COMMUNICATION MODEL FOR COOPERATIVE ROBOTIC SIMULATOR

By

Acharaporn Pattaravanichanon

M. S., Chulalongkorn University, 2002
B. S., Thammasat University, 1996

A PORTFOLIO

submitted in partial fulfillment of the
requirement for the degree

MASTER OF SOFTWARE ENGINEERING

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, KS

2004

Approved by:

Major Professor
Dr. William Hsu

ABSTRACT

The Communication Model is one of components of Cooperative Robotic Simulator, which is a research project in Computing and Information Science department at Kansas State University. This component provides robots an ability to communicate to each other. The Communication Model not only provides message passing in the system, but also provides an ability to simulate communication situation by defining the communication variables such as propagation delay time, maximum range limit, generating link down, and delivery probability. Since the Communication Model has no direct access to the robots, the Environment, which is the central part of the system, is responsible for passing message back and forth to the robots and the Communication Model. Furthermore, the Environment Control Panel, which is a graphical user interface tool controlling the current simulation, is responsible for obtaining communication variables from users and passing to the Communication Model. The Communication Model will handle messages based on these variables.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

PROJECT OVERVIEW

---

1. Background

Cooperative Robotics Simulator is a research project in Computing and Information Sciences department at Kansas State University (KSU). The objective of Cooperative Robotics Simulator is to perform simulations of many heterogeneous types of robots all working within a single, virtual environment. The main components of this project are Robot Simulator, Environment Control Panel and Environment Simulator.

Robot simulator consists of three parts: a robot hardware simulator, a robot control program, which will be user supplied, and an environment-based robot object. Robot control program can work with various robot hardware simulators via standard API. The robot hardware simulator will interface to the environment via requests for data from its sensor or requests for action from its actuators. The environment-based robots will be responsible for controlling the individual sensors, based on robot hardware simulator requests, and providing the appropriate data to the sensors.

Environment control panel will be a standalone system that connects to the environment simulator to monitor and control the current simulation.

The environment simulator is the central component in the system. The environment simulator is responsible for keeping track of the actual state of the environment, including each robot. The environment simulator receives requests from simulated robots to read sensors, initiate actuators, and to send and receive communication.

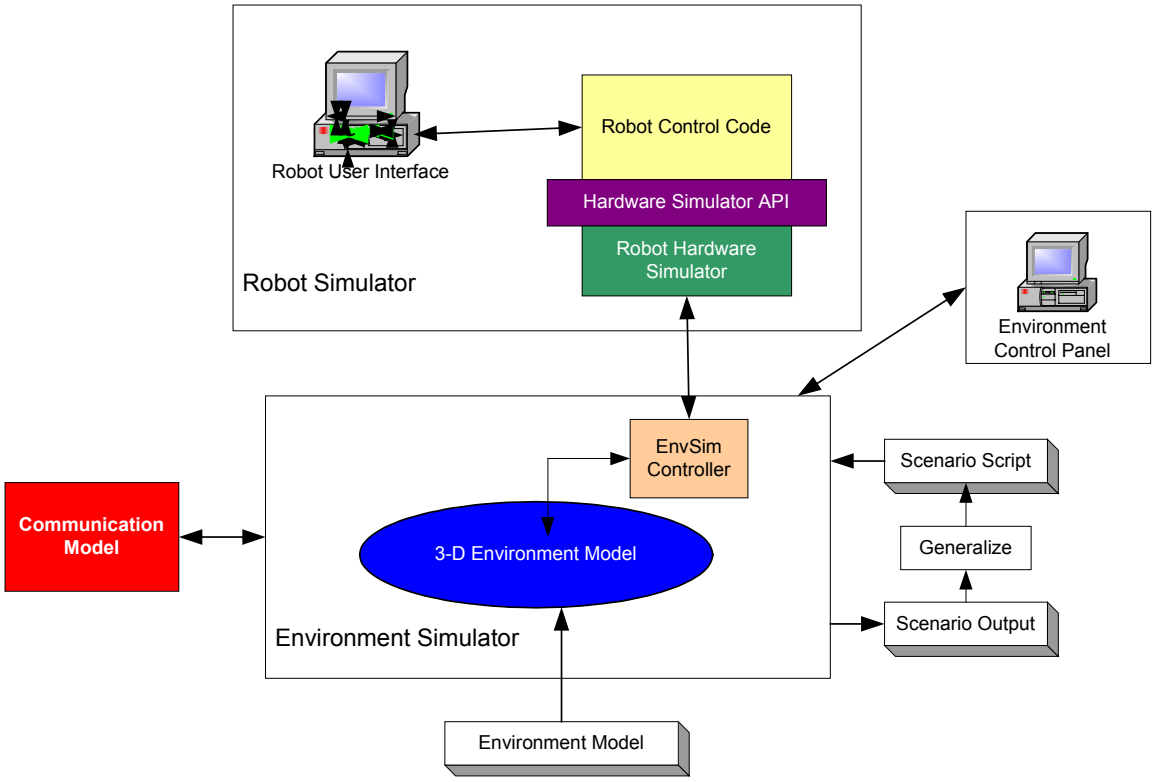

Figure 1. Cooperative Robots Simulator

2. Project Overview

The main focus of this MSE project is to provide basic communication between robot simulators. The Communication Model will be responsible for handling message passing between simulated robots. It will take requests to send messages to other robots and pass them to the correct robot depending on what method of communication. There are two types of communications; broadcast and point-to-point communication. Broadcast and point-to-point communication with appropriate delays will be developed to provide a fundamental communication between simulated robots. Furthermore, point-to-point communication will be extended by include range restriction to support wireless communication. Communication model will also provide an interface to environment control panel to simulate broken links, delay time, range limit and message delivery probability for each robot.

3. Goal

Develop communication capability in Cooperative Robotics Simulator in order to provide a basic communication between simulated robots.

4. Features

4.1 Provide broadcast and point-to-point communication.
4.2 Allow simulated robots to choose method of communication between broadcast and point-to-point communication

4.3 Allow environment control panel to start up or shutdown link between robot simulators.
4.4 Allow environment control panel to set delay time.
4.5 Allow environment control panel to set range limit for point-to-point communication.
4.6 Allow environment control panel to set message delivery probability for each robot.
** The environment control panel will access the interface via environment.

5. Constraints

5.1 This project will support capable of running in distributed manner, or on a single machine.
5.2 This project will be developed in Java using standard libraries. It should run in all JDK compliant platforms.

CHAPTER 2.

SOFTWARE REQUIREMENT SPECIFICATION

---

1. Introduction

1.1 Purpose

The purpose of this document is to define functionality of "Communication Model for Cooperative Robotics Simulators" project. The intended audiences are major project professor and project committees.

1.2 Scope

This document covers the requirement specification of "Communication Model for Robotics Simulator". The Communication Model for Robotics Simulator will provide fundamental communication for simulated robot to pass messages between simulated robots. The fundamental communication method consists of broadcast and point-to-point communication.

1.3 Definitions, acronyms and abbreviations

1.3.1 Broadcast communication refers to one-to-many communication which message is originated in one site and distributed to all simulated robots within the same local area network.

1.3.2 Point-to-point communication refers to one-to-one communication which each message has only one specified destination address.

1.3.3 Propagation delay refers to the time lag between the departure of a signal from the source and the arrival of the signal at the destination.

1.3.4 Range limit refers to the longest range which each simulated robot can communicate with others simulated robots.

1.3.5 JDK or A Java Development Kit is a program development environment for writing Java applications.

1.4 Overview

The rest of this document provides more detail in requirement specification of this software. Section 2 describes overall description including product perspective, product functions, user characteristics, constraints and assumptions. Section 3 provides specific requirement and critical use-case diagram.

2. Overall description

2.1 Product Perspective

This project is a part of "Cooperative Robotics Simulators" research project at CIS department, Kansas State University. The Cooperative Robotics Simulators consists of

three main components, Robot Simulator, Environment Control Panel and Environment Simulator. It requires an interface between communication model and environment simulator. This interface provides communication for simulated robots and provides the ability for environment control panel to set some variables for the communication model, which are propagation delay time, broken links, range limit, and message delivery probability of each simulated robot and for the entire system. The environment simulator is the central component for Robot Simulator and Environment Control Panel to connect to Communication Model.

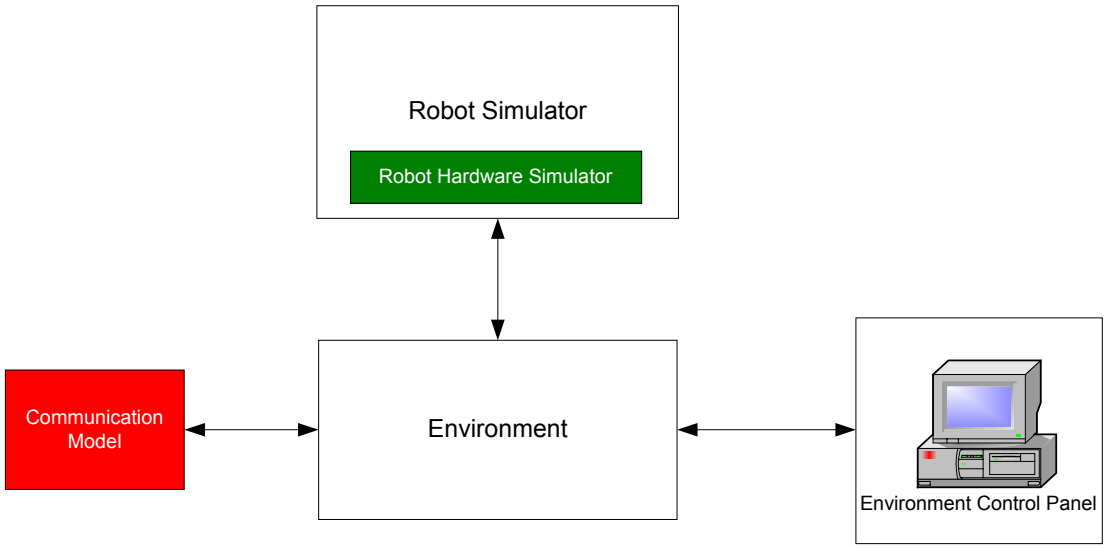

Figure 2. Interfaces between communication model and the other components of the Cooperative Robotics Simulator System.

2.2 Product functions

2.2.1 Provides broadcast communication for simulated robot to send messages within local area network.
2.2.2 Provides point-to-point communication for simulated robot to send messages to another robot.
2.2.3 Allow simulated robots to choose communication methods, which are broadcast, and point-to-point communication.
2.2.4 Allow Environment Control Panel to define propagation delay time.
2.2.5 Allow Environment Control Panel to simulate broken links.
2.2.6 Allow Environment Control Panel to define range limit for each simulated robot.
2.2.7 Allow Environment Control Panel to set message delivery probability for each robot and for the entire system.

** The environment control panel will access the interface via environment.

2.3 User characteristic

The intended user for this software is developer of the Cooperative Robotics Simulator who participates in the component related to communication.

2.4 Constraints

2.4.1 This project will be designed to provide capability of running in distributed manner, or on a single machine.
2.4.2 The project will be developed in Java using standard libraries.
2.4.3 The project should run on all JDK complaint platforms.
2.4.4 The design of the project should not rule out the web interface capability.

3. Specific requirements

3.1 Use cases

3.1.1 Primary High-Level, Use-Case diagram

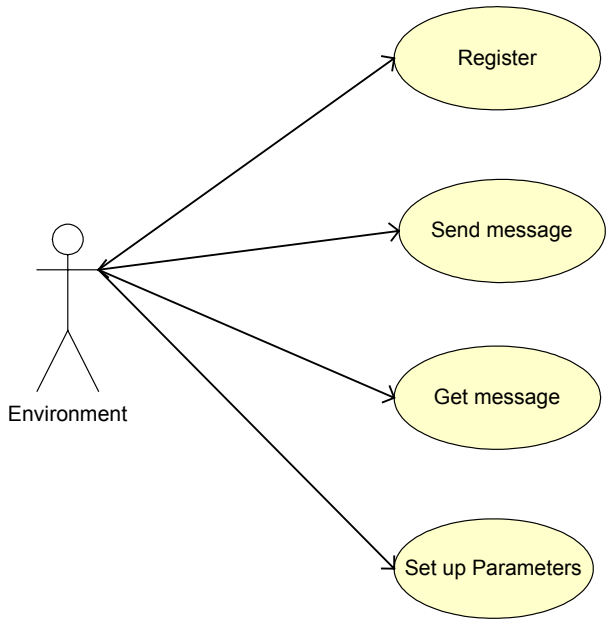

Figure 3. Use-Case Diagram for the Primary, High-Level, Use Cases of the Communication Model for Cooperative Robotics Simulator.

- **Register**

Simulated robot registers to the system before starting communication sessions to allow communication model knows about robots and set some start-up parameters such as having only some types of communication.

- **Send message**

Simulated robot has a request to send messages to the other simulated robots within the same local network.

- **Get message**

Simulated robot has a request to get messages from the system.

- **Set up parameter**

Environment can set the communication parameters, which will be used to control the communication between simulated robots

3.1.2 Primary Low-Level Use-Case diagram

3.1.2.1 Send message Use Case

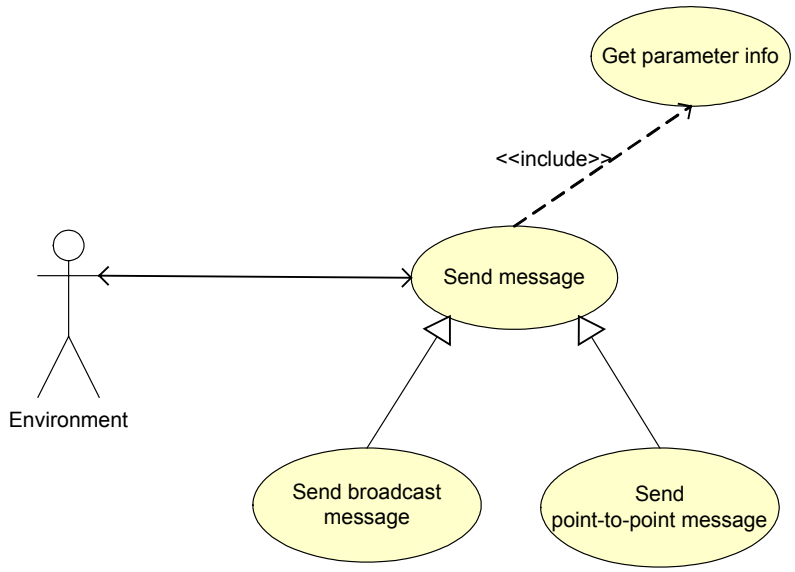

Figure 4. Use-Case Diagram for the Primary Low-Level Use Cases Detailing the Low-Level Use Cases Associated with the High-Level Send message Use Case.

- **Send broadcast message**

Simulated robots can send broadcast message along the local network, which can receive by all simulated robots in the same network.

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

Simulated robot can send message directly to another robot.

- **Get parameter info**

Both send broadcast message and send point-to-point message will send message depend on the communication parameter such as delay time, broken link and range limit.

3.1.2.2 Set up parameters Use Case

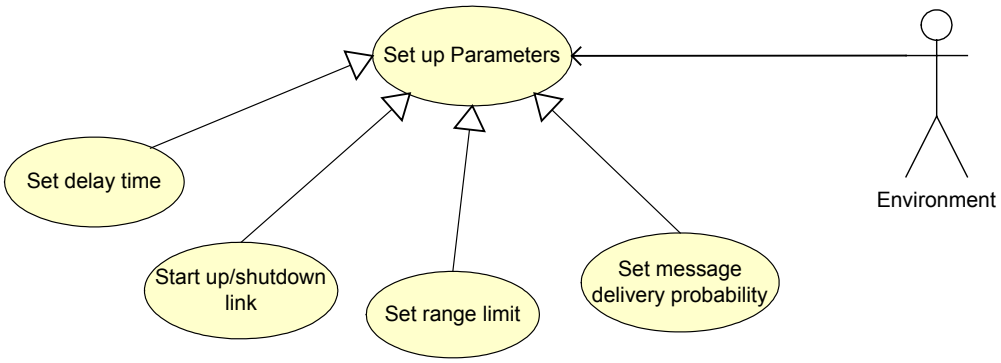

Figure 5. Use-Case Diagram for the Primary Low-Level Use Cases Detailing the Low-Level Use Cases Associated with the High-Level Set up parameters Use Case.

- **Set delay time**

Environment can set delay time, which will create a time lag between the source and the destination. Delay time can set between each pair of robots or for the entire system.

- **Start up / shutdown link**

Environment can start up and also shutdown the communication link for each robot. The link can be activate for send only or receive only or both of them. The communications will success only sending the messages through active link

- **Set range limit**

Environment can set the range limit, which is the longest destination that each simulated robot can send a message.

- **Set message delivery probability**

Environment can set the message delivery probability to simulate message loss in the network.

3.2 Requirements

Register Robot

3.2.1 *Communication model shall support the ability to register robot to the system.
3.2.2 *Communication model shall support the ability to set communication method to each robot after registering. There are two types of communication, broadcast communication and point-to-point communication.

Send broadcast message

3.2.3 *Communication model shall support the ability to send broadcast message.
3.2.4 *Communication model shall support the ability to receive broadcast message.
3.2.5 *Only simulated robot that has broadcast capability shall send or receive broadcast message.

3.2.6 *Simulated robots with active send link shall send broadcast message to other robots.
3.2.7 *Only simulated robot with active receive link shall receive broadcast communication.
3.2.8 Broadcast communication shall support the ability to delay message before sending out.

Send point-to-point message

3.2.9 *Communication model shall support the ability to send point-to-point message.
3.2.10 *Communication model shall support the ability to receive point-to-point message.
3.2.11 *Only simulated robot that has point-to-point capability shall send or receive point-to-point message.
3.2.12 *Point-to-point communication shall delivery to only one simulated robot specified by a destination address.
3.2.13 *Simulated robots with active send link shall send point-to-point message to only robot with active receive link
3.2.14 *Only simulated robot with active receive link shall receive point-to-point message.
3.2.15 Point-to-point communication shall support the ability to delay message before sending out.
3.2.16 Simulated robots with range restriction shall send and receive messages within range limit.

Set up parameters

3.2.17 *Communication model shall support the ability to start up or shutdown all the communication links or each robot. It shall support the ability to activate/deactivate send link only, receive link only or both of them.
3.2.18 Communication model shall support the ability to set delay time for each robot or the entire system.
3.2.19 Communication model shall support the ability to set range limit for each simulated robot or the entire system.
3.2.20 Communication model shall support the ability to set message delivery probability for each robot and for the entire system.

The item with * indicates the critical requirements.

CHAPTER 3

PROJECT PLAN

---

1. Introduction

This document provides an overview of project plan for Communication Model for Cooperative Robotics Simulator project.

1.1 Purpose

The purpose of this document is to provide a project plan, cost estimation and architecture elaboration plan of "Communication Model for Cooperative Robotics Simulator" project.

1.2 Scope

This document covers project plan for "Communication Model for Cooperative Robotics Simulator" including project schedule, cost estimation and architecture elaboration plan. Project plan will detail the phases, iterations, and milestones that will comprise the project. Cost estimation will provide detailed estimate on the size, cost and effort required for the project. The Architecture Elaboration plan will define the activities and actions that must be accomplished prior to the Architecture Presentation.

2. Project schedule

Phase I Inception Phase	**February 2 – February 27**
Study on material of presentation 1	February 2 – February 3
Project Plan	February 4 – February 6
Project overview	February 4 - February 5
Software Quality Assurance	February 4 – February 5
Architecture elaboration plan	February 5 –February 6
Software Requirements Specifications	February 9 – February 11
Cost Estimation	February 10 – February 11
Software demo	February 10 – February 13
First Presentation slides	February 16 – February 17
First presentation	February 27
Phase II Elaboration Phase	**February 26 – April 26**
Review formal requirement specification	February 26 – February 27
Refine vision document	February 27 – March 1
Architecture design	March 4 – March 10
Formal requirement specification	March 16 – March 23
Executable architecture prototype	March15 –March 22

Inspector checklist	March 23 – March 24
Implementation plan	March 29 – March 31
Updated project plan	April 1 – April 2
Updated cost estimation	April 5 – April 6
Test plan	April 6 – April 12
Second Presentation slides	April 21 – April 22
Second presentation	April 26
Phase III Construction Phase	**June 23 – August 19**
Modify Phase II document	June 23 – June 29
Create component diagram	June 29 – July 7
Coding	July 8 – July 13
Testing	July 14 – July 28
User manual	July 29 – July 30
Project evaluation	August 2 – August 6
Final presentation slides	August 9 – August 10
Prepare final documentation	August 11 – August 12
Final presentation	August 19

3. Cost Estimation

3.1 Function Point Analysis

First, Function Point Analysis is used to estimate number lines of code. The project's features are classified into five categories, inputs, outputs, inquiries, files and interfaces. The whole process of FPA consists of three major parts, which are calculating Unadjusted Function Points (UFP), calculate Adjusted Function Point (AFP) and calculate SLOC

3.1.1 Calculate Unadjusted Function Points

Measurement Parameters		**Simple**	**Average**	**Complex**	**Total**
Input	Sign on request Request to send messages. Request to start up/ shutdown links Set up parameters	3 3 3 3			12
Outputs	Success or failure to sign on Success or failure to set parameters. Distribute messages to all recipients	4 4	 5		13
Inquiries					0
Files					0
External Interfaces					0
Total	**Unadjusted Function Points**				**25**

3.1.2 Calculate Adjusted Function Points

To compute Adjusted Function Points, the following equation is used.

$$FP = \text{Unadjusted Function Points} * (0.65 + 0.01 * \Sigma F_i)$$

(where Fi are complexity adjustment factors)

The system is rated on a set of complexity adjustment factors on a scale from 0 to 5 where 0 is no influence and 5 is essential.

Complexity Adjustment Factors	Value
Are data communications required	5
Is the code designed to be reusable	3
Total Complexity Adjustment Factors	**8**

Therefore, total Adjusted Function Points is 25 * 0.73 = 18.25

3.1.3 Calculate number of lines of code (SLOC)

Language factor of Java programming is 50. Therefore, SLOC = 21.17 * 50 = 912.5 or 0.9125 KSLOC

3.2 COCOMO I

COCOMO or COnstructive COst Model developed by Barry Boehm will be used to estimate total cost the project in terms of time and effort, since COCOMO II is used widely in large development team. The cost estimation is based on that this project is uncomplicated; therefore the organic mode cost estimation relationship will be used.

$$Effort = 3.2 * EAF * (Size)^{1.05}$$

where:

Effort = number of person-months

EAF = an effort adjustment factor that characterizes the domain, personnel, environment, and tools used to produce the artifacts of the process. (Since EAF is difficult to determine, EAF is not considered at this time.)

Size = size of the end product (in human-generated source code), measured by the number of delivered source instructions.(in KSLOC)

$$Effort = 3.2 * (0.9125)^{1.05}$$

Effort = 2.90 person-months

$$Time = 2.5 * (Effort)^{0.38}$$

where:

Time = total number of months

$$Time = 2.5 * 2.90^{0.38}$$

Time = 3.74 months

In conclusion, this project requires one person to complete in 3.74 months. As Boehm mentioned, there are 152 working hours in a month. However number of working hours per person in a month is varied, time to complete this project will be different.

4. Architecture Elaboration Plan

4.1 Purpose

The Architecture Elaboration Plan will define the activities and actions that must be accomplished before the Architecture Presentation.

4.2 Updated Vision Document

The Vision documents, project overview and Software Requirements Specifications, will be updated with any modification based on the project committees' suggestion. The requirements will be ranked according to importance and a set of critical requirements will be identified.

4.3 Updated Project Plan

The project plan will be adjusted with any modification, corrected phase and deliverables along with the estimated completion date.

4.3.1 Cost Estimation

The cost estimation will be updated on the size, cost and effort required for the project implementation

4.3.2 Implementation Plan

The Implementation Plan will be developed to define the activities that must be accomplished during implementation.

4.4 Architecture Design

The complete architecture design will be documented using UML diagram such as class diagram, object diagram and sequence diagram.

4.5 Formal Requirement Specification

The class diagram from architecture design will be formally specified using UML/OCL methodology. The USE tool, a UML-based Specification Environment, will be used to implement UML/OCL.

4.6 Test Plan

The test plan will be developed to address the required tests to show that the product satisfies the requirements. Evaluation criteria for all critical requirements will be included in the test plan.

4.7 Formal Technical Inspection

The class diagram will be subjected to a formal technical inspection by two MSE students (inspectors), Esteban Guillen and Kevin Sung. Each inspector will provide a report on the result of his/her inspection that will be documented.

4.8 Executable Architecture Prototype

An executable prototype will be built in one or more iterations. The prototype will identify all the critical requirements, which is addressed in the vision document.

CHAPTER 4

SOFTWARE QUALITY ASSURANCE PLAN

---

1. Purpose

The objective of this document is to define the Software Quality Assurance Plan of the "Communication Model for Cooperative Robotics Simulator" which will be used throughout the software life cycle.

2. Reference Documents

- IEEE Guide for Software Quality Assurance Planning Std 730.1-1995.
- IEEE Standard for Software Quality Assurance Plans Std 730-1998.

3. Management

3.1 Organization

The organization of this project consists of a major professor, two committees, a developer, which is MSE student and two formal technical inspectors.

Major Processor

Dr. William H. Hsu

Committee

Dr. Scott A. DeLoach
Dr. William J. Hankley

Developer

Acharaporn Pattaravanichanon

Formal Technical Inspector

Esteban Guillen
Kevin Sung

3.2 Tasks

All the tasks in three phases: inception, elaboration and construction phase, to be performed in this project will be delineated in project plan.

3.3 Responsibilities

Major professor and committees are responsible for participating in three presentations and demonstrations given by the developer, give suggestion and provide feedback. In addition, major Professor will supervise, evaluate and approve every task done by the developer.

Developer will responsible for the entire task in developing this project throughout the software life cycle including planning, designing, implementation and documentation. The developer will responsible for scheduling the time for reviewing with major

professor and committees for all three phases. Furthermore, the developer will perform all tasks under the supervision of the major professor and suggestion of committees.

Formal Technical Inspectors will responsible for reviewing the architecture design artifact and submit a report, which will contain cover letter and a formal checklist that prepared by the developer.

4. Documentation

4.1 Purpose

The main purpose of the documentation is to ensure the development, verification and validation, use and maintenance of the software. The required documents are followed according to the MSE portfolio guideline for MSE student at CIS department of Kansas State University.

4.2 Minimum documentation requirements

4.2.1 Software Requirement Specification (SRS)

The SRS will describe the essential functionality of the software and external interfaces. The SRS can be modified in architecture phase to ensure that all the essential requirements are included.

4.2.2 Software Design Description (SDD)

SDD depicts how the software will be structured to satisfy the requirements in the SRS. It describes components and subcomponent of software design. Object diagram, class diagram and sequence diagram will be created using Rational Rose.

4.2.3 Project Plan

Project Plan will delineate time period, job and milestone. It can be adjusted in each phase to present the actual status of the project.

4.2.4 Software Test Plan

Software Test Plan will address the required test to show that the product satisfies the requirements.

4.2.5 Project Evaluation

The development process will be reviewed and evaluated including the accuracy of the estimation and the usefulness of the methodologies.

4.2.6 User Manual

User Manual will provide an overview, user commands, error messages and data formats.

All documents will be posted on the developer web page before presentation of each phase.

5. Standard, Practices, Conventions and Metrics

5.1 Purpose
This section describes the standards, practice, conventions and metrics used in Communication Model for Cooperative Robotics Simulator project.

5.2 Content

5.2.1 Documentation Standards
The IEEE standards will be used to develop some documents (where applicable). All documents will have brief explanation describing purpose of the document and version number, which will be increased by 0.1.

5.2.2 Logic Structure Standards
UML notation will be used for the analysis and design documents such as use case diagram, class diagram and sequence diagram.

5.2.3 Coding Standards
The software will use all standards coding practices; for examples the indent for each control structure, embedded standard comment.

5.2.4 Testing Standards
Unit testing, integration testing and acceptance testing will be conducted during testing process. All modules must pass unit testing before doing integration testing. Acceptance testing will be based on the evaluation criteria, which are outlined in test plan.

6. Reviews and Audits

6.1 Purpose
All documentation and software produced during the project will be subjected to regular reviews to ensure the highest level of correctness and quality. This section describes how the reviews and audits are accomplished.

6.2 Minimum Requirements
The three presentations will be conducted during the software development process. All documents for presentation must be submitted to major professor and committees at least one week before presentation. Moreover, there are two formal technical inspectors, which are MSE students. They will participate in reviewing architecture. The formal technical inspectors will provide a review report, which will be one of the required documents of the final presentation.

7. Test

The test plan will provide all the test activities, evaluation criteria, unit testing, system integration testing, and acceptance testing. All tests must meet the evaluation criteria. The test report will provide all test conducted and test result.

8. Problem Reporting and Corrective action

Problem that occurs during the software development will be logged for tracking, resolving and reporting. Only critical problem that impact the project progression will be reported to major professor the others will be resolved, recorded and informed to major professor.

9. Tools, techniques, and methodologies

UML notation is used for the analysis and design documents using Rational Rose 2002. Microsoft Project will be used for creating project plan. Eclipse will be used for software development.

10.Code control

The developer uses her own computer and maintains her software version using several subdirectories. The code and the binary files will be organized in different directories. The backup copies will be created every week on CD-ROM.

11.Training

CIS771 Software Specification
CIS748 Software Management
CIS740 Software Engineering
Furthermore, the developer will responsible for doing research about the system. The major professor and committees will provide additional knowledge of the system. Attending in weekly meeting of the Cooperative Robots Simulator research group will help in deeply understand of the system.

CHAPTER 5

ARCHITECTURE DESIGN

---

1. Introduction

The purpose of this document is to provide architecture design including class diagram, class diagram description, sequence diagram and sequence diagram description of "Communication Model for Cooperative Robotics Simulator". The architecture design is based on the Software Requirement Specification Version 1.1.

2. Class Diagram

There are eight classes in the system. The CommunicationsSystem class is served as a main interface, which interact with the environment. It provides function for robot and environment control panel.

The functions for robot are used to communicate with other robots, which are registerRobot, sendMessage and getMessage. Each simulated robot needs to register to the communication model to initiate the communication session and set what communication type is allowed, broadcast or point-to-point. When the environment takes requests from robot, it will pass them to the communication model by calling sendMessage in every time step. After communication model gets request to send message from the environment, it will process the message and add it to the receiver's priority queue. Same as sendMessage, the environment will call getMessage in every time step and pass the returned messages to the specified robot.

On the other hand, the functions for environment control panel are used to set robot's and system's parameter. The system parameters include system link status, system range, system delay and system delivery probability. The robot parameters are incoming and outgoing link status, maximum sending range, delay, delivery probability, broadcast and point-to-point ability.

The RobotCommRecord class is used to keep current robot parameter. Each RobotCommRecord has its own priority queue to keep messages, which will be retrieved when getMessage function is called. In other words, priority queue keeps a list of messages, which will be delivered to the corresponding robot

The PriorityQueue class keeps a set of messages ordered by receivedTime. After messages are delivered, they will be removed from the queue.

