Communication Model for Cooperative Robotics Simulator

User Manual

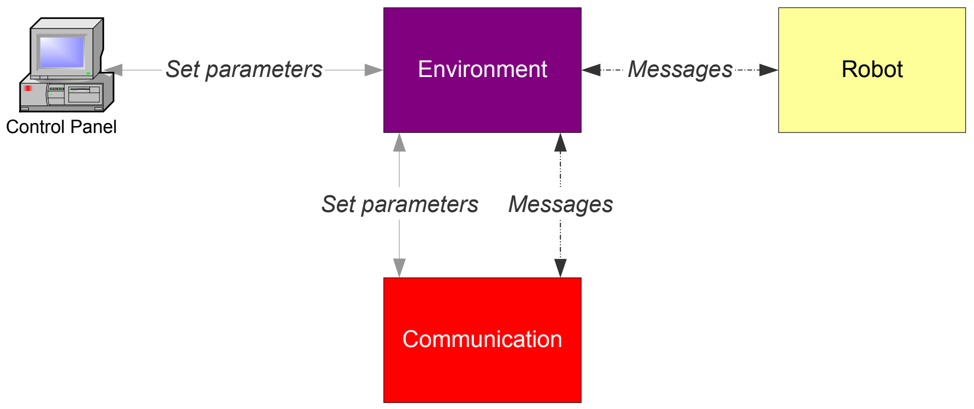
Version 1.0

1. Introduction

This section will explain how to set up, use and integrate the Communication Model with other parts in the system. In addition, it provides a brief overview of the Communication Model and its associated part.

2. Overview

Communication Model for Cooperative Robotics Simulator is a component of Cooperative Robotics Simulator. It provides communication services to the simulation system. The Communication Model component mainly interacts with the two parts: the Environment and the Control Panel. The Environment is the central component of the system. It starts every service in the system including communication. The Control Panel is a standalone system that connects to the environment simulator to monitor and control the current simulation. The Control Panel provides the Communication Model a graphical user interface to set up communication parameters, which will be delineated in the next section. The Communication Model uses the Environment as a medium to transfer messages to a robot. In addition, it also links to the Control Panel via the Environment. The following diagram shows how these parts link together.



3. Set up

3.1 Required Software

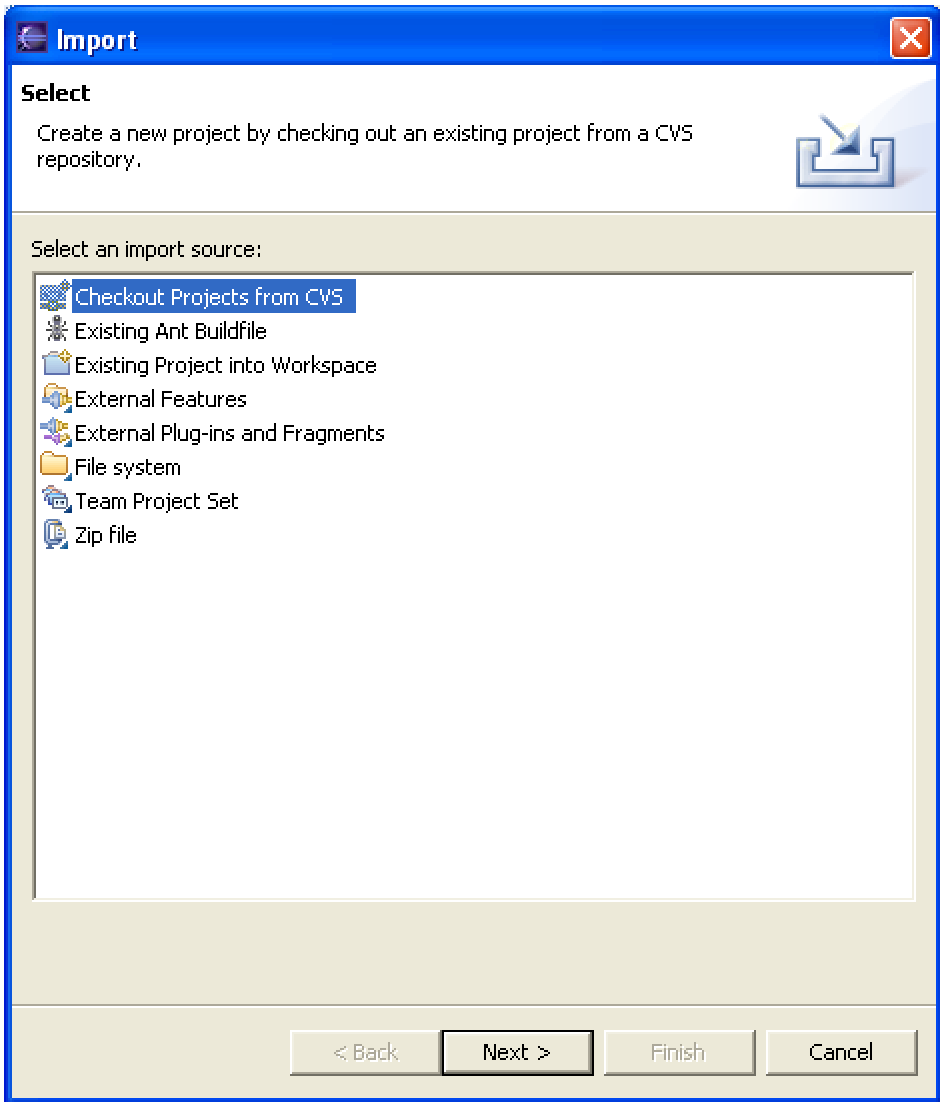
- Java 1.4.2 or later (http://java.sun.com/j2se/1.4.2/download.html)

