Programming assignment 1 – Eventually perfect failure detector

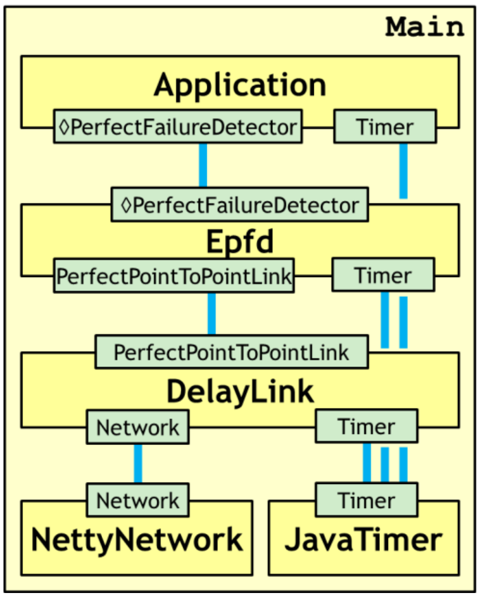
Introduction

In this programming assignment you are tasked with implementing the Eventually Perfect Failure Detector (EPFD) component. The algorithm given at the end of this document should be used; note that this algorithm is slightly modified from algorithm 2.7 in the textbook, by adding sequence numbers.

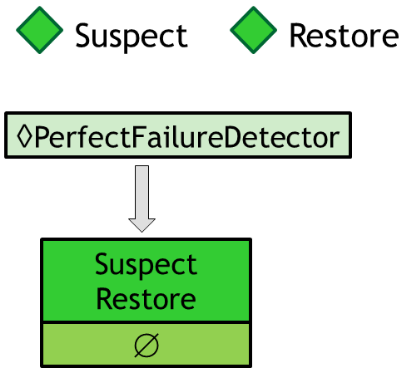
Installation

Download id2203-ass1-epfd.zip from the course website, https://canvas.instructure.com/courses/902299/pages/programming-assignments, unpack and import into Eclipse in the same way as for ass0.

Architecture



All components are provided, except for the Epfd component which you must implement. The Epfd component exposes its service through the EventuallyPerfectFailureDetector port type (the diamond shaped symbol ◊ should be read as “eventually”). It has two indication events: Suspect and Restore.



As is seen in the algorithm pseudo-code, the algorithm uses three “internal” events (events that are not seen outside the component): CheckTimeout, HeartbeatRequestMessage and HeartbeatReplyMessage.

◆ CheckTimeout $\subseteq$ ◆ Timeout

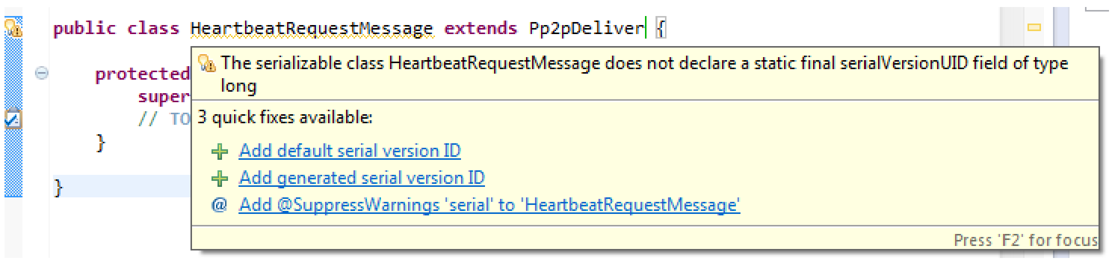
◆ HeartbeatRequestMessage $\subseteq$ ◆ Pp2pDeliver

◆ HeartbeatReplyMessage $\subseteq$ ◆ Pp2pDeliver

You will have to add one file for each of these events (so three files in total) in the `se.kth.ict.id2203.components.epfd` package. In this package there is already a file, `Epfd.java`, in which you shall write the code for the component.

