UNIVERSITÉ Concordia UNIVERSITY

GINA CODY SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

Distributed System Design

COMP 6231 – Winter 2020

Concordia University

Department of Computer Science and Software Engineering

Instructor: R. Jayakumar

Distributed Event Management System(DEMS) – Group Project

Group members:

Sepehr Jalayer – 40126236 – jalayersepehr@gmail.com

Deep Shivam Gautam – 40138490 – deepshivamgautam@gmail.com

Seyed Pouria Zahraei – 40115175 – Zpouria@gmail.com

Table of Contents

Overview

The distributed Event management system (DEMS) consists of three different servers which are located in different cities:

- Montreal(MTL)
- Sherbrooke(SHE)
- Quebec(QUE)

The clients of this system are of two types:

- eventManagers
- customers

We must ensure that these clients are connected to their own servers with Java RMI, and also the connection between our three servers are done through UDP/IP socket programming.

Manager specific functions:

- **addEvent():** managers can only add events in their own server
- **removeEvent():** managers can only remove events from their own server. *if an event was removed we must book another closest event for the customers registered in that event. !!Needs **UDP** for server-server connection.
- **listEventAvailability():** we must gather all events of a given type from all three servers. !!Needs **UDP** for server-server connection.

Client/Manager functions:

- **bookEvent():** customers can also book from other servers with a weekly 3 limit. !!Needs **UDP** for server-server connection.
- **getBookingSchedule():** show the customers booking schedule.
- **cancelEvent():** clients can remove an event from their own schedule. !!Needs **UDP** for server-server connection.
- **swapEvent():** clients can swap a booked event with another event. (a bookEvent + cancelEvent) -> needs to be atomic

Clients are recognized with their ClientID (8 character): serverID (3char) + clientType(C/M) + 4 digit identifier.

Events are recognized with their eventType: Conferences/Seminars/Trade Shows + their eventID(10 character): serverID (3char) + eventSlot (M/A/E) + eventDate (DDMMYY).

Design Architecture

There will be three replica managers (RM) each containing different server implementations of our three servers and a database of that replica.

This document explains how high availability or fault tolerance distributed event management system is achieved. This system needs to fulfil the following criteria:

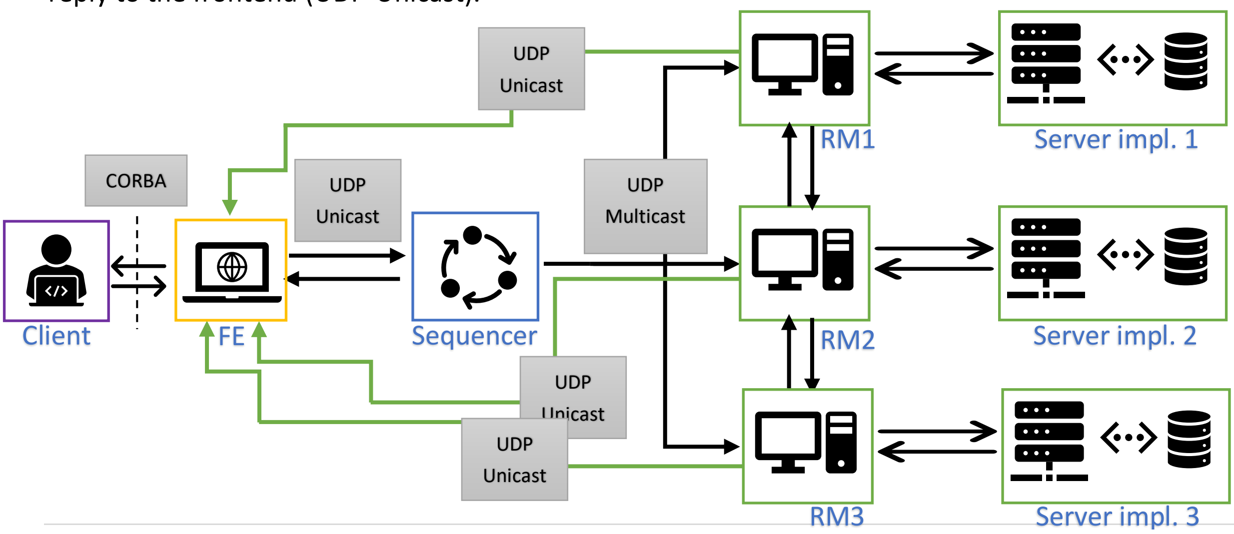
1. Data Consistency
2. Total Ordering
3. Dynamic Timeout
4. Fault tolerance (software bug)
5. UDP Multicast
6. UDP Reliability
7. Replica Recovery

Highly Available or Fault Tolerant Distributed Event Management System will be implemented over the CORBA project implemented as part of assignment2.