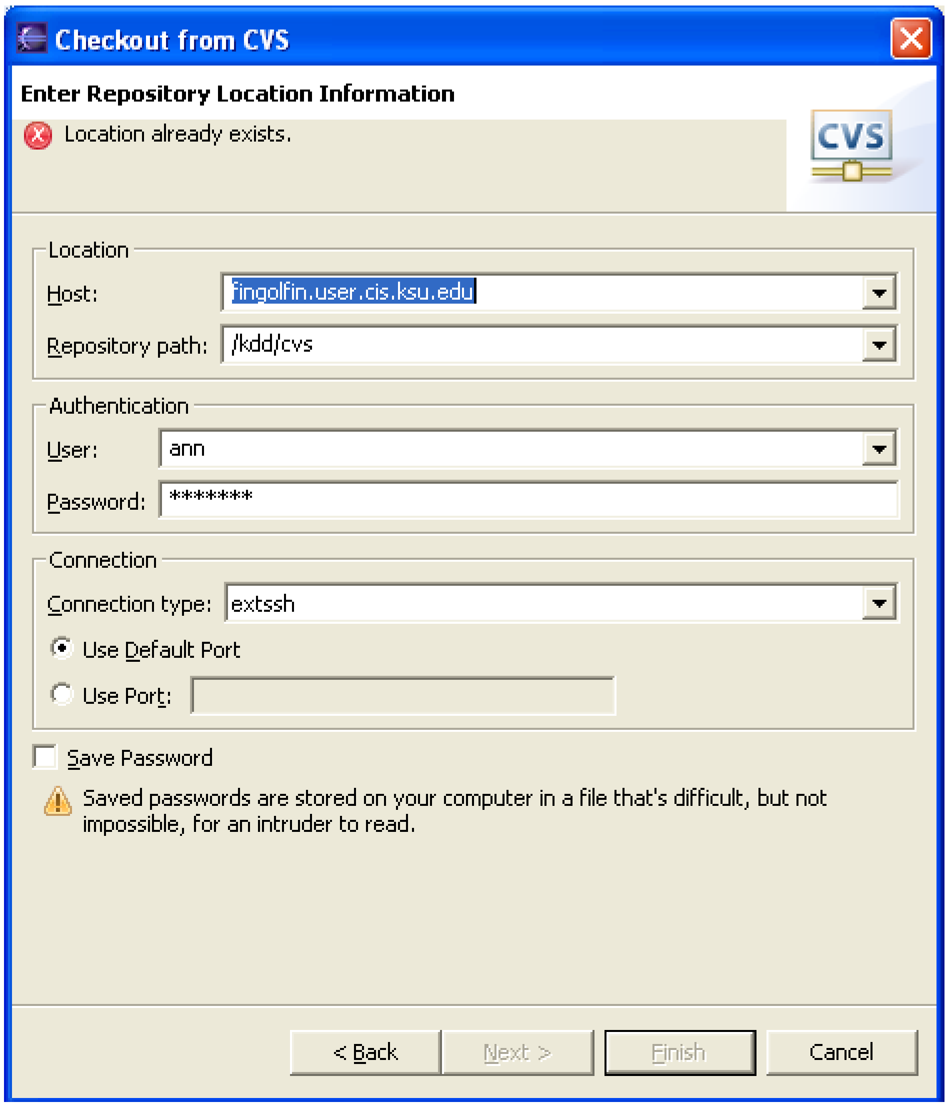
3.2 Recommended Software

Eclipse (http://www.eclipse.org/downloads/index.php)

3.3 Required Files

- All source code and executable files are included in CommunicatonModel.zip. All files must be installed under folder "edu/ksu/cis/cooprobot/simulator/communication/"
- Or Using Eclipse to checkout the source code from the CVS as followed
  Select Menu File->Import


Import
Select
Create a new project by checking out an existing project from a CVS repository.
Select an import source:
Checkout Projects from CVS
Existing Ant Buildfile
Existing Project into Workspace
External Features
External Plug-ins and Fragments
File system
Team Project Set
Zip file
< Back
Next >
Finish
Cancel



Checkout from CVS

Enter Repository Location Information

Location already exists.

