Communication Model for Cooperative Robotics Simulator

Project Evaluation

Version 1.0

---

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 8, 2004

Table 1. Expected and actual finish time in 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.

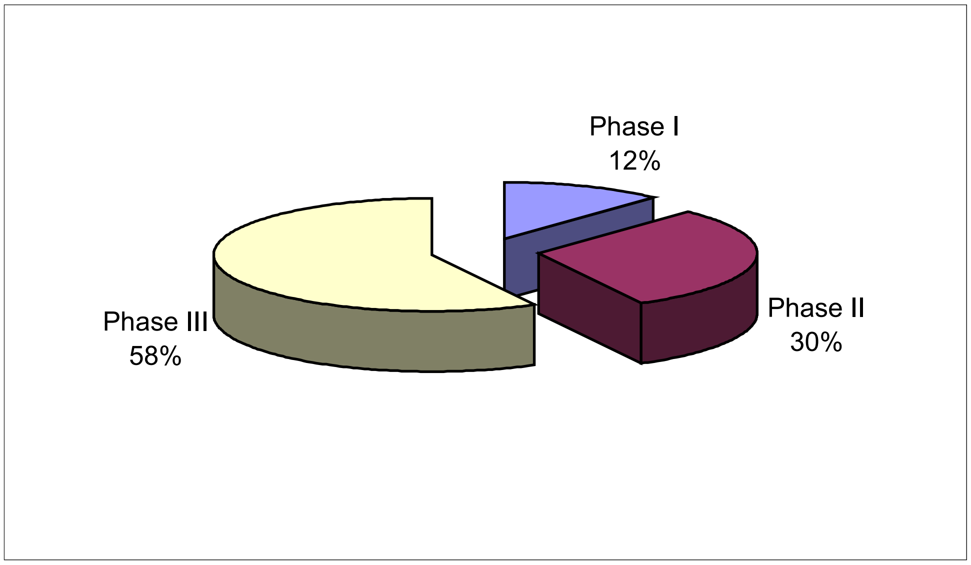


Figure 1. PhaseTime Breakdown

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

