Students: Joanna C. S. Santos and Yue Hua

1. Domain

It is a distributed application using in programming contests in which we have teams competing to solve algorithmic problems within a period of time. This application has a client-server architecture in which the client software runs in the machine of each team and it is in charge of sending source code from a team to a server. This server has a judgment process that it is going to provide feedback about the correctness of the code. Figure 1 shows a diagram of the system's infrastructure.

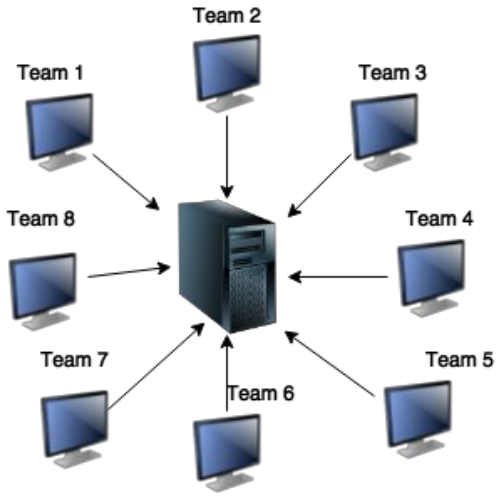

Figure 1. System's infrastructure

2. Recovery Design

To recover from failure, we are using Passive Redundancy combined with Rollback to reintroduce the system to a previous consistent state. Hence, there is a Monitoring process which starts two instances of the Judge process. One instance is in *running* state and the other is *sleeping*. Figure 2 shows a state diagram of the Judge process.

In the `Running` state, the judge has two threads executing concurrently: `Code Evaluator` thread and `Submission Manager` thread. These two threads communicate with each other through a consumer/producer approach. Thus, the `Submission Manager` receives incoming TCP connections, reads the submission message from the socket and adds the message to a FIFO queue. The `Code Evaluator` thread processes each submission request in the queue, which were added by the `Submission Manager`, until it becomes empty.

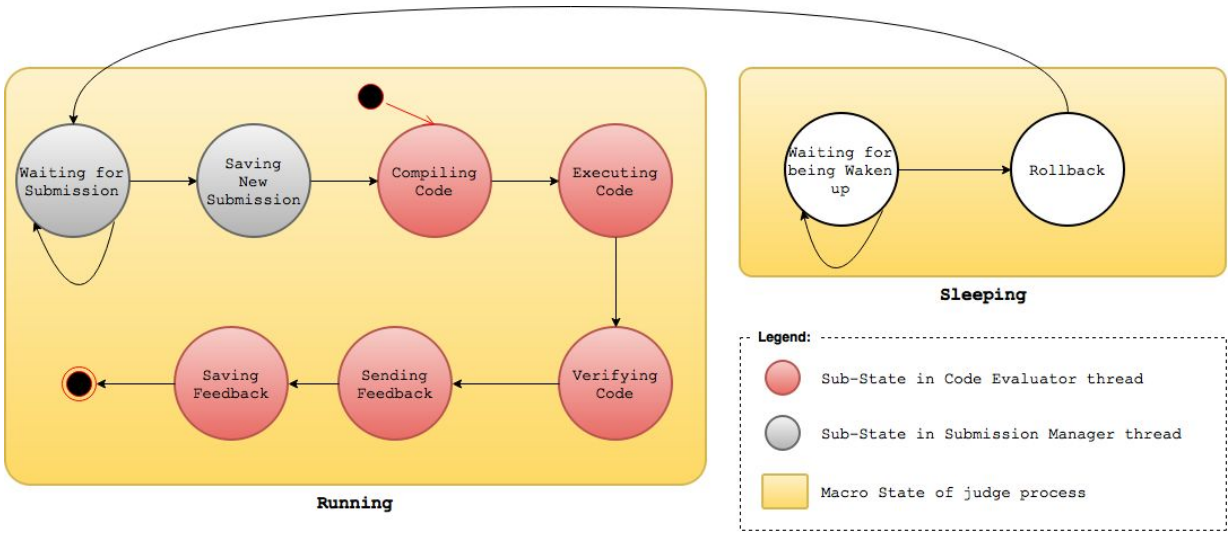

Figure 2. State diagram of the judge process

During the `Sleeping` state, the Judge waits for a request from the Monitor process to be awake. When it receives a request to be waken up, it rollbacks to the previous correct state by accessing the logs of the active instance, which are stored in a SQLite database. After restoring, the passive instance will enter the `Running` state.

