Programming assignment 5 – Total Order Broadcast

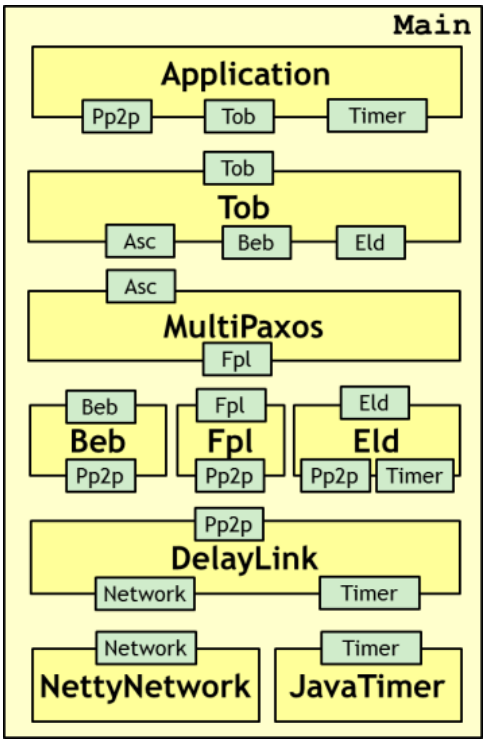
Introduction

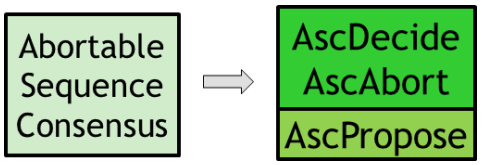
In this programming assignment you shall implement the Multi-Paxos component that provides the Abortable Sequence Consensus service. The algorithm is available at the end of this document (the algorithm is not available in the textbook).

Installation

Download id2203-ass5-tob.zip from the course website, unpack and import into Eclipse in the same way as for previous assignments.

Architecture





The `AbortableSequenceConsensus` port is defined in `se.kth.ict.id2203.ports.asc`.

Code to write

The component shall be implemented in the `MultiPaxos.java` file in the `se.kth.ict.id2203.components.multipaxos` package. You will have to add files for messages, etc, as you see fit.

Various notes

- You need to add the following messages (events) in the component's package:
  - `PrepareMessage` ⊆ `FplDeliver`
  - `PrepareAckMessage` ⊆ `FplDeliver`
  - `NackMessage` ⊆ `FplDeliver`

    - `AcceptMessage` ⊆ `FplDeliver`
    - `AcceptAckMessage` ⊆ `FplDeliver`
    - `DecideMessage` ⊆ `FplDeliver`
- You are highly encouraged to insert logging statements in the code to be able to trace the execution. If you have problems this will help you explain where things are not working.
- Assignment 5 is relatively new (as of last year). Hopefully there aren't any bugs, but there is always a small risk. Please check the forum if you encounter any issues.
- The `ArrayList` data structure is probably a good choice for implementing sequences.
- `prefix(σ,k)` returns the prefix of sequence σ with at most k elements.
- `suffix(σ,k)` returns the suffix of σ by skipping the k first elements and returning the rest (possibly an empty sequence if $|\sigma| \leq k$).
- $\sigma_1+\sigma_2$ is the concatenation of sequence σ_1 and σ_2.
- The values that are proposed and later delivered can have any type `T` that is a subtype of `Object`, and `T` must have an `equals` method.
- Any object that is sent into or received out of the framework must be treated as immutable (read-only).
    - In particular, any sequence passed in a message must be read-only and may not be modified after it has been sent or received. Typically, create a new `Object[]` array, copy the elements from the sequence into the array, and send the array.
- To truncate a sequence you can use `list.subList(from,to).clear()`, as described here: http://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html#subList(int, int)

Notation in pseudo-code vs. in lecture slides

The notation in the pseudo-code follows the notation used for the pseudo-code in programming assignment 4. Unfortunately this notation differs from the notation used in the lecture slides. Here follows a table with correspondences between the two notations:

Pseudo-code	Lecture slides
prepts	n_p
ats	n_a
av	v_a
al	l_d
pts	n_c
pv	v_c
pl	l_c
proposedValues	Missing in slides
readlist	S
accepted	a
decided	s

Exploration

In this assignment there are two applications that use the `MultiPaxos` component.