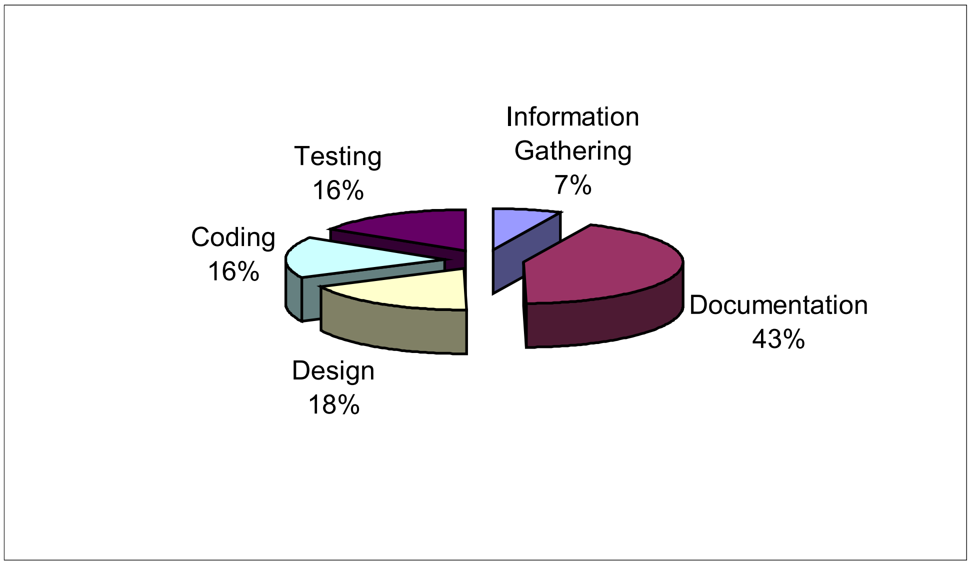


Figure 2. Project Work Breakdown

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

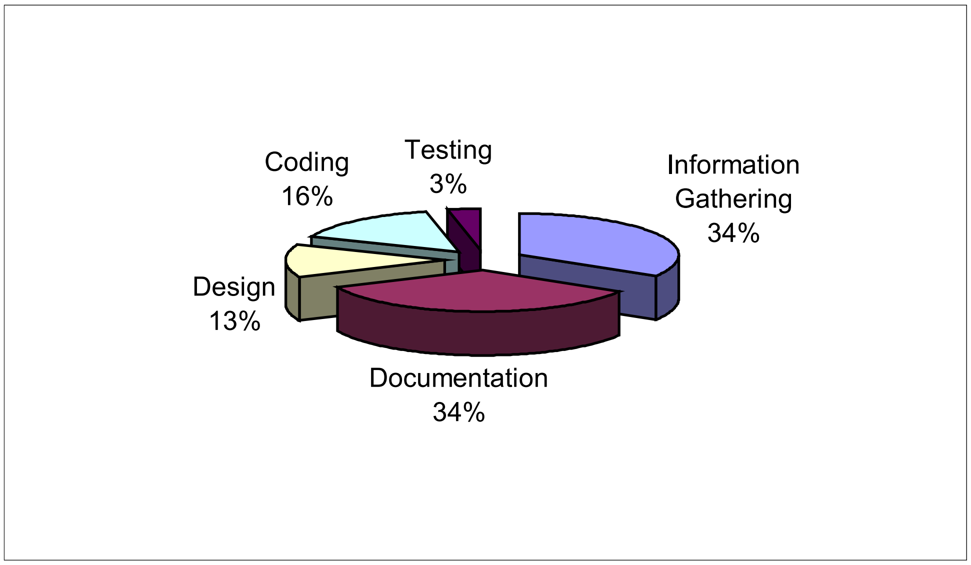


Figure 3. Phase I Work Breakdown diagram.

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

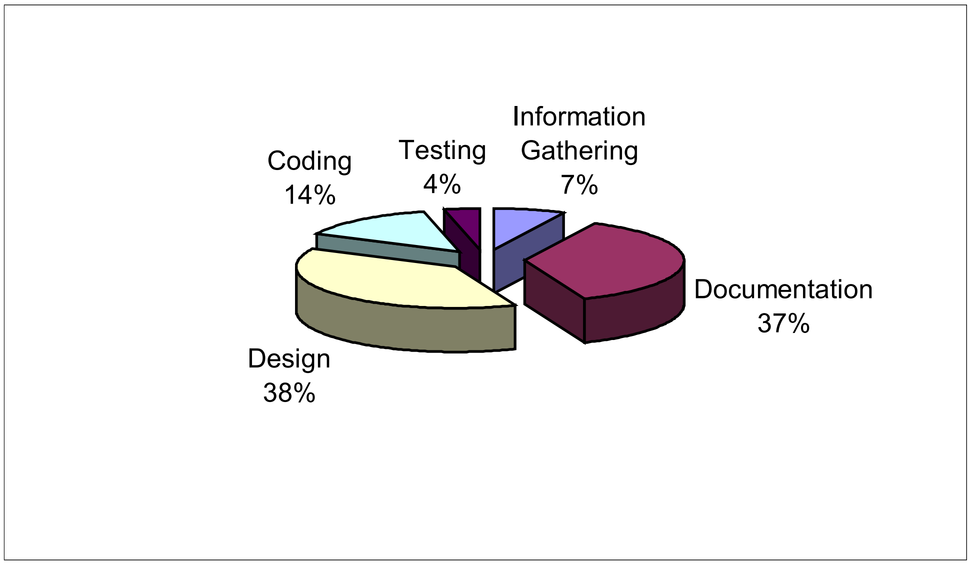


Figure 4. Phase II Work Breakdown diagram.

