Implantation d'algorithmes distribués dans *Sinalgo*

Alain Cournier et Stéphane Devismes

Université de Picardie Jules Verne

9 juin 2026

Plan

Plan

Le simulateur *Sinalgo*

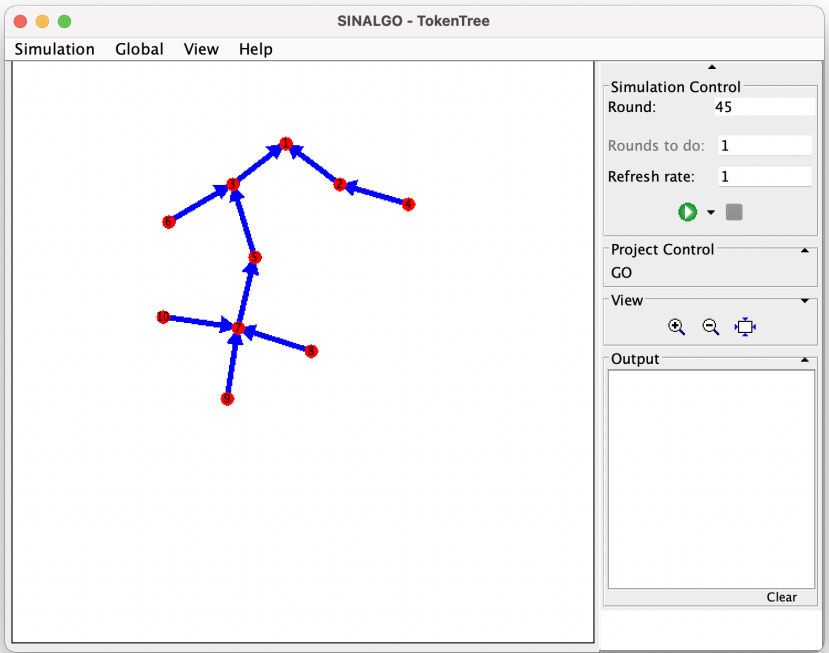
Ce simulateur évènementiel est en fait un plug-in `Java` pour `Eclipse` développé par le Distributed Computing Group à l'ETH Zurich (Suisse).

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 premier point.



Plan

Circulation de jeton (simple) dans un arbre

La spécification (rappel)

Sûreté :

1. Il existe au plus un jeton dans le réseau.
2. Au plus une décision est prise.
3. Si une décision est prise, alors tous les processus ont été visités par le jeton.

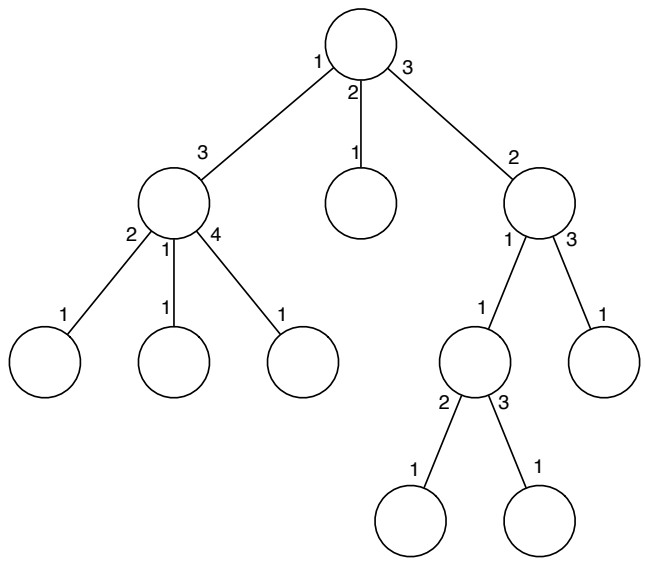
Vivacité :

1. L'exécution termine.
2. L'initiateur finit par décider.

Circulation de jeton (simple) dans un arbre

Les hypothèses (rappel)

- Processus et canaux asynchrones.
- Canaux étiquetés de 1 à δ_p pour tout processus p.
- Pas de faute.
- Topologie en arbre avec au moins deux nœuds.
- Mono-initiateur.



