

Sinalgo

Bruno Pereira

Universidade Federal de Minas Gerais

bruno.ps@dcc.ufmg.com

3 de setembro de 2018

Agenda

- 1 **Introdução**
 - Sinalgo
- 2 **Primeiros Passos com o Sinalgo**
 - Ambiente de Desenvolvimento
 - Instalação e Documentação
- 3 **Projetos no Sinalgo**
 - Árvore de Diretório de um Projeto
 - Criando um Projeto
 - Implementação do Projeto
- 4 **Exercícios**
 - Exemplos do Sinalgo
 - Árvore de Roteamento

Introdução

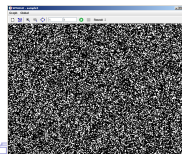
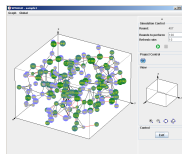
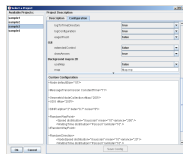
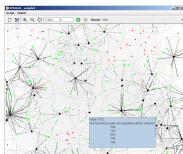
O que é?

- É uma ferramenta para testar e validar algoritmos de rede.
- Sinalgo abstrai de modo fácil e simples as camadas de enlace e física.
- Os nós enviam ou recebem mensagens unicast ou broadcast, reagem a mensagens recebidas, usam timers para escalonar ações, dentre outros.

Introdução

Características

- ❶ Torna fácil o desenvolvimento de algoritmos pois é implementado em *ctrl+space* JAVA.
- ❷ Alto desempenho, pois executa simulações com milhares de nós.
- ❸ Suporte a 2D e 3D.
- ❹ Simulação síncrona e assíncrona.
- ❺ Independente de plataforma (JAVA).
- ❻ Ambiente visual da simulação da rede.



Introdução

Fácil extensão (models)

- Mobilidade.
- Conectividade.
- Distribuição.
- Confiabilidade.
- Transmissão de Mensagens.

Getting Started

Pré-requisitos

- Java 5.0 (J2SE 5.0 JDK) ou superior.
- **Eclipse**.

Versões

- **Regular Release (recomendo)**.
- Toy Release.

Getting Started

Como instalar? (Regular Release com Eclipse)

- ❶ Crie um projeto ('File' → 'New' → 'Project')
- ❷ Escolha um projeto do tipo 'Java Project'.
- ❸ Dê um nome de sua preferência (ex: "SinalgoAula").
- ❹ Importe o Sinalgo Regular Release para seu Workspace.
 - ❶ Extraia sinalgo-x.xx.x.zip para um diretório temporário
 - ❷ Faça a cópia do conteúdo do Regular Release para o seu projeto
 - ❸ Certifique-se de que o arquivo **.classpath** foi sobrescrito.
- ❺ Verifique se Eclipse está configurado com Java 5.0 ou superior.
 - ❶ ('Window' → 'Preferences'), 'Java' → 'Compiler'
 - ❷ ('Window' → 'Preferences'), select 'Java' → 'Installed JREs'

- Mais detalhes... **Aqui.**

Getting Started

Como executar?

- 1 Clique com o botão direito na pasta “src” dentro da aba “Project Explorer” ou “Navigator” do Eclipse
- 2 Clique na opção “Run As” → “Java Application”
- 3 Na tela “Select Java Application”, selecione a classe “Run”.
- 4 Mais detalhes sobre **instalação** e **execução**.

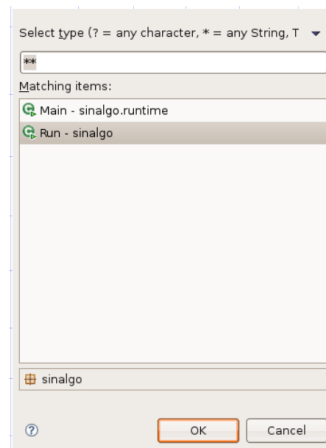


Figura: se tudo deu certo!

Getting Started

Como executar?

- 1 Clique com o botão direito na pasta “src” dentro da aba “Project Explorer” ou “Navigator” do Eclipse
- 2 Clique na opção “Run As” → “Java Application”
- 3 Na tela “Select Java Application”, selecione a classe “Run”.
- 4 Mais detalhes sobre **instalação e execução**.