Location

Host: fingolfin.user.cis.ksu.edu

Repository path: /kdd/cvs

Authentication

User: ann

Password: *******

Connection

Connection type: extssh

Use Default Port [✓]

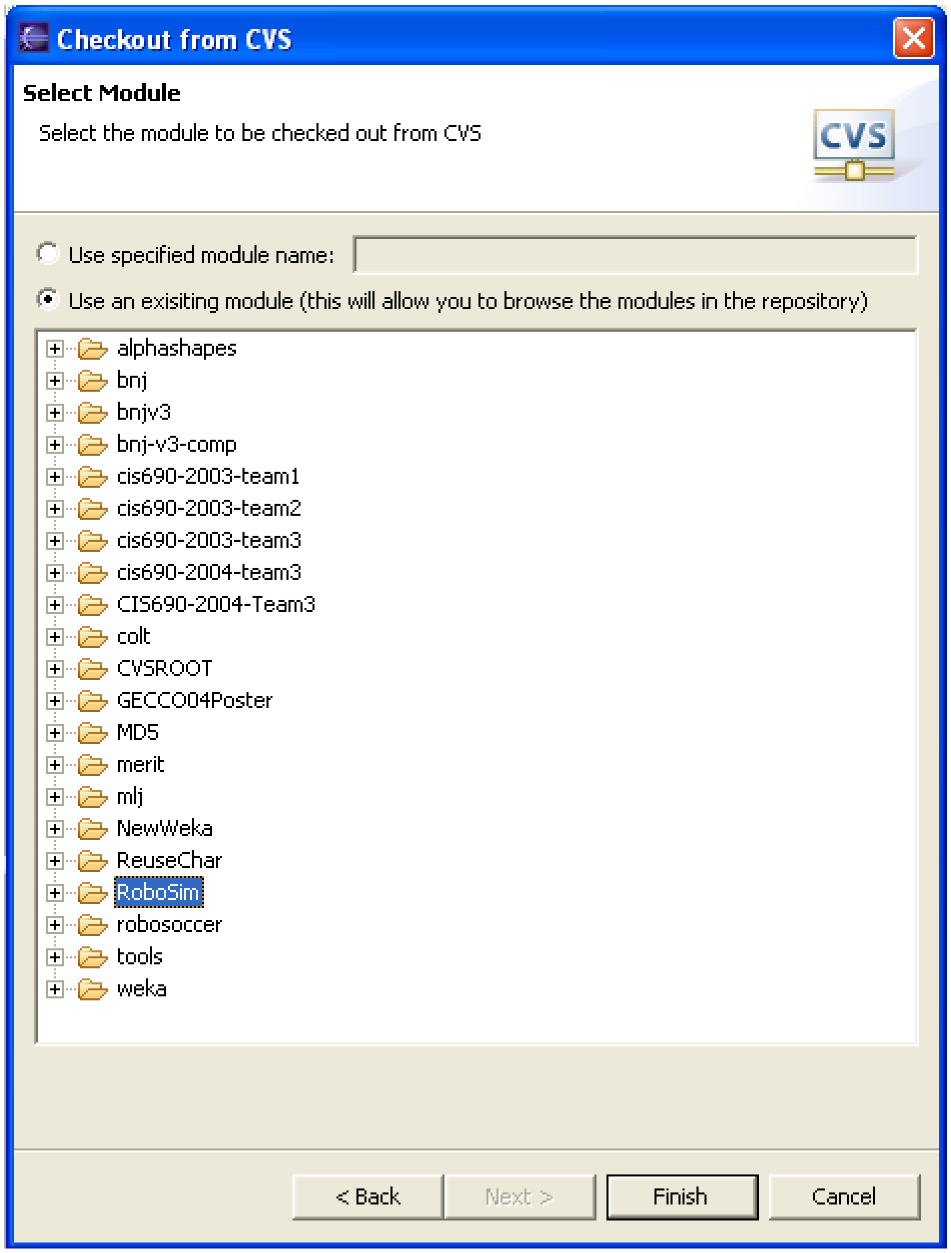
Use Port:

Save Password

Saved passwords are stored on your computer in a file that's difficult, but not impossible, for an intruder to read.

< Back | Next > | Finish | Cancel

Select RoboSim and click next (the Communication module is included in RoboSim project))



