Présentation JBotSim

Laurent PHILIPPE

Master 2 Informatique
Ingénierie des Systèmes et Logiciels

2022/2023

UNIVERSITé DE FRANCHE-COMTé

femto-st SCIENCES & TECHNOLOGIES

Présentation

- Simulation d'algorithmes distribués
- Bibliothèque qui offre des primitives de base pour prototyper, exécuter et visualiser
- Réseaux dynamiques : nœuds mobiles
- Développé au Labri : https ://jbotsim.io

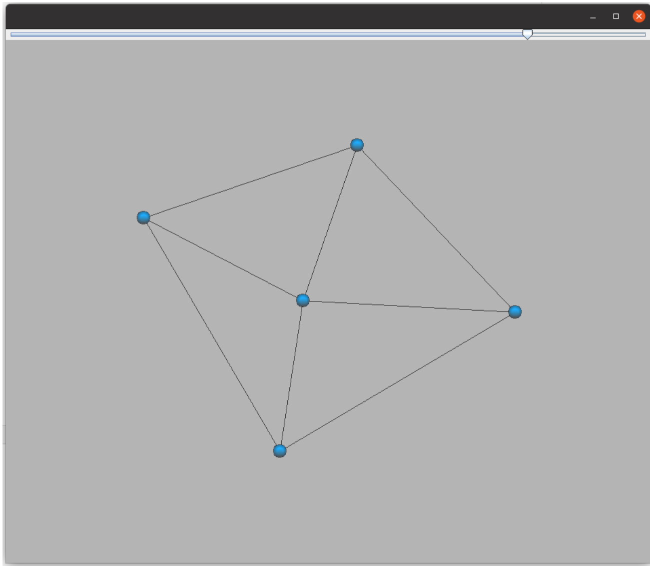
Utilisation en SyncD

- Réseau statique
- Process = Node
- Traitement des messages

En deux phases

- Programmation :
  - algorithme en Java
  - définition d'une topologie
  - traitement d'évènements
- Exécution :
  - définition du réseau
  - animation/simulation algorithme

Interface graphique



Programmation d'un algorithme

Main

- Définition de la topologie
- Définition de la classe de base pour les processus
- Définition de l'unité de temps

Exemple

```
public class Main {
  public static void main(String[] args){
    Topology tp = new Topology(500, 500);
    tp.setDefaultNodeModel(RobotProcess.class);
    tp.setTimeUnit(100);
    new JViewer(tp);
    tp.start();
  }
}
```

Programmation de l'algorithme

Classes de base

- `Class Node` : code de chaque processus
- `Class Link` : lien entre processus
- `Class Message` : définition des messages

Algorithmes : `Class Node`

Évènements : `@Override`

- `onStart()` : création de l'objet
- `onClock()` : pas de temps
- `onSelection()` : click souris
- `onMessage()` : réception

Attention : fonctions appelées séquentiellement => ne pas bloquer

Méthodes disponibles

- `getId()` : identificateur du processus
- `getNeighbors()` : liste des voisins
- `getLinks()` : liste des liens
- `send()` : envoi de messages
- `setColor()` : change la couleur

Classe Link : fonctions

- endpoints() : Liste des Node reliés
- endpoint(value) : Node

Classe Message : fonctions

- getSender() : Node émetteur
- getDestination() : Node destinataire
- getContent() : Object contenu

Communication

Envoi des messages

- `new Message( Object )` : création du message
- `send( Node, Message )` : envoi le message au processus donné en paramètre
- `sendAll( Node, Message )` : envoi le message à tous les voisins du processus

Réception des messages

- `onMessage( Message m )` : appelé à la réception d'un message par le processus
- `m.getSender()` : pour savoir de qui provient le message

Exemple - I

Premier programme

- Un robot envoi un message `ACK` à chacun des voisins
- Recevoir un message `ACK` de chacun des voisins

Développement

- IntelliJ / Projet Java / Classes Java
- `import : import io.jbotsim.core...;`
- External libraries : `io.jbotsim:jbotsim-all:1.2.0`
- Maven :

```
<dependency>
    <groupId>io.jbotsim</groupId>
    <artifactId>jbotsim-all</artifactId>
    <version>1.2.0</version>
</dependency>
```

Exemple - I

Message

```java
public class AckMessage {
  String name;
  int value;

  public AckMessage(String s, int n ) {
    this.name = s;
    this.value = n;
  }

  public String getName() {
    return name;
  }

  public int getValue() {
    return value;
  }
}
```

