

School of Computer Science
Carleton University

DisJ Cookbook (draft)
(DisJ version 1.1.0)

By: Rody Piyasin
Supervisor: Prof. Nicola Santoro

Revision Date: February 10, 2011

Table of Contents

Chapter 1. Overview.....	3
Chapter 2. Preliminaries.....	4
Chapter 3. Creating a Topology.....	7
Chapter 4 Creating a Protocol.....	29
Chapter 5. Running a Simulation.....	40
Chapter 6. Graphical and Statistics Reports.....	46
Chapter 7. Conclusion.....	49
Appendix.....	50

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 a protocol in Java language in Eclipse JDT
- 4 Execute the protocol in the defined topology using DisJ

Throughout the cookbook we will use two an example protocols; first protocol is “*As Far*” that represent message passing model for leader election in a ring with bidirectional links network, and second protocol is “*Black Hole Search*” that represent mobile agent with token model for counting number of nodes in a ring with bidirectional link network (see Appendix)

Chapter 2. Preliminaries

2.1 Install Eclipse

The DisJ is an Eclipse™ plug-in, therefore we need to have Eclipse™ (version 3.0 or higher) installed. Eclipse is an open platform built by an open source community, which it can be downloaded at <http://www.eclipse.org/downloads/>. There are many types of builds available, but we recommend “*Eclipse Classic*” or “*Eclipse IDE for Java Developer*”.

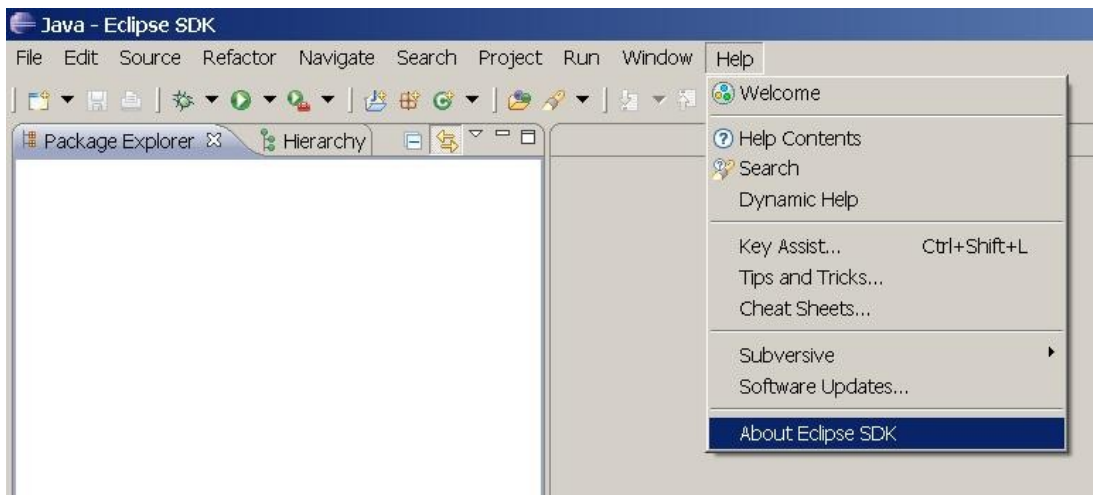
2.2 Install GEF libraries

Since DisJ graphical interface parts built on top of Graphical Editor Framework Plug-in (GEF), therefore, DisJ plug-in requires GEF Plug-in to be installed. GEF runtime library can be downloaded at <http://www.eclipse.org/gef/downloads/index.php>. There are many types of packages release, but we recommend “*All-In-One*” that includes Draw 2D, GEF and Zest.

2.3 Install DisJ

The installation of DisJ is similar to any others Eclipse Plug-ins by unzip the file and place the folder “*org.carleton.scs.disJ_1.1.0*” into *ECLIPSE_HOME/plugins* folder.

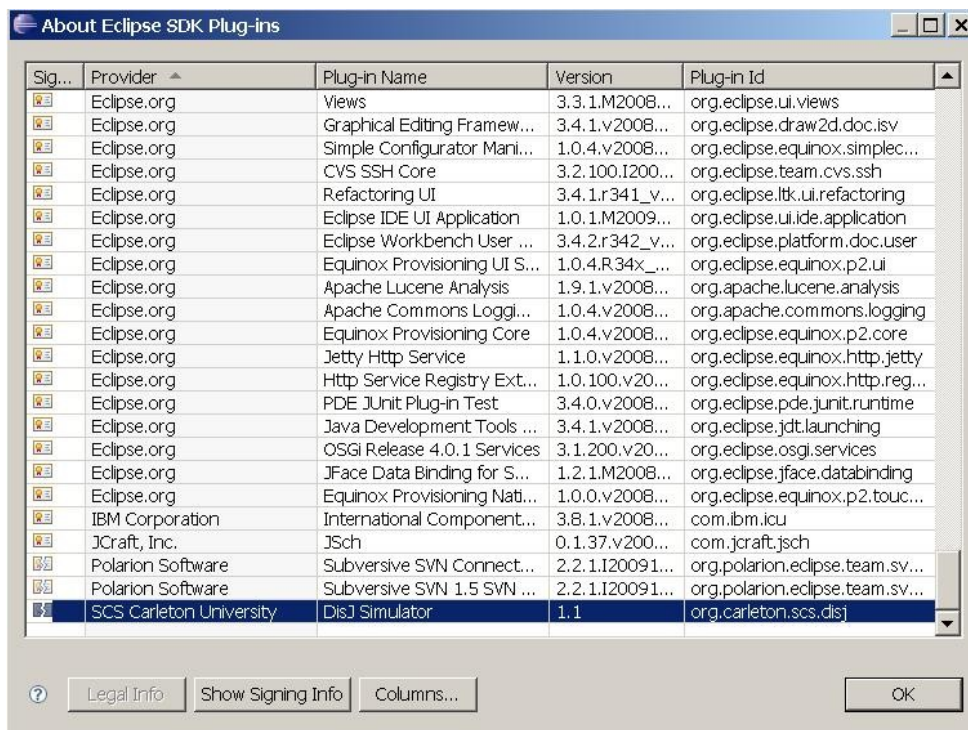
To make sure that the plug-in was successfully installed, restart Eclipse™. Go to the menu *Help* → *About Eclipse SDK*



Then a dialog box shown below must pop up on the screen




Then click on “*Plug-in Details*” button and a dialog with list of current installed plug-ins of the Eclipse will be listed which included “*DisJ Simulator*” (if the installation has been done correctly) as shown in the picture below.

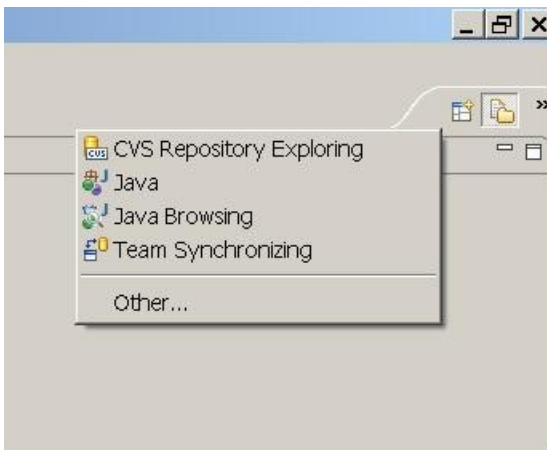


Note that any Plug-ins that has been installed into Eclipse successfully, the name of the Plug-ins will be listed in this dialog box, which include our GEF Plug-ins as well. Therefore, this is a good place to check and verify installed Plug-ins in your Eclipse.

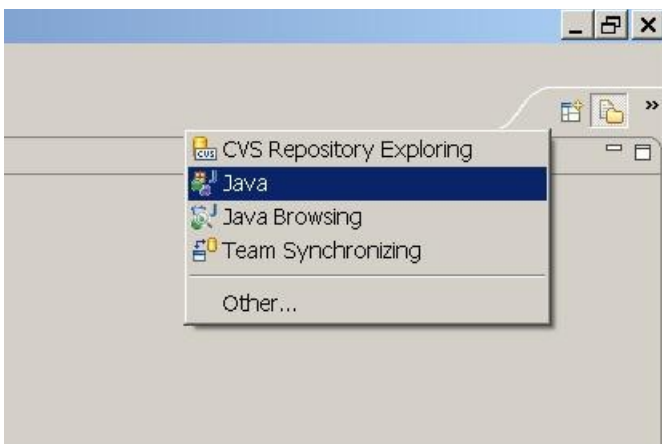
2.4 Changing Perspective

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 will automatically forces user to follow the rules and best practices in software development with Java.

Click at  icon at a top of your right hand to the list of available perspectives (as shown below).



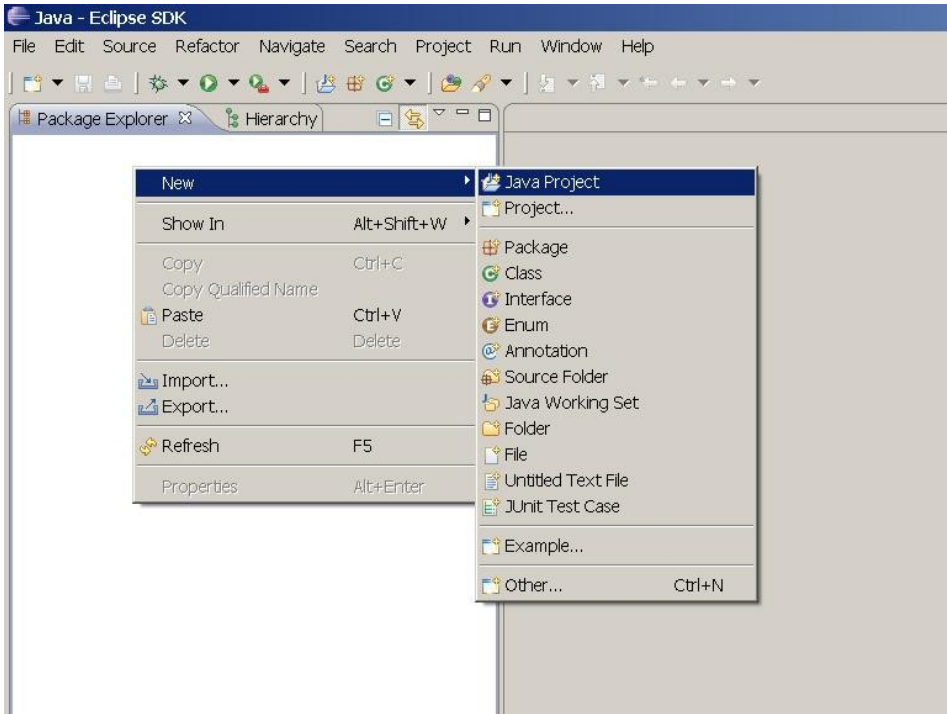
Then select Java perspective as shown below



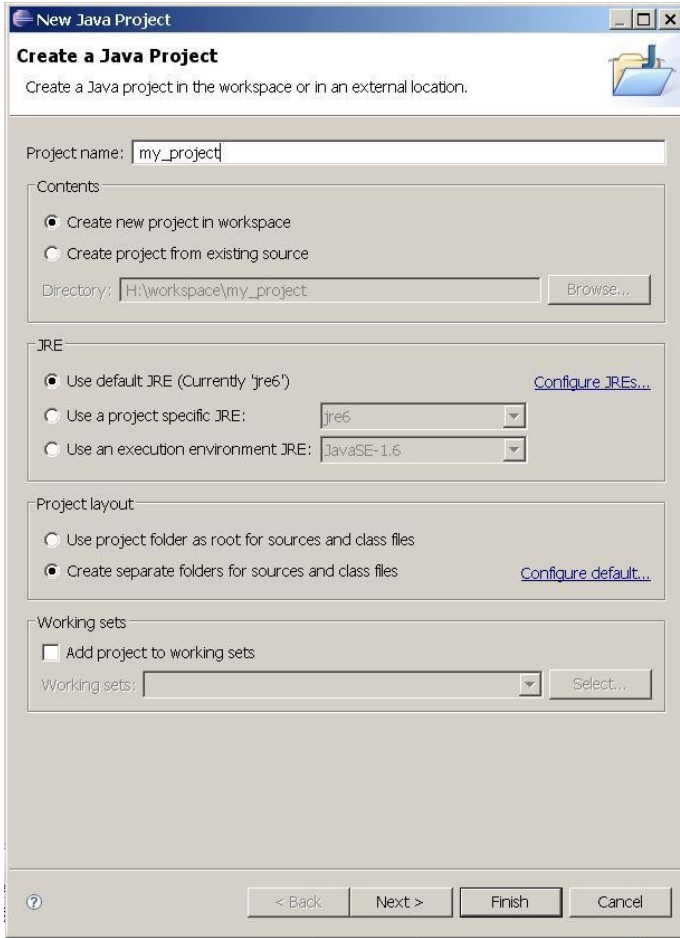
Chapter 3. Creating a Topology

3.1 Creating a Java project

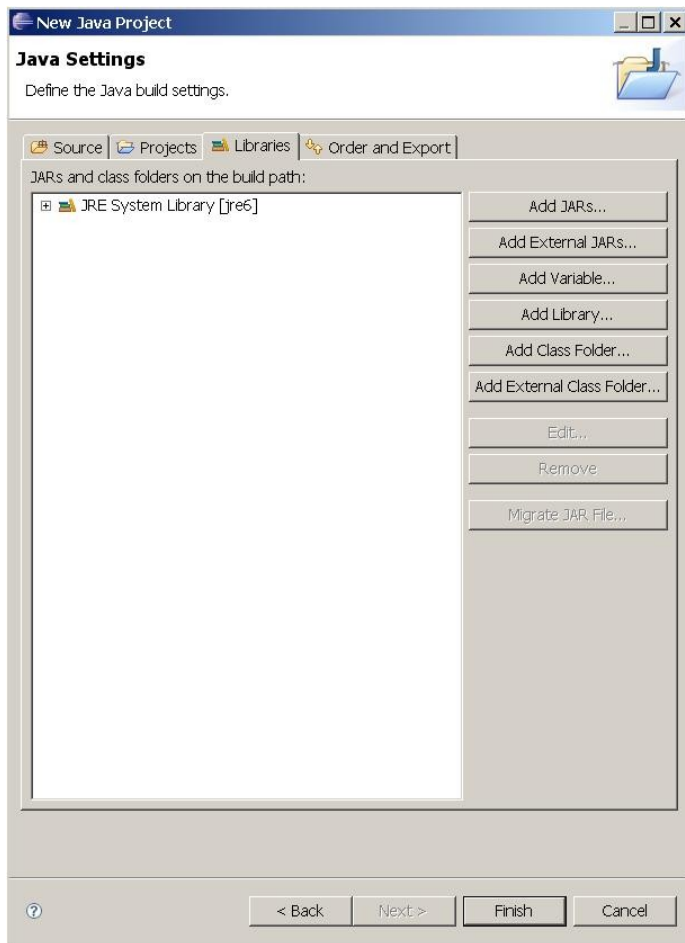
For a best practice in software development, we need to create a Java Project first by right click at “*Package Explorer*” view, then select *New* → *Java Project*