Checkout from CVS

Select Module

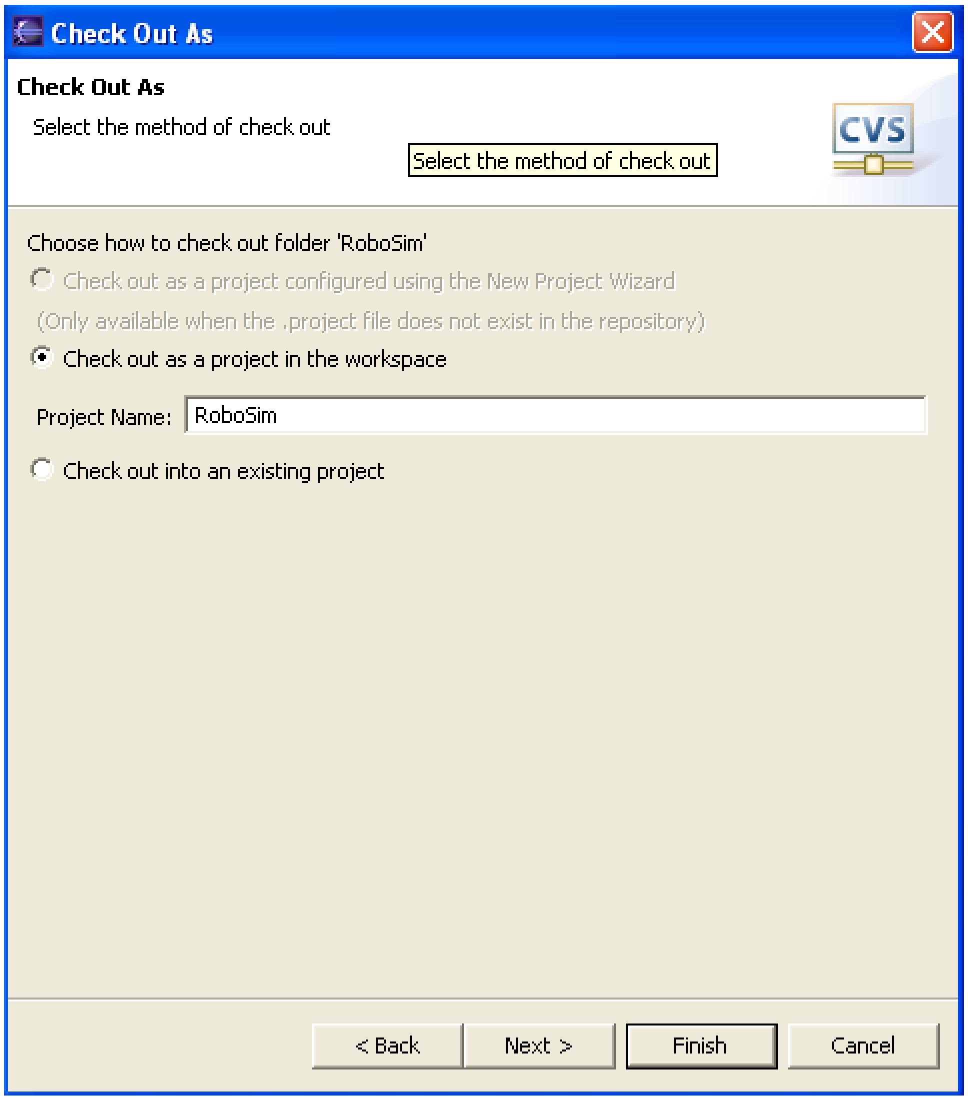
Select the module to be checked out from CVS

( ) Use specified module name:

[✓ Use an exisiting module (this will allow you to browse the modules in the repository)]

- alphashapes
- bnj
- bnjv3
- bnj-v3-comp
- cis690-2003-team1
- cis690-2003-team2
- cis690-2003-team3
- cis690-2004-team3
- CIS690-2004-Team3
- colt
- CVSROOT
- GECCO04Poster
- MD5
- merit
- mlj
- NewWeka
- ReuseChar
- RoboSim
- robosoccer
- tools
- weka

< Back | Next > | Finish | Cancel



Check Out As

Check Out As

Select the method of check out

Select the method of check out

Choose how to check out folder 'RoboSim'

○ Check out as a project configured using the New Project Wizard

(Only available when the .project file does not exist in the repository)

● Check out as a project in the workspace

Project Name: RoboSim

○ Check out into an existing project

4. Using Communication Model

The explanation of Communication Model usage will be classified by main users, which are the Control Panel and the Robot. The Control Panel uses the Communication Model to set up parameters while Robot uses it for message passing. Most of the functions are available in CommunicationsSystem object.

