School of Computer Science
Carleton University

# DisJ Cookbook

(Version 1.0.2)

By: Nothapol (Rody) Piyasin
Supervisor: Prof. Nicola Santoro

*Revision Date: September 23, 2005*

Appended By: Ruogu (Russell) Luo
Supervisor: Prof. Paola Flocchini
*Revision Date: December 2008*

# Table and Contents

# Chapter 1. Overview

This cookbook is a step-by-step description of the process of entering, executing and testing a reactive distributed algorithm using DisJ. It also describes how to use DisJ to simulate the algorithm and display the results in graphical interactive.

Here are four simple steps for using DisJ.

1 Install DisJ plug-in into Eclipse
2 Define a topology in DisJ graph editor
3 Write reactive protocol in Java language in Eclipse JDT
4 Execute the protocol in the defined topology using DisJ

Throughout the cookbook we will use as an example protocol "*As Far*" that elects a leader in a ring with bidirectional links network (see Appendix)

# Chapter 2. Preliminaries

## 2.1 Installing DisJ

### 2.1.1 Install Graphical Editor Framework (GEF)

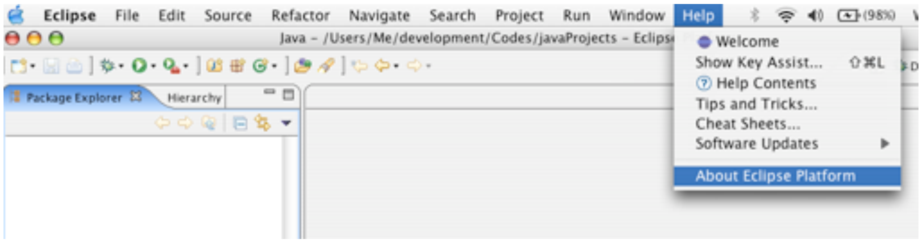
In order to use Graphic Editor in Eclipse™ we have to install Graphical Editor Framework (GEF) library. GEF is an open source project under support of Eclipse Foundation which can be downloaded at http://www.eclipse.org/gef/ .

To install GEF, we have to uncompress file and copy everything under features folder into ECLIPSE_HOME/features, then copy everything under plugins folder into ECLIPSE_HOME/eclipses as well.

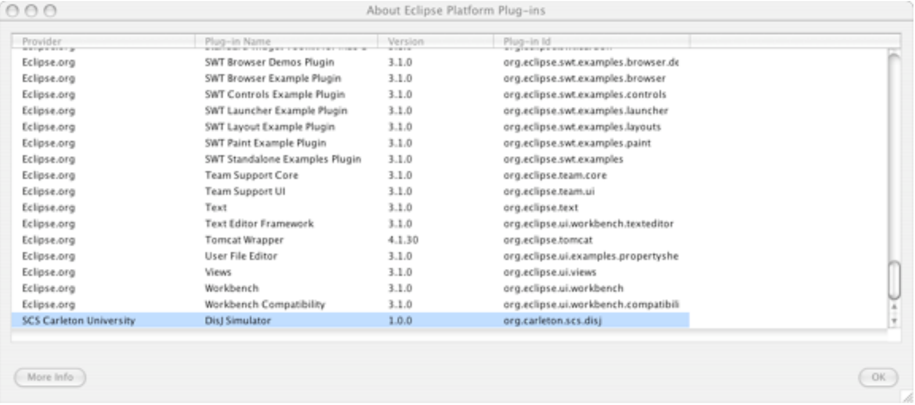
### 2.1.2 Installing DisJ distribution package

The DisJ is an Eclipse™ plug-in, therefore we need to have Eclipse™ (version 3.0 or higher) installed. Eclipse is an open platform for tool integration built by an open community of tool providers. Operating under a open source paradigm and it can be downloaded at *www.eclipse.org*. After Eclipse is installed, download DisJ and unzip and copy folder org.Carleton.scs.DisJ_1.0.0 into the installed *ECLIPSE_HOME/plugins* folder.

To make sure that the plug-in was successfully installed, **restart** Eclipse™. Go to the menu *Help -> About Eclipse Platform*



Then select "*Plug-ins Details*"; the dialog must appear on the screen, (as shown below) if the installation is correct. Inside the dialog must contain **DisJ Simulator** as in the example below.

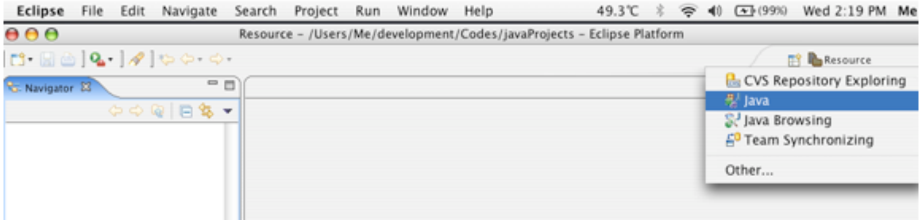


| Provider | Plug-in Name | Version | Plug-in Id |
|---|---|---|---|
| Eclipse.org | SWT Browser Demos Plugin | 3.1.0 | org.eclipse.swt.examples.browser.de |
| Eclipse.org | SWT Browser Example Plugin | 3.1.0 | org.eclipse.swt.examples.browser |
| Eclipse.org | SWT Controls Example Plugin | 3.1.0 | org.eclipse.swt.examples.controls |
| Eclipse.org | SWT Launcher Example Plugin | 3.1.0 | org.eclipse.swt.examples.launcher |
| Eclipse.org | SWT Layout Example Plugin | 3.1.0 | org.eclipse.swt.examples.layouts |
| Eclipse.org | SWT Paint Example Plugin | 3.1.0 | org.eclipse.swt.examples.paint |
| Eclipse.org | SWT Standalone Examples Plugin | 3.1.0 | org.eclipse.swt.examples |
| Eclipse.org | Team Support Core | 3.1.0 | org.eclipse.team.core |
| Eclipse.org | Team Support UI | 3.1.0 | org.eclipse.team.ui |
| Eclipse.org | Text | 3.1.0 | org.eclipse.text |
| Eclipse.org | Text Editor Framework | 3.1.0 | org.eclipse.ui.workbench.texteditor |
| Eclipse.org | Tomcat Wrapper | 4.1.30 | org.eclipse.tomcat |
| Eclipse.org | User File Editor | 3.1.0 | org.eclipse.ui.examples.propertyshe |
| Eclipse.org | Views | 3.1.0 | org.eclipse.ui.views |
| Eclipse.org | Workbench | 3.1.0 | org.eclipse.ui.workbench |
| Eclipse.org | Workbench Compatibility | 3.1.0 | org.eclipse.ui.workbench.compatibili |
| SCS Carleton University | DisJ Simulator | 1.0.0 | org.carleton.scs.disj |

## 2.2 Changing Perspective

After Eclipse has started a workbench is opened; the workbench can be viewed from many different perspectives; the default one is the "*Resource Perspective*". Since we are going to develop the distributed algorithm in Java™ it is better to switch to the "*Java Perspective*". This forces the user to follow the rules and best practices in software development.

Click at icon on a top left to change switch the perspective (as shown below).

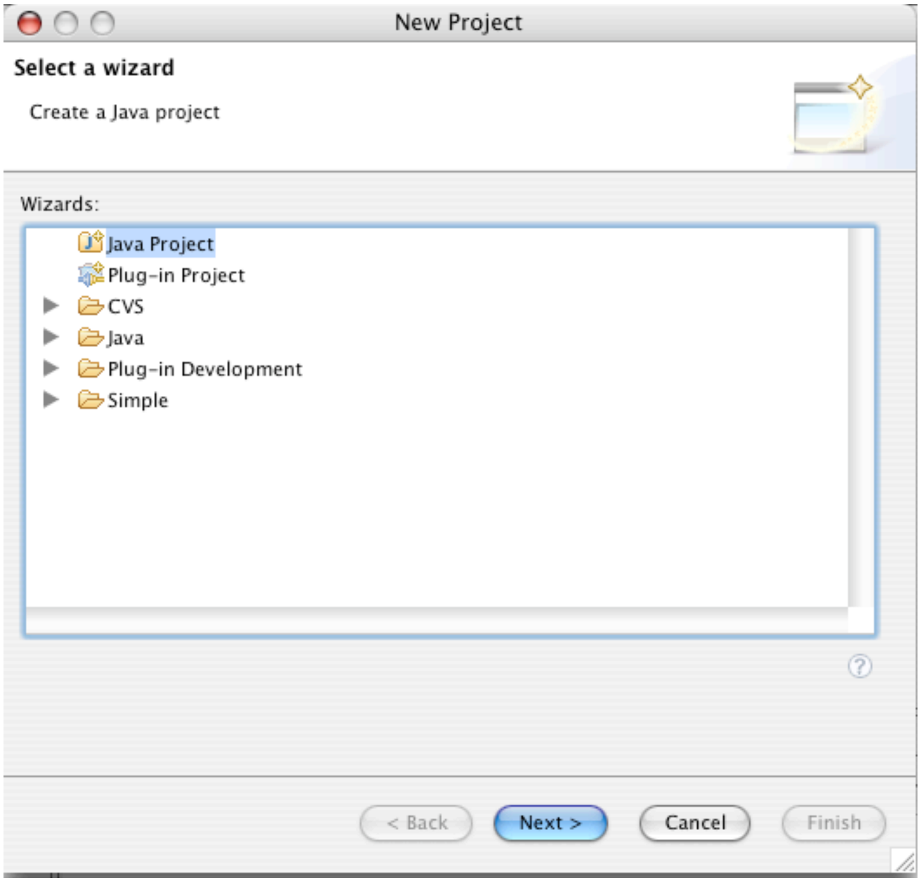

Eclipse File Edit Navigate Search Project Run Window Help 49.3℃ (99%) Wed 2:19 PM Me
Resource – /Users/Me/development/Codes/javaProjects – Eclipse Platform
Resource
Navigator
CVS Repository Exploring
Java
Java Browsing
Team Synchronizing
Other...

# Chapter 3. Defining the Topology

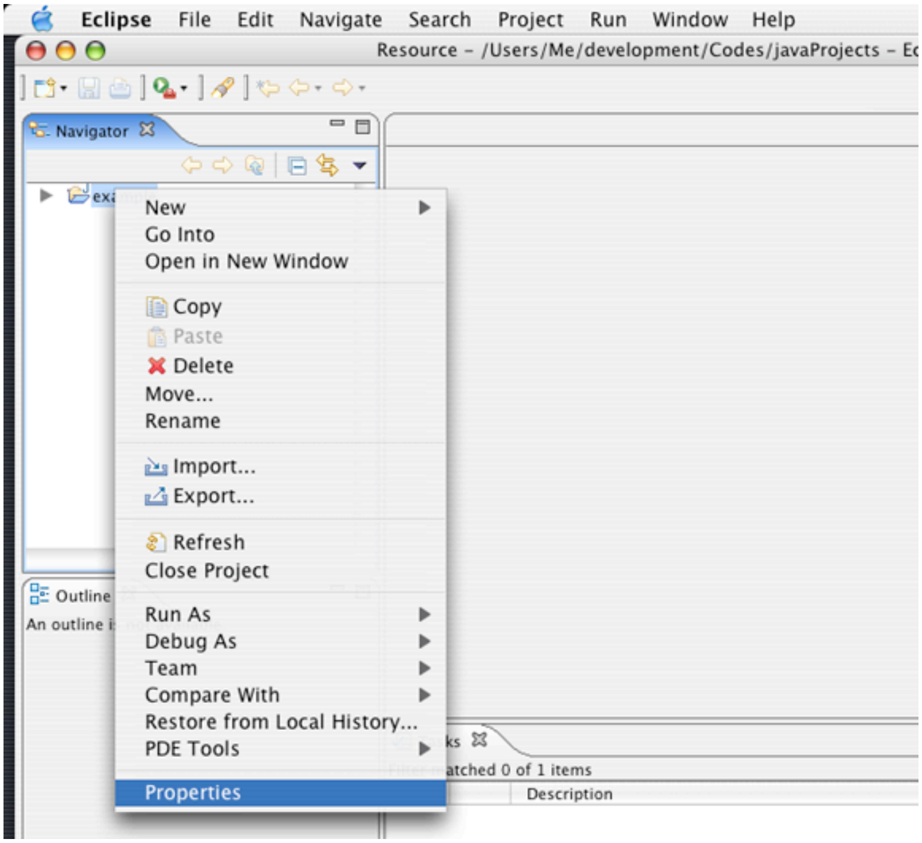
## 3.1 Creating a Graph File

There are 3 small steps to create a file.

3.1.1 Create a project folder by right click at "*Navigator View*" or at menu *File->new->project*; once the dialog shows up, then choose "*Java Project*" and follow the instructions as shown below.

