Practical Assignment III – Quality Attributes

1 Contextualization

Performance, Availability, Scalability and Usability are four of the most relevant Quality Attributes in most of the Software Architectures. The main goal of this assignment is the design and development of a platform the Software Architecture of which relies on those four Quality Attributes.

Each group of students plays the role of a "Software Architect" and the professor plays the role of the unique stakeholder. The stakeholder wrote this document but the Software Architect must be able to exceed the stakeholder's expectations resorting to a constant dialog with him.

1.1 Requirements

The stakeholder needs an infrastructure (cluster of servers) to provide its clients with mathematical computational services. For simplicity, the π (pi) is the only computational service available.

From the elected requirements, the following tactics were selected to be implemented to process the mathematical computations requests.

A. Performance
   1. Replicas of computation:
      a. Computation must be fairly distributed among the active servers;
      b. In this case, computation is equivalent to the number of iterations and not the number of requests;
   2. Concurrency:
      a. each request runs on its own Thread in a server; maximum 3 Threads by server;
      b. Bound queue size: each server accepts a maximum of 2 pending requests;
      c. at any moment, each server accepts simultaneous requests corresponding to a maximum of 20 execution iterations;
   3. Earliest-deadline-first: requests with earliest deadline have higher priority to be processed;

B. Availability
   1. Redundancy: to reallocate among the active servers the requests delegated to a server in case it crashes;
   2. Monitor: to supervise the cluster's status (up/down servers (heartbeat), up/down LB, clients: identification, pending requests, requests being processed, rejected requests, processed requests), etc., etc.).
   3. Select the tactics you need to implement high-availability of Load Balancers:
      a. There will be, always, at least one active Load Balancer and at most two active Load Balancers.
      b. When two Load Balancers are active, anyone of them can be shut down;
      c. When only one Load Balancer is active, another Load Balancer can become active;
      d. At any moment there is at most one LB playing the primary role;

C. Usability

1. All GUI must be driven by Usability for the EVALUATION PROCESS: usage, trace, verification, validation, etc. You can resort to the tactics and anything else you think is necessary. The stakeholder will use the GUI to validate your implementation.

1.2 Constraints

1. Define a Control Center, comprising a GUI, from which all processes of can be created and shut-down (see below, 1.3)
2. Define a Load-Balancer (LB) process comprising a GUI; there can be at most two instances of the LB (two processes).
3. Define a Monitor process comprising a GUI;
4. Each server must implemented as a running process with a GUI;
5. Each client must be implemented as a running process with a GUI;
6. For simplicity, you can consider that all processes have 100% availability and no failures take place, unless they are intentionally killed/shut-down;
7. Requests**: Clients -> LB -> Servers -> LB -> Clients**; all communications are based on TCP/IP sockets.
8. Each client can send any number of requests even if some replies are still pending;
9. For simplicity, consider that there is only one type of request/service (calculation of ∏ (pi)):
    - request: | client id | request id | 00 | 01 | number of iterations | 00 | deadline |
    - reply: | client id | request id | server id | 02 | number of iterations | pi | deadline |
    - reply: | client id | request id | server id | 03 | number of iterations | 00 | deadline |

    client id (Integer>0): unique identification for each client

    request id (Integer>0): unique request identification = 1000*(client id) + increment

    server id (Integer>0): unique identification for each server

    01 (Integer): request code for *pi* calculation

    02 (Integer): reply code for *pi* calculation

    03 (Integer): reply code for request rejection

    iterations (Integer>0): computation of pi takes 4 seconds for each iteration

    pi (Double): value of *pi.* For simplicity, always **3,14159 26535 89793 23846 26433** (one decimal place by iteration, for example if NI = 4, pi = 3.1415)

    deadline (Integer>0): number of seconds to reply when the prioritization scheduler runs.

1.3 Evaluation

In order to evaluate the implementation, some use cases will be used. Hereafter, typical use cases are shown. You are encouraged to enhance each use case type to test your application with different values and contexts. During the demos, the values used can be different.