Circulation de jeton (simple) dans un arbre

L'algorithme

Variables

1: *Initiateur* booléen initialisé à *faux*

Spontanément */* Démarrage de l'algorithme */*

/ exécuté en 1er par l'initiateur uniquement */*

2: *Initiateur* $\leftarrow$ *true*

3: `Envoyer` $\langle$*Jeton*$\rangle$ `à` 1

Réception de $\langle$*Jeton*$\rangle$ **de** q

4: **Si** *Initiateur* $\wedge$ $q \geq \delta_p$ **alors**

5: `decide`

6: **Sinon**

7: `Envoyer` $\langle$*Jeton*$\rangle$ `à` $(q \bmod \delta_p) + 1$

8: **Fin Si**

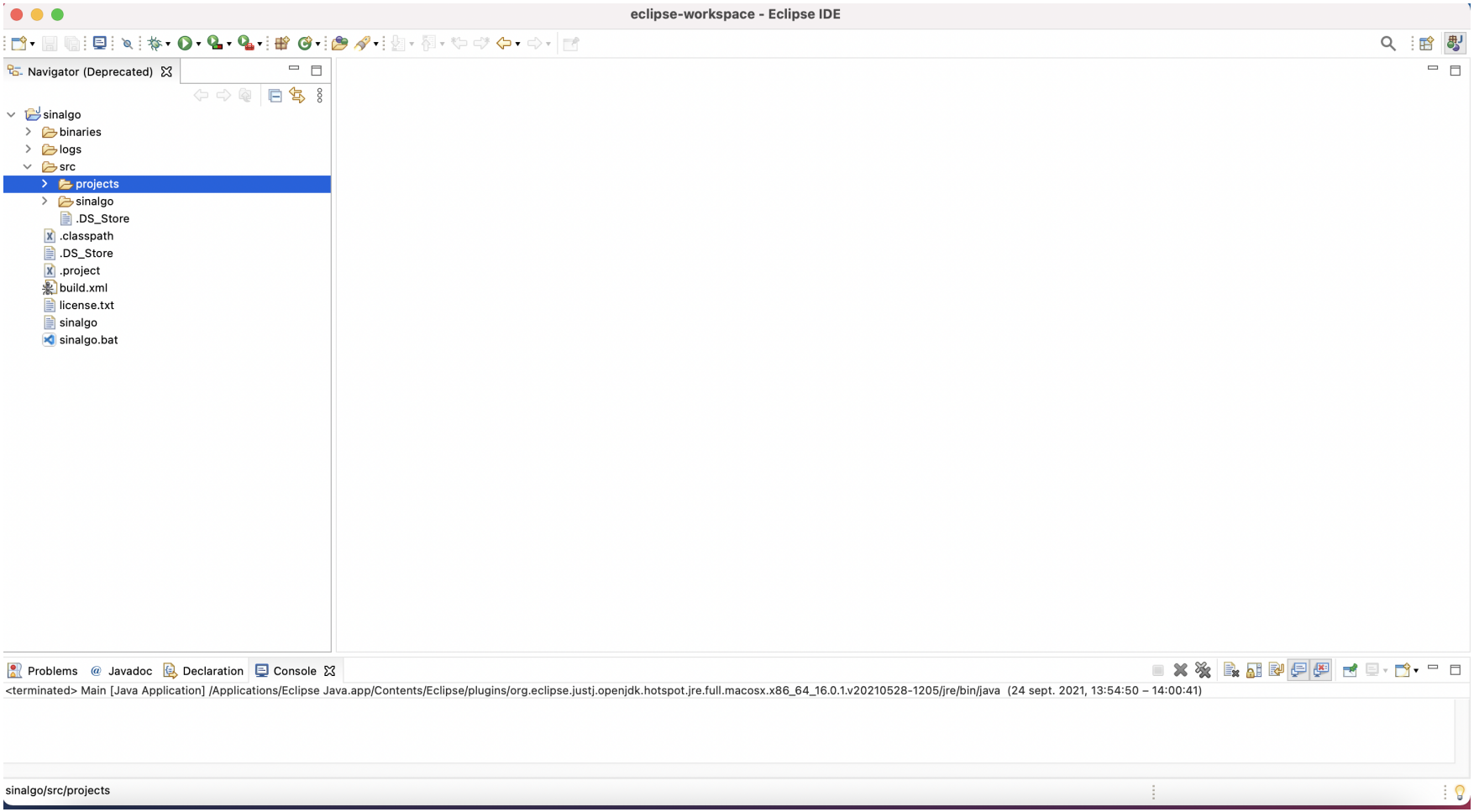
Plan

Ressources sur Moodle

`https://pedag.u-picardie.fr/moodle/upjv/course/view.php?id=8713`

1. Sources *Sinalgo*
2. Tutoriel d'installation
3. VMs

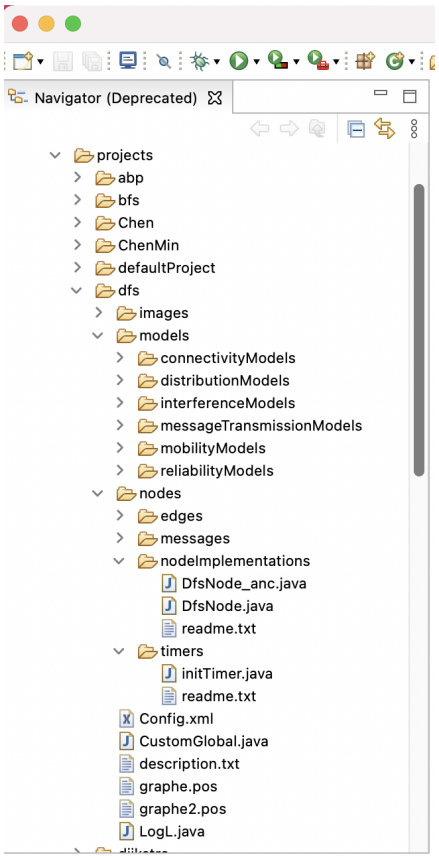
Structure général du simulateur



Fichiers principaux

Un algorithme = un répertoire dans `projects`

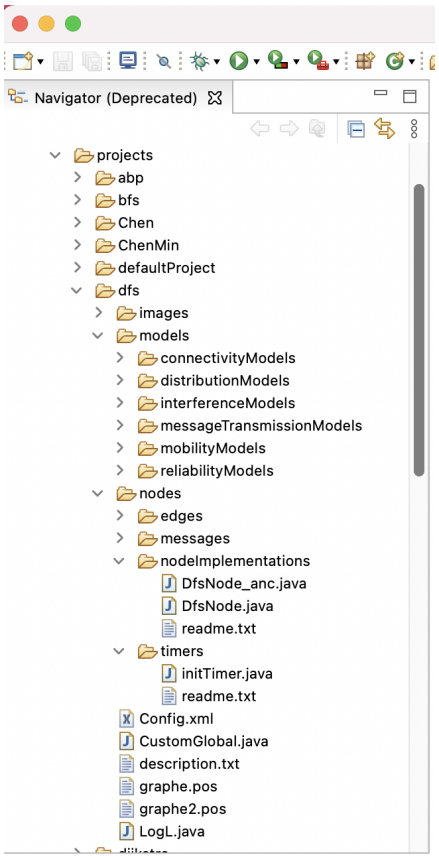
(copie du répertoire `template`)



Fichiers principaux

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 : circulation de jeton dans l'arbre

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

Puis, on remplit le fichier `Config.xml`

Plan

`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="FifoRandom2" />
```

Il faut ajouter dans la balise dans `<Custom>`

```
  <FifoRandom distribution="Uniform" min="1" max="20" />
```

Un message mets en moyenne 10 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 `<Custum>` 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`)

Pas d'interférence et pas de mobilité

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

`Config.xml` : champs principaux (6/7)

Fiabilité des liens :

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

Ici, les liens sont fiables (pas de perte)

`Config.xml` : champs principaux (7/7)

Où trouver le code des processus :