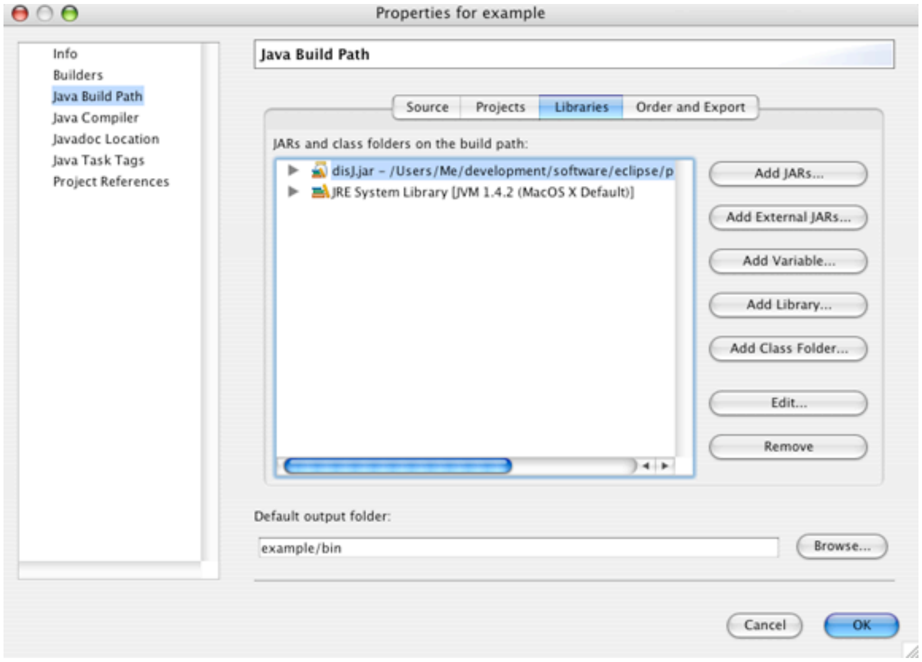


3.1.2 Right click at the project folder in "*Package Explorer View*", and select "*Properties*" (see below);



Once the dialog shows up, go to *Java Build Path-> Libraries->Add External Jars*, select a jar file from our unzipped folder, then click OK.

Here is the project properties dialog.



Properties for example

Java Build Path

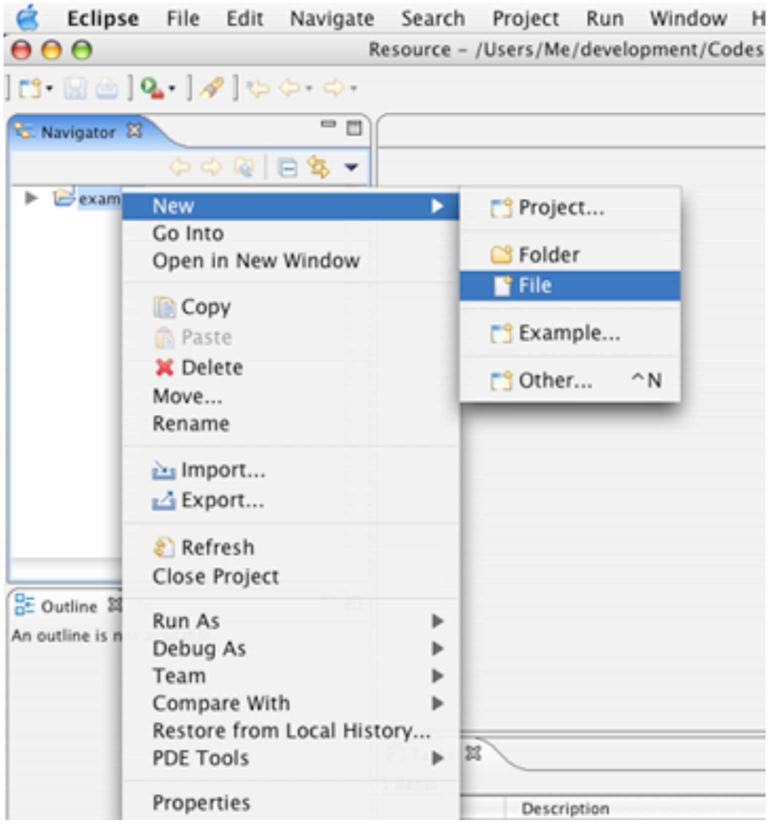
JARs and class folders on the build path:

- disJ.jar – /Users/Me/development/software/eclipse/p
- JRE System Library [JVM 1.4.2 (MacOS X Default)]

Default output folder:

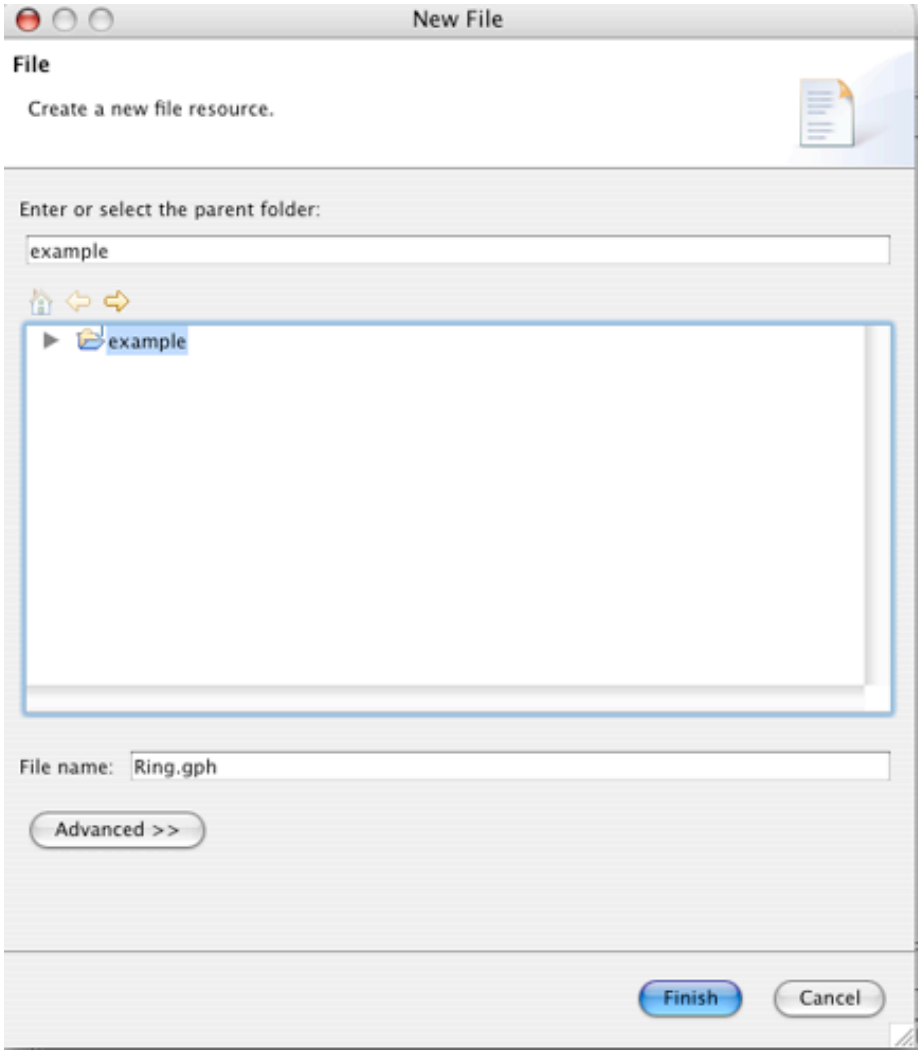
example/bin

3.1.3 To create a graph file, right click at the project in "*Package Explorer View*" then go to *New->File* (shown below);



Once the dialog shows up, assign a name to the file with extension "*.gph*", and click finish.

Here is a new file dialog.



New File

**File**

Create a new file resource.

Enter or select the parent folder:

example

example

File name: Ring.gph

## 3.2 Defining a graph

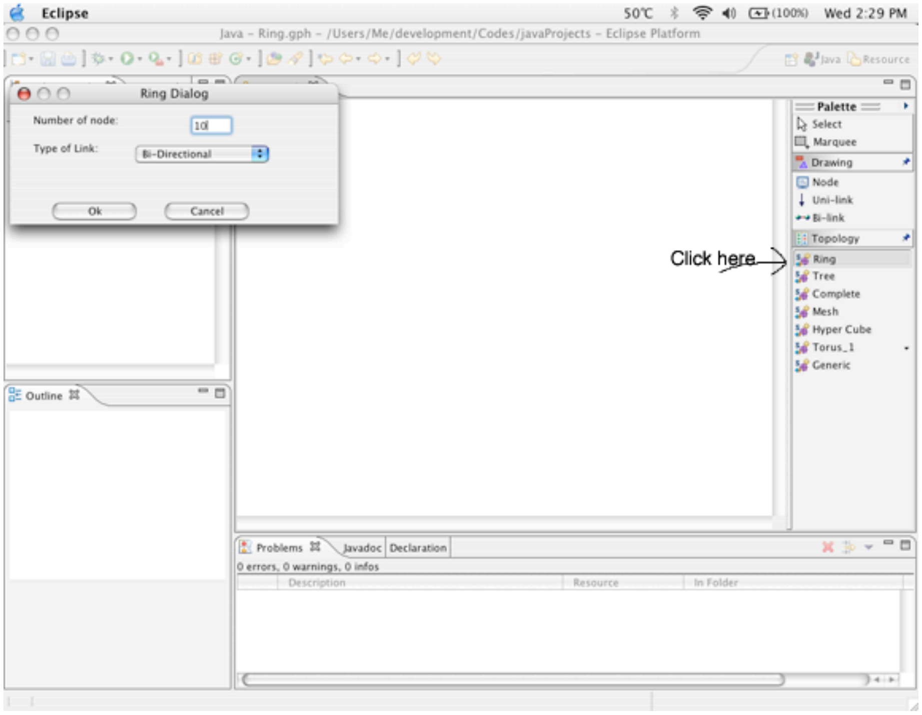
The DisJ Graph Editor shows up on the editor view. We start drawing by expanding the "*Palette*" on the right corner of the editor tab. There are two ways to define a graph:

3.2.1 **Draw** - this is a basic drawing tool based on simple objects such as node, uni-directional link and bi-directional link.

3.2.2 **Topology Library**- this provides a ready-made set of topologies that can be edited such as Ring, Tree, Complete Graph, Mesh, Hyper Cube, Arbitrary Graph and Torus.

Drawing the object on the editor is simple; as in other drawing applications, the user can choose to draw the objects manually from the drawing category or use a readymade draw from Topology Library.

Here is an example of defining a bidirectional Ring with 10 nodes using the Topology Library.



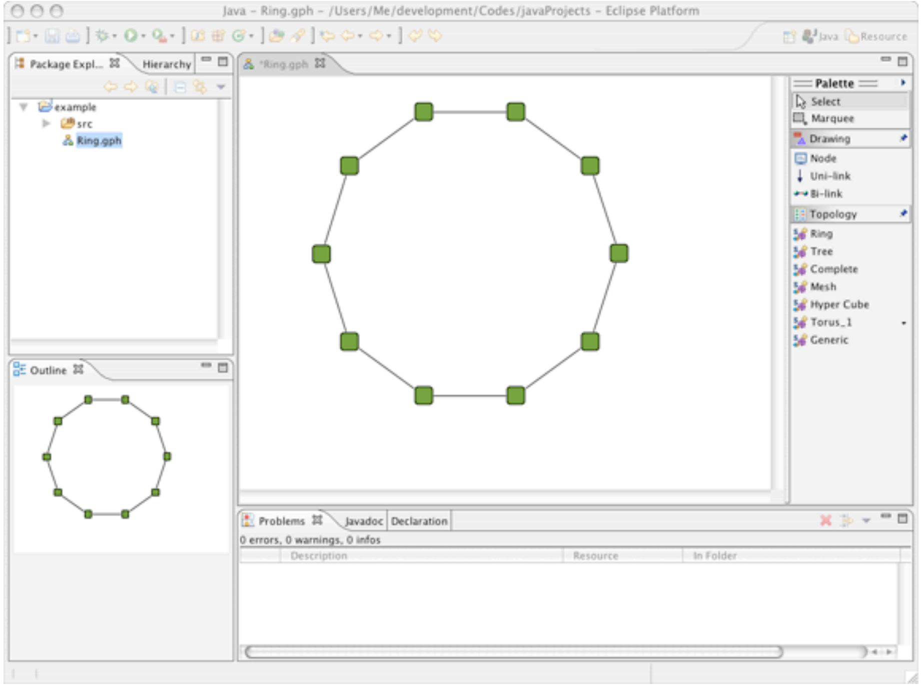
Ring Dialog

Number of node: 10

Type of Link: Bi-Directional

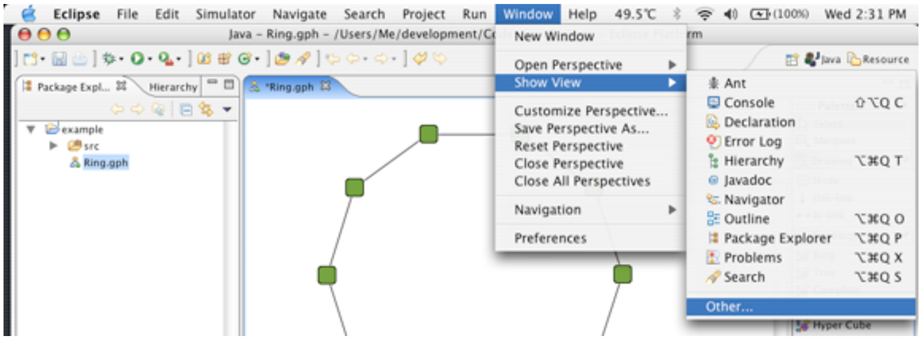
Click here

Here is the snapshot of DisJ when a Ring of size 10 is created.

