

ID2203 Tutorial 4

Cosmin Arad
icarad@kth.se

Handout: February 20, 2008
Due: February 27, 2008

1 Introduction

The goal of this tutorial is to understand and get accustomed to implementing *atomic register* shared memory abstractions for the synchronous and asynchronous system models.

2 Assignment

Using the TBN framework, you will implement two atomic register abstractions as components: a fail-stop atomic register with multiple writers, described in Algorithm 1 and Algorithm 2 (Read-Impose Write-Consult), and a fail-silent atomic register with multiple writers, described in Algorithm 3 and Algorithm 4 (Read-Impose Write-Consult-Majority).

You will use the same type of input as in the previous assignments, to specify your network of processes, i.e., a `topology.xml` file. You will reuse the communication component, the timer component, the delay component, the beb broadcast component, and the pfd component. You have to implement a `FailStopAtomicRegisterComponent` (RIWC) and a `FailSilentAtomicRegisterComponent` (RIWCM), together with the events that are accepted and triggered by these components: `AtomicRegisterReadEvent`, `AtomicRegisterReadReturnEvent`, `AtomicRegisterWriteEvent`, `AtomicRegisterWriteReturnEvent` and the messages that they exchange: `ReadMessage`, `WriteMessage`, `AckMessage`, and `ReadValueMessage`.

You will have a separate architecture for testing the RIWC and RIWCM, i.e., an `assignment4-riwc.xml` and an `assignment4-riwcm.xml`. When launching the `Assignment4.java` you pass a command line argument to specify which architecture to use, and a string containing the register operations issued by the application component:

```
java Assignment4 <topology.xml> <nodeId> riwc|riwcm <ops>.
```

You will slightly change the application component from the third assignment, to handle the `AtomicRegisterReadReturnEvent` and the `AtomicRegisterWriteReturnEvent`, and an `ApplicationEvent` raised by the main program and handled by the application component, that contains the string `ops`. The `ops` string contains read and write operations that are to be issued by the application component on the atomic register with index 0. Here is an example: `W5:R:W6:R:D200:W3:D100:R`. This means that the application component will issue a `Write(5)` operation on register 0. Upon completion of this operation, it will issue a `Read` operation on register 0. Upon completion of this operation, it will issue a `Write(6)` operation on register 0. Upon completion of this operation, it will do nothing (sleep/timeout) for 200 milliseconds and then it will issue a `Write(3)` operation on register 0. Upon completion of this operation, it will again wait for 100 milliseconds and issue a `Read` operation on register 0.

You use the same `InitEvent` to pass to the components the set of all processes (Π), or the number of processes, respectively. Note that the algorithms assume that each process knows and can communicate with all other processes in Π . This corresponds to a fully connected topology.

These RIWC algorithm, relying on the PFD, assumes that all processes are started within γ^1 milliseconds. To start all your processes at once, you can use the following commands on Windows (cmd) and Linux (bash) respectively:

```
FOR /L %G IN (0, 1, 5) DO start java Assignment4 topology.xml %G ...
for ((i=0;i<=5;i++)) \
    do java Assignment4 topology.xml $i ... > p$i.log 2>&1 & done
```

Replace 5 with your number of processes minus one. For Linux, you can inspect the output of your processes in the `p*.log` files.

Implement the RIWC and RIWCM components and experiment with them as instructed in the following exercises. Describe your experiments in a written report. For each exercise include the topology descriptors used, and explain the behavior that you observe.

You need to read Sections 4.1-4.4 of the textbook, in order to fully understand the ideas behind the two algorithms. These algorithms implement arrays of atomic registers and thus are parameterized by the size of the array. Specify this parameter in a configuration file, `riwc.properties`, that you use to initialize both atomic register components. Here is an example `riwc.properties` file:

¹the interval between sending two consecutive heartbeats in PFD.

`capacity = 10` `# number of registers in the shared memory`

The assignment is due on February 27th. You have to send your source code and written report by email before the next tutorial session. During the tutorial session you will present the assignment on a given topology description. You can work in groups of maximum 2 students. Be prepared to answer questions about your process's system architecture and explain the behavior of the algorithms. Any questions are welcome on the mailing list.

Algorithm 1 Read-Impose Write-Consult (part 1)

Implements:

(N, N) AtomicRegister (nn-areg).

Uses:

BestEffortBroadcast (beb);

PerfectPointToPointLinks (pp2p);

PerfectFailureDetector (\mathcal{P}).

```
1: upon event  $\langle \text{Init} \rangle$  do
2:   correct :=  $\Pi$ ;
3:   i := rank(self);
4:   for all  $r$  do
5:     writeSet[r] :=  $\emptyset$ ;
6:     reading[r] := false;
7:     reqid[r] := 0;
8:     readval[r] := 0;
9:     v[r] := 0;
10:    ts[r] := 0;
11:    mrank[r] := 0;
12:   end for
13: end event

14: upon event  $\langle \text{crash} \mid p_i \rangle$  do
15:   correct := correct  $\setminus \{p_i\}$ ;
16: end event

17: upon event  $\langle \text{nn-aRegRead} \mid r \rangle$  do
18:   reqid[r] := reqid[r]+1;
19:   reading[r] := true;
20:   writeSet[r] :=  $\emptyset$ ;
21:   readval[r] := v[r];
22:   trigger  $\langle \text{bebBroadcast} \mid [\text{WRITE}, r, \text{reqid}[r], (\text{ts}[r], \text{mrnk}[r]), \text{v}[r]] \rangle$ ;
23: end event

24: upon event  $\langle \text{nn-aRegWrite} \mid r, \text{val} \rangle$  do
25:   reqid[r] := reqid[r]+1;
26:   writeSet[r] :=  $\emptyset$ ;
27:   trigger  $\langle \text{bebBroadcast} \mid [\text{WRITE}, r, \text{reqid}[r], (\text{ts}[r]+1, i), \text{val}] \rangle$ ;
28: end event
```