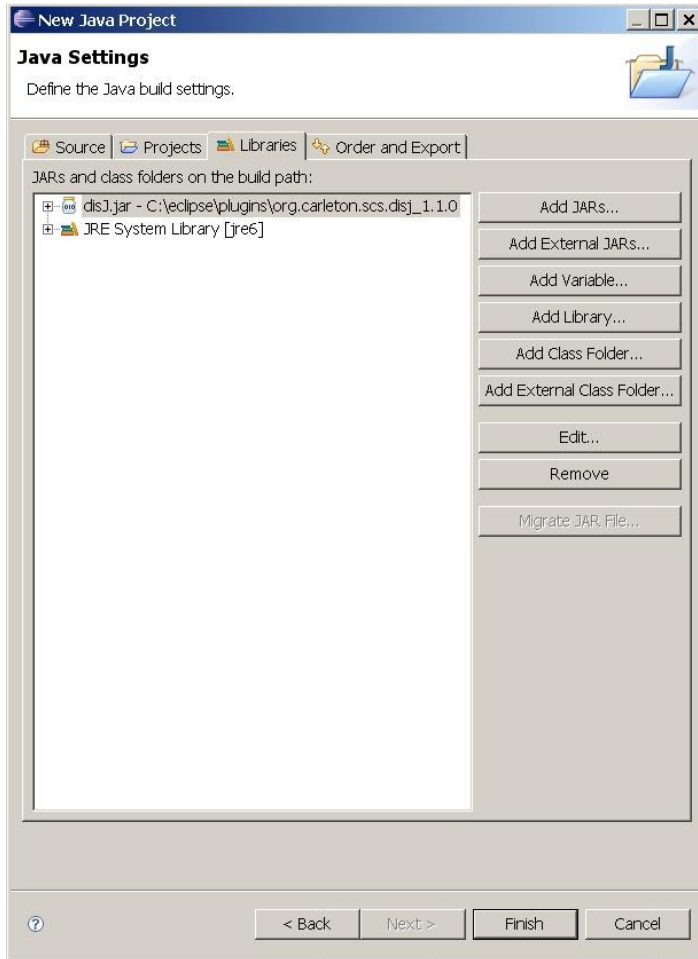
The project creation wizard dialog box will pop up as below and type the project name in lower case (for a best practice)



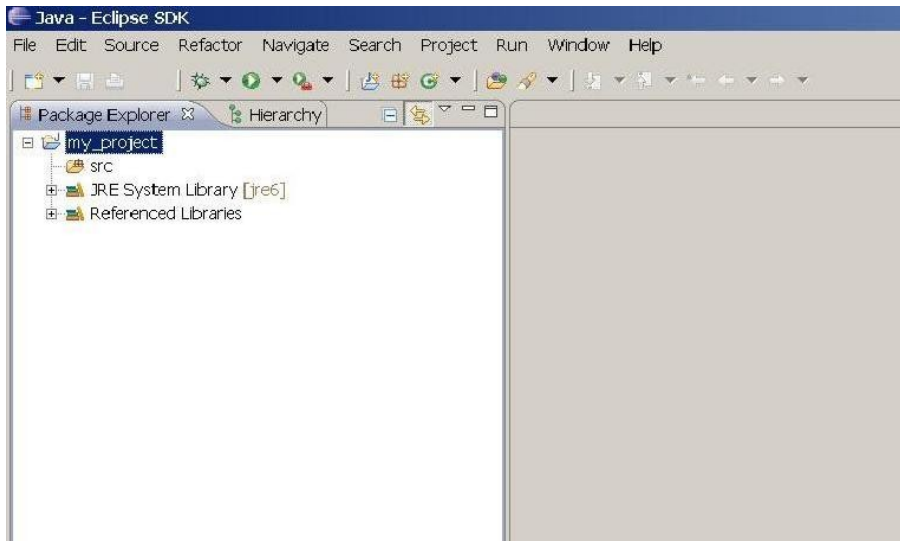
Leave the default setting as it is then click *Next* → *select Libraries* tab



In order to add “DisJ” runtime libraries and API for developing algorithm. So we will add “disJ.jar” which located at *ECLIPSE_HOME/plugins/org.carleton.scs.disj_1.1.0/* by clicking at “Add External JARs...” button and select disJ.jar from above said location. The dialog box will look like the following. Then click “Finish” button.

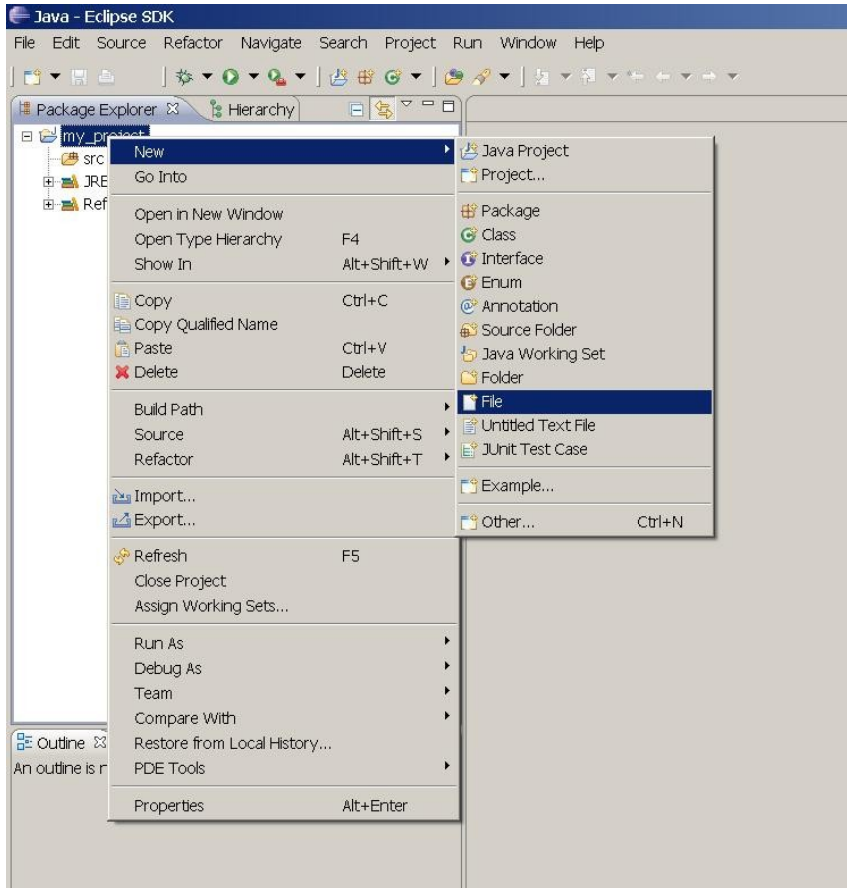


The workbench under Java Perspective will look like the following.



3.2 Creating a Graph File (.gph)

Right click at our Java Project in “*Package Explorer*”, and select “*New* → *File*” (see below);

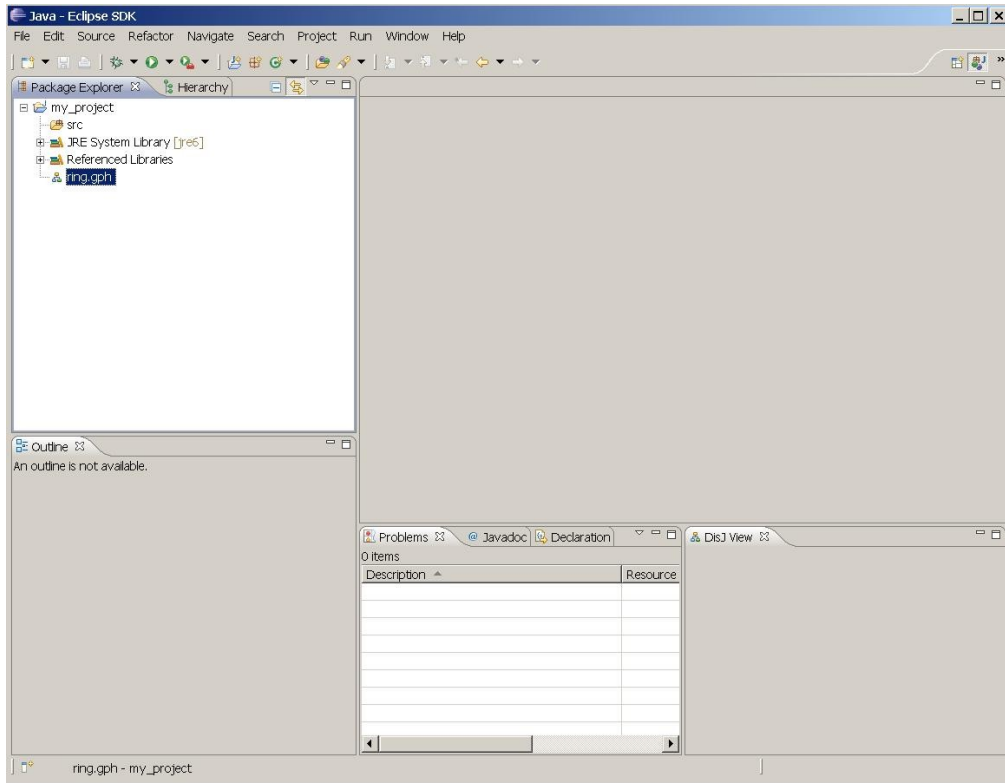


The a file creation dialog pop up as shown below, then fill up a file name with extension file type “.gph”

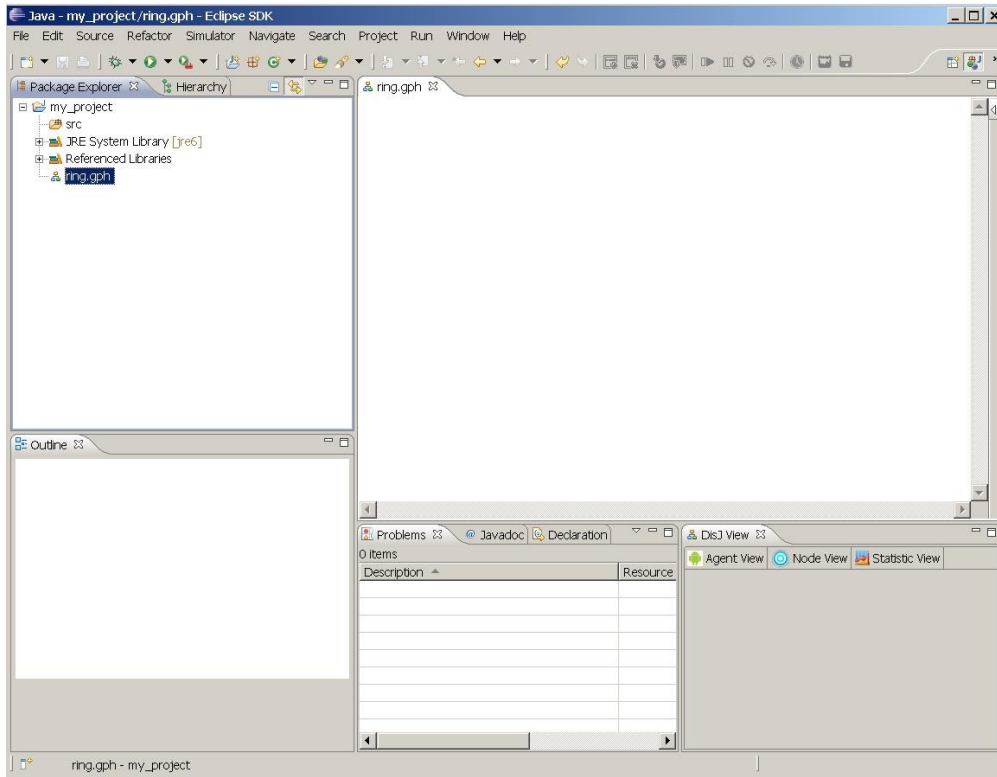


Then click “Finish”.


The workbench will look like the following

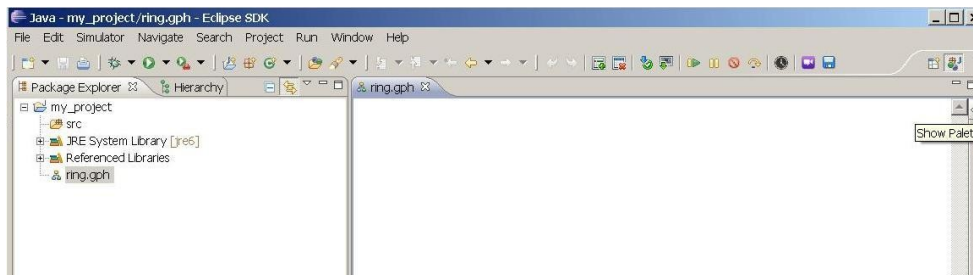


Then double click our new graph file, ring.gph, which will open up “DisJ Graph Editor” into the workbench and look like the following