4.1 Initialization

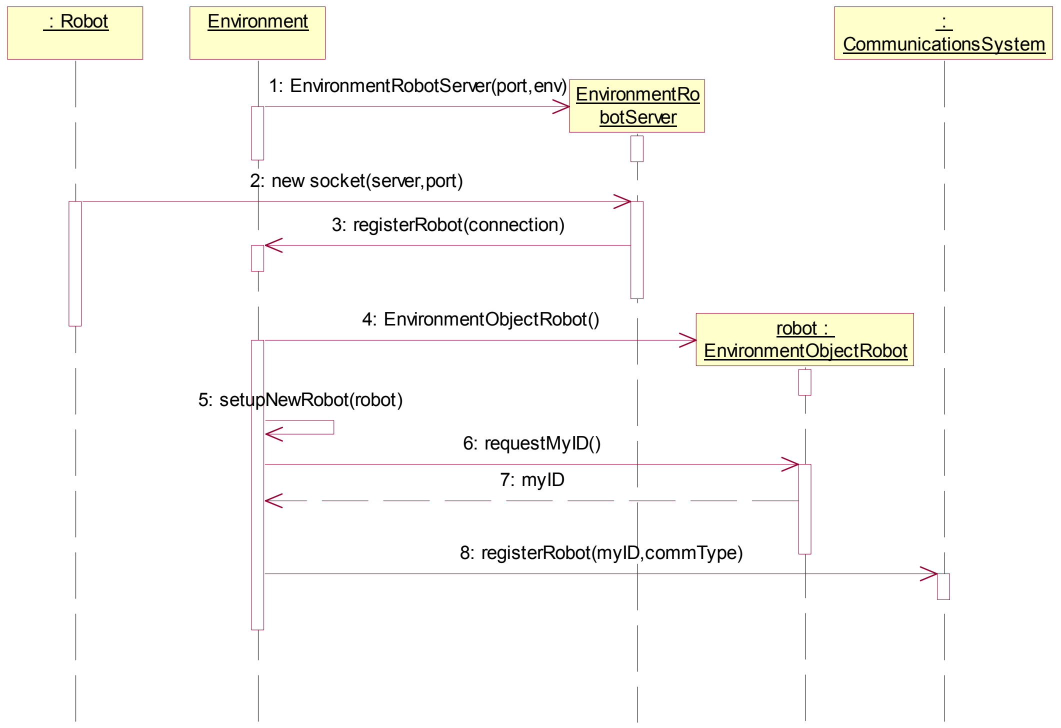
The environment is responsible for initializing the Communication System (4.1.1) and registering robots to the system (4.1.2).

4.1.1 Start up Communication System

```
CommunicationsSystem comm = new CommunicationsSystem(new
Environment env);
```

4.1.2 Register a robot to the system

All robots in the system must register to the communication system before starting communication session. The registration process happens after each robot connected to the environment. The environment registers a robot to the communication system after each robot connected to the environment within this method "`setupNewRobot(EnvironmentObjectRobot robot)`" The following sequence diagram explains how registerRobot method is used by the Environment.


: Robot
Environment
: CommunicationsSystem
1: EnvironmentRobotServer(port,env)
EnvironmentRobotServer
2: new socket(server,port)
3: registerRobot(connection)
4: EnvironmentObjectRobot()
robot : EnvironmentObjectRobot
5: setupNewRobot(robot)
6: requestMyID()
7: myID
8: registerRobot(myID,commType)

- Register a robot with broadcast and point-to-point capability

```
String robotname = "robotA";
Int commType = communicationsSystem.BROADCASTANDP2P;
comm.registerRobot(robotname,commType)
```

- Register a robot with broadcast capability

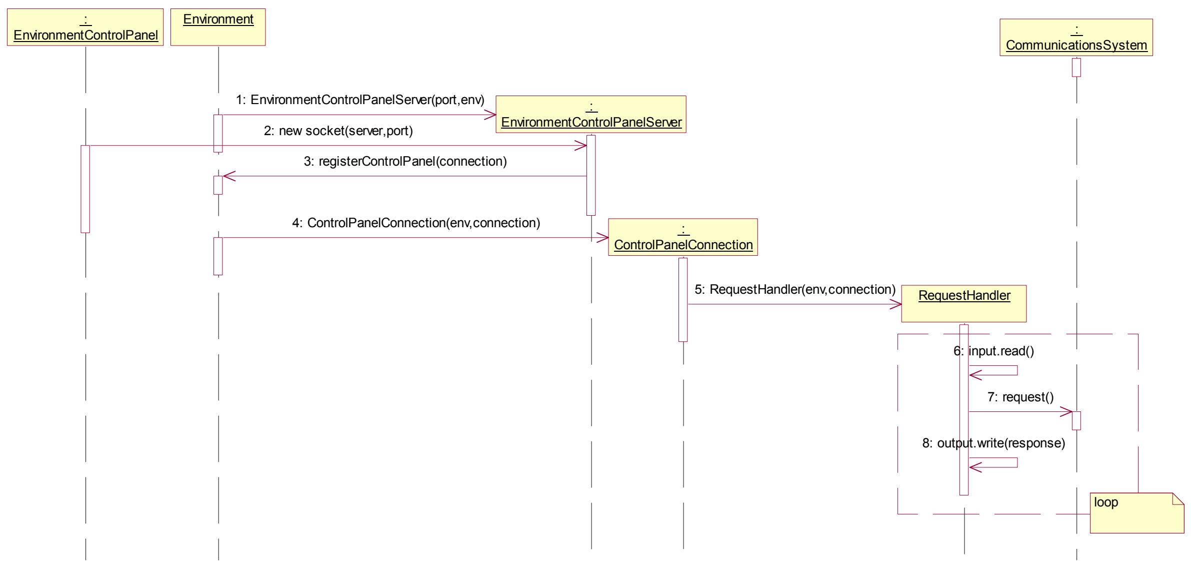
```
String robotname = "robotA";
int commType = communicationsSystem.BROADCAST;
comm.registerRobot(robotname,commType)
```