Exemple - I

BroadcastNode

```
public class BroadcastRobot extends Node {

  int msgNb = 0;

  @Override
  public void onStart() {
    setColor(null);
  }

  @Override
  public void onSelection() {

    AckMessage mm = new AckMessage("Node " + this.getID(), msgNb);
    Message m = new Message(mm);
    System.out.println(this.getID() + " sends a message");
    sendAll(m);
    setColor(Color.GREEN);
  }

  @Override
  public void onMessage(Message message) {
    msgNb ++;
    AckMessage mm = (AckMessage) message.getContent();
    ...println(this.getID() + " rcv from " + mm.getName() + " msg numb " + mm.getValue());
    if (msgNb == getNeighbors().size()) setColor(Color.RED);
  }
}
```

Exemple - I

BroadcastAckMessage

```
public class BroadcastAckMessage {
  public static void main(String[] args){
    //
    Topology tp = new Topology(1000, 1000);
    tp.setDefaultNodeModel(BroadcastRobot.class);
    tp.setTimeUnit(2000);
    new JViewer(tp);
    tp.start();
  }
}
```

Exemple - I

Simulation

- Lancer la topologie : run
- Créer un réseau
  - left-click : crée puis déplacer un nœud
  - right-click : menu ou détruire un nœud
  - links automatiquement créés, en fonction du communication range
- Simulation : middle-click, active le nœud (`onSelection`)

Menu

- Pause / restart
- Communication range
- System speed
- Load / save topologie

Implanter un algorithme de SyncD

Rappel

- Données : définitions locales à l'algorithme
- Messages : définir les différents types
- Régles : init, évènements, réception des messages

Principe

- Données : définitions locales au `Node`
- Messages : définir les différentes classes de messages
- Règles :
  - Init
  - Évènements
  - Réception des messages

Implanter un algorithme de SyncD

Règle Init

- `onStart()` : processus pas toujours finalisé
  - Voisins
  - Réseau
- `onClock()`

Implémentation `onClock()`

```
public class MyProcess extends Node {

  int nbTic = 0;
  ...
  @Override
  public void onClock() {

    nbTic++;

    // To be done once all the processes are started
    if ( nbTic == 3 ) {
      ...
    }
  }
}
```

Implanter un algorithme de SyncD

Autres évènements

Par exemple accès à la section critique

- Évènements réguliers : `onClock()`
  - Un compteur ; ts les 100
  - Un tirage aléatoire
- Contrôler les évènements : `onSelection()`

Types des messages - cas 1

Tous les messages ont les mêmes données

- Exemple : Lamport, tous les messages contiennent l'horloge
- Définition d'une seule classe de messages
- Identification du type à l'aide d'un `enum`
- Données associées dans l'objet
- Setter / Getter

Types des messages - cas 1

Implementation enum

```
public enum MsgType {
  REQ,
  ACK,
  REL;
}
```

Implementation Message

```
public class SyncMessage {
  MsgType type;
  int procId;

  public SyncMessage( MsgType t, int p ) { type = t; procId = p; }

  public MsgType getMsgType() { return type; }
  public int getMsgProc() { return procId; }
}
```

Types des messages - cas 1

Création Messages

```java
SyncMessage sm = new SyncMessage(MsgType.REQ, procId);
```

Utilisation Messages, identifier le type

```java
@Override
public void onMessage(Message message) {

    SyncMessage mm = (SyncMessage) message.getContent();
    switch ( mm.getMsgType()) {
        case REQ :
            receiveREQ( message.getSender() );
            break;
        case ACK :
            receiveACK( message.getSender() );
            break;
        case REL :
            receiveREL( message.getSender() );
            break;
        default:
            System.out.println("Error message type");
    }
}
```

Types des messages - cas 2

Tous les messages ont des données différentes

- Exemple : Suzuki-Kazami
  - Message `REQ` : horloge
  - Message `TOKEN` : tableau des accès

Solutions

1. Tous les messages possèdent toutes les données : même implémentation que précédemment
2. Messages avec des données différentes :
   - Une classe par type de message
   - Chaque classe possède ses propres données + setter/getter

Implementation des classes de messages

```
public class ReqMessage {
  int clock;
  int proc;

  public ReqMessage( int h, int p ) { clock = h; proc = p; }
  public int getMsgClock() { return clock; }
  public int getMsgProc() { return proc; }
}
```

Types des messages - cas 2

Implementation des classes de messages

```
public class TokenMessage extends Message {
  int[] tokenTab;
  int proc;

  public TokenMessage( int[] tt, int p ) { tokenTab = tt;   proc = p; }
  public int[] getTokenTab() { return tokenTab; }
  public int getMsgProc() { return proc; }
}
```