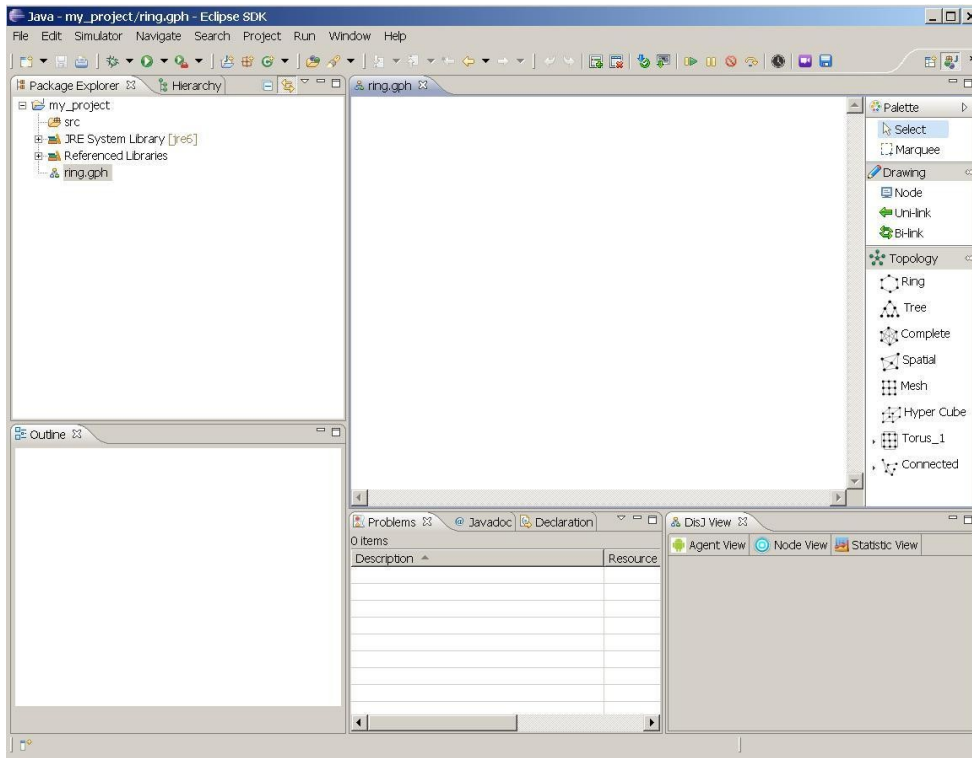


3.3 Creating a graph in DisJ Graph Editor

The DisJ Graph Editor shows up on the editor view. We start drawing by expanding the “*Palette*” on a top right corner of the editor tab by clicking at icon 



And it looks like the following

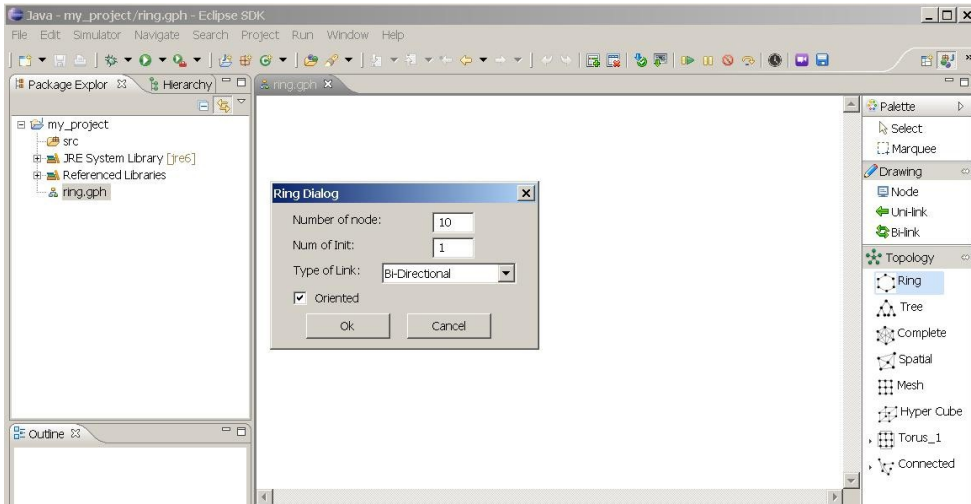


There are two ways to define a graph:

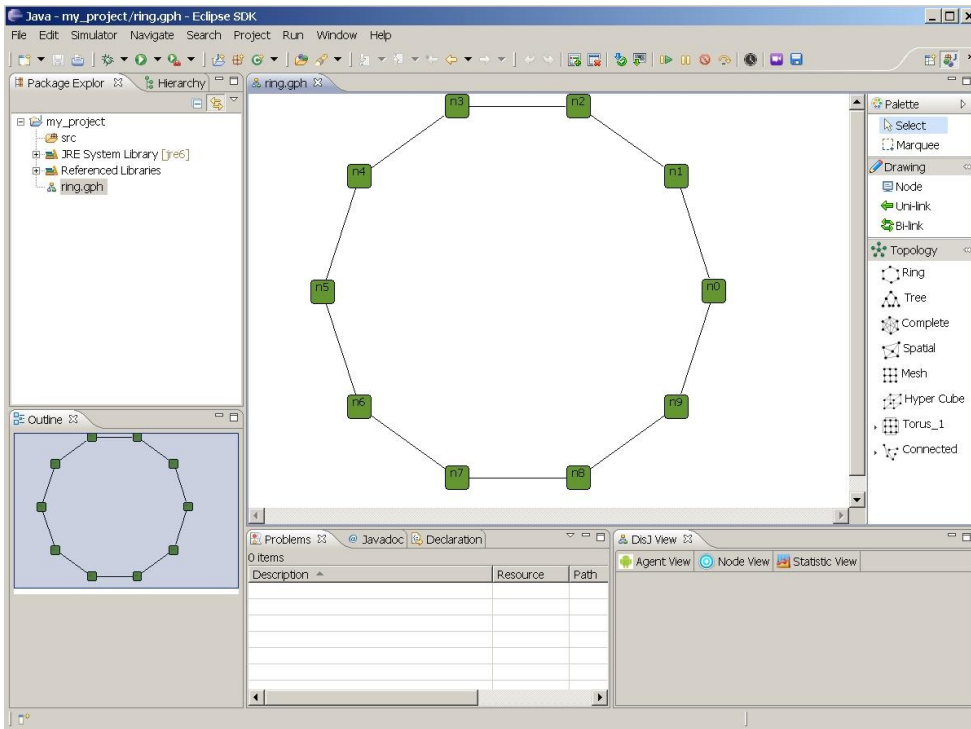
- 3.3.1 Draw** - this is a basic drawing tool based on simple objects such as node, uni-directional link and bi-directional link.
- 3.3.2 Topology Library**- this provides a ready-made set of topologies that can be edited such as Ring, Tree, Spatial, Complete, Mesh, Hyper Cube, Arbitrary and Torus.

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.

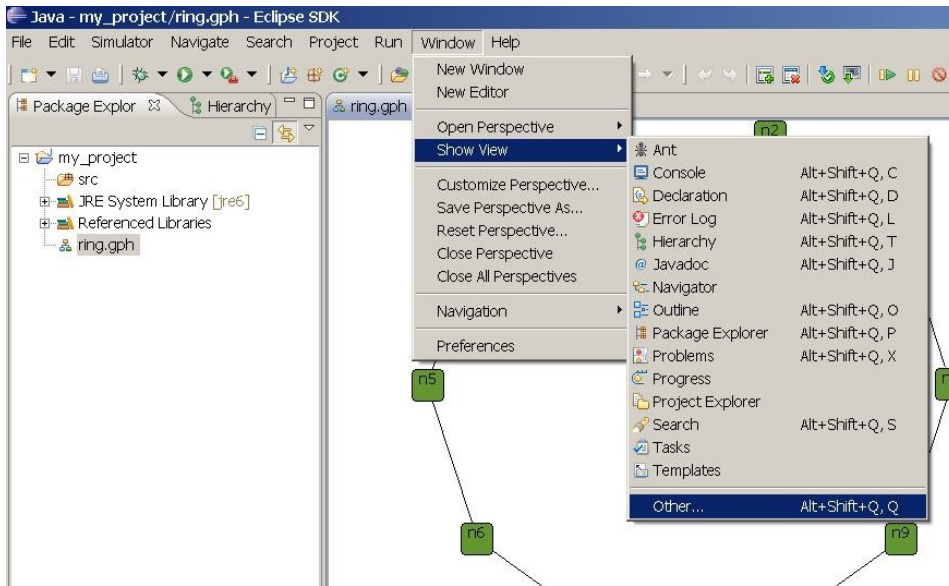
Here is an example of defining a bidirectional Ring with 10 nodes using the Topology Library.



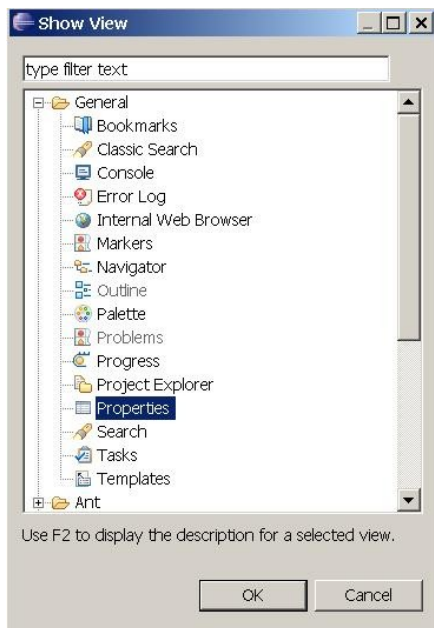
Here is the snapshot when a Ring of size 10 is created.



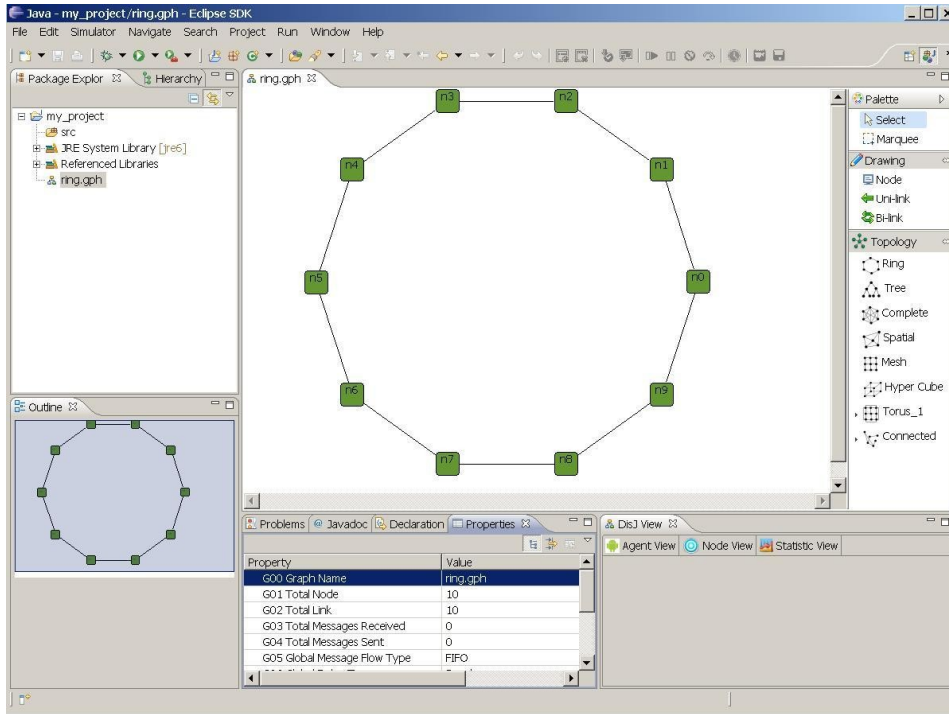
Once, the graph has been drawn, you can check if there is the correct number of links and nodes by opening a “*Properties View*” which will show the current states and properties. To open the “*Properties View*”, go to menu bar, select *Window->Show View->Properties*; if it does not list, select *Window->Show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



Once the dialog shows up, expand a “*General*” 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 in later section, the following is snapshot of our workspace with properties view.



3.4 Element Properties

3.4.1 Graph Properties

Property	Value
G00 Graph Name	ring.gph
G01 Total Node	10
G02 Total Link	10
G03 Total Messages Received	0
G04 Total Messages Sent	0
G05 Global Message Flow Type	FIFO
G06 Global Delay Type	Synchronous
G07 Global Delay Seed	1
G08 Protocol	
G09 Maximum Token Agent can Carr	1
G10 Number of Agent at Start	0
G11 Current Number of Agent	0

3.4.1.1 Graph Name (not editable)

This is the name of the file where the graph has been defined.

3.4.1.2 **Total Node**

Total number of nodes in this graph

3.4.1.3 **Total Link**

Total number of links in this graph

3.4.1.4 **Total Messages Received**

Total number of messages that have been received from the start of the execution of the protocol upto now

3.4.1.5 **Total Messages Sent**

Total number of messages that have been sent from the start of the execution of the protocol upto now

3.4.1.6 **Global Message Flow Type**

Flow type of every links in this graph, there are 2 types, FIFO and No Order. If one or more link has difference type than others in the graph, this field will be set to “Mix Order”

3.4.1.7 **Global Delay Type**

Type of traveling delay of every links in this graph, there are 5 types, Synchronous, Random Uniformed, Random Poisson, Random Customs, Customs. If one or more link has difference type than others in the graph, this field will be set to “Custom”

3.4.1.8 **Global Delay Seed**

A number of simulation time unit required on each link in the graph. This value will be used only when the delay type is Synchronous. If one or more link has difference value than others in the graph, this field will be set to -1 (negative value)

3.4.1.9 **Protocol**

A name of protocol that currently being executed in this graph

3.4.1.10 **Maximum Token Agent Can Carry**

A maximum number of every agent in this graph can carry token. This will be used only in Agent with Token Model simulation

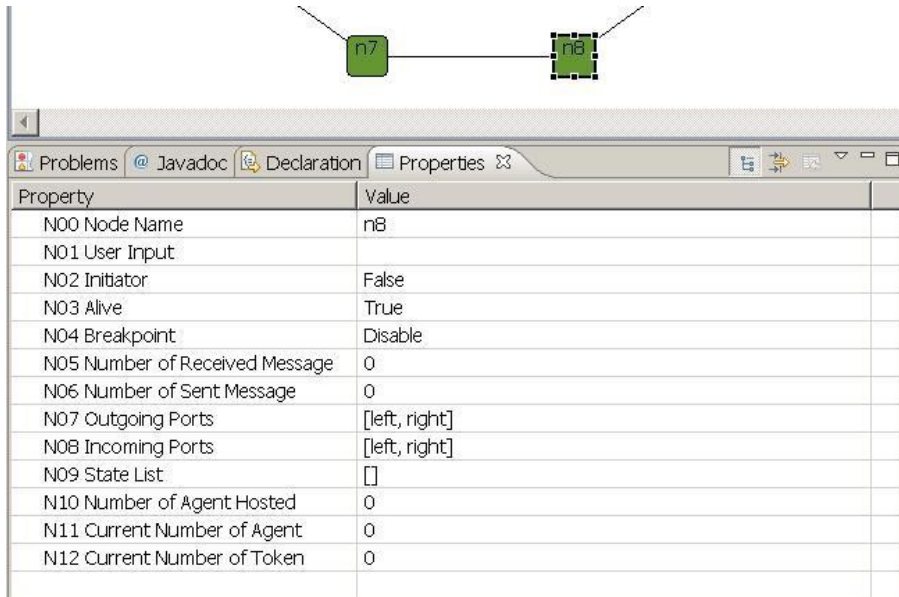
3.4.1.11 **Number of Agents at Start**

A number of agents exist in the graph at the beginning of simulation. This value will be used only in Agent with Token and Agent with Whiteboard Model simulations

3.4.1.12 **Current Number of Agents**

A number of agents currently exist. This value will be used only in Agent with Token and Agent with Whiteboard Model simulations

3.4.2 Node Properties



3.4.2.1 Node Name

A Name of selected node

3.4.2.2 User Input

Input text field that user can enter during executions of the simulation. The algorithm can retrieve this input via disJ API

3.4.2.3 Initiator

This property specifies whether the node is an initiator of the simulation execution. There must be at least one node in a graph to be an initiator in order to start the simulation. The only node with this property set to be True will execute function “distributed.plugin.runtime.enging.Entity.init()” in user code

3.4.2.4 Alive

This property reflect whether a node is alive (up and running) or death (crashed)

3.4.2.5 Breakpoint

This property is used for debugging purpose. Once a node’s breakpoint is enabled, the execution will be suspended as soon as a message/agent arrives at the node.

