Implantation d'algorithmes autostabilisants dans *Sinalgo*

Alain Cournier Stéphane Devismes

Université de Picardie Jules Verne

8 juin 2023

Plan

Roadmap

Principe

Principe

1. Toutes les variables sont initialisées avec des valeurs choisies au hasard dans leurs domaines de définition.

Principe

1. Toutes les variables sont initialisées avec des valeurs choisies au hasard dans leurs domaines de définition.
2. Régulièrement, chaque processus envoie son état courant à tous ses voisins.

Principe

1. Toutes les variables sont initialisées avec des valeurs choisies au hasard dans leurs domaines de définition.
2. Régulièrement, chaque processus envoie son état courant à tous ses voisins.
3. Chaque processus stocke le dernier état connu de chacun de ses voisins.
   - Sur réception d'un message contenant l'état du voisin v : mise à jour du dernier état connu de v.
   - (l'espace mémoire réservé pour stocker les états des voisins est lui aussi initialisé au hasard en fonction des domaines de définition)

Principe

1. Toutes les variables sont initialisées avec des valeurs choisies au hasard dans leurs domaines de définition.
2. Régulièrement, chaque processus envoie son état courant à tous ses voisins.
3. Chaque processus stocke le dernier état connu de chacun de ses voisins.
   - Sur réception d'un message contenant l'état du voisin v : mise à jour du dernier état connu de v.
   - (l'espace mémoire réservé pour stocker les états des voisins est lui aussi initialisé au hasard en fonction des domaines de définition)
4. Régulièrement, le processus inspecte son état et les derniers états connus de chacun de ses voisins pour voir si une règle s'applique.

Principe

1. Toutes les variables sont initialisées avec des valeurs choisies au hasard dans leurs domaines de définition.
2. Régulièrement, chaque processus envoie son état courant à tous ses voisins.
3. Chaque processus stocke le dernier état connu de chacun de ses voisins.
   - Sur réception d'un message contenant l'état du voisin v : mise à jour du dernier état connu de v.
   - (l'espace mémoire réservé pour stocker les états des voisins est lui aussi initialisé au hasard en fonction des domaines de définition)
4. Régulièrement, le processus inspecte son état et les derniers états connus de chacun de ses voisins pour voir si une règle s'applique.
   - Le cas échéant, le processus applique la règle activable.

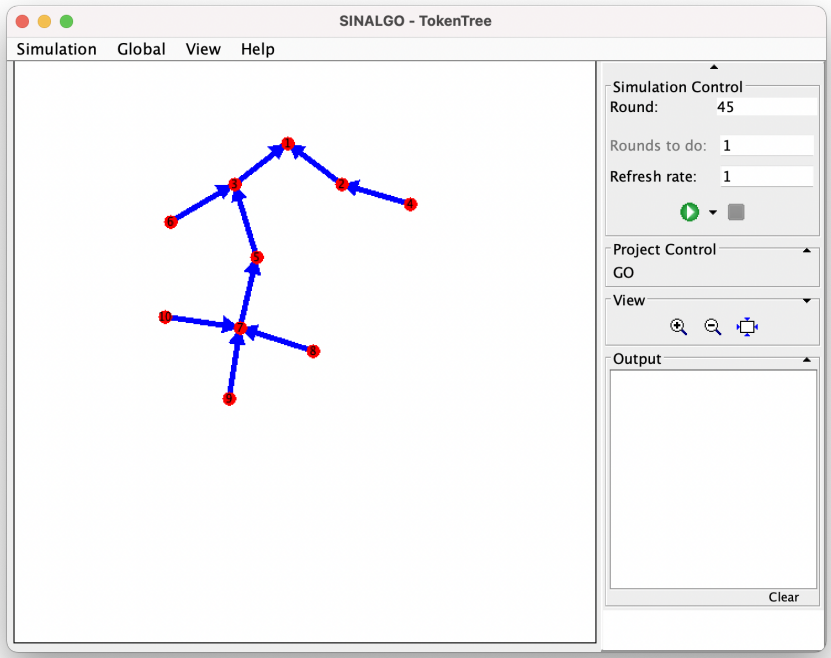
Le simulateur *Sinalgo*

(Rappel)

Ce simulateur évènementiel est en fait un plug-in `Java` pour `Eclipse` développé par le Distributed Computing Group à l'ETH Zurich (Suisse).