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

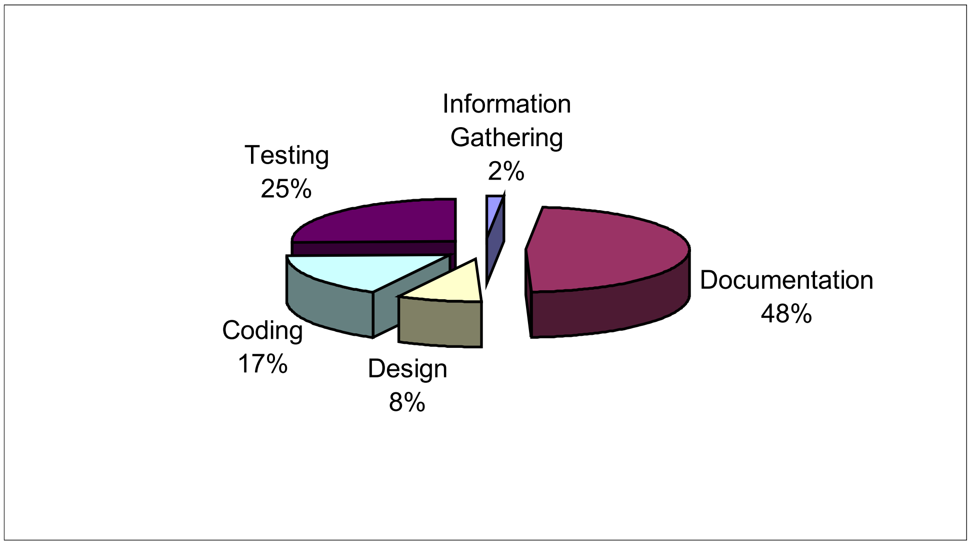


Figure 5. Phase III Work Breakdown diagram.

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.

5. 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.

5.1 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.

5.2 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.

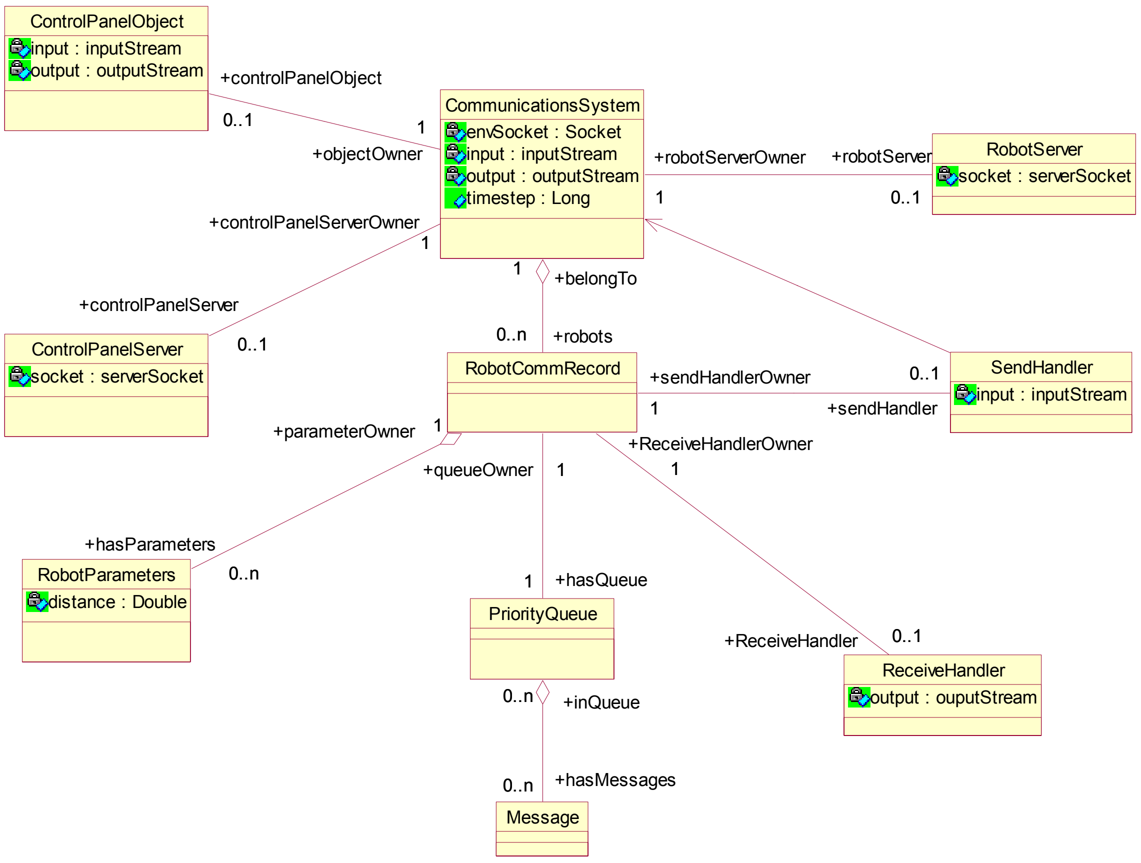


Figure 6. Redesigned - Communication Model Class Diagram

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.