```
<DefaultNodeImplementation value="TokenTree:TokenTreeNode" />
```

Ici dans le fichier `TokenTreeNode.java` du répertoire `TokenTree`

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

```
<Node defaultSize="20" />
```

Plan

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 `MonoInitiator` ou `MultiInitiator`, selon le nombre d'initiateurs 
3. `MonoInitiator` et `MultiInitiator` dérivent de

`BasicNode`

Principales méthodes de `BasicNode`

Structure de l'algorithme

Méthode	**Description**
`void initialization()`	doit contenir l'initialisation des variables
`void spontaneously()`	doit contenir le code de démarrage de l'algorithme
`void receipt` `(LinkedList<Msg> inbox)`	réception des messages les messages sont stockés dans la file `inbox` ATTENTION : appelée régulièrement, même si aucun message n'a été reçu le cas échéant, `inbox` est vide

Principales méthodes de `BasicNode`

Passage de messages

Méthode	**Description**
`void send(Msg m, int i)`	envoie le message `m` au voisin incident du canal de numéro `i`
`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 `BasicNode`

Connaissances du réseau

Méthode/Attribut	**Description**
`int this.ID`	identifiant unique du processus
`int nbProcesses()`	retourne le nombre total de processus
`int degree()`	retourne le nombre de voisins du processus
`boolean initiator()`	retourne vrai SSI le processus est initiateur
`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 `BasicNode`

Divers

Méthode	**Description**
`void decide()`	décision, stoppe la simulation
`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 reférence) retourne 0 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 !

Plan

Un seul type de message : Jeton

Jeton.java dans TokenTree/nodes/messages

```
package projects.TokenTree.nodes.messages;

import projects.defaultProject.nodes.messages.Msg;

public class Jeton extends Msg {

        public Jeton(){
        }

        public Msg clone() {
                return new Jeton();
        }
}
```

Code des processus : `TokenTreeNode.java` dans `TokenTree/nodes/nodeImplementations`

Imports

```
package projects.TokenTree.nodes.nodeImplementations;

import java.awt.Color;
import java.awt.Graphics;
import java.util.LinkedList;

import projects.TokenTree.nodes.messages.Jeton;
import projects.defaultProject.nodes.messages.Msg;
import projects.defaultProject.nodes.
       nodeImplementations.MonoInitiator;

import sinalgo.gui.transformation.PositionTransformation;
```

Code des processus : `TokenTreeNode.java` dans `TokenTree/nodes/nodeImplementations`

Méthodes obligatoires

```
public class TokenTreeNode extends MonoInitiator{

        public void initialization() {
        }

        public void spontaneously() {
        }

        public void receipt(LinkedList<Msg> inbox) {
        }

        // Dessin du processus

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

Code des processus : `TokenTreeNode.java` dans `TokenTree/nodes/nodeImplementations`

Variables du processus et leur initialisation

```
// variable utile pour l'affichage uniquement

public boolean jeton;

public void initialization() {
  // pour l'affiche des processus,
  // on distingue si le processus a déjà reçu le jeton
  jeton=false;
}
```

Code des processus : `TokenTreeNode.java` dans `TokenTree/nodes/nodeImplementations`

Démarrage

```
public void spontaneously() {
  this.send(new Jeton(),1);
  this.jeton=true;
}
```

Code des processus : `TokenTreeNode.java` dans `TokenTree/nodes/nodeImplementations`

Réception des messages

```
public void receipt(LinkedList<Msg> inbox) {
 for(Msg m: inbox) {
       if(m instanceof Jeton) {
        // Inutile dans ce cas,
        // mais utile lorsqu'il y a plusieurs types de message
       Jeton msg = (Jeton) m;
       if(this.initiator() && this.from(msg)==this.degree())
               this.decide();
       else {
             this.jeton=true;
             this.send(new Jeton(),
                   (this.from(msg)%this.degree())+1);
            }
       }
    }
}
```

