Distributed System Design

COMP 6231 – Winter 2020

Concordia University

Department of Computer Science and Software Engineering

Instructor: R. Jayakumar

Distributed Event Management System(DEMS) – Assignment 3 - WebServices

By:

Sepehr Jalayer (40126236)

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).

**Both servers and client maintain log files stored in the project directory.

Implementation

- Client – Server communication is done by SOAP based Webservice
  - Montreal service address: http://localhost:8080/montreal?wsdl
  - Sherbrook service address: http://localhost:8080/sherbrook?wsdl
  - Quebec service address: http://localhost:8080/quebec?wsdl
  - @WebService(endpointInterface = "com.web.service.WebInterface")
- Server – Server communication is done via UDP/IP Socket programming
  - Montreal UDP port: 8888
  - Quebec UDP port: 7777
  - Sherbrook UDP port: 6666
- To reduce the duplication code and facilitate changes and debugging we used single server implementation file and single interface implementation file.
- Both Server and Client maintain separate logfiles
- Server Log files are located under \project_directory\src\Logs\Server\serverName.txt
- Client Logs are located under \project_directory\src\Logs\Client\ClientID.txt
- We used concurrentHashMaps to store the data, to ensure maximum concurrency.
- We used synchronized blocks and methods in some cases to ensure thread safe operation
- The Most important part of the implementation was avoiding infinite loops in UDP calls specially in removeEvent() and listEventAvailability() methods.
- The Hardest method to implement was the removeEvent() when there were clients registered in the event and we some of them were from other servers.
- For the atomicity of the swap method, we booked the newEvent first (somewhat similar to reservation) then if it was a success -> we canceled our oldEvent. And if the cancel was not successful -> we canceled our formerly booked newEvent (cancelReservation)
- We added a shutdown() method for shutting down the ORB
- Added a test concurrency to client to check how our database is thread-safe or not? We concurrently request 5 book/cancel events to an event with 2 capacity

WebInterface

```java
package com.web.service;

import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

@WebService
@SOAPBinding(style = SOAPBinding.Style.RPC)
public interface WebInterface {
    /**
     * Only manager
     */
    public String addEvent(String eventID, String eventType, int bookingCapacity);

    public String removeEvent(String eventID, String eventType);

    public String listEventAvailability(String eventType);

    /**
     * Both manager and Customer
     */
    public String bookEvent(String customerID, String eventID, String eventType);

    public String getBookingSchedule(String customerID);

    public String cancelEvent(String customerID, String eventID, String eventType);

    public String swapEvent(String customerID, String newEventID, String newEventType, String oldEventID, String oldEventType);

}
```

Figure 1 – WebInterface.java

Class Diagram

SOAPBinding
- style() : Style
- use() : Use
- parameterStyle() : ParameterStyle

WebService
- name() : String
- targetNamespace() : String
- serviceName() : String
- portName() : String
- wsdlLocation() : String
- endpointInterface() : String

EventModel
- detectEventServer(String) : String
- detectEventTimeSlot(String) : String
- detectEventDate(String) : String
- addRegisteredClientID(String) : int
- removeRegisteredClientID(String) : boolean
- toString() : String

- eventType : String
- eventID : String
- eventRemainCapacity : int
- eventServer : String
- registeredClientIDs : List<String>
- eventCapacity : int
- eventDate : String
- registeredClientsIDs : List<String>
- full : boolean
- eventTimeSlot : String

Client
- main(String[]) : void
- init() : void
- startConcurrencyTest() : void
- getServerID(String) : String
- checkUserType(String) : int
- customer(String) : void
- manager(String) : void
- askForCustomerIDFromManager(String) : String
- printMenu(int) : void
- promptForEventType() : String
- promptForEventID() : String
- promptForCapacity() : int

WebInterface
- addEvent(String, String, int) : String
- removeEvent(String, String) : String
- listEventAvailability(String) : String
- bookEvent(String, String, String) : String
- getBookingSchedule(String) : String
- cancelEvent(String, String, String) : String
- swapEvent(String, String, String, String, String) : String

EventManagement
- getServerPort(String) : int
- addTestData() : void
- addEvent(String, String, int) : String
- removeEvent(String, String) : String
- listEventAvailability(String) : String
- bookEvent(String, String, String) : String
- getBookingSchedule(String) : String
- cancelEvent(String, String, String) : String
- swapEvent(String, String, String, String, String) : String
- removeEventUDP(String, String, String) : String
- listEventAvailabilityUDP(String) : String
- sendUDPMessage(int, String, String, String, String) : String
- getNextSameEvent(Set<String>, String, String) : String
- exceedWeeklyLimit(String, String) : boolean
- addCustomersToNextSameEvent(String, String, List<String>) : void
- eventExists(String, String) : boolean
- isEventOfThisServer(String) : boolean
- checkClientExists(String) : boolean
- clientHasEvent(String, String, String) : boolean
- removeEventIfExists(String, String, String) : boolean
- addCustomerAndEvent(String, String, String) : void
- addEventTypeAndEvent(String, String, String) : void
- isCustomerOfThisServer(String) : boolean
- onTheSameWeek(String, String) : boolean
- addNewEvent(String, String, int) : void
- addNewCustomerToClients(String) : void

