Programming assignment 4 – Consensus

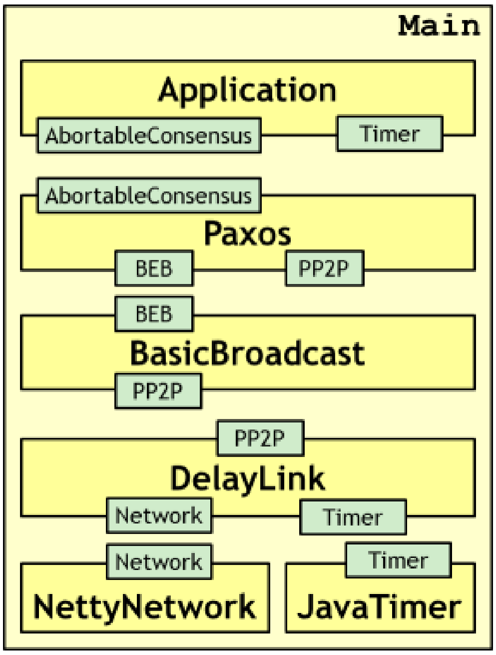
Introduction

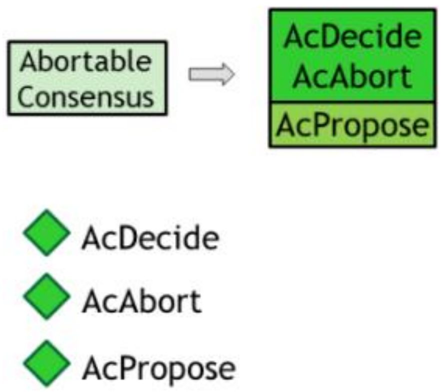
In this programming assignment you shall implement the Paxos component that provides the Abortable Consensus service. The algorithm is available at the end of this document, and it is the algorithm described in the "Paxos Made Simple" paper (the algorithm is not available in the textbook).

Installation

Download id2203-ass4-consensus.zip from the course website, unpack and import into Eclipse in the same way as for previous assignments.

Architecture





The `AbortableConsensus` port is defined in `se.kth.ict.id2203.ports.ac`.

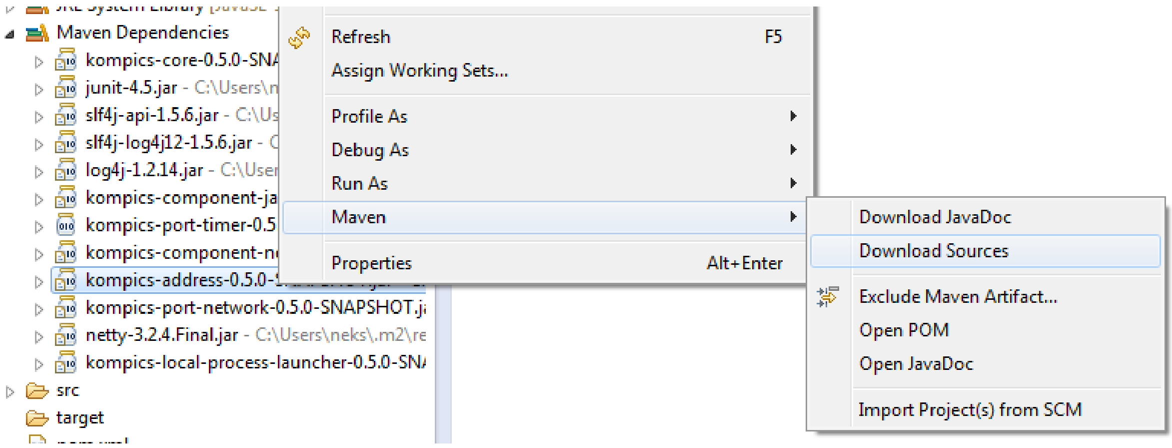
Code to write

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

Various notes

- You need to add the following messages (events) in the component's package:
  - `PrepareMessage` ⊆ `BebDeliver`
  - `PrepareAckMessage` ⊆ `Pp2pDeliver`
  - `NackMessage` ⊆ `Pp2pDeliver`
  - `AcceptMessage` ⊆ `BebDeliver`

- AcceptAckMessage ⊆ Pp2pDeliver
- Remember that you can download the source code for the Kompics classes if you wonder how something in Kompics works. This can easily be done inside Eclipse using Maven:



- The `se.sics.kompics.address.Address` class has well-defined `equals`, `hashCode` and `compareTo` methods, and an object of this class can therefore be used as a key in both a `HashMap` and a `TreeMap`.
- It is a good idea to insert logging statements in the code to be able to trace the execution. For example, in the handler for the `PrepareMessage` event, you might write:

```
logger.info(String.format("Prepare message(source = %s, ts = %d)",
                                                    event.getSource(), event.getTs()));
```

- The algorithm maintains logical clocks. They are used to come up with new, and successively higher, proposal numbers.