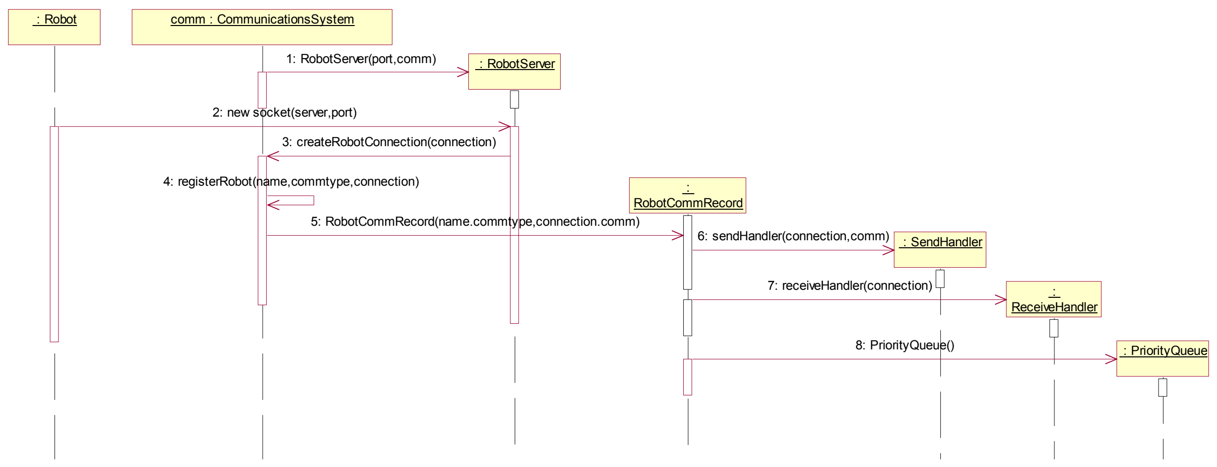

: Robot
comm : CommunicationsSystem
1: RobotServer(port,comm)
: RobotServer
2: new socket(server,port)
3: createRobotConnection(connection)
4: registerRobot(name,commtype,connection)
: RobotCommRecord
5: RobotCommRecord(name.commtype,connection.comm)
6: sendHandler(connection,comm)
: SendHandler
7: receiveHandler(connection)
: ReceiveHandler
8: PriorityQueue()
: PriorityQueue

Figure 7. 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.

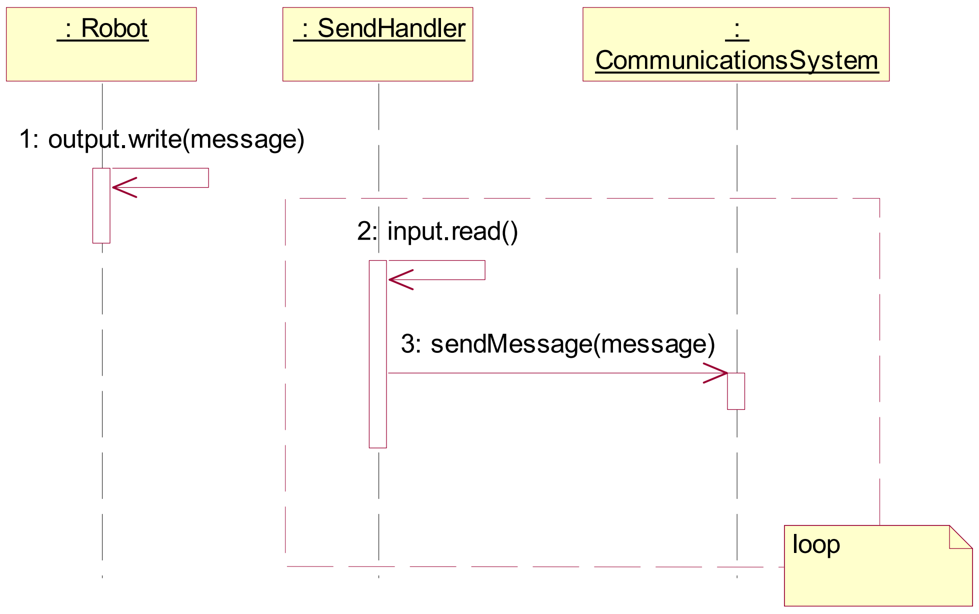


Figure 8. Redesigned-Send Message Sequence Diagram

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.