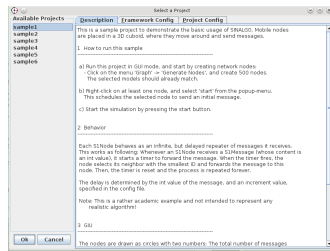


Figura: se tudo deu certo!

Getting Started

Documentação

- 1 O código é bem documentado
 - 1 Os 6 exemplos podem ser utilizados como base
- 2 doxygen
- 3 Sinalgo Site!
- 4 Mais detalhes sobre instalação e execução.

```
/**
 * The framework calls this method after starting the application and
 * before executing the first round.
 * <p>
 * This method may be used to perform any task that needs to be executed
 * before the simulation starts, e.g. initialize some datastructures.
 * <p>
 * By default, this method does nothing.
 */
public void preRun() {
    // No implementation here! Add your code to the CustomGlobal.java
    // file in your project.
}

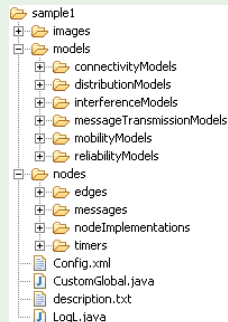
/**
 * The framework calls this method before each round. I.e. before the positions
 * of the nodes are updated or any node performs its step.
 */
public void preRound() {
    // No implementation here! Add your code to the CustomGlobal.java
    // file in your project.
}

/**
 * The framework calls this method after each round. I.e. after each node has
 * executed its step, and the interference has been checked, but before the
 * graph is redrawn (in GUI mode).
 */
public void postRound() {
    // No implementation here! Add your code to the CustomGlobal.java
```

Árvore de diretórios

Sinalgo Projects

- Sinalgo disponibiliza 6 projetos
- Um projeto é uma pasta dentro do diretório 'src/projects/' do Sinalgo



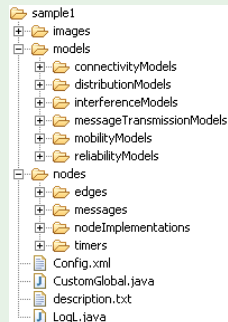
Árvore de diretórios

Sinalgo Projects

- Sinalgo disponibiliza 6 projetos
- Um projeto é uma pasta dentro do diretório 'src/projects/' do Sinalgo

Criando um novo projeto

- 1 Faça uma copia da pasta 'src/projects/template/' para o mesmo diretório
- 2 Modifique o nome 'template (copy)' para o nome do seu projeto
- 3 Execute a aplicação novamente...
- 4 Pronto!



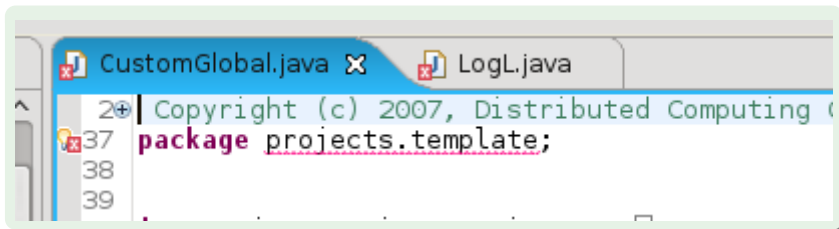
Erros

Erros aos copiar 'src/projects/template/'

É normal alguns erros de declaração dos packages.

Para corrigir basta modificar para:

package projects.nomeDoSeuProjeto;



Dica!

Obs: É altamente recomendado gerar um projeto para cada algoritmo simulado. Contudo, isso frequentemente resulta em código comum. Ao invés de copiar o mesmo código para todos os projetos, é preferível criar um novo projeto que armazenará esse código-comum e do qual todos os outros projetos terão acesso.

Meu Primeiro Projeto

Exemplo PingPong

Vamos criar uma simples simulação que demonstra os conceitos básicos.

Copie o diretório 'src/projects/template/' e modifique o nome para 'pingPong'

Meu Primeiro Projeto

Exemplo PingPong

Vamos criar uma simples simulação que demonstra os conceitos básicos.

Copie o diretório 'src/projects/template/' e modifique o nome para 'pingPong'

Comportamento PingPong

- Nesta simulação, dois nós vão trocar mensagens entre si.
- Os nós geram cores em RGB de modo aleatório.
- Cada cor gerada deve ser anexada em uma mensagem, que será enviada para o vizinho.
- Ao receber uma mensagem:
 - O nó deve alterar sua cor conforme os valores RGB recebidos.
 - Gerar uma nova cor RGB e enviar por broadcast.

