# Lab 6: Mindstorm Robot High Level Design
# Due: April 24/26, 2012 23:00 CDT

## 1. Key Lab Outcomes

- Design the classes necessary to control your Lego Robot
- Design software using the Observer pattern
- Construct a simple program which will drive the Robot car through the Proxy and robot software

## 2. Due Dates

| Deliverable | Responsible Part | Due Date |
|---|---|---|
| Submission of design report via SVN (word format and pdf format) | Team Leader | April 24 / 26, 2012 |
| Workplan is updated and checked into the SVN repository | Planning Manager | April 24 / 26, 2012 |
| Design model is checked into SVN | Development Manager | April 24 / 26, 2012 |
| Robot source code is checked into SVN for demo | Quality Manager | April 24 / 26, 2012 |
| Lead Implementation of prototype Mindstorm robot code. | Planning Manager | April 25 / 27, 2012 |
| Demonstrate the operation of the Lego Mindstorm Robot | Quality Manager | April 25 / 27, 2012 |
| Begin sketching design of GUI / begin prototyping the GUI | Support Manager | April 25 / 27, 2012 |

## 3. Introduction

Thusfar, we have analyzed the requirements for the Lego Mindstorm, as well as created a domain model which relates the pieces that are part of the system. In this week's lab, you will perform the design process. This design process will result in a completed design for your Mindstorm Robot. This design can then be used for implementation.

## 4. Final project requirements

For the final Mindstorm Robot deliverable, your car must be capable of performing the following operations

1. Controlling the robotic vehicle using the developed software (steering, speed, etc.)
2. Making the vehicle automatically stop if an obstruction is encountered when driving
3. Making an audible noise on the robot when the robot operates in reverse
4. Operating the grappling arm

5. Storing and displaying the total distance the car has traveled in feet to the nearest 1/10 of a foot.[1]
6. Displaying the distance traveled since the last rest of the trip odometer to the nearest 1/10 of a foot.
7. Displaying the compass direction of the vehicle on the GUI display
8. One other feature of your team's desire, either from the SRS from your own imagination. If it is from your imagination, you will need to develop at least a high level use case scenario for the feature.

# 5. Lab process

## *5.1. Update the work plan*

To begin with, the Process manager and all team members should determine how the work for the next week is to be divided. From that, the workplan should be developed with team members being assigned to tasks and providing estimated effort.

Prior to the completion of this lab, the quality manager should update the workplan with the actual effort expended on the lab project.

## *5.2. Subsystem Development*

The first step of this lab is to partition the design into pieces. Up to this point, the discussion has centered on the domain model. With the domain model, you were able to determine the pieces that exist within your project. But this may or may not have bearing on your software.

To start, you will need to create a fundamental architecture. For the purpose of this lab, this involves the creation of subsystems, packages, and tasks. As this is a relatively simple system, this effort may be small.

Essentially, from the domain model, you want to determine logically similar items and place them in packages. By doing this, a person can be responsible for a single package. While the exact process is to be left up to the team, the easiest method is to start with a hardcopy of the domain diagram. Indicate which package each item is going to live within right on this sheet. This can then easily be transferred into the EA model. When this step is completed, you will have a subsystem view of your Mindstorm Robot software, similar to that shown in Figure 8-8 of the textbook.

You may also do this step by sketching on the whiteboard.

[1] On fundamental thing you will need to determine is where the distance traveled is going to be stored. You can store this distance either on the PC or on the robot. One may be easier than the other, but may also be more problematic in the long run.

## 5.3. Detailed Behavioral Analysis

Once you have developed the subsystem model, it is now time to analyze the behavior which must be exhibited by each of the subsystem. There are many places in which state behavior may need to be captured. For each of these states, create a state machine defining the behavior. While starting by hand may make the most sense, you eventually will need to input the state machine into Enterprise Architect. State machines may use nested states, or they may be flat in their implementation. The detail is up to you, as is how they are structured.