Code des processus : `TokenTreeNode.java` dans `TokenTree/nodes/nodeImplementations`

Affichage

```
public Color Couleur(){
    if(this.jeton) return Color.red;
        return Color.blue;
}

// Affichage du processus :
// en rouge, puis bleu après qu'il est reçu le jeton
// pour la première fois.
// On affiche aussi l'identité du noeud.

public void draw(Graphics g, PositionTransformation pt,
                             boolean highlight){
 this.setColor(this.Couleur());
 String text = ""+this.ID;
 super.drawNodeAsDiskWithText(g, pt, highlight,
                             text, 20, Color.black);
}
```

Comparaison

Algorithme 1 Circulation de jeton dans un arbre

Variables

1: *Initiateur* booléen initialisé à $faux$

Spontanément /* *Démarrage de l'algorithme* */
/* *exécuté en* 1^{er} *par l'initiateur uniquement* */

2: $Initiateur \leftarrow true$
3: `Envoyer` $\langle Jeton \rangle$ `à` 1

Réception de $\langle Jeton \rangle$ **de** q

4: **Si** $Initiateur \wedge q \geq \delta_p$ **alors**
5: `decide`
6: **Sinon**
7: `Envoyer` $\langle Jeton \rangle$ `à` $(q \bmod \delta_p) + 1$
8: **Fin Si**

```
public void initialization() {}

public void spontaneously() {
 this.send(new Jeton(),1);
}

public void receipt(LinkedList<Msg> inbox) {
  for(Msg m: inbox) {
     if(m instanceof Jeton) {
        Jeton msg = (Jeton) m;
        if(this.initiator() &&
```

Plan

Proposition

Mise en évidence des arêtes visitées

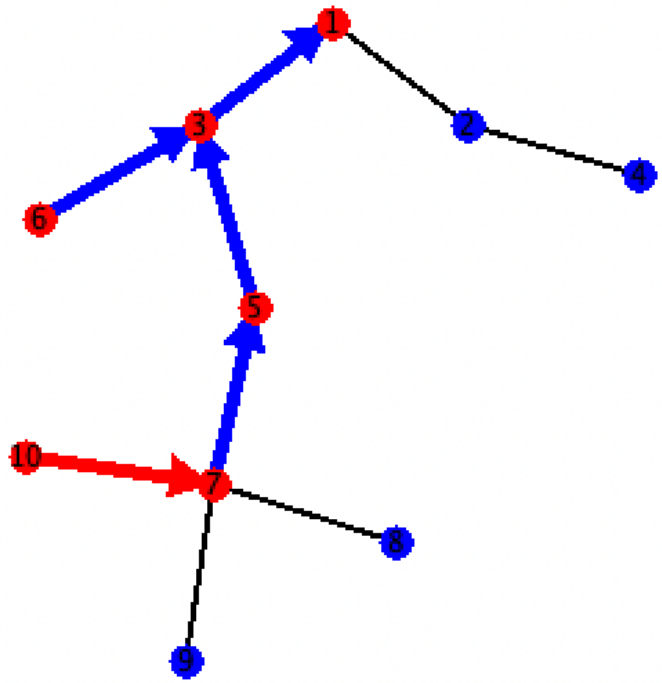
Déjà visitée : flèche vers le ≪ parent ≫

Pas visitée : trait simple

Mise en évidence du lien contenant le jeton

Rouge : contient le jeton

Noir : ne contient pas le jeton



Les flèches

Autoriser et définir la taille des flèches

Modification de `Config.xml`

1. `<drawArrows value="true" />`
2. `<arrowLength value="30" />`
3. `<arrowWidth value="15" />`

Pointeur `parent` dans le code

```
public int parent;
public void initialization() {
        jeton=false;
        parent=0; // pour l'affichage des arêtes
}

public void spontaneously() {
        this.send(new Jeton(),1);
        this.jeton=true;
}

public void receipt(LinkedList<Msg> inbox) {
for(Msg m: inbox) {
        if(m instanceof Jeton) {
        Jeton msg = (Jeton) m;
        if(this.initiator() && this.from(msg)==this.degree())
                    this.decide();
        else {
        this.jeton=true;
        if (this.parent==0 && !this.initiator())
                    this.parent=this.from(msg);
        this.send(new Jeton(),(this.from(msg)%this.degree())+1);
        }
      }
    }
  }
```

