Mar 22, 2013

Team 19 Design Document

Laboratory # 5: Design

Morgan, Laura
Miaw, Jireh
Hauser, Steven
Dworak, Catherine
Bertoglio, David

Work Product

Description of the design of the robot on-board software, including high level description, UML class and sequence diagrams, state diagram, concurrent structure, and class interfaces in Java

Document Revision Information

3/22/2013 – Design Document Created

Approval Sheet

All group members whose names are listed below approve of the document and contributed fairly.

Morgan, Laura
Miaw, Jireh
Hauser, Steven
Dworak, Catherine
Bertoglio, David

Pledge

On my honor, as a student, I have neither given nor received unauthorized aid on this assignment.

Morgan, Laura
Miaw, Jireh
Hauser, Steven
Dworak, Catherine
Bertoglio, David

Contents

High-level system architecture

The robot on-board software will be object-oriented. It will consist of 3 classes, Activator, Driver, and MessageHandler. The Activator will contain instances of Driver and MessageHandler. Driver and MessageHandler will not be able to access each others' fields and methods directly; any interaction between Driver and MessageHandler must go through the Activator class.

Activator

The Activator class contains the main method. This class is the only one that deals with the Bluetooth connection. It will contain fields and methods to create the connection and check if the connection is there. It creates 3 threads: timer, read, and output. The timer thread is used to determine how much time has elapsed between sending the last message from the on-board system and receiving an acknowledgment from the base computer. The input and output threads are the channels to send and receive messages from the base computer.

The activator receives messages from the base computer, then sends them to the MessageHandler class for decoding, then channels the usable message to the Driver class to implement the required action.