Similar to what has been implemented in assignment 1, the communication between the passive instance of the Judge process and the Monitor process occurs through RMI. Therefore, the passive instance binds itself to the RMI registry and the Monitor process retrieves a stub from the registry in order to perform a remote method call to wake up the passive instance. Figure 3 shows the new architecture of the software after adding recovery components to the previous architecture shown in assignment 1.

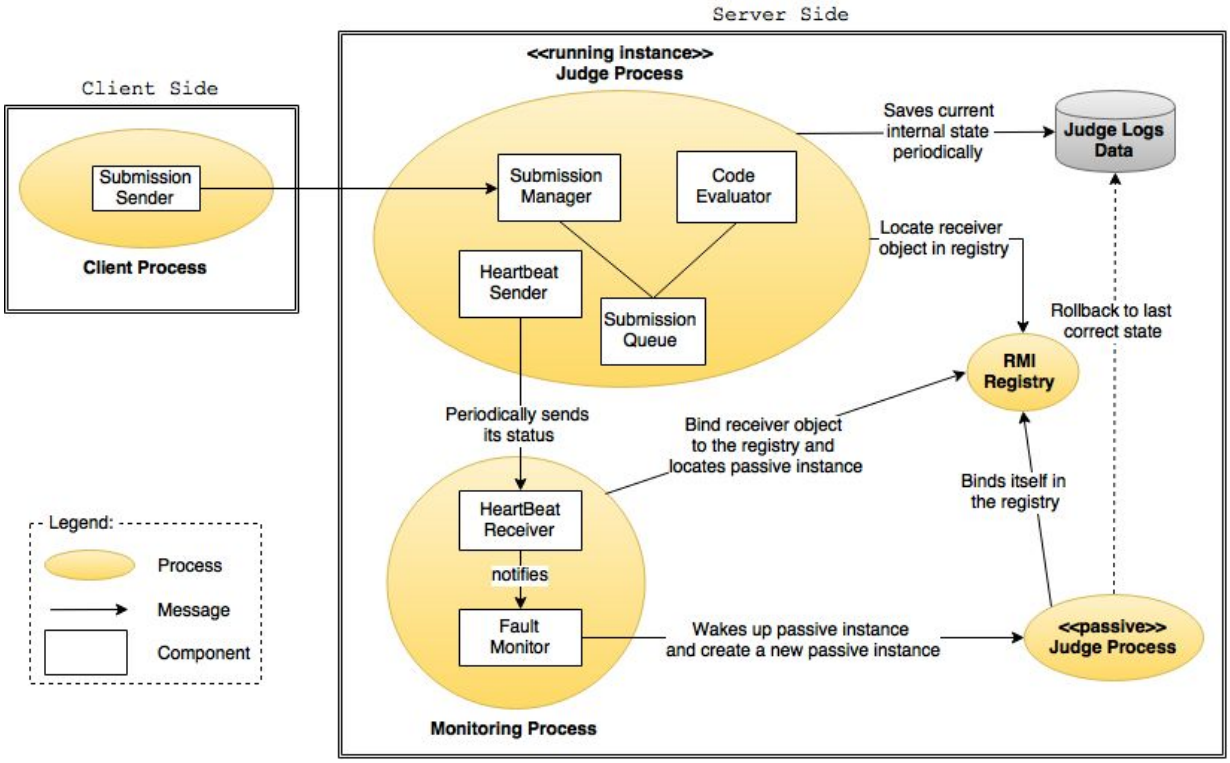

Figure 3. Software Architecture

3. Sample execution

When the server code is started, there is a GUI which shows the outputs of the two Judge instances (in running and sleeping states) and the monitor process (Figure 4).