Meu Primeiro Projeto

Por onde começo a implementar?

R: Em todos projetos temos que implementar:

- 1 Comportamento do nó
- 2 Modelos
- 3 Criar um arquivo de configuração

Como implementar o comportamento do nó?

Básica implementação

- ❶ Cada nó simulado é uma instância (sub-classe) da classe **"sinalgo.nodes.Node"**.
- ❷ Salve a classe em **'nodes/nodeImplementations/'** do projeto pingPong.
- ❸ Cada nó implementa seu próprio comportamento:
 - Ex: cada nó tem um método que é chamado quando uma mensagem é recebida. Também existe um método para enviar mensagens para seus vizinhos.
 - Você deve implementar o comportamento adequado para cada mensagem recebida.
- ❹ Cada nó tem sua própria instância de Mobilidade, Confiabilidade, Interferência, e Conectividade.
- ❺ Existe um modelo de Transmissão de Mensagens que é global.

Como implementar o comportamento do nó?

Criando um nó

- 1 Crie uma classe que estenda '**sinalgo.nodes.Node**' chamada "PingPongNode.java".
- 2 `handleMessages()` é o método onde o comportamento deve ser implementado.

OBS

Obrigatoriamente, deve-se implementar o método `handleMessages()` e, opcionalmente, qualquer um dos métodos abstratos da classe `sinalgo.nodes.Node`

```
1 package projects.pingPong.nodes.nodeImplementations;
import ...

3

public class pingPongNode extends Node {
5     private int local_r, local_g, local_b;
    @Override
7     public void handleMessages(Inbox inbox) {}

9     @Override
    public void preStep() {}

11

    @Override
13     public void init() {}

15

    @Override
    public void postStep() {}

17

    @Override
19     public void checkRequirements() throws
        WrongConfigurationException {}
}
```

Listing 1: "pingPongNode.java"

Como implementar o comportamento do nó?

Criando uma mensagem

- 1 Antes de implementar o `handleMessages()`, vamos criar nossa **Mensagem**.
- 2 A comunicação no Sinalgo é feita através de mensagens.
- 3 Toda mensagem deve herdar da classe **'sinalgo.nodes.messages.Message'**
- 4 As mensagens devem ser salvas no diretório:
 - **'nodes/messages/'** do projeto pingPong.
- 5 A classe abstrata `Message` requer que seja implementada apenas um método.
 - **`public Message clone();`**

Como implementar o comportamento do nó?

OBS

Quando um nó envia uma mensagem para um nó vizinho, assume-se que o destino recebe o conteúdo da mensagem que foi enviado pelo método `send()`. O framework, contudo, não tem como saber se o transmissor ainda tem uma referência para o objeto de mensagem enviado, e conseqüentemente, ele pode alterar a mensagem enviada. Para enviar tais problemas, o framework envia cópias separadas para todos os nós que recebem uma mensagem.

```
1 package projects.pingPong.nodes.messages;
2
3
4 import sinalgo.nodes.messages.Message;
5
6 public class PingPongMessage extends Message {
7
8     private int r, g, b;
9
10    public PingPongMessage(int i, int j, int k) { this.r = i;
11        this.g = j; this.b = k; }
12
13    @Override
14    public Message clone() { return new PingPongMessage(this.r
15        , this.g, this.b); }
16
17    /*Getters and Setters*/
18 }
```

Listing 2: "PingPongMessage.java"

Como implementar o comportamento do nó?

Criando uma mensagem

- A classe é auto-explicativa.
- Basicamente definimos os campos da mensagem e a implementação do método `clone()`, que será usado para gerar uma cópia da mensagem.

Como implementar o comportamento do nó?

Voltando a implementação do nó...