The first application in `se.kth.ict.id2203.pa.multipaxos` uses the `MultiPaxos` component directly, similarly to how the application in assignment 4 used the `Paxos` component. The application provides the `Pn` and `Cn` commands, as in assignment 4. You should run the three scenarios in `Executor.java` and verify that you get the desired outcome for each scenario.

The second application is in `se.kth.ict.id2203.pa.tob` and uses the `Tob` component, which indirectly uses the `MultiPaxos` component. The application uses the `Bm` command to broadcast message `m` (a string). Run the three scenarios in `Executor.java` and make sure that messages are delivered at all correct processes in a total order.

Automatic correction

When everything is working you run the `AutomaticCorrection.java` file in the `se.kth.ict.id2203.pa.tob` package 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 Multi-Paxos: Prepare Phase

Implements:

AbortableSequenceConsensus, **instance** *asc*.

Uses:

FIFOPerfectPointToPointLinks, **instance** *fpl*.

1: **upon event** $\langle\ asc, Init\ \rangle$ **do**
2: $t := 0$; ▷ logical clock
3: $prepts := 0$; ▷ acceptor: prepared timestamp
4: $(ats, av, al) := (0, \langle\rangle, 0)$; ▷ acceptor: timestamp, accepted seq, length of decided seq
5: $(pts, pv, pl) := (0, \langle\rangle, 0)$; ▷ proposer: timestamp, proposed seq, length of learned seq
6: $proposedValues := \langle\rangle$; ▷ proposer: values proposed while preparing
7: $readlist := [\bot]^N$;
8: $accepted := [0]^N$; ▷ proposer's knowledge about length of acceptor's longest accepted seq
9: $decided := [0]^N$; ▷ proposer's knowledge about length of acceptor's longest decided seq

10: **upon event** $\langle\ asc, Propose \mid v\ \rangle$ **do**
11: $t := t + 1$;
12: **if** $pts = 0$ **then**
13: $pts := t \times N + rank(self)$;
14: $pv := prefix(av, al)$;
15: $pl := 0$;
16: $proposedValues := \langle v \rangle$;
17: $readlist := [\bot]^N$;
18: $accepted := [0]^N$;
19: $decided := [0]^N$;
20: **for all** $p \in \Pi$ **do**
21: **trigger** $\langle\ fpl, Send \mid p, [\text{PREPARE}, pts, al, t]\ \rangle$;
22: **else if** $\#(readlist) \leq \lfloor N/2 \rfloor$ **then**
23: $proposedValues := proposedValues + \langle v \rangle$; ▷ append to sequence
24: **else if** $v \notin pv$ **then**
25: $pv := pv + \langle v \rangle$;
26: **for all** $p \in \Pi$ **such that** $readlist[p] \neq \bot$ **do**
27: **trigger** $\langle\ fpl, Send \mid p, [\text{ACCEPT}, pts, \langle v \rangle, \#(pv) - 1, t]\ \rangle$;

28: **upon event** $\langle\ fpl, Deliver \mid q, [\text{PREPARE}, ts, l, t']\ \rangle$ **do**
29: $t := max(t, t') + 1$;
30: **if** $ts < prepts$ **then**
31: **trigger** $\langle\ fpl, Send \mid q, [\text{NACK}, ts, t]\ \rangle$;
32: **else**
33: $prepts := ts$;
34: **trigger** $\langle\ fpl, Send \mid q, [\text{PREPAREACK}, ts, ats, suffix(av, l), al, t]\ \rangle$;

35: **upon event** $\langle\ fpl, Deliver \mid q, [\text{NACK}, pts', t']\ \rangle$ **do**
36: $t := max(t, t') + 1$;
37: **if** $pts' = pts$ **then**
38: $pts := 0$;
39: **trigger** $\langle\ asc, Abort\ \rangle$