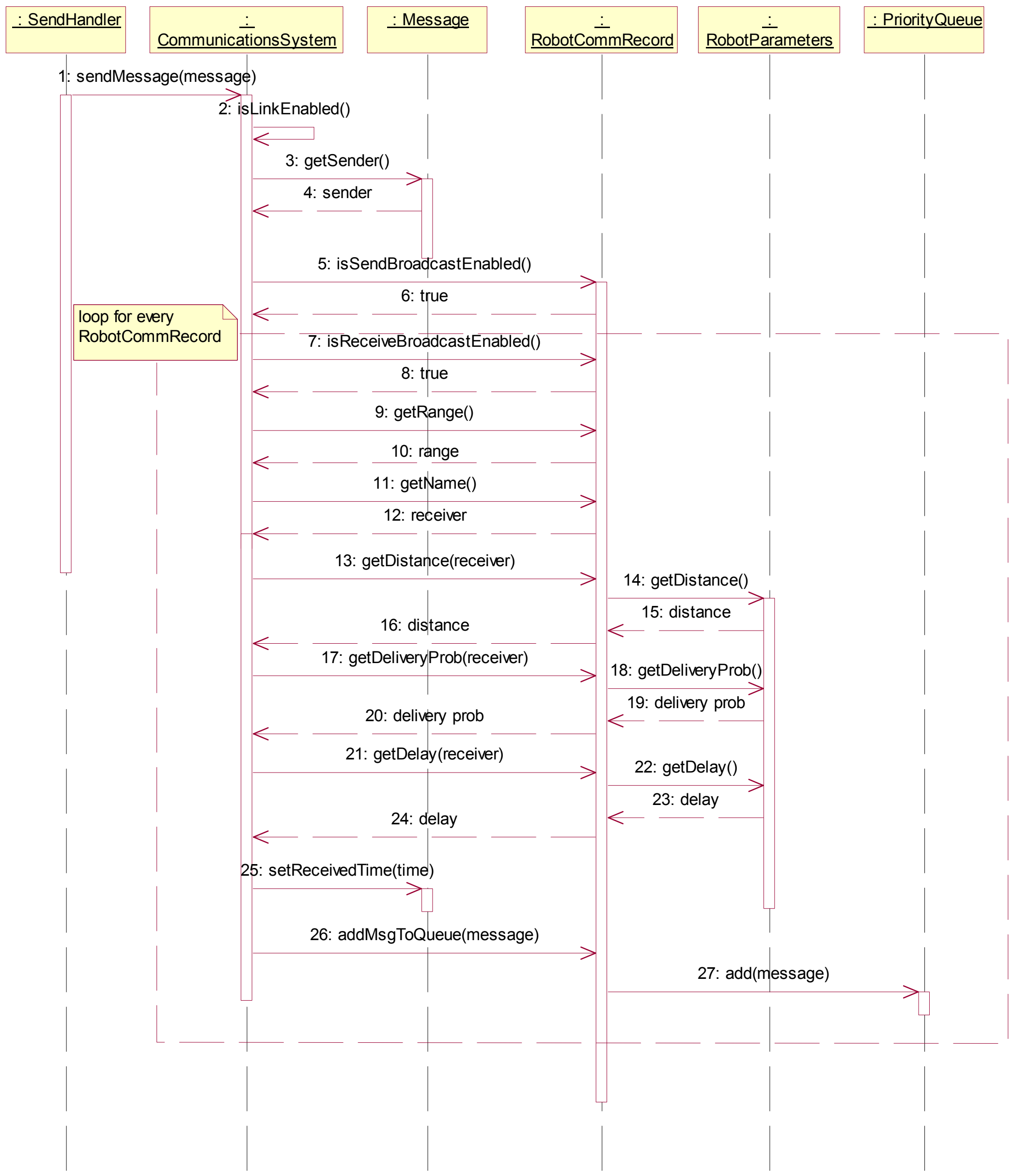


Figure 9. Redesigned - CommunicationsSystem Send Broadcast Message Sequence Diagram