- We will implement a Front-End (FE) module which receives a client request as a CORBA invocation, forwards the request to the sequencer, receives the results from the replicas and sends a single correct result back to the client as soon as possible. The FE also informs all the RMs of a possibly failed replica that produced incorrect result.
- We will implement the replica manager (RM) which creates and initializes the actively replicated server system. The RM also implements the failure detection and recovery for the required type of failure.
- We will implement a failure-free sequencer which receives a client request from a FE, assigns a unique sequence number to the request and reliably multicast the request with the sequence number and FE information to all the three server replicas.

A simple request – response flow is mentioned below:

Front-End module would accept the Client Request (CORBA invocation). Front End will then send the request to the sequencer (UDP-Unicast) and the sequencer send the messages with a sequence id to replica managers (UDP-Multicast). It requires totally ordered and reliable multicast so that all RMs perform the same operations in the same order. Then RMs will process each request identically and send it to the corresponding servers. Specific servers are going to execute that request and going to reply to the frontend (UDP-Unicast).



1.Data Consistency

In order to achieve data consistency, we need to make sure that:

- Every request is executed by all or none of the replicas. (UDP Reliability, Dynamic Timeout)
- Every request is executed in the same order in all the replicas. (Total Ordering)

2.Total Ordering

Total ordering can be achieved using sequence numbers attached by the sequencer to every request. We use FIFO for the order of the processing.

Also, in case of a request lost, any RM receiving new request with any sequence number will check if the new sequence number is subsequent to the last executed request, if not; it will ask other RMs for the missing sequence numbers.

Other things that will help total ordering are:

- Dynamic timeout
- UDP Reliability
- UDP Multicasting

3.Dynamic Timeout

Each FE process initializes with a constant timeout (e.g.: 10 seconds). Then after execution of the first request it will change the timeout to the longest response time multiplied by 2.

If timeout reaches and FE does not get response from a server after 3 times, it will conclude that the server has crashed and informs other RMs to begin Recovery of that replica.

If timeout reaches and FE does not get response from any of the RMs, it will conclude that the request was lost and did not reach the RMs, therefore sends the request again(with the same sequence number).

4.Fault tolerance (software bug)

This is achieved through Front-End. The FE gathers the responses from all 3 RMs. It then decides the correctness simply by taking the majority. If an RM send 3 wrong responses, FE notifies all the RMs of the faulty RM in order to take the necessary actions.

Also each replica should be connected to a unique server implementation which includes all the methods and the data for Montreal, Quebec, Sherbrooke.

5.UDP Multicast

To decrease the number of requests send through the network, the connection between the sequencer and the RMs, also the connection between RMs are sent through UDP-Multicast.

6.UDP Reliability

UDP by itself is unreliable, therefore, we need to make the necessary steps to make it reliable.

- Timeout-resend: By the FE in order to overcome the request lost
- Multicasting each new request received by an RM to other RMs.
- If a new request received by an RM was not subsequent to the last executed request(there might have been a/some request loss), it will ask other RMs for the missing request if exists and then runs it/them before the new request.