Since some parameters are defined as a value for each pair of robot, the RobotParameter class is created to keep the parameter for the owner robot and the other robots.

The Message class defines the sender, receiver, message content, sent time and received time. The sent time is the time step that message is sent by the environment to the communication model. The received time is the expect time step that robot will get the message.

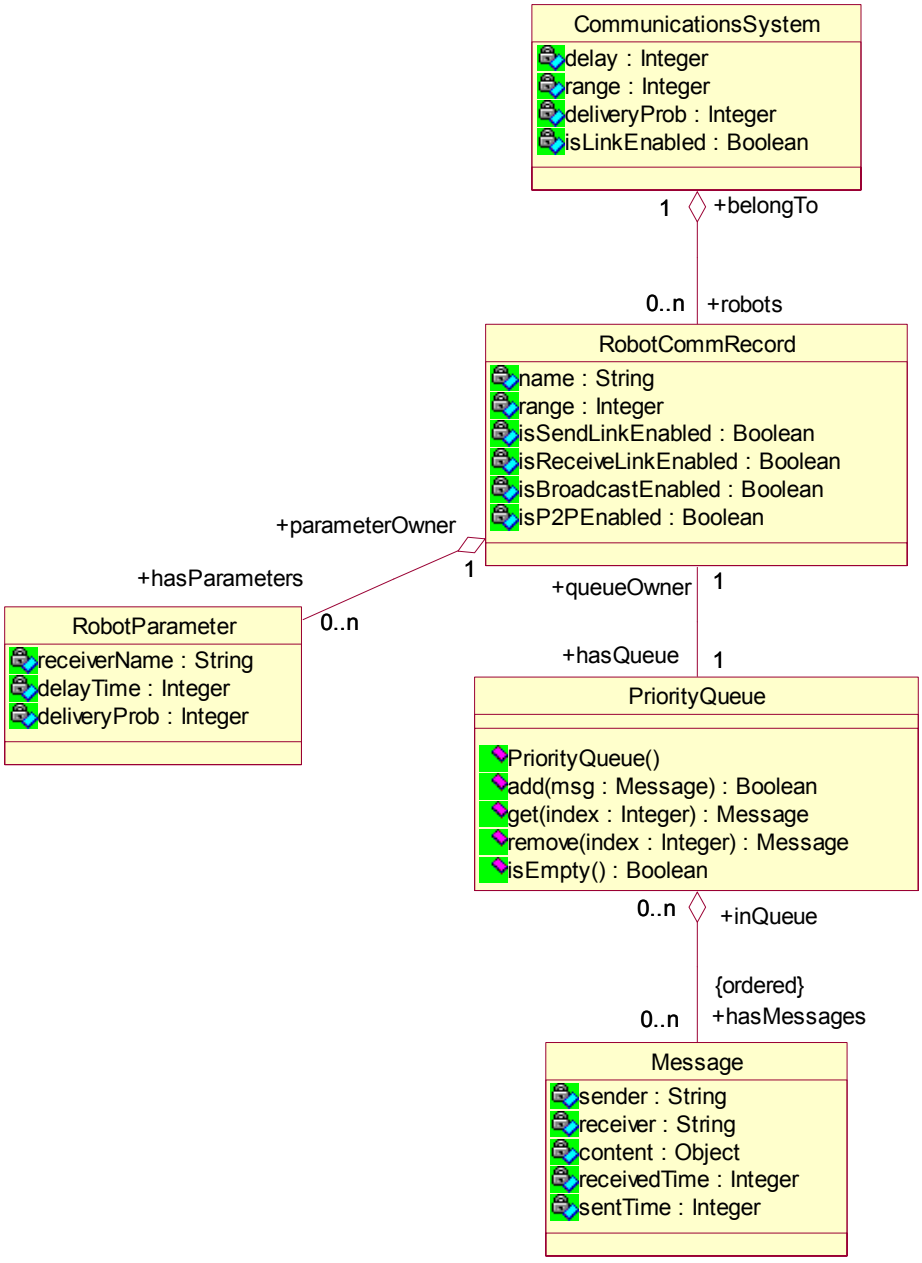

Figure 6. Communication Model for Cooperative Robotic Simulator Class Diagram

3. Sequence diagram

This section shows the sequence diagrams of the basic scenarios of Communication Model, which are register robot, send message and get message. The send message scenario includes two cases: sending broadcast and sending point-to-point

3.1 Register robot

After every robot registers to the environment, the environment will send the request to register robot with robot name and communication type to the CommunicationsSystem. If this robot name does not exist in the system, RobotCommRecord object will be created and set name as robot name. Then it will create their own priority queue and set communication ability: broadcast, point-to-point or both depends on the communication type.

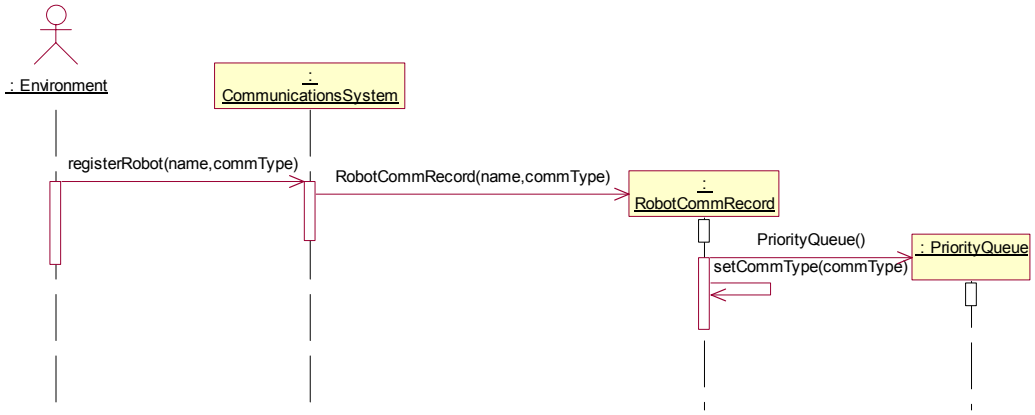

Figure 7. Register Robot Sequence Diagram

3.2 Send message (broadcast)

Environment sends a request to send message with message and time step as parameters. If the system link is enabled, then it will check if this sender has the ability to send broadcast message. If the sender has ability to send broadcast message, it will check each robot in the system if they qualify to get this broadcast message. If they qualified, the message will be added to their priority queue.

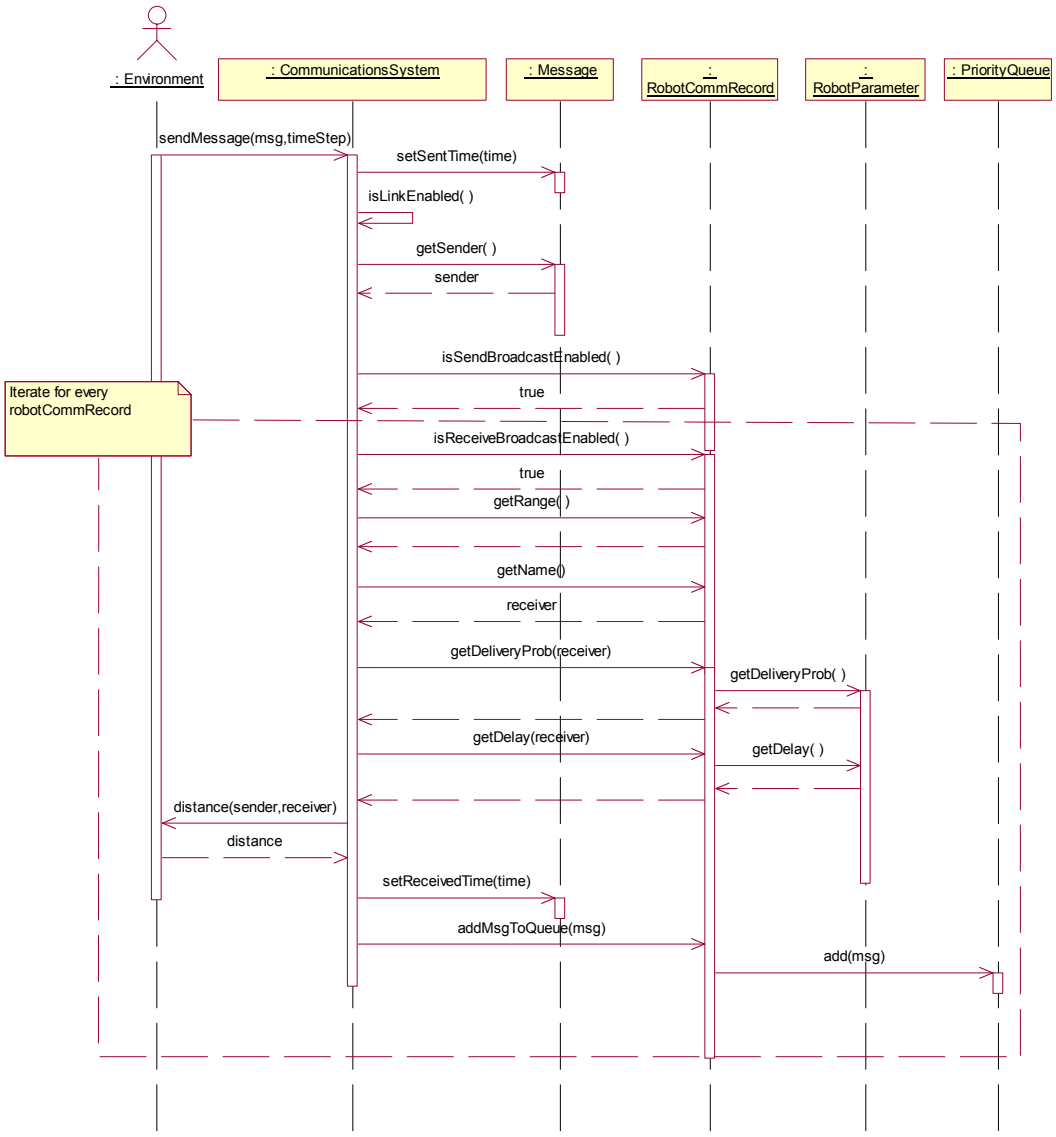

Figure 8. Send Message (Broadcast Message) Sequence Diagram

3.3 Send message (point-to-point)

Environment sends a request to send message with message and time step as parameters. If the system link is enabled, then it will check if this sender has the ability to send point-to-point message. If the sender has ability to send point-to-point message, it will check if the receiver qualifies to get this message. If it qualifies, the message will be added to the priority queue.

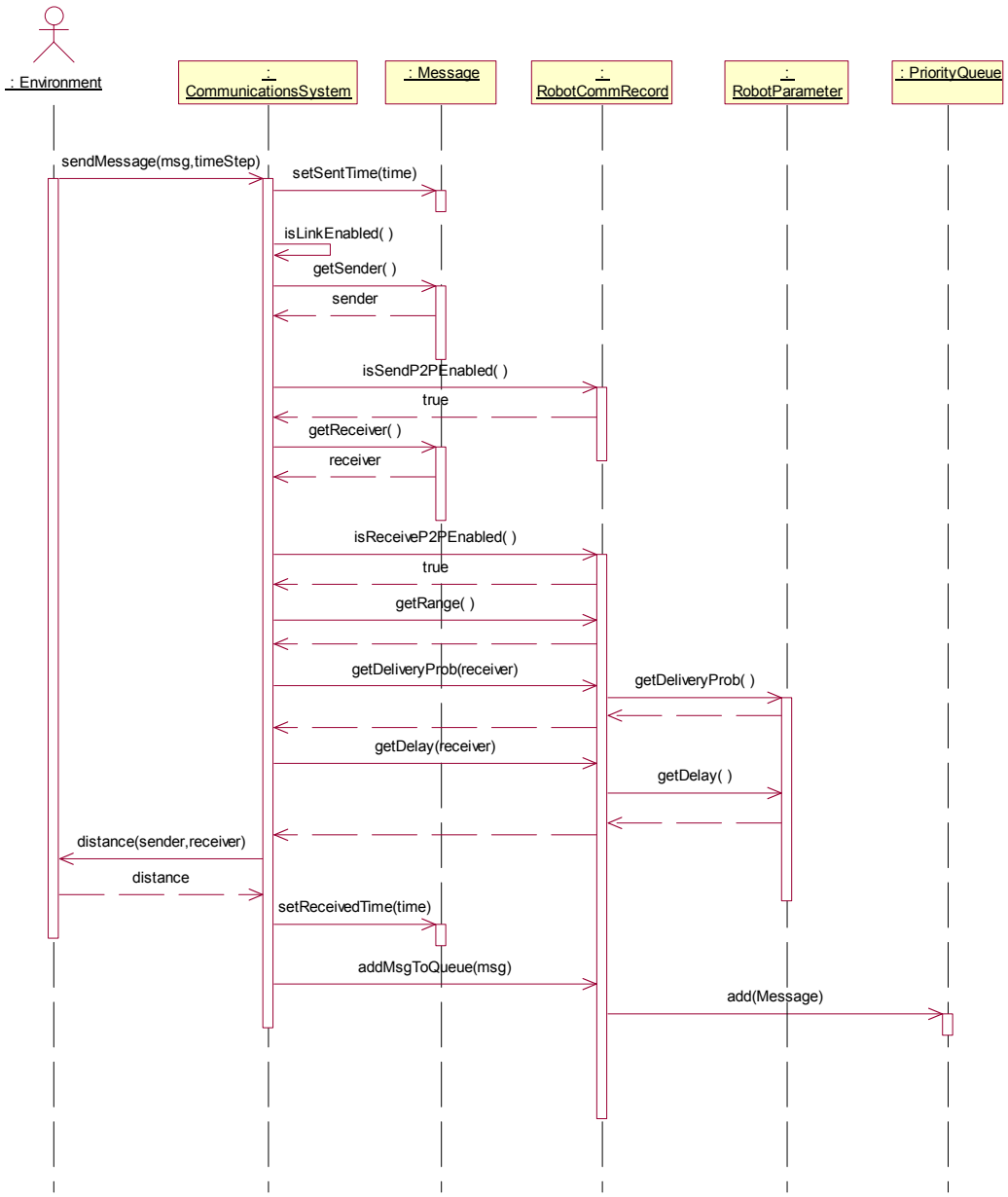

Figure 9. Send Message (Point-to-point Message) Sequence Diagram

3.4 Get message

Environment sends request to retrieve message for each robot in every time step. The system will retrieve the messages which has received time equals to time step from the robot's priority queue.

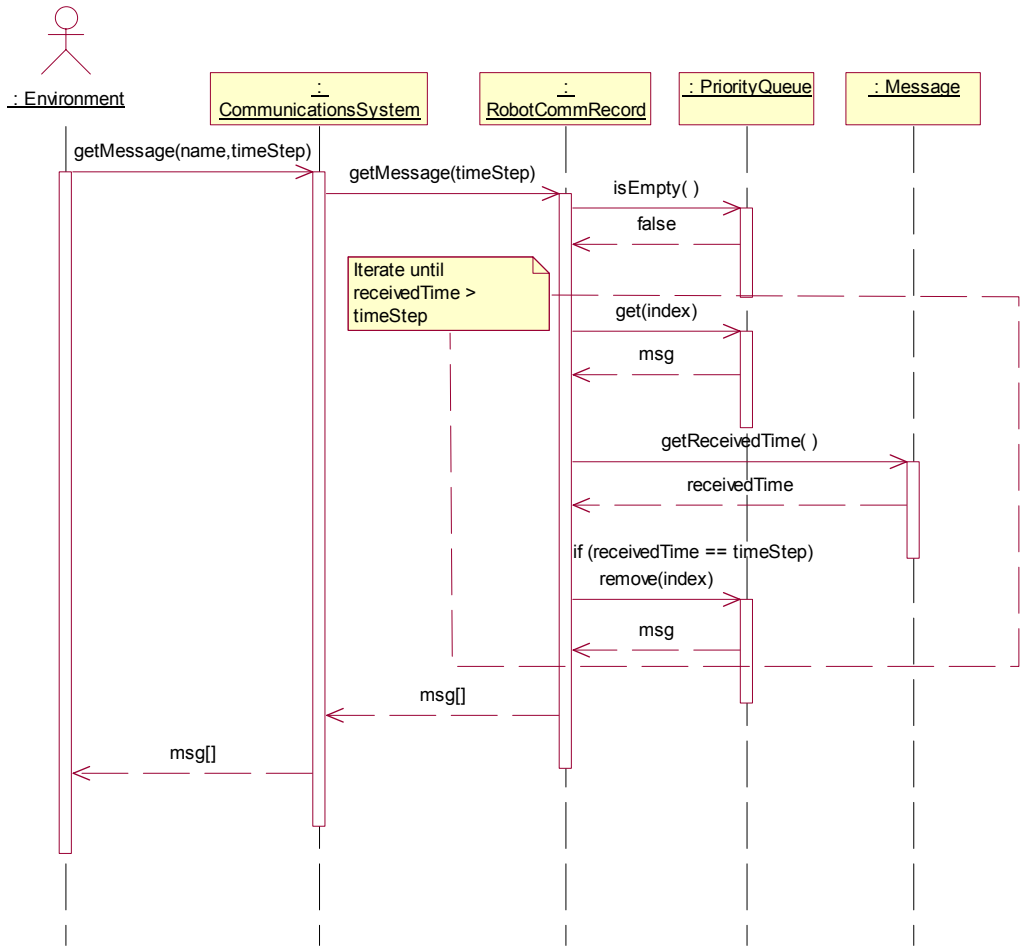

Figure 10. Get Message Sequence Diagram

4. Class Description

This section provides class description in detail.

4.1 CommunicationsSystem

This class is the main interface of the system, which provides a set of methods used by Environment. The environment uses the interface for sending/receiving messages and passes them to the targeted robot. The environment also uses a set of methods to set and get parameter values and pass to environment control panel. The parameters are system parameters, and robot parameters. This class also used to keep system parameter including range, link, delivery probability and delay.

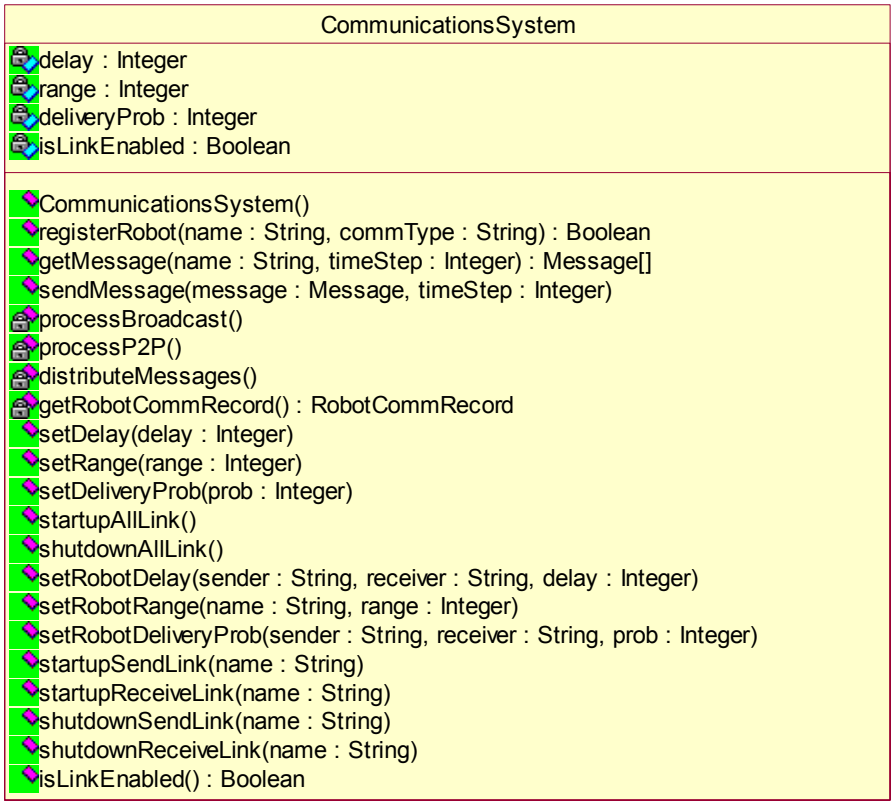

Figure 11. CommunicationsSystem class with attributes and operations.

4.2 RobotCommRecord

This class is used to keep robot parameters, which are range, active link, delivery probability, delay, and broadcast or point-to-point ability. It provides methods to set these parameters. In addition, it validates each incoming message, and distribute message to the specified robot.

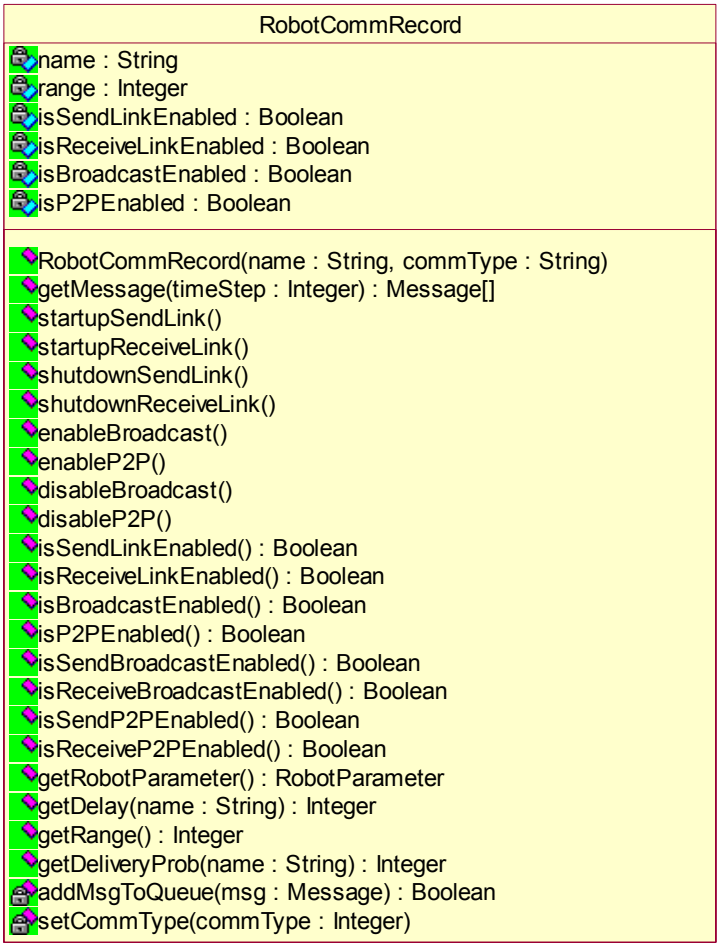

Figure 12. RobotCommRecord class with attributes and operations.

4.3 RobotParameter

It keeps the parameters for each pair of robot (the owner and the other robots).

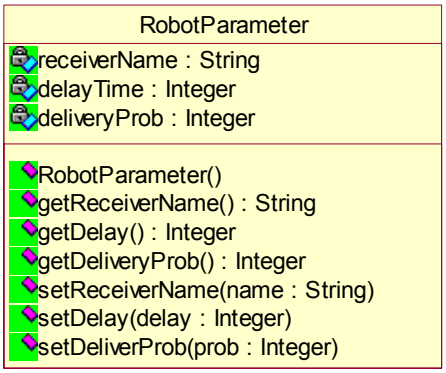

Figure 13. RobotParameter class with attributes and operations.

4.4 PriorityQueue

This class is the priority queue, which will keep outgoing messages (in terms of this system, but it is the incoming message for robot) ordered by received time. It also provides method to add and remove from queue.

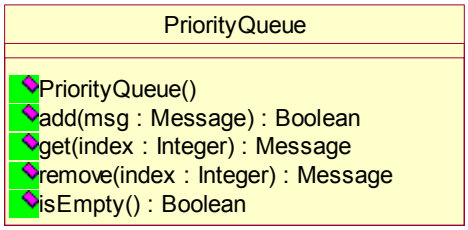

Figure 14. PriorityQueue class with attributes and operations.

4.5 Message

This class defines the field use to communicate between robots including sender, receiver, sent time, received time, and message content.

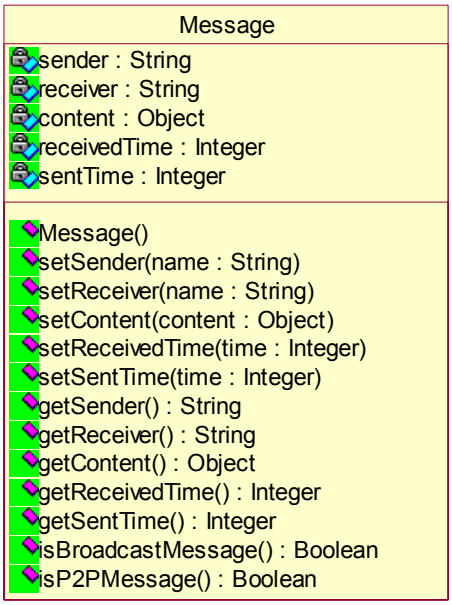

Figure 15. Message class with attributes and operations.

5. CommunicationsSystem State Diagram

The State Diagram shows the possible ways in which the objects respond to events, which occur in the system. There are five states in the CommunicationsSytem State Diagram, wait, registering, sending, sending broadcast, sending point-to-point, adding to queue and getting message.

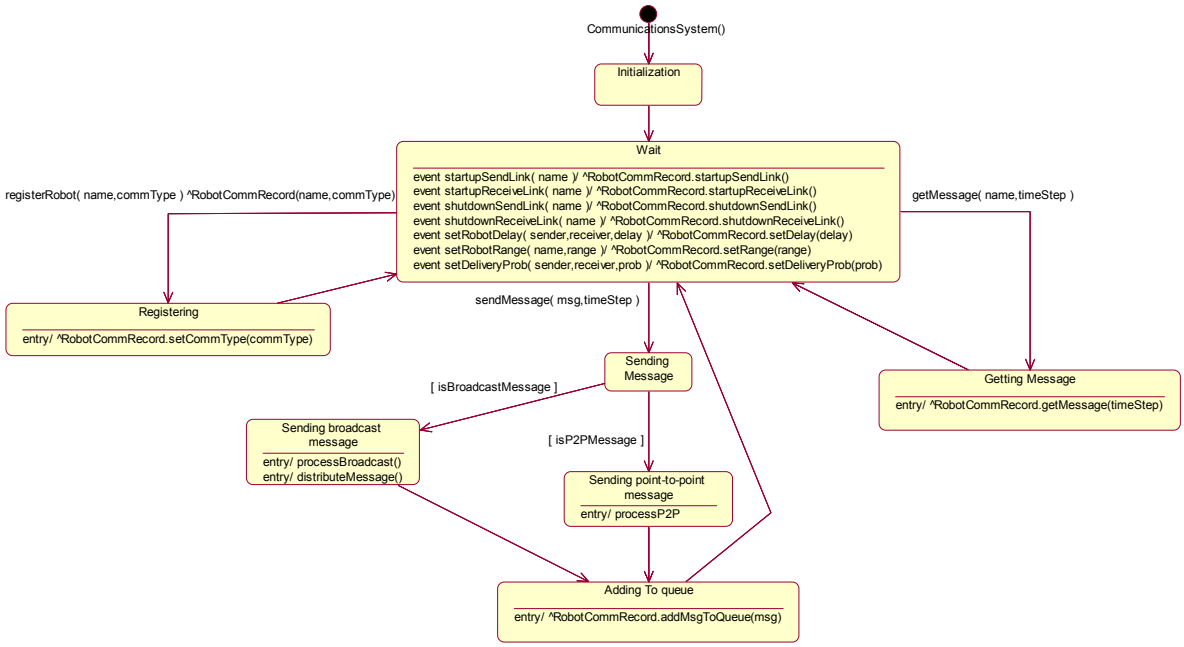

Figure 16. CommunicationsSystem State Chart Diagram.

CHAPTER 6

FORMAL REQUIREMENT SPECIFICATION

---

1. Introduction

The purpose of this document is to provide formal requirement specification of "Communication Model for Cooperative Robotics Simulator". This specification uses UML/OCL methodology. The constraint and variant in the specification are based on the critical requirement as stated in the Software Requirement Specification Version 1.1 and Class Diagram presented in Architecture Design Version 1.0. Furthermore, we use the UML- based Specification Environment (USE) tool to check the type and syntax to ensure correctness of the specification.

2. Scope

In the specification, the variant, pre and post condition of interest properties are defined to ensure that these properties will hold in the system model. These properties are:

- Robot has unique name.
- Only robot with broadcast ability can send and receive broadcast message.
- Only robot with point-to-point ability can send and receive point-to-point message.
- Only robot with active send link can send message.
- Only robot with active receive link can receive message.
- Messages are kept into the right queue.
- Messages are kept in priority queue ordered by received time.
- If message's received time is defined, then received time is equal or greater than sent time.
- If all links are shutdown, robot cannot send or receive message.
- Robots cannot receive their own sending message.