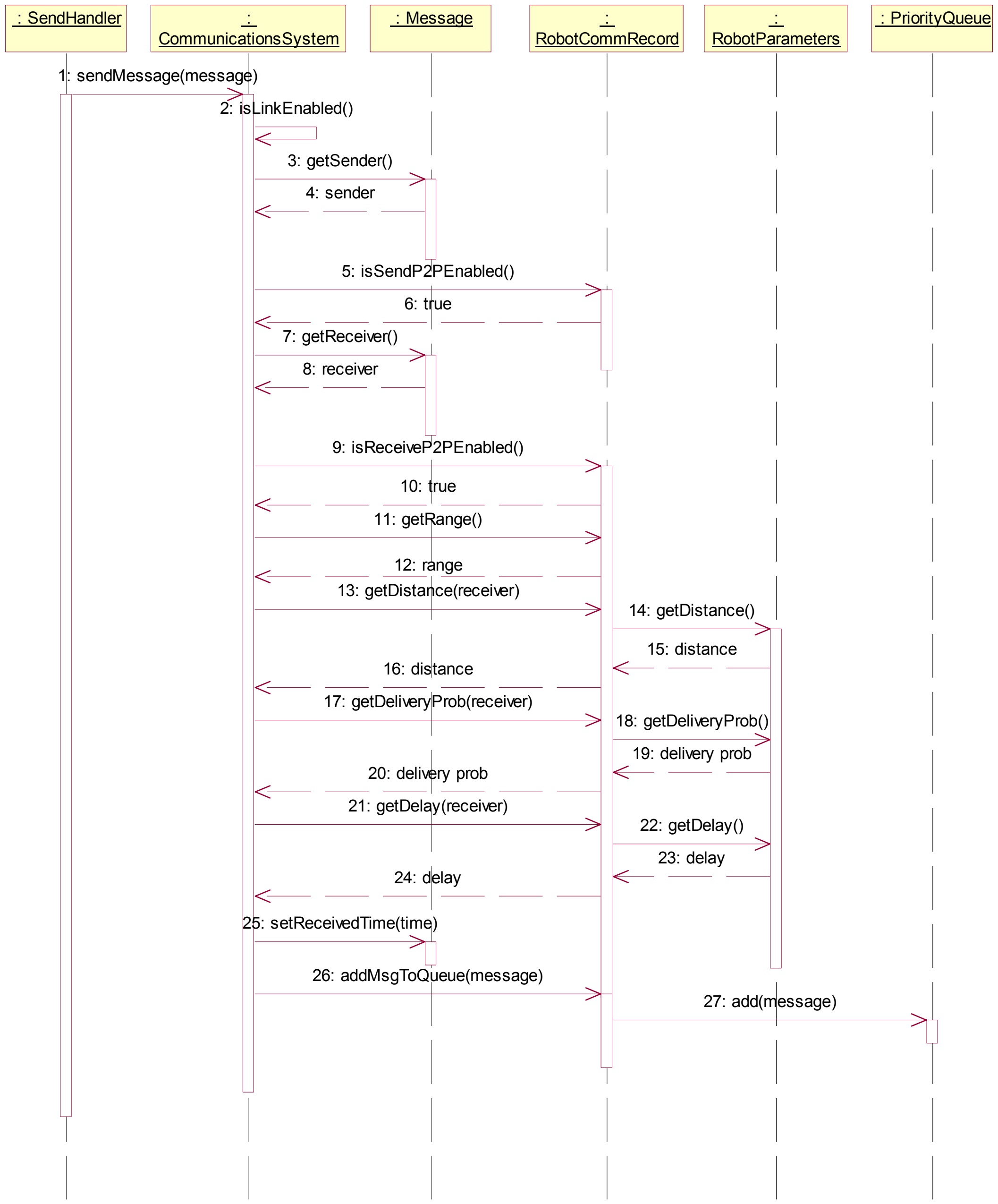


Figure 10. Redesigned - CommunicationsSystem Send Point-to-Point Message Sequence Diagram

- **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.