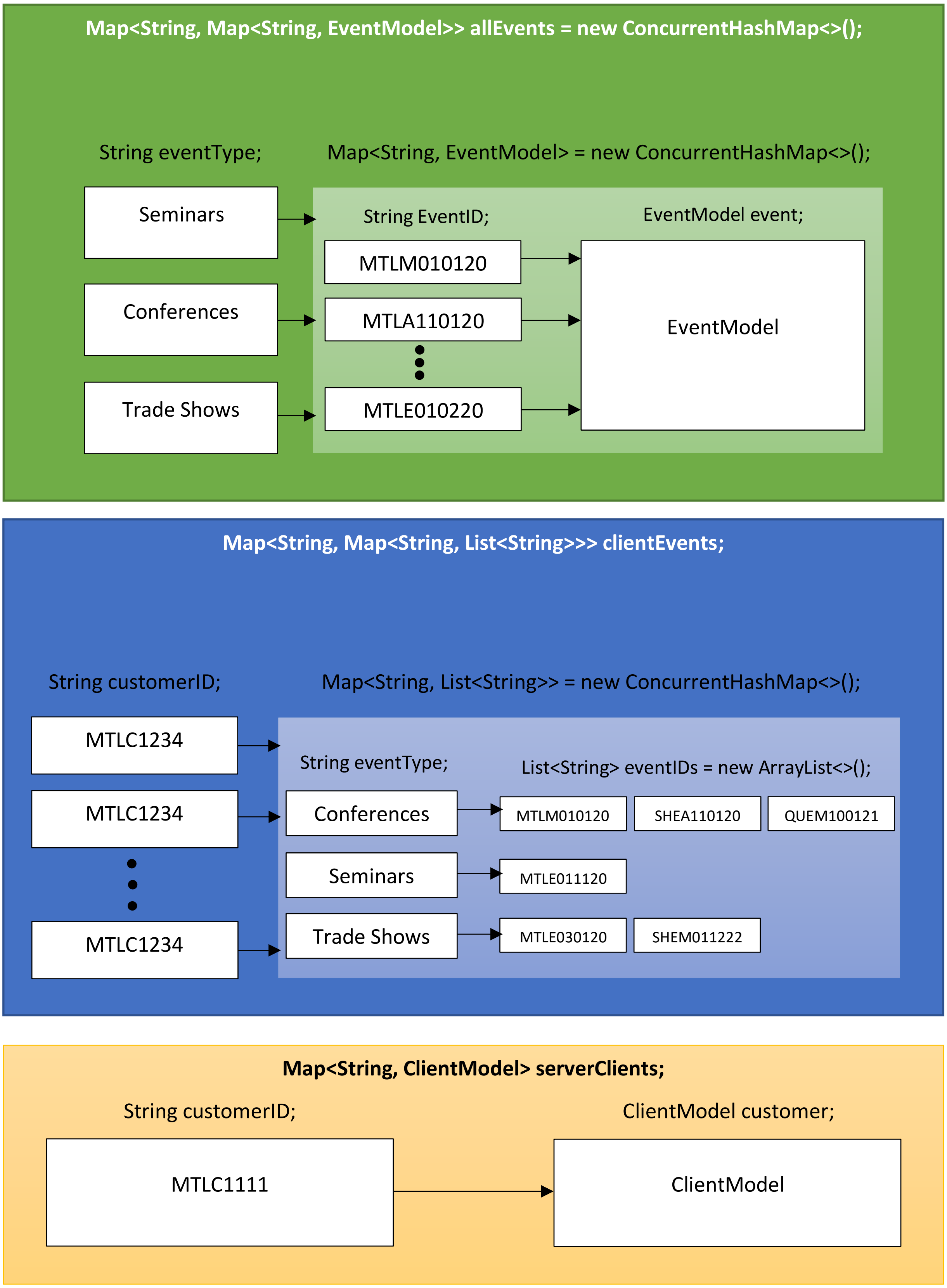
7.Replica Recovery

In order to achieve high-availability we need to ensure that a server crash can be tolerated. To do so, whenever a crash was detected (by FE) after not receiving a response from an RM for 3 times, or (by other RMs) after not receiving a heartbeat for some time (e.g.: 10 seconds) they will initiate a replica recovery for that RM.

In the replica recovery mode/state: the RM, asks other RMs for all the processed requests up to that point of recovery, then it will execute them one by one. After finishing the list of past requests, the replica is up and running.

Data Structures

All the data is maintained within each server, using three Map structures shown in the figure below.



Test Scenarios

#	Type of Test	Scenario	Cases
1	Login	UserName	1.Event Manager ID 2.Customer ID
2	Menu Items		
3		Logout	1.Log out menu Item
4	Event Manager	addEvent()	1.invalid EventID -> not added 2.new EventID -> added 3.Existing EventID (LowerCapacity) -> not allowed 4.Existing EventID (HigherCapacity) -> capacity Updated 5.Duplicate Event -> not happening 6.EventID of Other Servers -> not allowed
5		removeEvent()	1.invalid EventID 2.EventID not exist 3.Event without anyone registered -> removed event 4.Event with someone registered -> Removed event + registered to same eventType if possible (UDP if needed) 5.EventID in other servers -> not allowed
6		listEventAvailability()	1.list all events of a given type from all three servers (UDP needed) 2.Event type is forced correctly with showing only options available
7		Ask for customerID	1.Access Customer methods
8	Event Manager + Customer	bookEvent()	1.on own server -> allowed 2.if event is full -> not allowed 3.on other servers -> only three in a week (UDP needed) 4. invalid EventID -> not allowed
9		getBookingSchedule()	1.Show booking schedule of customer 2.invalid customerID -> not allowed 3.customer not exist ->ok
10		cancelEvent()	1.cancel on own server -> ok 2.cancel on other server -> ok(UDP needed) 3.cancel a not registered event -> error shown 4.invalid eventide -> not allowed

11		swapEvent()	1.swap with a non-existent event -> not allowed 2.swap a not registered event -> not allowed 3.swap with a event from other server -> same conditions as a book event 4.swap with an event from other server in same week (weekly limit reached)

Test Cases

Test cases will be added after the implementation finished.