Graphically modeling a Bridge as Critical Section

CS 5800

MD Azharul Islam Anurag Panwar
Nitish Uplavikar

May 16, 2015

Date Performed: May 14, 2015
Instructor: Dr. Wei Jiang

1 Objective

To design a graphical user interface to display the movement(access) of the people(processes) on a bridge(critical section).

2 Introduction

This project is a simplification of the mutual exclusion problem which occurs when two or more persons/processes access the same resource concurrently. As per the specification, the software operates in two modes - *Normal* and *Concurrent*.

2.1 Normal

In this mode, at a time, in any one direction, only one person can cross the bridge. The rest of the persons that want to cross the bridge should wait till they get a hold of the bridge(critical section) using the Ricart-Agarwala's algorithm(see Figure 1).

2.2 Concurrent

This mode is similar to *Normal mode*, but, instead of one, more than one person can cross a bridge at a time, however, all of them must go towards the same direction.

```
program    mutex2
define        m   :   msg
              try, want, in:  boolean
              N     :        integer {number of acknowledgments}
              t     :        timestamp
              A     :        array [0..n-1] of boolean
initially  try = false  {turns true when process wants to enter CS}
           want = false    {turns  false when process exits CS}
           N = 0
           A[k] = false   {∀ k: 0 ≤ k ≤ n-1}
1     do   try                     →   m := (i, req, t);
                                       ∀ j: j ≠ i :: send m to j;
                                       try := false; want := true
2a    □   (m.type = request) ∧
           (¬ want ∨ m.ts < t)     →  send (i, ack, t') to m.sender
2b    □   (m.type = request) ∧
           (want ∧m.ts > t)        →   A [sender] := true
3     □   (m.type = ack)           →   N := N + 1;
4    □  (N = n-1)              →   in := true;
                                   {process enters CS}
                                   want:= false
5    □   in ∧   ¬ want         →   in := false; N : = 0;
                                   ∀k : A[k] ::
                                   send (i, ack, t') to k;
                                   ∀k : A[k] :: A[k] := false;

     od
```

Figure 1: Ricart-Agarwala's algorithm

3 Design

An object oriented software model is used to implement this project. Following are the different classes used within the project:

3.1 Class Description

A brief description about classes, their important members and methods is found below.

3.1.1 Person

Class *Person* realizes to the actual Person object crossing the bridge. Notable members include *direction* and *velocity*. Each Person's *receive* method runs as an independent thread, sending, listening for different message types such as *REQUEST*, *ACK*, defined in class *Constants*, from other Persons in the system.

3.1.2 AnimationPanel

AnimationPanel extends the Java Panel(*JPanel*). It also maintains a list structure to add, remove the persons to be shown in the graphics.

3.1.3 ControlPanel

ControlPanel is a Java Swing Frame(*JFrame*) which provides the interface to set an *Person* object's speed and allows setting the mode of the program ie. either *Normal* or *Concurrent* mode.

3.1.4 MainClass

As the name suggests, is the beginning of the program. Draws the *AnimationPanel* and *ControlPanel*.

3.1.5 Constants

This class defines the constant values for global variables used throughout the program eg. type of messages, listening port numbers for the processes etc.

3.1.6 Message

This class implements the Java *Serializable* class. Stores the message's sender id, type and timestamp.

3.2 UML Diagrams

Following are some of the major diagrams that convey the details of the project.