For this step, you will probably want to draw the state machines on the whiteboard before attempting to use the design tools.

## 5.4. Detailed Interaction Analysis

Once you have completed the analysis of the behavior, you will need to define the interactions between subsystems. In essence, each of the subsystems will have an interface, and the interactions must go through the designed interface. The interactions will be expressed using sequence diagrams. Messages represent communications between elements.

For your high level design, you will need to model the interactions necessary for the system to operate. For each use case, create one or more sequence diagrams indicating the interaction between elements. Essentially, there should be one sequence diagram for each major use case that your system is to have. The interfaces will effectively be the Java interfaces used for your system. In fact, if your model is created properly, you can auto generate the Java interfaces from the UML models.

You may want to start this phase by drawing on the whiteboard prior to trying to draw the diagrams with the tool.

As a side note, you are required to use the Observer pattern at least twice in your design. It is recommended that this usage be somewhere within the user interface, though this may not be the only place where it makes sense to do so. The usage of this pattern may impact your interactions and what you need to develop.

## 5.5. Vehicular Control Proof of Concept

One other goal of this set of lab sessions is to demonstrate the ability to control the Robot in a simple fashion. In short, the support manager (or designee) should implement a Java program on the PC which will send commands through the proxy to cause the vehicle to perform the following operations:

1. Drive forward for 1 second
2. Stop for one second
3. Drive backward for one second
4. Stop for one second
5. Drive forward and to the right for one second

6. Stop for one second
7. Drive backward and to the right for one second
8. Stop for one second
9. Drive forward and to the left for one second.
10. Stop for one second
11. Drive backward and to the left for one second.
12. Stop for one second
13. Repeat.

In order to do this, it is necessary to establish an interface between the PC and the robot. The proxy, as it exists, only allows the transmission of integers back and forth. While this limits the potential designs, it is not an uncommon limitation of network interfaces. One was to overcome this problem is to treat the integer as an unsigned number and essentially split it in half. The upper two bytes would indicate the command or report to or from the robot and the lower two bytes would indicate the specific details of the command. For example, you might define the constants

public static final int STEER_COMMAND = 0x0100
public static final int STEER_WHEELS_STRAIGHT = 0x0000
public static final int STEER_WHEELS_LEFT = 0x0001
public static final int STEER_WHEELS_RIGHT = 0x0002

in the file RobotControlInterface.java. A copy of this file would be placed in the project which runs on the robot and a copy would be placed in the host project. To send a command to turn the vehicle wheels to the left you would simply use the following code:

myProxy.enqueue(RobotControlInterface. STEER_COMMAND | RobotControlInterface .STEER_WHEELS_LEFT);

To pause for 1 second, you may want to look at the Thread.sleep() API call within Java.

When completed, the support manager will now be capable of answering questions about the interface to the device driver and the basics of interacting with the robot and the proxy. This information may prove very beneficial when designing the other pieces of the system.

This program must be demonstrated in Lab on April 25 / 27, 2012.

# 6. Lab Deliverables

## *6.1. Report*

Each team shall submit a mini-report consisting of the following:

1. Title Page

   a. Name of team
   b. Team Members
   c. Submission Title
2. Introduction
   a. Explain, in your own words, what this lab was trying to accomplish
   b. Explain, in your own words, how you went about the creation of the design and how your determined the correct way to
3. Subsystem Diagram
   a. Submit a figure explaining the subsystem layout. For each piece of the subsystem, explain its purpose and what it is responsible for doing.
   b. Explain where your system will use the observer pattern
4. Behavioral Analysis
   a. Submit copies of your state diagrams, explaining their purpose and how they relate to the use case.
   b. For pieces of the system which have other forms of behavior that are not represented in state machines, explain what the behavior of the system is. This may be a mathematical expression or other representation.
5. Detailed Interactions
   a. Submit copies of each sequence diagram as it exists within the system.
   b. For each diagram, provide an explanation and trace its existence back to a use case for the system.
6. Things Gone Right / Things Gone Wrong
   a. Describe briefly the things that went right with this activity.
   b. Describe briefly the problems you had completing this exercise.
7. Conclusions
   a. What has been learned from this exercise?
   b. Has this exercise helped to develop a more cohesive understanding of the problem and how to approach solving the problem?