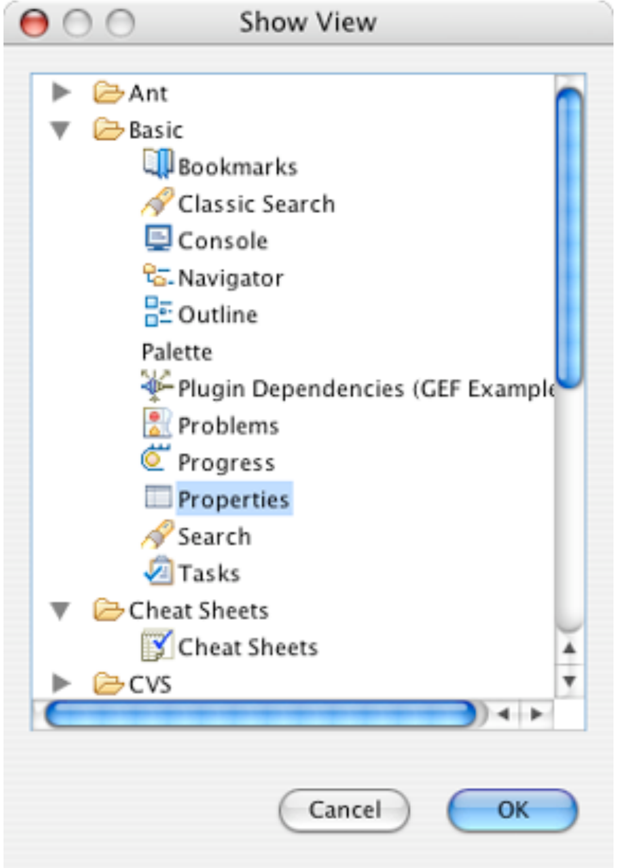


0 errors, 0 warnings, 0 infos

Once, the graph has been drawn, you can check if there is the correct number of links and nodes by opening a "*Properties View*" which will show the current states and properties. To open the "*Properties View*", go to menu bar, select *Window->Show View-> Properties*; if it does not list, select *Window->Show View->Others*.



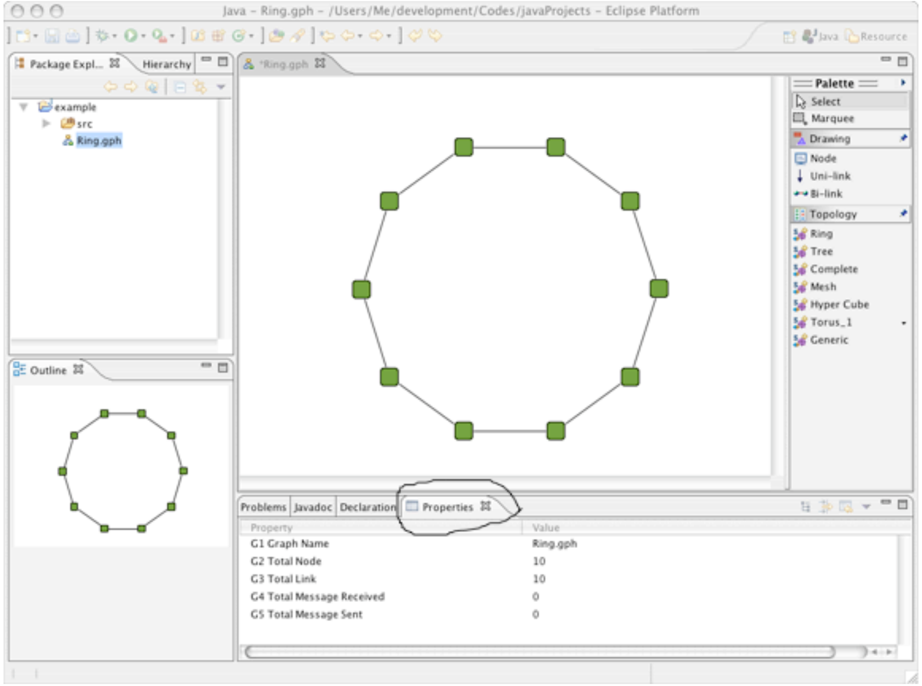
Once the dialog shows up, expand a “*Basic*” folder and select “*Properties*”.



In addition to the graph properties; there are properties of the nodes and of the links. We will now describe them in some detail.

## 3.3 Element Properties

### 3.3.1 Graph Properties



| Property | Value |
|---|---|
| G1 Graph Name | Ring.gph |
| G2 Total Node | 10 |
| G3 Total Link | 10 |
| G4 Total Message Received | 0 |
| G5 Total Message Sent | 0 |

3.3.1.1 **Graph Name** (not editable)
This is the name of the file where the graph has been defined.

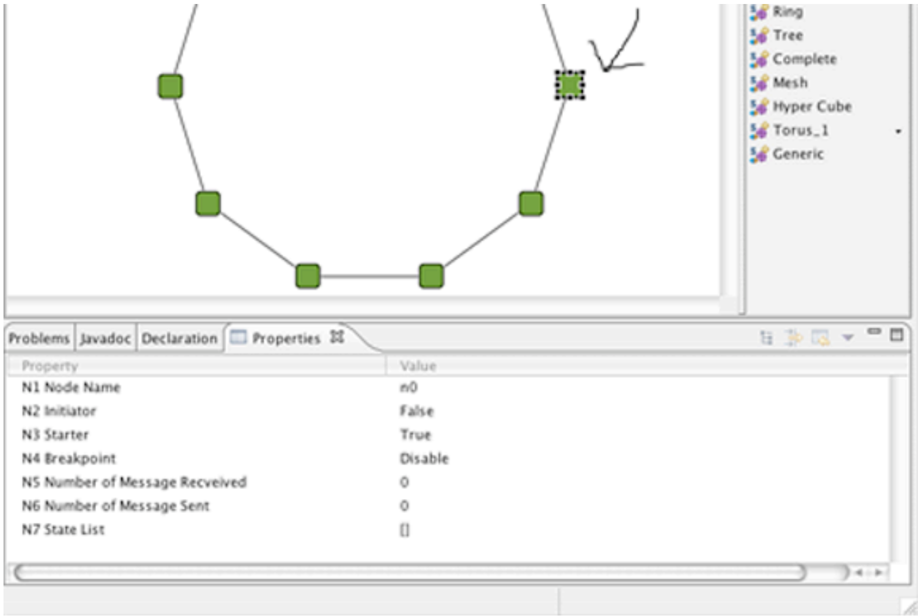
3.3.1.2 **Total Link** (not editable)
Total number of links in this graph.

3.3.1.3 **Total Node** (not editable)
Total number of nodes in this graph.

3.3.1.4 **Total Number of Messages Received** (not editable)
Total numbers of messages that have been received from the start of the execution of the protocol up to now.

3.3.1.5 **Total Number of Messages Sent** (not editable)
A total numbers of messages that have been sent from the start of the execution of the protocol up to now.

### 3.3.2 Node Properties



| Property | Value |
|---|---|
| N1 Node Name | n0 |
| N2 Initiator | False |
| N3 Starter | True |
| N4 Breakpoint | Disable |
| N5 Number of Message Recveived | 0 |
| N6 Number of Message Sent | 0 |
| N7 State List | [] |

3.3.2.1 **Breakpoint** (editable)

This property is used for debugging purpose. Once a node's breakpoint is enabled, the execution will be suspended as soon as a message arrives to that node. In this way the user is able to observe the current states of everything in the graph.

3.3.2.2 **Initiator** (editable)

This property is for specifying if the node is an initiator state of the protocol. Only a node with this property set to True will execute "`distributed.plugin.runtime.enging.Entity.init()`" (*see Chapter 4 Writing Protocol*).

3.3.2.3 **Starter** (editable)

This property is used for a physical start of an execution. The user must set it to be True in at least one node in order to run the algorithm.

3.3.2.4 **Node Name** (editable)

This represent s the user-defined a name of the node; it can be unique or in common with other nodes, as the user prefers

3.3.2.5 **Number of Messages Received** (not editable)

This is statistics report stating the number of messages that the node has received (from any port) from the start of the execution until now.

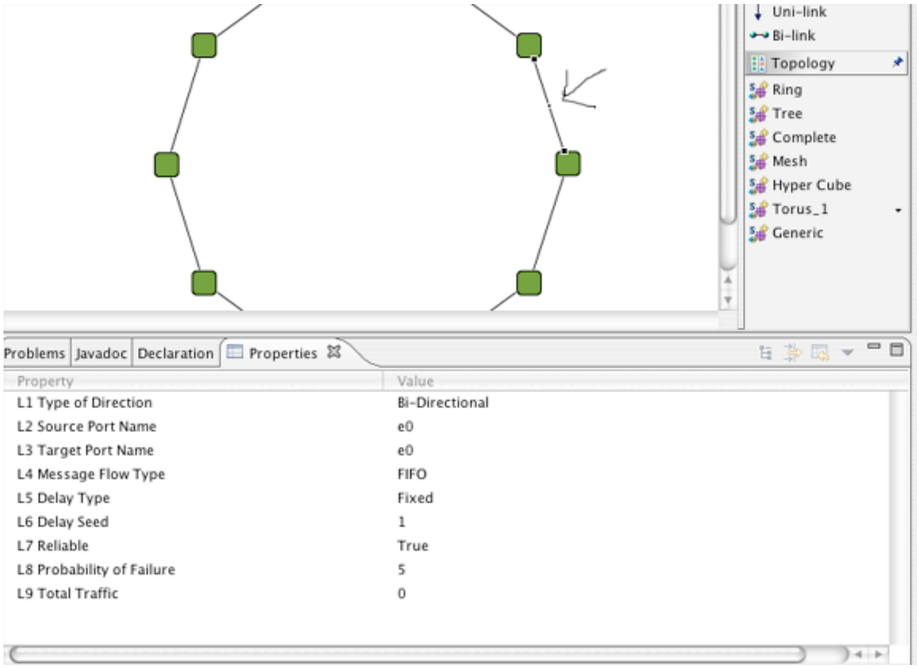
3.3.2.6 **Number of Messages Sent** (not editable)

This is statistics report stating the number of messages that the node has sent (to any port) from the start of the execution until now.

3.3.2.6 **State List** (not editable)

This property reports the sequence of states that the node has entered from the start of the execution until now

### 3.3.3 Link Properties



| Property | Value |
|---|---|
| L1 Type of Direction | Bi-Directional |
| L2 Source Port Name | e0 |
| L3 Target Port Name | e0 |
| L4 Message Flow Type | FIFO |
| L5 Delay Type | Fixed |
| L6 Delay Seed | 1 |
| L7 Reliable | True |
| L8 Probability of Failure | 5 |
| L9 Total Traffic | 0 |

3.3.3.1 **Delay Seed** (editable)

This property is for setting the amount of time spent in transmission on this link. This is not taken into account if the *Delay Type* is set to *Random.*

3.3.3.2 **Delay Type** (editable)

This property is specify the type of delay, which there are three types

- Fixed: the delay time is fixed with a given Delay Seed and it is same for every messages sent on this link.
- Uniform Random: a delay time for each message that pass through will be generated at random with uniformly distribution. If message flow is FIFO *(see Chapter3.3.3.6)*, the generator will satisfy.
- Customs: a delay time for each message base on a user given library that plug-in into DisJ (see DisJ System Manual) by default it is the same as Uniform Random.

3.3.3.3 **Target Port Name** (editable)

The local name of incoming port of Target node; if a link is bidirectional it just mean a local port name of one end node of the link.

3.3.3.4 **Probability of Failure** (editable)

This property allows user to specify the probability that a message will fail on this link, if a *Reliable* of the link is set to false.

#### 3.3.3.5 **Reliable** (editable)

This property is for verifying the reliability of a link; if it is True no message will be lost, else it will be lost with uniformly distributed random with probably that set in *Probability of Failure*.

#### 3.3.3.6 **Message Flow Type** (editable)

This property specifies the flow type of messages in a link. There are only two types: FIFO and No Order. These two are no difference if *Delay Type* is *Fixed.*

#### 3.3.3.7 **Source Port Name** (editable)

The local name of outgoing port of Source node; if a link is bi-directional it just mean a local port name of one end node of the link.

#### 3.3.3.8 **Total Traffic** (not editable)

This is statistics report on number of messages transmitted on t he a link (this includes messages lost on this link if *Reliable* is set to False)

#### 3.3.3.9 **Type of Direction** (not editable)

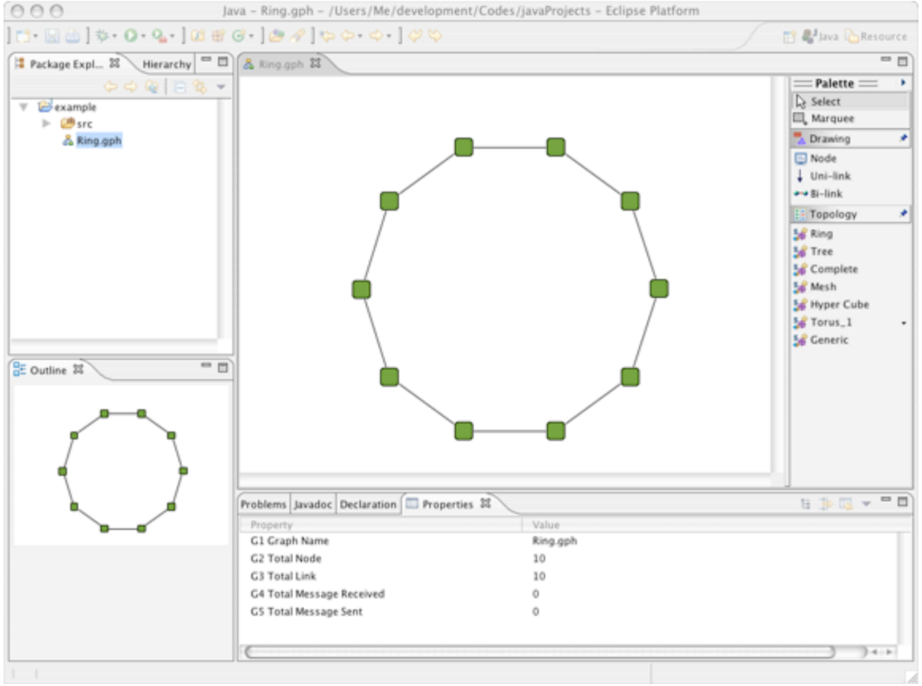
There are two types of links for DisJ: unidirectional and bidirectional link.