3. Formal Specification Description

This section explains the Communication Model component specification based on Class diagram presented in Architecture Design Version 1.0

3.1 Classes

3.1.1 CommunicationsSystem class

There is one attribute in this class, which is isLinkEnabled, used to keep status of the system link. RegisterRobot is the operation used for registering a robot to the communication model and set type of communication that will be allowed for this robot. This operation has two parameters, n-robot name and c-communication type. There are

three possible values of communication type, 1 – broadcast communication, 2 – point-to-point communication and 3 – both.

```
class CommunicationsSystem
attributes
  delay : Integer
  range : Integer
  deliveryProb : Integer
  isLinkEnabled : Boolean
operations
  registerRobot(n:String,c:Integer)
  getMessage(n:String,timeStep:Integer):Set(Message)
  sendMessage(msg:Message,timeStep:Integer)
  processBroadcast()
  processP2P()
  distributeMessage()
  getRobotCommRecord():RobotCommRecord
  setDelay(delay:Integer)
  setRange(range:Integer)
  setDeliveryProb(prob:Integer)
  startupAllLink()
  shutdownAllLink()
  setRobotDelay(sender:String,receiver:String,delay:Integer)
  setRobotRange(name:String,range:Integer)
  setRobotDeliveryProb(sender:String,receiver:String,prob:Integer)
  startupSendLink(name:String)
  startupReceiveLink(name:String)
  shutdownSendLink(name:String)
  shutdownReceiveLink(name:String)
  isLinkEnabled():Boolean
end
```

3.1.2 RobotCommRecord class

The attributes of this class are name, range, isSendLinkEnabled, isReceivedLinkEnabled, isBroadcastEnabled and isP2PEnabled. The attribute name is used for storing name of robot. Range is used for keeping the maximum sending range. The attribute isSendLinkEnabled is used for storing the status of the outgoing link of the robot. The attribute isReceiveLinkEnabled is used for storing the status of the incoming link of the robot. The attribute isBroadcastEnabled is used for storing if broadcast communication is allowed for this robot. Finally, the attribute isP2PEnabled is used for storing if point-to-point communication is allowed for this robot.

```
class RobotCommRecord
attributes
  name : String
  range : Integer
  isSendLinkEnabled : Boolean
  isReceiveLinkEnabled : Boolean
  isBroadcastEnabled : Boolean
  isP2PEnabled : Boolean
operations
  getMessage(timeStep:Integer):Set(Message)
  startupSendLink()
  startupReceiveLink()
```

```
  shutdownSendLink()
  shutdownReceiveLink()
  enableBroadcast()
  enableP2P()
  disableBroadcast()
  disableP2P()
  isSendLinkEnabled():Boolean
  isReceiveLinkEnabled():Boolean
  isBroadcastEnabled():Boolean
  isP2PEnabled():Boolean
  isSendBroadcastEnabled():Boolean
  isReceiveBroadcastEnabled():Boolean
  isSendP2PEnabled():Boolean
  isReceiveP2PEnabled():Boolean
  getRobotParameter():RobotParameter
  getDelay(name:String):Integer
  getRange():Integer
  getDeliveryProb(name:String):Integer
  addMsgToQueue(msg:Message):Boolean
  setCommType(commType:Integer)
end
```

3.1.3 RobotParameter class

This class is used to keep the parameter for each pair of robot such as delay time, delivery probability.

```
class RobotParameter
attributes
  receiverName : String
  delayTime : Integer
  deliverProb : Integer
operations
  getReceiveName():String
  getDelay():Integer
  getDeliveryProb():Integer
  setReceiveName(name:String)
  setDelay(delay:Integer)
  setDeliveryProb(prob:Integer)
end
```

3.1.4 PriorityQueue class

This class is used to keep outgoing messages for each robot.

```
class PriorityQueue
attributes
operations
  add(msg:Message):Boolean
  get(index:Integer):Message
  remove(index:Integer):Message
  isEmpty():Boolean
end
```

3.1.5 Message class

This class is the format of the message that will be used to communicate with other robots. The attributes are sender, receiver, receivedTime, sentTime and content. The receivedTime is the time, which the message will be sent out of the system. It is also the time in which the receiver will get this message. The sentTime is the time, which the sender sent the message. It is also the time that the communication system gets this message.

```
class Message
attributes
  sender : String
  receiver : String
  content : MessageContent
  receivedTime : Integer
  sentTime : Integer
operations
  setSender(name:String)
  setReceiver(name:String)
  setContent(content:MessageContent)
  setReceivedTime(time:Integer)
  setSentTime(time:Integer)
  getSender():String
  getReceiver():String
  getContent():MessageContent
  getReceivedTime():Integer
  getSentTime():Integer
  isBroadcastMessage():Boolean
  isP2PMessage():Boolean
end
```

3.2 Associations

```
association robot between
  CommunicationsSystem[1] role belongTo
  RobotCommRecord[*] role robots
end


association parameter between
  RobotCommRecord[1] role parameterOwner
  RobotParameter[*] role hasParameters
end


association queue between
  RobotCommRecord[1] role queueOwner
  PriorityQueue[1] role hasQueue
end


association message between
  PriorityQueue[*] role inQueue
  Message[*] role hasMessages ordered
end
```

3.3 Invariants

3.3.1 Robot has unique name.

```
context RobotCommRecord
  inv UniqueName:
      RobotCommRecord.allInstances->forAll(p1,p2| p1 <> p2
      implies p1.name <> p2.name)
```

3.3.2 Only robot with broadcast ability can send and receive broadcast message.

```
context RobotCommRecord
  inv BroadcastAbility1:
      RobotCommRecord.allInstances.hasQueue.hasMessages
        ->select(receiver='broadcast' and sender=self.name)->notEmpty
        implies self.isBroadcastEnabled = true


context r:RobotCommRecord
  inv BroadcastAbility2:
      r.hasQueue.hasMessages
        ->select(receiver='broadcast')->notEmpty
        implies r.isBroadcastEnabled = true
```

3.3.3 Only robot with point-to-point ability can send and receive point-to-point message.

```
context RobotCommRecord
  inv P2PAbility1:
      RobotCommRecord.allInstances.hasQueue.hasMessages
        ->select(receiver <> 'broadcast' and sender=self.name)
        ->notEmpty implies self.isP2PEnabled = true

context r:RobotCommRecord
  inv P2PAbility2:
      r.hasQueue.hasMessages
        ->select(receiver <> 'broadcast')->notEmpty
        implies r.isP2PEnabled = true
```

3.3.4 Only robot with active send link can send message.

```
context RobotCommRecord
  inv sendAbility:
      RobotCommRecord.allInstances.hasQueue.hasMessages
        ->select(sender=self.name)->notEmpty
        implies self.isSendLinkEnabled = true
```

3.3.5 Only robot with active receive link can receive message.

```
context r:RobotCommRecord
  inv receiveAbility:
      r.hasQueue.hasMessages->notEmpty implies
      r.isReceiveLinkEnabled = true
```

3.3.6 Messages are kept into the right queue.

```
context RobotCommRecord
   inv rightQueue:
       hasQueue.hasMessages->forAll((receiver=self.name) or
(receiver='broadcast'))
```

3.3.7 Messages are kept in priority queue ordered by received time.

```
context p:PriorityQueue
  inv priorityQueue:
      Sequence{1..(p.hasMessages->size-1)}
      ->forAll(i | p.hasMessages->at(i).receivedTime
      <= p.hasMessages->at(i+1).receivedTime)
```

3.3.8 If message's received time is defined, then received time is equal or greater than sent time.

```
context m:Message
  inv rightTime:
      m.receivedTime.isDefined implies m.receivedTime >= m.sentTime
```

3.3.9 If all links are shutdown, robot cannot send or receive message.

```
context c:CommunicationsSystem
  inv allLinkShutdown:
      RobotCommRecord.allInstances.hasQueue.hasMessages->notEmpty
       implies c.isLinkEnabled = true
```

3.3.10 Robots cannot receive their own sending message.

```
context r:RobotCommRecord
  inv sendToYourself:
      r.hasQueue.hasMessages->forAll(sender <> r.name)
```

3.4 Operations

3.4.1 Register Robot

```
context CommunicationsSystem::registerRobot(n:String,c:Integer)
  pre precond_1:    n.isDefined
  pre precond_2:    c.isDefined
  pre precond_3:    (c=1 or c=2 or c=3)
  pre precond_4:    robots->select(name=n)->isEmpty
  post postcond_1:  robots->exists(r | r.oclIsNew and r.name = n)
  post postcond_2:  robots=robots@pre->union(robots->select(name=n))
  post postcond_3:  robots->select(name=n)->size = 1
  post postcond_4:  (c=1)implies
                    robots->select(name=n and
                    isBroadcastEnabled=true and
                    isP2PEnabled=false)->notEmpty
  post postcond_5:  (c=2) implies
                    robots->select(name=n and
                    isP2PEnabled=true and
                    isBroadcastEnabled=false)->notEmpty
  post postcond_6:  (c=3) implies
                    robots->select(name=n and
                    isBroadcastEnabled=true and
                    isP2PEnabled=true)->notEmpty
```

This part of specification defines pre and post condition of register robot operation. It takes two arguments, n – robot name and c – communication type. The precond_1 states that robot name is defined. The precond_2 states that communication type is defined. The precond_3 states that communication type can be 1, 2 or 3 only. The last pre condition is there is no RobotCommRecord named n in the system before registering this robot. There are six post conditions. The first post condition is RobotCommRecord is new created and named “n”. The second post condition says that the new set of RobotCommRecord is the previous set plus the new RobotCommRecord, which is just created and named “n”. The third post condition states that there is only one RobotCommRecord named “n”. The last three post conditions depend on type of communication. If communication type is 1 then RobotCommRecord, which is named “n”, has isBroadcastEnabled attribute sets to true while isP2PEnabled attribute sets to false. If communication type is 2 then RobotCommRecord, which is named “n”, has isBroadcastEnabled attribute sets to false while isP2PEnabled attribute sets to true. Finally, setting robot to have both broadcast and point-to-point capability, which is if communication type is 3 then RobotCommRecord, which is named “n”, has both isBroadcastEnabled attribute and isP2PEnabled attribute set to true

3.4.2 Send Message operation

```
context CommunicationsSystem::sendMessage(msg:Message,timeStep:Integer)
  pre precond_1:    timeStep > 0 and timeStep.isDefined
  pre precond_2:    msg.isDefined
  pre precond_3:    msg.sender.isDefined
  pre precond_4:    msg.receiver.isDefined
  pre precond_5:    isLinkEnabled = true
```

```
pre precond_6:    robots->select(name=msg.sender)
                   ->forAll(isSendLinkEnabled = true)

pre precond_7:    msg.receiver = 'broadcast'
                   implies robots->select(name=msg.sender)
                   ->forAll(isBroadcastEnabled = true)

pre precond_8:    msg.receiver <> 'broadcast'
                     implies robots->select(name=msg.sender)->
                     forAll(isP2PEnabled = true)

post postcond_1:  msg.sentTime = timeStep

post postcond 2:  msg.receiver <> 'broadcast' implies
                     robots->select(name=msg.sender)->
                     forAll(hasParameters->
                     forAll(receiverName=msg.receiver implies
                     msg.receivedTime = parameterOwner.belongTo.delay
                     + timeStep + delayTime))

post postcond 3:  msg.receiver = 'broadcast' implies
                     robots->select(name=msg.sender)->
                     forAll(hasParameters->
                     forAll(msg.receivedTime =
                     parameterOwner.belongTo.delay
                     + timeStep + delayTime))

post postcond_4:  msg.receiver = 'broadcast' implies
                     robots->select(name <> msg.sender
                     and isReceiveLinkEnabled = true
                     and isBroadcastEnabled = true)
                     ->forAll(r| r.hasQueue.hasMessages->asSet
                     = r.hasQueue.hasMessages@pre
                     ->including(msg)->asSet)

post postcond_5:  msg.receiver <> 'broadcast' implies
                     robots->select(name = msg.receiver
                     and isReceiveLinkEnabled = true
                     and isP2PEnabled = true)
                     ->forAll(r| r.hasQueue.hasMessages->asSet
                     = r.hasQueue.hasMessages@pre
                     ->including(msg)->asSet)
```

This part of specification defines pre and post condition for sendMessage operation. The pre conditions are time step is greater than 0, time step is defined, message is defined, the system link is active, sender's send link is active. Furthermore, if the message is broadcast then sender must have broadcast capability; otherwise sender must have point-to-point capability. The post conditions are message's sentTime and receivedTime are set; the message will be included in the receiver's priority queue if the receiver is qualified. The process of qualifying receiver is check if receiver's receive link is active and if the message is broadcast then the receiver must have broadcast capability; otherwise the receiver must have point-to-point capability

3.4.3 Get Message operation

context CommmunicationsSystem::getMessage(n:String,timeStep:Integer)
:Set(Message)

```
  pre precond_1:    n.isDefined
  pre precond_2:    robots.exists(r| r.name=n)
  pre precond_3:    timeStep.isDefined
  pre precond_4:    timeStep > 0
  post postcond_1:  robots->select(name = n)
                     ->forAll(r | r.hasQueue.hasMessages->asSet
                     = r.hasQueue.hasMessages@pre->asSet
                     - r.hasQueue.hasMessages@pre
                     ->select(receivedTime = timeStep)->asSet)

  post postcond_2:  robots->select(name=n)
                     ->forAll(r| result = r.hasQueue.hasMessages@pre
                     ->select(receivedTime = timeStep)->asSet)
```

This part is the specification for getMessage operation. The post conditions are n (robor name) is defined, there exists robot named n, time step is defined and greater than 0. The post conditions are priority queue of robot n excludes messages which receivedTime is equal to time step and the operation will return the result which is a set of messages which receivedTime is equal to time step.

CHAPTER 7

TEST PLAN

---

1. Test plan identifier

TestPlan-CommRobot-001

2. Introduction

This test plan is used to address the required test of "Communication Model for Cooperative Robotics Simulators". The software test items and features will include the feature as described in Software Requirement Specification Version 1.1

3. Test Items

- CommunicationsSystem class
- RobotCommRecord class
- RobotParameters class
- PriorityQueue class
- Message class

4. Features to be tested

The following list described the features that will be tested based on the Software Requirement Specification.

Feature identifier	Description	Requirement number
T-001	Register robot.	3.2.1,3.2.2
T-002	Sending broadcast message	3.2.3,3.2.5,3.2.6
T-003	Sending point-to-point message	3.2.9,3.2.11,3.2.13
T-004	Receiving broadcast message	3.2.4,3.2.5,3.2.7
T-005	Receiving point-to-point message	3.2.10,3.2.11,3.2.12,3.2.14
T-006	Start up all links	3.2.17
T-007	Shutdown all links	3.2.17
T-008	Set system range	3.2.19
T-009	Set system delay	3.2.18
T-010	Set system delivery probability	3.2.20
T-011	Start up robot's send link	3.2.17
T-012	Shutdown robot's send link	3.2.17
T-013	Start up robot's receive link	3.2.17
T-014	Shutdown robot's receive link	3.2.17
T-015	Enable broadcast capability	3.2.2
T-016	Disable broadcast capability	3.2.2

T-017	Enable point-to-point capability	3.2.2
T-018	Disable point-to-point capability	3.2.2
T-019	Set robot range	3.2.19,3.2.16
T-020	Set robot delay	3.2.18,3.2.15,3.2.8
T-021	Set robot deliver probability	3.2.20

Table 1. Features to be tested

5. Approach

Unit testing

All executable java files, which are identified in section 3, will be tested. The JUnit, a testing framework for java unit testing, will be used to generate automated testing. One or more drivers and stubs will be developed to conduct unit testing. All executable files would be passed before starting integration testing.

Integration testing

The integration testing will include the combined function of several classes to perform the main software feature. Bottom-up integration will be used to conduct integration testing.

System testing

After integration testing, the system testing will be performed with other modules in the system to ensure all requirements are satisfied.

6. Environmental Needs.

6.1 Hardware

The testing will be done on Sun Sparc and Intel-based machine.

6.2 Software

J2sdk version 1.4 is used to compile and execute program.

6.3 Operating systems.

Windows XP Professional
Unix Solaris

7. Test Cases

7.1 Unit testing

7.1.1 Register robot

Test item: CommunicationsSystem class

Input: robot name, communication type (broadcast, point-to-point, or both)

Pass criteria:

- New RobotCommRecord class is created.
- The RobotCommRecord has a name as the input robot name.
- Unique RobotCommRecord.
- Broadcast, point-to-point, or both capabilities are enabled for this robot depending on the communication type.

Fail criteria

- New RobotCommRecord is not created.
- The RobotCommRecord has a name different from the input robot name.
- There is a duplication of RobotCommRecord (has the same name)..

7.1.2 Sending broadcast message.

Test item: CommunicationsSystem class
Input: Message object, time step
Pass criteria:

- All receivers get the message and save it to their priority queue.

Fail criteria:

- One or more receiver does not get the message or not save it to their priority queue.

7.1.3 Sending point-to-point message.

Test item: CommunicationsSystem class
Input: Message object, time step
Pass criteria:

- The specified receiver gets the message and save it to their priority queue.
- The received message is the same as the sending message.

Fail criteria:

- The specified does not get the message or not save it to its priority queue.
- The other receives get the message or save it to their priority queue.
- The received message is not the same as the sending message.

7.1.4 Receiving broadcast message.

Test item: CommunicationsSystem class
Input: robot name, current time step
Output: a set of Message object.
Pass criteria:

- Only Message objects, which has time step equal to current time step, are returned.
- The returned Message objects are removed from robot's priority queue.

Fail criteria:

- One or more returned Message objects, which has time step not equal to current time step, are returned.
- The returned Message objects are not removed from robot's priority queue.

7.1.5 Receiving point-to-point message.

Test item: CommunicationsSystem class
Input: robot name, time step
Output: a set of Message object.
Pass criteria:

- Only Message objects, which have time step equal to current time step, are returned.

- The returned Message objects are removed from robot's priority queue.
- Only Message objects, which have receiver name equal to robot name, are returned.

Fail criteria:

- One or more returned Message objects, which has time step not equal to current time step, are returned.
- The returned Message objects are not removed from robot's priority queue.
- One or more Message objects, which have receiver name equal to robot name, are returned.

7.1.6 Start up all links

Test item: CommunicationsSystem class

Pass criteria:

- IsLinkEnabled attribute is set to true.

Fail criteria:

- IsLinkEnabled attribute is set to something else.

7.1.7 Shutdown all links

Test item: CommunicationsSystem class

Pass criteria:

- IsLinkEnabled attribute is set to false.

Fail criteria:

- IsLinkEnabled attribute is set to something else.

7.1.8 Set system Range

Test item: CommunicationsSystem class

Input: System range

Pass criteria:

- Range attribute is set to system range.

Fail criteria:

- Range attribute is not set or set to something else.

7.1.9 Set system delay

Test item: CommunicationsSystem class

Input: System delay

Pass criteria:

- Delay attribute is set to system delay.

Fail criteria:

- Delay attribute is not set or set to something else.

7.1.10 Set system delivery probability

Test item: CommunicationsSystem class

Input: System delivery probability

Pass criteria:

- DeliveryProb attribute is set to system delivery probability.

Fail criteria:

- DeliveryProb attribute is not set or set to something else.

7.1.11 Start up robot's send link

Test item: RobotCommRecord class
Pass criteria:

- isSendLinkEnabled attribute is set to true.

Fail criteria:

- isSendLinkEnabled attribute is set to something else.

7.1.12 Shutdown robot's send link

Test item: RobotCommRecord class
Pass criteria:

- isSendLinkEnabled attribute is set to false.

Fail criteria:

- isSendLinkEnabled attribute is set to something else.

7.1.13 Start up robot's receive link

Test item: RobotCommRecord class
Pass criteria:

- isReceiveLinkEnabled attribute is set to true.

Fail criteria:

- isSendLinkEnabled attribute is set to something else.

7.1.14 Shutdown robot's receive link

Test item: RobotCommRecord class
Pass criteria:

- isReceiveLinkEnabled attribute is set to false.

Fail criteria:

- isReceiveLinkEnabled attribute is set to something else.

7.1.15 Set robot range

Test item: RobotCommRecord class
Input: robot range
Pass criteria:

- Range attribute is set to robot range.

Fail criteria:

- Range attribute is not set or set to something else.

7.1.16 Set robot delay

Test item: RobotCommRecord class
Input: robot1 name, robot2 name, robot delay
Pass criteria:

- Robot1 has a RobotParameter record that has robot2 as receiver name and robot delay as delay time.

- Robot2 has a RobotParameter record that has robot1 as receiver name and robot delay as delay time.

Fail criteria:

- There is no record of robot1's robotparameter that has robot2 as receiver name and robot delay as delay time.
- There is no record of robot2's robotparameter that has robot1 as receiver name and robot delay as delay time.

7.1.17 Set robot delivery probability

Test item: RobotCommRecord class
Input: robot1 name, robot2 name, robot delivery probability
Pass criteria:

- Robot1 has a RobotParameter record that has robot2 as receiver name and robot delivery probability as deliveryProb.
- Robot2 has a RobotParameter record that has robot1 as receiver name and robot delivery probability as deliveryProb.

Fail criteria:

- There is no record of robot1's robotparameter that has robot2 as receiver name and robot delivery probability as deliveryProb.
- There is no record of robot2's robotparameter that has robot1 as receiver name and robot delivery probability as deliveryProb.

7.1.18 Enable broadcast capability

Test item: RobotCommRecord class
Input: robot name
Pass criteria:

- The RobotCommRecord with this robot name has isBroadcastEnabled attribute set to true.

Fail criteria:

- The RobotCommRecord with this robot name has isBroadcastEnabled attribute set to something else.

7.1.19 Enable point-to-point capability

Test item: RobotCommRecord class
Input: robot name
Pass criteria:

- The RobotCommRecord with this robot name has isP2PEnabled attribute set to true.

Fail criteria:

- The RobotCommRecord with this robot name has isP2PEnabled attribute set to something else.

7.1.20 Disable broadcast capability

Test item: RobotCommRecord class
Input: robot name

Pass criteria:

- The RobotCommRecord with this robot name has isBroadcastEnabled attribute set to false.

Fail criteria:

- The RobotCommRecord with this robot name has isBroadcastEnabled attribute set to something else.

7.1.21 Disable point-to-point capability

Test item: RobotCommRecord class
Input: robot name
Pass criteria:

- The RobotCommRecord with this robot name has isP2PEnabled attribute set to false.

Fail criteria:

- The RobotCommRecord with this robot name has isP2PEnabled attribute set to something else.

7.1.22 Start up send link

Test item: RobotCommRecord class
Input: robot name
Pass criteria:

- The RobotCommRecord with this robot name has isSendLinkEnabled attribute set to true.

Fail criteria:

- The RobotCommRecord with this robot name has isSendLinkEnabled attribute set to something else.

7.2 Integration testing

7.2.1 Sending and receiving broadcast message

All robot records tested in these scenarios have broadcast capability and set parameters to default unless they are set by the scenario. The default parameters are incoming link and outgoing link enables, system link enables, delay time is zero, delivery probability is 100% and no maximum range is set for the entire system and for each robot.

7.2.1.1 Scenario 1 -- Default scenario

Pass criteria:

- All robots get the broadcast messages.

Fail criteria:

- One or more robots do not get the broadcast messages.

7.2.1.2 Scenario 2 -- Disable broadcast capability.

Pass criteria:

- A robot without broadcast capability does not get any broadcast messages from other robots

- Other robots do not get any broadcast messages from a robot without broadcast capability.

Fail criteria:

- A robot without broadcast capability gets broadcast messages from other robots
- One or more robot gets broadcast messages from a robot without broadcast capability.

7.2.2 Sending and receiving point-to-point message

All robots tested in these scenarios have point-to-point capability and set parameters to default unless they are set by the scenario. The default parameters are incoming link and outgoing link enables, system link enables, delay time is zero, delivery probability is 100% and no maximum range is set for both system and for each robot.

7.2.2.1 Scenario 3 -- Default scenario

Pass criteria:

- A robot with specified address gets a point-to-point message.

Fail criteria:

- Other robots that are not the specified address get a point-to-point message.

7.2.2.2 Scenario 4 -- Disable point-to-point capability.

Pass criteria:

- A robot without point-to-point capability does not get any point-to-point messages from other robots
- Other robots do not get point-to-point messages from a robot without point-to-point capability.

Fail criteria:

- A robot without point-to-point capability gets point-to-point messages from other robots
- One or more robot gets point-to-point messages from a robot without point-to-point capability.

7.2.3 Both broadcast and point-to-point message

The following scenarios apply for sending both broadcast and point-to-point messages. All robot records tested in these scenarios have broadcast capability, point-to-point capability and set parameters to default unless they are set by the scenario. The default parameters are incoming link and outgoing link enables, system link enables, delay time is zero, delivery probability is 100% and no maximum range is set for both system and for each robot.

7.2.3.1 Scenario 5 -- Disable system link

Pass criteria:

- All robots do not get any messages from other robots.

Fail criteria:

- One or more robots get messages from other robots.

7.2.3.2 Scenario 6 -- Disable outgoing link of a robot.

Pass criteria:

- No robots get the message sending from a robot with disabled outgoing link.

Fail criteria:

- One or more robots get the message sending from a robot with disabled outgoing link.

7.2.3.3 Scenario 7 -- Disable incoming link of a robot.

Pass criteria:

- A robot with disabled incoming link does not get any messages from other robots.

Fail criteria:

- A robot with disabled incoming link gets messages from other robots.

7.2.3.4 Scenario 8 – Set system delay

Pass criteria:

- Receiver gets message from sender after the time of sender sent message plus system delay. (Received time = sent time + system delay)

Fail criteria:

- Receiver gets message from sender in different time other than sent time plus system delay. (Received time <> sent time + system delay)

7.2.3.5 Scenario 9 – Set robot delay

Pass criteria:

- Receiver gets message from sender after sender sent message plus robot delay time. (Received time = sent time + robot delay)

Fail criteria:

- Receiver gets message from sender in time other than sent time plus robot delay. (Received time <> sent time + robot delay)

7.2.3.6 Scenario 10 – Set system delay and robot delay

Pass criteria:

- Receiver gets message from sender after sender sent message plus system delay plus robot delay time. (Received time = sent time + system delay + robot delay)

Fail criteria:

- Receiver gets message from sender in time other than sent time plus system delay plus robot delay. (Received time <> sent time + system delay + robot delay)

7.2.3.7 Scenario 11 – Set system range

Pass criteria:

- Only robots within *this range* of sender get the message.

Fail criteria:

- One or more robots outside *this range* of sender get message.

* *this range* refers to system range.

7.2.3.8 Scenario 12 – Set robot range

Pass criteria:

- Only robots within *this range* of sender get the message.

Fail criteria:

- One or more robots beyond *this range* of sender get message.

* *this range* refers to robot range.

7.2.3.9 Scenario 13 – Set system range and robot range

Pass criteria:

- Only robots within *this range* of sender get the message.

Fail criteria:

- One or more robots outside *this range* of sender get message.

* *this range* refers to system range + robot range.

7.2.3.10 Scenario 14 – Set system delivery probability

Pass criteria:

- Only randomly selected message within the probability range will delivery to the receivers.
- Randomly selected message beyond the probability range will not delivery to the receivers.

Fail criteria:

- Randomly selected message within the probability does not delivery to the receivers.
- Randomly selected message beyond the probability range delivery to the receivers.

7.2.3.11 Scenario 15 – Set robot delivery probability

Pass criteria:

- Only randomly selected message within the probability range will delivery to the receivers.
- Randomly selected message beyond the probability range will not delivery to the receivers.

Fail criteria:

- Randomly selected message within the probability does not delivery to the receivers.
- Randomly selected message beyond the probability range delivery to the receivers.

7.2.3.12 Scenario 16 – Set system delivery probability and robot delivery probability

*The probability range will be the average of system and robot delivery probability**.**

Pass criteria:

- Only randomly selected message within the probability range will delivery to the receivers.

- Randomly selected message beyond the probability range will not delivery to the receivers.

Fail criteria:

- Randomly selected message within the probability does not delivery to the receivers.
- Randomly selected message beyond the probability range delivery to the receivers.

7.3 System testing

The System testing will be done by having Environment use the communications interface to send and receive message. Robots create their own messages and send or receive message through the communication model via the environment. The environment control panel can also set the parameters via the environment. The test will be conducted using the same test cases as integration testing. The system testing will ensure the compatibility with other modules in the system.

8. Schedule

All testing will be performed during July14 – July 28, 2004. Unit testing will be performed after finish coding each module. Integration testing will begin after all units testing are done.

CHAPTER 8

FORMAL TECHNICAL INSPECTION

---

1. Introduction

The purpose of this document is to provide formal technical inspection checklist of the design architecture for "Communication Model for Cooperative Robotic Simulator". Two independent MSE students will inspect the design architecture and provide a report on the result of their inspection. The formal inspection is the review process to ensure the quality of the software design, which will be useful toward the development process.

2. Items to be inspected

The architecture design of "Communication Model for Cooperative Robotic Simulator" will be inspected including use-case diagram, class diagram and sequence diagram. Some reference documents will be provided to give some background of the system. The reference documents are Software Requirement Specification version 1.0 and Project Overview version 1.0.

The following are the architecture design will be inspected.

- Use case diagram
- Class diagram
- Sequence diagram

3. Formal technical inspector

- Kevin Sung
- Estaban Guillen

4. Formal technical inspection checklist

Inspection list	Pass/Fail/Partial	Comment
1. The symbols using in use case diagram conform to UML diagram.		
2. The symbols using in class case diagram conform to UML diagram.		
3. The symbols using in sequence diagram conform to UML diagram.		
4. Use case diagram and descriptions are clear and well organized.		
5. Class diagram and descriptions are clear and well organized.		
6. Each message passing in sequence diagram is the method in class diagram.		

7. Each message passing in sequence diagram must be defined as public method.		
8. Class names are well defined and indicate their meaning		
9. The architecture design covers the entire requirement defined in Software Requirement Specification.		

Table 2. Formal Technical Inspection Checklist

CHAPTER 9

COMPONENT DESIGN

1. Introduction

This section will provide description of class diagram for the Communication Model for Cooperative Robotics Simulator.

2. Class Diagram

The communication Model contains five classes, CommunicationsSystem, RobotCommRecord, PriorityQueue, RobotParameters and Message. More details will be described in the next section.

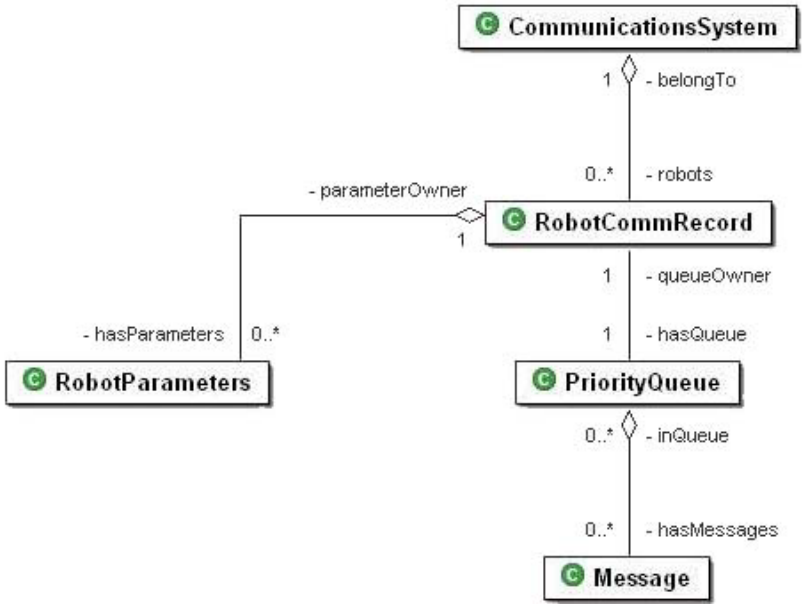

Figure 17. Communication Model Class Diagram

3. Class Descriptions

The following section will provide class diagram member in detail. It will provide only public attributes and functions. A detail description of private attributes and functions is provided in JavaDoc documentation.

3.1 CommunicationsSystem Class

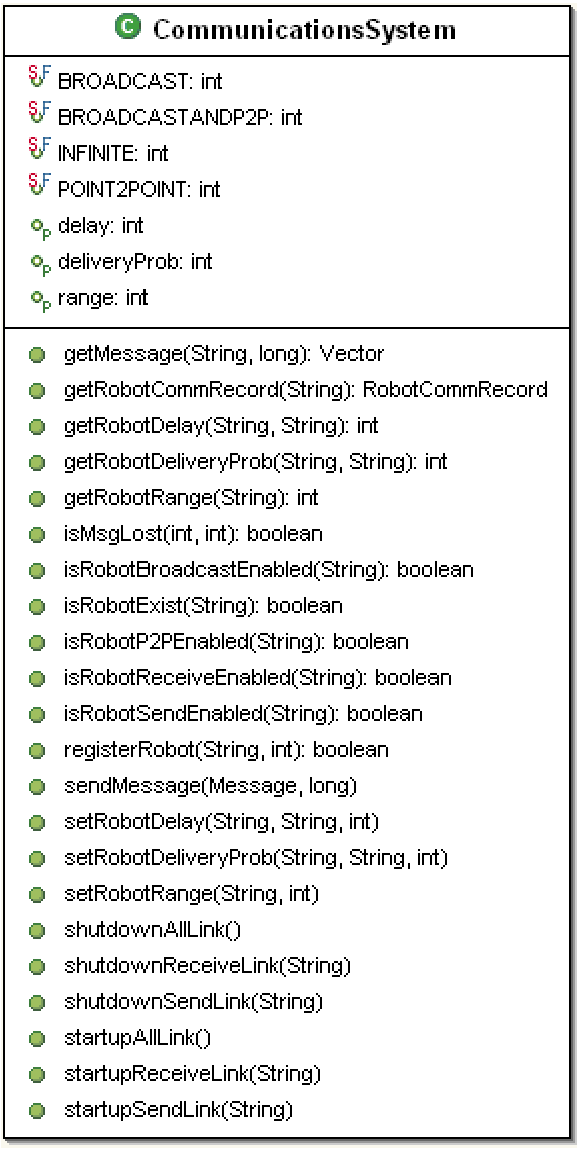

Figure 18. CommunicationsSystem Class

3.1.1 Detailed Description

The CommunicationsSystem class is served as a main interface, which interact with the environment. It provides function for robot and environment control panel. It is also responsible for sending message, delivering message to the right robot including the process of verifying participants.