The lab report also should be checked into your svn archive.

## 6.2. Work plan

The updated work plan showing estimates and effort should be checked into SVN

## 6.3. Design Model

Your design model in EA should be routinely checked into SVN by those who are working on it.

## 6.4. Robot Source Code

Source code for your robot and initial program should be checked into SVN as well.

# 7. Looking Ahead

If time permits, it might also be wise for a developer to begin preliminary work on the GUI system. While it can be developed using Swing components and the material taught in SE1021, you may also use a GUI generation tool. NetBeans offers such a tool, and another available tool is Jigloo (http://cloudgarden.com/jigloo/). These tools allow you to develop portions of the GUI in a WSIWYG fashion. You are not required to do your development this way, but it might be a mechanism that you can use. Your colleagues in the software engineering program have used both of these tools and can offer assistance with them if needed.

# Appendix A: Preliminary Grading Rubric

| | Weight Factor | Rubric Score | |
|---|---|---|---|
| | .25 | | Title Page<br>- Name<br>- Section<br>- Instructor<br>- Date<br>- Assignment |
| | .5 | | Introduction<br>- Full explanation of the scope of the lab assignment<br>- Explanation of how the design was developed<br>- Written in one's own words<br>- Written in multiple complete sentences |
| | 1 | | Subsystem Diagram<br>- Subsystem diagram shows relevant subsystems for the developed software<br>- Subsystems placed in independent packages<br>- Complete system can be developed using the subsystems provided<br>- Subsystems provide a somewhat optimized solution to the problem<br>- |
| | .5 | | Images<br>- Images submitted in report are legible.<br>- Images appropriately labeled and titled |
| | 2 | | Behavioral Analysis<br>- State diagrams submitted for classes which have state behavior<br>- State diagrams related to use cases for the system<br>- All transitions properly labeled.<br>- All state machines cleared designate initial state.<br>- All transitions have triggering events<br>- No state machines have duplicate triggering events leading from the same state without guard conditions. |
| | 2 | | Interaction Diagrams<br>- Interaction Diagrams clearly designate objects which generate and receive messages.<br>- Messages correspond with events on state diagrams when applicable.<br>- Messages specific enough to differentiate unique transitions but not overly specific<br>- Interactions occur between interfaces for associated subsystems<br>- Interactions make sense in the context of the system. |
| | .5 | | TGR-TGW<br>- Things gone right with lab discussed<br>- Problems encountered in project discussed.<br>- Multiple, complete sentences. |
| | 1 | | Work plan<br>- Work plan committed into archive at least twice, once during the initial planning and once at the end of the lab session with actual effort amounts recorded. |

| | Weight Factor | Rubric Score | |
|---|---|---|---|
| | | | - Work plan shows roughly equal contributions to project (at least during estimation) |
| | .5 | | SVN Repository<br>- Report checked into the SVN repository in native word doc or docx format.<br>- All submissions have meaningful submission comments. |
| | 1 | | Conclusions filled out.<br>- What was learned from this experience? |
| | 1 | | Robot Demonstration<br>- Demonstrate the ability to control the car using the device driver |
| | 1 | | Individual Contribution<br>- Individual contribution to the project appropriate<br>- Individual shows tasks both in the workplan as well as demonstrates completion of work in SVN. |
| | .25 | | Report generally grammatically correct<br>-Written in 3rd person<br>-No usage of I, we, etc.<br>-No significant spelling or grammar errors.<br>-Complete sentences with clear meaning. |