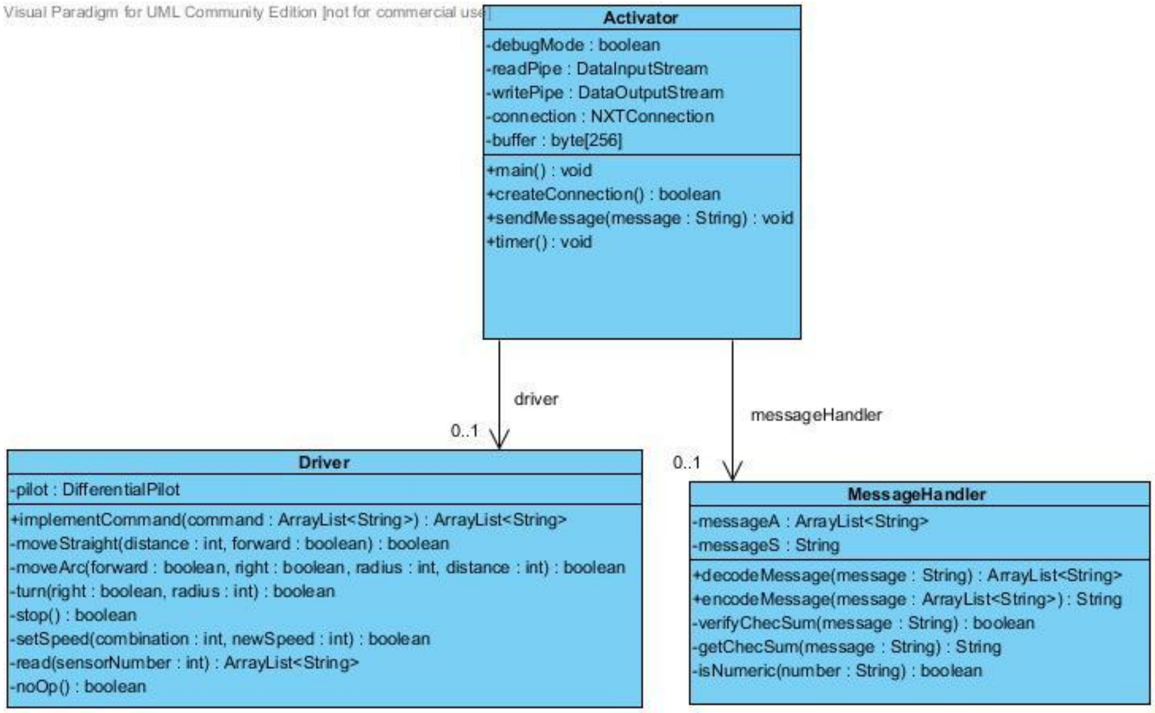
MessageHandler

The MessageHandler class has one purpose: to deal with messages. It will be capable of decoding a message from the base station, validating the checksum, endcoding a new message to send to the base station, and creating a checksum for the new message. It will take messages in the format designated by the Communications Protocol and transform them into a format that the Driver can use to perform actions. On the reverse, it will take messages (acknowledgments or sensor data), and put them into the communications protocol format, so they can be sent over the Bluetooth channel from the Activator class. All encoded and decoded messages are passed back to the Activator class, and from there are sent to their final destination.

Driver

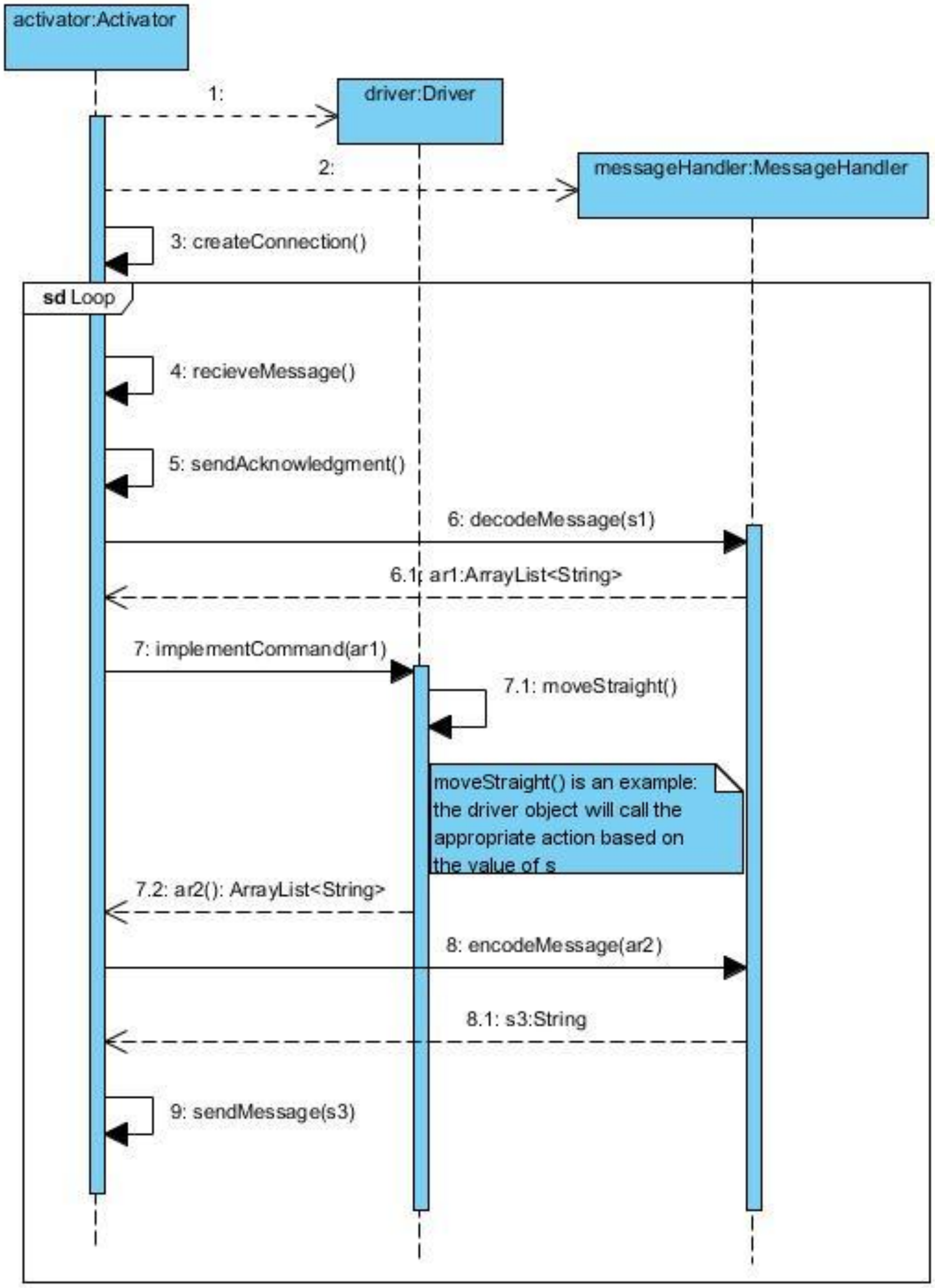
The Driver class is in charge of performing robot actions. It will contain an instance of the Differential Pilot Object from Lejos, which contains classes that control robot movement, such as setting the speed and rotating. The Driver class will contain a method for each action the robot should be able to perform: moveStraight, moveArc, turn, stop, setSpeed, read, and noOp. Additionally, it will have a method called implementAction, which will take in a decoded message and call the correct method to perform the required action.