## 3.4 Different Types of Topology

Franz Oppacher
**Comment:** Get a new and complete set of snapshot for each topology

DisJ provides a Topology Library that offers a set of readymade graph and allows the user to create different types of topologies by editing them.

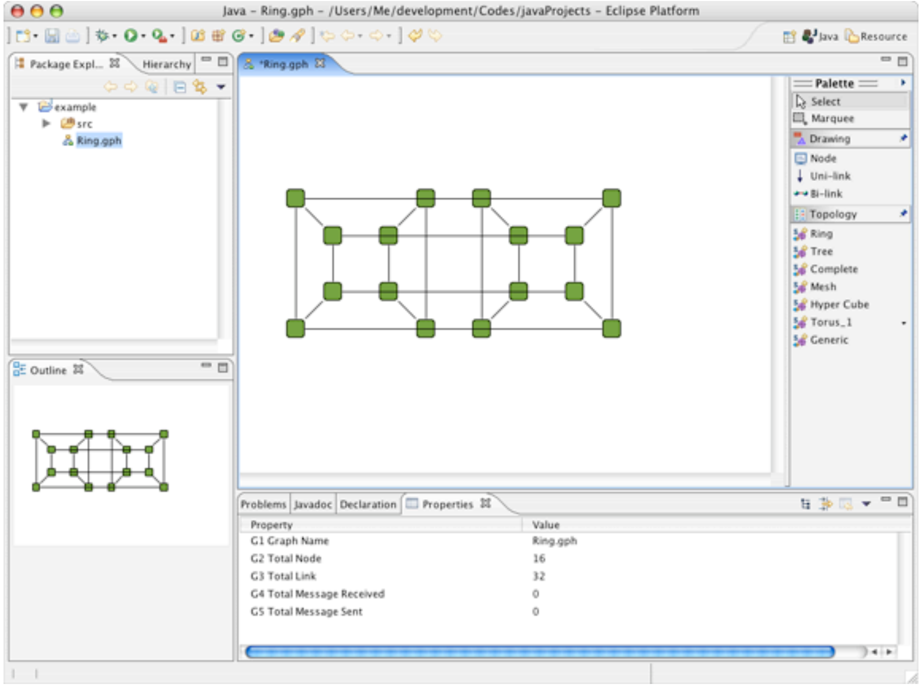
### 3.4.1 Ring



| Property | Value |
|---|---|
| G1 Graph Name | Ring.gph |
| G2 Total Node | 10 |
| G3 Total Link | 10 |
| G4 Total Message Received | 0 |
| G5 Total Message Sent | 0 |

### 3.4.2 Hyper Cube

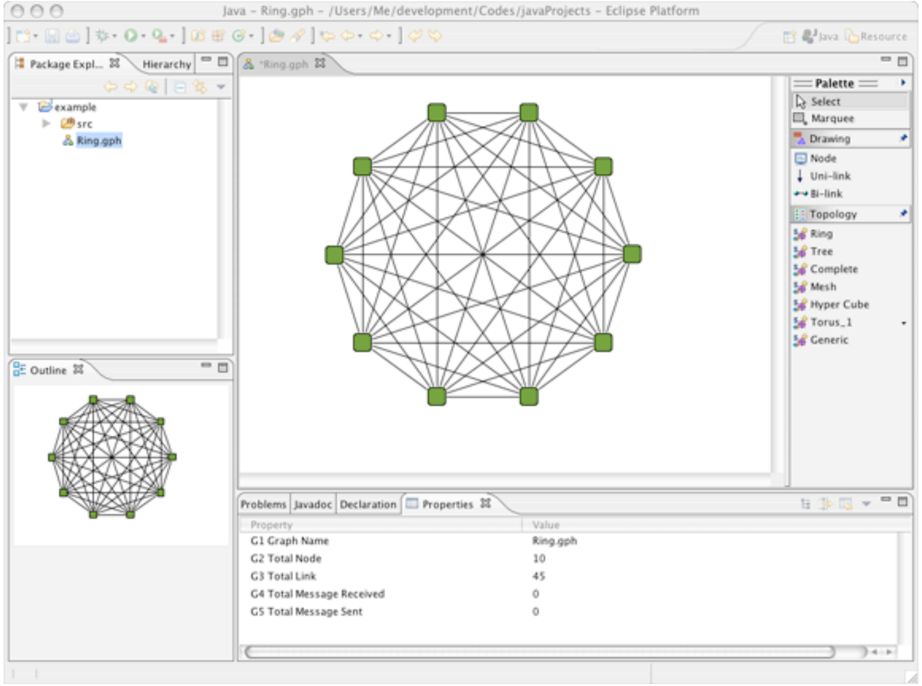
Franz Oppacher
**Comment:** It supports up to 6 dimensions only



| Property | Value |
|---|---|
| G1 Graph Name | Ring.gph |
| G2 Total Node | 16 |
| G3 Total Link | 32 |
| G4 Total Message Received | 0 |
| G5 Total Message Sent | 0 |

User can modify the location of links or nodes by click and drag.

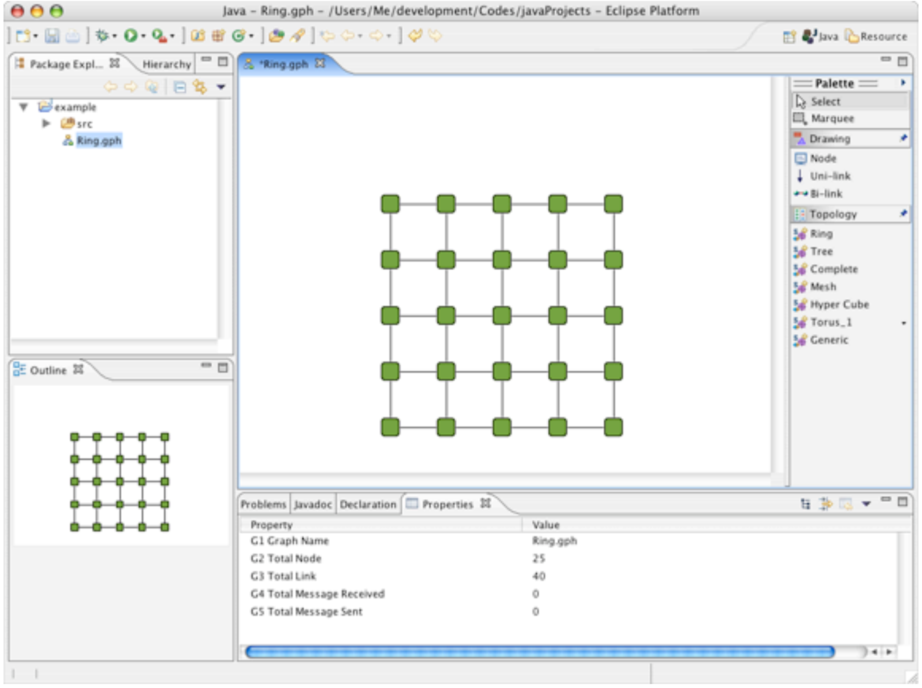
### 3.4.3 Complete Graph



| Property | Value |
|---|---|
| G1 Graph Name | Ring.gph |
| G2 Total Node | 10 |
| G3 Total Link | 45 |
| G4 Total Message Received | 0 |
| G5 Total Message Sent | 0 |

To undo or redo an action by right click on a canvas, then select an option.

### 3.4.4 Mesh

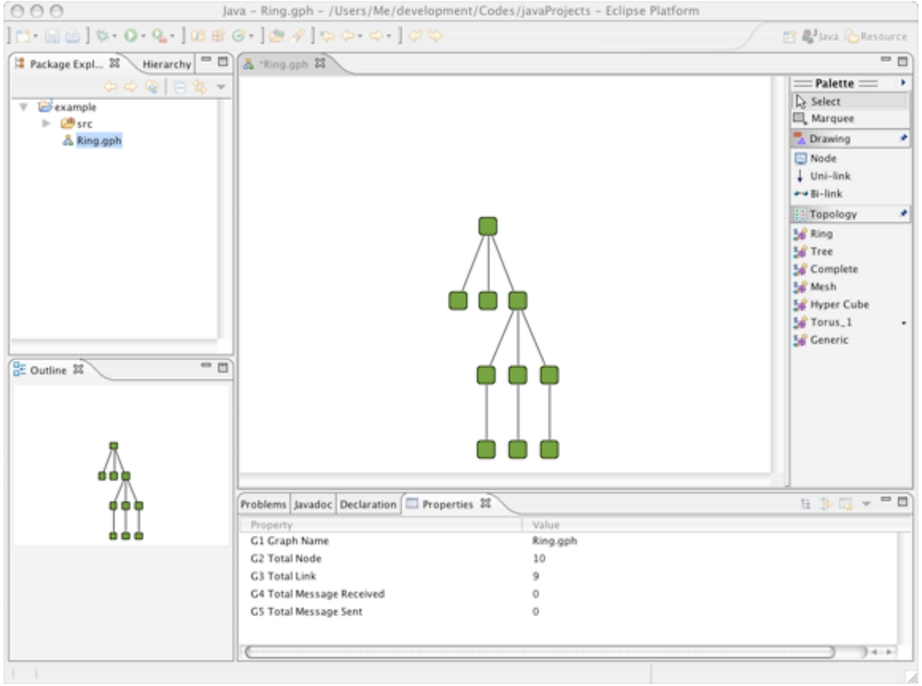


| Property | Value |
|---|---|
| G1 Graph Name | Ring.gph |
| G2 Total Node | 25 |
| G3 Total Link | 40 |
| G4 Total Message Received | 0 |
| G5 Total Message Sent | 0 |

### 3.4.5 Tree

There are 2 types of Trees provided by the library.

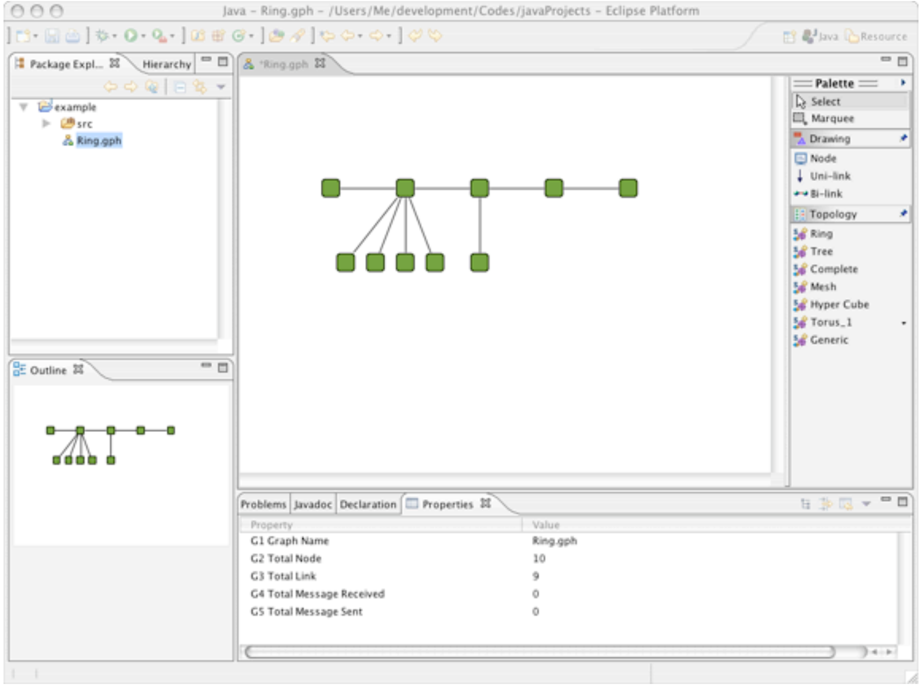
3.4.5.1 **A random rooted tree**- the user has to provide the number of nodes and the max number of children each node can have.



| Property | Value |
|---|---|
| G1 Graph Name | Ring.gph |
| G2 Total Node | 10 |
| G3 Total Link | 9 |
| G4 Total Message Received | 0 |
| G5 Total Message Sent | 0 |

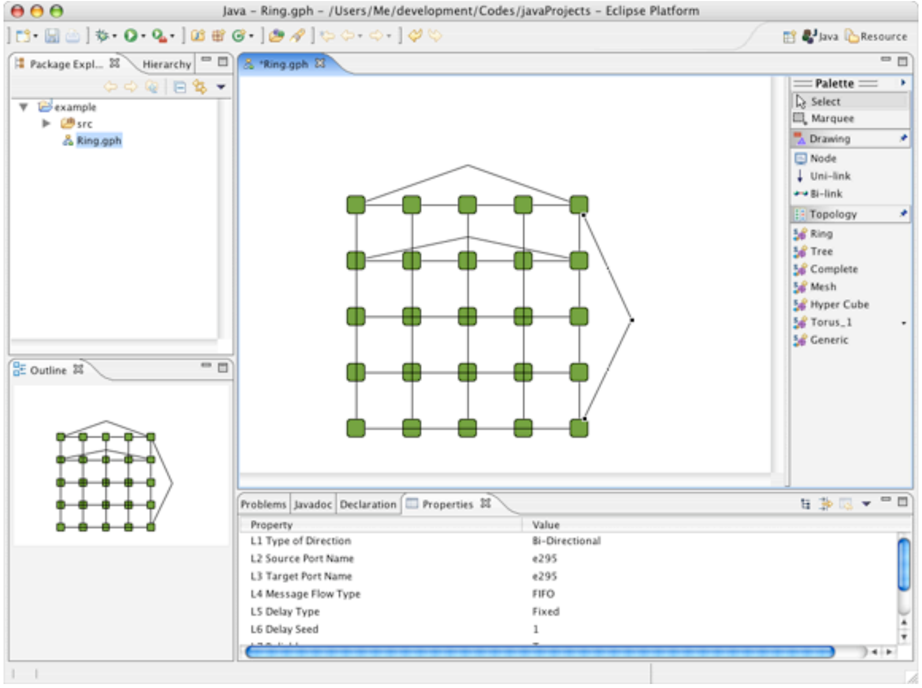
3.4.5.2 **A random tree**- the user has to provide the number of nodes and the diameter.

