Time Stamped Anti-Entropy protocol

Author: Joan Manuel Marquès

Distributed Systems course

Spring 2013

3. Phases of the practical assignment

3.2 <u>Phase 2</u>: Implementation of a reduced version of the application and TSAE protocol: only add operation; no purge of log

Implement the TSAE protocol described in section *5.1.2 The timestamped anti-entropy protocol* (from [1]) in the following classes (Package: `recipesService.tsaeDataStructures`):

- `TSAESessionOriginatorSide`: Originator protocol for TSAE (figure 5.7 without acks)
- `TSAESessionPartnerSide`: Partner's protocol for TSAE (figure 5.8 without acks)

`ServerData` class (package `recipesService`) contains `Server`'s data structures required by the TSAE protocol (`log`, `summary`, `ack`) and the application (`recipes`).

- You can add any required method to allow `TSAESessionOriginatorSide` and `TSAESessionPartnerSide` manipulate these data structures.

Use the following methods to send and receive data between servers (package `communication`):

- `readObject`() from `ObjectInputStream`_DS class.
- `writeObject`() from `ObjectOutputStream`_DS class.

Use the following message classes for the communication between partners:

(package `recipesService.communication`)

- `MessageAErequest`: message sent to request an anti-entropy session.
- `MessageOperation`: message sent each time an operation is exchanged during an anti-entropy session.
- `MessageEndTSAE`: message sent to finish an anti-entropy session.

In this phase:

1. Only *add* operations (`addRecipe` method in `ServerData` class) are issued.
2. No purge of Log.

NOTE: be **careful with concurrent access to data structures**. Two actions issued by different threads may interleave. **Use** some **synchronization mechanism to avoid interferences**.

<u>Adaption of pseudocode of figures 5.7 and 5.8</u>

As part of the evaluation of your implementation of the TSAE protocol we will run your solution interacting with the teachers' solution. To make sure that both implementations agree the implementation of `TSAESessionOriginatorSide` and `TSAESessionPartnerSide` classes should follow the following templates.

```
TSAESessionOriginatorSide
      // Send to partner: local's summary and ack
      ...
      Message     msg = new MessageAErequest(localSummary, localAck);
      out.writeObject(msg);

      // receive operations from partner
      msg = (Message) in.readObject();
      while (msg.type() == MsgType.OPERATION){
            ...
            msg = (Message) in.readObject();
      }

      // receive partner's summary and ack
      if (msg.type() == MsgType.AE_REQUEST){
            ...
            // send operations
            ...

            // send and "end of TSAE session" message
            msg = new MessageEndTSAE();
            out.writeObject(msg);

            // receive message to inform about the ending of the TSAE session
            msg = (Message) in.readObject();
            if (msg.type() == MsgType.END_TSAE){
                  //
            }
      }
```

```
TSAESessionPartnerSide
      // receive originator's summary and ack
      msg = (Message) in.readObject();
      if (msg.type() == MsgType.AE_REQUEST){
            ...

            // send operations
            ...

            // send to originator: local's summary and ack
            ...
            msg = new MessageAErequest(localSummary, localAck);
            out.writeObject(msg);

            // receive operations
            msg = (Message) in.readObject();
            while (msg.type() == MsgType.OPERATION){
                  ...
                  msg = (Message) in.readObject();
            }

            // receive message to inform about the ending of the TSAE session
            if (msg.type() == MsgType.END_TSAE){
                  // send and "end of TSAE session" message
                  msg = new MessageEndTSAE();
                  out.writeObject(msg);
                  }
      }
```

Phase 2.1 Your Tasks

Implement the protocol (without acks).

To test if your solution works properly use the provided test environment. Section 4 contains more details about it.

Annex B. Activity simulation and dynamicity

Simulation of communication failures

`ActivitySimulation` class (`package recipesService.activitySimulation`) decides when to simulate the disconnection (or network failure) of a host and when to reconnect it.

To simulate communication failures, a disconnects host abruptly closes all its Input and Output streams, what results in an exception in the two partners that are using the stream.