3.4.2.6 Number of Messages Received

Total number of messages that have been received by this node from the start of the execution of the protocol upto now

3.4.2.7 Number of Messages Sent

Total number of messages that have been sent by this node from the start of the execution of the protocol upto now

3.4.2.8 Outgoing Ports

A list of all outgoing ports available at this node

3.4.2.9 Incoming Ports

A list of all incoming ports available at this node

3.4.2.10 States Transition

This property reports the sequence of states transition of the node that has been changed from the start of the simulation until now

3.4.2.11 Number of Agent Hosted

Total number of agents that this node hosted has “Home Base” from the beginning of the simulation. This is used with Agent with Whiteboard and Agent with Token models only

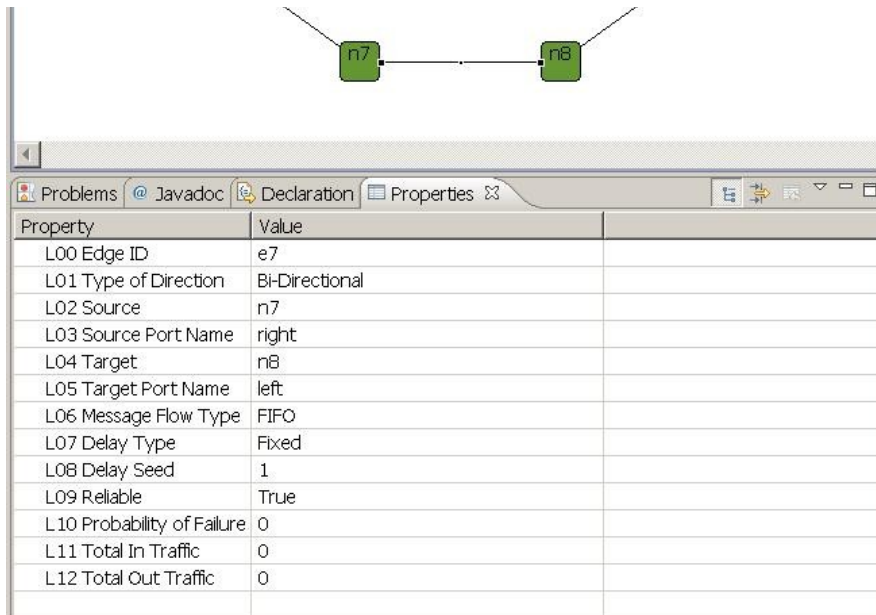
3.4.2.12 Current Number of Agents

Total number of agents currently resides at this node. This is used with Agent with Whiteboard and Agent with Token models only

3.4.2.13 Current Number of Tokens

Total number of Tokens currently resides at this node. This is used with Agent with Whiteboard and Agent with Token models only

3.4.3 Link Properties



Property	Value
L00 Edge ID	e7
L01 Type of Direction	Bi-Directional
L02 Source	n7
L03 Source Port Name	right
L04 Target	n8
L05 Target Port Name	left
L06 Message Flow Type	FIFO
L07 Delay Type	Fixed
L08 Delay Seed	1
L09 Reliable	True
L10 Probability of Failure	0
L11 Total In Traffic	0
L12 Total Out Traffic	0

3.4.3.1 Edge ID

An ID of selected link

3.4.3.2 Type of Direction

The type of communication link

3.4.3.3 Source Name

The source node's name of the link

3.4.3.4 Source Port Name

The source port's name of the link

3.4.3.5 Target Name

The target node's name of the link

3.4.3.6 Target Port Name

The target port's name of the link

3.4.3.7 Message Flow Type

This property specifies the flow type of messages inside the link. There are only two types: FIFO and No Order.

3.4.3.8 Delay Type

Type of traveling delay of every links in this graph, there are 5 types, Synchronous, Random Uniformed, Random Poisson, Random Customs, Customs

3.4.3.9 Delay Seed

A number of simulation time unit required on each link in the graph. This value will be used only when the delay type is Synchronous

3.4.3.10 Reliable

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*”.

3.4.3.11 Probability of Failure

This property specifies the probability rate that a message will fail on this link, if a *Reliable* of the link is set to “False”.

3.4.3.12 Total In Traffic

Total number of messages/agents entering into this link

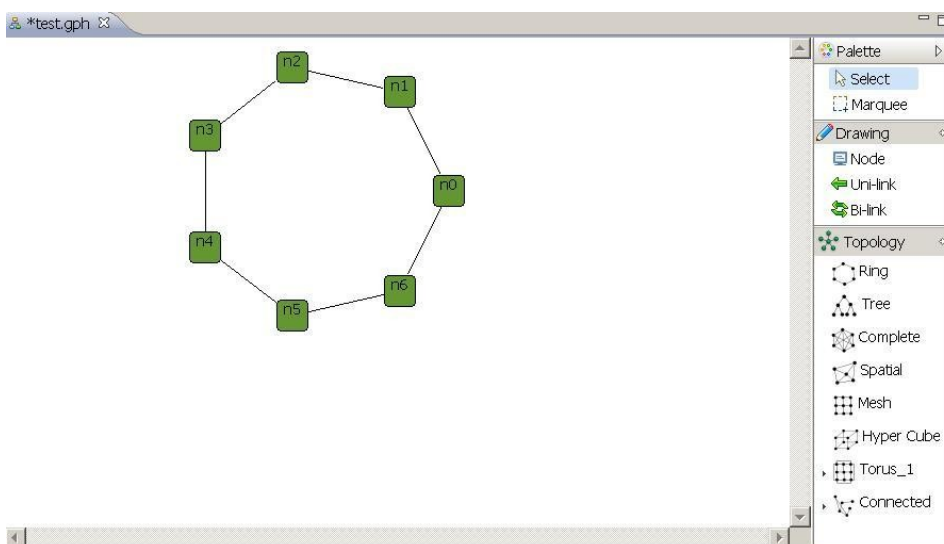
3.4.3.13 Total Out Traffic

Total number of messages/agents departing from this link

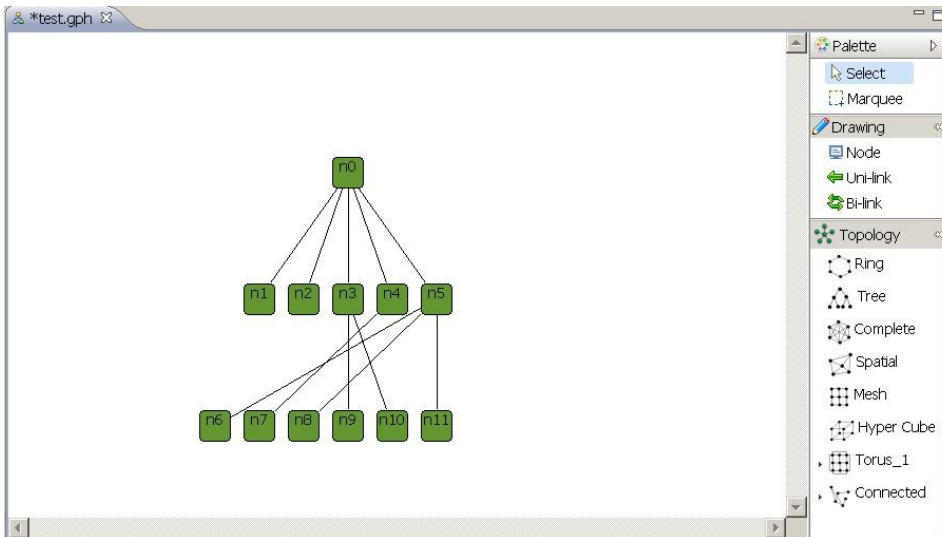
3.5 Difference type of Topologies

DisJ Graph Editor provides a Topology Library that offers a set of readymade graphs

