Manuel d'utilisation de Sim 1.0

Laurent Beaudou, Etienne Duchesne, Emmanuel Filiot

7 mai 2003

Table des matières

1 Introduction

Tout d'abord vous pourrez trouver la javadoc à l'adresse http ://www.ens-lyon.fr/ efiliot/poogl/

Le logiciel SIM est un simulateur de dynamiques sur des réseaux d'interactions sociales. Un réseau d'interactions sociales est représenté par un graphe, orienté ou non, dont chaque sommet possède un état. La dynamique sur le graphe est définie par une fonction de transition $\delta : \prod_{k=1}^{n} \mathcal{U}_{i=1}^{d} S^i \to S$ où S est l'ensemble des états, d le degré entrant maximum d'un sommet, et n le nombre de sommets. Nous noterons plutôt $\delta = (\delta_1, \ldots, \delta_n)$, où chaque δ_i est la fonction de transition partielle associée au sommet i. Chaque δ_i associe aux états des voisins entrants du sommet i le nouvel état du sommet i.
Etant donnée une configuration initiale C_0, le but est d'étudier l'orbite, i.e. la suite $(C_i)_i$ de configurations obtenues en itérant l'application de la fonction de transition.

SIM permet de créer de grands graphes (de l'ordre de 10^5 sommets), de définir une dynamique, de l'appliquer, et de définir et d'appliquer des mesures sur le graphe.
Il y a deux modes d'exécution, un mode graphique, avec une interface graphique, et un mode console.

2 Installation

Pour éviter les problèmes de compilation dus à l'interface graphique et à une version trop ancienne de Java, il y a deux modes de compilation, un mode Console et Graphique, et un mode Console uniquement. Si des problèmes surviennent à la compilation en mode Console et Graphique, il est préférable de ne compiler qu'en mode Console, dans ce cas les fonctionnalités de SIM ne s'étendent plus à l'interface graphique.
L'installation est détaillée dans le fichier `install`, mais nous en rappelons ici les principes :

- `./make.sh install` pour le mode graphique et console
- `./make.sh install -C` pour le mode console uniquement
- `./make.sh clean` pour effacer tous les fichiers .class
- `./make.sh init` pour compiler les fonctions d'initialisation du répertoire sim/protocol/init
- `./make.sh function` pour compiler les fonctions de transition du répertoire sim/apply/function
- `./make.sh doc` pour construire la javadoc dans le répertoire doc/
- `./make.sh measure` pour compiler toutes les mesures du répertoire sim/measure/

Il est conseillé de faire `./make.sh clean` entre deux compilations en mode console, et en mode console et graphique.

Il est aussi conseillé de bien sauvegarder tous les fichiers avant de faire la javadoc (un fichier ouvert et non sauvegardé à cet instant entraînerait l'échec de la construction de la javadoc).

3 Lancement de SIM

- Se placer dans le répertoire contenant le fichier `make.sh`.
- `java sim/Sim` pour lancer SIM en mode graphique, s'il est disponible.
- `java sim/Sim <initfunction>` pour lancer SIM en mode graphique, s'il est disponible, avec la fonction d'initialisation initfunction.
- `java sim/Sim -protocol <initfunction1> <initfunction2> ...` pour lancer un protocole en mode console (point détaillé dans une section suivante).
- `java sim/Sim -protocol [Options] <initfunction1> <initfunction2> ...` pour lancer SIM en mode console avec les options détaillée ci-dessous :

Options

`-realtime` pour faire les mesures temps réel (i.e. entre chaque transition) définies dans initfuncion1, initfunction2, ...
`-f <filename>` pour enregistrer le compte-rendu des simulations dans filename, par défaut le compte-rendu est enregistré dans account.log. ATTENTION cette option doit toujours être après -realtime dans le cas où -realtime est indiqué.
Voir sim/Sim.java et la javadoc pour plus de détails.