Use Case I - evaluate if **computation** is fairly distributed across the cluster:

- Environment:
  - N running servers (N = 3)
- Stimulus:
  - M requests (M = 7, with correct NI and low deadline)
- Expected response:
  - The replies show that the distribution is fair

Use Case II – evaluate if requests are processed in individual threads

- Environment:
  - N running servers (N = 1)
- Stimulus:
  - M requests are issued with enough specific decreasing NI (M = 3, with enough NI)
- Expected response:
  - The replies show that the requests are computed by concurrent threads

Use Case III – evaluate maximum simultaneous Threads, queue size and computation (iterations)

- Environment:
  - N running servers (N=1)
- Stimulus:
  - M requests are issued (M > 7, with enough NI)
- Expected response:

    Cluster reacts as expected.

Use Case IV – evaluate prioritization scheduler (earlier-deadline first)

- Environment:
  - N running servers (N=1)
- Stimulus
  - M requests (M = 5, with specific NI and *deadline*)
- Expected response:
  - Replies are received in the correct order

Use Case V – evaluate availability of servers

- Environment:
  - N running servers (N=3)
  - M requests being processed (M = 7, with sufficient NI)
- Stimulus:
  - One server is shut down

- Expected result:
    - The requests running on the shut-down server must be fairly (computation) distributed among the other N-1 servers

Use case VI – evaluate availability of LB

- Environment:
    - Run with different number of LB without restarting the simulation
    - N running servers (N=2)
- Stimulus:
    - Send requests for different environments of LB
- Expected response:
    - Cluster reacts as expected

Use Case VII – GUI evaluation

Each GUI will be evaluated in order to check its state and the information it shows in each use case

- Control Center
    - Interface to launch and shut-down processes (clients, servers, monitor and LBs)
    - Be ambitious, improve the GUI o your CC
- LB
    - Interface to configure the LB
    - Be ambitious, improve the GUI of your LB
- Monitor
    - Interface to configure the Monitor
    - Interface to show the requests being managed by the LB
    - Interface to show the state of all requests being processed by each server
    - Interface to show the state of all servers (UP/DOWN)
    - Interface to show the state of all LB (UP/DOWN)
    - Be ambitious, improve the GUI of your Monitor
- Server:
    - Interface to configure the server
    - Interface to show the details of all requests received
    - Interface to show the details of all processed requests
    - Be ambitious, improve the GUI of your servers
- Client:
    - Interface to configure the client
    - Interface to create new requests
    - Interface to show the pending requests (pending requests >=0)
    - Interface to show the executed requests
    - Be ambitious, improve the GUI of your clients

1.4 Project

Build a unique NetBeans project. Name and root folder: PA3_GY, Y->group id and follow the provided structure published in the e-learning platform. In spite of several UC, each process must be developed to address all the UC simultaneously.

Please do not forget that the GUI of each process type must be carefully designed because the required functionalities (tactics, constraints, use cases, etc.) must be observed and checked through the GUI. This means that it would not be necessary to inspect the source code to check if the requirements are correctly implemented. This does not mean that the source code will not be inspected.

The DEMOS must show that the system works and also that the Architectural Tactics are implemented correctly.

Additionally:

- All Thread Classes must start with the letter T
- All Monitors must start with the letter M, such as MFifo
- Each Monitor must be an individual java class
- All Interfaces must start with the letter I
- The access to Monitors must be through Interfaces provided by their methods
- Threads (Active Entities) that access Monitors must receive as arguments the correspondent interfaces
- All variables must be private (if not, a justification must be provided)
- Whenever possible, Constructors must be declared as private (use a *getInstance* method to instantiate classes and return an interface)
- Variables whose initial value does not change must be declared as final
- The usage of ReentrantLock is mandatory instead of *synchronized methods*
- Methods' and variables' names must be semantically oriented whenever convenient
- You cannot use Maven

1.5 Evaluation

UC 1, UC 4, UC 6:	4.0 points each
UC 2, UC 3, UC 5:	2.0 points each
UC 7:	2.0 points

Each UC must work correctly when tested individually and also when tested in combination with other UC.

1.6 Report

The report must include:

- A detailed description of how each Architectural Tactic was implemented. Must include source code snapshots to prove the description.
- A description of what has not been accomplished, partially and totally.

- by each group element (without this information, your grade will not be provided) who has done what:
    - what he has done in terms of Architectural Tactics
    - the total percentage contribution for the all project.

1.7 Submission

When completed, the report and the project must be compressed (from and including the project root folder: PA3_GY) using **7zip** (mandatory) and the zipped file sent to omp@ua.pt. If you do not comply with these rules, you will be penalized by 2 points.