Algorithm 2 Multi-Paxos: Accept Phase

40: **upon event** $\langle\ fpl, Deliver \mid q, [\textsc{PrepareAck}, pts', ts, vsuf, l, t']\ \rangle$ **do**
41: $\quad t := max(t, t') + 1;$
42: $\quad$ **if** $pts' = pts$ **then**
43: $\quad\quad readlist[q] := (ts, vsuf);$
44: $\quad\quad decided[q] := l;$
45: $\quad\quad$ **if** $\#(readlist) = \lfloor N/2 \rfloor + 1$ **then**
46: $\quad\quad\quad (ts', vsuf') := (0, \langle\rangle);$
47: $\quad\quad\quad$ **for all** $(ts'', vsuf'') \in readlist$ **do**
48: $\quad\quad\quad\quad$ **if** $ts' < ts'' \vee \big(ts' = ts'' \wedge \#(vsuf') < \#(vsuf'')\big)$ **then**
49: $\quad\quad\quad\quad\quad (ts', vsuf') := (ts'', vsuf'');$
50: $\quad\quad\quad pv := pv + vsuf';$
51: $\quad\quad\quad$ **for all** $v \in proposedValues$ **such that** $v \notin pv$ **do**
52: $\quad\quad\quad\quad pv := pv + \langle v \rangle;$
53: $\quad\quad\quad$ **for all** $p \in \Pi$ **such that** $readlist[p] \neq \perp$ **do**
54: $\quad\quad\quad\quad l' := decided[p];$
55: $\quad\quad\quad\quad$ **trigger** $\langle\ fpl, Send \mid p, [\textsc{Accept}, pts, suffix(pv, l'), l', t]\ \rangle;$
56: $\quad\quad$ **else if** $\#(readlist) > \lfloor N/2 \rfloor + 1$ **then**
57: $\quad\quad\quad$ **trigger** $\langle\ fpl, Send \mid q, [\textsc{Accept}, pts, suffix(pv, l), l, t]\ \rangle;$
58: $\quad\quad\quad$ **if** $pl \neq 0$ **then**
59: $\quad\quad\quad\quad$ **trigger** $\langle\ fpl, Send \mid q, [\textsc{Decide}, pts, pl, t]\ \rangle;$

60: **upon event** $\langle\ fpl, Deliver \mid q, [\textsc{Accept}, ts, vsuf, offs, t']\ \rangle$ **do**
61: $\quad t := max(t, t') + 1;$
62: $\quad$ **if** $ts \neq prepts$ **then**
63: $\quad\quad$ **trigger** $\langle\ fpl, Send \mid q, [\textsc{Nack}, ts, t]\ \rangle;$
64: $\quad$ **else**
65: $\quad\quad ats := ts;$
66: $\quad\quad$ **if** $offs < \#(av)$ **then**
67: $\quad\quad\quad av := prefix(av, offs);$ ▷ truncate sequence
68: $\quad\quad av := av + vsuf;$
69: $\quad\quad$ **trigger** $\langle\ fpl, Send \mid q, [\textsc{AcceptAck}, ts, \#(av), t]\ \rangle;$

70: **upon event** $\langle\ fpl, Deliver \mid q, [\textsc{AcceptAck}, pts', l, t']\ \rangle$ **do**
71: $\quad t := max(t, t') + 1;$
72: $\quad$ **if** $pts' = pts$ **then**
73: $\quad\quad accepted[q] := l;$
74: $\quad\quad$ **if** $pl < l \wedge \#(\{p \in \Pi \mid accepted[p] \geq l\}) > \lfloor N/2 \rfloor$ **then**
75: $\quad\quad\quad pl := l;$
76: $\quad\quad\quad$ **for all** $p \in \Pi$ **such that** $readlist[p] \neq \perp$ **do**
77: $\quad\quad\quad\quad$ **trigger** $\langle\ fpl, Send \mid p, [\textsc{Decide}, pts, pl, t]\ \rangle;$

78: **upon event** $\langle\ fpl, Deliver \mid q, [\textsc{Decide}, ts, l, t']\ \rangle$ **do**
79: $\quad t := max(t, t') + 1;$
80: $\quad$ **if** $ts = prepts$ **then**
81: $\quad\quad$ **while** $al < l$ **do**
82: $\quad\quad\quad$ **trigger** $\langle\ asc, Decide \mid av[al]\ \rangle;$ ▷ zero-based indexing
83: $\quad\quad\quad al := al + 1;$