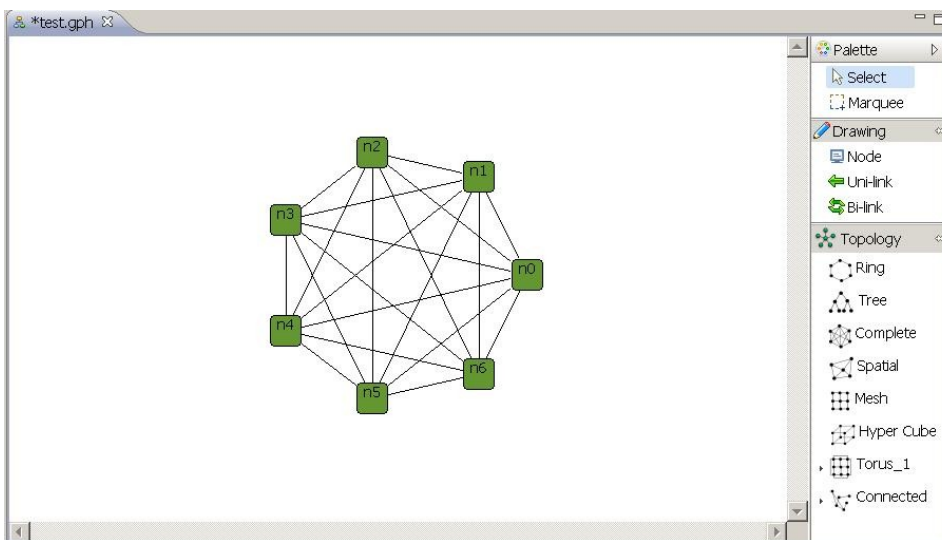
3.5.1 Ring Graph



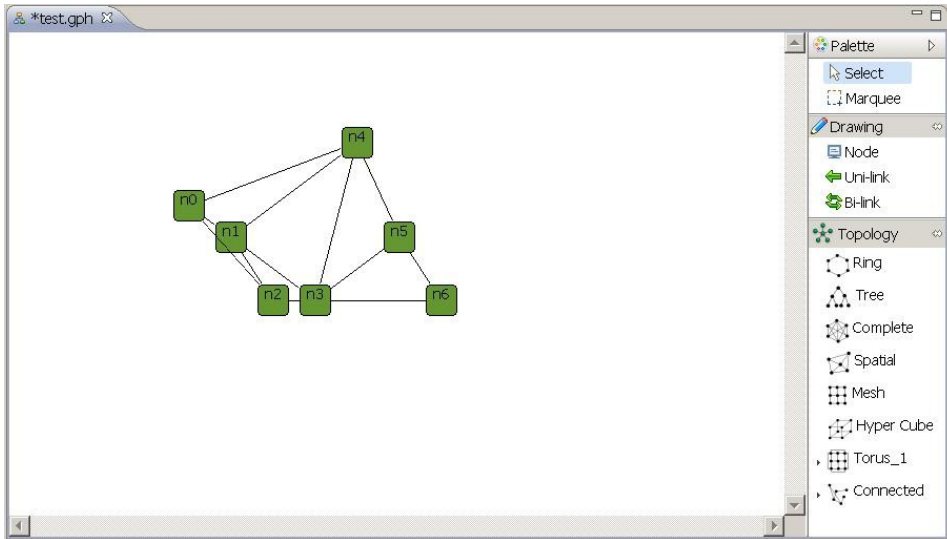
3.5.2 Tree Graph



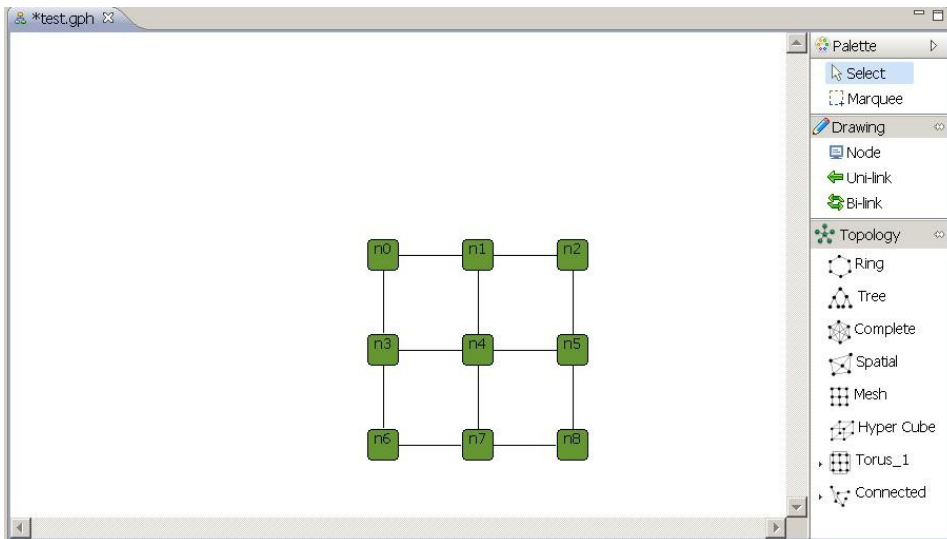
3.5.3 Complete Graph



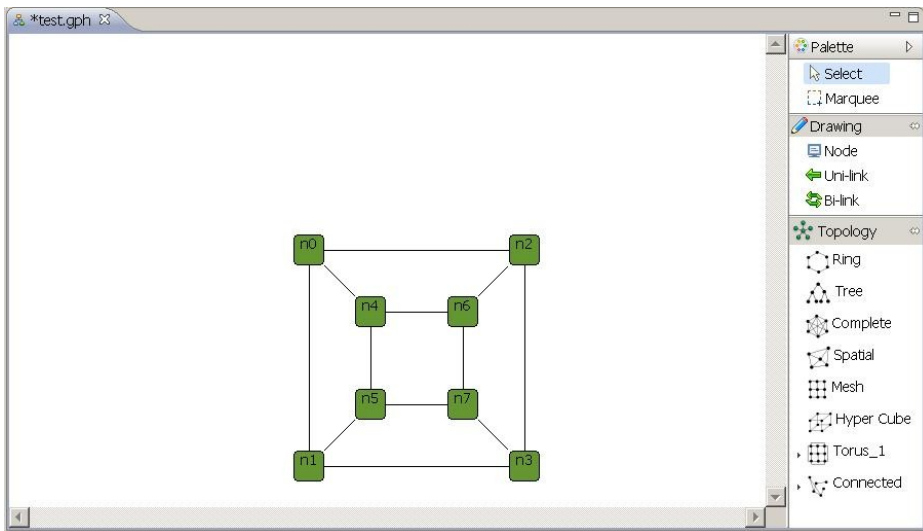
3.5.4 Spatial Graph



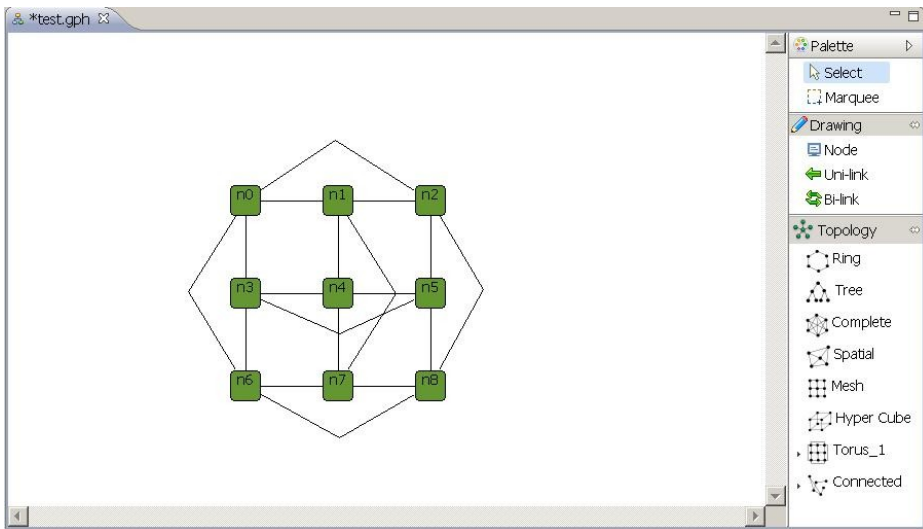
3.5.5 Mesh Graph



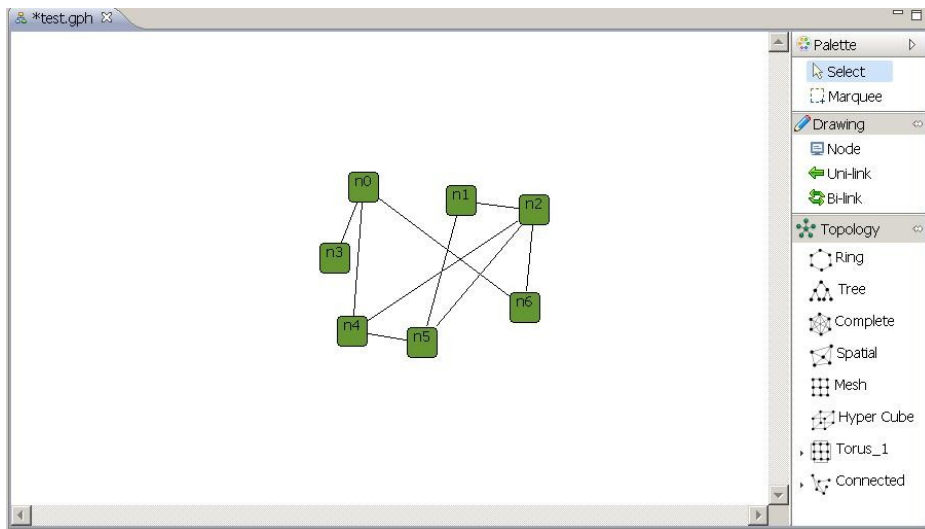
3.5.6 Mesh Graph



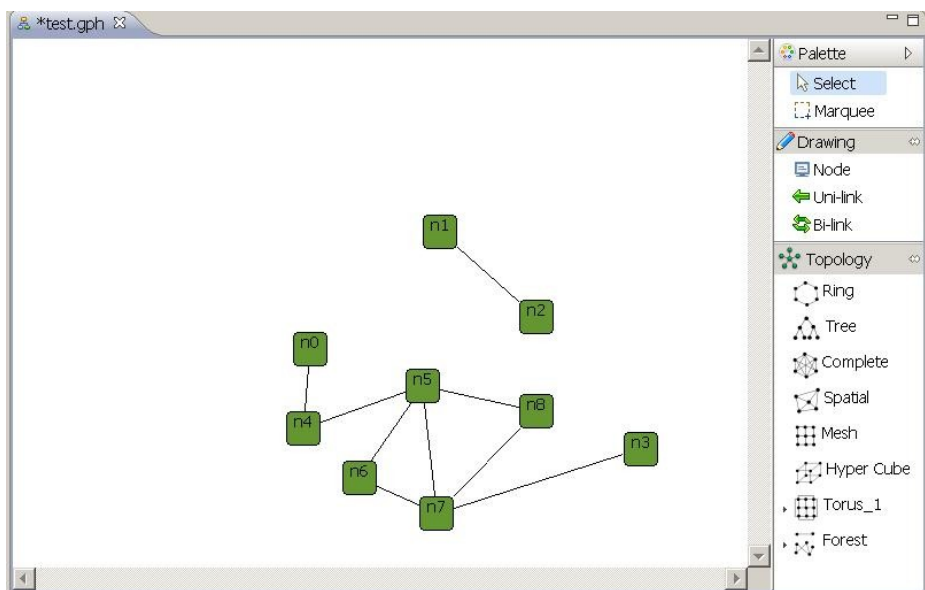
3.5.7 Torus Graph



3.5.8 Random Connected Graph



3.5.9 Random Forest Graph



3.5.10 Summary

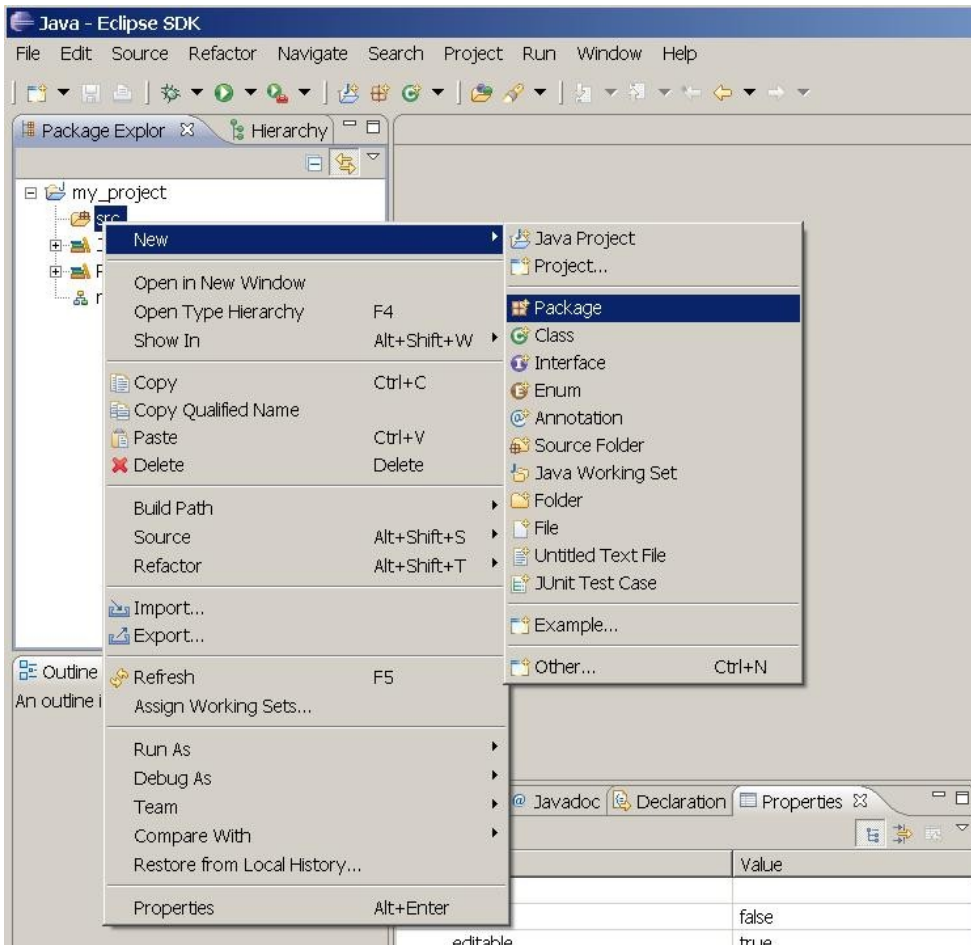
A user can create a graph by using a basic Draw objects or using Topology Library that provide a common readymade graph. Once a graph is created, there are some points about property setting and modifying a graph that user must keep in mind.

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.

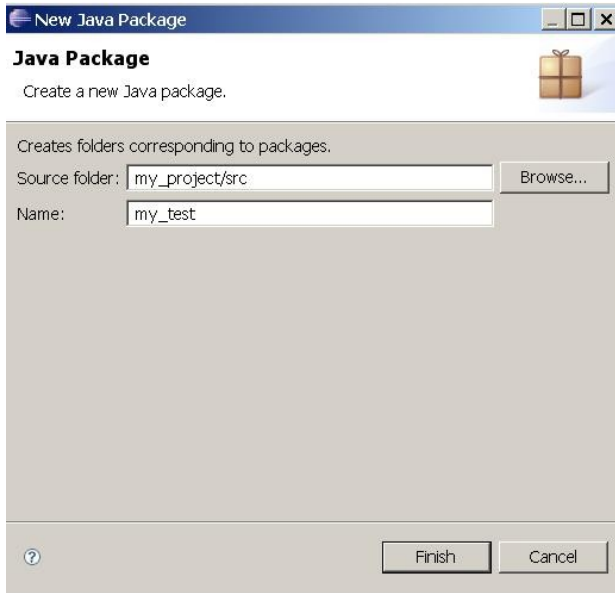
Chapter 4 Creating a Protocol

4.1 Creating a Protocol File (Java Class)

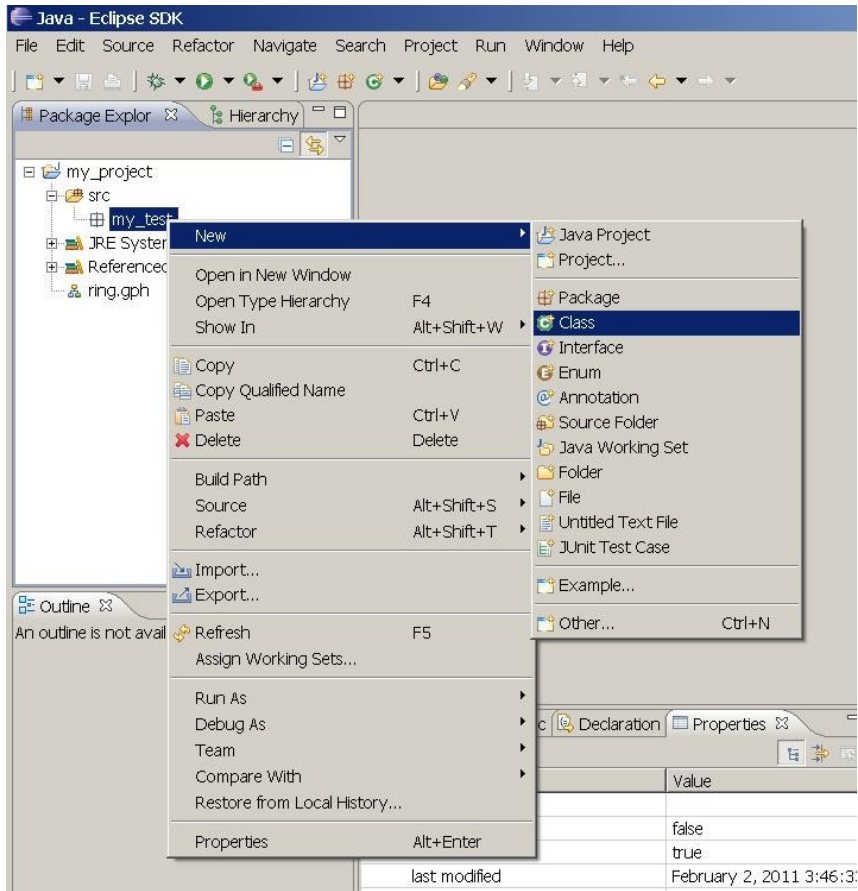
Creating a protocol is the same as creating Java class and file, which we will have to follow best practices by first creating package by “*Right Click*” at “*src*” under our project folder then select *New* → *Package* as shown below



The dialog will pop up, then type in a package name in lowercase i.e. my_test as shown below, then click “Finish button”



The package name we just created will be appeared under “src” folder. Now, create a new Java class by “Right Click” at “my_test” package, then select *New* → *Class* as show below



A file creation dialog will pop up, then type in a Java Class name i.e. *AsFar*. Since *AsFar* is a protocol for Message Passing Model, therefore the superclass is **“distributed.plugin.runtim.engine.Entity”**, click at “Browse” button to find and select our superclass (see Chapter4.2) then “check” the checkbox “Constructor from superclass” and click “Finish” button

New Java Class

Java Class

Create a new Java class.