- 1 O método mais importante é o `handleMessage(Inbox inbox)`.
- 2 `handleMessages(Inbox inbox)` é o método onde o comportamento deve ser implementado.
- 3 Esse método é usado para tratar as mensagens que o nó recebeu.
- 4 Cada nó armazena as mensagens que recebe em uma instância da classe `Inbox`.
- 5 `Inbox` provê um iterador para acessar cada mensagem armazenada.
- 6 Para cada mensagem, o iterador armazena meta-informações, como o transmissor (`sender`) da mensagem.

```
1 public void handleMessages(Inbox inbox)
   while (inbox.hasNext())
3     Message message = inbox.next();
     if(message instanceof PingPongMessage)
5       PingPongMessage pkt = (PingPongMessage) message;
       this.local_r = pkt.getR();
7       this.local_g = pkt.getG();
       this.local_b = pkt.getB();
9
       this.setColor(new Color(local_r, local_g, local_b));
11
       UniformDistribution ud = new UniformDistribution(0,
13       255);

       pkt.setR( (int) ud.nextSample());
15       pkt.setG( (int) ud.nextSample());
       pkt.setB( (int) ud.nextSample());
17
       PingPongTimer timer = new PingPongTimer(pkt);
19       timer.startRelative(1, this);
```

Listing 3: "handleMessage"

Como implementar o comportamento do nó?

Voltando a implementação do nó...

- 1 A classe `pingPongNode.java` já é capaz de receber e tratar as mensagens recebidas.
- 2 **Mas como mandar mensagens?**
- 3 O que é **PingPongTimer**?

Como implementar o comportamento do nó?

Temporizadores

- 1 Temporizadores servem para especificarmos quando uma determinada ação deverá ser executada.
- 2 No nosso exemplo, criamos um temporizador com a seguinte declaração:
 - **PingPongTimer timer = new PingPongTimer(pkt);**
 - Perceba que passamos como parâmetro, para o construtor, a mensagem que queremos transmitir. Mais a frente, explicaremos o porque.

Como implementar o comportamento do nó?

Considerações

- Iniciamos o temporizador através do seguinte comando:
 - **timer.startRelative(1, this);**
 - O primeiro parâmetro indica em qual turno o temporizador será disparado. O segundo parâmetro indica qual objeto está disparando o temporizador.
 - O tempo 1 indica que o temporizador será disparado no próximo round.

Como implementar o comportamento do nó?

Criando um temporizadores

- ❶ Todo temporizador herda da classe **'sinalgo.nodes.timers.Timer'**
- ❷ Crie uma classe chamada PingPongTimer.java dentro de 'nodes/timers/':
- ❸ Basicamente, implementamos o método **fire()**, que é a ação a ser executada quando o temporizador for disparado.
- ❹ No nosso exemplo, queremos que:
 - Toda vez que o temporizador for disparado, ou seja, quando fire() é executado,
 - o método broadcast do nó que iniciou o temporizador será disparado para enviar a mensagem.

```
1 package projects.pingPong.nodes.timers;
3 import ...
5 public class PingPongTimer extends Timer {
6     private PingPongMessage pkt = null;
7
8     public PingPongTimer(PingPongMessage pkt) {
9         this.pkt = pkt;
10    }
11
12    @Override
13    public void fire() {
14        ((pingPongNode)node).broadcast(this.pkt);
15    }
16 }
```

Listing 4: "PingPongTimer.java"

Como implementar o comportamento do nó?

Criando um método para iniciar a comunicação

- 1 Como iniciar a troca de mensagens?
- 2 Adicione o seguinte método na classe PingPongNode.java:

Como implementar o comportamento do nó?

Criando um método para iniciar a comunicação

- 1 Como iniciar a troca de mensagens?
- 2 Adicione o seguinte método na classe PingPongNode.java:

```
@NodePopupMethod(menuText = "Init pingPong")
2 public void initPingPong() {
    PingPongMessage msg = new PingPongMessage(0,0,0);
4    PingPongTimer t = new PingPongTimer(msg);
    t.startRelative(1, this);
6 }
```

Listing 6: "NodePopupMethod"

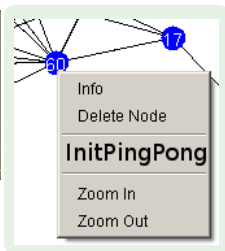
Como implementar o comportamento do nó?

Criando um método para iniciar a comunicação

- 1 Como iniciar a troca de mensagens?
- 2 Adicione o seguinte método na classe PingPongNode.java:

```
@NodePopupMethod(menuText = "Init pingPong")
2 public void initPingPong() {
    PingPongMessage msg = new PingPongMessage(0,0,0);
4    PingPongTimer t = new PingPongTimer(msg);
    t.startRelative(1, this);
6 }
```

Listing 7: "NodePopupMethod"



Como implementar o comportamento do nó?