Types des messages - cas 2

Création Messages

```
ReqMessage rm = new ReqMessage(H, procId);
TokenMessage tm = new TokenMessage(TokenMsg, procId);
Message m = new Message( rm ou tm );
```

Utilisation Messages, identifier le type

```
@Override
public void onMessage(Message message) {

  Object m = message.getContent();

  if ( m instanceof ReqMessage ) {
    ReqMessage rm = (ReqMessage) m;
    algo.receiveREQ( rm.getMsgClock(), rm.getMsgProc(), door );
  } else if ( m instanceof TokenMessage ) {
    TokenMessage tm = (TokenMessage) m;
    algo.receiveTOKEN( tm.getTokenTab(), tm.getMsgProc() );
  } else {
    System.out.println("Error message type");
  }
```

Implanter un algorithme de SyncD

Réception des messages

- Différents types de messages
- `switch` en fonction du type de message
- Un type de message déclenche une règle = fonction

Code réception

```
// Message reception function
public void onMessage(essage msg) {
  SyncDMessage sm =  msg.getMessage();
  switch ( sm.getMsgType()) {
    case REQ :
    receiveREQ( sm.getMsgClock(), msg.getSender() );
    break;
    case ACK :
    receiveACK( sm.getMsgClock(), msg.getSender() );
    break;
    case REL :
    receiveREL( sm.getMsgClock(), msg.getSender() );
    break;
    default:
    System.out.println("Error message type");
  }
}
```

Pour aller plus loin

Problèmes :

- Visibilité des données
- Affichage des messages
- Contrôle du temps et de la topologie
- Accéder à un autre processus

Affichage des données

Problèmes :

- Déroulement de l'algorithme peu visible
- État des processus = couleur, mais mise à jour seulement à la fin d'une fonction
- Données à l'instant t : affichage des données du processus avec `System.out` dans la console, mais ne donne pas une vision globale
- Messages invisibles

Affichage des données

Solution

- Affichage de l'état des processus, pas pris en charge par JBotSim
- Définir des zones d'affichage par processus :
  - Fonction d'affichage dans le processus
  - Créer des `Frames`
  - Appel de la fonction d'affichage lorsque l'état du processus change : `display`
- Affichage possible dans la fenêtre de lancement de JBotSim

Affichage de l'état du processus

JFrame Java

```
public class DisplayFrame {
  JFrame frame;
  JTextArea textArea;
  int size;
  public DisplayFrame(int proc, int x, int y) {
    frame = new JFrame("Process " + proc);
    frame.setAlwaysOnTop (true);
    frame.setLocation(x, y);
    JPanel panel = new JPanel();
    frame.getContentPane().add(panel);
    // Create initial text area
    String empty = "     "; size = empty.length();
    // The size of the text area must be adapted here
    textArea = new JTextArea(empty, 10, 20);
    textArea.setEditable(false);
    panel.add(textArea);
    //Display the window.
    frame.pack();
    frame.setVisible(true);
  }
  public void display(String s) {
    int nextSize = s.length();
    textArea.replaceRange(s, 0, size);
    size = nextSize;
  }
}
```

Affichage de l'état du processus

Dans l'algorithme

```
@Override
public void init() {
  ...
  df = new DisplayFrame( procId );
  ... ; displayState();
  ... ; displayState();

}
```

Fonction `displayState()`

```
void displayState() {
  String state = new String("Clock = " + clock + "\n");
  if ( inCritical )
  state = state + "** ACCESS CRITICAL **";
  else if ( waitForCritical )
  state = state + "* WAIT FOR *";
  else
  state = state + "-- SLEEPING --";
  df.display( state );
}
```

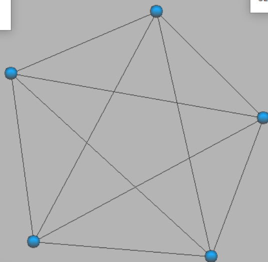
Affichage des données

```
public void onClock() {
    nbTic++;
    ... processes are started
        neighborList =
        nbNeighbors =
        F_H = new int[
        for (int i = 0;
        F_M = new MsgTy

        if (df != null)
        df = new Displa
        displayState();
    }

    // Reset color
    if ( !waitForCritic
        setColor(null);
    }

    // Manage access t
    if ( inCritical ) {
```



Process 1

Clock = 0

F_H[0] = 0	F_M[0] = null
F_H[1] = 0	F_M[1] = null
F_H[2] = 0	F_M[2] = null
F_H[3] = 0	F_M[3] = null
F_H[4] = 0	F_M[4] = null