Il permet, en autre, de :

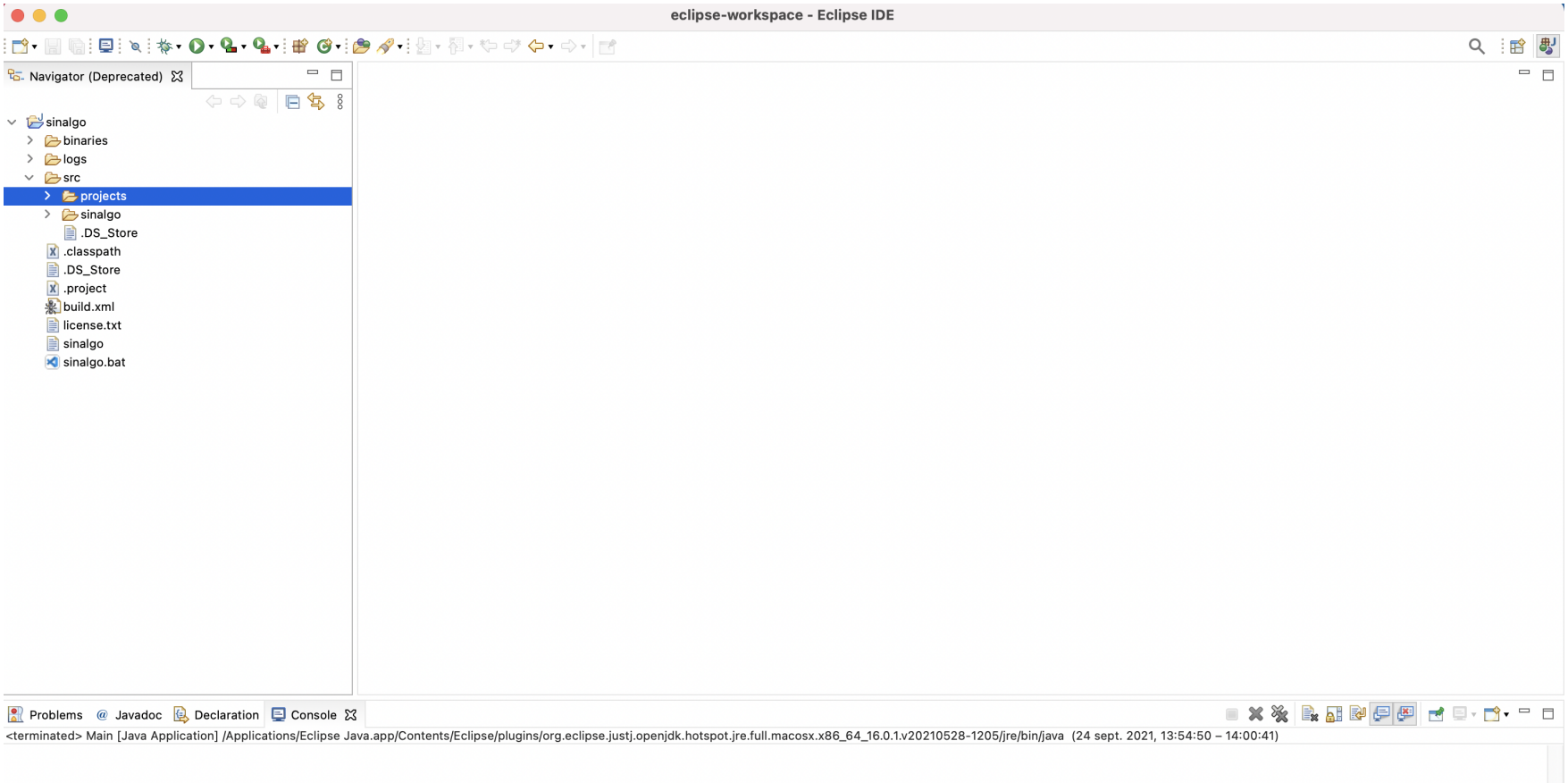
1. visualiser une exécution d'un algorithme distribué dans un environment choisi (topologie, asynchronie, fiabilité des liens, ...)
2. faire des simulations par lot (batch simulation) pour obtenir des évaluations de performance empiriques



Nous nous intéresserons ici qu'au 1er point.