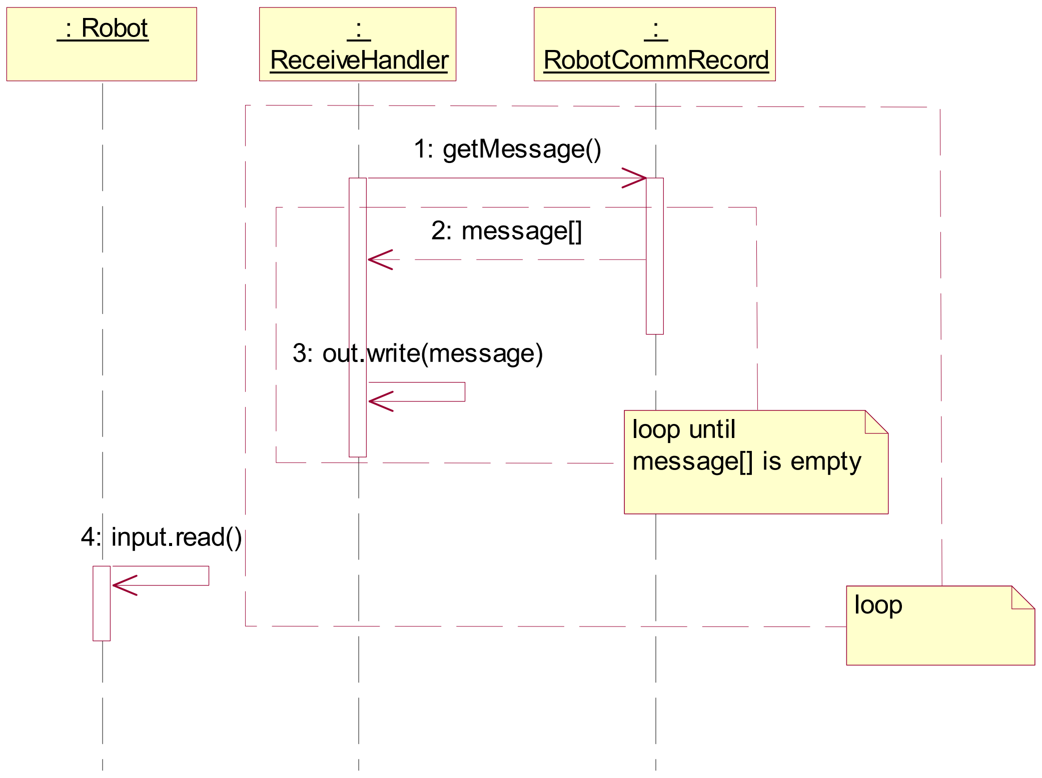
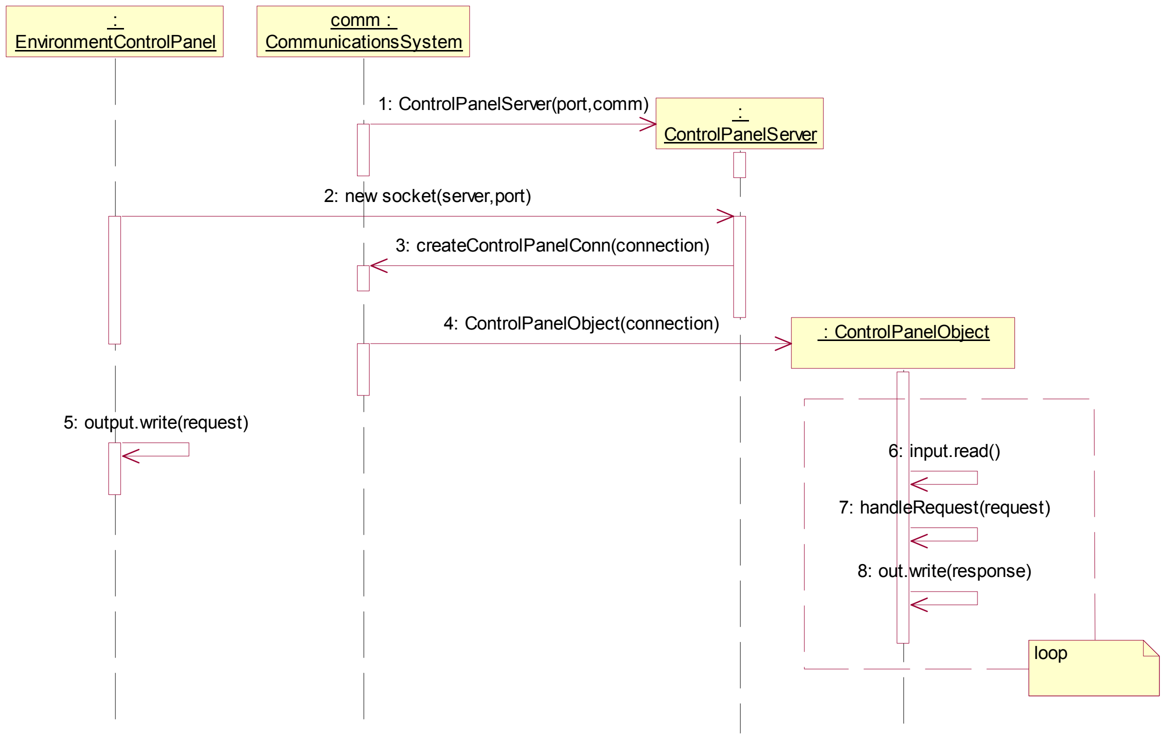


Figure 11. Redesigned- Receive Message Sequence Diagram

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.