Source folder:

Package:

Enclosing type:

Name:

Modifiers: public default private protected
 abstract final static

Superclass:

Interfaces:

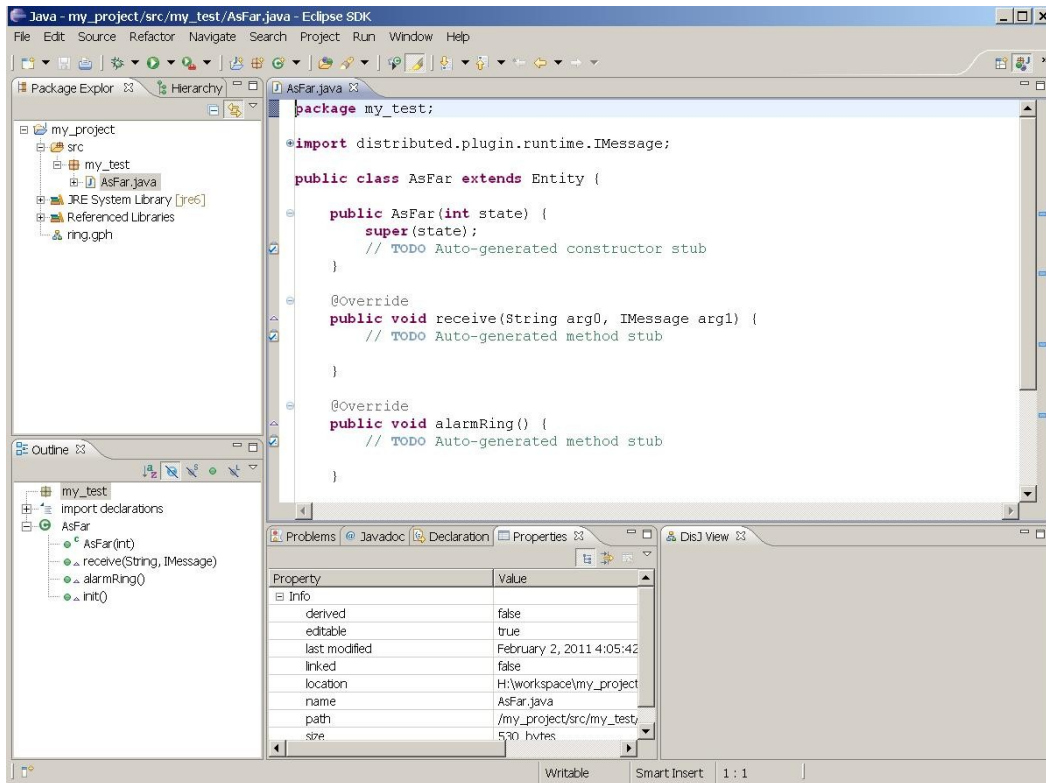
Which method stubs would you like to create?

public static void main(String[] args)
 Constructors from superclass
 Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

Generate comments

The workbench will be looked like as follow



4.2 Rules and Conventions.

DisJ provides three types of distributed computing models and each has its own superclass for user to inherit as follows:

- a. Message Passing Model →
“**distributed.plugin.runtime.engine.Entity**”
- b. Agent with Whiteboard Model →
“**distributed.plugin.runtime.engine.BoardAgent**”
- c. Agent with Token Model →
“**distributed.plugin.runtime.engine.TokenAgent**”

Therefore, a user class has to inherit one of these superclasses based on the model, and it will be an entry point class for DisJ simulator to run the simulation of user protocol.

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

- 4.2.1 The entry class of the protocol MUST **extends** one of above superclasses based on the model that user are working on and creates an **EMPTY parameter Constructor**(default Constructor). This class 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*” which the cases are **insensitive**
- 4.2.3 Assign an initial state to an entity by passing a state value into “**super (int initState)**” inside the default constructor created in 4.2.1
- 4.2.4 User should **NOT** invoke any DisJ API inside the *constructor*, since the entity object has not yet been created
- 4.2.5 There are three methods that user class has inherited from superclass, **Entity**, of *Message Passing Model* and user might need to implement

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 3.4.2.3*)

public void alarmRing(): this method is invoked when an internal alarm clock of a node is rang. The alarm time can be set at a node by invoking method setAlarm(int countDown)(*see Chapter 4.2.10*)

public void receive(String inPortLabel, IMessage msg):this method is invoked when a node receives a message. This method is a main message processing functions of node algorithm

- 4.2.6 There are four methods that user class has inherited from superclass, **TokenAgent**, of *Agent Token Model*, and **BoardAgent** of *Agent Whiteboard Model* and user might need to implement

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 3.4.2.3*)

public void alarmRing(): this method is invoked when an internal alarm clock of a node is rang. The alarm time can be set at a node by invoking method setAlarm(int countDown)(*see Chapter 4.2.10*)

public void arrived(String viaPortLabel):this method is invoked when an agent arrived at a node. This method is a main action processing functions of agent algorithm

public void notified(NotifyType type): this method is invoked when there is registered action happening at a node where an agent currently resides and registered. This method is a function that will activate agent that waiting something to happen at the node to continue its process. In order to register event at a node, agent must call method **registerHostEvent(NotifyType)** at current resides node with preferred type of event, the register will be removed automatically when the agent leave the node. There are four **NotifyType** as follow

- AGENT_ARRIVAL: Activate when any agent arrived at this node
- AGENT_DEPARTURE: Activate when any agent leave this node
- TOKEN_UPDATE: Activate when number of token at this node has been changed
- BOARD_UPDATE: Activate when whiteboard of this node has been modified

4.2.7 DisJ library has provided standard programming interfaces (API) for user to access communication and infrastructure libraries for his/her reactive distributed algorithms. The following are examples of common and useful API that can be used in the algorithm

Message Passing Model

```
sendTo()  
become()  
getState()  
getName()  
getInPorts()  
getOutPorts()
```

Agent with Token Model

```
moveTo()  
become()  
getAgentId()  
getNodeId()  
getState()  
getNodeState()  
getInPorts()  
getOutPorts()  
countHostToken()  
countMyToken()  
dropToken()  
pickupToken()
```

Agent with Board Model

```
moveTo()
```

```
become()  
getAgentId()  
getNodeId()  
getState()  
getNodeState()  
getInPorts()  
getOutPorts()  
readFromBoard()  
removeFromBoard()  
appendToBoard
```

For **full list** of available APIs can be found at [DisJ Java API Documents](#) or using *code assist* from Eclipse IDE.

4.3 Coding a Protocol

The convention for distributed algorithm in DisJ is based on reactive model defined in *state x event* \rightarrow *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 will write a protocol for Ring Election named “*As Far*” with multiple initiators, unique ID, bidirectional links, synchronous, and total reliable environment.

```
Java - my_project/src/my_test/AsFar.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help

AsFar.java x ring.gph
package my_test;

import distributed.plugin.runtime.IMessage;
import distributed.plugin.runtime.engine.Entity;

/**
 * Leader election protocol with multiple initiators, unique ID,
 * synchronous and total reliable communication.
 *
 * The initiators (candidates) will send messages with its ID to all
 * directions, and a message with smallest ID will kill every candidates
 * and return back to the sender. A candidate with smaller ID will kill
 * all arrival messages with bigger ID as well.
 */
public class AsFar extends Entity {

    // all possible states
    public static final int STATE_SLEEP = 0;
    public static final int STATE_ELECTION = 1;
    public static final int STATE_PASSIVE = 2;
    public static final int STATE_FOLLOWER = 3;
    public static final int STATE_LEADER = 4;

    private static final String MSG_LABEL_ELECTION = "Election";
    private static final String MSG_LABEL_NOTIFY = "Notify";

    // tracking that both of my messages have returned
    private boolean leftMin;
    private boolean rightMin;

    public AsFar() {
        super(STATE_SLEEP);
    }

    public void init() {
        String myId = this.getName();
        this.sendToAll(MSG_LABEL_ELECTION, myId);
        this.become(STATE_ELECTION);
    }
}
```

Here an example of implementing **Entity.receive()** method

```

public void receive(String incomingPort, IMessage message) {
    String msg = (String) message.getContent();
    String msgLabel = message.getLabel();

    if (this.getState() == STATE_SLEEP) {
        this.sendToOthers(MSG_LABEL_ELECTION, msg);
        this.become(STATE_PASSIVE);
    }
    else if (this.getState() == STATE_PASSIVE) {
        if (msgLabel.equals(MSG_LABEL_NOTIFY)) {
            this.sendToOthers(MSG_LABEL_NOTIFY, msg);
            this.become(STATE_FOLLOWER);
        }
        else {
            this.sendToOthers(MSG_LABEL_ELECTION, msg);
        }
    }
    else if (this.getState() == STATE_ELECTION) {
        if (msgLabel.equals(MSG_LABEL_ELECTION)) {
            this.electing(incomingPort, msg);
        }
        else {
            // should not happen!!
        }
    }
    else if (this.getState() == STATE_LEADER) {
        if (msg.equals(MSG_LABEL_NOTIFY)) {
            System.out.println("Election completed: " + this.getName() + " is a leader");
        }
        else {
            // should not happen!!
        }
    }
}

/*
 * Helping function that will compare a received ID with my ID
 */
private void electing(String incomingPort, String id) {

    String myId = this.getName();

    if (id.compareTo(myId) < 0) {
        this.sendToOthers(MSG_LABEL_ELECTION, id);
        this.become(STATE_PASSIVE);
    }
    else if (id.compareTo(myId) == 0) {
        // my election msg has returned
        if (this.leftMin == true)

```

Some inherited function that the protocol does not need, user can leave it empty as function **Entity.alarmRing()** as shown below

```

public void init() {
    String myId = this.getName();
    this.sendToAll(MSG_LABEL_ELECTION, myId);
    this.become(STATE_ELECTION);
}

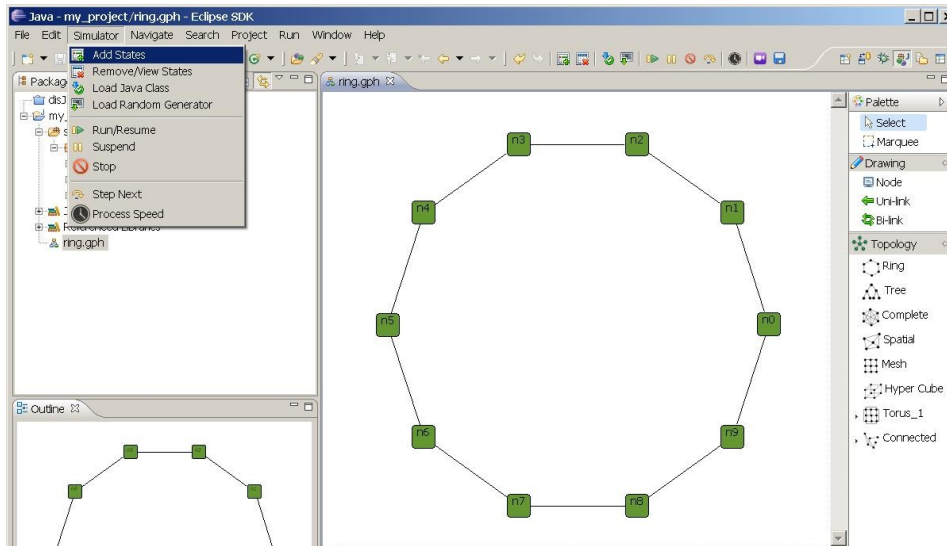
@Override
public void alarmRing() {
    // this protocol does not need this function
}

public void receive(String incomingPort, IMessage message) {
    String msg = (String) message.getContent();
    String msgLabel = message.getLabel();

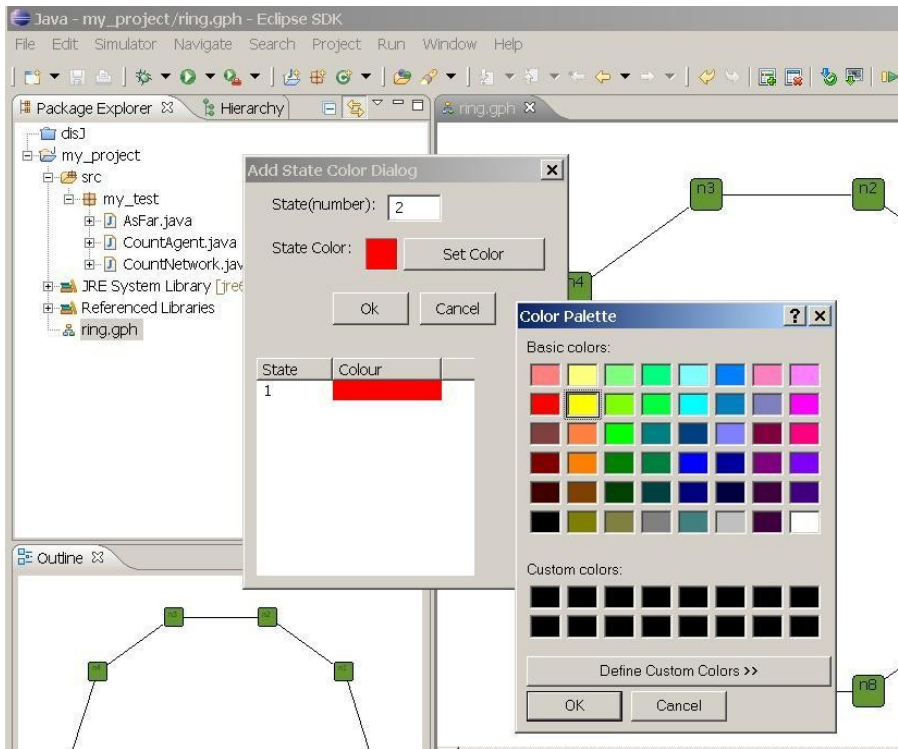
```

4.4 Setting State Color

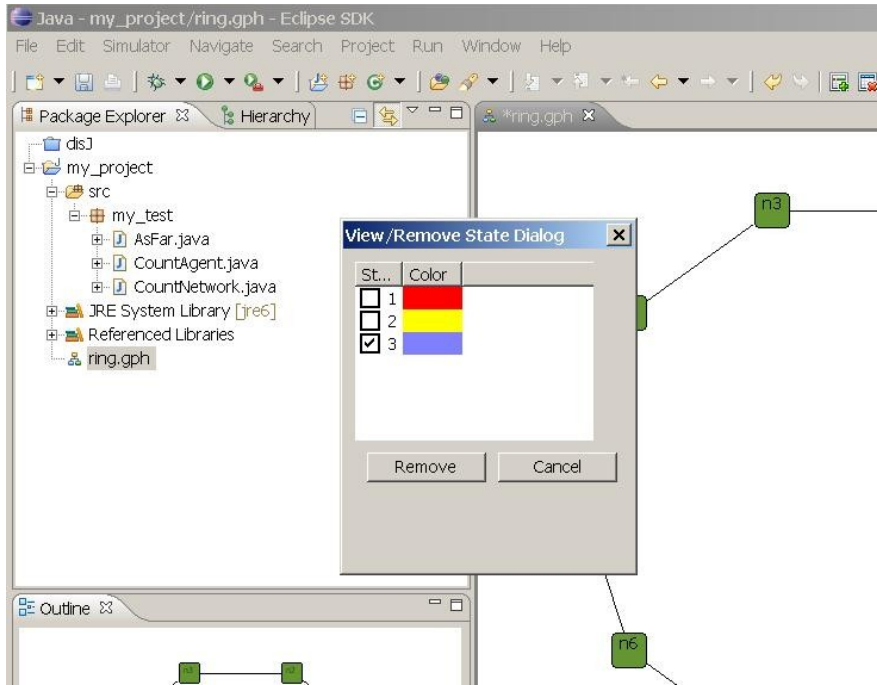
In order to see real-time nodes state update in DisJ Graph Editor during the simulation; the user must specify a color for each state that defined in the protocol (*see Chapter 4.2.2*) by set the workbench focus to DisJ Graph Editor, then go to menu bar, select *Simulator* → *Add States* as shown below



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.