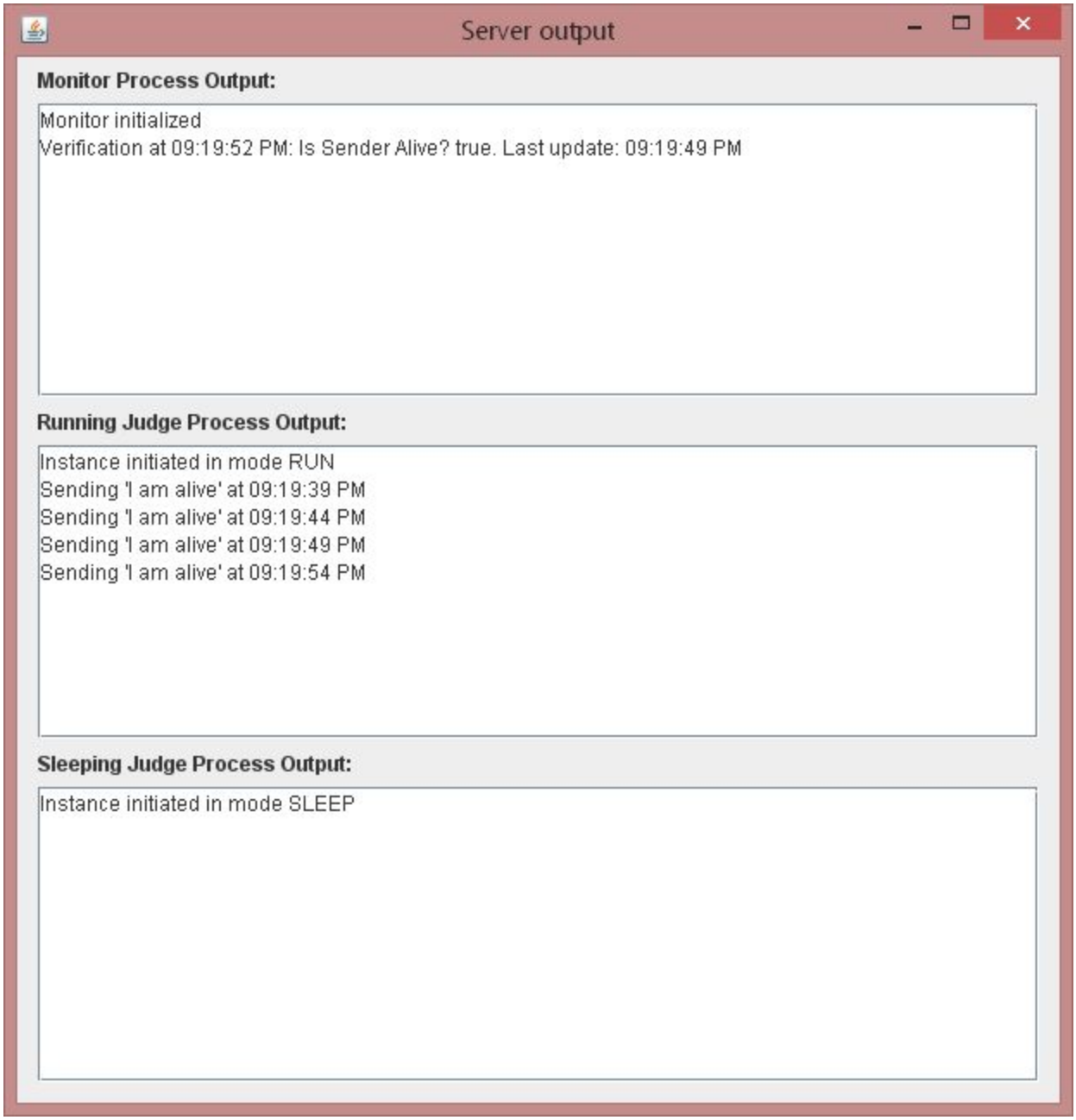

Figure 4. Server-side UI

As shown in Figure 4, the Monitor process firstly prints a message saying that it initialized successfully. The Running instance of the judge prints its state (RUN) followed by logs of the heartbeat messages being sent periodically. Meanwhile, the passive instance of the judge only outputs that it initialized in sleeping mode.