Franz Oppacher
**Comment:** Get a sample snapshot of a random tree

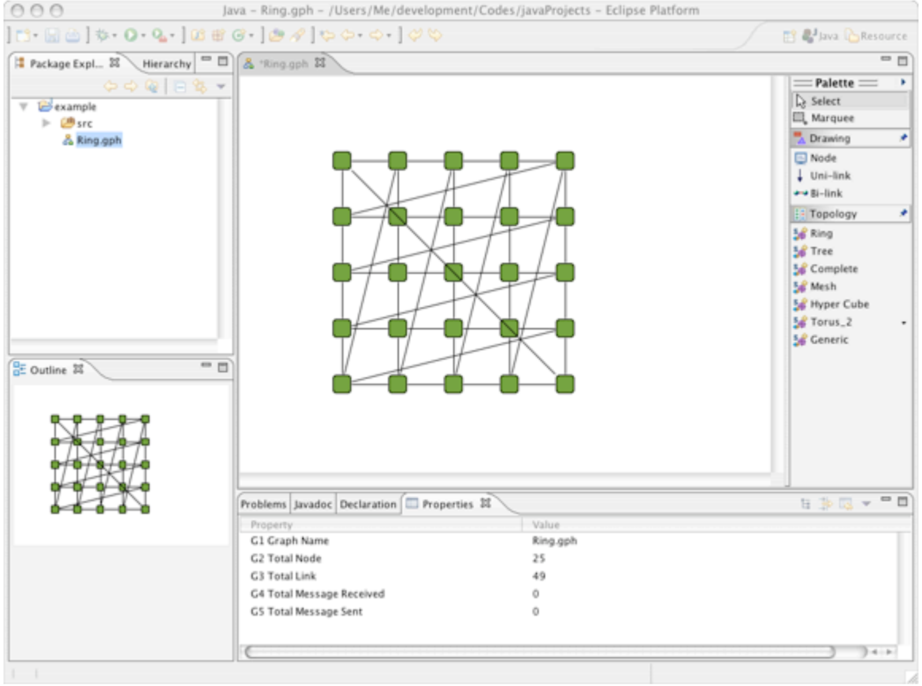


| Property | Value |
|---|---|
| G1 Graph Name | Ring.gph |
| G2 Total Node | 10 |
| G3 Total Link | 9 |
| G4 Total Message Received | 0 |
| G5 Total Message Sent | 0 |

### 3.4.6 Torus



| Property | Value |
|---|---|
| L1 Type of Direction | Bi-Directional |
| L2 Source Port Name | e295 |
| L3 Target Port Name | e295 |
| L4 Message Flow Type | FIFO |
| L5 Delay Type | Fixed |
| L6 Delay Seed | 1 |



| Property | Value |
|---|---|
| G1 Graph Name | Ring.gph |
| G2 Total Node | 25 |
| G3 Total Link | 49 |
| G4 Total Message Received | 0 |
| G5 Total Message Sent | 0 |

## 3.5 Summary

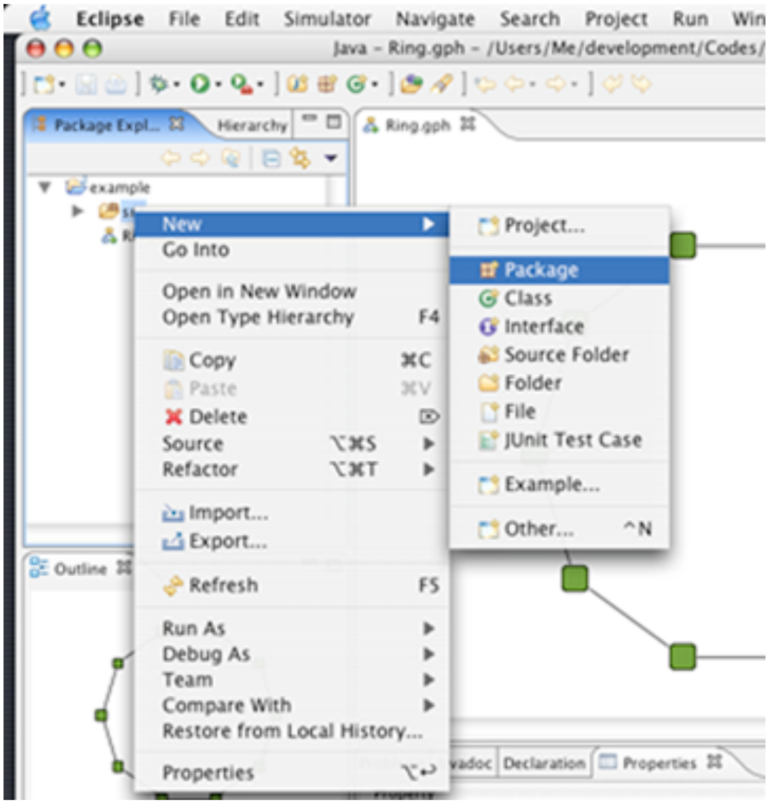
A user can create a graph by using a basic Draw objects or using Topology Library that provide a common readymade graph. Once a graph is created, there are some points about property setting and modifying a graph that user must keep in mind.

A graph that created by Topology Library can be modified like add/remove node(s) or link(s), adjust location of displaying nodes or links, also user can mix all different types of topologies into one graph and save it as a single file.

# Chapter 4 Writing a Protocol

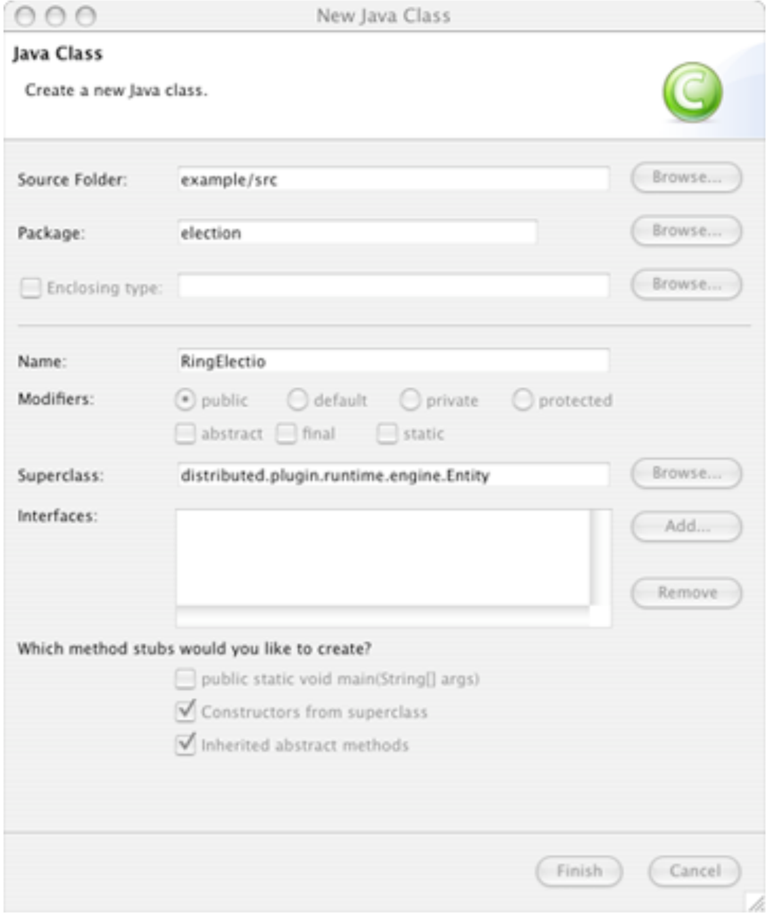
## 4.1 Creating a Protocol

Before creating a protocol file, the user has to create a Java package by right clicking at the project folder created in Chapter 3; then select *New->Package*; once a dialog shows up, assign a name, then click finish.



To create a java file; right click at the package, select *New->Class*; Once a dialog shows up, put a java Class name, select a proper Modifier and super Class and follow instructions.

Here is a snapshot of Java Class dialog.



New Java Class

Java Class

Create a new Java class.

Source Folder: example/src

Package: election

□ Enclosing type:

Name: RingElectio

Modifiers: ● public ○ default ○ private ○ protected

□ abstract □ final □ static

Superclass: distributed.plugin.runtime.engine.Entity

Interfaces:

Which method stubs would you like to create?

□ public static void main(String[] args)

☑ Constructors from superclass

☑ Inherited abstract methods

## 4.2 Rules and Conventions.

There are a couple of conventions for writing a distributed algorithm with DisJ that the user has to follow.

4.2.1 The entry class of the protocol must extend "**`distributed.plugin.runtime.engine.Entity`**" with an empty parameter Constructor , and it does not need "`public static void main(String[])`" (even if the user has one it will never be called)

4.2.2 State for Entity in distributed protocol must be "**`public static final int`**" or "**`public final int`**" and the name of the state must begin with "***state***" or "***_state***" case insensitive.

4.2.3 Assign an initial state to an entity by passing a value into "**`super(int initState)`**" within the constructor.

4.2.4 Follow Java rules and conventions.

There are three methods that user has to implement.

4.2.5 **`public void init():`**this method is invoked on a node if and only if the node property "*Initiator*" is set to be True (*see Chapter 3.3.2 and Chapter 6.1*)

4.2.6 **`public void alarmRing():`** this method is invoked when an alarm clock is rang. The user can set an alarm clock to activate the event by invoking method `setAlarm(short)`(*see Chapter 4.2.10*).

4.2.7 **`public void receive():`**,this method is invoked when a node receives a message.

DisJ library has provided standard programming interfaces for user to use in their reactive distributed algorithm s.

4.2.8 **`public void send():`**a common message sending interface family. This include **`sendTo()`**, **`sendtToAll()`** and **`sendToOthers()`** with different types of parameters argument.

4.2.9 **`public int getState():`** is for checking a current state of an entity;

4.2.10 **`public void become():`** is for setting a state of an entity to be at a given state;

4.2.11 **`public void setAlarmRing()`** is for setting an internal alarm clock to activated an event after a given time period.

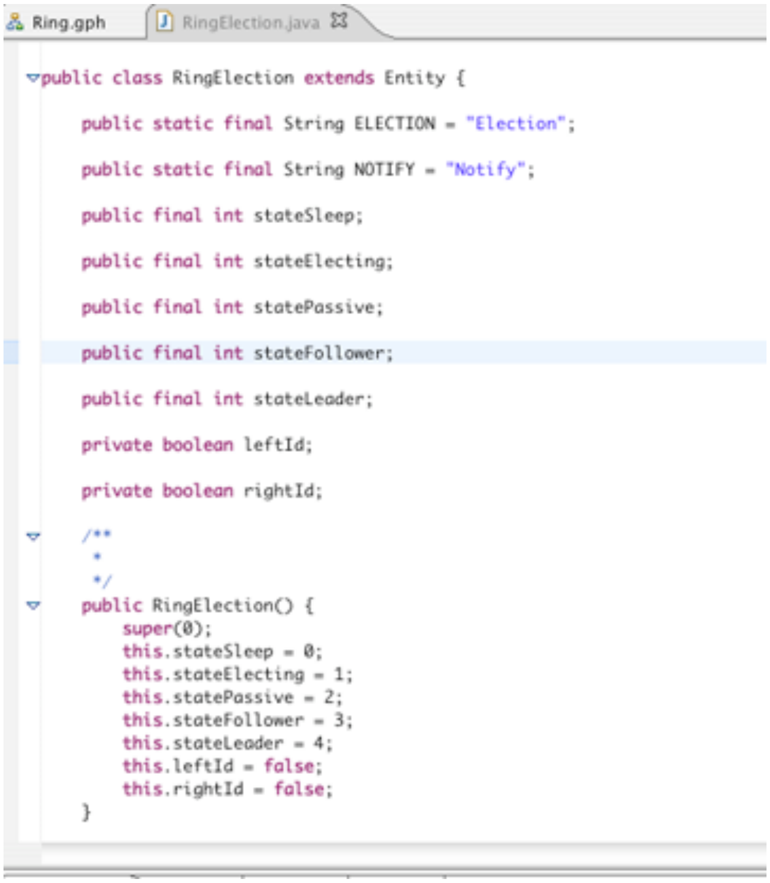
For full detail of available APIs can be found at DisJ Java API Documents.

## 4.3 Converting from pseudo code to Java algorithm

The convention for distributed algorithm in DisJ is based on reactive model defined in *state* x *event* → *action* (*see Appendix*)

This section will discuss how to convert "*state-event*" driven pseudo algorithm to "*even-state*" driven in Java algorithm. In order to make it easy to understand we still using **Ring Election protocol** "*As Far*" among the initiators to be an example.

