefaultMobilityModel` : défini le modèle de mobilité si on a choisi des nœuds mobiles. Ici sa valeur devra être `"NoMobility"`.
- `DefaultReliabilityModel` : défini la fiabilité des liens. Deux choix possibles :
  - `value="ReliableDelivery"` pour avoir des communications fiables.
  - `value="LossyDelivery" dropRate="0.6"` pour avoir, par exemple, des pertes de 60%. De plus, vous devez ajouter la balise `<LossyDelivery DropRate="0.6"/ >` à l'intérieur de la balise `<Custom>` (à la fin du fichier XML).
  - `DefaultNodeImplementation` : défini le type de nœud à déployer. Par exemple, si on écrit `value="essai:essainode"`. Les nœuds créés seront des nœuds `essainode` défini dans le projet `essai`.
- `Node` : dans son paramètre `defaultSize` est spécifié la taille de nœuds pour l'affichage.

3 Un petit exemple...

Lancez une exécution de l'exemple *sample1*. Vous pouvez le lancer via **Eclipse** ou directement en tapant :

`java -cp binaries/bin sinalgo.Run -project sample1`

Une fenêtre apparait alors avec un plan en 3 dimensions. Pour lancer la simulation, allez dans le menu *Simulation* et cliquez sur *Generate Nodes*. Créez 100 noeuds puis cliquez sur l'icone verte pour démarrer la simulation.

Vous pouvez ensuite paramétrer la simulation directement à partir de la ligne de commande. Par exemple, taper :

`java -cp binaries/bin sinalgo.Run -project sample1 -gen 1000 sample1 :S1Node RandomC=UDG -rounds100.`

Cette commande lance la simulation *sample1* avec 1000 noeuds de type $S1Node$, ces noeuds sont placés de manière aléatoire, le modèle de connectivté utilisé est UDG, enfin la simulation s'arrête après 100 unités de temps.

4 Exclusion Mutuelle de Le Lann

Pour créer notre simulation, nous allons tout d'abord faire une copie du répertoire *template* situé dans le répertoire *src/ projects* dans le même répertoire. Cette copie sera re-nommée *tokenRing*.

Nous allons ensuite créer de nouveau type de noeuds, messages, et timers. Pour cela, téléchargez les fichiers *Config.xml*, *tokenMessage.java*, *tokenNode.java*, *demandeTimer.java*, *waitTimer.java* et *initTimer.java* situés à l'adresse suivante : `http://www-verimag.imag.fr/~devismes/SINALGO/`).

Dans votre répertoire de projet *tokenRing* :

- Copiez le fichier *Config.xml* à la racine de répertoire.
- Remplissez le fichier *description.txt*.
- Copiez le fichier *tokenMessage.java* dans le répertoire *nodes/messages*.
- Copiez le fichier *tokenNode.java* dans le répertoire *nodes/nodeImplementations*.
- Copier les fichiers *demandeTimer.java*, *waitTimer.java* et *initTimer.java* dans le répertoire *nodes/timers*.

Lancez une exécution avec 20 noeuds, puis étudiez le code et le contenu du fichier Config.xml.