4 Paquetages et classes fondamentales

4.1 Introduction

SIM est divisé en plusieurs paquetages :

- `sim.graph` : contient toutes les classes permettant de définir des graphes
- `sim.apply` : contient les classes permettant de définir l'éxécution, asynchrone ou synchrone (voir une section suivante)
- `sim.apply.function` : contient l'interface `FunctionTransition` définissant les fonctions de transitions, et toutes les implémentations de cette interface.
- `sim.state` : contient les états et les ensembles d'états
- `sim.measure` : contient l'interface `Measure` définissant une mesure, ses implémentations, et d'autres classes gérant des groupes de mesures.
- `sim.protocol` : contient les classes définissant un protocole de simulation pour le mode console

- `sim.protocol.init` : contient l'interface `InitFunction` et ses implémentations, i.e. les fonctions d'initialisation utilisée pour définir un protocole, ou dont l'une est lancée au démarrage de SIM.
- `sim.view` : contient tout ce qui concerne l'interface graphique
- `sim.save` : contient tout ce qui concerne la sauvegarde
- `sim.tools` : contient la class `Tools` dans laquelle sont définies des méthodes statiques utiles.
- `sim` : contient la classe `Sim` définissant la méthode statique `main` qui lance le logiciel.

Pour comprendre le fonctionnement général de SIM, nous introduisons d'abord quelques classes et notions utilisées dans le logiciel.

Les sommets du graphe contiennent un état, qui peut être entier, ou réel, et qui peut être redéfini en respectant l'interface State. Nous avons crée une interface `StateSet` pour faciliter la gestion des états, un ensemble d'état peut être fini (il contient alors des états entiers), dénombrable (contient alors des états entiers, mais longs) ou réel (contient des états réels représentés par des doubles). Ainsi, pour faire une simulation, les objets suivant sont indispensables :

- le mode d'affichage, graphique ou console
- le mode d'exécution, borné (i.e. avec un nombre d'itérations fixé), ou non.
- le graphe
- un ensemble d'états.
- la configuration des états des sommets du graphe, qui est définie à l'aide des méthodes de la classe `SetConfiguration` du paquetage state.
- une fonction de transition définie par implémentation de l'interface `FunctionTransition` (i.e. une fonction qui va de l'ensemble des états des voisins d'un sommet vers le nouvel état du sommet).
- une exécution, i.e. un ensemble de méthodes permettant de faire toutes les transitions, les calculs, de manière synchrone, ou non.
- on peut aussi ajouter des mesures qui seront faites entre chaque transition. Une mesure doit implémenter l'interface `Measure`.

L'ensemble de ces données sont regroupées alors dans un objet `Configuration` dans le paquetage sim.protocol . La simulation peut commencer dès lors que la configuration est définie. L'interface graphique permet de définir la configuration avec des fenêtres, des menus, etc...En mode console, il faut regrouper toutes ces informations dans une classe spéciale que nous appelons fonction d'initialisation (`InitFuntion`). Cette interface impose de définir une méthode `init` qui initialise la configuration. C'est ainsi qu'en définissant plusieurs fonctions d'initialisation, nous pouvons définir ce que nous appelons un protocole de simulation (i.e. une suite de simulations) que nous développerons plus loin.
Cette courte introduction nous permet d'aborder de façon plus détaillée les notions que nous venons de voir.

4.2 Vertex

Un sommet est représenté par la classe `Vertex`, il contient un numéro, un état, et une liste de voisins (entrants dans le cas orienté), donc une liste d'objets Vertex. Un graphe est alors représenté par une liste de sommets, voir la section suivante.

4.3 Les Graphes

Salut à toi ô utilisateur potentiel, ce manuel va te révéler les merveilles de ma partie du logiciel, alors accroche toi c'est parti :

- La hiérarchie du package sim.graph est la suivante : en haut on trouve la classe abstraite Graph dont héritent les deux sous-classes abstraites Directed et Undirected qui elles même sont les mères des classes RandomGraph, Regular et Defined correspondantes (on rajoute U devant le nom de la classe pour spécifier qu'elle hérite de Undirected). On remarquera qu'il n'y a pas de classe Regular car les graphes réguliers ne peuvent pas être orientés.
- L'utilisateur a accès à quasiment tous les constructeurs et méthodes du package. Il peut en particulier :
  - Construire tous les types de graphes.
  - Effectuer des parcours en largeur ou en longueur sur chaque type de graphe.
  - Connexifier un graphe quelconque.
- Nous avons préféré stocker les graphes sous forme de listes d'adjacences, parce que les matrices c'est trop gros si on étudie des graphes à 100000 sommets.
- Que dire de plus sinon que des méthodes identiques ont été recopiées dans les classes les plus basses car il fallait instancier un graphe à l'intérieur et que c'est infaisable dans une classe abstraite.