Arquivo de configuração

- ❶ Nossa simulação já está pronta!!!
- ❷ Falta somente configurar o projeto:
 - Edite o arquivo **pingPong/config.xml**

```
1 <Document>
2   <Framework>
3     ...
4   </Framework>
5   <Custom>
6     ...
7   </Custom>
8 </Document>
```

Listing 8: "Config.xml"

```
1 <Document>
  <Framework>
3    ...
  </Framework>
5 <Custom>
  <QUDG rMin="30" rMax="50" ProbabilityType="constant"
    connectionProbability="0.6"/>
7
  <SINR alpha="2" beta="0.7" noise="0"/>
9
  <RandomWayPoint>
11    <Speed distribution="Gaussian" mean="10" variance="20" /
    >
    <WaitingTime distribution="Poisson" lambda="10" />
13  </RandomWayPoint>
15
  <RandomDirection>
    <NodeSpeed distribution="Gaussian" mean="10" variance="
20" />
17    <WaitingTime distribution="Poisson" lambda="10" />
    <MoveTime distribution="Uniform" min="5" max="20" />
19  </RandomDirection>
  </Custom>
21 </Document>
```



Git é vida!

```
git clone git@github.com:bps90/aula-simuladores-redes-sem-fio.git
```

Tarefa 1

Exemplos do Sinalgo

- 1 Execute os 6 exemplos do Sinalgo.
- 2 Descreva a finalidade do exemplo.
- 3 Quais conceitos visto em sala de aula que são demonstrados em cada exemplo.
- 4 Descreva as limitações de cada exemplo.
- 5 Quais os pontos fortes e fracos do Sinalgo?

Tarefa 2

Árvore para coleta de dados

- ❶ Faça uma inundação para descobrir o menor caminho (em saltos) de cada nó para uma Estação Base (EB).
- ❷ A EB deve ser iniciada através de **@NodePopupMethod**
- ❸ As mensagens podem ser do tipo rota ou dados
- ❹ O nó deve identificar se a mensagem é de construção de rota ou de dados.
 - Se a mensagem é de construção de rota
 - O nó deve atualizar sua rota para a EB.
 - Se a mensagem é de dados
 - O nó deve mandar uma mensagem unicast para o próximo salto até a EB, caso exista uma rota válida

Tarefa 2

Árvore para coleta de dados

- 1 A EB ao receber uma mensagem deve imprimir no Output informações sobre a mensagem coletada.
- 2 Deve ser permitido que um nó envie dados periodicamente (10 rounds) para EB, isto deve ser ativado através de um **@NodePopupMethod**. Observe que o nó deve ter um caminho válido para esta opção.
- 3 O payload da mensagem de dado pode ser uma amostra de temperatura ou alguma variável de ambiente geralmente analisada por RSSF.
- 4 A mensagem não pode trafegar mais que um tempo de vida estabelecido (Ex: 30 saltos)

Tarefa 2

DICAS

- ① Use Tools.java para:
 - Escrever no output
 - Números aleatórios ***
 - Manipular erros
 - Informações sobre a simulação
 - Acessar conjunto de nós
 - Acessar mensagens
 - Usar métodos relacionados com a GUI
 - Parar a simulação
- ② Como mandar mensagens unicast?
 - R: documentação sinalgo! =D

Tarefa 2

DICAS

- ❶ Quais classes devo criar?
 - Uma classe para representar o nó (Ex: "MyNode.java")
 - A classe deve conter uma referência para o próximo salto até a EB.
 - Deve ter uma variável para indicar sua distância até a EB.
 - Criar um método menu popup para iniciar a EB na classe "MyNode.java".
 - Implemente um mecanismo para evitar loop das mensagens (Dica: use o campo número de saltos da mensagem)

Tarefa 2

DICAS

- ❶ Quais classes devo criar?
 - Uma ou duas classes para representar as mensagens (rota e dados)
 - Cabeçalho indicado:
 - número de seqüência.
 - tempo de vida.
 - destino e origem.
 - nó que encaminhou a mensagem
 - número de saltos até a EB
 - tipo (se fez uma classe só)

Tarefa 2

DICAS

- 1 Quais classes devo criar?
 - Um ou dois timers para enviar as mensagens do flood (para descoberta das rotas) e para enviar periodicamente as mensagens de dados.
 - Implementar um temporizador que armazene um pacote.
 - implementar adequadamente o método **fire()**.

Até a próxima aula!