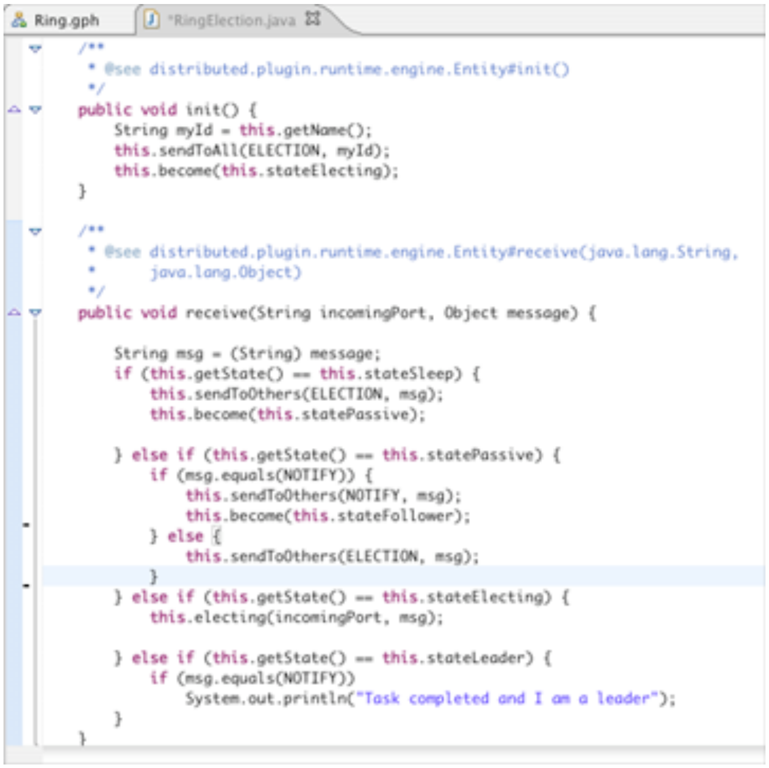
### 4.3.1 Snap shots from Eclipse JDT with an implementation of RingElection Class.



```
public class RingElection extends Entity {

    public static final String ELECTION = "Election";

    public static final String NOTIFY = "Notify";

    public final int stateSleep;

    public final int stateElecting;

    public final int statePassive;

    public final int stateFollower;

    public final int stateLeader;

    private boolean leftId;

    private boolean rightId;

    /**
     *
     */
    public RingElection() {
        super(0);
        this.stateSleep = 0;
        this.stateElecting = 1;
        this.statePassive = 2;
        this.stateFollower = 3;
        this.stateLeader = 4;
        this.leftId = false;
        this.rightId = false;
    }
```

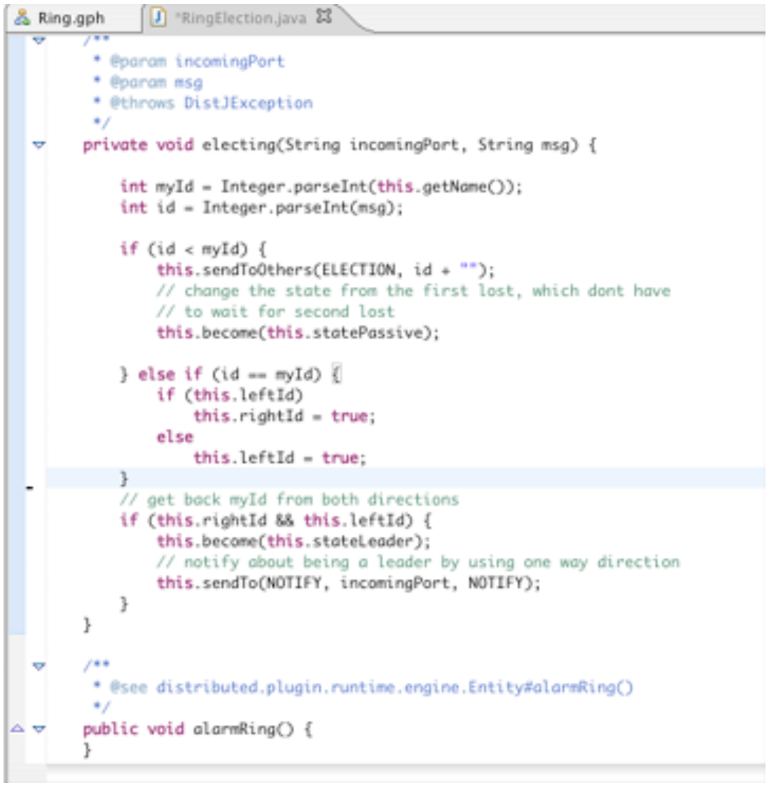
Sample1: Code sample for declaring state variables.

Here is how to make own helping method to follow Object Oriented best practice



```
/**
 * @see distributed.plugin.runtime.engine.Entity#init()
 */
public void init() {
    String myId = this.getName();
    this.sendToAll(ELECTION, myId);
    this.become(this.stateElecting);
}

/**
 * @see distributed.plugin.runtime.engine.Entity#receive(java.lang.String,
 *      java.lang.Object)
 */
public void receive(String incomingPort, Object message) {

    String msg = (String) message;
    if (this.getState() == this.stateSleep) {
        this.sendToOthers(ELECTION, msg);
        this.become(this.statePassive);

    } else if (this.getState() == this.statePassive) {
        if (msg.equals(NOTIFY)) {
            this.sendToOthers(NOTIFY, msg);
            this.become(this.stateFollower);
        } else {
            this.sendToOthers(ELECTION, msg);
        }
    } else if (this.getState() == this.stateElecting) {
        this.electing(incomingPort, msg);

    } else if (this.getState() == this.stateLeader) {
        if (msg.equals(NOTIFY))
            System.out.println("Task completed and I am a leader");
    }
}
```

Sample2: Code sample for receiving action



```java
    /**
     * @param incomingPort
     * @param msg
     * @throws DistJException
     */
    private void electing(String incomingPort, String msg) {

        int myId = Integer.parseInt(this.getName());
        int id = Integer.parseInt(msg);

        if (id < myId) {
            this.sendToOthers(ELECTION, id + "");
            // change the state from the first lost, which dont have
            // to wait for second lost
            this.become(this.statePassive);

        } else if (id == myId) {
            if (this.leftId)
                this.rightId = true;
            else
                this.leftId = true;
        }
        // get back myId from both directions
        if (this.rightId && this.leftId) {
            this.become(this.stateLeader);
            // notify about being a leader by using one way direction
            this.sendTo(NOTIFY, incomingPort, NOTIFY);
        }
    }

    /**
     * @see distributed.plugin.runtime.engine.Entity#alarmRing()
     */
    public void alarmRing() {
    }
```

Sample3: Code sample for a helping method

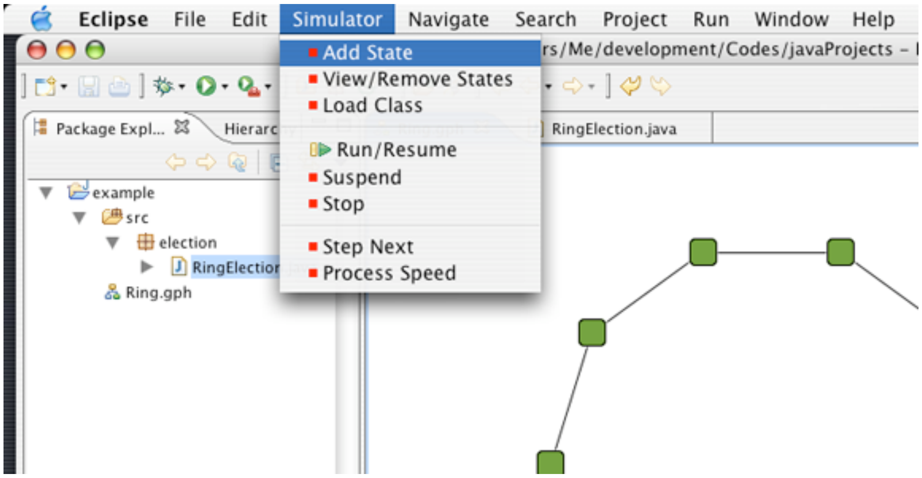
## 4.4 Summary

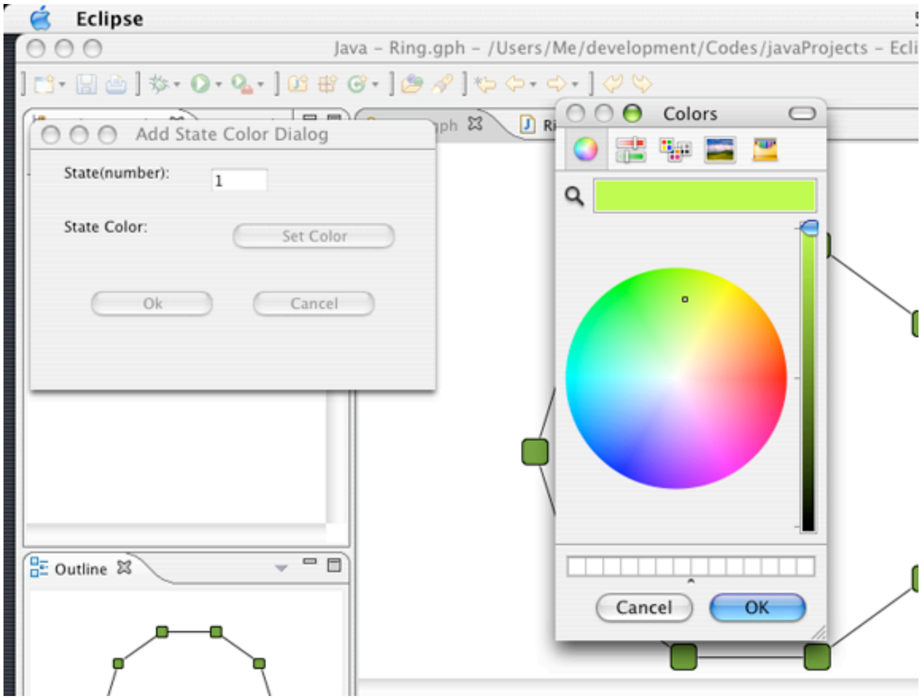
Once the user creates a java file for writing a distributed protocol, the user must follow the rules and conventions in order to run, test and simulate the protocol. The user can look an example and DisJ API Documents for further information. The next chapter (Chapter  5) will describe on how to put the graph and the protocol together.

# Chapter 5. Putting them together

## 5.1 Setting State Color

In order to see the state change of a node from the simulator;  the user must specify a color for each state that defined in the protocol (*see Chapter 4*) by go to menu *Simulator->Add State*; Once a dialog box shows up, give a value (number) that match to the state value in protocol  and select the color from a "*Color Palette*", the value will recorded in number-color pair; if a same state value are given twice, it will override the existing color.  Once the graph is saved, the states of color will be saved along the graph. Therefore, if the user wants to remove or see the list of state color pairs, go to menu *Simulator->Remove State*;  a dialog box will show a list of number-color pairs. To remove, check the checked box at a pair that the user wants to remove, then click Ok. Note that the simulator will ignore all the number-color pairs that do not match the states declared in protocol.  Therefore, only a match number- color pairs will be used during the simulation.

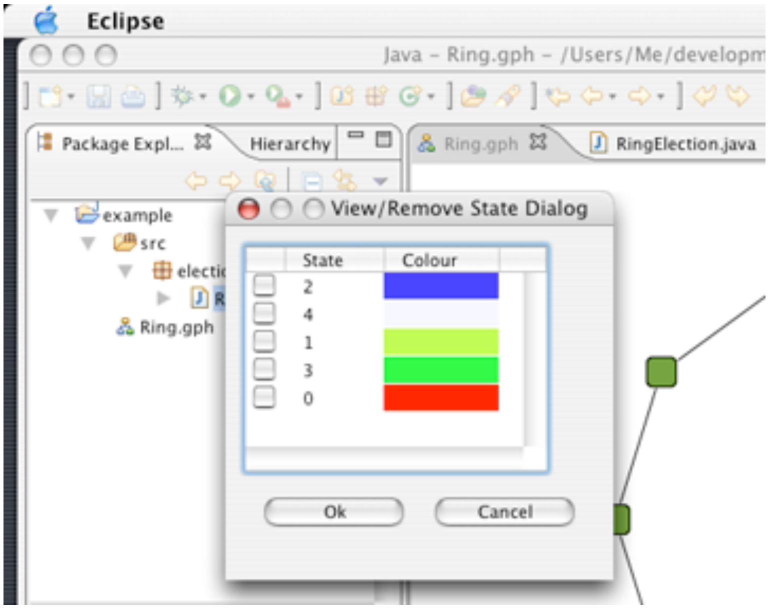




State(number): 1

State Color:

Here is Remove State Dialog



| | State | Colour |
|---|---|---|
| □ | 2 | 🟦 |
| □ | 4 | ⬜ |
| □ | 1 | 🟨 |
| □ | 3 | 🟩 |
| □ | 0 | 🟥 |

## 5. 2 Setting Node Properties

User can set the properties of each node by edit directly at "*Properties View*"(*see Chapter 3.3.2*)

5.2.1 **Breakpoint**: (not yet support).
5.2.2 **Initiator**: True means this node is an initiator otherwise False.
5.2.3 **Starter**: True means this node will activate when the simulation engine start.
5.2.4 **Node Name**: Any name can be given to this node and it can be duplicated. The user can retrieve the name of the node from a protocol API, `Entity.getName()`.
5.2.5 **Number of Message Received**: cannot be modified.
5.2.6 **Number of Message Sent**: cannot be modified.
5.2.7 **State List**: cannot be modified directly from "*Properties View*".

Franz Oppacher
**Comment:** Confusion between starter and initiator check the engin

## 5.3 Setting Link Properties

User can set the properties of each link by edit directly at "*Properties View*"(*see Chapter 3.3.3*)

5.3.1 **Type of Direction:** cannot be modified directly from "*Properties View*".

5.3.2 **Source Port Name:** a local outgoing port's name of a source node. If the link is bidirectional, the node has the same port's name for both incoming and outgoing ports on this link.

5.3.3 **Target Port Name:** a local incoming port's name of a destination node. If the link is bidirectional, the node has the same port's name for both incoming and outgoing ports on this link.

5.3.4 **Message Flow Type:** There are two types

- *First In First Out* (FIFO)- all message in and out are ordered by first in first out.
- *No Order*- the messages are sent out base on *Delay Seed* and it can be more than one message at a time.

