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 (criteria based on the number of iterations being processed and waiting to be processed in each server) 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
      b. Bound queue size: servers accept a maximum of 2 pending requests.
      c. at any moment, each server accepts simultaneous requests corresponding to a maximum of 20 iterations (pending and on execution)
   3. Earliest-deadline-first: requests with earliest deadline have higher priority to be processed.

B. Availability
   1. Redundancy: to reallocate 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:
         i. there is a default LB which will always be the primary;
         ii. any of the LB 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. Do not forget, the GUI will be used to validate your implementation.

1.2 Constraints

1. Define a Load-Balancer (LB) process comprising a GUI; there can be at most two instances of the LB (two processes).
2. Define a Monitor process (tactic) comprising a GUI;
3. Each Server and each Client must be implemented as a running process with a GUI;
4. Each Client can send any number of request even if they are still being processed.
5. For simplicity, you can consider that all processes have 100% availability and no failures take place - unless they are intentionally killed/shut-down;
6. Requests**: Clients -> LB -> Servers -> Clients**; all communications are based on TCP/IP sockets.
7. 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 5 seconds for each iteration

   pi (Double): value of *pi.* For simplicity, always 3.1415926589793 (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 form the ones here presented.

Use Case I - evaluate if **computation** is fairly (criteria based on the number of iterations being processed and waiting to be processed in each server) distributed across the cluster:

- Environment:
  - N running Servers (N=3)
  - M running Clients (M=2)
- Stimulus:
  - R requests (R = 7, with correct NI and low deadline)
- Expected response:
  - The Replies show that the distribution was fair

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

- Environment:
  - N running Servers (N=1)
  - M running Clients (M=2)
- Stimulus:
  - R Requests are issued with enough specific decreasing NI (M = 3, with enough NI)
- Expected response:
  - The Replies show that the Requests were computed by concurrent threads

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

- Environment:
  - N running Servers (N=1)
  - M running Clients (M=2)
- Stimulus:
  - R Requests are issued (R > 7, with enough NI)
- Expected response:
  - Server reacts as expected rejecting the Requests that exceed its maximum capacity

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

- Environment:
  - N running Servers (N=1)
  - M running Clients (M=1)
- Stimulus
  - R Requests (R=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 running Clients (M=2)
    - R Requests being processed (R = 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 if the cluster is scalable:

- Environment:
    - N running Servers (N=2)
    - M running Clients (M=2)
    - 2*R requests being computed (with enough NI)
- Stimulus:
    - A new server is added
    - A new request is issued
- Expected response:
    - The new Request is deployed in the new added server

Use case VII – 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:
    - Requests are always correctly sent to Servers

Use Case VIII – GUI evaluation

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

- 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_GXX, XX->group id. Create 4 packages, one for each Process: Client, Server, LB and Monitor.

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.

1.5 Evaluation

UC5, UC 6:	3.0 points each
UC1, UC 2, UC 3, UC4, UC7:	2.0 points each
UC 7:	4.0 points

Each UC must work correctly when tested individually and also when tested in combination with other UC. After completing a demo of a UC, it should not be necessary to reboot either the clients or the servers for the next UC demo.

1.6 Submission

When completed, a report and the project must be compressed (from and including the project root folder: PA3_GXX) using the 7z format (mandatory) and sent to omp@ua.pt.

The report must include:

- What is implemented and working correctly
- What is not implemented or not working correctly
- The contribution of each element group in %

If you do not comply with these rules, you will be penalized with 2 points.