4.4 Les états

Pour l'instant SIM contient trois états, état entier (`StateInt`), état entier long (`StateLong`), et état réel (`StateFloat`) implémentant tous l'interface `State`.
Il existe donc trois ensembles d'états correspondants aux trois états ci-dessus. Un ensemble fini d'états représenté par un tableau d'états (`FiniteStateSet`),

un ensemble dénombrable, et un ensemble d'états réels.
Chaque fonction de transition s'applique à un type d'état
particulier. De la même façon qu'il est possible de dfinir de
nouvelles fonctions de transitions, comme nous le verrons plus loin, il est possible de définir de
nouveaux états : il suffit qu'ils implémentent l'interface *State*.
Chaque classe pouvant servir d'état va de paire avec une classe
implémentant *StateSet* qui sert à décrire l'ensemble des
états correspondant. Son usage est laissé à la discrétion du
programmeur de la fonction de transition qui l'utilise, elle peut par
exemple dans le cas d'un nombre d'état fini contenir une
référence sur chaque état (et permettre ainsi une économie de mémoire).

4.5 Configuration des états des sommets

Il existe dans la classe `SetConfiguration` des méthodes pour affecter des états aux sommets, de façon aléatoire (espérance pour chaque état 1 / (nombre d'états)), ou selon une distribution générée aléatoirement (une probabilité est affectée à chaque état, dans le cas d'un ensemble fini seulement).

4.6 Les fonctions de transition et l'exécution

Les fonctions de transition sont les fonctions que l'on applique à un
sommet pour avoir son état après la transition. Elles doivent
implémenter l'interface *FunctionTransition* et être
placées dans le répertoire *sim/apply/function*. Toute classe
vérifiant ceci peut être choisi comme fonction de transition et
être selectionnée dans le menu correspondant de l'interface graphique.
L'execution proprement dite se déroule en appliquant la fonction de
transition selectionnée sur les sommets du graphe : soit à tous les
sommets dans le cas d'une execution synchrone, soit à certains
sommets tirés aléatoirement lors d'une execution asynchrone
(chaque sommet a une certaine probabilité d'effectuer sa transition).

4.7 Mesures

Une mesure doit implémenter l'interface `Measure`. L'utilisateur peut donc définir autant de mesures qu'il veut, tant qu'elles implémentent cette interface. Notamment il doit gérer lui-même l'affichage des résultats de la mesure, (voir la javadoc pour plus de détails).
Dans le cas des mesures effectuées à chaque transition, on peut se rendre compte que le terme de "Mesure" n'est pas approprié, car il est tout à fait possible de créer une mesure qui ne mesure rien du tout, tant qu'elle respecte l'interface. On peut par exemple créer une mesure qui change le graphe à chaque transition (voir DynamicChangeGraph).

4.8 Sauvegarde

Le graphe et la configuration (distribution des états sur le graphe) peuvent être sauvegardés séparement. Lorsque l'on sauvegarde le graphe, les états ne sont pas enregistrés et sont indéfinis lorsqu'on le charge. Une configuration sauvée contient l'ensemble d'état (le "StateSet") et la liste des états du graphe. Tout graphe possédant le même nombre de sommet que la taille de la configuration peut alors être placé dans ces états. Ceci permet de tester une configuration avec plusieurs fonctions de transition différentes, un graphe avec plusieurs configurations différentes...

5 Interface Graphique et Utilisation

L'interface graphique est définie par une classe majeure `MainFrame`, fenêtre d'affichage de base.
Lorsque SIM est lancé en mode graphique sans arguments, la configuration est automatiquement initialisée avec la fonction d'initialisation `StandartInitFunction`, afin de pouvoir lancer une simulation rapidement. Il suffit alors de choisir des mesures et d'appuyer sur START pour lancer la simulation. Lorsque un argument fonction d'initialisation est précisé au lancement de SIM, c'est avec cette fonction que la configuration est initialisée.
Le bouton RESET permet d'appeler une nouvelle fois la fonction d'initialisation si l'utilisateur clique deux fois dessus (`StandartInitFunction` ou la fonction d'initialisation passée en argument de SIM), dans ce cas les mesures sélectionnées pour la simulation précédente sont gardées pour la nouvelle simulation.
A tout instant il est possible d'arrêter la simulation, en appuyant sur BREAK, de faire des mesures en appuyant sur MAKE MEASURES, d'en ajouter par ADD MEASURES, d'effectuer une seule transition en appuyant sur ONE TRANSITION, et de reprendre la simulation en appuyant de nouveau sur START.
Ceci est rendu possible car la simulation est propulsé par un thread. Les mesures sélectionnées au début, ou ajoutées, sont effectuées à chaque transition. Il est aussi possible de voir la progression de la simulation en appuyant sur SEE_PROGRESS.
Au centre de la fenêtre, il y a deux panneaux juxtaposés, l'un contient l'affichage des résultats des mesures, si le mode SEE IN FRAME n'est pas sélectionné (sinon dans l'autre cas les résultats des mesures sont affichées dans des fenêtres indépendantes), l'autre contient des informations concernant la configuration (nombre de sommets, nombre d'arêtes, nom de la fonction de transition, etc...), ce dernier panneau est actualisé toutes les secondes par un thread.