5.3.5 **Delay Type:** There are three delay types

- *Fixed*- every message has a same waiting time with a length from a given *Delay Seed* property.
- *Uniform Random*- every message has different waiting time base on uniform random generator
- *Custom*- every message has a waiting time base on user plug-in library (not yet support)

Note that the orders of messages are based on the *Message Flow Type*.

5.3.6 **Delay Seed:** A positive integer value for the link to hold each message for that length of time.

Franz Oppacher
**Comment:** Check if it is a negative number and max length

5.3.7 **Reliable:** True means there is no losing of message in this link, otherwise a message might get lose base on a given probability from *Probability of Failure* property.

5.3.8 **Probability of Failure:** A positive integer value between zero to one hundred inclusive, [0-100], which is used to determine the probability of message lose on this link, if a *Reliable* property is set to False otherwise ignored.

5.3.9 **Total Traffic:** cannot be modified.

## 5.4 Summary

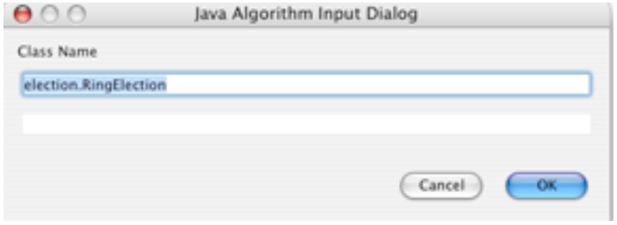
The user must set the properties of nodes and links in order to create a proper environment for a simulation. Especially, with a node's properties "*Initiator*" and "*Starter*" must set to *True* **at least** one node in order to activate the execution; by default "*Initiator*" is set to be *False* and "*Starter*" is set to be *True*.

# Chapter 6. Running a Simulation

There are few things that we have to know in order to run the simulator and it is described as follow.

## 6.1 Loading protocol into a Topology

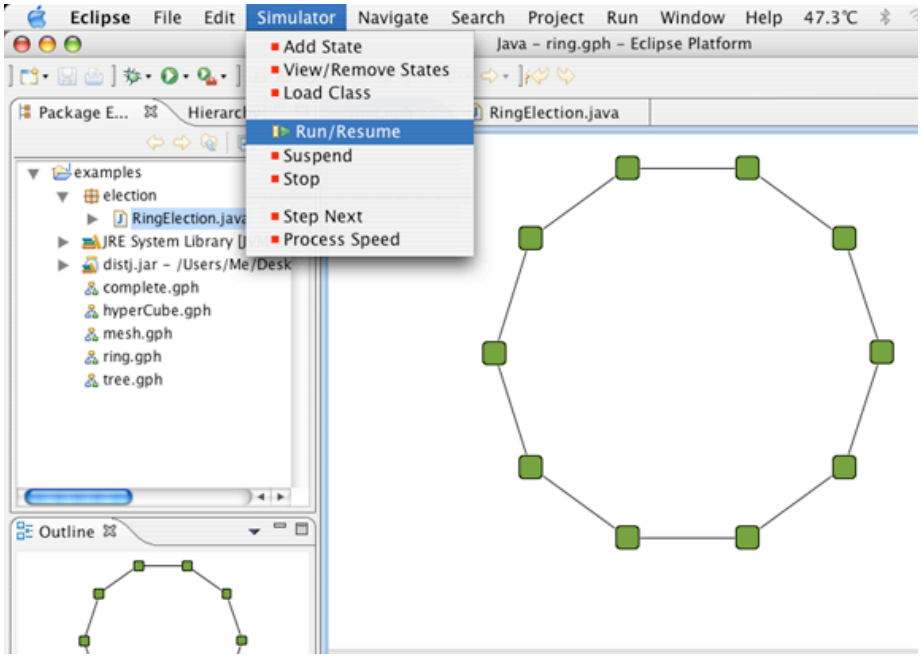
Once the topology and the protocol are defined;  To run the protocol in the protocol, user has to load the protocol into the topology by go to menu *Simulator->Load Class*; Once a dialog shows up, type in a *Fully Qualified Class Name* of the entry class (a class that implement `distributed.runtime.engine.Entity`), in this example is "`election.RingElection`" (*as show below*) then click OK. Remember that all the protocol classes must be in a same project folder as the graph (*.gph file*) and each graph are independent from each other, means loading the class into one topology into the workspace does not effect other topology.



Class Name

election.RingElection

Here, how to run from a menu bar.



## 6.2 Run/Resume the simulator

Once the class is loaded into the topology. Go to menu *Simulator->Run/Resume*, now DisJ is simulating the protocol. In order to simulate another protocol with a same topology, the user must reload a new protocol into the topology again, but if the user just modifies the protocol, then the use does not have to reload since it has a same class name.

## 6.3 Stop the simulator

Go to menu *Simulator->Stop*. After the simulator stopped all the current states are wipe out. Therefore, the user cannot see any state in the "*Property View"* anymore. Therefore, before rerun the simulator make sure the simulator is stopped before, otherwise the states will be accumulated.

## 6.4 Suspend the simulator

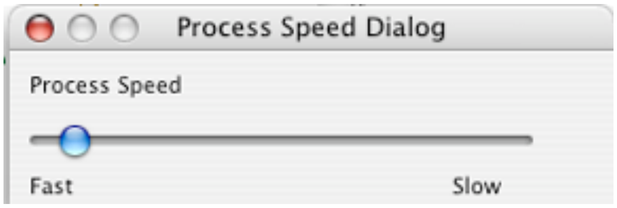
Go to menu *Simulator->Suspend* at anytime after class has been loaded and user can resume the execution by go to menu *Simulator->Run/Resume*. Remember that suspend does not wipe out the current states therefore, user can observe the current states in graph.

## 6.5 Step Next

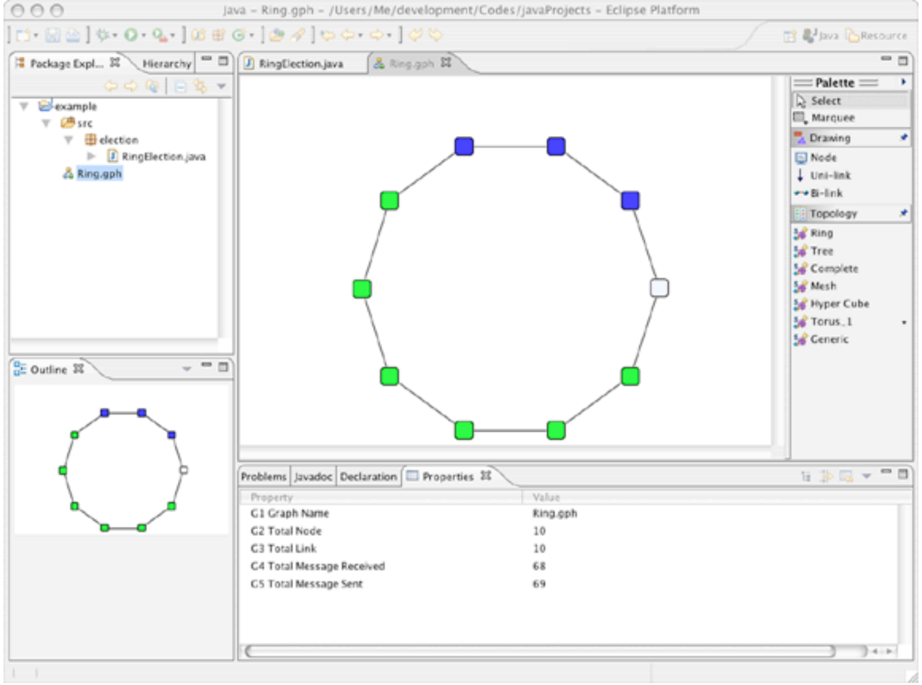
This feature is not yet support.

## 6.6 Process Speed

This feature allows user to setup the speed of the processor in execution the protocol, by go to menu *Simulator->Process Speed*; a dialog will show up as below snap shot.

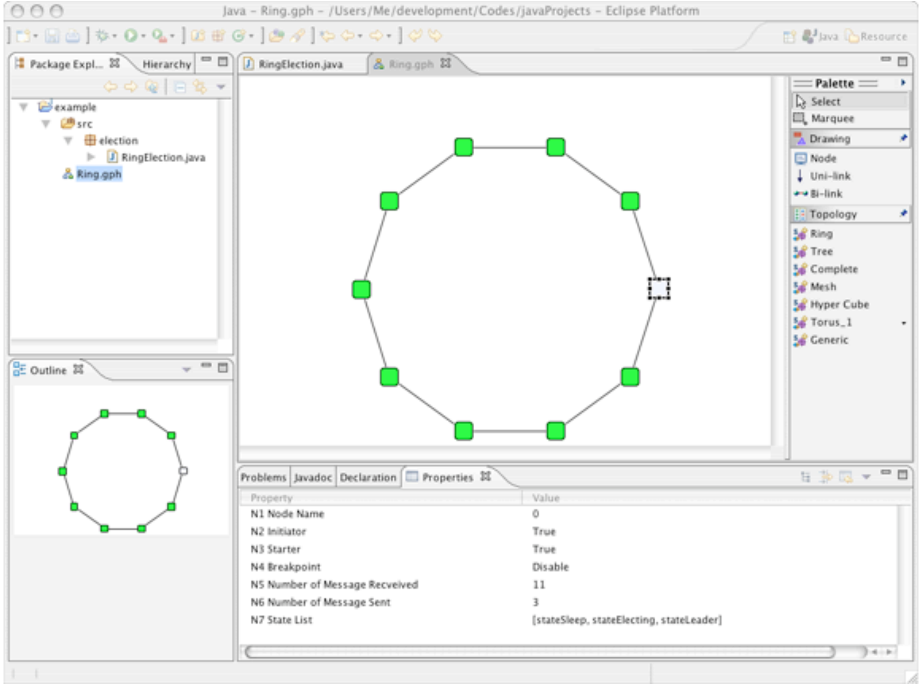


A snap shot when DisJ is executing



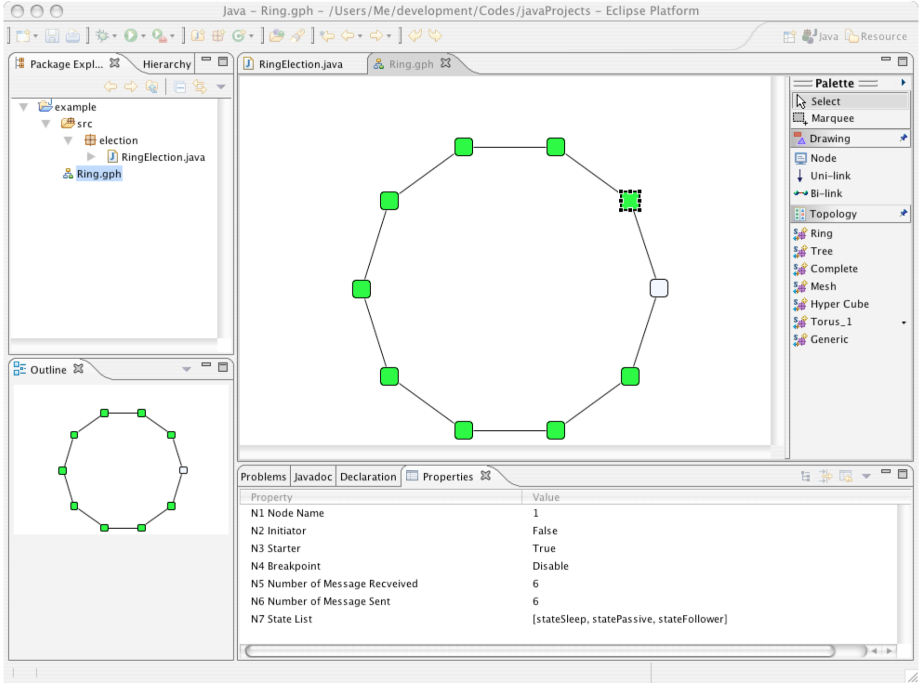
| Property | Value |
|---|---|
| G1 Graph Name | Ring.gph |
| G2 Total Node | 10 |
| G3 Total Link | 10 |
| G4 Total Message Received | 68 |
| G5 Total Message Sent | 69 |

Click at a node or link to see a current states



| Property | Value |
|---|---|
| N1 Node Name | 0 |
| N2 Initiator | True |
| N3 Starter | True |
| N4 Breakpoint | Disable |
| N5 Number of Message Recveived | 11 |
| N6 Number of Message Sent | 3 |
| N7 State List | [stateSleep, stateElecting, stateLeader] |

Once use execute stop all the states that has been changed are reset to the origin state color pairs



| Property | Value |
|---|---|
| N1 Node Name | 1 |
| N2 Initiator | False |
| N3 Starter | True |
| N4 Breakpoint | Disable |
| N5 Number of Message Recveived | 6 |
| N6 Number of Message Sent | 6 |
| N7 State List | [stateSleep, statePassive, stateFollower] |

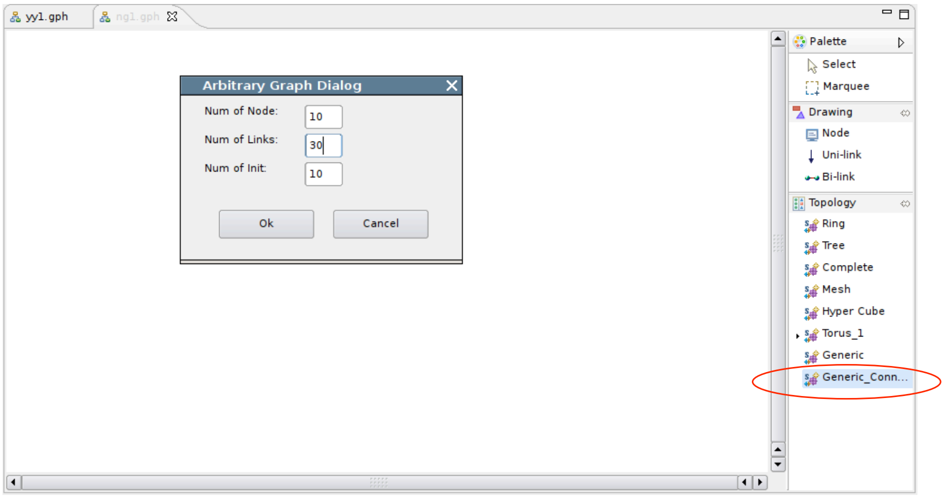
# Chapter 7. Conclusion

DisJ is a lightweight tool for writing, test and simulate the reactive distributed algorithm in Java™. DisJ provides virtual processors in a given network environment (graph) with graphical interaction that allows user to view and debug the protocol. Also it ease user from setting up sophisticate infrastructure while preserve (almost) reality of distributed environment. Moreover, DisJ provides a nice Java™ Development Tool (JDT) from Eclipse™, which helps user in writing and debugging a complex protocol with Java™ language.