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 wants to remove and click Remove as shown below.



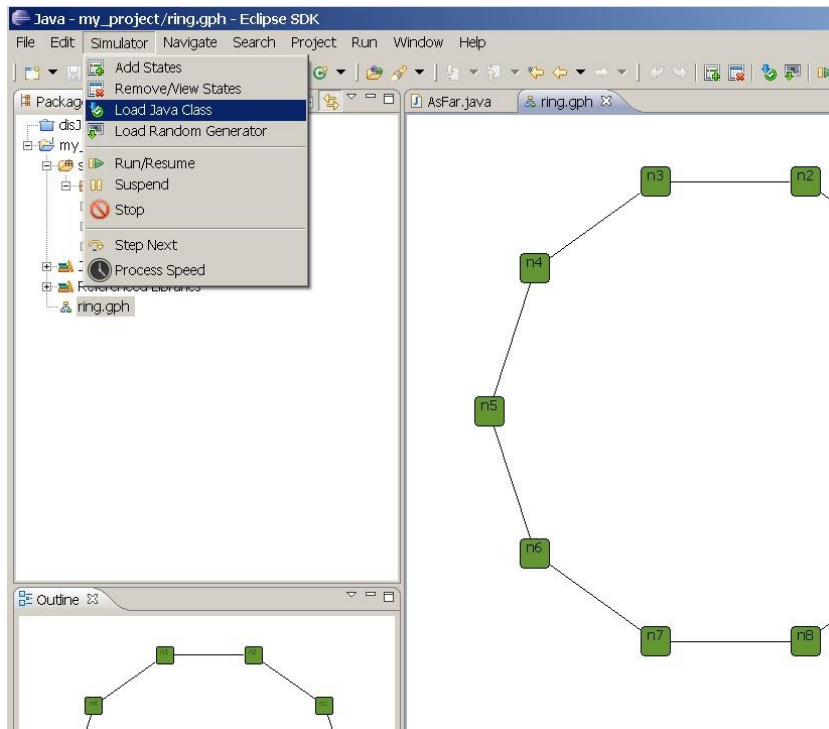
Note that the simulator will ignore all the number-color pairs that do not match the states declared in protocol. Therefore, only a match number- color pairs will be used during the simulation.

Chapter 5. Running a Simulation

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

5.1 Loading protocol into a Topology

Once topology(graph file) and protocol are created; To simulate the protocol in the simulator, user has to load the protocol into the topology by go to menu select *Simulator* → *Load Java Class* as shown below



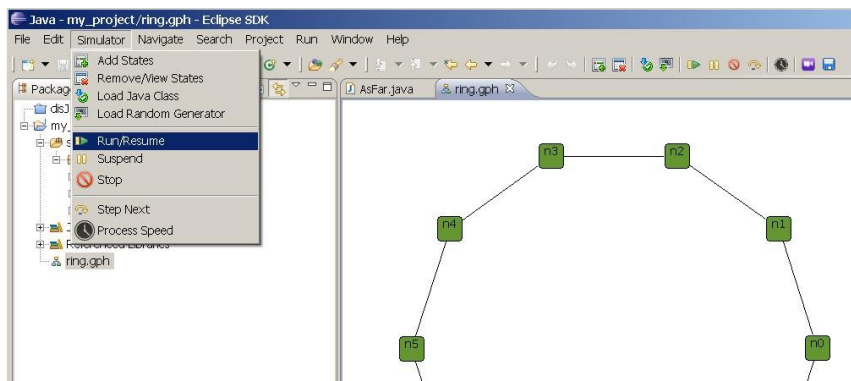
Once a dialog shows up; there are three models to select, in this example is *Message Passing Model*. Then type in a *Fully Qualified Class Name* of the entry class (a class that implement **distributed.runtime.engine.Entity**), in this example is “*my_test.AsFar*” (*as show below*) then click OK. Remember that all the protocol classes and the graph file (.gph) must be under a same project folder, in this example is “*my_project*”. Each graph file is independent from other graph file in a same project, means loading the entry class into one graph file does not affect other graph file.



Important: If a protocol has been loaded into a graph file, then later on a **.java** file of the protocol has been **modified**, the graph file has to be closed and reopen again in order to reload a new modified Java source into the graph file.

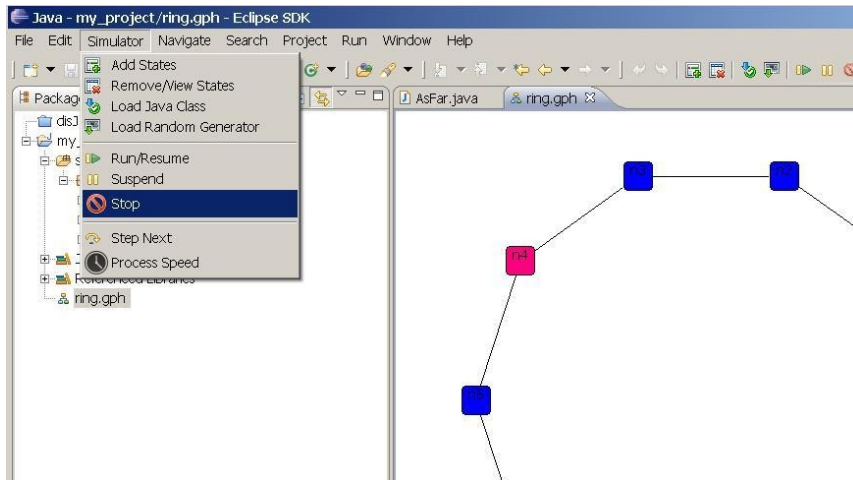
5.2 Run/Resume the simulator

Once the class is loaded into the topology; then go to menu select *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



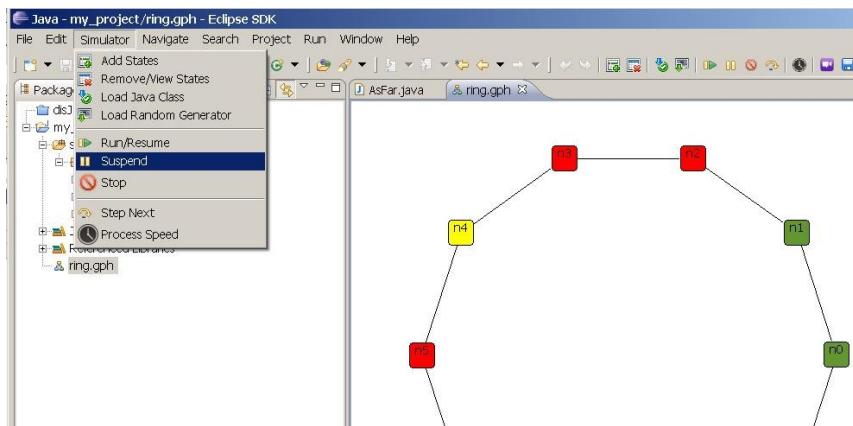
5.3 Stop the simulator

In order to stop the simulation, go to menu and select *Simulator* → *Stop* at anytime once the simulation is started. After the simulator stopped all the current states and information are wiped out. Therefore, the user cannot see any information in the “*Properties View*” anymore. Therefore, before rerun or run a new a simulation, user must make sure that the simulator is stopped



5.4 Suspend the simulator

In order to pause the simulation, go to menu and select *Simulator* → *Suspend* at anytime once the simulation is started, and to resume the execution by go to menu and select *Simulator* → *Run/Resume* as well. Remember that suspend does not wipe out the current states and information therefore, user can observe the current states in graph and “*Properties View*” as well.



5.5 Step Next

This feature is not yet support.


5.6 Setting Processing Speed

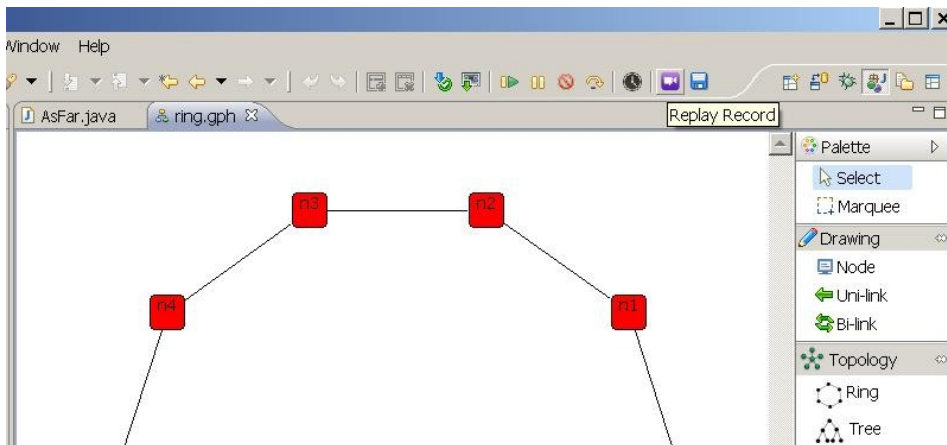
This feature allows user to setup the speed of the processor in execution the protocol, by first the simulation **MUST** be **suspended** then go to menu and select *Simulator* → *Process Speed* and the process speed scale will show up and allow user to slide in order to adjust the current speed of this process as shown below



5.7 Replay Record

This feature allow user to *replay* the simulation that has been run based on a saved record file (.rec) Currently, every simulation will be **saved automatically** once the simulation is ended under the *project/bin/* in this example is ".../my_project/bin/", the .rec file name will be a name of a topology (graph file .gph) e.g. *ring.rec* for this example. Therefore, the .rec file will be override automatically if the simulation has been run on a same graph file more than once, so, it is user responsibility to rename a .rec file if user willing to save the record file for further usage.

In order to replay, user select an icon  and a File Open dialog will pop up, and then select a .rec file e.g. ring.rec. The replay will start immediately, and user may use suspend, resume, adjust speed and stop as if the simulation is running. Note that the results, statistic report and behavior of the replay is the exactly the same as what happening during the simulation of the record and cannot be edited.



Here is a snap shot when DisJ is executing

Property	Value
G00 Graph Name	ring.gph
G01 Total Nodes	10
G02 Total Links	10
G03 Total Messages Received	12
G04 Total Messages Sent	24
G05 Global Message Flow Type	FIFO
G06 Global Delay Type	Synchronous
G07 Global Delay Seed	1
G08 Protocol	my_test.AsFar
G09 Max Number of Tokens per Agent	1
G10 Number of Agents at Start	0
G11 Current Number of Agents	0

To view a node/link properties by selecting a node/link in editor at anytime of simulation execution

Property	Value
N00 Node Name	n7
N01 User Input	
N02 Initiator	True
N03 Alive	True
N04 Breakpoint	Disable
N05 Number of Messages Received	8
N06 Number of Messages Sent	12
N07 Outgoing Ports	[left, right]
N08 Incoming Ports	[left, right]
N09 States Transition	[STATE_ELECTION, STATE_PASSIVE, STATE_FOLLOWER]
N10 Number of Agents Hosted	0
N11 Current Number of Agents	0
N12 Current Number of Tokens	0

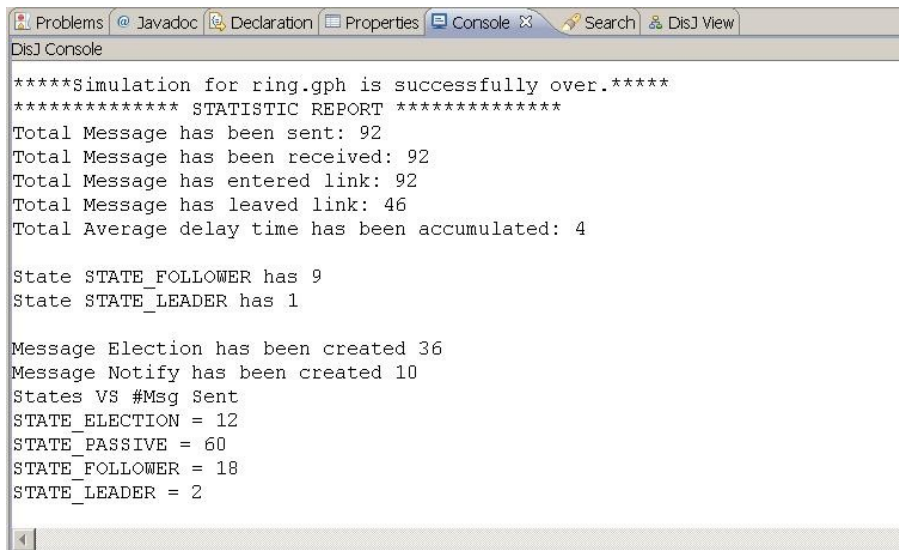
The screenshot shows a network simulation software interface. At the top, a network diagram displays four blue square nodes connected in a diamond shape. A central horizontal edge connects two nodes, with a double-headed arrow indicating bidirectional flow. The other three edges are single-headed arrows pointing towards the central edge. On the right side, a legend lists network components: 'Mesn', 'Hyper Cube', 'Torus_1', and 'Connected'. Below the diagram is a toolbar with icons for 'Problems', 'Javadoc', 'Declaration', 'Properties', 'Console', and 'Search'. The 'Console' tab is active, displaying a table of properties for the selected edge.