Structure général du simulateur

(Rappel)

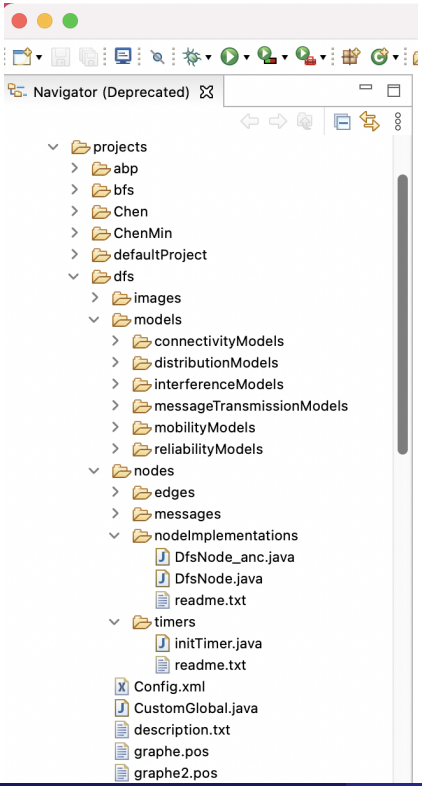


Fichiers principaux

(Rappel)

Un algorithme = un répertoire dans `projects`

(copie du répertoire `template`)

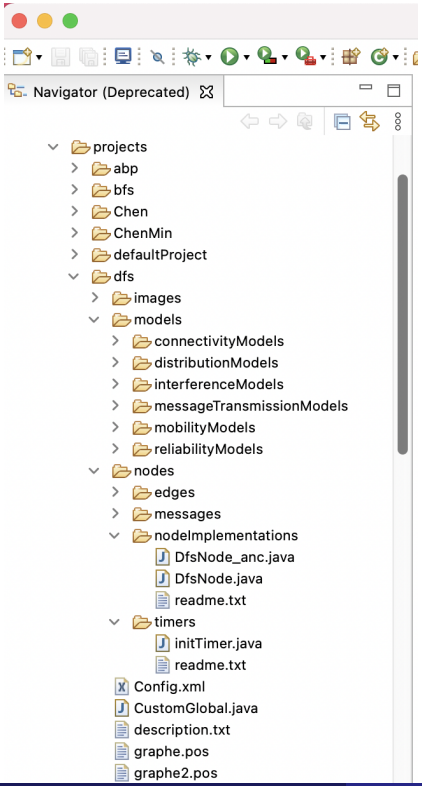


Fichiers principaux

(Rappel)

Un algorithme = un répertoire dans `projects`

(copie du répertoire `template`)



Contenu :

1. `CustomGlobal.java` : accès à la configuration globale (utilité : détection de terminaison)
2. `description.txt` : description de l'algorithme qui sera affichée dans la fenêtre « menu »

   Pour les TPs, la description donnera la méthode de génération du réseau
3. `Config.xml` : paramètres d'environnement de simulation (en particulier les hypothèses)
4. Répertoires : `images, models, nodes`

Exemple : MIS autostabilisant

On crée un répertoire `SSMIS` copie du répertoire `template` dans le répertoire `projects`

Puis, on remplit le fichier `Config.xml`

Roadmap

Config.xml

Champs principaux (1/7)

Le réseau est déployé sur un plan rectangulaire 2D

```
<dimensions value="2" />
```

Taille du rectangle

```
<dimX value="1000" />

<dimY value="1000" />
```

Config.xml

Champs principaux (2/7)

Processus asynchrone ? Pour simplifier non (seuls les liens le seront)

```
<asynchronousMode value="false" />
```

Processus mobile ? non

```
<mobility value="false" />
```

Pas d'interférence sur la ligne

```
<interference value="false" />
```

Liens bidirectionnels

```
<edgeType
     value="sinalgo.nodes.edges.BidirectionalEdge" />
```