When the client application sends a code to be processed by the running Judge process, the Judge process first saves the new submission data into a SQLite database. After that, it adds the submission to the a queue that is available to the `Code Evaluator` thread to process the submission. When the `Code Evaluator` thread finishes processing the submission and has a

feedback available, it saves the response into the database as well. Figure 5 shows the output messages for checkpointing the internal data of the judge and the response sent back to the client.

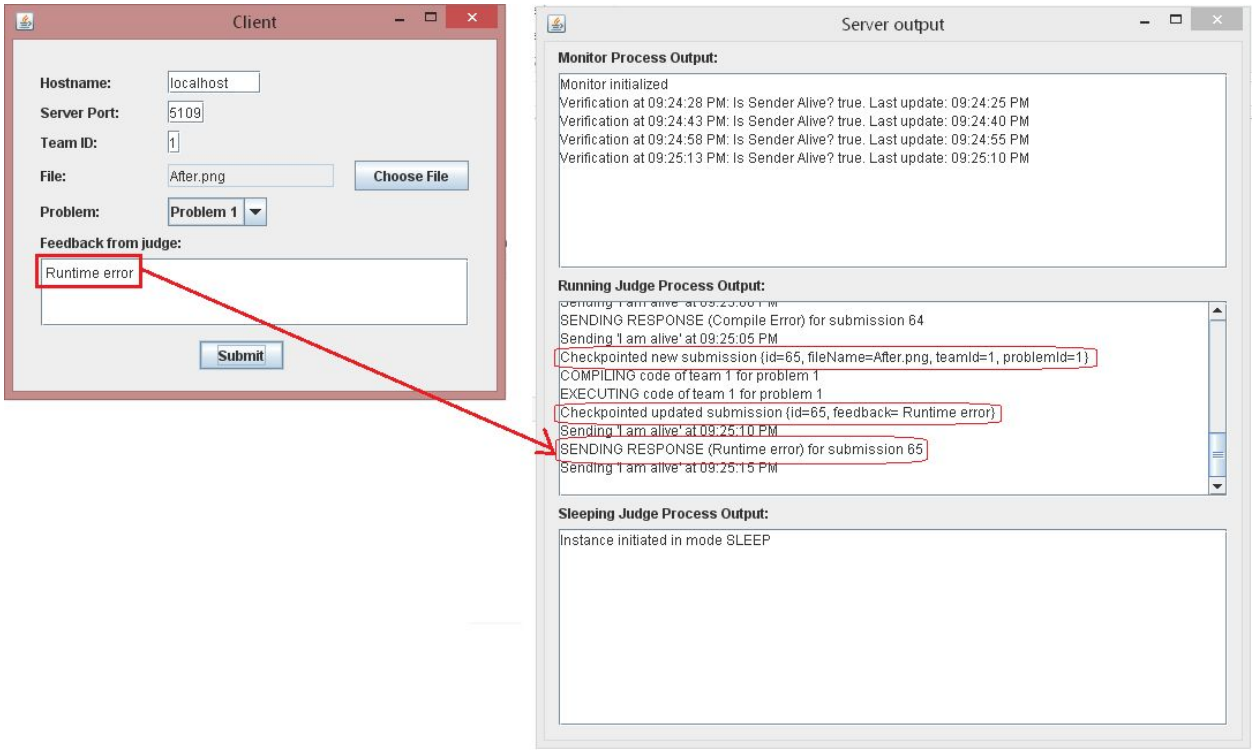

Figure 5. Server-side UI when the client submit a source code

During a simulation of the execution of the source code by the judge, a failure can be randomly injected. If the failure occurs, then the following events happen:

1. `Heartbeat Sender` from the running judge instance stops sending heartbeat messages.
2. `Fault Monitor` detects the failure, since no new heartbeat messages are being received
3. `Fault Monitor` wakes up the passive redundant instance
4. The passive instance retrieves all the uncompleted submissions from the Judge logs stored in the SQLite database (rollback)
5. The passive instance enters the `Running` mode
6. A new passive instance is started (mode = sleeping)
7. The Server-side UI updates the outputs shown. So, the previous passive instance is shown in the second text area now, and the outputs of the new passive instance created are shown in the third text area.