3.1.2 Constructor

- **CommunicationsSystem(Environment env)**
  This is default CommunicationsSystem constructor with Environment as argument. This Environment is used for retrieving distance between two robots.

3.1.3 Public Member Attributes

The following attributes are constant value using in the communication system

- static final int **BROADCAST**
  The constant value indicates broadcast ability. It is used to define that a robot has broadcast ability. It is one of communication types that uses in the registerRobot method. For instances,
  registerRobot("robotA",CommunicationsSystem.BROADCAST)

- static final int **BROADCASTANDP2P**
  The constant value indicates broadcast and point-to-point ability. It is used to define that a robot has both broadcast and point-to-point ability. It is one of communication types that uses in the registerRobot method. For instances,
  registerRobot("robotA",CommunicationsSystem.BROADCASTANDP2P)

- static final int **INFINITE**
  The constant value indicates infinite range limit. It is used to define that there is no range limit for a robot.

- static final int **POINT2POINT**
  The constant value indicates point-to-point ability. It is used to define that a robot has point-to-point ability. It is one of communication types that uses in the registerRobot method. For instances,
  registerRobot("robotA",CommunicationsSystem.POINT2POINT)

3.1.4 Public Member Functions

- **Vector getMessage(String name,long timeStep)**
  Retrieve message from the communication system. The parameters are robot name and time step. The robot name is the name of the owner, who would like to retrieve messages at the defined time step.
  Parameters:
  name - - the owner of the message
  timeStep - - the time step of messages which robot want to get messages.
  Returns:
  A vector of messages.

- **RobotCommRecord getRobotCommRecord(String name)**
  Get RobotCommRecord with identified name
  Parameters:
  name - - robot name
  Returns:
  Identified RobotCommRecord

- **int getRobotDelay(String robot1,String robot2)**
  Get delay time between a pair of robot.
  Parameters:
  robot1 - - first robot name
  robot2 - - second robot name
  Returns:
  Delay time between robot1 and robot2. The unit of delay time is in time step, which is about 500 milliseconds per time step

- **int getRobotDeliveryProb(String robot1,String robot2)**
  Get delivery probability of a pair of robot
  Parameters:
  robot1 - - first robot name
  robot2 - - second robot name
  Returns:
  Delivery probability between robot1 and robot2. (Delivery probability has value between 0-100)

- **int getRobotRange(String name)**
  Get maximum range limit of a robot.
  Parameters:
  name - - robot name
  Returns:
  maximum range limit (unit is meter)

- **boolean isMsgLost(int random, int prob)**
  Determine if message is lost or not by comparing random value of a message with total delivery probability.
  Parameters**:**
  random - - message random value
  prob - - total delivery probability (value between 0-100)
  Returns:
  true if random > prob , a message will be lost if random value is beyond the probability.

- **boolean isRobotBroadcastEnabled(String name)**
  Check if broadcast abililty is set to true
  Parameters:
  name - - robot name

Returns:

true - if broadcast is enabled, false otherwise

- **boolean isRobotExist(String name)**
  Check if this robot exists in the system.
  Parameters:
  name - - name of a robot
  Returns:
  true if this robot exists, false otherwise

- **boolean isRobotP2PEnabled(String name)**
  Check if point to point ability is set to true
  Parameters:
  name - - robot name
  Returns:
  true - if point to point is enabled, false otherwise

- **boolean isRobotReceiveEnabled(String name)**
  Check if robot's incoming link is set to true
  Parameters:
  name - - robot name
  Returns:
  true - if incoming link is enabled, false otherwise

- **boolean isRobotSendEnabled(String name)**
  Check if robot's outgoing link is set to true
  Parameters:
  name - - robot name
  Returns:
  true - if outgoing link is enabled, false otherwise

- **boolean isLinkEnabled()**
  Checking if system link is enabled or not
  Returns:
  Status of system link

- **boolean registerRobot(String name, int commType)**
  Register robot into the system before starting communication session.
  Parameters:
  name - - robot name
  commType - - communication types which registered robot can use to communicate. The possible values are BROADCAST, POINT2POINT and BROADCASTANDP2P.
  Returns:
  true if successfully registered, false otherwise

- **void sendMessage(Message msg, long timeStep)**
  Send message to other robots. There are two parameters, which are sending message and the current time step. Every message has sent time and received time. Sent time is the time step, which this message is sending out. Received time is the time step which receiver should get this message. Received time is defined based on system delay and robot delay.
  Parameters:
  msg - - message to others robot.
  timeStep - - current time step.

- **void setRobotDelay(String robot1, String robot2,int delay)**
  Set delay between a pair of robots.
  Parameters:
  robot1 - - the first robot's name.
  robot2 - - the second robot's name.
  delay - - delay time between these two robots. The unit of delay time is in time step, which is about 500 milliseconds per time step.

- **void setRobotDeliveryProb(String robot1, String robot2, int prob)**
  Set delivery probability between a pair of robot.
  Parameters:
  robot1 - - first robot's name
  robot2 - - second robot's name
  prob - - delivery probability (value between 0-100)

- **void setRobotRange(String name,int range)**
  Set maximum range limit, which is the maximum distance of receiver from which this robot receive message.
  Parameters:
  name - - robot name
  range - - maximum range limit (unit is meter)

- **void shutdownAllLink()**
  To shutdown system link

- **void shutdownReceiveLink(String name)**
  Shutdown robot's incoming link
  Parameters:
  name - - robot name

- **void shutdownSendLink(String name)**
  Shutdown robot's outgoing link
  Parameters:
  name - - robot name

- **void startupAllLink()**
  To start up system link

- **void startupReceiveLink(String name)**
  Startup robot's imcoming link
  Parameters:
  name - - robot name

- **void startupSendLink(String name)**
  Start up robot's outgoing link
  Parameters:
  name - - robot name

3.2 RobotCommRecord Class

Figure 19. RobotCommRecord Class

3.2.1 Detailed Description

RobotCommRecord class keeps communication parameters of each robot. Communication parameters include range, delay, link and delivery probability between each robot.

- Range is the maximum distance, which a robot can send messages to other robots.
- Delay is the delay time between each pair of robot to simulate traffic in the system.
- Link includes send link and receive link which both can be start up or shutdown.
- Delivery Probability is the probability of message that will be delivered to the destination. It will help generating message lost in the system.

This class manipulates the parameters and also determines if sender and receiver robot are qualified to pass and get of both point-to-point and broadcast message.

3.2.2 Constructor

- **RobotCommRecord(String name, int commType)**
  Default constructor of RobotCommRecord takes two parameters, robot name and communication type.
  Parameters:
  name - - Robot name
  commType - - Communication type (BROADCAST,POINT2POINT or BROADCASTANDP2P)

3.2.3 Public Member Attributes

There is no public member attribute in RobotCommRecord class

3.2.4 Public Member Functions

- **void addMsgToQueue(Message msg)**
  Method to add a message to a priority queue
  Parameters:
  msg - - message adding to a queue

- **boolean addRobotParameters(RobotParameters robot)**
  Adding robot parameter, which related to this robotCommRecord. It will check if there exists. If not, the record will be added into the vector.
  Parameters:
  robot - - robot parameter adding to the vector
  Returns:
  true if sucessfully added, false otherwise

- **void disableBroadcast()**
  Disable Broadcast capability

- **void disableP2P()**
  Disable Point-to-point capability

- **void enableBroadcast()**
  Enable Broadcast capability

- **void enableP2P()**
  Enable Point-to-point capability

- **int getDelay(String name)**
  Get delay time
  Parameters:
  name – robot name
  Returns:
  Delay time. The unit of delay time is in time step, which is about 500 milliseconds per time step

- **int getDeliveryProb(String name)**
  Get delivery probability
  Parameters:
  name – robot name
  Returns:
  Delivery probability value (between 0-100)

- **Vector getMessage(long time)**
  Get a Vector of message
  Parameters:
  time - - received time of messages which will be taking out from the queue.
  Returns:
  a vector of message

- **boolean isReceiveBroadcastEnabled()**
  Check if robot can get broadcast message
  Returns:
  true if robot can get broadcast message, false otherwise

- **boolean isReceiveP2PEnabled()**
  Check if robot can get point-to-point message
  Returns:
  true if robot can get point-to-point message, false otherwise

- **boolean isSendBroadcastEnabled()**
  Check if robot can send broadcast message
  Returns:
  true if robot can send broadcast message, false otherwise

- **boolean isSendP2PEnabled()**
  Check if robot can send Point-2-point message
  Returns:
  true if robot can send point-to-point message

- **void setDelay(String name, int delay)**
  Set delay time of a robot
  Parameters:
  name - - robot name
  delay - - delay time. The unit of delay time is in time step, which is about 500 milliseconds per time step

- **void setDeliveryProb(String name**, **int prob)**
  Set delivery probability of a robot
  Parameters:
  name - - robot name
  prob - - delivery probability (value between 0-100)

- **void shutdownReceiveLink()**
  Shutdown incoming link

- **void shutdownSendLink()**
  Shutdown outgoing link

- **void startupReceiveLink**()
  Start up incoming link

- **void startupSendLink()**
  Start up outgoing link

3.3 RobotParameters class

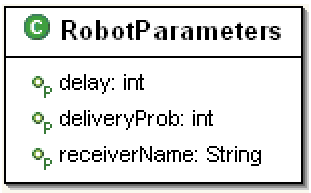

Figure 20. RobotParameters Class

3.3.1 Detailed Description

This RobotParameters class represents communication parameter. Each robot owns multiple records of RobotParameters. Each record of RobotParameters contains

- Receiver Name - the communicating robot

- Delay time - the communication delay time between the owner and this receiver.
- Delivery Probability - the probability which messages will be delivered to the receiver

3.3.2 Constructor

- **RobotParameters(String name)**
  RobotParameters Constructor with the name of the receiver
  Parameters:
  name - - Receiver name

- **RobotParameters(String name, int delay, int prob)**
  RobotParameters Constructor with the name of the receiver, delay time and delivery probability corresponds to this receiver.
  Parameters:
  name - - Receiver name
  delay - - Delay time. The unit of delay time is in time step, which is about 500 milliseconds per time step
  prob - - Delivery Probability (value between 0-100)

3.3.3 Public Member Attributes

There is no public member attribute in this class.

3.3.4 Public Member Function

The public member functions in this class are only Get and Set method.

3.4 PriorityQueue Class

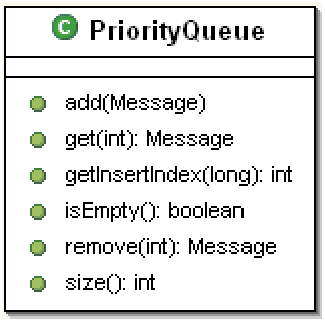

Figure 21. PriorityQueue Class

3.4.1 Detailed Description

This PriorityQueue class is used to keep Message in Queue in ascending order of receivedTime. Robots have their own priority queue. It keeps incoming message and wait for the owner call getMessage operation to retrieve message from the queue.

3.4.2 Constructor

- **PriorityQueue**()
  PriorityQueue Constructor

3.4.3 Public Member Attributes

There is no public member attribute in this class.

3.4.4 Public Member Function

- **void add(Message msg)**
  Adding message to a queue in order of receivedTime
  Parameters:
  msg - - message adding to a queue

- **Message get(int index)**
  Get specific message from the queue without deleting
  Parameters:
  index - - the position of message
  Returns:
  a message

- **int getInsertIndex(long time)**
  Helper method to get the right adding position.
  Parameters:
  time - - receive time of new message adding to a queue

- **boolean isEmpty()**
  Check if queue is empty
  Returns:
  true if queue is empty, false otherwise

- **Message remove(int index)**
  Removing specific message from the queue
  Parameters:
  index - - the position of removing message
  Returns:
  a message

- **int size()**
  Get size of priority queue
  Returns:
  size of queue

3.5 Message Class

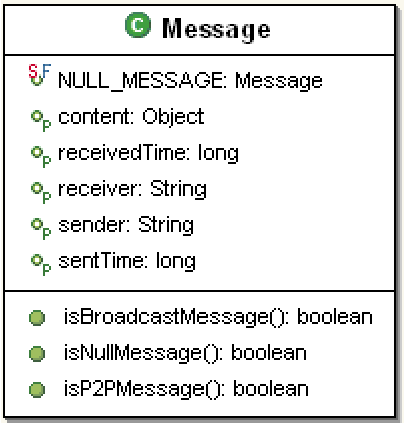

Figure 22. Message Class

3.5.1 Detailed Description

This class represents format of message, which passes back and forth between robots in the system. The content in a message is composed of

- Sender
- Receiver
- The time that message was sent.
- The expected time that message will be delivered to the receiver.
- The real content of a message

The sent time will be automatically filled in with the current time step. By the time message is created the received time will be blank until the message is passed to the process of sending message. The received time will be filled out based on system delay and robot delay.

3.5.2 Constructor

- **Message**()
  This is a constructor of Message class without argument.

- **Message**(String sender, String receiver, Object content)
  This is a constructor of Message class.
  Parameters:
  sender - - message sender
  receiver - - message receiver
  content - - actual content of message

3.5.3 Public Member Attributes

- static final Message **NULL_MESSAGE**
  Null message is used to verify that this is the last message. It is used by environment to check if it is the last message from and to a robot.

3.5.4 Public Member Function

- **boolean isBroadcastMessage()**
  Check if this message is broadcast message.
  Returns:
  true if receiver is broadcast, false otherwise

- **boolean isNullMessage()**
  Check is this message is null message.
  Returns:
  true if it is NULL_MESSAGE, false otherwise

- **boolean isP2PMessage()**
  Check if this message is point-to-point message.
  Returns:
  true if receiver is not broadcast, false otherwise

CHAPTER 10

ASSESSMENT EVALUATION

---

1. Introduction

This document will provide the test result of the Communication Model for Cooperative Robotics Simulator based on Test Plan 1.0

2. Testing Result Summary

Feature Identifier	Test Case	Result
T-001	Register robot.	Passed
T-002	Sending broadcast message	Passed
T-003	Sending point-to-point message	Passed
T-004	Receiving broadcast message	Passed
T-005	Receiving point-to-point message	Passed
T-006	Start up all links	Passed
T-007	Shutdown all links	Passed
T-008	Set system range	Passed
T-009	Set system delay	Passed
T-010	Set system delivery probability	Passed
T-011	Start up robot's send link	Passed
T-012	Shutdown robot's send link	Passed
T-013	Start up robot's receive link	Passed
T-014	Shutdown robot's receive link	Passed
T-015	Enable broadcast capability	Passed
T-016	Disable broadcast capability	Passed
T-017	Enable point-to-point capability	Passed
T-018	Disable point-to-point capability	Passed
T-019	Set robot range	Passed
T-020	Set robot delay	Passed
T-021	Set robot deliver probability	Passed

Table 3. Testing Result Summary

3. Testing Result Details

3.1 T-001 Register robot

This test case was successfully passed. The new RobotCommRecord was created and was assigned to have communication capability as stated in communication type.

3.2 T-002 Sending broadcast message

This test case was successfully passed. Only the robots with broadcast capability can send out messages.

3.3 T-003 Sending point-to-point message

This test case was passed. Only the robots with point-to-point capability can send point-to-point message

3.4 T-004 Receiving broadcast message

This test was failed but finally passed. When sending broadcast message, a message needs to be distributed to the other robots to save in their queue. Due to the distribution process, received time attribute will be modified to be an actual received time. Each robot's received time will be varied. It depends on robot's parameters, which are system delay and robot delay. Since Message object implements serialization, if we put a message in a robot's queue without recreation, modifying an attribute will affect a message, which is already inserted into a queue. As a result, the problem has been solved by replicating the message before adding to a queue.

In addition, receivers get messages based on their parameters, which are delay, range and delivery probability. The results of each scenario are as followed.

- **System delay was set**
  Receivers got message from sender after the time of sender sent message plus system delay. (Received time = sent time + system delay)

- **Robot delay was set**
  Receivers got message from sender after sender sent message plus robot delay time. (Received time = sent time + robot delay)

- **System delay and Robot delay were set**
  Receivers got message from sender after sender sent message plus system delay plus robot delay time. (Received time = sent time + system delay + robot delay)

- **System range was set**
  Only robots within system range limit of sender got the message.

- **Robot range was set**
  Only robots within robot range of sender got the message.

- **System range and robot range were set**
  Only robots within *this range* of sender got the message. In this case, *this range* is system range plus robot range.

- **System delivery Probability was set**
  Only randomly selected message within the system delivery probability range was delivered to the receivers.

- **Robot delivery Probability was set**
  Only randomly selected message within the robot delivery probability range was delivered to the receivers.

- **System delivery Probability and robot delivery probability were set**

Only randomly selected message within the delivery probability range was delivered to the receivers. In this case, the delivery probability is the average of system delivery probability and robot delivery probability.

3.5 T-005 Receiving point-to-point message

This test was passed. The robot with point-to-point capability and enabled incoming link received all point-to-point messages. Robots received messages correctly based on their parameters, which are delay, range, delivery probability and link status. The results of each scenario are as followed.

- **System delay was set**
  Receivers got message from sender after the time of sender sent message plus system delay. (Received time = sent time + system delay)

- **Robot delay was set**
  Receivers got message from sender after sender sent message plus robot delay time. (Received time = sent time + robot delay)

- **System delay and Robot delay were set**
  Receivers got message from sender after sender sent message plus system delay plus robot delay time. (Received time = sent time + system delay + robot delay)

- **System range was set**
  Only robots within system range limit of sender got the message.

- **Robot range was set**
  Only robots within robot range of sender got the message.

- **System range and robot range were set**
  Only robots within *this range* of sender got the message. In this case, *this range* is system range plus robot range.

- **System delivery Probability was set**
  Only randomly selected message within the system delivery probability range was delivered to the receivers.

- **Robot delivery Probability was set**
  Only randomly selected message within the robot delivery probability range was delivered to the receivers.

- **System delivery Probability and robot delivery probability were set**

Only randomly selected message within the delivery probability range was delivered to the receivers. In this case, the delivery probability is the average of system delivery probability and robot delivery probability.

3.6 T-006 Start up all links

This test was passed. When StartupAllLink was called, the system link variable was set to true.

3.7 T-007 Shutdown all links

This test was passed. When ShutdownAllLink was called the system link variable was set to false.

3.8 T-008 Set system range

This test was passed. The system range variable was correctly set.

3.9 T-009 Set system delay

This test was passed. The system delay variable was correctly set.

3.11 T-010 Set system delivery probability

This test was passed. The system delivery probability variable was correctly set.

3.12 T-011 Start up robot's send link

This test was passed. The outgoing link of a particular robot was set to true.

3.13 T-012 Shutdown robot's send link

This test was passed. The outgoing link of a specified robot was set to false.

3.14 T-013 Start up robot's receive link

This test was passed. The incoming link of a particular robot was set to true.

3.15 T-014 Shutdown robot's receive link

This test was passed. The incoming link of a particular robot was set to false;

3.16 T-015 Enable broadcast capability

This test was passed. The variable for broadcast status was set to true.

3.17 T-016 Disable broadcast capability

This test was passed. The variable for broadcast status was set to false.

3.18 T-017 Enable point-to-point capability

This test was passed. The variable for point-to-point status was set to true.

3.19 T-018 Disable point-to-point capability

This test was passed. The variable for point-to-point status was set to false.

3.20 T-019 Set robot range

This test was passed. The range of a particular robot was correctly set.

3.21 T-020 Set robot delay

This test was passed. The delay of a particular robot was correctly set.

3.21 T-021 Set robot deliver probability

This test was passed. The delivery probability of a particular robot was correctly set.

3.22 Integration Testing

The integration testing was passed. I have done this successful testing by integrating it with the Environment. This test was done by testing in two main functions: robot functions and control panel functions. The robot functions include message passing between the robots and the communication model. The control panel functions consist of get and set system and robot parameters. Both tests were passed, although there were some modules in the Environment Control Panel that invoked incorrect methods. There were also some problems with the graphical user interface that displayed incorrectly and accepted wrong value.

CHAPTER 11

USER MANUAL

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.

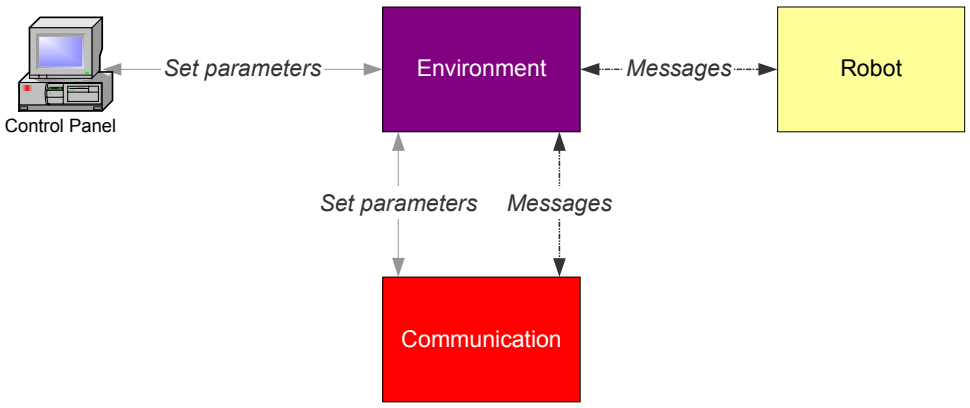

Figure 23. Interactions between Communication, Environment, Control Panel and Robot Diagram