Algorithm 2 Read-Impose Write-Consult (part 2)

```
1: upon event  $\langle bebDeliver \mid p_j, [WRITE, r, id, (t, j), val] \rangle$  do
2:   if  $(t, j) > (ts[r], mrank[r])$  then
3:      $v[r] := val$ ;
4:      $ts[r] := t$ ;
5:      $mrank[r] := j$ ;
6:   end if
7:   trigger  $\langle pp2pSend \mid p_j, [ACK, r, id] \rangle$ ;
8: end event

9: upon event  $\langle pp2pDeliver \mid p_j, [ACK, r, id] \rangle$  do
10:  if  $(id = reqid[r])$  then
11:     $writeSet[r] := writeSet[r] \cup \{p_j\}$ ;
12:  end if
13: end event

14: upon exists  $r$  such that  $correct \subseteq writeSet[r]$  do
15:  if  $(reading[r] = true)$  then
16:     $reading[r] := false$ ;
17:    trigger  $\langle nn-aRegReadReturn \mid r, readval[r] \rangle$ ;
18:  else
19:    trigger  $\langle nn-aRegWriteReturn \mid r \rangle$ ;
20:  end if
21: end
```

Algorithm 3 Read-Impose Write-Consult-Majority (part 1)

Implements: (N, N) AtomicRegister (nn-areg).**Uses:**

BestEffortBroadcast (beb);

PerfectPointToPointLinks (pp2p).

```
1: upon event  $\langle \text{Init} \rangle$  do
2:    $i := \text{rank}(\text{self});$ 
3:   for all  $r$  do
4:      $\text{writeSet}[r] := \emptyset;$ 
5:      $\text{readSet}[r] := \emptyset;$ 
6:      $\text{reading}[r] := \text{false};$ 
7:      $\text{reqid}[r] := 0;$ 
8:      $v[r] := 0;$ 
9:      $\text{ts}[r] := 0;$ 
10:     $\text{mrnk}[r] := 0;$ 
11:   end for
12: end event

13: upon event  $\langle \text{nn-aRegRead} \mid r \rangle$  do
14:    $\text{reqid}[r] := \text{reqid}[r] + 1;$ 
15:    $\text{reading}[r] := \text{true};$ 
16:    $\text{readSet}[r] := \emptyset;$ 
17:    $\text{writeSet}[r] := \emptyset;$ 
18:   trigger  $\langle \text{bebBroadcast} \mid [\text{READ}, r, \text{reqid}[r]] \rangle;$ 
19: end event

20: upon event  $\langle \text{nn-aRegWrite} \mid r, \text{val} \rangle$  do
21:    $\text{reqid}[r] := \text{reqid}[r] + 1;$ 
22:    $\text{writeval}[r] := \text{val};$ 
23:    $\text{readSet}[r] := \emptyset;$ 
24:    $\text{writeSet}[r] := \emptyset;$ 
25:   trigger  $\langle \text{bebBroadcast} \mid [\text{READ}, r, \text{reqid}[r]] \rangle;$ 
26: end event

27: upon event  $\langle \text{bebDeliver} \mid p_j, [\text{READ}, r, \text{id}] \rangle$  do
28:   trigger  $\langle \text{pp2pSend} \mid p_j, [\text{READVAL}, r, \text{id}, (\text{ts}[r], \text{mrnk}[r]), v[r]] \rangle;$ 
29: end event
```

Algorithm 4 Read-Impose Write-Consult-Majority (part 2)

```
1: upon event  $\langle pp2pDeliver \mid p_j, [READVAL, r, id, (t, rk), val] \rangle$  do
2:   if  $(id = reqid[r])$  then
3:      $readSet[r] := readSet[r] \cup \{((t, rk), val)\};$ 
4:   end if
5: end event

6: upon exists  $r$  such that  $|readSet[r]| > N/2$  do
7:    $((t, rk), v) := highest(readSet[r]);$ 
8:    $readval[r] := v;$ 
9:   if  $(reading[r] = \text{true})$  then
10:    trigger  $\langle bebBroadcast \mid [WRITE, r, reqid[r], (t, rk), readval[r]] \rangle;$ 
11:   else
12:    trigger  $\langle bebBroadcast \mid [WRITE, r, reqid[r], (t+1, i), readval[r]] \rangle;$ 
13:   end if
14: end

15: upon event  $\langle bebDeliver \mid p_j, [WRITE, r, id, (t, j), val] \rangle$  do
16:   if  $(t, j) > (ts[r], mrank[r])$  then
17:      $v[r] := val;$ 
18:      $ts[r] := t;$ 
19:      $mrank[r] := j;$ 
20:   end if
21:   trigger  $\langle pp2pSend \mid p_j, [ACK, r, id] \rangle;$ 
22: end event

23: upon event  $\langle pp2pDeliver \mid p_j, [ACK, r, id] \rangle$  do
24:   if  $(id = reqid[r])$  then
25:      $writeSet[r] := writeSet[r] \cup \{p_j\};$ 
26:   end if
27: end event

28: upon exists  $r$  such that  $|writeSet[r]| > N/2$  do
29:   if  $(reading[r] = \text{true})$  then
30:      $reading[r] := \text{false};$ 
31:     trigger  $\langle nn-aRegReadReturn \mid r, readval[r] \rangle;$ 
32:   else
33:     trigger  $\langle nn-aRegWriteReturn \mid r \rangle;$ 
34:   end if
35: end
```