: EnvironmentControlPanel
comm : CommunicationsSystem
: ControlPanelServer
1: ControlPanelServer(port,comm)
2: new socket(server,port)
3: createControlPanelConn(connection)
4: ControlPanelObject(connection)
: ControlPanelObject
5: output.write(request)
6: input.read()
7: handleRequest(request)
8: out.write(response)
loop

Figure 12. 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.

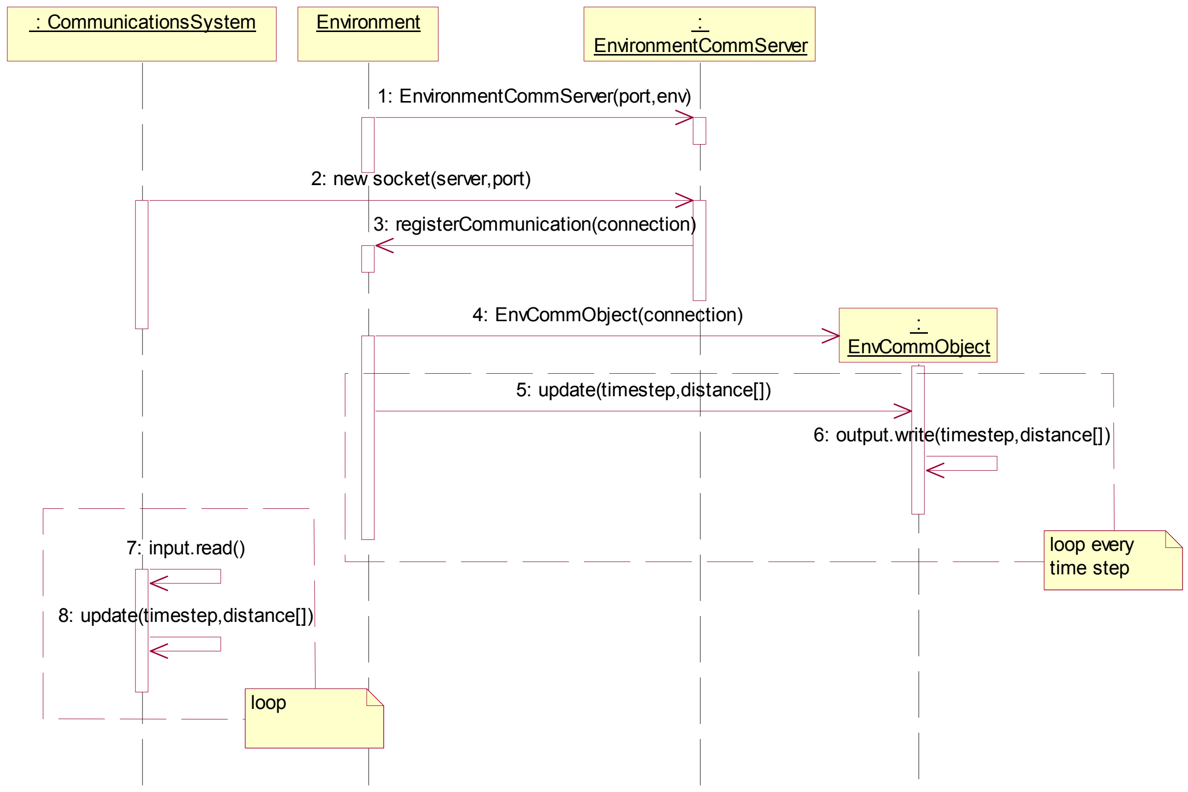

: CommunicationsSystem
Environment
: EnvironmentCommServer
1: EnvironmentCommServer(port,env)
2: new socket(server,port)
3: registerCommunication(connection)
4: EnvCommObject(connection)
: EnvCommObject
5: update(timestep,distance[])
6: output.write(timestep,distance[])
loop every time step
7: input.read()
8: update(timestep,distance[])
loop

Figure 13. Redesigned-Get Time Step and Distance Sequence Diagram