Property	Value
L00 Edge ID	e7
L01 Type of Direction	Bi-Directional
L02 Source Name	n7
L03 Source Port Name	right
L04 Target Name	n8
L05 Target Port Name	left
L06 Message Flow Type	FIFO
L07 Delay Type	Fixed
L08 Delay Seed	1
L09 Reliable	True
L10 Probability of Failure	0
L11 Total In Traffic	10
L12 Total Out Traffic	5

Chapter 6. Graphical and Statistics Reports

DisJ simulator provides basic information and statistic reports. Currently there are two types of reports, Java Console print out and DisJ View, which are not 100% done based on build milestone. However, it still provides some useful information for user to observe and analyze the simulating protocol. The following are snapshot of each type of reports

The Console output reports



```
DisJ Console
*****Simulation for ring.gph is successfully over.*****
***** STATISTIC REPORT *****
Total Message has been sent: 92
Total Message has been received: 92
Total Message has entered link: 92
Total Message has leaved link: 46
Total Average delay time has been accumulated: 4

State STATE_FOLLOWER has 9
State STATE_LEADER has 1

Message Election has been created 36
Message Notify has been created 10
States VS #Msg Sent
STATE_ELECTION = 12
STATE_PASSIVE = 60
STATE_FOLLOWER = 18
STATE_LEADER = 2
```

DisJ View reports

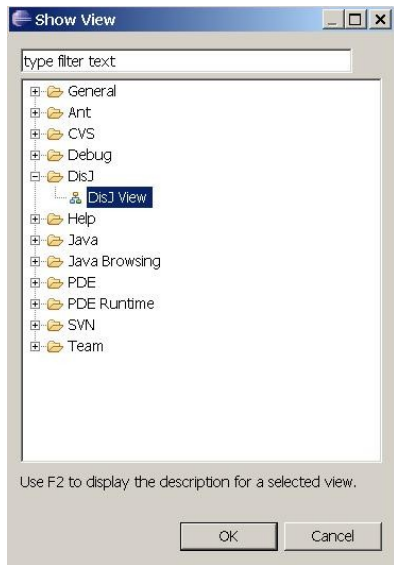
The screenshot displays the DisJ View in Eclipse. At the top, a network diagram shows nodes n5, n6, n7, n8, and n9 connected in a ring topology. Nodes n5, n6, n8, and n9 are red, while n7 is yellow. Node n0 is also yellow and connected to n9. A legend on the right lists various views: Complete, Spatial, Mesh, Hyper Cube, Torus_1, and Connected. Below the diagram is a table with the following data:

Node ID	isAlive	Node State	# Token	Whiteboard
n1	True	STATE_PASSIVE	0	board info is here
n0	True	STATE_ELECTION	0	
n5	True	STATE_PASSIVE	0	
n4	True	STATE_ELECTION	0	
n3	True	STATE_PASSIVE	0	
n2	True	STATE_PASSIVE	0	
n9	True	STATE_PASSIVE	0	
n8	True	STATE_PASSIVE	0	
n7	True	STATE_ELECTION	0	
n6	True	STATE_PASSIVE	0	

In order to open “DisJ View” in Eclipse is the same way of opening “Properties View” by go to menu and select *Window* → *Show View* → *Other...* as shown below

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays a project structure with a file named 'ring.gph'. The 'Window' menu is open, and the 'Show View' option is selected. A secondary menu is displayed, listing various views such as Ant, Console, Declaration, Error Log, Hierarchy, Javadoc, Navigator, Outline, Package Explorer, Problems, Progress, Project Explorer, Search, Tasks, and Templates. The 'Other...' option at the bottom of this menu is highlighted, indicating the path to open the DisJ View.

Once the Show View dialog pop up, select *DisJ* → *DisJ View* as shown below



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 topology (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.

Appendix

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

In the convention used in DADA, rules are grouped by *state*, listing the difference “*event* \rightarrow *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 *Example 1*, 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.

```
package my_test;
```

```

import distributed.plugin.runtime.IMessage;
import distributed.plugin.runtime.engine.Entity;

/**
 * Leader election protocol with multiple initiators, unique ID,
 * synchronous and total reliable communication.
 *
 * The initiators (candidates) will send messages with its ID to all
 * directions, and a message with smallest ID will kill every candidates
 * and return back to the sender. A candidate with smaller ID will kill
 * all arrival messages with bigger ID as well.
 */
public class AsFar extends Entity {

    // all possible states
    public static final int STATE_SLEEP = 0;
    public static final int STATE_ELECTION = 1;
    public static final int STATE_PASSIVE = 2;
    public static final int STATE_FOLLOWER = 3;
    public static final int STATE_LEADER = 4;

    // all message label will be used in the protocol
    private static final String MSG_LABEL_ELECTION = "Election";
    private static final String MSG_LABEL_NOTIFY = "Notify";

    // tracking that both of my messages have returned
    private boolean leftMin;
    private boolean rightMin;

    /**
     * Default constructor
     */
    public AsFar() {
        super(STATE_SLEEP);
    }

    public void init() {
        String myId = this.getName();
        this.become(STATE_ELECTION);
        this.sendToAll(MSG_LABEL_ELECTION, myId);
    }

    @Override
    public void alarmRing() {
        // this protocol does not need this function
    }

    public void receive(String incomingPort, IMessage message) {

        String msg = (String) message.getContent();
        String msgLabel = message.getLabel();

        if (this.getState() == STATE_SLEEP) {
            this.become(STATE_PASSIVE);
            this.sendToOthers(MSG_LABEL_ELECTION, msg);
        } else if (this.getState() == STATE_PASSIVE) {
            if (msgLabel.equals(MSG_LABEL_NOTIFY)) {

```



```

        this.become(STATE_FOLLOWER);
        this.sendToOthers(MSG_LABEL_NOTIFY, msg);
    }else{
        this.sendToOthers(MSG_LABEL_ELECTION, msg);
    }
} else if (this.getState() == STATE_ELECTION) {
    if (msgLabel.equals(MSG_LABEL_ELECTION)) {
        this.electing(incomingPort, msg);
    }else{
        // should not happen!!
    }
} else if (this.getState() == STATE_LEADER) {
    if (msg.equals(MSG_LABEL_NOTIFY)){
        System.out.println("Election completed: " +
            this.getName()
            + " is a leader");
    }else{
        // should not happen!!
    }
}
}

/*
 * Helping function that will compare a received ID with my ID
 */
private void electing(String incomingPort, String id) {
    String myId = this.getName();

    if (id.compareTo(myId) < 0) {
        this.become(STATE_PASSIVE);
        this.sendToOthers(MSG_LABEL_ELECTION, id);
    } else if (id.compareTo(myId) == 0) {
        // my election msg has returned
        if (this.leftMin == true)
            this.rightMin = true;
        else
            this.leftMin = true;
    }
}
if (this.rightMin && this.leftMin) {
    this.become(STATE_LEADER);

    // send notify to one direction only
    this.sendTo(MSG_LABEL_NOTIFY, incomingPort, "I " +
        myId + " is your leader");
}
}
}

```

Example 4:

The Black Hole Search protocol with Co-Locate agents using Java Code.

```
package test;

import java.util.List;

import distributed.plugin.runtime.engine.BoardAgent;

/**
 * Whiteboard:
 * A Black Hole search in bidirectional un-oriented Ring with
 * 2 co-locate agents, known N (number of nodes), known K
 * (number of agents), total reliability and FIFO
 *
 */
public class BHC extends BoardAgent {

    public static final int STATE_NODE_UNKNOWN = 0;
    public static final int STATE_NODE_CLEAN = 1;
    public static final int STATE_NODE_TO_BH = 2;

    public static final int STATE_AGENT_WORKING = 3;
    public static final int STATE_AGENT_FOUND_BH = 4;
    public static final int STATE_AGENT_DONE = 5;

    private boolean explored;
    private boolean confirm;
    private boolean traverse;
    private boolean forward;
    private int round;
    private int numReq;
    private int numDone;

    public BHC() {
        super(STATE_NODE_UNKNOWN);
        this.round = 0;
        this.numReq = 0;
        this.numDone = 0;
        this.confirm = false;
        this.explored = false;
        this.traverse = false;
        this.forward = false;
    }
}
```

```

public void init() {
    this.become(STATE_AGENT_WORKING);
    this.setNodeState(STATE_NODE_CLEAN);

    this.round = 1;
    int o = (this.getNetworkSize() - 1)/2;
    int e = this.getNetworkSize() - o - 1;
    List<String> ports = this.getOutPorts();
    List<String> info = this.readFromBoard();
    String p = null;
    System.out.println("Board: " + info);
    if(info.isEmpty()){
        // first agent
        this.numReq = o;
        p = ports.get(0);
        this.appendToBoard("active:" + p);
    }else{
        // second agent
        this.numReq = e;
        String s = info.get(0);
        String[] v = s.split(":");
        for(int i =0;i < ports.size(); i++){
            p = ports.get(i);
            if(!p.equals(v[1])){
                break;
            }
        }
        this.appendToBoard("active:" + p);
    }
    // set for next exploring
    this.moveTo(p);
}

/*
 * My first visit to this node, and it is the first
 * visit from this port
 */
private void myFirstVisit(String port){
    // First arrived safely from safe node
    this.appendToBoard("safe:" + port);

    // Then go back to say that this node also safe
    this.explored = true;
    this.confirm = true;
    this.moveTo(port);
}

```

```

}

/*
 * Confirming port check to a previous node that this port
 * is safe then get back to the port to finish of the work
 */
private void confirmVisit(List<String> info, String port){
    String s = null;
    String[] v = null;
    for(int i =0; i < info.size(); i++){
        s = info.get(i);
        v = s.split(":");

        if(v.length == 2){
            // get my port status
            if(v[1].equals(port)){
                if(v[0].equals("active")){
                    this.removeRecord(s);
                    this.appendToBoard("safe:" + port);
                }
            }
        }else{
            // someone left a note
            // reset my work plan
            int share = Integer.parseInt(v[2]);
            this.numReq = this.numReq - share;
            this.round = Integer.parseInt(v[1]);
            this.removeRecord(s);
        }
    }
    // back to the port
    this.confirm = false;
    this.numReq--;
    this.moveTo(port);
}

public void arrive(String port) {

    // traversing back
    if(this.traverse == true){
        this.traversBack(port);
        return;
    }

    // move to my current end territory
    if(this.forward == true){

```

```

        this.toMyEnd(port);
        return;
    }

    // keep exploring
    List<String> info = this.readFromBoard();
    if(info.isEmpty()){
        // I am the first to this node
        this.myFirstVisit(port);
        this.setNodeState(STATE_NODE_CLEAN);

    }else {
        String s = null;
        String[] v = null;

        if(this.confirm == true){
            this.confirmVisit(info, port);
            return;
        }
        System.out.println("Agent: " + this.getAgentId() + " numReq " +
numReq);

        // my second visit after confirm safe port to previous node
        if(this.explored == true){
            // check my round
            if(numReq == 0){
                // complete round, go backward to find partner
                //current location
                System.out.println("Start traversing back from
node: " + this.getNodeId());

                System.out.println("whiteboard:      "      +
this.readFromBoard());

                this.traverse = true;
                this.numDone = 0;
                String p = this.getAnotherPort(port);
                this.traversBack(p);
                return;
            }

            if(info.size() == 1){
                // still the first at this node
                s = info.get(0);
                v = s.split(":");
                if(v[1].equals(port)){
                    if(v[0].equals("active")){

```

```

// should not happen, i just marked
safe on my first arrival
System.err.println("should not
happen 1");
}

// find another port to explore
List<String> ports = this.getOutPorts();
String p = null;
for(int i =0; i < ports.size(); i++){
    p = ports.get(i);
    if(!p.equals(port)){
        this.appendToBoard("active:"
+ p);

        // move to new port
        this.explored = false;
        this.moveTo(p);
    }
}

}else{
    // should not happen
    // i just visited on my first arrival
    System.err.println("should not happen 2");
}
}else{
    // other has visited this node from another port

for(int i =0; i < info.size(); i++){
    s = info.get(i);
    v = s.split(":");
    if(v[0].equals("active")){
        if(v[1].equals(port)){
            // should not happen
            // a port that i came from is
safe
            System.err.println("should
not happen 3");
        }else{
            // someone is exploring it
right now!
            // so wait.

this.registerHostEvent(NotifyType.BOARD_UPDATE);
        }
}

```

```

} else {
    if(v[1].equals(port)){
        // my port, ignore it

    } else {
        // another port that was
        // safe by other (No black
        // repeat my node territory
        // should not happen
        System.err.println("should
        not happen 4");
        break;
    }
}
}
}
} else {
    // my first visit but other has been here before
    this.myFirstVisit(port);
}
}
}

```

```

private void traversBack(String port){
    String p = this.getAnotherPort(port);
    this.numDone++;
    System.out.println("Travers back to port " + p);
    if(isSafe(p)){
        this.moveTo(p);
    } else {
        // we found under process node
        // done traversing
        this.traverse = false;

        // check whether is it a last safe node
        if(this.numDone == (this.getNetworkSize() - 1)){
            // we found black hole
            this.become(STATE_AGENT_FOUND_BH);

            // the active port is a port to BH

```

```

        this.setNodeState(STATE_NODE_TO_BH);
        System.out.println("Agent: " + this.getAgentId() + " found
BH from Node " + this.getNodeId());

```

```

        // move to my end node
        this.numDone = 0;
        this.forward = true;
        this.toMyEnd(p);

```

```

    }else{

```

```

        // compute new share

```

```

        int remain = this.getNetworkSize() - this.numDone;
        System.out.println("Found the struggling node " +
this.getNodeId() + " with remain " + remain);

```

```

        int share = remain/2;
        this.round++;
        this.numReq = share;

```

```

        // post to board that i will do extra share from my end
        this.appendToBoard("round:" + round + ":" + share);

```

```

        // move to my end node
        this.numDone = 0;
        this.forward = true;
        this.toMyEnd(p);

```

```

    }

```

```

}

```

```

}

```

```

private void toMyEnd(String port){
    String p = this.getAnotherPort(port);
    if(isSafe(p)){
        this.moveTo(p);
    }else{
        // arrive at my end
        this.forward = false;
        System.out.println("Arrived my end Node: " + this.getNodeId() + "
numReq " + this.numReq);
        if(this.getState() != STATE_AGENT_FOUND_BH){
            // arrive at my end of previous round
            List<String> info = this.readFromBoard();
            String s = null;
            String[] v = null;

```



```

        System.out.println("Board " + info);
        if(info.size() > 1){
            for(int i = 0; i < info.size(); i++){
                s = info.get(i);
                v = s.split(":");
                if(!v[0].equals("safe")){
                    this.explored = false;
                    this.confirm = false;
                    // continue exploring
                    System.out.println("Heading to " +
v[0] + " port " + v[1]);
                    this.moveTo(v[1]);
                    break;
                }
            }
        }
        }else{
            this.explored = false;
            this.appendToBoard("active:" + p);
            this.moveTo(p);
        }
    }else{
        this.become(STATE_AGENT_DONE);
        this.setNodeState(STATE_NODE_TO_BH);
    }
}

private boolean isSafe(String port){
    List<String> info = this.readFromBoard();
    boolean safe = false;
    String s = null;
    String[] v = null;
    for(int i =0; i < info.size(); i++){
        s = info.get(i);
        v = s.split(":");
        if(v.length == 2){
            if(v[0].equals("safe") && v[1].equals(port)){
                safe = true;
                break;
            }
        }
    }
    return safe;
}

```

```

/*
 * Get opposite port from a given port in a ring
 */
private String getAnotherPort(String port){
    List<String> ports = this.getOutPorts();
    String p = null;
    for(int i =0; i < ports.size(); i++){
        p = ports.get(i);
        if(!p.equals(port)){
            return p;
        }
    }
    return p;
}

public void notified(NotifyType arg0) {
}

public void alarmRing() {
}
}

\end{verbatim}

```