Figure 6 shows an example of the outputs of the three processes executing in the server when a failure occurs.

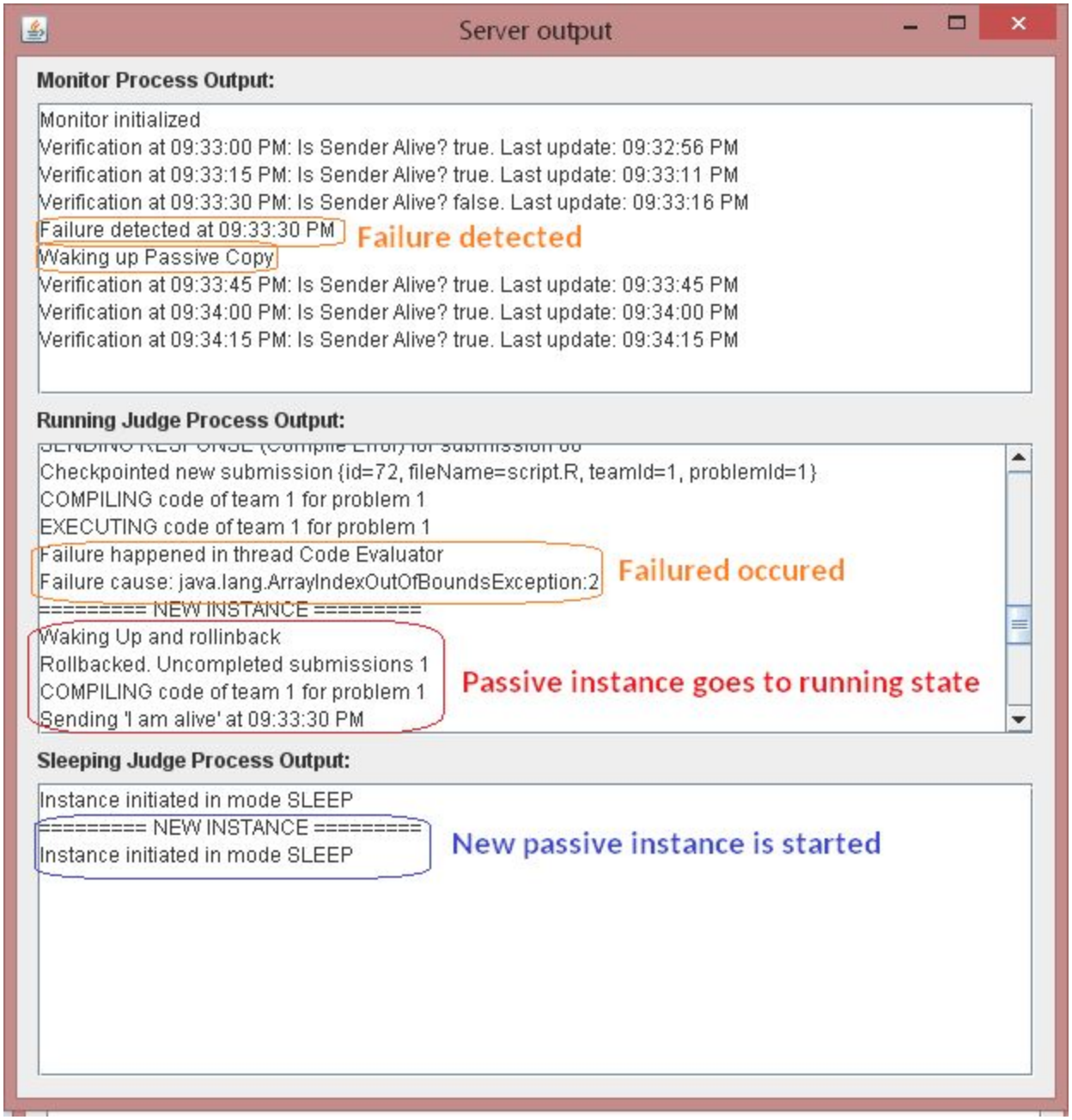

Figure 6. Server-side UI when a failure happens