Various notes

- `CheckTimeout` shall inherit from the `Timeout` event-class. You can look at the `ApplicationContinue` class in the `se.kth.ict.id2203.pa.epfd` package for an example.
- `HeartbeatRequestMessage` and `HeartbeatReplyMessage` shall inherit from the `Pp2pDeliver` class. You can look at the `Pp2pMessage` class for an example.
    - Since these two events will be sent over the network they must be serializable, and so every field in the class must be serializable (this is usually the case).
    - They must also have a unique `serialVersionUID` field. If this field is missing then Eclipse will display a yellow squiggly line under the class name, and if you hover the mouse over the class name a context-menu will be shown with the option to add a generated serial version ID.



```
public class HeartbeatRequestMessage extends Pp2pDeliver {

    protected
        super
        // TO
    }

}
```

The serializable class HeartbeatRequestMessage does not declare a static final serialVersionUID field of type long

3 quick fixes available:

- Add default serial version ID
- Add generated serial version ID
- Add @SuppressWarnings 'serial' to 'HeartbeatRequestMessage'

Press 'F2' for focus

- Remember that you are not allowed to trigger events in the component constructor. Instead, subscribe to the `Start` event on the `control` port, and trigger events there.
- You can look in the `doSleep()` function in the Application component for an example of how to start a timer.
- The Epfd component shall create three ports: it provides the EventuallyPerfectFailureDetector port, and requires the PerfectPointToPointLink and the Timer ports (the use of the Timer port is not written explicitly in the algorithm pseudo-code).
- The `java.util.HashSet` class can be used to implement the `alive` and `suspected` sets used in the algorithm. Documentation is available here: http://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html.
- You may use the logger to output debug information which you can see when the program is running. Example: `logger.info("The value of variable x is {}", x);`

Running

The program is started in the same way as ass0, by right-clicking the Executor.java file and clicking Run As -> Java Application.

If there are problems with the component construction code then the application may not be possible to exit using the 'X' command; if this happens, then press **Ctrl-K** instead to exit.

Exploration

In your code you should use the logger to write a message each time the delay variable is increased (at line 9 of the algorithm pseudo-code).

When the program is started the initial delay is set to 1500 milliseconds (ms) and the delta delay is 500 ms (these values are provided in the EpfdInit object, at line 70 of `Main.java`). The delay of the links between the processes is 2000 ms (see line 35 in `Executor.java`).

You shall run the program and verify that:

- Processes will suspect each other to be crashed, and then restore the trust and increase the delay variable.
- After a while, the delay will be larger than the process-to-process round-trip time (twice the link delay), and at this point the processes will not suspect each other any longer.
- After the delay has stabilized, exit one process (to simulate a crash) using the 'X' command, and after a little while the other processes should suspect that process to be crashed.

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.

What to do if you get stuck

Ask questions in the discussion forum on Canvas!

https://canvas.instructure.com/courses/902299/discussion_topics

Please help answer each other's questions! The course staff will look at the forum and answer questions, but if you can help each other that is even better. You are also encouraged to discuss the assignments and programming in Kompics in general with fellow students. However, the final code that you submit must be written by you alone.

Algorithm 1 Increasing Timeout with sequence numbers

Implements:

EventuallyPerfectFailureDetector, **instance** $\Diamond\mathcal{P}$.

Uses:

PerfectPointToPointLinks, **instance** *pp2p*.

```
upon event ⟨ $\Diamond\mathcal{P}, Init$ ⟩ do
    $seqnum := 0$;
    $alive := \Pi$;
    $suspected := \emptyset$;
    $delay := TimeDelay$;
    $startTimer(delay, \textsc{Check})$;

upon event ⟨ $Timeout \mid \textsc{Check}$ ⟩ do
    if $alive \cap suspected \neq \emptyset$ then
        $delay := delay + \Delta$;
    $seqnum := seqnum + 1$;
    for all $p \in \Pi$ do
        if $(p \notin alive) \wedge (p \notin suspected)$ then
            $suspected := suspected \cup \{p\}$;
            trigger ⟨ $\Diamond\mathcal{P}, Suspect \mid p$ ⟩;
        else if $(p \in alive) \wedge (p \in suspected)$ then
            $suspected := suspected \setminus \{p\}$;
            trigger ⟨ $\Diamond\mathcal{P}, Restore \mid p$ ⟩;
        trigger ⟨ $pp2p, Send \mid p, [\textsc{HeartbeatRequest}, seqnum]$ ⟩;
    $alive := \emptyset$;
    $startTimer(delay, \textsc{Check})$;

upon event ⟨ $pp2p, Deliver \mid q, [\textsc{HeartbeatRequest}, sn]$ ⟩ do
    trigger ⟨ $pp2p, Send \mid q, [\textsc{HeartbeatReply}, sn]$ ⟩;

upon event ⟨ $pp2p, Deliver \mid p, [\textsc{HeartbeatReply}, sn]$ ⟩ do
    if $sn = seqnum \vee p \in suspected$ then
        $alive := alive \cup \{p\}$;
```