-- SLEEPING --

Process 2

Clock = 0

F_H[0] = 0	F_M[0] = null
F_H[1] = 0	F_M[1] = null
F_H[2] = 0	F_M[2] = null
F_H[3] = 0	F_M[3] = null
F_H[4] = 0	F_M[4] = null

-- SLEEPING --

Process 3

Clock = 0

F_H[0] = 0	F_M[0] = null
F_H[1] = 0	F_M[1] = null
F_H[2] = 0	F_M[2] = null
F_H[3] = 0	F_M[3] = null
F_H[4] = 0	F_M[4] = null

-- SLEEPING --

Process 0

Clock = 0

F_H[0] = 0	F_M[0] = null
F_H[1] = 0	F_M[1] = null
F_H[2] = 0	F_M[2] = null
F_H[3] = 0	F_M[3] = null
F_H[4] = 0	F_M[4] = null

-- SLEEPING --

Process 4

Clock = 0

F_H[0] = 0	F_M[0] = null
F_H[1] = 0	F_M[1] = null
F_H[2] = 0	F_M[2] = null
F_H[3] = 0	F_M[3] = null
F_H[4] = 0	F_M[4] = null

-- SLEEPING --

Affichage des messages

Problèmes

- Messages envoyés depuis le code
- On ne voit pas quand ils partent, arrivent
- Pas de contrôle sur la vitesse

Solution : `Moving Messages`

- JBotSim permet des Nodes mobiles
- fixer la direction
- arrivée `onSensingIn()` : autre `Node` dans son champ de sensing

Affichage des messages

```
public class MovingMessage extends Node {

  Node emet, dest; Message msg;

  public MovingMessage( Node e, Node d, Message m ) {

    this.emet = e; this.dest = d; this.msg = m;
  }

  @Override
  public void onStart() {

    this.setCommunicationRange(0);  // Evite affichage des links
    this.setSensingRange(10);
  }

  @Override
  public void onClock() {

    setDirection( dest.getLocation() );
    move ( 10 );          // Controle du temps de transmission
  }

  @Override
  public void onSensingIn(Node node) {

    if (node instanceof BroadcastNode) { // Verifie type de node different de MovingMessage
      if ( dest.compareTo( node ) == 0 ) {  // Verifie dest
        ((BroadcastNode) node).deliver(this);
        this.die();
      }
    }
  }
}
```

Contrôle du temps

Problème : simuler les temps d'exécution

- Des processus : temps de travail, temps en section critique, ...
- Transmissions de messages

Solutions

- JBotSim permet de contrôler la vitesse d'exécution des processus et des transmissions de messages
  - `Topology.setTimeUnit(100);` en millisecondes
  - menu (right-click) + set system speed
- Temps passé sur un travail, par exemple temps de recharge :
  - Définir des temps d'exécution par tâche
  - Tirer valeur aléatoire, pour desynchroniser
  - Décrémenter sur `onClock()`
- Temps de transmission des messages :
  - longueur du lien
  - `DelayMessageEngine`

Topologie

Définition

- Définir à la main vs. Sauver/Recharger
- Communication range
  - menu
  - `Topology.setCommunicationRange()`
  - Pas toujours possible de faire ce qu'on veut
- Définir dans le programme
  - `Topology.addNode( position )`
  - `Topology.addLink(node1, node2)`

Accéder à un processus

Exemples

- Lelann : accès au successeur
- Suzuki-Kasami : envoi au premier processus qui a une valeur de demande supérieure à la date de sa dernière visite
- Naimi-Tréhel : envoi au `next`

Problèmes

- `Node.getNeighbors()` et `Node.getLinks()` liste non triées
- `sendAll` ne pose pas de problème
- `send` peut nécessiter des listes de correspondance
- Pas complète à `onStart()`

Accéder à un processus

Solution pour l'initialisation

```
@Override
public void onClock() {

  if (!init) {
    if (this.getID() != 0) {

      neighborList = getNeighbors();
    }
    init = true;
  }
}
```

Attention

Si les Nodes sont créés à la main il faut attendre plus de temps.

```
@Override
public void onClock() {

  if ( (nbTick > 100) \&\& !init )  {
    neighborList = getNeighbors();
    init = true;
  }
  nbTick++;
}
```

Pour la suite...

Documentations

- Site du projet : `https://jbotsim.io`
- Javadoc :
  `https://jbotsim.io/javadoc/1.2.0/index.html`
- Plus les exemples du cours