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

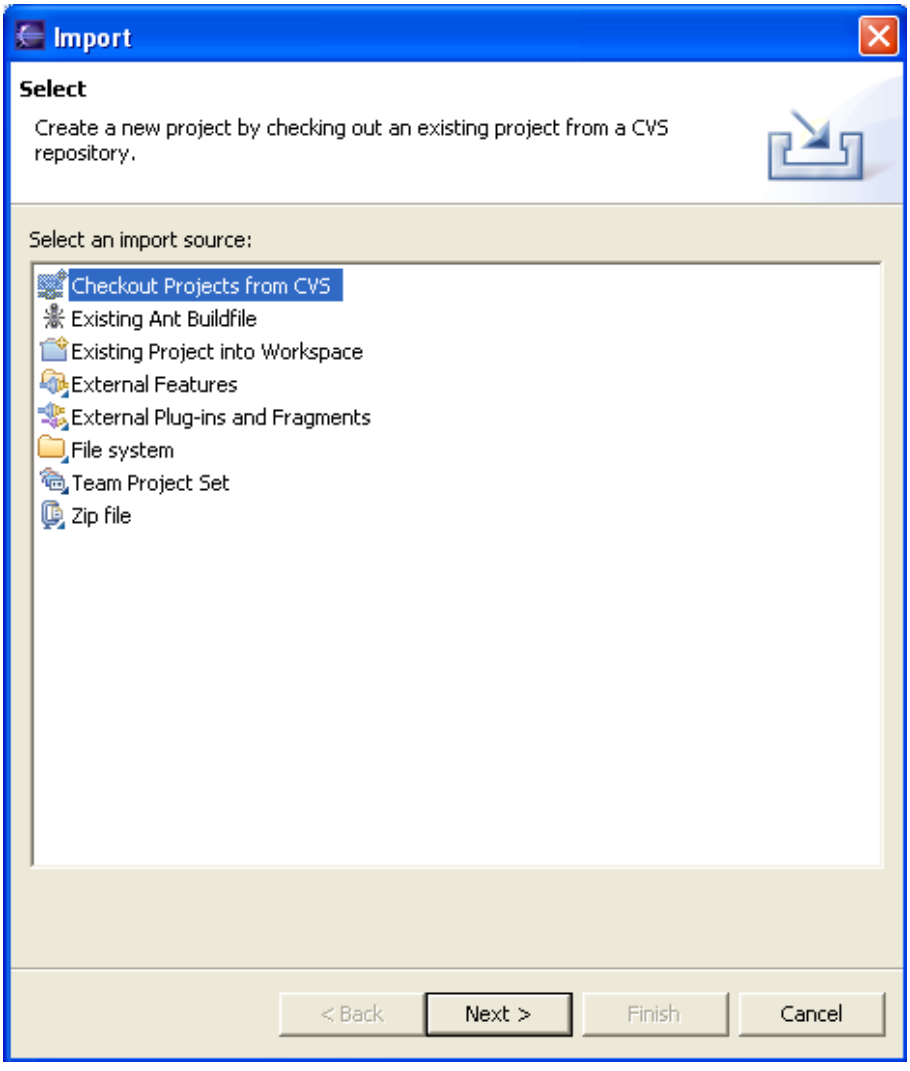

Figure 24. Using Eclipse to Check out from CVS – 1

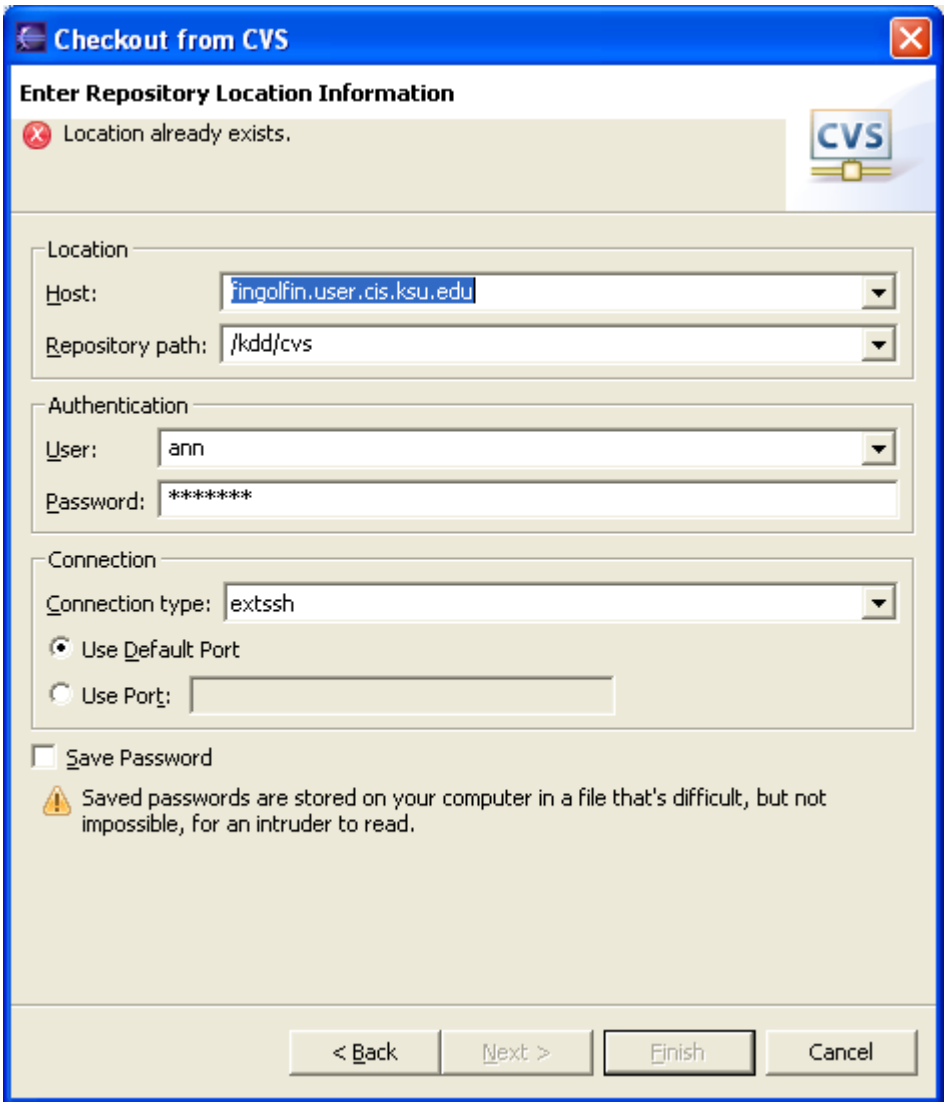

Figure 25. Using Eclipse to Check out from CVS – 2

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

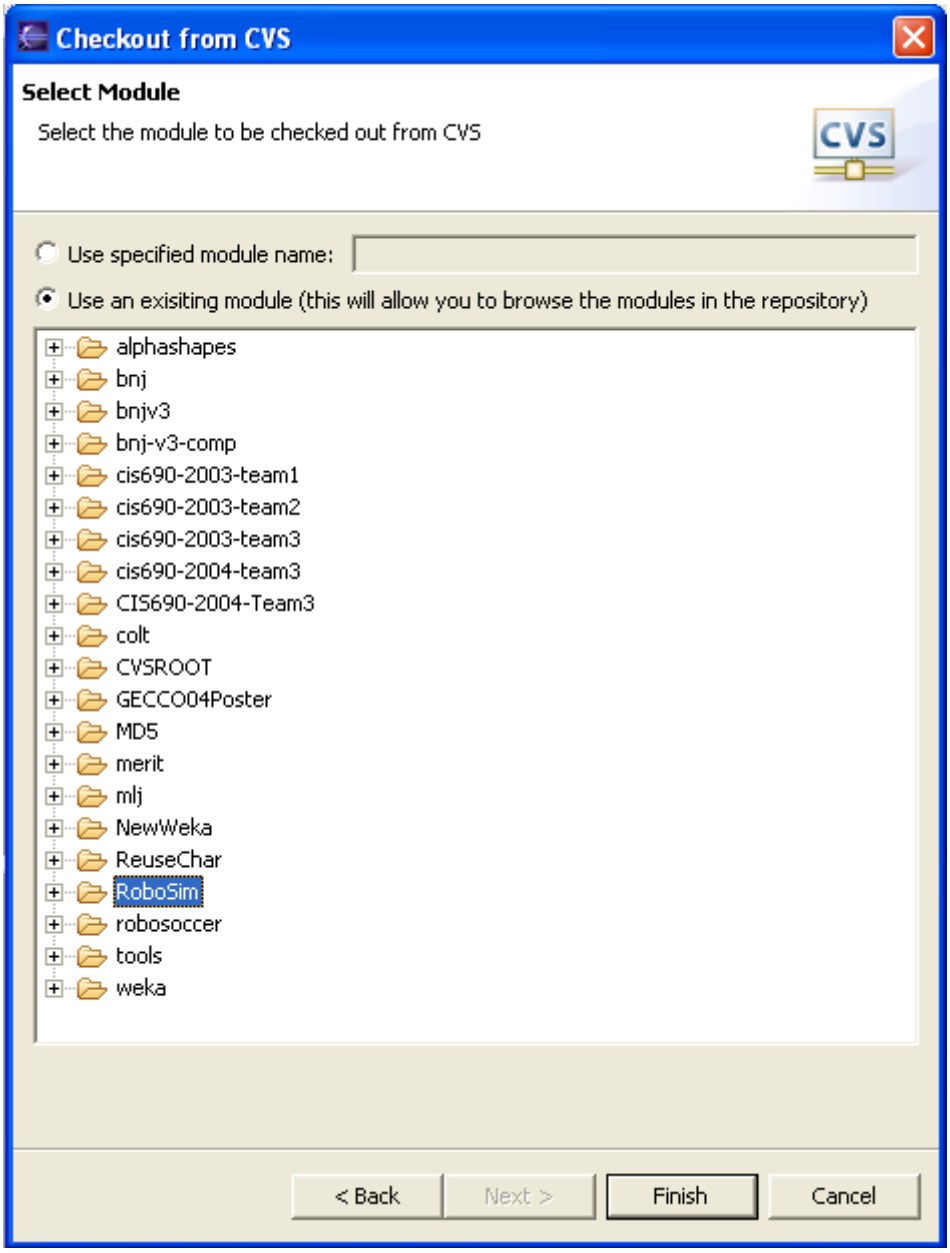

Figure 26. Using Eclipse to Check out from CVS – 3

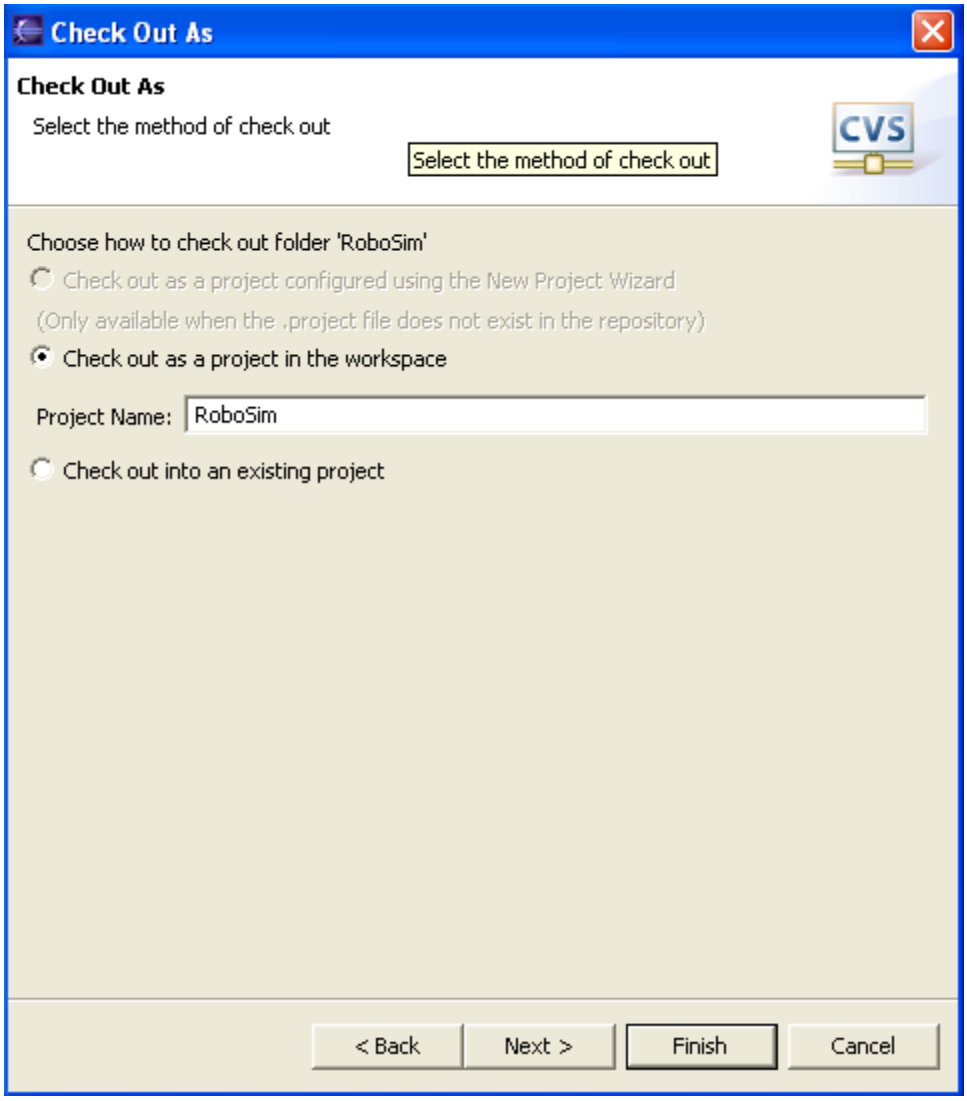

Figure 27. Using Eclipse to Check out from CVS – 4

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.

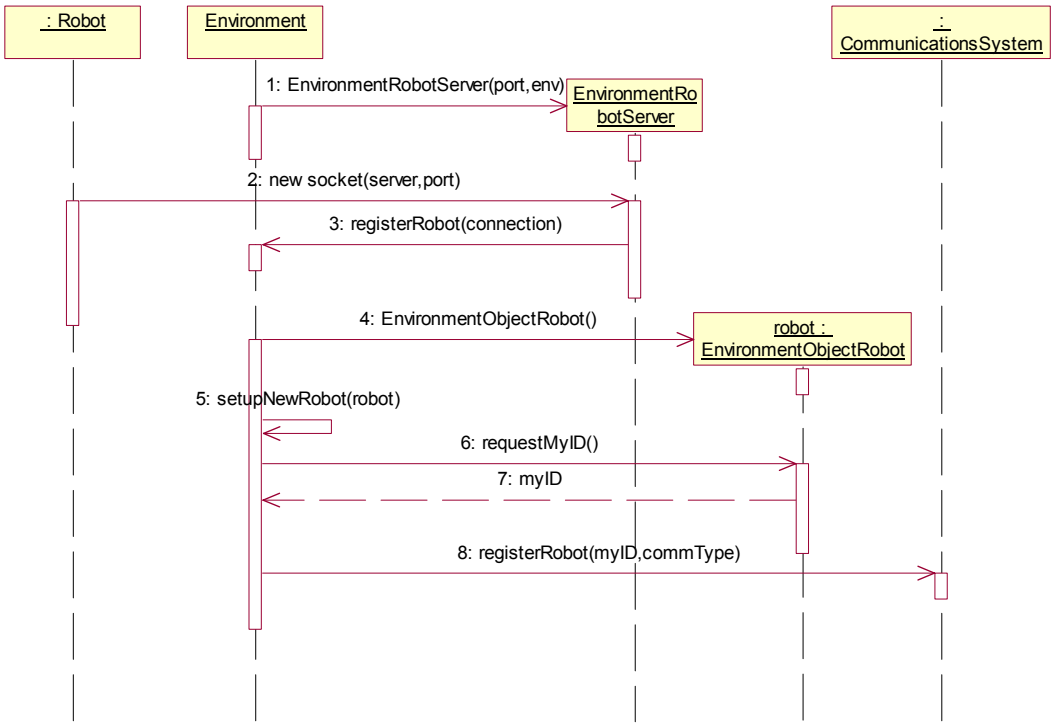

Figure 28. Interaction between Robot and Environment for registering robot

- 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.

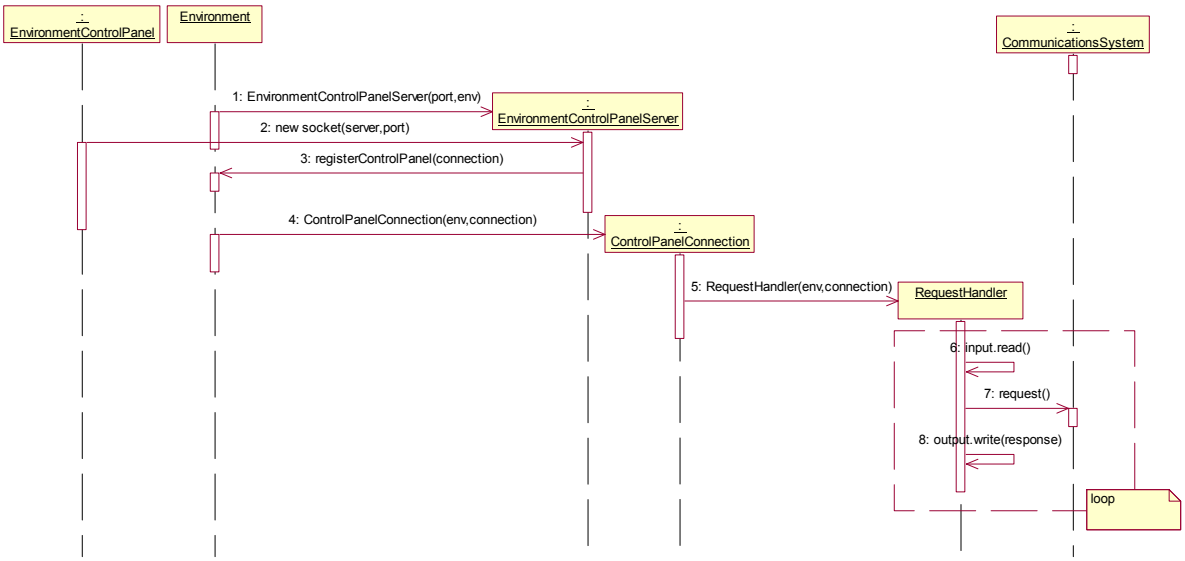

Figure 29. Interaction between Control Panel and Environment for setting parameters.

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 4. 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 5. 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.

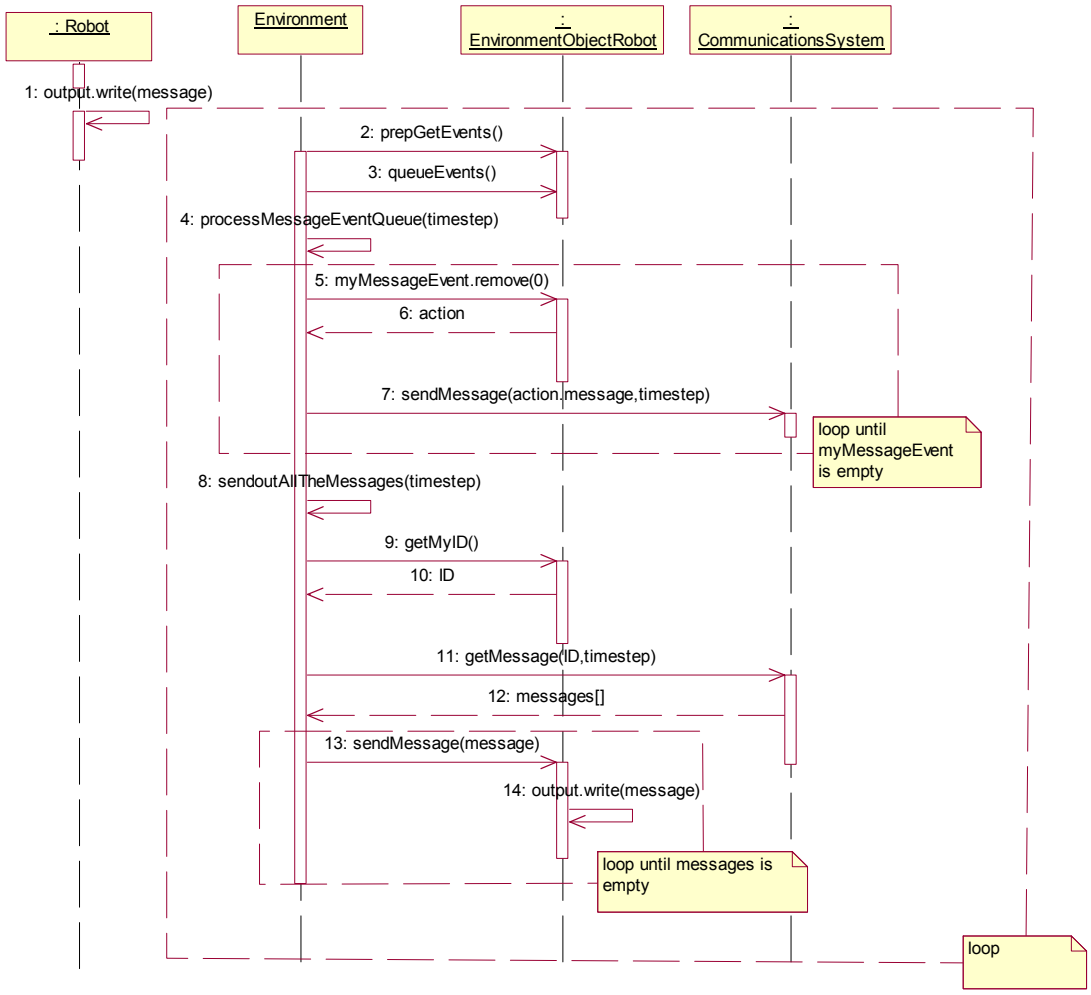

Figure 30. Interaction between Robot and Environment for message passing

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.