- Register a robot with point-to-point capability

```
String robotname = "robotA";
int commType = communicationsSystem.POINT2POINT;
comm.registerRobot(robotname,commType)
```

4.2 Functions for Control Panel

As stated above, the Control Panel is responsible for setting up communication parameters. However, the Control Panel have no direct access to the CommunicationsSystem, it will pass request to the Environment and the Environment will call CommunicationsSystem method directly. The following sequence diagram shows how CommunicationsSystem get request and send response back to the Environment Control Panel. The RequestHandler class in Environment package initiates a connection between the Environment and the EnvironmentControlPanel. It is responsible for processing request and returning response back to the Environment ControlPanel. The RequestHandler will determine what request is and call a CommunicationsSystem method correspond to that request.



: EnvironmentControlPanel
Environment
: CommunicationsSystem
1: EnvironmentControlPanelServer(port,env)
: EnvironmentControlPanelServer
2: new socket(server,port)
3: registerControlPanel(connection)
4: ControlPanelConnection(env,connection)
: ControlPanelConnection
5: RequestHandler(env,connection)
RequestHandler
6: input.read()
7: request()
8: output.write(response)
loop

There are two groups of communication parameters, system parameters and robot parameters. The following tables will describe each kind of parameters.

Parameter Name	**Possible Value**	**Description**
System link status	• *True* • False	This parameter controls all links status. If it is set to true, the message passing is activated. Otherwise it is not activated. This means messages cannot be passed around the system.
System Range	• *-1* (no range limit) • Positive integer	This is a maximum distance limit which all robots are able to send messages out. All receivers within this maximum distance from the sender will get the message.
System Delay	• *0* (no delay) • Positive integer	This parameter simulates traffic in the system. It will delay messages to the receivers. Unit of delay time is in time step, which is set by the Environment. Time step is about 500 milliseconds.
System delivery probability	• 0-*100*	This parameter simulates a message lost situation. It applies to all messages traversing in the system. 0 means all messages are lost. 100 means all messages are delivered.

* *Default value*

Table 1 System Parameter Description Table.

Parameter Name	**Possible Value**	**Description**
Send link (Outgoing link)	• *True* • False	This parameter controls the outgoing link status of a robot. If it is set, the robot cannot send any messages out.
Receive link (Incoming link)	• *True* • False	This parameter controls the incoming link status of a robot. If it is set, the robot cannot get any messages from the other robots.
Range	• *-1* (no range limit) • Positive integer	The difference between System Range and Robot Range is that the Robot Range applies to a specific robot.
Delay	• *0* (no delay) • Positive integer	It is as same as the system one, but applies to only messages sent by a specific robot to a particular robot. Since this parameter will be set for each pair of robot.
Delivery probability	• 0-*100*	It is as same as the system one, but applies to only messages sent by a specific robot to a particular robot. Since this parameter will be set for each pair of robot.

* Default value

Table 2 Robot Parameter Description Table.

In case of both system and robot parameter are set, the value of each parameter will be as followed

Range = Summation of system range and robot range.
Delay = Summation of system delay and robot delay.
Delivery Probability = Average of system delivery probability and robot delivery probability.

4.2.1 System Parameter Usage

- **Start up all link**

```
comm.startupAllLink();
```

- **Shutdown all link**

```
comm. shutdownAllLink();
```

- **Set system range limit**

In this example, the system range is set to 20.

```
comm.setRange(20);
```

- **Set system delay time**

In this example, the system delay is set to 5.

```
comm.setDelay(5);
```

- **Set system delivery probability**

In this example, the system delivery probability is set to 90.

```
comm.setDeliveryProb(90);
```

- **Get system link status**

```
boolean status = comm.isLinkEnabled();
```

- **Get system range limit**

```
int range = comm.getRange();
```

- **Get system delay time**

```
int delay = comm.getDelay();
```

- **Get system delivery probability**

```
int probability = comm.getDeliveryProb();
```

4.2.2 Robot Parameter Usage

All these parameters can be set only if the robot has been registered to the communication system.

- **Start up robot's outgoing link**

The outgoing link of "robotA" is activated by the following code.

```
String robotname = "robotA";
comm.startUpSendLink(robotname);
```

- **Shutdown robot's outgoing link**

The outgoing link of "robotA" is deactivated by the following code.

```
String robotname = "robotA";
comm.shutdownSendLink(robotname);
```

- **Start up robot's incoming link**

The incoming link of "robotA" is activated by the following code.

```
String robotname = "robotA";
comm.startUpReceiveLink(robotname);
```

- **Shutdown robot's incoming link**

The incoming link of "robotA" is deactivated by the following code.

```
String robotname = "robotA";
comm.shutdownReceiveLink(robotname);
```

- **Set range limit for a robot**

```
String robotname = "robotA";
int range = 20;
comm.setRobotRange(robotname,range);
```

- **Set delay time between a pair of robot**

```
String robotname1 = "robotA";
String robotname2 = "robotB";

int delay = 5;
comm.setRobotRange(robotname1,robotname2,delay);
```

or

```
comm.setRobotRange(robotname2,robotname1,delay);
```

These two statements are the symmetric operations. Both of them will set delay time between “robotA” and “robotB” to 5. Therefore, using either one of these operations will give the same result.