Les liens de communications sont créés au démarrage de la simulation

```
<initializeConnectionsOnStartup value="true" />
```

Config.xml

Champs principaux (3/7)

Transmission FIFO asynchrone :

```
<DefaultMessageTransmissionModel value="FifoRandom3" />
```

Il faut ajouter dans la balise dans <Custom>

```
  <FifoRandom distribution="Uniform" min="1" max="10" />
```

Un message mets en moyenne 5 unités de temps à arriver

Config.xml

Champs principaux (4/7)

Deux processus sont voisins s'ils sont à moins d'une certaine distance (portée) :

```
<DefaultConnectivityModel value="UDG" />
```

Il faut ajouter dans deux balises dans <Custom> pour spécifier la portée

```
<GeometricNodeCollection rMax="150"/>
<UDG rMax="150"/>
```

Portée 150 pixels

Config.xml

Champs principaux (5/7)

Topologie :

```
<DefaultDistributionModel value="PositionFile" />
```

Ici, issue d'un fichier (sinon, Circle, Line2D, Grid2D ou Random)

Pas d'interférence et pas de mobilité

```
<DefaultInterferenceModel value="NoInterference" />
<DefaultMobilityModel value="NoMobility" />
```

Config.xml

Champs principaux (6/7)

Fiabilité des liens :

```
<DefaultReliabilityModel value="LossyDelivery"/>
```

Ici, les liens ne sont pas fiables (pertes de message)

Il faut ajouter dans la balise dans <Custom>

```
<LossyDelivery DropRate="0.8" />
```

Ici, 80 % des messages sont perdus !

Config.xml

Champs principaux (7/7)

Où trouver le code des processus :

```
<DefaultNodeImplementation value="SSMIS:MISNode" />
```

Ici dans le fichier `MISNode.java` du répertoire `SSMIS`

Dans `<Custom>`, taille minimum des processus à affichage :

```
<Node defaultSize="30" />
```

Roadmap

Algorithme distribué

Messages + Code des processus

Messages :

1. Fichier `nom_message.java` dans le répertoire `nodes/messages`
2. `nom_message.java` contient une classe (publique) `nom_message` qui dérive de `Msg`

Code :

1. Fichier `nom_algo.java` dans le répertoire `nodes/nodeImplementations`
2. `nom_algo.java` contient une classe (publique) `nom_algo` qui dérive de `TimerNode.java` (tous les nœuds sont initiateurs !)
3. `TimerNode.java` dérive de `BasicNode`

Principales méthodes de `TimerNode`

Structure de l'algorithme

Méthode	**Description**
`void initialization()`	doit contenir l'initialisation des variables
`void regularly()`	doit contenir le code de l'algorithme à exécuter régulièrement
`void receipt (LinkedList<Msg> inbox)`	réception des messages les messages sont stockés dans la file `inbox`

Principales méthodes de `TimerNode`

Passage de messages

Méthode	**Description**
`void send(Msg m, int i)`	envoie le message `m` au voisin incident du canal de numéro `i` ATTENTION : la numérotation commence à 0 !
`void broadcast(Msg m)`	envoie le message `m` à tous les voisins
`int from(Msg m)`	retourne le numéro de canal par lequel est arrivé le message `m`
`void sendSuccessor(Msg m)`	envoie le message `m` au successeur dans l'anneau À utiliser uniquement avec une topologie `Circle`

Principales méthodes de `TimerNode`

Connaissances du réseau

Méthode/Attribut	**Description**
`int this.ID`	identifiant unique du processus
`int nbProcesses()`	retourne le nombre total de processus
	Méthode statique
`int degree()`	retourne le nombre de voisins du processus
`int degreeMax()`	retourne le degré du réseau
	Méthode statique
`boolean root()`	retourne vrai SSI le processus est la racine
`String toString()`	informations de bases sur le processus
	(identité et liste des identités des voisins)
	(utile pour l'affichage dans la console lors d'un debuggage)

Principales méthodes de `TimerNode`

Sorties

Méthode	**Description**
`void stop(String S)`	stoppe la simulation et affiche la chaîne `S` dans la console Méthode statique
`void affiche(String S)`	affiche la chaîne `S` dans la console
`void clearConsole()`	supprime les messages affichés dans la console Méthode statique

