Programming assignment 3 – Shared Memory

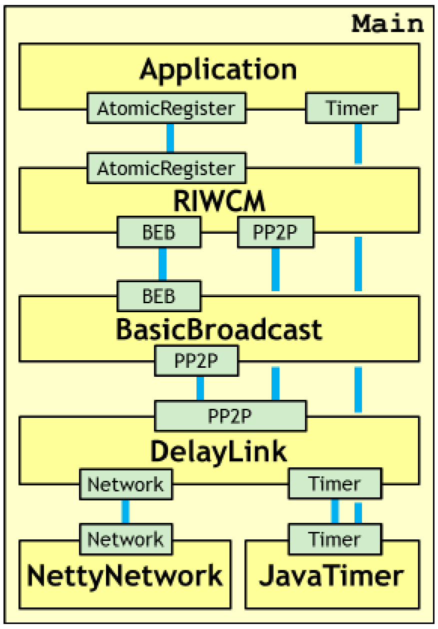
Introduction

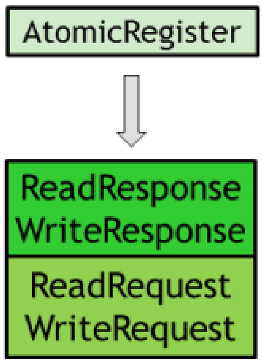
In this programming assignment you shall implement the Read-Impose Write-Consult-Majority component that provides the Atomic Register service. You shall use the algorithm from the textbook (algorithm 4.10 and 4.11). The algorithm is also reproduced at the end of this document.

Installation

Download id2203-ass3-shm.zip from the course website, unpack and import into Eclipse in the same way as for previous assignments.

Architecture





The `AtomicRegister` port is defined in `se.kth.ict.id2203.ports.ar`.

Code to write

The component shall be implemented in the `ReadImposeWriteConsultMajority.java` file in the `se.kth.ict.id2203.components.riwcm` package. You will have to add files for internal events as you see fit.

Exploration

You shall do the following:

- There are a number of scenarios in `Executor.java`; run each of them and make sure that you understand how/why it works. Write down a valid linearization for each execution.
    - If too many log messages makes it hard to see the relevant ones, remember that you can suppress log messages (typically uncomment line 18 of `log4j.properties`).

- Play around with executing reads and writes as you like (either interactively using the `R` and `Wn` commands, or as a scenario).
    - You can change the topology (number of processes and link delays) in `Executor.java`, and also add normal distributed variance to the link delay by changing the `sigma` parameter (currently set to 0 ms) at line 72 of `Main.java`.

Automatic correction

When everything is working you run the `AutomaticCorrection.java` file to test the component and submit the assignment to the http://cloud7.sics.se:11700/ server. Remember to change the email and password strings before running.

Algorithm 1 Read-Impose Write-Consult-Majority (part 1, read and consult)

```
Implements:
        AtomicRegister, instance nnar.
Uses:
        BestEffortBroadcast, instance beb;
        PerfectPointToPointLink, instance pp2p.

 1: upon event ⟨ nnar, Init ⟩ do
 2:     $(ts, wr, val) := (0, 0, 0)$;
 3:     $acks := 0$;
 4:     $writeval := \bot$;
 5:     $rid := 0$;
 6:     $readlist := [\bot]^N$;
 7:     $readval := \bot$;
 8:     $reading :=$ FALSE;

 9: upon event ⟨ nnar, Read ⟩ do
10:     $rid := rid + 1$;
11:     $acks := 0$;
12:     $readlist := [\bot]^N$;
13:     $reading :=$ TRUE;
14:     trigger ⟨ beb, Broadcast | [READ, rid] ⟩;

15: upon event ⟨ beb, Deliver | p, [READ, r] ⟩ do
16:     trigger ⟨ pp2p, Send | p, [VALUE, r, ts, wr, val] ⟩;

17: upon event ⟨ pp2p, Deliver | q, [VALUE, r, ts', wr', v'] ⟩ such that $r = rid$ do
18:     $readlist[q] := (ts', wr', v')$;
19:     if $\#(readlist) > N/2$ then
20:         $(maxts, rr, readval) := highest(readlist)$;
21:         $readlist := [\bot]^N$;
22:         if $reading =$ TRUE then
23:             trigger ⟨ beb, Broadcast | [WRITE, rid, maxts, rr, readval] ⟩;
24:         else
25:             trigger ⟨ beb, Broadcast | [WRITE, rid, maxts + 1, rank(self), writeval] ⟩;
```

Algorithm 2 Read-Impose Write-Consult-Majority (part 2, write and write-back)

```
26: upon event ⟨ $nnar, Write \mid v$ ⟩ do
27:     $rid := rid + 1$;
28:     $writeval := v$;
29:     $acks := 0$;
30:     $readlist := [\perp]^N$;
31:     trigger ⟨ $beb, Broadcast \mid$ [READ, $rid$] ⟩;

32: upon event ⟨ $beb, Deliver \mid p$, [WRITE, $r, ts', wr', v'$] ⟩ do
33:     if $(ts', wr')$ is larger than $(ts, wr)$ then
34:         $(ts, wr, val) := (ts', wr', v')$;
35:     trigger ⟨ $pp2p, Send \mid p$, [ACK, $r$] ⟩;

36: upon event ⟨ $pp2p, Deliver \mid q$, [ACK, $r$] ⟩ such that $r = rid$ do
37:     $acks := acks + 1$;
38:     if $acks > N/2$ then
39:         $acks := 0$;
40:         if $reading$ = TRUE then
41:             $reading$ := FALSE;
42:             trigger ⟨ $nnar, ReadReturn \mid readval$ ⟩;
43:         else
44:             trigger ⟨ $nnar, WriteReturn$ ⟩;
```