- **Set delivery probability between a pair of robot**

```
String robotname1 = "robotA";
String robotname2 = "robotB";

int probability = 90;
comm.setRobotDeliveryProb(robotname1,robotname2,probability);
```

or

```
comm.setRobotDeliveryProb(robotname2,robotname1,probability);
```

These two statements are the symmetric operations. Both of these operations will set delivery probability between “robotA” and “robotB” to 90. Hence, using either one of these operations will give the same result.

- **Get Robot Outgoing Link Status**

```
String robotname = "robotA";
boolean status = comm.isRobotSendEnabled(robotname);
```

- **Get Robot Incoming Link Status**

```
String robotname = "robotA";
boolean status = comm.isRobotReceiveEnabled(robotname);
```

- **Get robot broadcast capability status**

```
String robotname = "robotA";
boolean status = comm.isRobotBroadcastEnabled(robotname);
```

- **Get robot point-to-point capability status**

```
String robotname = "robotA";
boolean status = comm.isRobotP2PEnabled(robotname);
```

- **Get robot range limit**

```
String robotname = "robotA";
int range = comm.getRobotRange(robotname);
```

- **Get robot delay time**

```
String robotname1 = "robotA";
String robotname2 = "robotB";
```

```
int delay = comm.getRobotDelay(robotname1,robotname2);
```

or

```
int delay = comm.getRobotDelay(robotname2,robotname1);
```

These two statements are the symmetric operations. Both of these operations will return the delay time between “robotA” and “robotB”. As a result, using either one of these operations will return the same result.

- **Get robot delivery probability**

```
String robotname1 = “robotA”;
String robotname2 = “robotB”;

int probability =
comm.getRobotDeliveryProb(robotname1,robotname2);
```

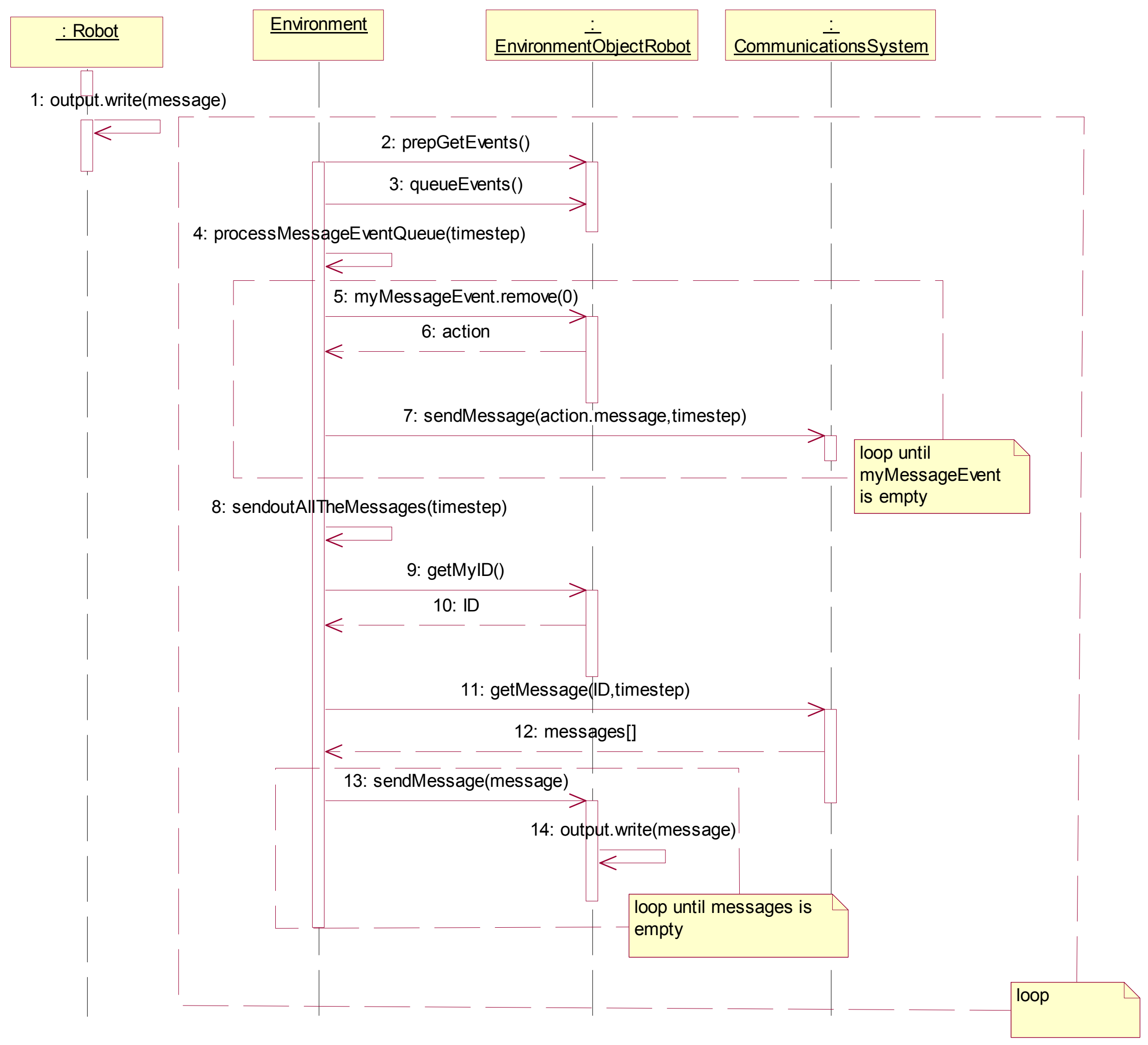
or

```
int probability =
comm.getRobotDeliveryProb(robotname2,robotname1);
```