Exploration

The Application has two commands for invoking Abortable Consensus:

- `Pn` – Propose value n. The attempt to reach consensus might fail, resulting in an abort, or a value might be decided.
- `Cn` – Propose value n, and if the attempt fails then the application will automatically try to propose again until a value is decided. Please have a look at the `handleAbort` method in `Application.java` to see how this is implemented.

You shall do the following:

- In `Executor.java` there are three scenarios (one should be uncommented at a time), and you should run each of them.
  - In the first scenario, process 1 will propose value 1 alone, and should succeed and get a decision with value 1.
  - In scenario 2, all three processes will propose the values 1, 2 and 3 using command ‘`P`’. Only one process should succeed and get a decision, and the others should abort.
  - In scenario 3, all three processes will propose the values 1, 2 and 3 using the ‘`C`’ command. Processes will content and they will abort each other. After a process has

aborted, it will wait for a random time in the interval [0, 100] ms before proposing the same value again. After some time it is likely that processes will decide.

  - Try to change this time interval (line 123 of `Application.java`) to see how that will affect the time it takes until processes decide.

- For each scenario, make sure that the agreement property is never violated.

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 Paxos: Prepare Phase

Implements:

AbortableConsensus, **instance** *ac*.

Uses:

BestEffortBroadcast, **instance** *beb*;
PerfectPointToPointLinks, **instance** *pp2p*.

upon event $\langle\ ac, Init\ \rangle$ **do**
    $t := 0;$ ▷ logical clock
    $prepts := 0;$ ▷ prepared timestamp
    $(ats, av) := (0, \perp);$ ▷ timestamp and value accepted
    $(pts, pv) := (0, \perp);$ ▷ proposer's timestamp and value
    $readlist := [\perp]^N;$
    $acks := 0;$

upon event $\langle\ ac, Propose \mid v\ \rangle$ **do**
    $t := t + 1;$
    $pts := t \times N + rank(self);$
    $pv := v;$
    $readlist := [\perp]^N;$
    $acks := 0;$
    trigger $\langle\ beb, Broadcast \mid [\textsc{Prepare}, pts, t]\ \rangle;$

upon event $\langle\ beb, Deliver \mid q, [\textsc{Prepare}, ts, t']\ \rangle$ **do**
    $t := max(t, t') + 1;$
    if $ts < prepts$ **then**
        trigger $\langle\ pp2p, Send \mid q, [\textsc{Nack}, ts, t]\ \rangle;$
    else
        $prepts := ts;$
        trigger $\langle\ pp2p, Send \mid q, [\textsc{PrepareAck}, ats, av, ts, t]\ \rangle;$

Algorithm 2 Paxos: Accept Phase

```
upon event ⟨ pp2p, Deliver | q, [NACK, pts′, t′] ⟩ do
    t := max(t, t′) + 1;
    if pts′ = pts then
        pts := 0;
        trigger ⟨ ac, Abort ⟩

upon event ⟨ pp2p, Deliver | q, [PREPAREACK, ts, v, pts′, t′] ⟩ do
    t := max(t, t′) + 1;
    if pts′ = pts then
        readlist[q] := (ts, v);
        if #(readlist) > N/2 then
            (ts, v) := highest(readlist);          ▷ pair with greatest timestamp
            if ts ≠ 0 then
                pv := v;
            readlist := [⊥]^N;
            trigger ⟨ beb, Broadcast | [ACCEPT, pts, pv, t] ⟩;

upon event ⟨ beb, Deliver | q, [ACCEPT, ts, v, t′] ⟩ do
    t := max(t, t′) + 1;
    if ts < prepts then
        trigger ⟨ pp2p, Send | q, [NACK, ts, t] ⟩;
    else
        ats := prepts := ts;
        av := v;
        trigger ⟨ pp2p, Send | q, [ACCEPTACK, ts, t] ⟩;

upon event ⟨ pp2p, Deliver | q, [ACCEPTACK, pts′, t′] ⟩ do
    t := max(t, t′) + 1;
    if pts′ = pts then
        acks := acks + 1;
        if acks > N/2 then
            pts := 0;
            trigger ⟨ ac, Return | pv ⟩;
```