3.2.1 Class dependencies

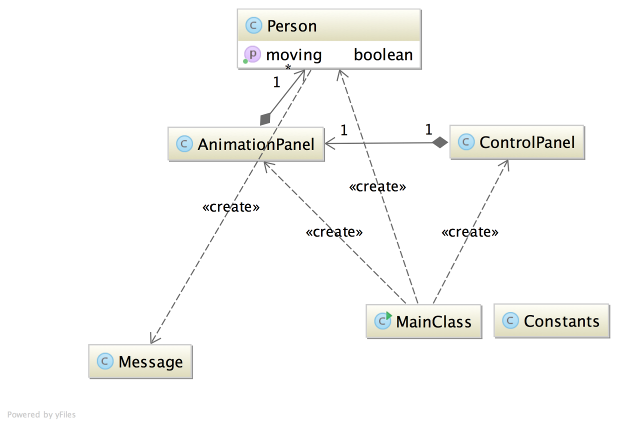


Figure 2: Diagram showing the dependencies among the classes

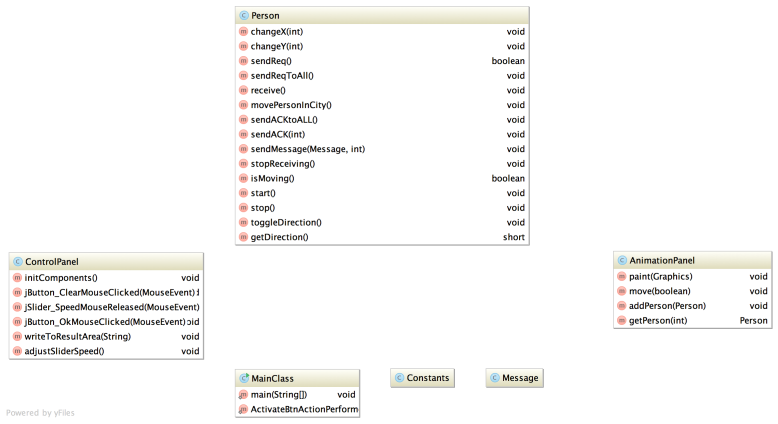


Figure 3: Diagram showing the behaviors exhibited by the classes

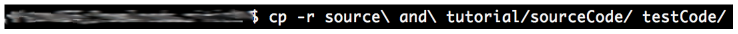
3.2.2 Class behaviors

4 Installation and running Steps

4.1 Installation

*Please follow the following steps for successfully running the source code that is given.

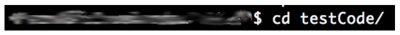
a. Copy the folder sourceCode to any directory.



```
[redacted]$ cp -r source\ and\ tutorial/sourceCode/ testCode/
```

Figure 4: Execution of Installation Step a

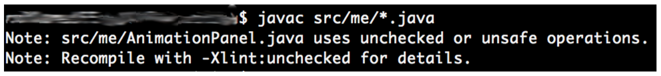
b. From commandline go to the directory *sourceCode*.
Command: cd sourceCode.



```
[redacted]$ cd testCode/
```

Figure 5: Execution of Installation Step b

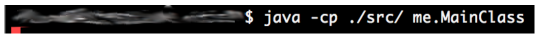
c. **Compile the source files:
Command: javac src/me/*.java



```
[redacted]$ javac src/me/*.java
Note: src/me/AnimationPanel.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

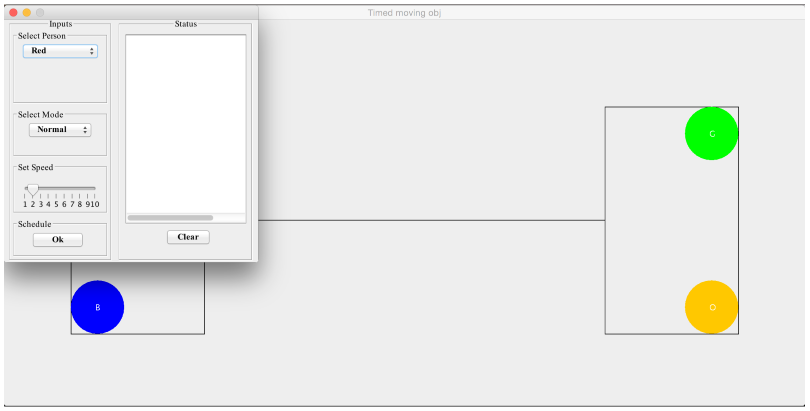
Figure 6: Execution of Installation Step c

d. **Run the program
Command: java -cp ./src/ me.MainClass



