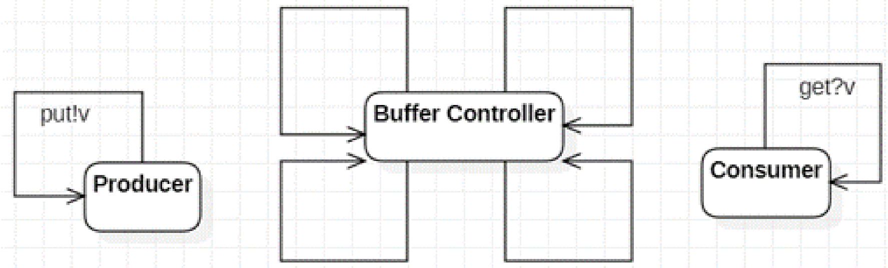
Assignment 5 Part 2

(may be done by a team of at most two students)
Assigned: November 25
Due: Weds, December 4 (11:59 pm), for Parts 1 and 2

Part 2: Communicating State Charts



File buffer.mdj contains a **Star UML State Chart** (reproduced above) giving the outline of the controller for a circular buffer. This buffer is accessed by two *concurrent* processes, Producer and Consumer, who communicate with the buffer controller using two *channels,* put and get. The producer repeatedly does a put! and the consumer similarly does a get?, and the buffer controller performs the respective complementary operations.

File Circular_Buffer.java contains a Java program providing an outline of the implementation of the above scenario using the *Message Passing Library* (see Lecture 18). It gives the main program, the classes Producer and Consumer, and also the outline of class Circular_Buffer.

Class Circular_Buffer uses an integer array, data, of size n in order to hold its data. It has a field count that gives, at any given time, the number of values that can be taken out of the buffer. It also has two indices p and g to point, respectively, to the places in the array where the next value is to be put by the producer and taken out by the consumer. These indices are incremented (modulo n) as put/get operations take place. The actual insertion and retrieval of values are performed by two methods put() and get() respectively.

What you should do:

1. Complete the Star UML State Chart by providing suitable labels on the four transitions shown. Each label is of the form ***event [guard] / action*** – see Lecture 25 slides 26-31 for details. The labels should collectively express the synchronization policy of the buffer controller, namely, that:

    (i) when the buffer is empty (count == 0) only a put operation is permitted;
    (ii) when the buffer is full (count == n) only a get operation is permitted;
    (iii) otherwise, both put and get are permitted - the selection is non-deterministic.

    In specifying the transition labels in Star UML:

    (i) each event is a channel send/receive, and is specified as a *Trigger Event*;
    (ii) each guard is a boolean expression and specified in the *Properties* section; and

(iii) each action is a call on one of the methods put() or get(), and specified as an *Effect Behavior* → *OpaqueBehavior*.

Update the state chart buffer.mdj as per the above specification.

2. Complete the run() method in class Circular_Buffer providing an implementation of the above synchronization policy. As the state chart specifies that the buffer controller operates in a repetitive cycle, the top level of the run() method should be a while(true) {…} loop.

   In Eclipse, right-click on the project, then select *Build Path* → *Configure Build Path* → *Add External JAR* → *browse and select MessagePassing2.jar*. The file *MessagePassing2.jar* is given in this directory.

   Run the completed program under JIVE after adding Scheduler.* to *Debug Configurations* → *JIVE* → *Exclusion Filter*. Check that the *Console* output shows the strings Put:1 … Put:10 and Get:1 … Get:10. This output will be interleaved with the underlying scheduler's log showing that it scheduled 20 send-receive pairs.

   Save the *Execution Trace* in a file, buffer.csv, and load it into the *Property Checker*. Add Circular_Buffer:1.state to the *Key Attributes*. Enter Circular_Buffer:1.state = s in the *Abbreviations* textbox. Finally, copy the contents of the file property.txt (with comments) into the *Properties* textbox. Press *Validate* and check that all properties are satisfied; otherwise, the program has an error which needs to be corrected.

What to Submit:

Prepare a top-level directory named *A5_Part1_UBITId1_UBITId2* if the assignment is done by a team of two students; otherwise, name it as *A5_Part1_UBITId* if the assignment is done solo. (Order the *UBITId*s in alphabetic order, in the former case.) In this directory, place the updated buffer.mdj and Circular_Buffer.java, and also buffer.csv. Compress the directory and submit the compressed file using the submit_cse522 command. Only one submission per team is required.

End of Assignment 5 Part 2