Principales méthodes de `TimerNode`

Divers

Méthode	**Description**
`nextInt(int b)`	retourne un entier au hasard entre 0 et `b`−1
`void draw(Graphics g, PositionTransformation pt, boolean highlight)`	représentation du processus en mode graphique
`int getIndex(Node x)`	retourne le numéro du canal dont le voisin `x` est incident (`x` est une référence) retourne -1 si `x` n'est pas un voisin
`Node getNeighbor(int indice)`	retourne la référence du voisin incident du canal de numéro `indice` si l'indice n'existe pas, retourne la référence processus (`this`)

Les deux dernières méthodes ne doivent pas être utilisées dans le code de l'algorithme !

Roadmap

Calcul Silencieux d'un Ensemble Indépendant Maximal

Code de l'algorithme pour le processus p

Constantes :

- $\mathcal{N}_p$, ensemble des voisins
- id_p, identité de p

Variables : $S_p \in \{Dominant, dominé\}$

Actions :

$$Leave \;::\; S_p = Dominant \wedge (\exists q \in \mathcal{N}_p, S_q = Dominant \wedge id_q < id_p) \mapsto S_p \leftarrow dominé$$

$$Join \;::\; S_p = dominé \wedge (\forall q \in \mathcal{N}_p, S_q = dominé \vee id_q > id_p) \mapsto S_p \leftarrow Dominant$$

TypeS.java dans SSMIS/nodes/nodeImplementations

```
package projects.SSMIS.nodes.nodeImplementations;

// Nouveau type pour coller à l'algorithme
public enum TypeS {
    Dominant,domine;

    public static TypeS RandomInit(){
        return ((int) (Math.random() * 10000.0))%2 == 1 ?
                Dominant:domine;
        }
}
```

EtatVoisin.java dans SSMIS/nodes/nodeImplementations

```
package projects.SSMIS.nodes.nodeImplementations;

import projects.defaultProject.nodes.nodeImplementations.TimerNode;

public class EtatVoisin{
   public int ID;
   public TypeS S;

   public EtatVoisin(){
     this.ID=(int) (Math.random()*100.0*TimerNode.nbProcesses())
                                          %TimerNode.nbProcesses();
     this.S=TypeS.RandomInit();
   }

   public void Set(TypeS S,int ID) {
      this.S=S;
      this.ID=ID;
   }
}
```

Data.java dans SSMIS/nodes/messages

```
package projects.SSMIS.nodes.messages;

import projects.SSMIS.nodes.nodeImplementations.TypeS;
import projects.defaultProject.nodes.messages.Msg;

public class Data extends Msg {
    public TypeS S;
    public int ID;

    public Data(TypeS S,int ID){
        this.S=S;
        this.ID=ID;
    }

    public Data clone() {
        return new Data(this.S,this.ID);
    }
}
```

MISNode.java dans SSMIS/nodes/nodeImplementations

Import

```
package projects.SSMIS.nodes.nodeImplementations;

import java.awt.Color;
import java.awt.Graphics;
import java.util.LinkedList;
import projects.SSMIS.nodes.messages.Data;
import projects.defaultProject.nodes.messages.Msg;
import projects.defaultProject.nodes.nodeImplementations.TimerNode;
import sinalgo.gui.transformation.PositionTransformation;
```

MISNode.java dans SSMIS/nodes/nodeImplementations

Les variables

```
public class MISNode extends TimerNode {

        // état du processus
        public TypeS S;

        // pour stocker le dernier état connu de chaque voisin
        public EtatVoisin[] Voisins;

        ...
}
```

MISNode.java dans SSMIS/nodes/nodeImplementations

fonctions obligatoires

```
public class MISNode extends TimerNode {
    // initialisation (au hasard) des variables
    public void initialization() {
      ... }

    // code à exécuter régulièrement
    public void regularly() {
      ... }

    // réception de messages
    public void receipt(LinkedList<Msg> inbox) {
      ... }

    // affichage du noeud
    public void draw(Graphics g,
    PositionTransformation pt, boolean highlight){
      ... }
}
```