```
[redacted]$ java -cp ./src/ me.MainClass
```

Figure 7: Execution of Installation Step d



Select Person: Red
Select Mode: Normal

Figure 8: Execution of Program

4.2 Running steps

As shown in figure 8, user can modify the input fields - *Person*, *Mode*, *Speed* and *Schedule*. *Status* shows the messages corresponding to a user setting these items etc.

a. Select appropriate Person from the drop-down list.

b. Likewise, also select the mode to operate the software. If you have already started the execution by clicking 'Schedule' and you want to change the mode of operation, you have to restart the program by terminating it first before restarting(See last step for terminating instruction).

c. Using the speed sliding window, for the selected person, set the appropriate speed for it to move.

d. Click on Schedule to begin the graphical animation.

e. The program can be terminated by clicking the red terminate key(X) for the window.

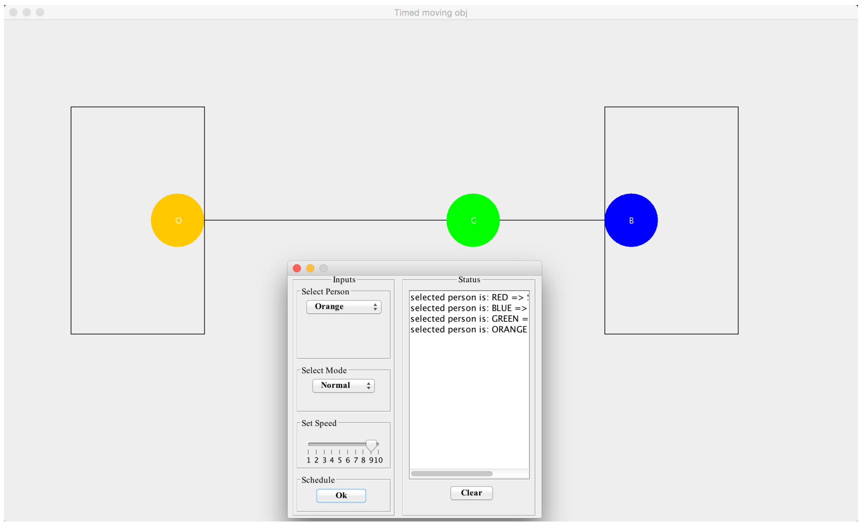
4.3 Notes

** Please note that, before running the commands it is required that the environment path is properly set up for java. For setting up environment path the following tutorial can be followed: `http://www.tutorialspoint.com/java/java_environment_setup.htm`.
* The compilation was done on a system with following details:
Hardware Overview:

Model Name:	MacBook Pro
Model Identifier:	MacBookPro11,1
Processor Name:	Intel Core i5
Processor Speed:	2.6 GHz
Number of Processors:	1
Total Number of Cores:	2
L2 Cache (per Core):	256 KB
L3 Cache:	3 MB
Memory:	8 GB
Boot ROM Version:	MBP111.0138.B14
SMC Version (system):	2.16f68

5 Output Screenshots

Following are some of the screen-shots showing the output in different scenarios.
-

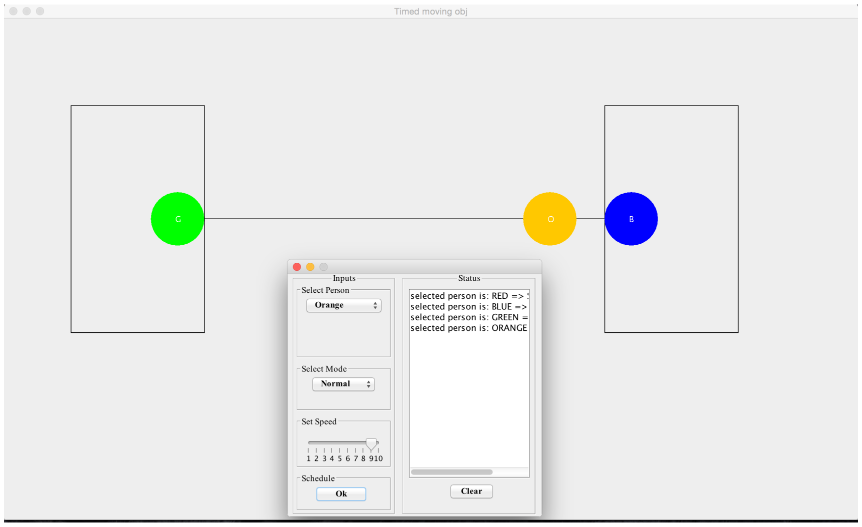


Select Person: Orange

Select Mode: Normal