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);
```

CHAPTER 11

PROJECT EVALUATION

---

1. Introduction

The purpose of this project evaluation is to assess the effectiveness of all methodologies exercised throughout the development of this project and the accuracy of time estimation. In addition, the product will be reviewed and evaluated whether it has accomplished the goal presented in the initial overview and for the quality of the product. Furthermore, the effort utilized in this project will be classified and diagramed. Finally, the future work regarding add-on features and performance improvement will be described.

2. Usefulness of methodologies

2.1 UML diagram

The Unified Modeling Language (UML) is a standard for object-oriented analysis and design. It consists of standard diagrams: class diagram, sequence diagram, collaboration diagram, etc. This standard is very practical and helps communicate with other people during the first stage of development and analysis. Moreover, with a comprehensive design and logical diagram, this will assist programmer to develop software and deliver quality software much more with ease.

2.2 Object Constraint Language

The Unified Modeling Language (UML) has become an industrial standard that has integrated different modeling notations into a single modeling language notation. It is beneficial for documenting object-oriented analysis and design.

The UML itself is imprecise and ambiguous, consequently people can interpret differently. The Object Constraint Language (OCL) has been created to describe additional integrity constraint in the model to help communicate among users, designers and programmers. There are several OCL tool available for download. In this project, I selected USE tool to model UML/OCL. USE tool is a graphical tool modeling UML class diagram with OCL. It is very constructive and can simulate different scenarios to validate the specification. Although OCL is easy to read and write, it would still require some experience in UML and OCL before starting the project.

Mostly, I spent a lot of time on this project creating and validating the OCL specifications, since some of constraints are complex and need a lot of effort to test and verify. Even though many test scripts were created to test against the model to prove that the specification is correct, the USE tool is a great tool to use for validating the specification.

3. Estimation

3.1 Number of lines of code

The estimation of number of lines of code for this project predicted in the first phase was about 1,325 SLOC based on Function Point Analysis (FPA). Due to the reason that I have less knowledge about this project, the result seems to be overestimated compare with the second phase estimation, which was only 912 SLOC. The project complexity estimated in the second phase was less than expected in the first phase, because I gained more extensive knowledge on this project, which could help me design and develop this project more precisely. As a result, some of unnecessary output and response were eliminated which reduced the estimation of number of lines of code down to 912 SLOC. Nevertheless, the actual number of lines of code is 874 SLOC, which is close to the estimation.

3.2 Time duration

The estimation of time used to complete this project during the first phase was 661 hours or 4.35 months based on COCOMO I model. Since there are 152 working hours per month as Boem stated, 4.35 months*152 hours/month is equal to 661 hours. Conversely, the time to complete this project was refined during the second phase based on the SLOC that was decreased to 912 SLOC. Therefore, the project duration changed from 4.35 months or 661 hours to 3.74 months or 568 hours. In addition, the bottom-up approach estimation was developed during the second phase to estimate the time duration of the rest of the project or the third phase using Work Breakdown Structure (WBS). The Work Breakdown Structure estimation was 37 days or 259 hours (more detail is available in the implementation plan section), which is close to the actual time, which is 229 hours. The expected and actual finish times are as followed.

Phase	Expected Finish Time	Actual Finish Time
Phase I	February 27, 2004	February 23, 2004
Phase II	April 26, 2004	May 5, 2004
Phase III	August 19, 2004	October 5,2004

Table 6. Expected and actual finish time for each phase

During the first phase, it was finished ahead of time because I was taking only MSE project. Hence, the rest of time was dedicated on this project. During the second phase, the time was delayed because I got an offer to work as a graduate assistant at KSU foundation. Consequently, I spent about 20 hours a week working at KSU foundation. As a result, less time spent on the project. During the last phase, the time was also delayed again due to the reason that I had to go back to my country (Thailand) for about 5 weeks. Importantly, I had difficulties on integrating with the other parts of the system. I had to modify many parts of the integrated module to work better with my part. Furthermore, I had to produce a demo GUI for my presentation, which is integrated with other parts and wrote a document to improve this module in order to run separately from the Environment module.

The following pie chart diagrams show the work breakdown and total time spent in each phase.

This diagram shows total time spent break down to each phase.

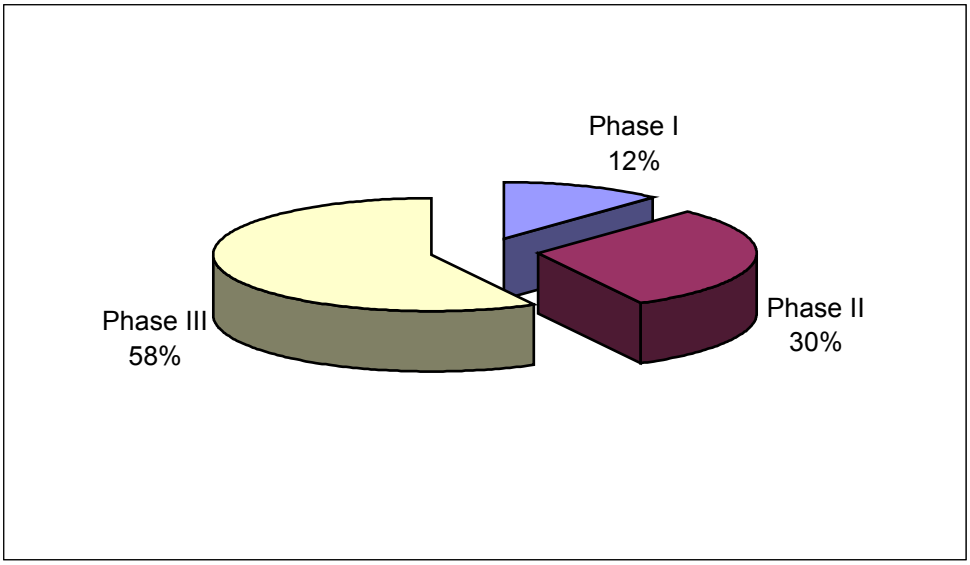

Figure 31. Phase Time Breakdowns

This chart diagram represents work breakdown time spent during the whole project.

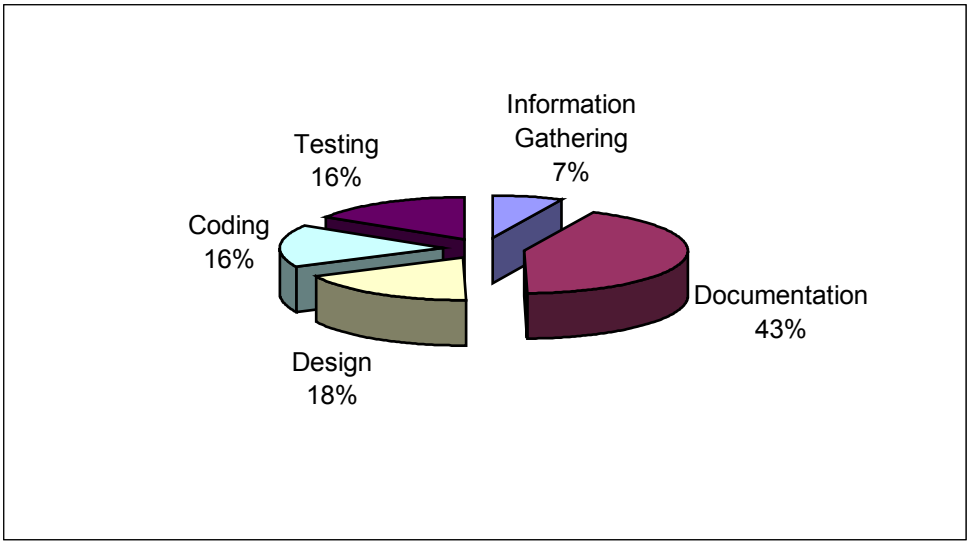

Figure 32. Project Work Breakdowns

This chart diagram shows work breakdown time spent for coding, testing, information gathering, design and documentation during the first phase.

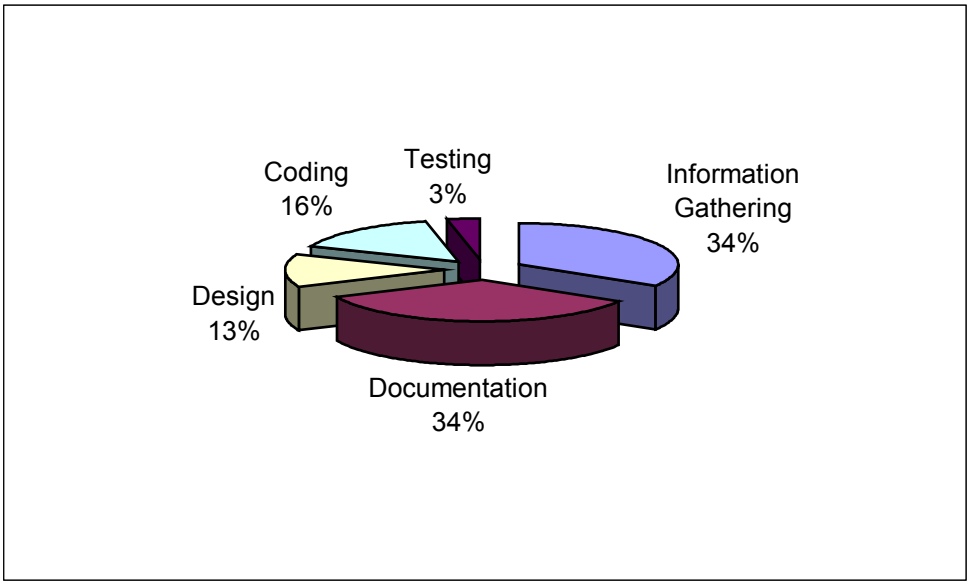

Figure 33. Phase I Work Breakdown diagram.

This chart diagram displays work breakdown time spent for coding, testing, information gathering, design and documentation during the second phase.

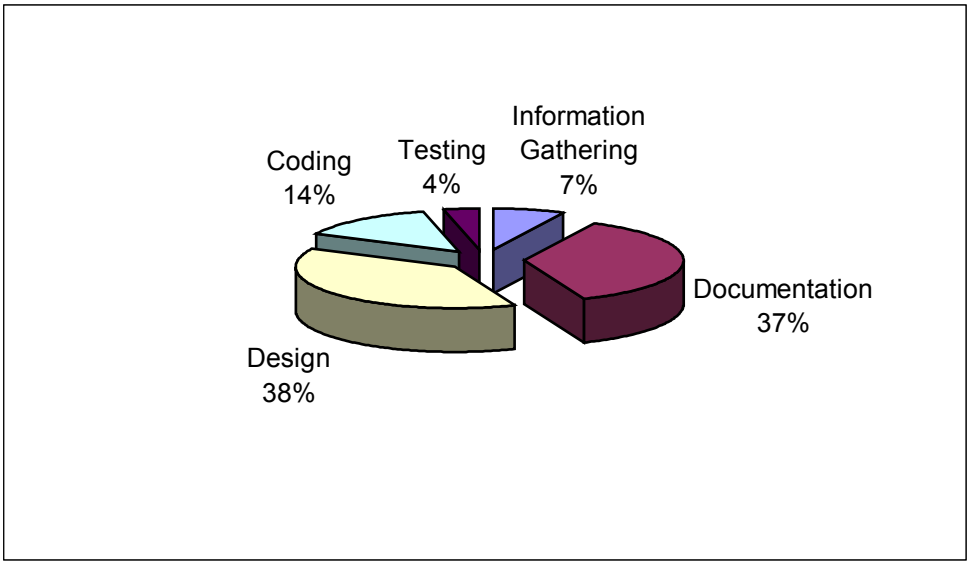

Figure 34. Phase II Work Breakdown diagram.

This chart diagram illustrates work breakdown time spent for coding, testing, information gathering, design and documentation during the third phase.

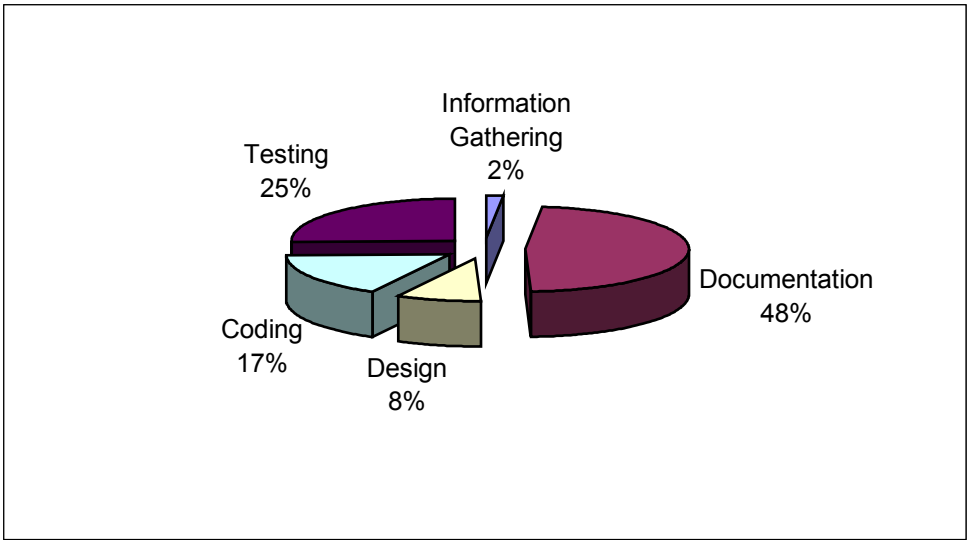

Figure 35. Phase III Work Breakdown diagram.

4. Product quality

With the amount of effort and time spent in this project for designing, documenting, coding and testing, the final product has accomplished the goal presented in the requirement specification.

5. Lessons Learned

This project gave me extensive experience in software development process as well as working in a team. In most cases, inexperience programmers more than likely to construct a software without thinking about design, as a result, deliver a flawed program with many bugs and less feature than it is supposed to have. In my opinion, this project has a great contribution on developing a professional working environment. Planning and designing is critical for a successful programmer in order to develop a good program in a timely manner. Planning and design has helped me save a lot of time in coding the program. I have redesigned the UML diagram several times to make it meet the requirements. The redesign process sometimes happened while I was coding. I got the feeling that the iterative software development approach applies to the real situation of the development process. Unlike the old approach where each step is done before stepping into the next one, each step in the iterative development process can be modified while working in the next step. I also received some guidance from the test plan, which I wrote up before testing. First I did not follow the test plan while I was testing the software. After I realized that I had a test plan on the shelf, I looked into it and followed

the plan. It had really helped me out on some test cases, which I had overlooked before. I had to incorporate my part of the program into other people's program. The Cooperative Robotic Simulator team meeting, which was held once a week, gave me an opportunity to communicate with my teammates who provided me some resolutions to integrate with their parts.

4. Future Work

While all the features listed in Requirement Specification were implemented, more features should be added to enhance the ability of the system. Furthermore, the Communication Model should be separated from Environment to give a better performance. As a result, the future work will be described as two categories, add-on feature and performance improvement.

4.1 Add-on Feature

The communication model should support multicast communication. To make it better, it should have the features to add robot into the group list and provide functions to join and leave the group.

4.2 Performance Improvement Guideline

The following is a guideline to redesign the Communication Model to make it independent from the Environment to provide more efficiency to the system. However, this is just one of the options to provide more efficient system.

In order to make the Communication Model run separately, a number of connections need to be constructed. Since the Communication Model has to communicate with the Robot, the Environment and the Environment Control Panel, three TCP/IP connections will be generated. The connection between the Communication Model and the Robot is to pass messages back and forth. The connection between the Communication Model and the Environment is used for synchronization (update time step) as well as a list of distance. The list of distance, which is a list of distance between two robots, is used to determine if a robot is within the range limit of a sender. The last connection, the Communication Model and the Environment Control Panel, is applied to update the system and robot parameters.

The protocol of these three connections needs to be clarified. The following is the example of these protocols.

- **Communication Model – Environment Protocol**

  For each time step, the Environment will send current time step and a list of updated distance record to the Communication Model automatically. A list of updated distance record is a list of distance between two robots, which has changed from the previous time step.

- **Communication Model – Robot Protocol**

After connection is initiated, Robot sends its name and communication type to the Communication Model respectively. These two values, robot name and communication type, are used to register a robot to the communication system. Thus, Messages will be passed back and forth from this point.

- **Communication Model – Environment Control Panel Protocol**

This is a two-way communication. The Environment Control Panel will send request to the communication system and communication system will return a response back to the Environment Control Panel. Most of request is to set and get system**,** or robot parameters.

Class Diagram

This is the modified Communication Model Class Diagram to support the separation. There are five classes added into this new model, RobotServer, ControlPanelServer, ControlPanelObject, SendHandler, and ReceiveHandler. RobotServer class acts as a server socket waiting for connection from robots. The ControlPanelServer class also acts as a server socket but is used to establish connection to the Environment Control Panel. The ControlPanelObject class is used to communicate with the Environment Control Panel. It keeps input and output stream connection to the Environment Control Panel. This class will be run as a thread.

There are some changes in the current classes, CommunicationsSystem class, RobotCommRecord class and RobotParameters class, whereas the others will be the same. Some attributes regarding connection will be added to the CommunicationsSystem. Also, the CommunicationsSystem class will run as a thread, since it requires the current time step and a list of distance from the Environment. Some methods provided in the CommunicationsSytem will be modified to support this changes. In addition to keep the communication information for a robot, the RobotCommRecord will create two more processes, sendHandler and receiveHandler. SendHandler is responsible for managing incoming message from a robot and pass messages to his queue. ReceiveHandler will handle outgoing message by checking a robot's queue if there are messages to be sent out to the robot. SendHandler and ReceiveHandler will run as threads and handle each robot individually. Finally, distance between a pair of robot will be added to the RobotParameter class.

More details are shown as diagrams, which are provided in the next section. Class Diagram will show how new classes tie with the old one and some attributes correspond to the connection that need to be supplied in each class. Furthermore, the procedures of the significant functions, which are registering robot, sending message, receiving message, setting parameter and getting time step, will be described as sequence diagrams.

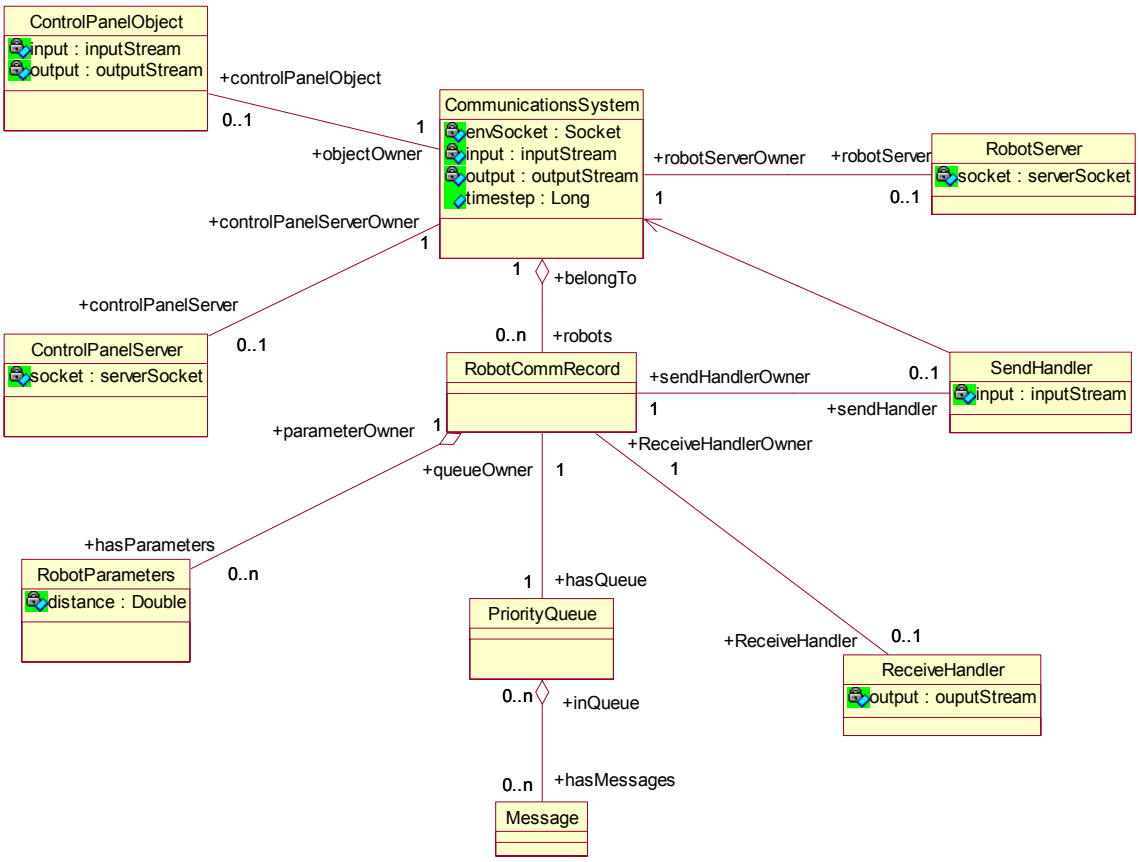

Figure 36. Redesigned – Communication Model Class Diagram

Sequence Diagram

- **Register Robot – Sequence Diagram**

First, CommunicationsSystem starts RobotServer as a server socket and waiting for connections from robots. Robot establishes connection to the RobotServer. Then, RobotServer accepts the connection and passes connection to CommunicationsSystem to register this robot into the system. After that, A RobotCommRecord is created with robot name, communication type and connection. Finally, the RobotCommRecord will create sendHandler and receiveHandler to deal with incoming and outgoing messages.

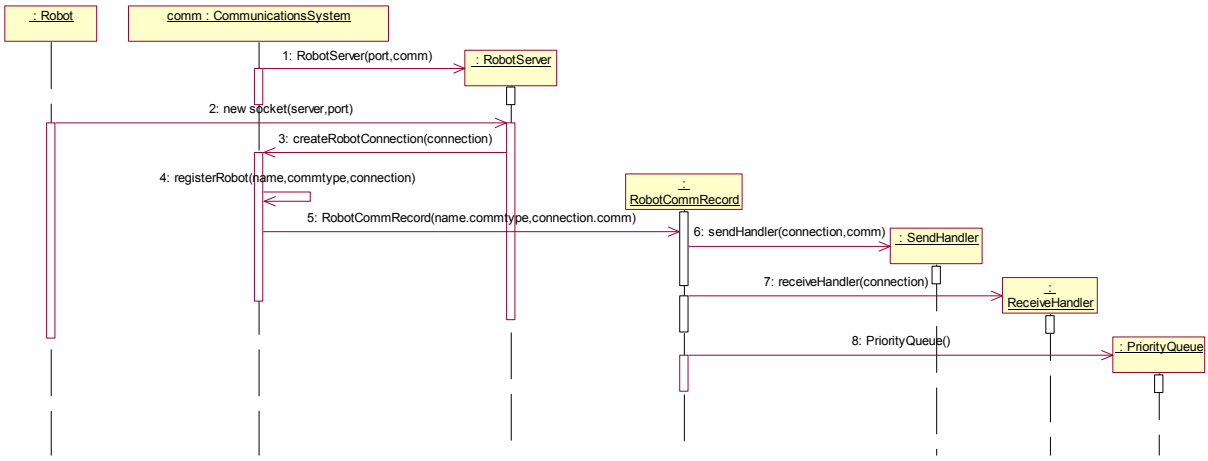

Figure 37. Redesigned – Register Robot Sequence Diagram

- **Send Message – Sequence Diagram**

Since SendHandler runs as a thread, whenever there is an incoming message written to the socket, it will read from the socket and process this message by calling sendMessage method.

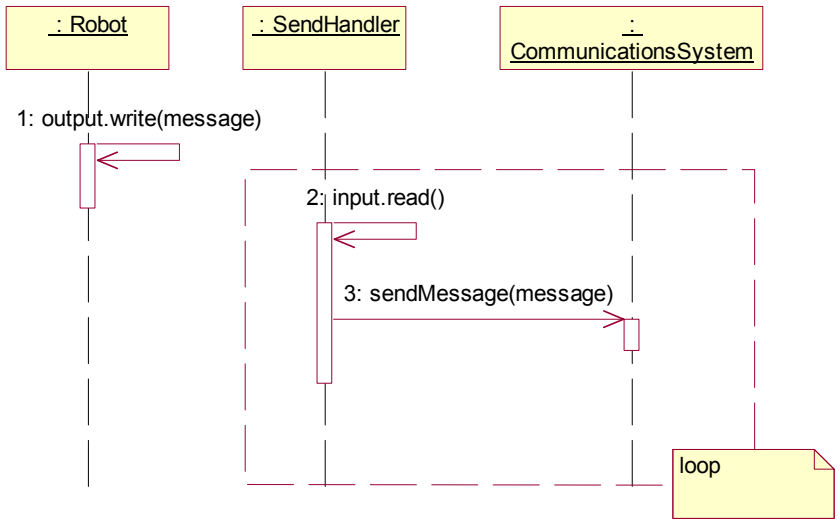

Figure 38. Redesigned – Send Message Sequence Diagram

Although, this sendMessage method is already provided in the CommunicationsSystem, it requires some changes to make it compatible with the new system. The previous sendMessage method requires two arguments: message and time step. However this new system will define the current time step as a public attribute, the time step argument passing along with the sendMessage method will be eliminated. Therefore, there will be only message argument passing with the sendMessage method (as in the third message in the diagram). Moreover, the sequence diagram of send message will be changed, since the distance between each pair of robot can be retrieved from the RobotParameter class instead of taking directly from the Environment. The followings are the redesigned sequence diagram of sendMessage method in CommunicationsSystem class. The first sequence diagram is sending broadcast message diagram and the latter is sending point-to-point message. Both of these diagrams are similar to the current system (as in Chapter 5 , Figure 8 and 9), but the actor, who called the sendMessage method, is changed from the Environment to the SendHandler. Futhermore, the process of retrieving distance between two robots is changed from taking from the Environment to get it directly from the RobotParameter.

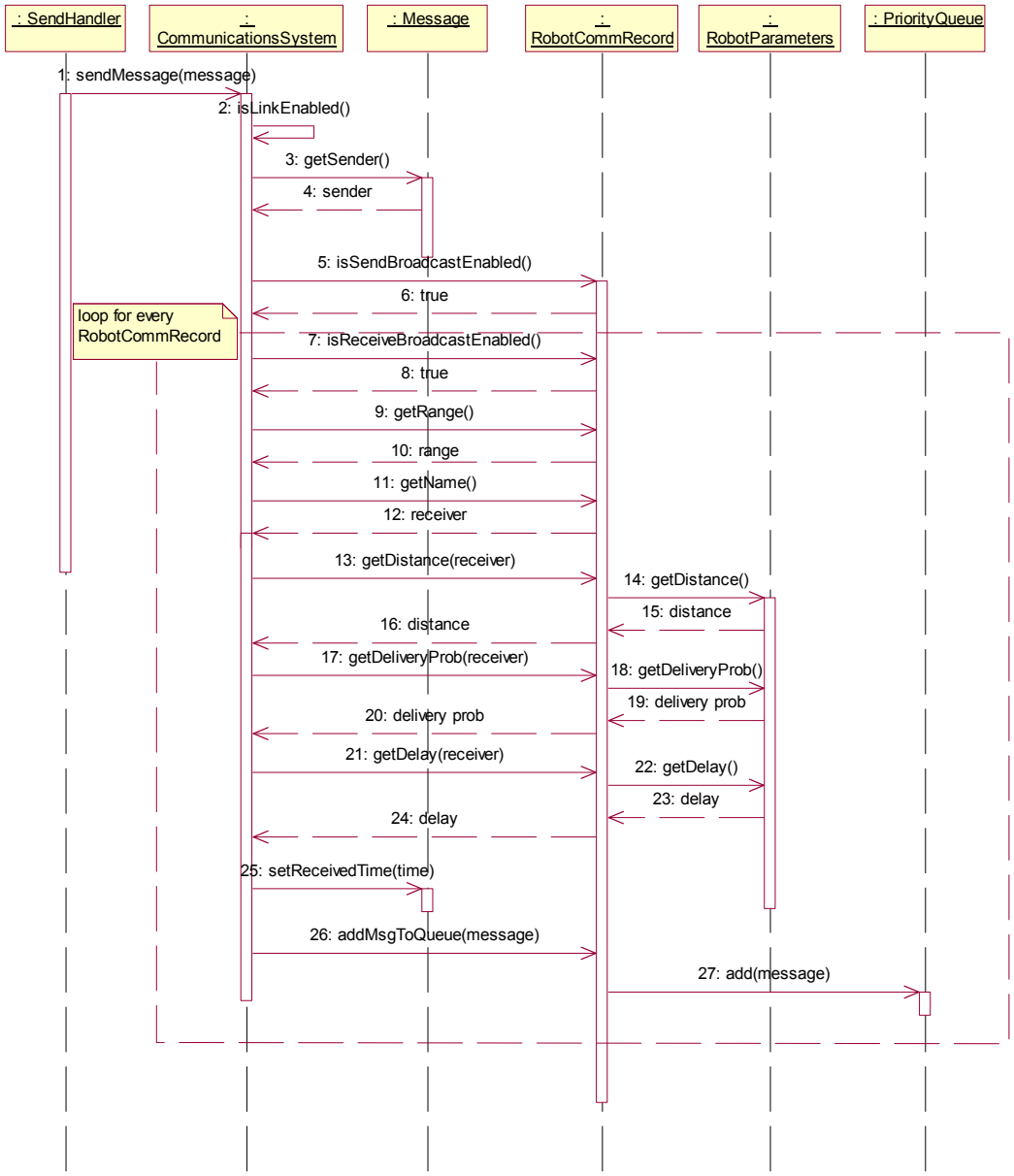

Figure 39. Redesigned – CommunicationsSystem Send Broadcast Message Sequence Diagram

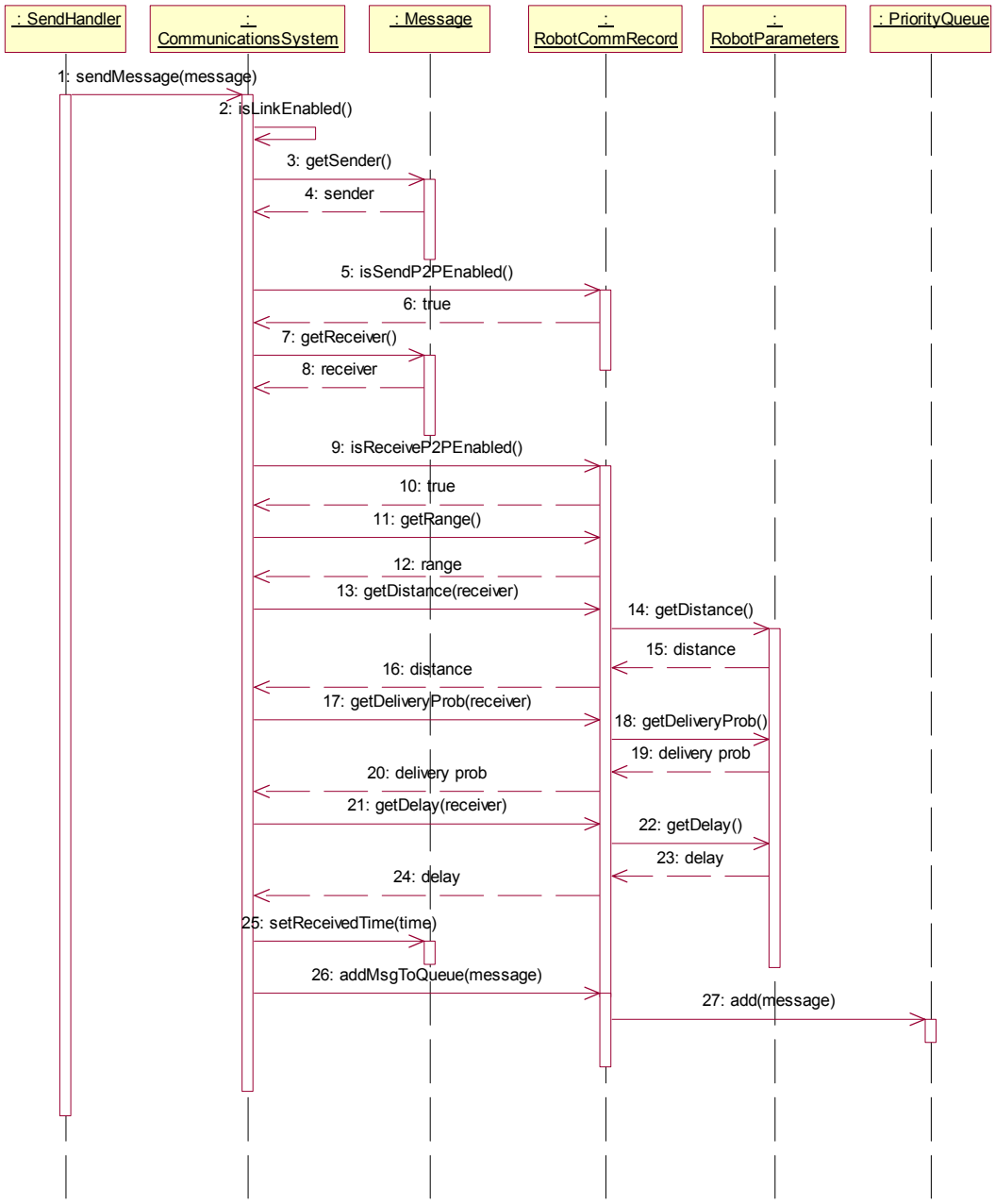

Figure 40. Redesigned – CommunicationsSystem Send Point-to-Point Message Sequence Diagram

- **Receive Message – Sequence Diagram**

ReceiveHandler will handle outgoing messages for a robot to which has direct link. ReceiveHandler also runs as a thread; as a result, each outgoing message will be delivered to the robot in real time. The getMessage method will return a list of messages corresponding to the current time step. Finally the messages will be dispatched to the owner via the socket.

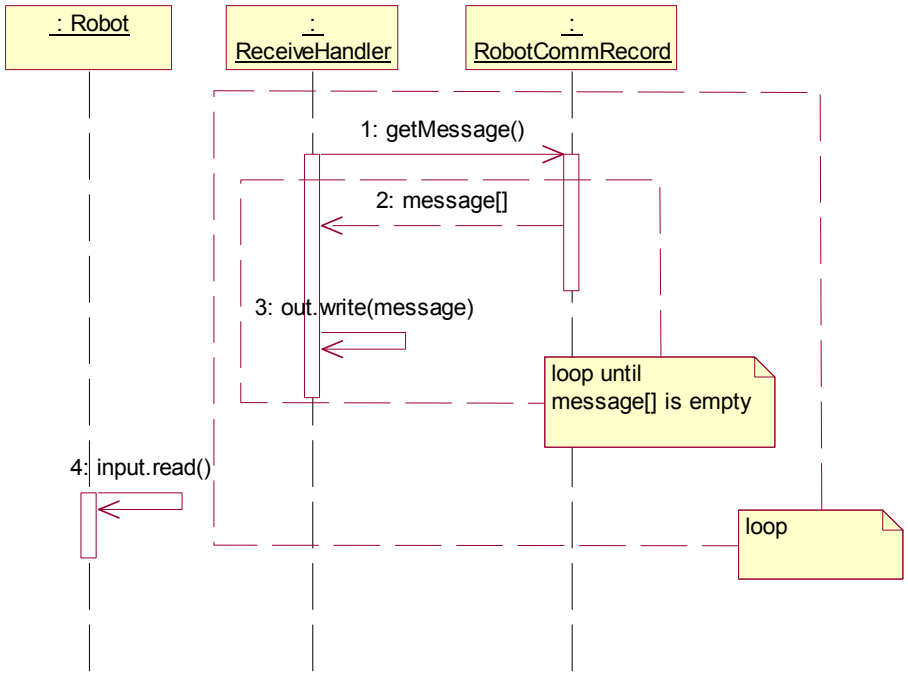

Figure 41. Redesigned – Receive Message Sequence Diagram

Due to the fact that the CommunicationsSystem of the current system acts as the main interface between the Environment and the Communication Model, every method call is provided in the CommunicationsSystem. Nevertheless, the new system does not require this feature; the ReceiveHandler can call getMessage method directly from the RobotCommRecord instead of calling getMessage method provided in CommunicationsSytem. Therefore, the getMessage method in CommunicationsSystem can be removed. However, the getMessage method provided in the RobotCommRecord will be modified to take no arguments because the current time step will be defined in the CommunicationsSystem as a public attribute.

- **Set parameters – Sequence Diagram**

This task will be done by the Environment Control Panel. CommunicationsSystem starts the ControlPanelServer which is server socket that is waiting for connection from the EnvironmentControlPanel. The EnvironmentControlPanel creates a socket to connect to the ControlPanelServer. After the connection has been created, the CommunicationsSystem will create a ControlPanelObject to keep the connection between the CommunicationsSystem and the EnvironmentControlPanel. For every request from the EnvironmentControlPanel, the ControlPanelObject will process and return a response back to the EnvironmentControlPanel through the socket.

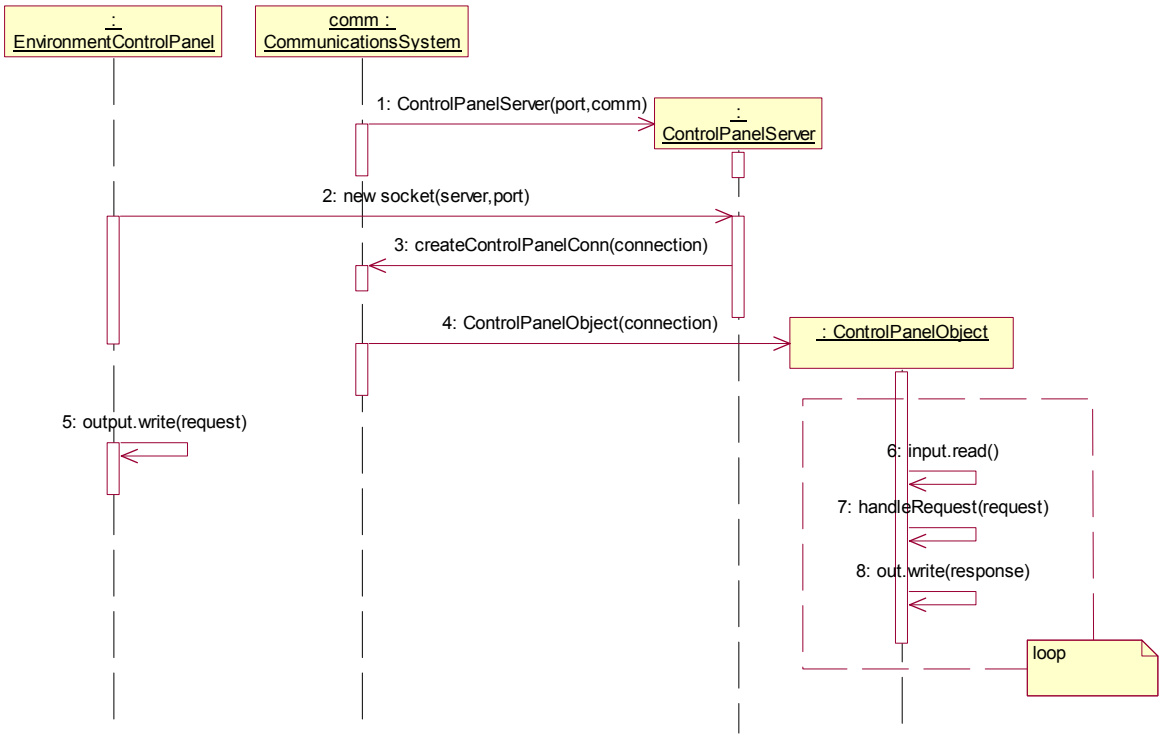

Figure 42. Redesigned – Set Parameter Sequence Diagram

- **Get time step and distance from Environment – Sequence Diagram**

This section is the process of getting current time step from the Environment. Since the Environment is the central part of the system, the server socket will be started by the Environment. As stated in the diagram, this class is EnvironmentCommServer. It will wait for a connection from the CommunicationsSystem. When the connection has been established the EnvCommObject will be generated to keep connection information. Whenever the current time step has been updated, the Environment will send this information along with a list of updated distance to the EnvCommObject. After EnvCommObject gets this information, it will inform these changes to the CommunicationsSystem via the socket and the CommunicationsSystem will update these two values in the system to reflect the changes.

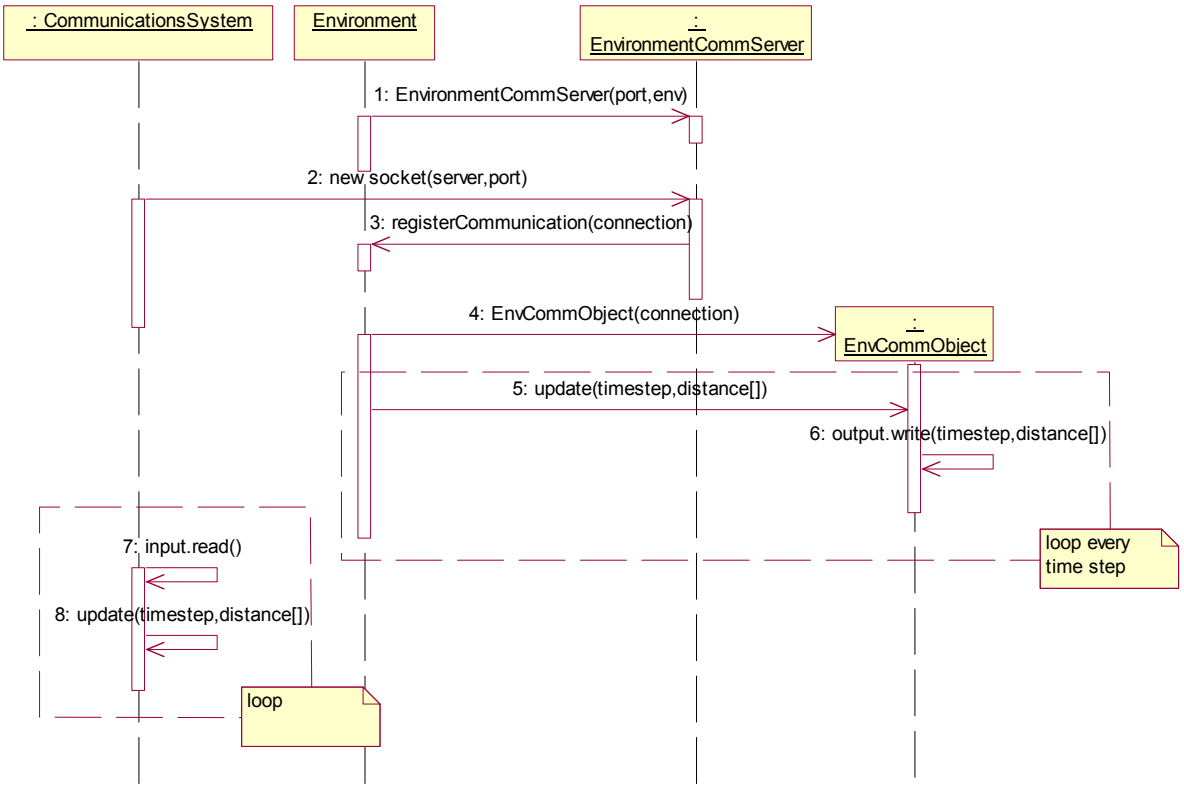

Figure 43. Redesigned – Get Time Step and Distance Sequence Diagram

APPENDIX A

UML/OCL SPECIFICATION

```
--
-- Description: Formal Requirement Specification based on
-- Communications System for Cooperative Robotics Simulator
-- architecture design using UML/OCL methodology.
--
-- Author: Acharaporn Pattaravanichanon
-- File name: communicationSystem.use
-- Course: CIS895 MSE project
-- Computing and Information Systems Department
-- Kansas State University, Spring 2003
-- Date: March 12, 2004
--
--
--

model Communication

--
-- C L A S S E S
--
class CommunicationsSystem
attributes
  delay : Integer
  range : Integer
  deliveryProb : Integer
  isLinkEnabled : Boolean
operations
  registerRobot(n:String,c:Integer)
  getMessage(n:String,timeStep:Integer):Set(Message)
  sendMessage(msg:Message,timeStep:Integer)
  processBroadcast()
  processP2P()
  distributeMessage()
  getRobotCommRecord():RobotCommRecord
  setDelay(delay:Integer)
  setRange(range:Integer)
  setDeliveryProb(prob:Integer)
  startupAllLink()
  shutdownAllLink()
  setRobotDelay(sender:String,receiver:String,delay:Integer)
  setRobotRange(name:String,range:Integer)
  setRobotDeliveryProb(sender:String,receiver:String,prob:Integer)
  startupSendLink(name:String)
  startupReceiveLink(name:String)
  shutdownSendLink(name:String)
  shutdownReceiveLink(name:String)
  isLinkEnabled():Boolean
end
```

```
class RobotCommRecord
attributes
  name : String
  range : Integer
  isSendLinkEnabled : Boolean
  isReceiveLinkEnabled : Boolean
  isBroadcastEnabled : Boolean
  isP2PEnabled : Boolean
operations
  getMessage(timeStep:Integer):Set(Message)
  startupSendLink()
  startupReceiveLink()
  shutdownSendLink()
  shutdownReceiveLink()
  enableBroadcast()
  enableP2P()
  disableBroadcast()
  disableP2P()
  isSendLinkEnabled():Boolean
  isReceiveLinkEnabled():Boolean
  isBroadcastEnabled():Boolean
  isP2PEnabled():Boolean
  isSendBroadcastEnabled():Boolean
  isReceiveBroadcastEnabled():Boolean
  isSendP2PEnabled():Boolean
  isReceiveP2PEnabled():Boolean
  getRobotParameter():RobotParameter
  getDelay(name:String):Integer
  getRange():Integer
  getDeliveryProb(name:String):Integer
  addMsgToQueue(msg:Message):Boolean
  setCommType(commType:Integer)
end

class RobotParameter
attributes
  receiverName : String
  delayTime : Integer
  deliverProb : Integer
operations
  getReceiveName():String
  getDelay():Integer
  getDeliveryProb():Integer
  setReceiveName(name:String)
  setDelay(delay:Integer)
  setDeliveryProb(prob:Integer)
end

class PriorityQueue
attributes
operations
  add(msg:Message):Boolean
  get(index:Integer):Message
  remove(index:Integer):Message
  isEmpty():Boolean
```

```
end

class Message
attributes
  sender : String
  receiver : String
  content : OclAny
  receivedTime : Integer
  sentTime : Integer
operations
  setSender(name:String)
  setReceiver(name:String)
  setContent(content:OclAny)
  setReceivedTime(time:Integer)
  setSentTime(time:Integer)
  getSender():String
  getReceiver():String
  getContent():OclAny
  getReceivedTime():Integer
  getSentTime():Integer
  isBroadcastMessage():Boolean
  isP2PMessage():Boolean
end

--
-- A S S O C I A T I O N S
--

-- robot: a communication system consists of many robots

association robot between
  CommunicationsSystem[1] role belongTo
  RobotCommRecord[*] role robots
end

-- parameter : a RobotCommRecord has a list of robot parameter

association parameter between
  RobotCommRecord[1] role parameterOwner
  RobotParameter[*] role hasParameters
end

-- queue : a RobotCommRecord has a priorityQueue

association queue between
  RobotCommRecord[1] role queueOwner
  PriorityQueue[1] role hasQueue
end

-- message : A queue consists of some messages

association message between
  PriorityQueue[*] role inQueue
  Message[*] role hasMessages ordered
end
```

```
--
-- C O N S T R A I N T S
--

constraints

--
-- UniqueName
-- Robot has unique name
--

context RobotCommRecord
  inv UniqueName:
      RobotCommRecord.allInstances->forAll(p1,p2| p1 <> p2
      implies p1.name <> p2.name)

--
-- BroadcastAbility:
-- Robot has broadcast ability can send and receive broadcast messages.
--

context RobotCommRecord
  inv BroadcastAbility1:
      RobotCommRecord.allInstances.hasQueue.hasMessages
        ->select(receiver='broadcast' and sender=self.name)->notEmpty
        implies self.isBroadcastEnabled = true


context r:RobotCommRecord
  inv BroadcastAbility2:
      r.hasQueue.hasMessages
        ->select(receiver='broadcast')->notEmpty
        implies r.isBroadcastEnabled = true


--
-- P2PAbility
-- Robot has point-to-point ability can send and receive point-to-point
-- messages.
--

context RobotCommRecord
  inv P2PAbility1:
      RobotCommRecord.allInstances.hasQueue.hasMessages
        ->select(receiver <> 'broadcast'
        and sender=self.name)->notEmpty
        implies self.isP2PEnabled = true

context r:RobotCommRecord
  inv P2PAbility2:
      r.hasQueue.hasMessages
        ->select(receiver <> 'broadcast')->notEmpty
        implies r.isP2PEnabled = true
```