Enfin quand l'utilisateur n'appuie qu'une seule fois sur RESET, la simulation en cours s'arrête, et les boutons de la bordure de gauche s'activent, dans l'ordre

de sélection des éléments. D'abord le graphe, ensuite l'ensemble d'états, etc...Ces boutons créent de nouvelles fenêtres de sélection de éléments de la Configuration définies dans le paquetage sim.view. et ayant tous un nom explicite commençant par Select... ou Set... . Comme il est laissé à l'utilisateur le droit de redéfinir de nouvelles fonctions de transition et de mesures directement en java, le logiciel intègre dynamiquement les nouvelles fonctions implémentants les interfaces requisent. Nul besoin de recompiler tout le logiciel, il suffit simplement de compiler les nouvelles fonctions. Il est aussi possible d'ouvrir des nouvelles fenêtres principales, et d'effectuer plusieurs simulations en même temps, ainsi que de sauver, de charger, la configuration, ou simplement le graphe.

6 Définir un protocole en mode Console

Pour définir un protocole il faut définir des fonctions d'initialisation et les passer en arguments de SIM à son lancement, précédés de l'option `-protocol`. Le logiciel vérifie automatiquement qu'elle aient le bon type. Les fonctions d'initialisation, représentant des simulations, sont exécutées les unes à la suite des autres, et le compt-rendu du protocol est stocké dans un fichier. Les résultats des mesures dans ce cas sont stockés eux aussi dans le fichier de compte-rendu, et leur affichage est géré par l'utilisateur. Ce mode est conçu pour pouvoir réaliser des simulations avec des graphes dont la génération prend beaucoup de temps. Dans une fonction d'initialisation, l'utilisateur doit définir les mesures "statiques" qui seront effectuées au début et à la fin des simulations, ainsi que les mesures qui seront effectuées à chaque transition.

7 Définir mesures, fonctions de transition, et fonctions d'initialisation

7.1 Mesure

Définir une implémentation de la classe `Measure` en oubliant pas de lever les exceptions nécessaires, et la placée dans sim/measure, et la compiler, avec ./make.sh measure, ou directement avec javac.
Il est possible d'utiliser le modèle des mesures qui existent déjà.
Se référer à la javadoc pour plus de renseignements.

7.2 Fonction de transition

Définir une implémentation de la classe `FunctionTransition` en oubliant pas de lever les exceptions nécessaires, et la placée dans sim/apply/function, et la compiler, avec ./make.sh function, ou directement avec javac.
Il est possible d'utiliser le modèle des mesures qui existent déjà.
Se référer à la javadoc pour plus de renseignements.

7.3 Fonction d'initialisation

Définir une implémentation de la classe `InitFunction` en oubliant pas de lever les exceptions nécessaires, et la placée dans sim/protocol/init, et la compiler, avec ./make.sh init, ou directement avec javac.
Il est possible d'utiliser le modèle des mesures qui existent déjà.
Se référer à la javadoc pour plus de renseignements.

8 Améliorations Possibles

- fournir des outils plus conviviaux pour la visualisation des mesures
- interfacer SIM avec un logiciel de visualisation de graphes
- gérer la sauvegarde des mesures, notamment pour le protocole (afficher les résultats de toutes les mesures en mode texte dans le même fichier peut parfois être assez illisible).
- permettre de recommencer une simulation en gardant simplement le graphe de la précédente

9 Contact et Renseignements Complémentaires

lbeadou@ens-lyon.fr
educhesn@ens-lyon.fr
efiliot@ens-lyon.fr

La javadoc sur http ://www.ens-lyon.fr/ efiliot
Sur chaque page de documentation de la javadoc ont été ajoutés les emails des auteurs des classes, il faut donc les contacter en priorité pour un problème particulier.