Modifier l'apparence des arêtes

Spécialisation `Arete` de la classe `BidirectionalEdge`

1. Fichier `Arete.java` dans le répertoire `nodes/edges`
2. Une classe (publique) `Arete` dans le fichier `Arete.java`
3. `Arete` hérite de `BidirectionalEdge`
4. Modification de `Config.xml` :

```
<edgeType value="TokenTree:Arete" />
```

Arete.java

Imports

```
package projects.TokenTree.nodes.edges;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Stroke;

import projects.TokenTree.nodes.nodeImplementations.TokenTreeNode;

import sinalgo.gui.helper.Arrow;
import sinalgo.gui.transformation.PositionTransformation;
import sinalgo.nodes.Position;
import sinalgo.nodes.edges.BidirectionalEdge;
```

Arete.java

Code de la classe Arete (1/3)

```
public class Arete extends BidirectionalEdge {
public void draw(Graphics g, PositionTransformation pt) {

        // p1 donne les coordonnées d'un processus incident à l'arête
        Position p1 = startNode.getPosition();

        // On se translate en p1
        pt.translateToGUIPosition(p1);

        // Sauvegarde des coordonnées de p1
        int fromX = pt.guiX, fromY = pt.guiY;

        // p2 donne les coordonnées du deuxième processus incident à l'arête
        Position p2 = endNode.getPosition();

        // On se translate en p2
        pt.translateToGUIPosition(p2);

        // On caste en TokenTreeNode pour pouvoir accéder
        // aux variables publiques des deux processus incidents
        TokenTreeNode deb=(TokenTreeNode) this.startNode;
        TokenTreeNode fin=(TokenTreeNode) this.endNode;

        // Par défaut l'arête sera dessinée en rouge
        // (si elle contient des messages en fait)
        Color c=Color.red;

        // g2 est une boite à outils graphique
        Graphics2D g2=(Graphics2D) g;
```

Arete.java

Code de la classe Arete (2/3)

```
// Si le processus deb désigne le processus fin comme parent
// (ATTENTION l'arête est orientée)
if(deb.parent !=0 && (deb.getNeighbor(deb.parent)) == fin){

// si l'arête ne contient pas de message dans les deux sens,
// alors on la colore en bleu.
if(this.numberOfMessagesOnThisEdge==0 &&
     this.oppositeEdge.numberOfMessagesOnThisEdge==0)
        c=Color.blue;

// On grossit le trait
Stroke stroke = new BasicStroke(5f, BasicStroke.CAP_ROUND,
     BasicStroke.JOIN_MITER, 10.0f);
g2.setStroke(stroke);

// Et on affiche une flèche de deb vers fin.
Arrow.drawArrow(fromX, fromY, pt.guiX, pt.guiY, g2, pt,c);
}
```

Arete.java

Code de la classe Arete (3/3)

```
        else {

        // Sinon, on dessine un trait plus fin
        // de deb vers fin uniquement dans le cas où
        // fin ne désigne pas deb comme parent.
        // Si on dessine ce trait et que l'arête ne contient pas de message
        // dans les deux sens, alors l'arête est de couleur noire.

        Stroke stroke = new BasicStroke(2f, BasicStroke.CAP_ROUND,
                BasicStroke.JOIN_MITER, 10.0f);
        g2.setStroke(stroke);

        if(fin.parent != fin.getIndex(deb)) {
          if(this.numberOfMessagesOnThisEdge==0
                && this.oppositeEdge.numberOfMessagesOnThisEdge==0)
                        c=Color.black;
                g2.setColor(c);
                g.drawLine(fromX, fromY, pt.guiX, pt.guiY);
                }
        }
    }
}
```

À vous de jouer !