Static structure


Visual Paradigm for UML Community Edition [not for commercial use]
Activator
-debugMode : boolean
-readPipe : DataInputStream
-writePipe : DataOutputStream
-connection : NXTConnection
-buffer : byte[256]
+main() : void
+createConnection() : boolean
+sendMessage(message : String) : void
+timer() : void
driver
0..1
messageHandler
0..1
Driver
-pilot : DifferentialPilot
+implementCommand(command : ArrayList<String>) : ArrayList<String>
-moveStraight(distance : int, forward : boolean) : boolean
-moveArc(forward : boolean, right : boolean, radius : int, distance : int) : boolean
-turn(right : boolean, radius : int) : boolean
-stop() : boolean
-setSpeed(combination : int, newSpeed : int) : boolean
-read(sensorNumber : int) : ArrayList<String>
-noOp() : boolean
MessageHandler
-messageA : ArrayList<String>
-messageS : String
+decodeMessage(message : String) : ArrayList<String>
+encodeMessage(message : ArrayList<String>) : String
-verifyChecSum(message : String) : boolean
-getChecSum(message : String) : String
-isNumeric(number : String) : boolean

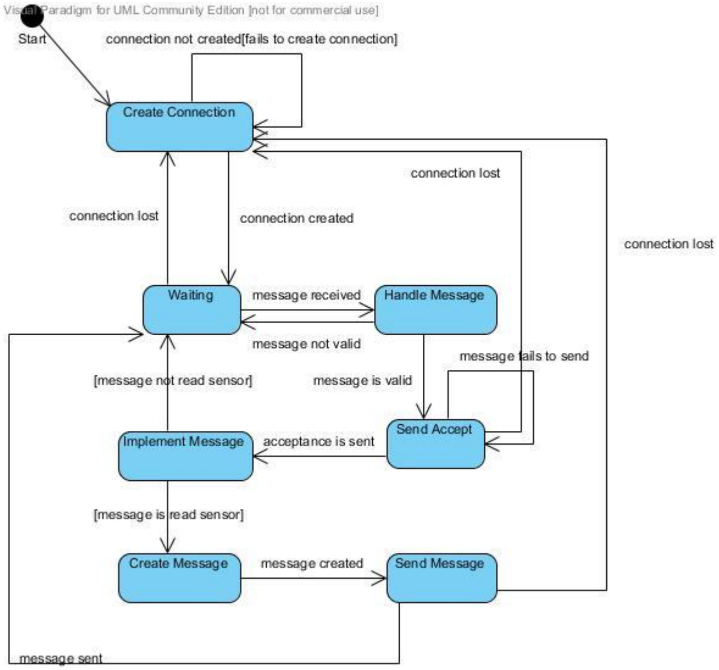
Object interaction structure

UML Sequence Diagram

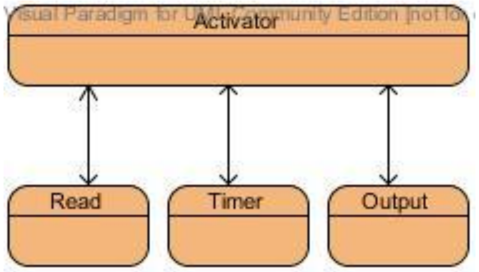


moveStraight() is an example: the driver object will call the appropriate action based on the value of s