# Chapter 8. New Features added in Dec 2008

### 1. Defining a connected graph with the number of nodes, links, and initiators

This feature allows users to generate a connected graph with specific number of nodes, links, and initiators.



Arbitrary Graph Dialog

Num of Node: 10

Num of Links: 30

Num of Init: 10

Figure1. Generic_Connected_Graph 1

Click the entry cycled with red link, an Arbitrary Graph Dialog window will pop up. After a user keys in the numbers, a graph will be generated like the below one:

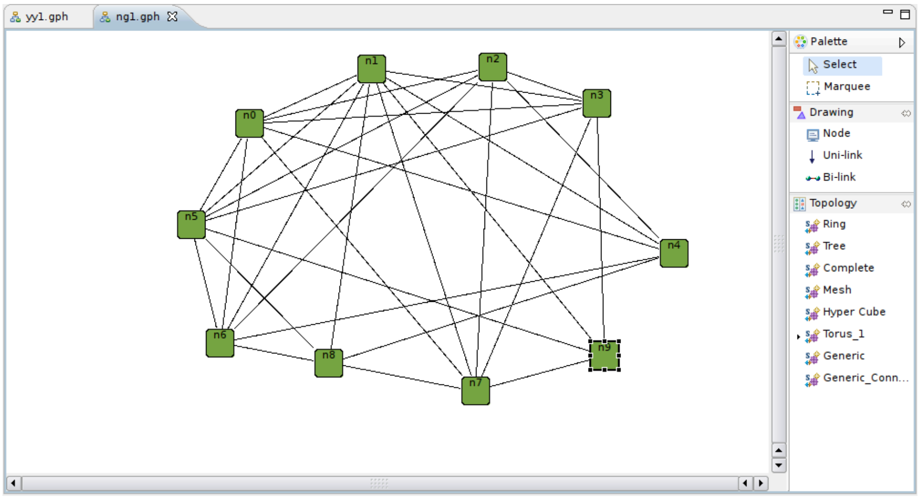
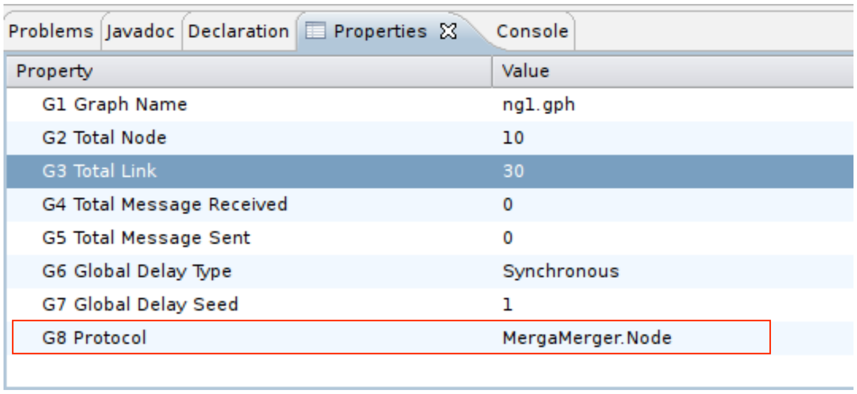


Figure2. Generic_Connected_Graph 2

### 2. Defining associated protocol in the properties sheet

This feature allows users to set a protocol to associate with a graph by double clicking the value field in the properties sheet. Once the association is set, it will always be there until users change it. It will save the effort to keep entering the protocol class name whenever a graph is opened.

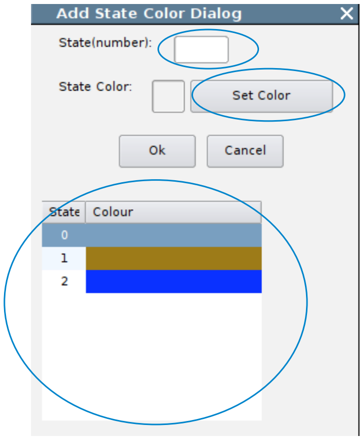


| Property | Value |
|---|---|
| G1 Graph Name | ng1.gph |
| G2 Total Node | 10 |
| G3 Total Link | 30 |
| G4 Total Message Received | 0 |
| G5 Total Message Sent | 0 |
| G6 Global Delay Type | Synchronous |
| G7 Global Delay Seed | 1 |
| G8 Protocol | MergaMerger.Node |

Figure3. Properties Sheet

### 3. New Add-State-Color-Dialog window

The new Add State Color Dialog window allows users to add multiple states at one time.



| State | Colour |
|---|---|
| 0 | |
| 1 | 🟫 |
| 2 | 🟦 |

Figure4. Add State Color Dialog

Input a state number in the text box, and then click the Set Color button to pick a color. The new input color will be added to the table at the bottom. After all the colors are set, click Ok button to save it.

### 4. Output to the Console

Now you can use “System.out.println()” to print information onto the console.

## 5. Step Forward on Control Panel

The control panel can be used for controlling the execution of protocols.

There four buttons on the menu bar are Start/Pause/Stop/Step Forward.
Step Forward button is used in this way: after an execution pauses, you can click Step Forward button to process steps in next time slot.

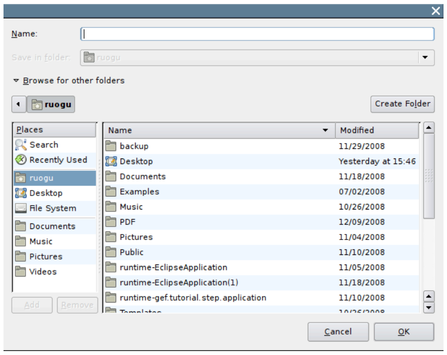
## 6. Playback an execution

### 6.1 Automatically save an execution

For example, if the file name of the graph is abc.gph, the default record file will be named abc.rec. After a protocol is finished, the file abc.rec will be automatically saved to the bin folder. Each execution will generate a new record file with the default file name. So if you want to keep a record file, you should rename it once it is generated.

### 6.2 Save an execution with a specific name

On the menu bar there is a save button. Click the button and a dialog window will pop up, there you can specify where to save the record file. The record file should be ending with an extension of .rec.



Name:

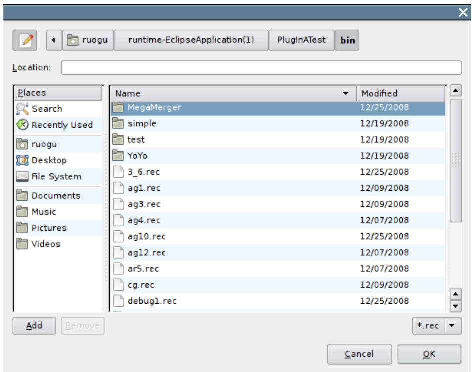
Save in folder: ruogu

| Name | Modified |
|---|---|
| backup | 11/29/2008 |
| Desktop | Yesterday at 15:46 |
| Documents | 11/18/2008 |
| Examples | 07/02/2008 |
| Music | 10/26/2008 |
| PDF | 12/09/2008 |
| Pictures | 11/04/2008 |
| Public | 11/10/2008 |
| runtime-EclipseApplication | 11/05/2008 |
| runtime-EclipseApplication(1) | 11/18/2008 |
| runtime-gef.tutorial.step.application | 11/10/2008 |

Figure5. Specify the location and name of a record file

### 6.3 Load a record file

Click the Load button on the menu bar , a dialog window will pop up, and you can select the record file and load it in. The record file must be ending with an extension of .rec.



Location:

| Name | Modified |
|---|---|
| MegaMerger | 12/25/2008 |
| simple | 12/19/2008 |
| test | 12/19/2008 |
| YoYo | 12/19/2008 |
| 3_6.rec | 12/25/2008 |
| ag1.rec | 12/09/2008 |
| ag3.rec | 12/09/2008 |
| ag4.rec | 12/07/2008 |
| ag10.rec | 12/25/2008 |
| ag12.rec | 12/07/2008 |
| ar5.rec | 12/07/2008 |
| cg.rec | 12/09/2008 |
| debug1.rec | 12/25/2008 |

*.rec

Figure6. Specify the location and name of a record file

### 6.4 Control panel switching

Control panel can be used to control both execution of protocols and playback. By default, the control panel is for controlling execution of protocol. But if you open a record file through clicking , the control panel will switch to control playback; after you load a protocol through clicking , the control panel will switch back to control the execution of protocol.

## 7. Change a link's visibility during execution

It is used to clean some “useless” links in the execution process.
Usage: invoke setLinkVisibility (String port_name, Boolean visibility) in class Entity
Constraint: port_name must be one of Entity's ports
Example : if node n1 has a port of e1, then when you can make the link from e1 invisible by invoking n1's instance method *setLinkVisibility(e1,false)*.

## Appendix

The protocol is expressed as a set of rules of the form "*state* x *event* → *action*", according to the reactive model defined in DADA.

In the convention used in DADA, rules are grouped by *state*, listing the difference "*event* → *action*" for that state.

**Example 1:**

A Protocol "***As Far***", which elect a leader among the initiators in a bidirectional link Ring.

```
States: S = {ASLEEP, ELECTING, PASSIVE, FOLLOWER, LEADER}
        S(init) = {ASLEEP}
        S(terminate) = {FOLLOWER, LEADER}

Restrictions: RI U Bidirectional Ring

ASLEEP:
        Spontaneously
        begin
                send ("Election", id(x)) to all;
                become ELECTING;
        end

        Receiving("Election",  id)
        begin
                send ("Election", id) to others;
                become PASSIVE;
        end

ELECTING
        Receiving("Election", id)
        begin
                if id < min then
                        send("Election", id) to others;
                        become PASSIVE;

                else if id = min from both directions then
                        send("Notify") to all;
                        become LEADER;

                endif
        end
```

```
  PASSIVE
        Receiving("Election", id)
        begin
              send ("Election", id) to others;
        end

        Receiving("Notify")
        begin
              send("Notify") to others;
              become FOLLOWER;
        end

LEADER
        Receiving("Notify")
        begin
              done election
        end
```

In DisJ, the dual convention is used grouping the rules by *event*, listing the difference “*state→action*” for that event.

**Example 2:**

The same protocol from *Example1*, expressed using the dual convention.

```
States: S = {ASLEEP, ELECTING, PASSIVE, FOLLOWER, LEADER}
        S(init) = {ASLEEP}
        S(terminate) = {FOLLOWER, LEADER}

Restrictions: RI U Bidirectional Ring

SPONTANEOUSLY:
        Asleep
        begin
              send ("Election", id(x)) to all;
              become ELECTING;
        end


RECEIVING
        Asleep ("Election", id)
        begin
              send ("Election", id) to others;
              become PASSIVE;
        end
```

```
  Passive (“Election”, id)
begin
      send (“Election”, id) to others;
end

Passive (“Notify”, msg)
begin
      send (“Notify”, msg) to others;
      become FOLLOWER
end

Electing (“Election”, id)
begin
      if id < min
        send(“Election”, id) to others;
        become PASSIVE

      else if id = min from both directions then
            send(“Notify”) to all;
            become LEADER;

      endif
end

Leader(“Notify”)
begin
      done election
end
```

**Example 3:**

The same protocol from *Example2*, expressed using Java Code.

```
// all possible states
public final int stateSleep = 0;
public final int stateElecting = 1;
public final int statePassive = 2;
public final int stateFollower = 3;
public final int stateLeader = 4;

private boolean leftMin;
private boolean rightMin;

public RingElection(){
   super(this.stateSleep);
}

public void init(){
   String myId = this.getName();
```

```
 this.sendToAll(“Election”,  myId);
 this.become(this.stateElecting);
}

public void receive(String incomingPort, Object message) {

   String msg = (String) message;
   if (this.getState() == this.stateSleep) {
        this.sendToOthers("Election", msg);
        this.become(this.statePassive);

   } else if (this.getState() == this.statePassive) {
           if(msg.equals("Notify")){
               this.sendToOthers("Notify",msg);
               this.become(this.stateFollower);
            }
           this.sendToOthers("Election", msg);

   } else if (this.getState() == this.stateElecting) {
              this.electing(incomingPort, msg);

   } else if( this.getState() == this.stateLeader) {
         if(msg.equals(“Notify”))
               System.out.println(“Election completed”);
   }
}

private void electing(String incomingPort, String msg) {

    int myId = Integer.parseInt(this.getName());
    int id = Integer.parseInt(msg);

    if(id < myId){
         this.sendToOthers(“Election”,id+””);
         this.become(this.statePassive);

    } else if (id == myId) {
         if(this.leftId == true)
           this.rightId = true;
         else
           this.leftId = true;

    }

   if(this.rightId && this.leftId) {
      this.become(this.stateLeader);
      this.sendTo(“Notify”, incomingPort, “Notify”);
   }
}
```