MISNode.java dans SSMIS/nodes/nodeImplementations

Fonctions dédiées (1/2)

```
public class MISNode extends TimerNode {
    ...

    public boolean Dominant() {
        for(int i=0;i<this.degree();i++)
            if(Voisins[i].S==TypeS.Dominant && Voisins[i].ID<this.ID)
                return false;
        return true;
    }

    public boolean Join() {
        return this.S==TypeS.domine && Dominant(); }

    public boolean Leave() {
        return this.S==TypeS.Dominant && !Dominant(); }

    ...
}
```

MISNode.java dans SSMIS/nodes/nodeImplementations

Fonctions dédiées (2/2)

```
public class MISNode extends TimerNode {
    ...

   public void Algo() {
      if(Join()) this.S=TypeS.Dominant;
      else
         if(Leave()) this.S=TypeS.domine;
   }

   Color couleur() {
      if (this.S==TypeS.Dominant) return Color.blue;
      return Color.yellow;
   }

      ...
}
```

MISNode.java dans SSMIS/nodes/nodeImplementations

Détail des fonctions obligatoires : initialisation

```
public void initialization() {
    this.S=TypeS.RandomInit();

    if(this.degree()<1) return;

    Voisins=new EtatVoisin[this.degree()];

    for(int i=0;i<this.degree();i++)
        Voisins[i]=new EtatVoisin();
}
```

MISNode.java dans SSMIS/nodes/nodeImplementations

Détail des fonctions obligatoires : code à exécuter régulièrement

```
public void regularly() {
    Algo();
    this.broadcast(new Data(this.S,this.ID));
}
```

MISNode.java dans SSMIS/nodes/nodeImplementations

Détail des fonctions obligatoires : réception de messages

```
public void receipt(LinkedList<Msg> inbox) {
      for(Msg m: inbox) {
         Data donnee=(Data) m;
         int i=this.from(donnee);
         Voisins[i].Set(donnee.S,donnee.ID);
      }
}
```

MISNode.java dans SSMIS/nodes/nodeImplementations

Détail des fonctions obligatoires : affichage du processus

```
public void draw(Graphics g, PositionTransformation pt,
     boolean highlight){

     String text="";
     if(this.ID>0)    text= ""+this.ID+"";
     this.setColor(couleur());
     super.drawNodeAsDiskWithText(g, pt, false, text,
                30, Color.black);

  }
```

MISNode.java dans SSMIS/nodes/nodeImplementations

Gestion de la dynamicité

```
public void neighborhoodChange() {
    this.initialization();
}
```

CustomGlobal.java dans SSMIS/

Packages

```
package projects.SSMIS;

import javax.swing.JOptionPane;
import projects.SSMIS.nodes.messages.Data;
import projects.SSMIS.nodes.nodeImplementations.MISNode;
import projects.SSMIS.nodes.nodeImplementations.TypeS;
import projects.defaultProject.nodes.nodeImplementations.
       TimerNode;
import projects.defaultProject.models.messageTransmissionModels.
       FifoRandom3;
import projects.defaultProject.models.messageTransmissionModels.
       Packet;
import sinalgo.nodes.Node;
import sinalgo.runtime.AbstractCustomGlobal;
import sinalgo.tools.Tools;
```

CustomGlobal.java dans SSMIS/ : hasTerminated()

```
public boolean hasTerminated() {
    TimerNode.clearConsole();
    for (Node b : Tools.getNodeList()){
        MISNode n=(MISNode) b;
        if(n.Dominant()!=(n.S==TypeS.Dominant)) return false;
        for(int i=0;i<n.degree();i++) {
          if(n.Voisins[i].S!=((MISNode) n.getNeighbor(i)).S)
                return false;
          if(n.Voisins[i].ID!=((MISNode) n.getNeighbor(i)).ID)
                return false;
    }}
    for(Packet p: ((FifoRandom3)
        Tools.getMessageTransmissionModel()).OnGoingMessages()) {
        MISNode n=(MISNode) p.getSender();
        Data m=(Data) p.getMessage();
        if(n.S != m.S || n.ID != m.ID) return false;
    }
     TimerNode.stop("Système␣stabilisé␣!");
    return false; }
```

Changer l'affichage des arêtes

Voir le cours de l'an dernier !

À vous de jouer !