```
--
-- sendAbility
-- Robot with active send link can send messages to other robots
--

context RobotCommRecord
  inv sendAbility:
      RobotCommRecord.allInstances.hasQueue.hasMessages
        ->select(sender=self.name)->notEmpty
        implies self.isSendLinkEnabled = true

--
-- receiveAbility
-- Robot with active receive link can receive messages from other
robots
--

context r:RobotCommRecord
  inv receiveAbility:
      r.hasQueue.hasMessages->notEmpty
      implies r.isReceiveLinkEnabled = true

--
-- rightQueue
-- Messages are distributed to the right robot queue
--

context RobotCommRecord
   inv rightQueue:
       hasQueue.hasMessages->
       forAll((receiver=self.name) or (receiver='broadcast'))

--
-- priorityQueue
-- The messages in priorityQueue are ordered by received time
--
--

context p:PriorityQueue
  inv priorityQueue:
      Sequence{1..(p.hasMessages->size-1)}->
      forAll(i | p.hasMessages->at(i).receivedTime
      <= p.hasMessages->at(i+1).receivedTime)
--
-- rightTime
-- If received time is defined, then received time is equal or greater
-- than sent time.
--

context m:Message
  inv rightTime:
      m.receivedTime.isDefined implies m.receivedTime >= m.sentTime
```

```
--
-- allLinkShutdown
-- Robots cannot send or receive any messages if all the links are --
-- shutdown
--

context c:CommunicationsSystem
  inv allLinkShutdown:
      RobotCommRecord.allInstances.hasQueue.hasMessages->notEmpty
       implies c.isLinkEnabled = true

--
-- sendToYourself
-- Robots cannot get the messages which are sent my themselves.
--

context r:RobotCommRecord
  inv sendToYourself:
      r.hasQueue.hasMessages->forAll(sender <> r.name)

--
-- O P E R A T I O N S
--
-- registerRobot pre/post-conditions
--  .. pre-conditions
--       1. robot name (n) is defined
--       2. communication type (c) is defined
--       3. c must be only 1,2 or 3
--       4. there's no exist robot named "n" in the RobotCommRecord set
--  .. post-conditions
--       1. new Robot named "n" is created
--       2. new set of RobotCommRecord is the previous set plus new
--          robot named "n"
--       3. there is only one record of RobotCommRecord which is named
--          "n"
--       4. if c=1 then Robot has broadcast capability
--       5. if c=2 then Robot has Point-to-point capability
--       6. if c=3 then Robot has both broadcast and point-to-point
--          capability

context CommunicationsSystem::registerRobot(n:String,c:Integer)
  pre precond_1:    n.isDefined
  pre precond_2:    c.isDefined
  pre precond_3:    (c=1 or c=2 or c=3)
  pre precond_4:    robots->select(name=n)->isEmpty
  post postcond_1:  robots->exists(r | r.oclIsNew and r.name = n)
  post postcond_2:  robots=robots@pre->union(robots->select(name=n))
  post postcond_3:  robots->select(name=n)->size = 1
  post postcond_4:  (c=1) implies robots->select(name=n and
                    isBroadcastEnabled=true
                    and isP2PEnabled=false)->notEmpty
  post postcond_5:  (c=2) implies robots->select(name=n and
                    isP2PEnabled=true and
                    isBroadcastEnabled=false)->notEmpty
  post postcond_6:  (c=3) implies robots->select(name=n and
                    isBroadcastEnabled=true and
```

```
                    isP2PEnabled=true)->notEmpty

--
-- sendMessage pre/post-conditions
--  .. pre-conditions
--       1. timeStep is greater than zero
--       2. message is defined
--       3. sender is defined
--       4. receiver is defined
--       5. System link is enabled
--       6. sender's send link is active
--       7. if it is broadcast message, sender has broadcast capability
--       8. if it is point-to-point message, sender has point-to-point
--          capability
--
--  .. post-conditions
--       1. sentTime which is timeStep is added to the Message
--       2,3. receivedTime is added based on system delay and robot ---
--          delay
--       The msg is added to the receiver's queue
--       4. if it is broadcast message then
--            message is added to all robot's queue except sender's --
--            queue
--       5. if it is Point-to-point message then
--            the message is added to the specified robot's queue

context CommunicationsSystem::sendMessage(msg:Message,timeStep:Integer)
  pre precond_1:    timeStep > 0 and timeStep.isDefined
  pre precond_2:    msg.isDefined
  pre precond_3:    msg.sender.isDefined
  pre precond_4:    msg.receiver.isDefined
  pre precond_5:    isLinkEnabled = true
  pre precond_6:    robots->select(name=msg.sender)
                     ->forAll(isSendLinkEnabled = true)
  pre precond_7:    msg.receiver = 'broadcast'
                      implies robots->select(name=msg.sender)->
                      forAll(isBroadcastEnabled = true)
  pre precond_8:    msg.receiver <> 'broadcast'
                      implies robots->select(name=msg.sender)->
                      forAll(isP2PEnabled = true)

  post postcond_1:  msg.sentTime = timeStep
  post postcond 2:  msg.receiver <> 'broadcast' implies
                      robots->select(name=msg.sender)->
                      forAll(hasParameters->
                      forAll(receiverName=msg.receiver implies
                      msg.receivedTime = parameterOwner.belongTo.delay
                      + timeStep + delayTime))

  post postcond 3:  msg.receiver = 'broadcast' implies
                      robots->select(name=msg.sender)->
                      forAll(hasParameters->
                      forAll(msg.receivedTime =
                      parameterOwner.belongTo.delay
                      + timeStep + delayTime))
```

```
  post postcond_4:  msg.receiver = 'broadcast' implies
                      robots->select(name <> msg.sender
                      and isReceiveLinkEnabled = true
                      and isBroadcastEnabled = true)
                      ->forAll(r| r.hasQueue.hasMessages->asSet
                      = r.hasQueue.hasMessages@pre
                      ->including(msg)->asSet)

  post postcond_5:  msg.receiver <> 'broadcast' implies
                      robots->select(name = msg.receiver
                      and isReceiveLinkEnabled = true
                      and isP2PEnabled = true)
                      ->forAll(r| r.hasQueue.hasMessages->asSet
                      = r.hasQueue.hasMessages@pre
                      ->including(msg)->asSet)


--
-- getMessage pre/post-conditions
--  .. pre-conditions
--       1. n (robot name) is defined
--       2. robot named n exists in the system
--       3. timestep is defined
--       4. timestep is greater than zero
--
--  .. post-conditions
--       1. The messages in priority queue of robot n
--          excludes message which has receivedTime = timestep
--       2. Return result which is a set of messages
--          which receivedTime = timestep
--

context CommunicationsSystem::getMessage(n:String,timeStep:Integer)
:Set(Message)
  pre precond_1:    n.isDefined
  pre precond_2:    robots.exists(r| r.name=n)
  pre precond_3:    timeStep.isDefined
  pre precond_4:    timeStep > 0
  post postcond_1:  robots->select(name = n)
                     ->forAll(r | r.hasQueue.hasMessages->asSet
                     = r.hasQueue.hasMessages@pre->asSet
                     - r.hasQueue.hasMessages@pre
                     ->select(receivedTime = timeStep)->asSet)

  post postcond_2:  robots->select(name=n)
                     ->forAll(r| result = r.hasQueue.hasMessages@pre
                     ->select(receivedTime = timeStep)->asSet)
```

APPENDIX B

USE TEST SCRIPT

All test scripts are used counter example to test the correctness of the specification.

1. Robots Robot has unique name.

Scenario: The test script has two robots named "A"

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true

!create robotA:RobotCommRecord
!create robotB:RobotCommRecord
!set robotA.name = 'A'
!set robotB.name = 'A'

!insert (comm1,robotA) into robot
!insert (comm1,robotB) into robot

!create queue1:PriorityQueue
!create queue2:PriorityQueue
!insert (robotA,queue1) into queue
!insert (robotB,queue2) into queue
```

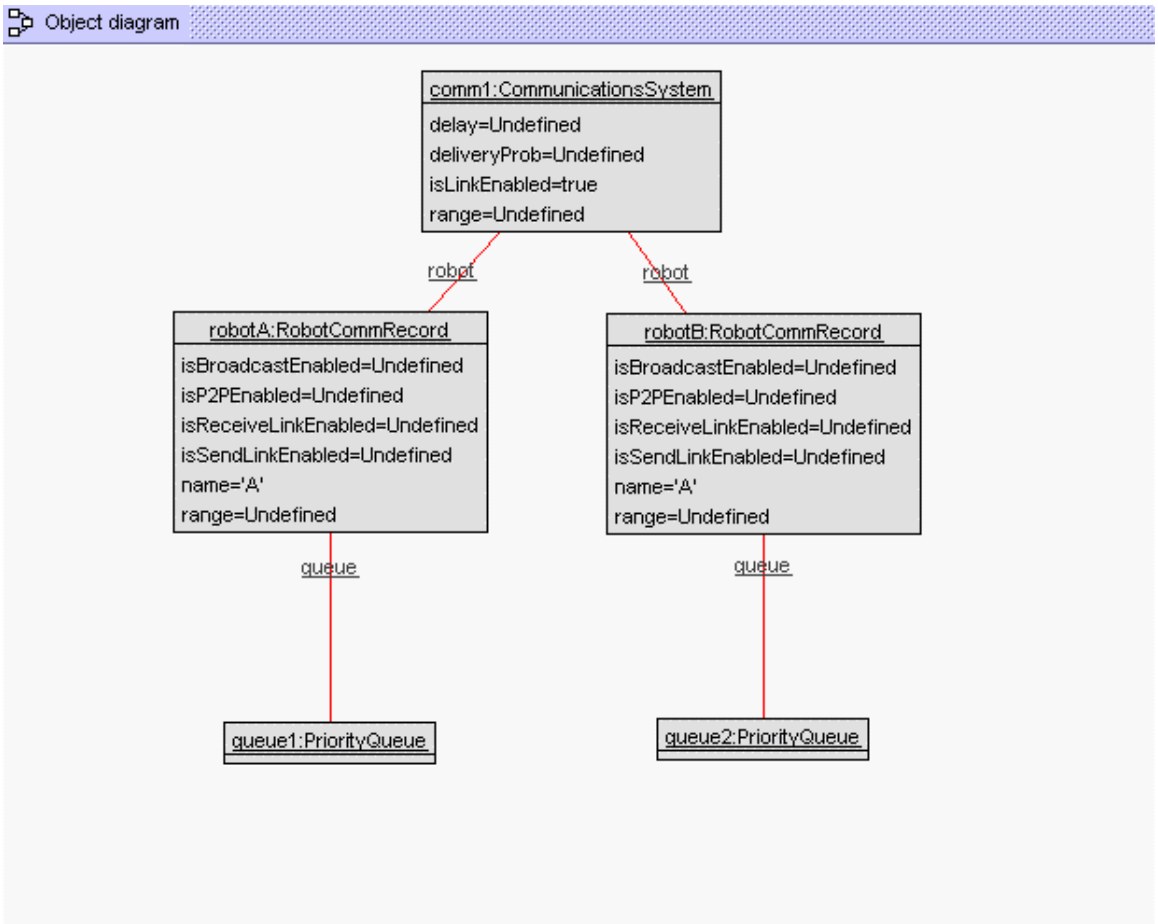

Figure 44. USE Object Diagram – UniqueName Constraint

5. Only robot with broadcast ability can send and receive broadcast message.

Scenario 1: RobotB and RobotC received a broadcast message from RobotA which broadcast is disabled.

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true

!create robotA:RobotCommRecord
!create robotB:RobotCommRecord
!create robotC:RobotCommRecord

!set robotA.name = 'A'
!set robotB.name = 'B'
!set robotC.name = 'C'
!set robotA.isBroadcastEnabled = false
!set robotA.isSendLinkEnabled = true
!set robotA.isReceiveLinkEnabled = true
```

```
!set robotB.isBroadcastEnabled = true
!set robotB.isSendLinkEnabled = true
!set robotB.isReceiveLinkEnabled = true

!set robotC.isBroadcastEnabled = true
!set robotC.isSendLinkEnabled = true
!set robotC.isReceiveLinkEnabled = true

!insert (comm1,robotA) into robot
!insert (comm1,robotB) into robot
!insert (comm1,robotC) into robot

!create queueA:PriorityQueue
!create queueB:PriorityQueue
!create queueC:PriorityQueue

!insert (robotA,queueA) into queue
!insert (robotB,queueB) into queue
!insert (robotC,queueC) into queue

!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'broadcast'

!insert (queueB,msg1) into message
!insert (queueC,msg1) into message
```

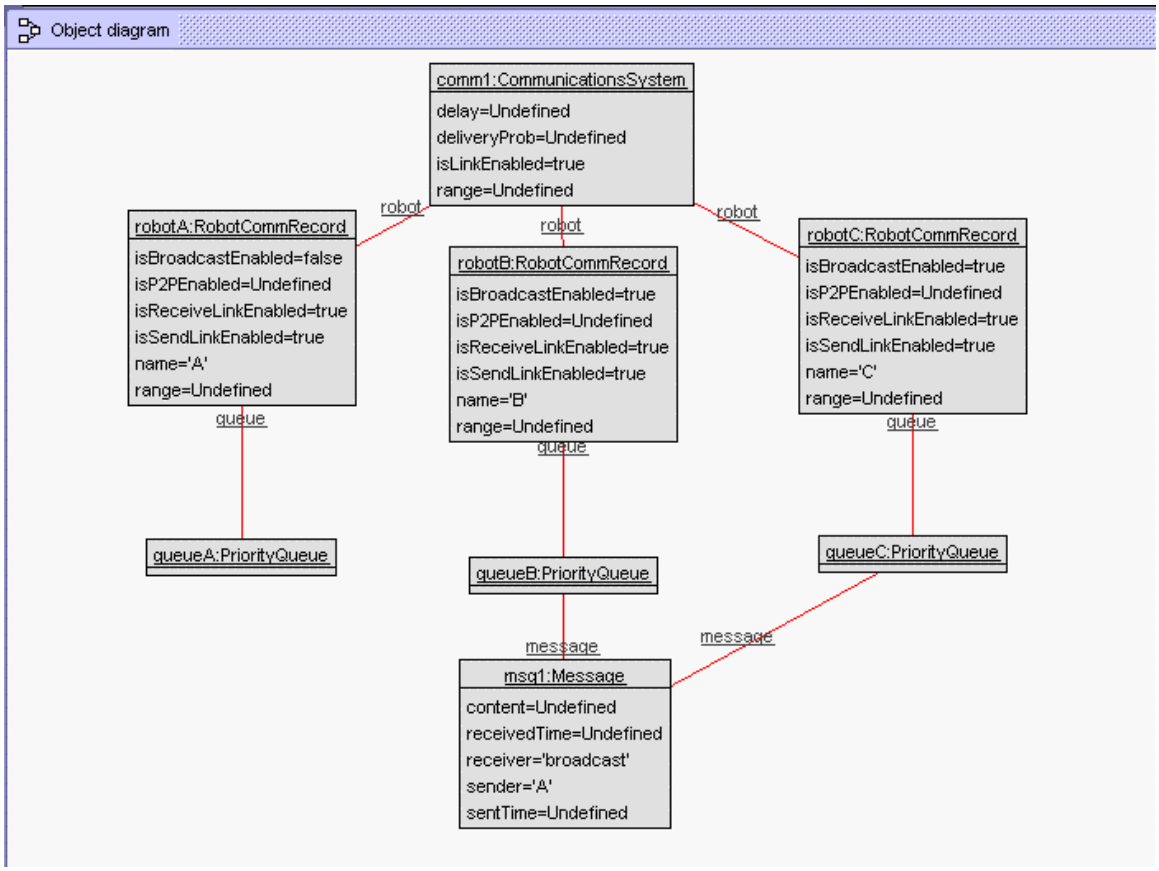

Figure 45. USE Object Diagram – BroadcastAbility1 Constraint

Scenario2: RobotB with disabled broadcast ability received a broadcast message from RobotA.

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true

!create robotA:RobotCommRecord
!create robotB:RobotCommRecord
!create robotC:RobotCommRecord

!set robotA.name = 'A'
!set robotB.name = 'B'
!set robotC.name = 'C'
!set robotA.isBroadcastEnabled = true
!set robotA.isSendLinkEnabled = true
!set robotA.isReceiveLinkEnabled = true

!set robotB.isBroadcastEnabled = false
!set robotB.isSendLinkEnabled = true
!set robotB.isReceiveLinkEnabled = true

!set robotC.isBroadcastEnabled = true
!set robotC.isSendLinkEnabled = true
!set robotC.isReceiveLinkEnabled = true

!insert (comm1,robotA) into robot
!insert (comm1,robotB) into robot
!insert (comm1,robotC) into robot

!create queueA:PriorityQueue
!create queueB:PriorityQueue
!create queueC:PriorityQueue

!insert (robotA,queueA) into queue
!insert (robotB,queueB) into queue
!insert (robotC,queueC) into queue

!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'broadcast'

!insert (queueB,msg1) into message
!insert (queueC,msg1) into message
```

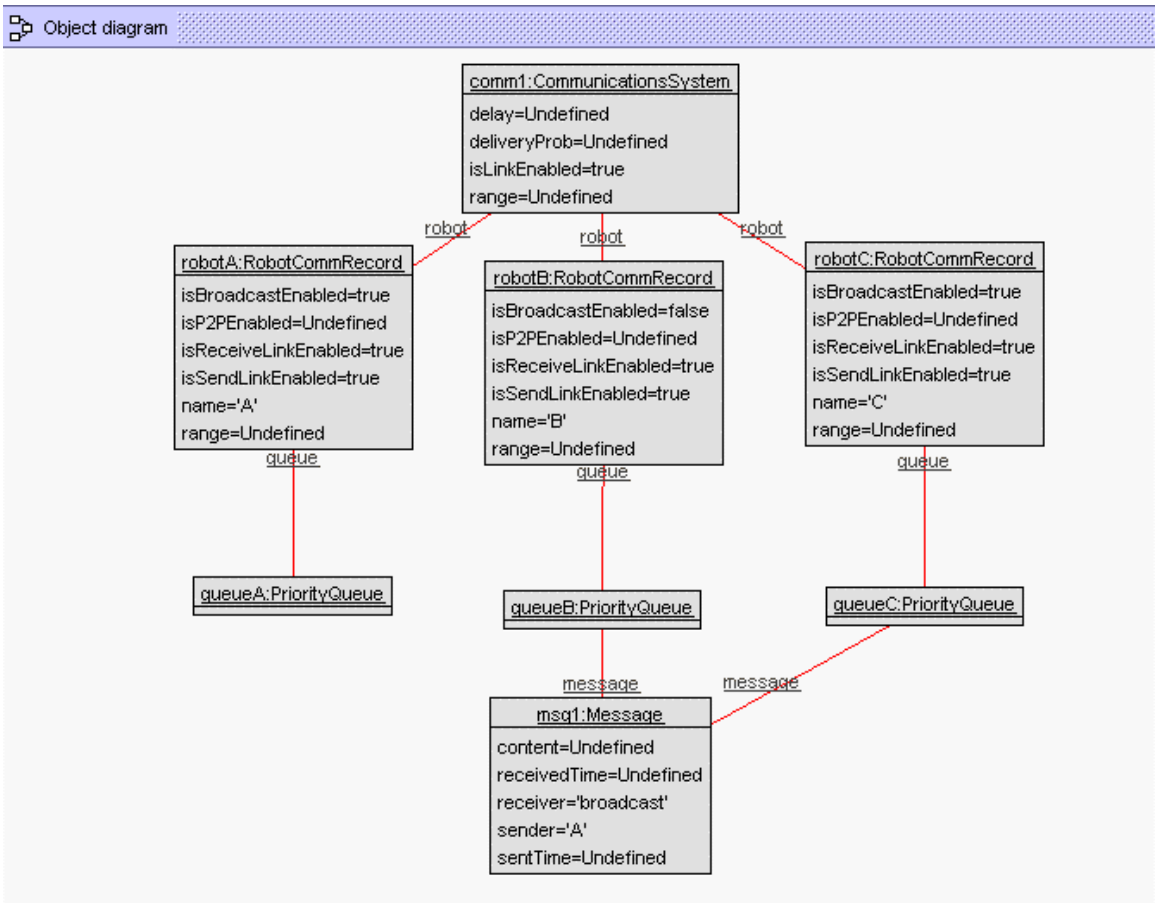

Figure 46. USE Object Diagram – BroadcastAbility2 Constraint

6. Only robot with point-to-point ability can send and receive point-to-point message.

Scenario 1: RobotB received a message from RobotA which point-to-point is disabled.

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true

!create robotA:RobotCommRecord
!create robotB:RobotCommRecord

!set robotA.name = 'A'
!set robotB.name = 'B'
!set robotA.isBroadcastEnabled = true
!set robotA.isP2PEnabled = false
!set robotA.isSendLinkEnabled = true
!set robotA.isReceiveLinkEnabled = true

!set robotB.isBroadcastEnabled = false
!set robotB.isP2PEnabled = true
!set robotB.isSendLinkEnabled = true
!set robotB.isReceiveLinkEnabled = true

!insert (comm1,robotA) into robot
!insert (comm1,robotB) into robot

!create queueA:PriorityQueue
!create queueB:PriorityQueue

!insert (robotA,queueA) into queue
!insert (robotB,queueB) into queue

!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'B'

!insert (queueB,msg1) into message
```

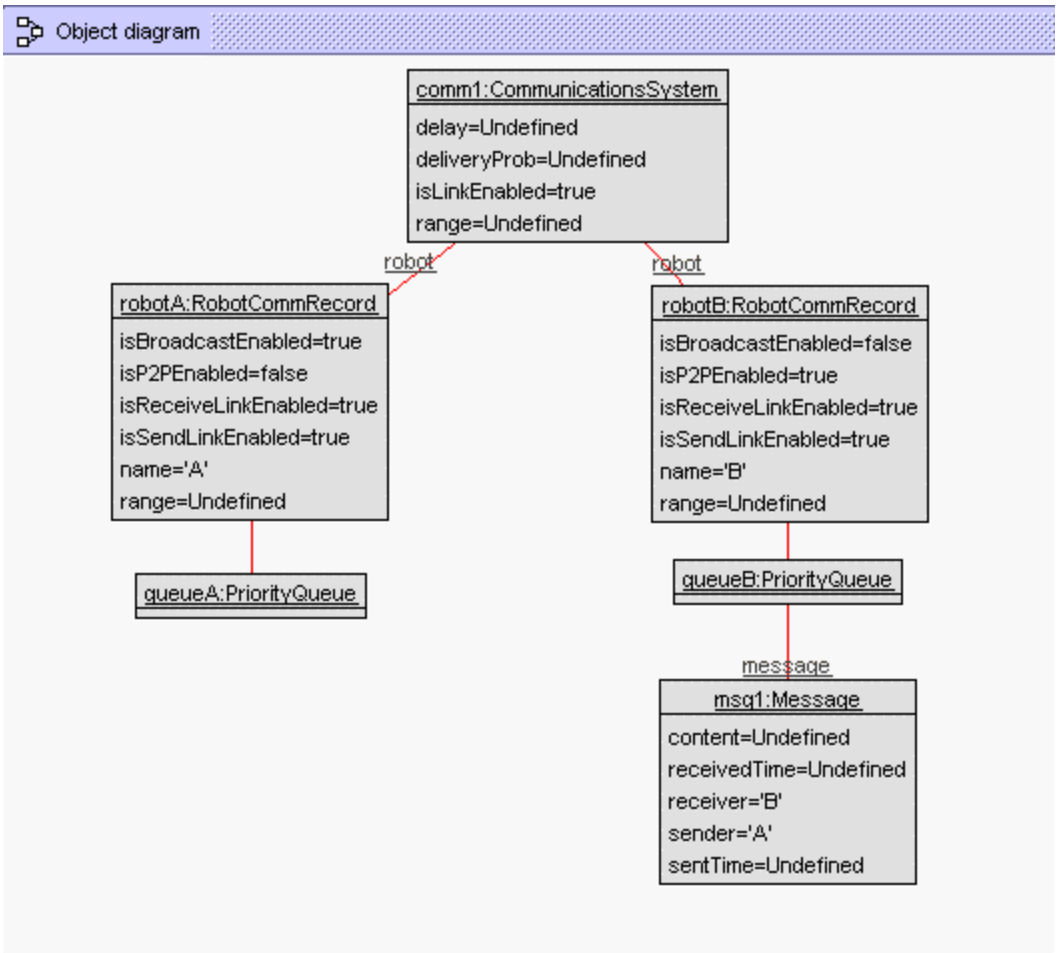

Figure 47. USE Object Diagram – P2PAbility1 Constraint

Scenario 2: RobotB with disabled point-to-point ability received a message from RobotA.

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true

!create robotA:RobotCommRecord
!create robotB:RobotCommRecord

!set robotA.name = 'A'
!set robotB.name = 'B'
!set robotA.isBroadcastEnabled = true
!set robotA.isP2PEnabled = true
!set robotA.isSendLinkEnabled = true
!set robotA.isReceiveLinkEnabled = true

!set robotB.isBroadcastEnabled = false
!set robotB.isP2PEnabled = false
!set robotB.isSendLinkEnabled = true
!set robotB.isReceiveLinkEnabled = true
```

```
!insert (comm1,robotA) into robot
!insert (comm1,robotB) into robot

!create queueA:PriorityQueue
!create queueB:PriorityQueue

!insert (robotA,queueA) into queue
!insert (robotB,queueB) into queue

!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'B'

!insert (queueB,msg1) into message
```

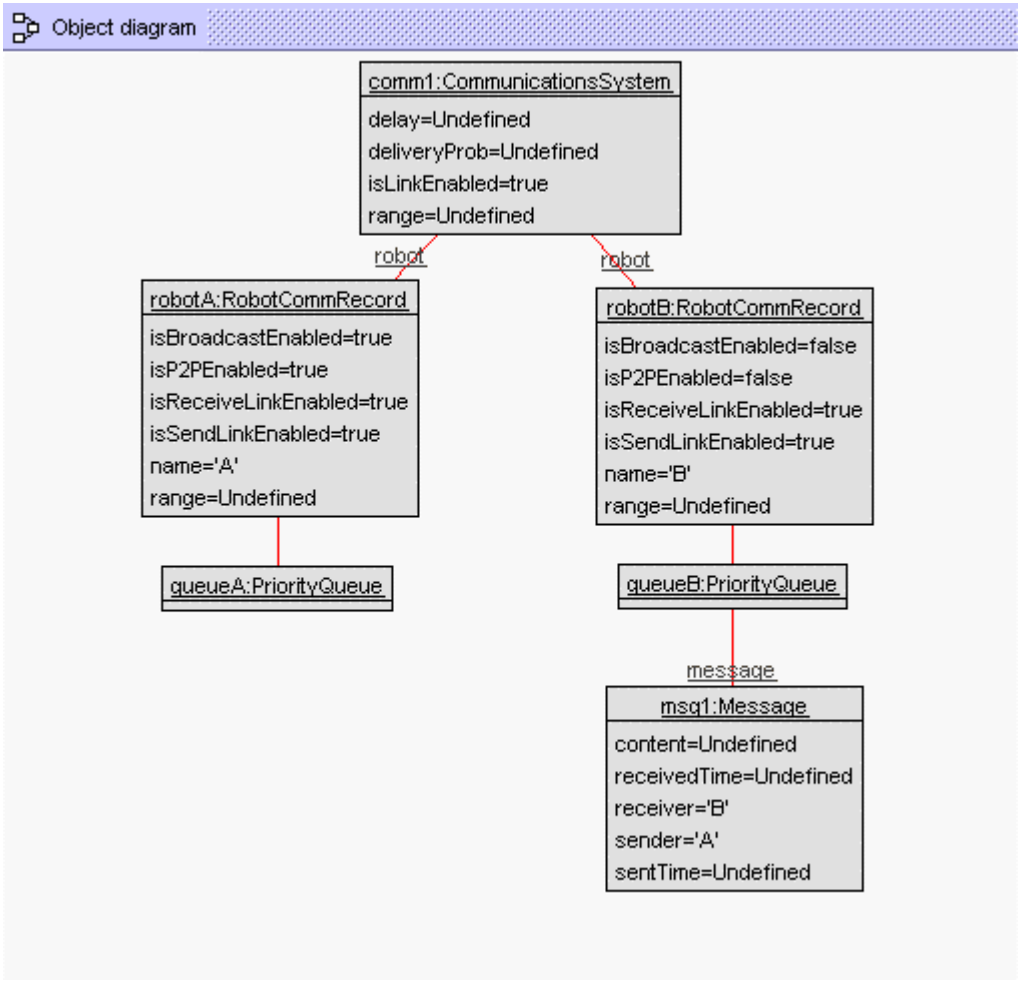

Figure 48. USE Object Diagram – P2Pability2 Constraint

7. Only robot with active send link can send message.

Scenario: RobotB received a message from RobotA which outgoing link is disabled

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true

!create robotA:RobotCommRecord
!create robotB:RobotCommRecord

!set robotA.name = 'A'
!set robotB.name = 'B'
!set robotA.isBroadcastEnabled = true
!set robotA.isP2PEnabled = true
!set robotA.isSendLinkEnabled = false
!set robotA.isReceiveLinkEnabled = true

!set robotB.isBroadcastEnabled = false
!set robotB.isP2PEnabled = true
!set robotB.isSendLinkEnabled = true
!set robotB.isReceiveLinkEnabled = true
!insert (comm1,robotA) into robot
!insert (comm1,robotB) into robot

!create queueA:PriorityQueue
!create queueB:PriorityQueue
!insert (robotA,queueA) into queue
!insert (robotB,queueB) into queue

!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'B'