```
selected person is: RED =>
selected person is: BLUE =>
selected person is: GREEN =
selected person is: ORANGE
```

Figure 9: Normal Mode: R,O,B - Waiting; G - Crossing

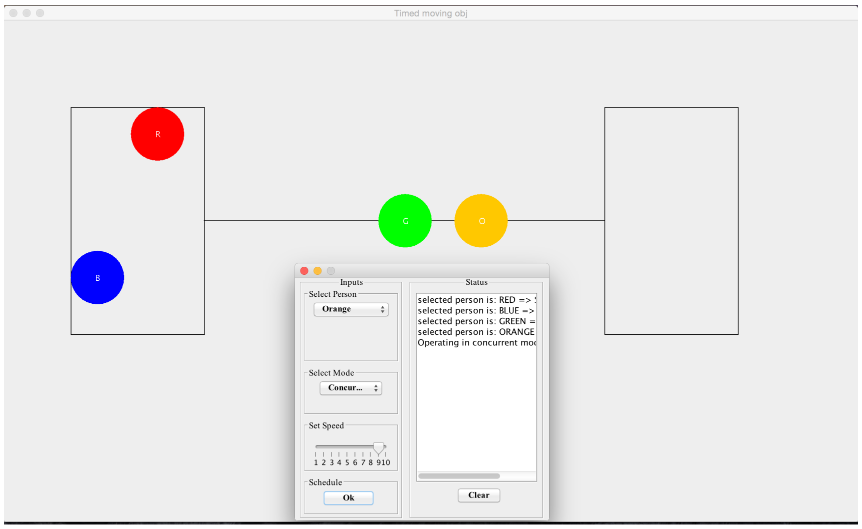


Select Person: Orange

Select Mode: Normal

```
selected person is: RED =>
selected person is: BLUE =>
selected person is: GREEN =
selected person is: ORANGE
```

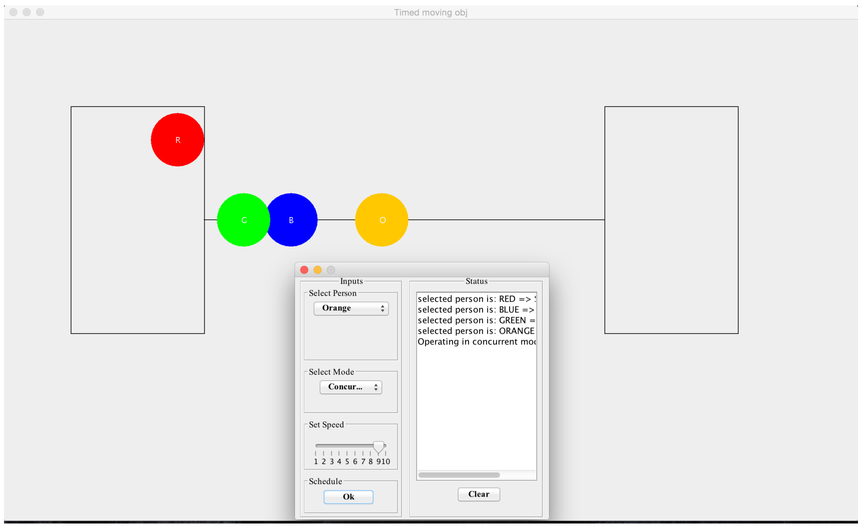
Figure 10: Normal Mode: R,G,B - Waiting; O - Crossing



Select Person: Orange
Select Mode: Concur...
Set Speed
Schedule