Finite State Diagram



Concurrent structure



Class interfaces

Driver

```
/*This class hides the design decisions behind how to
control the actual
*functionality of the robot.
*/

public Class Driver{
   private DifferentialPilot pilot;

   //creates the DifferentialPilot
   public Driver();

   /*
   * public method that implements commands
   * command an array that breaks down each parameter
avaible for any command
   * type
   */
   public String[] implementCommand(String[] command);

   /*
   * private method that hides how movement in a straight
direction works
   * boolean forward move robot forward when true,
backwards when false
   * distance is the distance for the robot to move
   */
   private boolean moveStraight(boolean forward, int
distance);

   /*
   * private method that hides how movement in an arc works
   * boolean forward moves robot forward in arc when true,
and backwards when false
  * boolean right arcs the robot to the right when true,
left when false
   * distance determines the distance for the robot to move
   * radius determines the radius to move along
   */
   private boolean moveArc(boolean forward, boolean right,
int distance, int radius);
```

```
   /*
   * private method that hides how turning works
   * boolean right turns the robot right when true, and
left when false
   * radius determines what radius in degrees to turn
   */
   private boolean turn(boolean right, int radius);

   /*
   * private method stop abstracts how stopping works
   */
   private boolean stop();

   /*
  * private method that hides how setting speed works
   * int combination determines which motor or motor
combination to set speed for
   * newSpeed determines the new speed to set to
   */
   private boolean setSpeed(int combination, int newSpeed);

   /*
   * private method read controls reading a sensor
   * int sensor number determines the sensor to read from
   */
   private ArrayList<String> read(int sensorNumber);

   /*
   * Does nothing, no operation
   */
   private boolean noOp();
   }
```

Activator

```
/*
* This class is designed to handle the connection and
activating
* both driving of the robot hardware and message handling.
*/
public class Activator {
  //Driver that controls the hardware side of robot
   private Driver driver;

   //MessageHandler that creates, encodes, and decodes
messages to be sent
   private MessageHandler messageHandler;

   //boolean used to determine whether to allow
debugCommands or not
   private boolean debugMode;

   //Pipes for reading and writing messages to and from the
base station
   private DataInputStream readPipe;
   private DataOutputStream writePipe;

   /*
   * NXTConnection that acts as the bluetooth connection
between base station
   * and robot
   */
   private NXTConnection connection;

   //buffer used for reading from the stream
   private byte[256] buffer;

   /*
   * main method that controls the creation of connection
and actual running
   * of the robot system
   */
   public static void main(String[] args);

   /*
   * creates the connection between robot and base station
   * allows for multiple connections to be made
   */
   public boolean createConnection();

   /*
   * method that sends message created by messageHandler to
base station
   * message is a message created by messageHandler
   */
   public void sendMessage(String message);

   /*
   * Method that creates the timer for checking timeouts on
messages
   */
   public void timer();
}
```

MessageHandler

```
/*
* This class abstracts away the implementation of the
communications protocol
* This class contains methods that are required to decode
and encode various
* messages that the robot needs to send to the base
station.
*/

public Class MessageHandler{

   /*
   * decodeMessage takes a message and decodes into
parameters for
   * the Driver to use.
   * Parameter message is the message to be decoded
   */
   public ArrayList<String> decodeMessage(String message);

   /*
   * encodeMessage uses parameters from the Driver to
create a message
   * to be sent to the base station.
   * Parameter message is ArrayList of Strings to be used
to crease message
   */
   public String encodeMessage(ArrayList<String> message);

   /*
   * Verify checksum verifies if the calculated checksum is
equivalent
   * to the checksum sent in the message
   * Parameter message is String on which to check checksum
   */
   private boolean verifyChecksum(String message);

   /*
   * Calculates the checksum of the provided message
   * Parameter message is the message on which to get the
checksum
   */
   private String getChecksum(String message);

   /*
   * Checks to see if number is of a numeric type (i.e. it
can be converted to number)
   * Parameter number is the String to check whether the
number is a boolean
  */
   private boolean isNumeric(String number);
}
```