!insert (queueB,msg1) into message
```

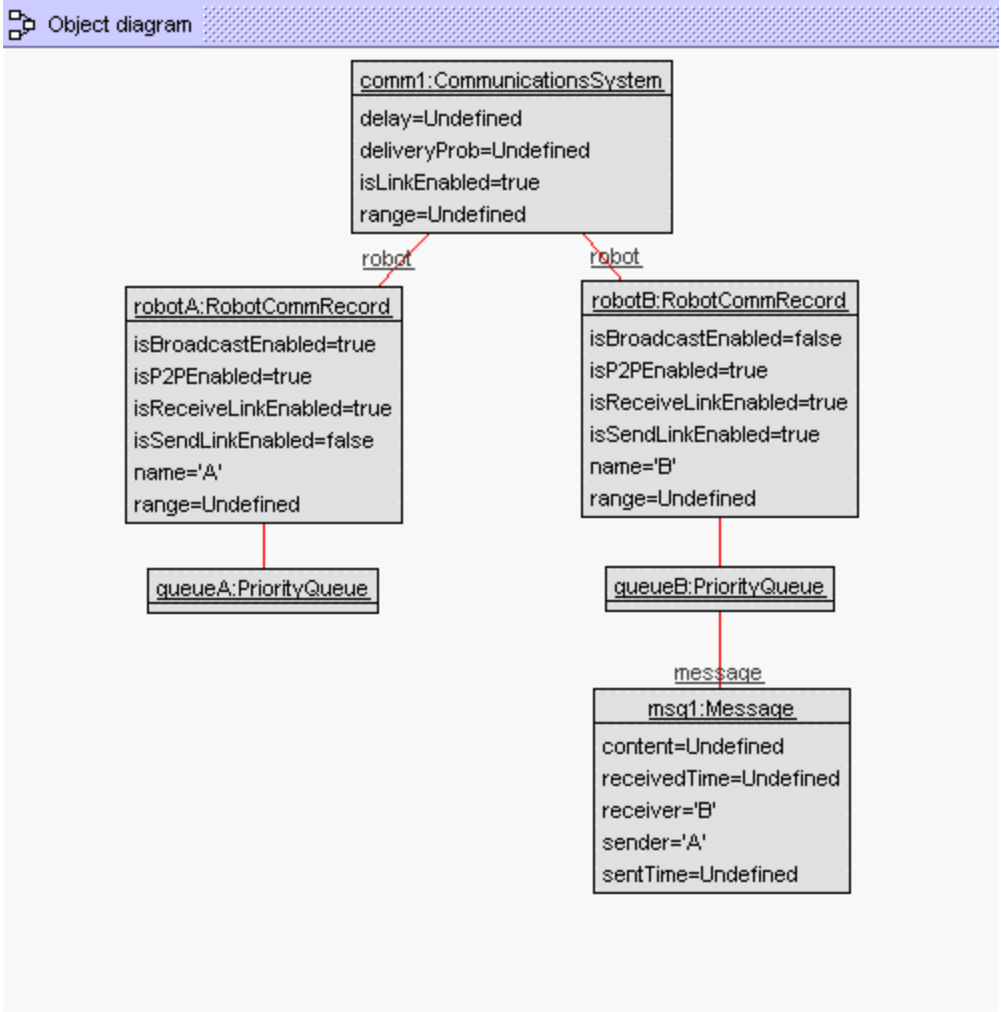

Figure 49. USE Object Diagram – SendAbility Constraint

8. Only robot with active receive link can receive message.

Scenario: RobotB with disabled incoming link received a message from RobotA,

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true

!create robotA:RobotCommRecord
!create robotB:RobotCommRecord

!set robotA.name = 'A'
!set robotB.name = 'B'
!set robotA.isBroadcastEnabled = true
!set robotA.isP2PEnabled = true
!set robotA.isSendLinkEnabled = true
!set robotA.isReceiveLinkEnabled = true
```

```
!set robotB.isBroadcastEnabled = true
!set robotB.isP2PEnabled = true
!set robotB.isSendLinkEnabled = true
!set robotB.isReceiveLinkEnabled = false

!insert (comm1,robotA) into robot
!insert (comm1,robotB) into robot

!create queueA:PriorityQueue
!create queueB:PriorityQueue

!insert (robotA,queueA) into queue
!insert (robotB,queueB) into queue

!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'B'

!insert (queueB,msg1) into message
```

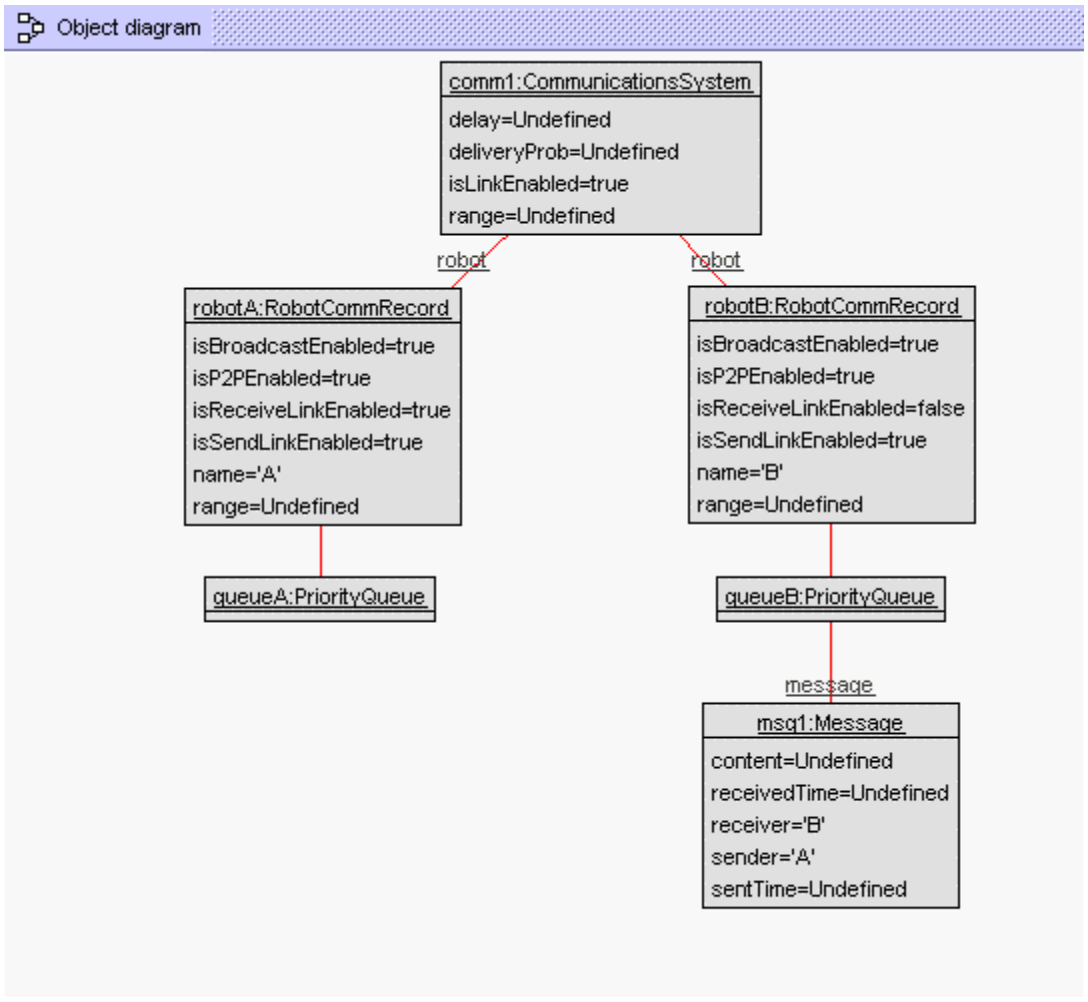

Figure 50. USE Object Diagram – ReceiveAbility Constraint

9. Messages are kept into the right queue.

Scenario: RobotC's priorityQueue has a message which belongs to A.

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true

!create robotA:RobotCommRecord
!create robotB:RobotCommRecord
!create robotC:RobotCommRecord

!set robotA.name = 'A'
!set robotB.name = 'B'
!set robotC.name = 'C'
!set robotA.isBroadcastEnabled = true
!set robotA.isP2PEnabled = true
!set robotA.isSendLinkEnabled = true
!set robotA.isReceiveLinkEnabled = true

!set robotB.isBroadcastEnabled = true
!set robotB.isP2PEnabled = true
!set robotB.isSendLinkEnabled = true
!set robotB.isReceiveLinkEnabled = true

!set robotC.isBroadcastEnabled = true
!set robotC.isP2PEnabled = true
!set robotC.isSendLinkEnabled = true
!set robotC.isReceiveLinkEnabled = true

!insert (comm1,robotA) into robot
!insert (comm1,robotB) into robot
!insert (comm1,robotC) into robot

!create queueA:PriorityQueue
!create queueB:PriorityQueue
!create queueC:PriorityQueue

!insert (robotA,queueA) into queue
!insert (robotB,queueB) into queue
!insert (robotC,queueC) into queue

!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'B'

!insert (queueC,msg1) into message
```

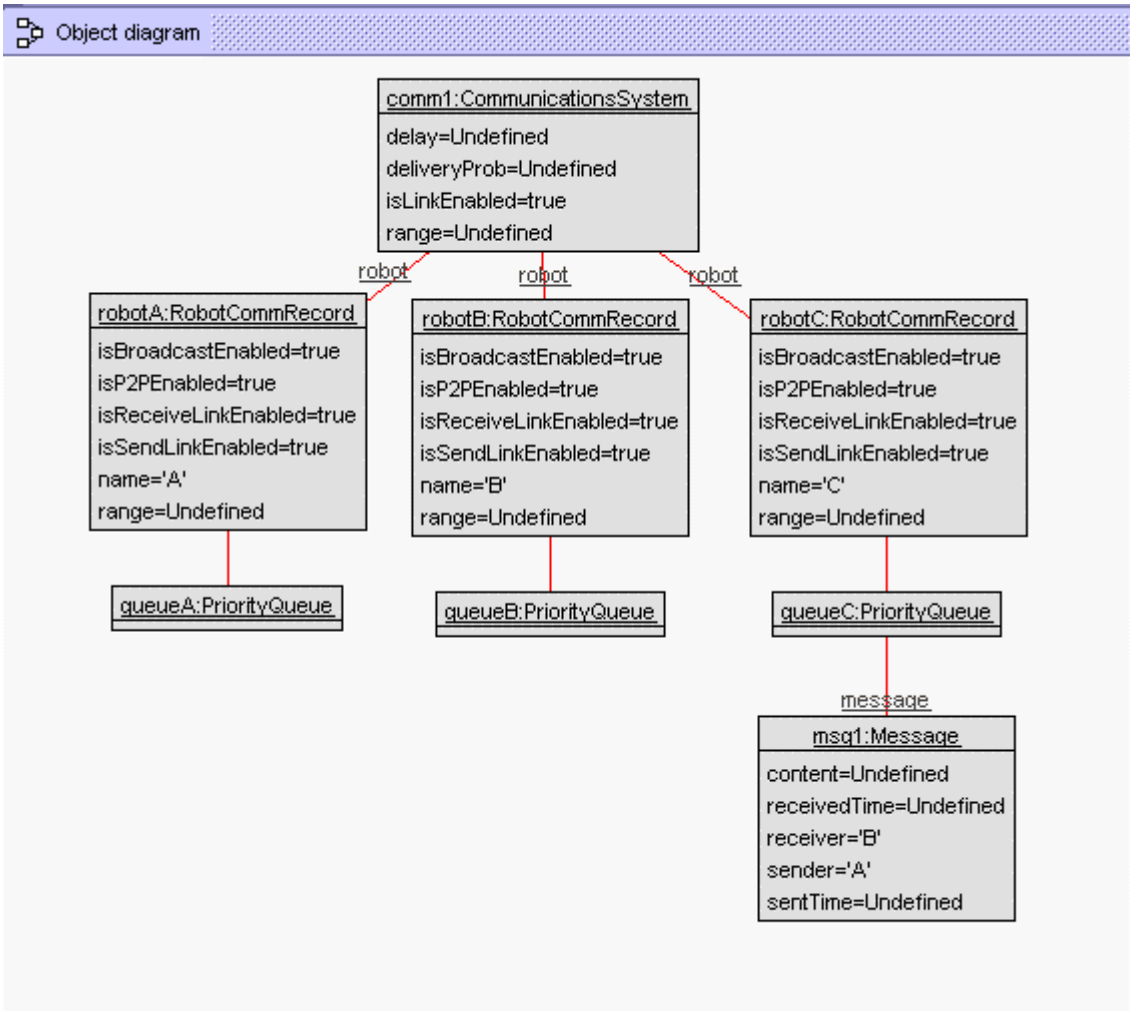

Figure 51. USE Object Diagram – RightQueue Constraint

10.Messages are kept in priority queue ordered by received time.

Scenario: QueueB has msg2, which received time is 2,located before msg1, which received time is 1

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true

!create robotA:RobotCommRecord
!create robotB:RobotCommRecord
!create robotC:RobotCommRecord

!set robotA.name = 'A'
!set robotB.name = 'B'
!set robotC.name = 'C'
!set robotA.isBroadcastEnabled = true
!set robotA.isP2PEnabled = true
!set robotA.isSendLinkEnabled = true
```

```
!set robotA.isReceiveLinkEnabled = true

!set robotB.isBroadcastEnabled = true
!set robotB.isP2PEnabled = true
!set robotB.isSendLinkEnabled = true
!set robotB.isReceiveLinkEnabled = true

!set robotC.isBroadcastEnabled = true
!set robotC.isP2PEnabled = true
!set robotC.isSendLinkEnabled = true
!set robotC.isReceiveLinkEnabled = true

!insert (comm1,robotA) into robot
!insert (comm1,robotB) into robot
!insert (comm1,robotC) into robot

!create queueA:PriorityQueue
!create queueB:PriorityQueue
!create queueC:PriorityQueue

!insert (robotA,queueA) into queue
!insert (robotB,queueB) into queue
!insert (robotC,queueC) into queue

!create msg1:Message
!create msg2:Message

!set msg1.sender = 'A'
!set msg1.receiver = 'B'
!set msg1.sentTime = 1
!set msg1.receivedTime = 1
!set msg2.sender = 'C'
!set msg2.receiver = 'B'
!set msg2.sentTime = 2
!set msg2.receivedTime = 2

!insert (queueB,msg2) into message
!insert (queueB,msg1) into message
```

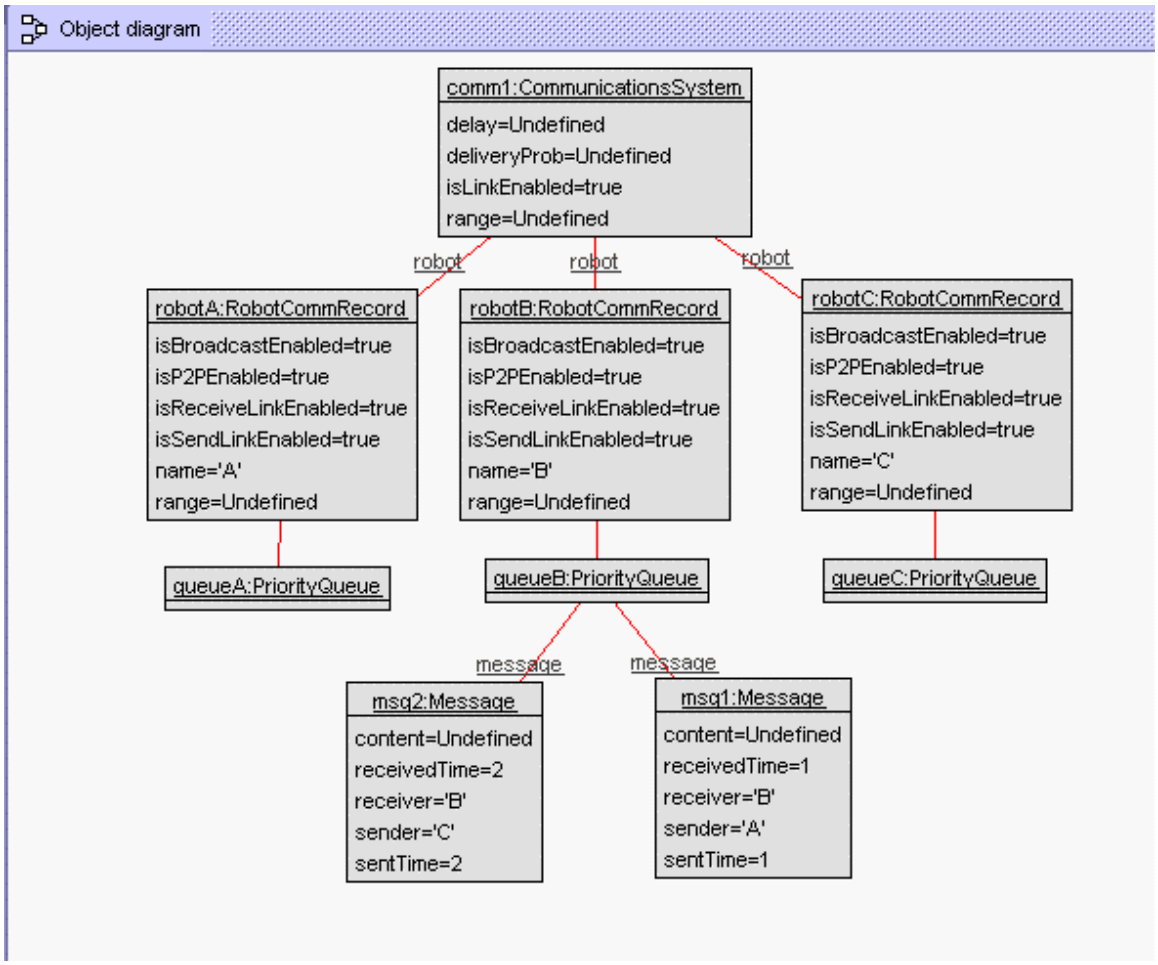

Figure 52. USE Object Diagram – PriotityQueue Constraint

11. If message's received time is defined, then received time is equal or greater than sent time.

Scenario: msg1 has received time equals to 1 which is less then sent time which is 2

```
!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'broadcast'
!set msg1.sentTime = 2
!set msg1.receivedTime = 1
```

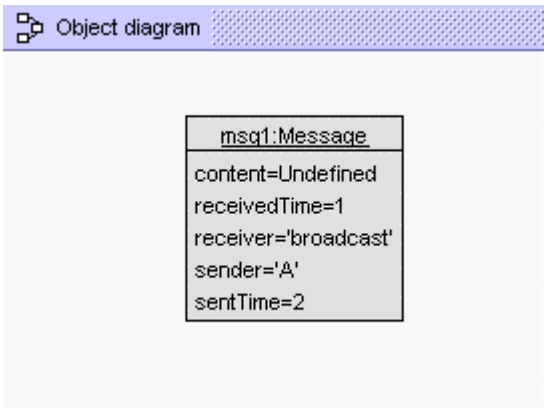

Figure 53. USE Object Diagram – RightTime Constraint

12.If all links are shutdown, robot cannot send or receive message.

Scenario: All links in the system is shutdown, but RobotB received a message.

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = false

!create robotA:RobotCommRecord
!create robotB:RobotCommRecord

!set robotA.name = 'A'
!set robotB.name = 'B'
!set robotA.isBroadcastEnabled = true
!set robotA.isP2PEnabled = true
!set robotA.isSendLinkEnabled = true
!set robotA.isReceiveLinkEnabled = true

!set robotB.isBroadcastEnabled = true
!set robotB.isP2PEnabled = true
!set robotB.isSendLinkEnabled = true
!set robotB.isReceiveLinkEnabled = true

!insert (comm1,robotA) into robot
!insert (comm1,robotB) into robot

!create queueA:PriorityQueue
!create queueB:PriorityQueue

!insert (robotA,queueA) into queue
!insert (robotB,queueB) into queue

!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'B'

!insert (queueB,msg1) into message
```

Figure 54. USE Object Diagram – AllLinkShutdown Constraint

13. Robots cannot receive their own sending message.

Scenario: RobotA received a message, which is sent by him.

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true

!create robotA:RobotCommRecord
!create robotB:RobotCommRecord

!set robotA.name = 'A'
!set robotB.name = 'B'
!set robotA.isBroadcastEnabled = true
!set robotA.isP2PEnabled = true
!set robotA.isSendLinkEnabled = true
!set robotA.isReceiveLinkEnabled = true
```

```
!set robotB.isBroadcastEnabled = true
!set robotB.isP2PEnabled = true
!set robotB.isSendLinkEnabled = true
!set robotB.isReceiveLinkEnabled = true

!insert (comm1,robotA) into robot
!insert (comm1,robotB) into robot

!create queueA:PriorityQueue
!create queueB:PriorityQueue

!insert (robotA,queueA) into queue
!insert (robotB,queueB) into queue

!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'broadcast'

!insert (queueA,msg1) into message
```

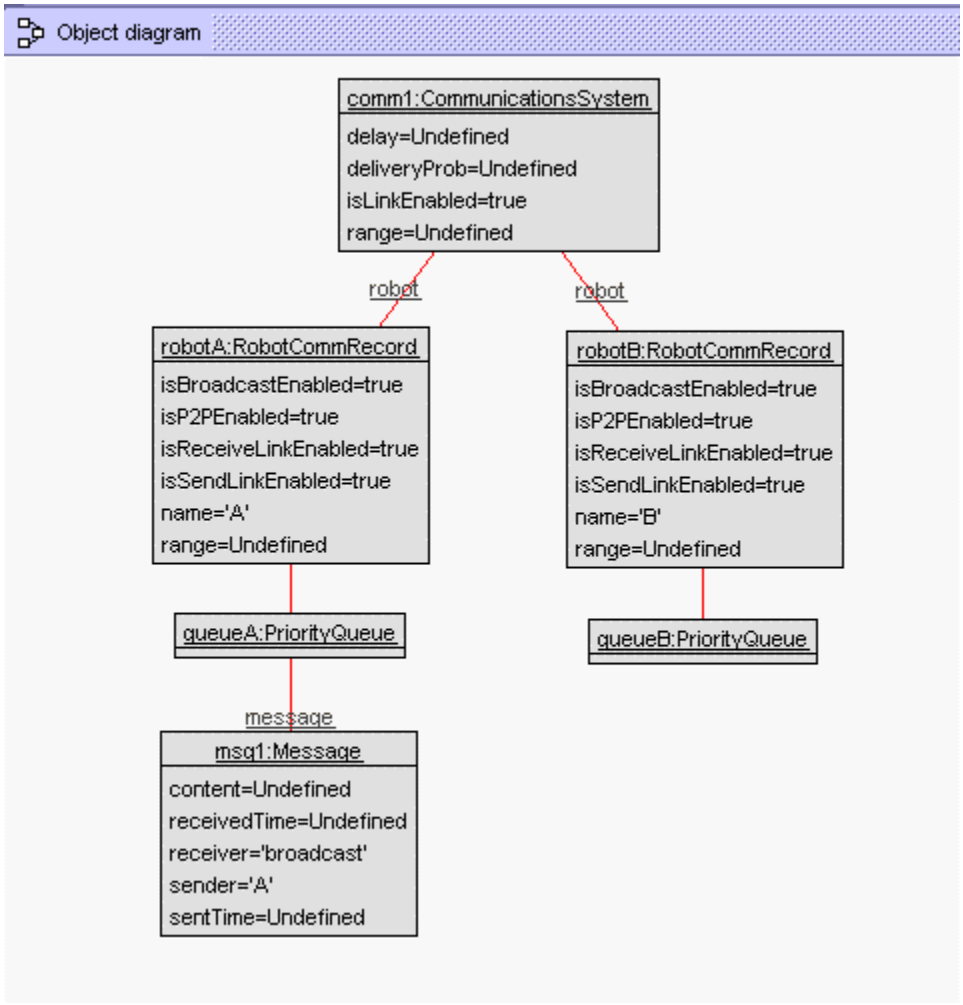

Figure 55. USE Object Diagram – SendToYourSelf Constraint

14.registerRobot Operation

Scenario: The script adds new Robot named A which already exists in the system. This script violates post condition 1,2,3 and 5.

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true
!create RobotA:RobotCommRecord
!set RobotA.name ='A'
!insert (comm1,RobotA) into robot
!create queueA:PriorityQueue
!insert (RobotA,queueA) into queue
!openter comm1 registerRobot('A',2)

!create RobotB:RobotCommRecord
!set RobotB.name = 'A'
!set RobotB.isP2PEnabled = true
!set RobotB.isBroadcastEnabled = false
!insert (comm1,RobotB) into robot
!create queueB:PriorityQueue
!insert (RobotB,queueB) into queue

!opexit
```

15.SendMessage operation

Scenario: There is no message added to RobotB's priorityQueue after message is sent. This script violates post condition 5.

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true
!set comm1.delay = 2
!create RobotA:RobotCommRecord
!set RobotA.name ='A'
!set RobotA.isP2PEnabled = true
!set RobotA.isBroadcastEnabled = true
!set RobotA.isSendLinkEnabled = true
!set RobotA.isReceiveLinkEnabled = true
!insert (comm1,RobotA) into robot
!create queueA:PriorityQueue
!insert (RobotA,queueA) into queue
!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'B'

!create Param1:RobotParameter
!set Param1.receiverName = 'B'
!set Param1.delayTime = 1
!insert (RobotA,Param1) into parameter

!create RobotB:RobotCommRecord
!set RobotB.name = 'B'
!set RobotB.isP2PEnabled = true
!set RobotB.isBroadcastEnabled = false
```

```
!set RobotB.isSendLinkEnabled = true
!set RobotB.isReceiveLinkEnabled = true
!insert (comm1,RobotB) into robot
!create queueB:PriorityQueue
!insert (RobotB,queueB) into queue

!create Param2:RobotParameter
!set Param2.receiverName = 'A'
!set Param2.delayTime = 1
!insert (RobotB,Param2) into parameter

!openter comm1 sendMessage(msg1,1)
!set msg1.sentTime = 1
!set msg1.receivedTime = 4

!opexit
```

16.GetMessage Operation

Scenario1: There is no message removed from RobotB's priorityQueue after calling getMessage operation. (It is supposed to remove the messages which received time is equal to current time step. For this case, message msg1 and msg2 should be removed and the result should return msg1 and msg2). This script violates post condition 1.

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true
!create RobotA:RobotCommRecord
!set RobotA.name ='A'
!set RobotA.isP2PEnabled = true
!set RobotA.isBroadcastEnabled = true
!set RobotA.isSendLinkEnabled = true
!set RobotA.isReceiveLinkEnabled = true
!insert (comm1,RobotA) into robot
!create queueA:PriorityQueue
!insert (RobotA,queueA) into queue
!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'B'

!create RobotB:RobotCommRecord
!set RobotB.name = 'B'
!set RobotB.isP2PEnabled = true
!set RobotB.isBroadcastEnabled = true
!set RobotB.isSendLinkEnabled = true
!set RobotB.isReceiveLinkEnabled = true
!insert (comm1,RobotB) into robot
!create queueB:PriorityQueue
!insert (RobotB,queueB) into queue
!set msg1.sentTime = 1
!set msg1.receivedTime = 1

!create msg2:Message
!set msg2.sender = 'A'
!set msg2.receiver = 'B'
```

```
!set msg2.sentTime = 1
!set msg2.receivedTime = 1
!insert (queueB,msg2) into message
!insert (queueB,msg1) into message

!openter comm1 getMessage('B',1)
!opexit Set{msg1,msg2}
```

Scenario2: There is only one message which received time is 1 in RobotB's queue, but the result returns two messages, msg1 and msg2. This script violates post condition 2.

```
!create comm1:CommunicationsSystem
!set comm1.isLinkEnabled = true
!create RobotA:RobotCommRecord
!set RobotA.name ='A'
!set RobotA.isP2PEnabled = true
!set RobotA.isBroadcastEnabled = true
!set RobotA.isSendLinkEnabled = true
!set RobotA.isReceiveLinkEnabled = true
!insert (comm1,RobotA) into robot
!create queueA:PriorityQueue
!insert (RobotA,queueA) into queue
!create msg1:Message
!set msg1.sender = 'A'
!set msg1.receiver = 'B'


!create RobotB:RobotCommRecord
!set RobotB.name = 'B'
!set RobotB.isP2PEnabled = true
!set RobotB.isBroadcastEnabled = true
!set RobotB.isSendLinkEnabled = true
!set RobotB.isReceiveLinkEnabled = true
!insert (comm1,RobotB) into robot
!create queueB:PriorityQueue
!insert (RobotB,queueB) into queue

!set msg1.sentTime = 2
!set msg1.receivedTime = 2


!create msg2:Message
!set msg2.sender = 'A'
!set msg2.receiver = 'B'
!set msg2.sentTime = 1
!set msg2.receivedTime = 1
!insert (queueB,msg2) into message
!insert (queueB,msg1) into message

!openter comm1 getMessage('B',1)
!delete (queueB,msg2) from message
!opexit Set{msg1,msg2}
```

APPENDIX C

FORMAL INSPECTION CHECKLIST

- **Formal Inspection Checklist – Kevin Sung**

Inspection list	Pass/Fail/Partial	Comment
1. The symbols using in use case diagram conform to UML diagram.	Pass	
2. The symbols using in class case diagram conform to UML diagram.	Pass	
3. The symbols using in sequence diagram conform to UML diagram.	Pass	
4. Use case diagram and descriptions are clear and well organized.	Pass	
5. Class diagram and descriptions are clear and well organized.	Pass	
6. Each message passing in sequence diagram is the method in class diagram.	Pass	
7. Each message passing in sequence diagram must be defined as public method.	Pass	
8. Class names are well defined and indicate their meaning	Pass	
9. The architecture design covers the entire requirement defined in Software Requirement Specification.	Pass	

Table 7. Formal Technical Inspection Checklist - Kevin

- **Formal Inspection Checklist – Estaban Guillen**

Inspection list	Pass/Fail/Partial	Comment
1. The symbols using in use case diagram conform to UML diagram.	Pass	It looks good
2. The symbols using in class case diagram conform to UML diagram.	Pass	It looks good
3. The symbols using in sequence diagram conform to UML diagram.	Pass	It looks good
4. Use case diagram and descriptions are clear and well organized.	Pass	Yes it was clear to me
5. Class diagram and descriptions are clear and well organized.	Pass	I had a few questions. I didn't see a constructor for the RobotParameter class and was wondering how it would get created. Also in the RobotParamater class there are four "get" methods but there are no return types shown.
6. Each message passing in sequence diagram is the method in class diagram.	Pass	Yes it matches up
7. Each message passing in sequence diagram must be defined as public method.	Pass	Yes
8. Class names are well defined and indicate their meaning	Pass	Yes
9. The architecture design covers the entire requirement defined in Software Requirement Specification.	Pass	Yes nice work. I just had a question about section 4.3 in your Architecture Design doc. The sequence diagram looks good, but the description above talks about sending broadcast messages while the diagram is showing point-to-point.

Table 8. Formal Technical Inspection Checklist - Esteban

APPENDIX D

FORMAL INSPECTION LETTER

To Whom It May Concern:

I, Kevin Sung, have checked and confirm the documentation have fulfilled the criteria that is listed in the Formal Inspection Checklist

Kevin Sung

Dr. Hsu and Committees:

I have completed the Formal Inspection Checklist for the Architecture design of the Communication Model for Cooperative Robotics Simulator Project. I have not found any critical problems and all items in the checklist are passed.

Esteban Guillen

REFERENCES

Lee, Richard C., Tepfenhart, William M., *Practical Object-Oriented Development With UML and Java*, Prentice Hall, 2002.

Warmer, Jos B., Kleppe, Anneke G., *The Object Constraint Language Precise Modeling with UML*, Addison-Wesley, 1998.

Royce, Walker. *Software Project Management A unified framework*, 1st ed. Addison-Wesley, 1998.

Pressman, Roger. *Software Engineering a practitioner's approach*, 5th ed. McGraw-Hill international edition, 2001.

USE manual, University of Bremen (http:// www.db.informatik.uni-bremen.de/project/USE

"Cooperative Robotics Simulator" project overview description http://www.cis.ksu.edu/~sdeloach/CRSimulator/downloads/Cooperative%20Robotic%20Simulator.pdf

IEEE Guide for Software Quality Assurance Planning Std 730.1-1995.

IEEE Standard for Software Quality Assurance Plans Std 730-1998.

IEEE Recommended Practice for Software Requirements Specification IEEE Std 830-1998