```
selected person is: RED => 
selected person is: BLUE =>
selected person is: GREEN =
selected person is: ORANGE
Operating in concurrent mo
```

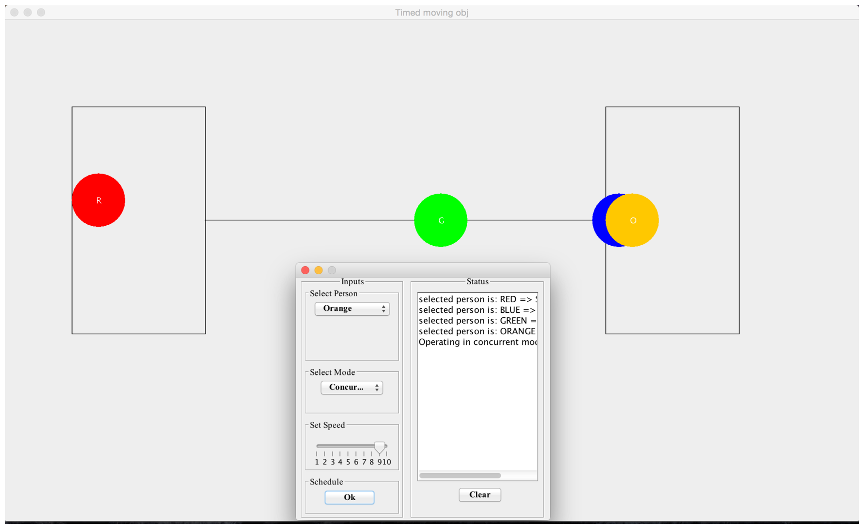
Figure 11: Concurrent Mode: R,B - Not Waiting; O,G - Crossing



Select Person: Orange
Select Mode: Concur...
Set Speed
Schedule

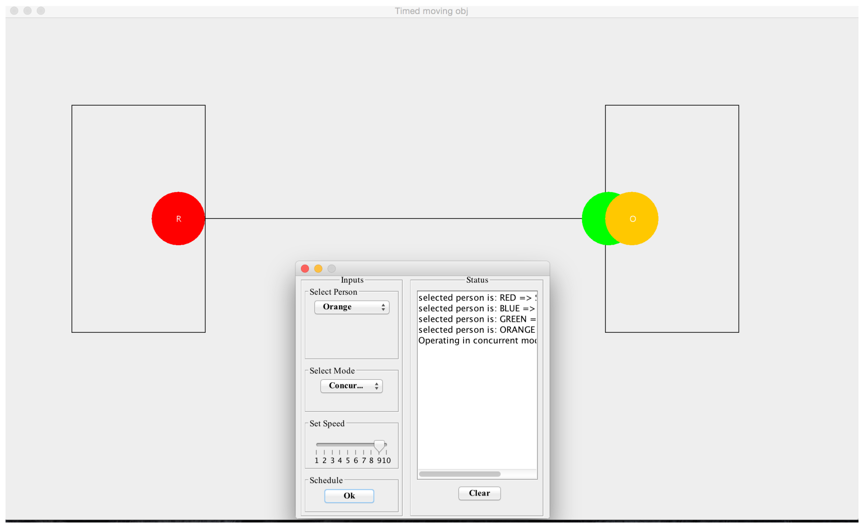
```
selected person is: RED => 
selected person is: BLUE =>
selected person is: GREEN =
selected person is: ORANGE
Operating in concurrent mo
```

Figure 12: Concurrent Mode: R - Not Waiting; G,B,O - Crossing



```
selected person is: RED => 
selected person is: BLUE => 
selected person is: GREEN =
selected person is: ORANGE
Operating in concurrent mo
```

Figure 13: Concurrent Mode: R - Not Waiting; O - Waiting; B,G - Crossing



```
selected person is: RED => 
selected person is: BLUE => 
selected person is: GREEN =
selected person is: ORANGE
Operating in concurrent mo
```

Figure 14: Concurrent Mode: Fairness property being preserved
R - Waiting, has earlier request as compared to B,O;
B,O - Waiting, they don't have earlier request than R so not going; G - Crossing

6 Bibliography

Sukumar Ghosh, *Distributed Systems An Algorithmic Approach*, Chapman and Hall CRC, Boca Raton, Florida, 2007.