These two statements are the symmetric operations. Both of these operations will return delivery probability between “robotA” and “robotB”. Therefore, using either one of these operations will return the same result.

4.3 Functions for Robot

Sending and Receiving messages are core functions provided for Robot. Since messages are passed to the Robot via the Environment, all these functions will be used by the Environment. This sequence diagram describes how to send and receive message within the Environment package.



: Robot
Environment
: EnvironmentObjectRobot
: CommunicationsSystem
1: output.write(message)
2: prepGetEvents()
3: queueEvents()
4: processMessageEventQueue(timestep)
5: myMessageEvent.remove(0)
6: action
7: sendMessage(action.message,timestep)
loop until myMessageEvent is empty
8: sendoutAllTheMessages(timestep)
9: getMyID()
10: ID
11: getMessage(ID,timestep)
12: messages[]
13: sendMessage(message)
14: output.write(message)
loop until messages is empty
loop

EnvironmentObjectRobot is the class that establishes TCP/IP connection to Robot. Each EnvironmentObjectRobot has a queue to keep incoming message from robot. In every time step the Environment will read messages in this queue and forward to the CommunicationsSystem by processMessageEventQueue method. The process of receiving message is done by sendOutAllMessages method. The Environment calls getMessage method from the CommunicationsSystem and forward these messages to the EnvironmentObjectRobot to write out to the socket that connects to the robot.

The Environment will process sending and receiving messages in every time step. It will process sending message (processMessageEventQueue) and receiving message (sendOutAllMessages) respectively, since receivers should get the message without delay at the same time step as sending time. The following method gets messages at current time step from every robot and forwards to the communication system

```
// get message from robot forward to communication system
processMessageEventQueue(currentTime);

private void processMessageEventQueue(long timestep)
{
 for (int i=0; i < robots.size(); i++)
 {
  EnvironmentObjectRobot robot = (EnvironmentObjectRobot)robots.get(i);
  while(!robot.myMessageEvents.isEmpty())
   {
    RobotRequest action =
(RobotRequest)robot.myMessageEvents.remove(0);
    commSystem.sendMessage(action.message, timestep);
   }
 }
}
```

The following method retrieves each robot's messages from the communication system and forwards to the owner robot message by message.

```
//get message from communication system and forward to robot.
sendOutAllTheMessages(currentTime);

private void sendOutAllTheMessages(long timestep)
{
 for (int i=0; i < robots.size(); i++)
 {
  EnvironmentObjectRobot robot = (EnvironmentObjectRobot)robots.get(i);
  Vector messages = commSystem.getMessage(robot.myID, timestep);
  while(!messages.isEmpty())
 {
  Message mess = (Message) messages.remove(0);
  robot.sendMessage(mess);
 }
 robot.sendMessage(Message.NULL_MESSAGE);
 }
}
```

4.3.1 Send Message

- **Send Broadcast Message**

In case of sending broadcast message, the receiver name within Message object must be "broadcast".

```
// create a broadcast message
// robotA is the sender
String sender = "robotA";
String receiver = "broadcast";
String content = new String("broadcast message from A");
Message msg = new Message(sender,receiver,content);

// set current time step
long timeStep = 1;
comm.sendMessage(msg,timeStep);
```

- **Send point-to-point message**

```
// create a message
// robotA is the sender
String sender = "robotA";
// robotB is the receiver
String receiver = "robotB";
String content = new String("sending a message to B from A");
Message msg = new Message(sender,receiver,content);

// set current time step
long timeStep = 1;
comm.sendMessage(msg,timeStep);
```

4.3.2 Receive Message

```
// Get all messages with time step = 1 for "robotA"
String robotname = "robotA";
long timeStep = 1;
Vector msgAVector = comm.getMessage(robotname,timeStep);
```