- ORB : ORB
- clientEvents : Map<String, Map<String, List<String>>>
- serverClients : Map<String, ClientModel>
- allEvents : Map<String, Map<String, EventModel>>

Logger
- clientLog(String, String, String, String) : void
- clientLog(String, String) : void
- serverLog(String, String, String, String, String) : void
- serverLog(String, String) : void
- deleteALogFile(String) : void
- getFileName(String, int) : String
- getFormattedDate() : String

ClientModel
- detectClientServer() : String
- detectClientType() : String
- toString() : String

- clientID : String
- clientServer : String
- clientType : String

ServerInstance
- listenForRequest(EventManagement, int, String, String) : void
- addTestData(EventManagement) : void

TestData
- main(String[]) : void
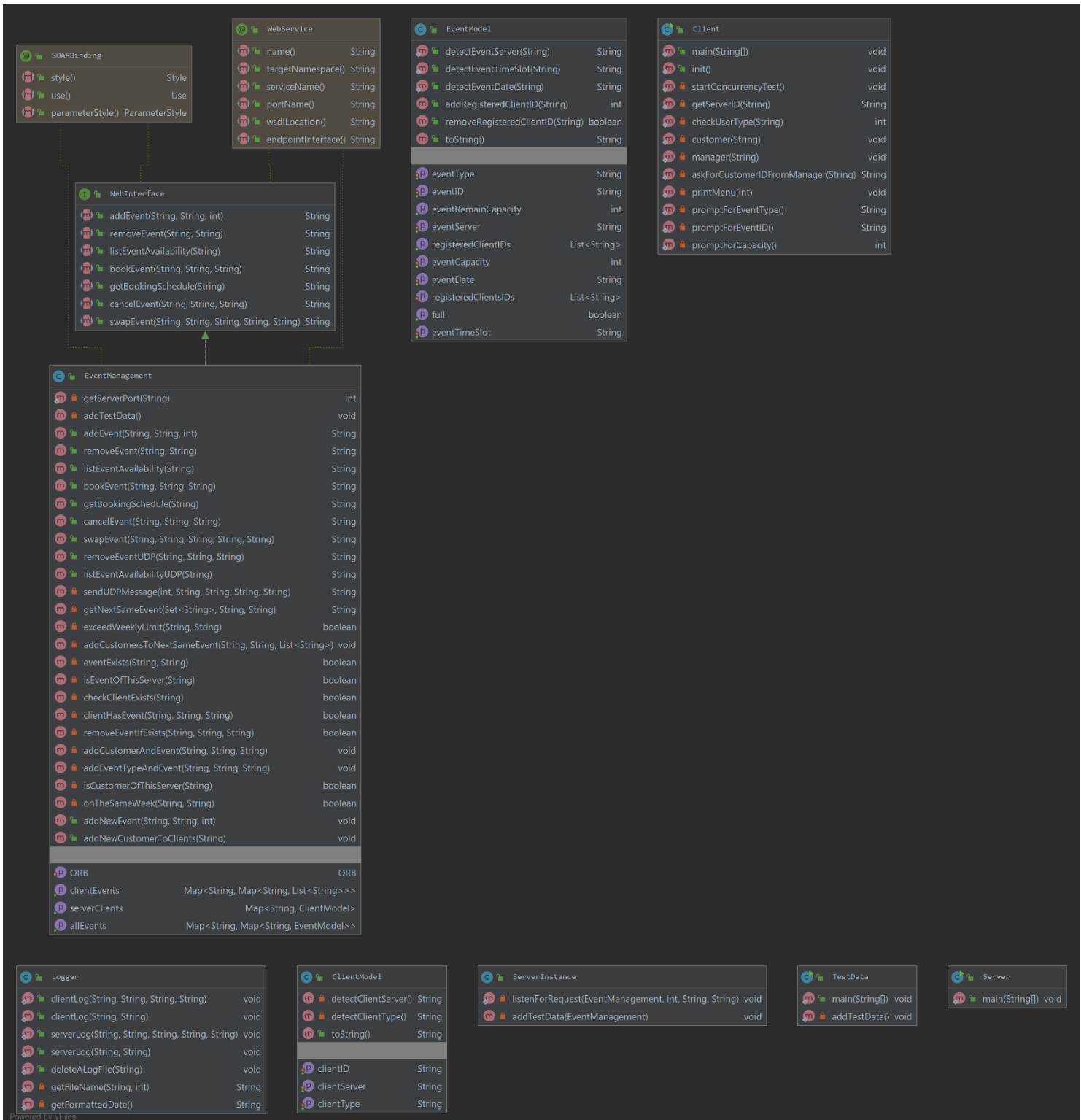- addTestData() : void

Server
- main(String[]) : void

Figure 2 - Class Diagram with properties

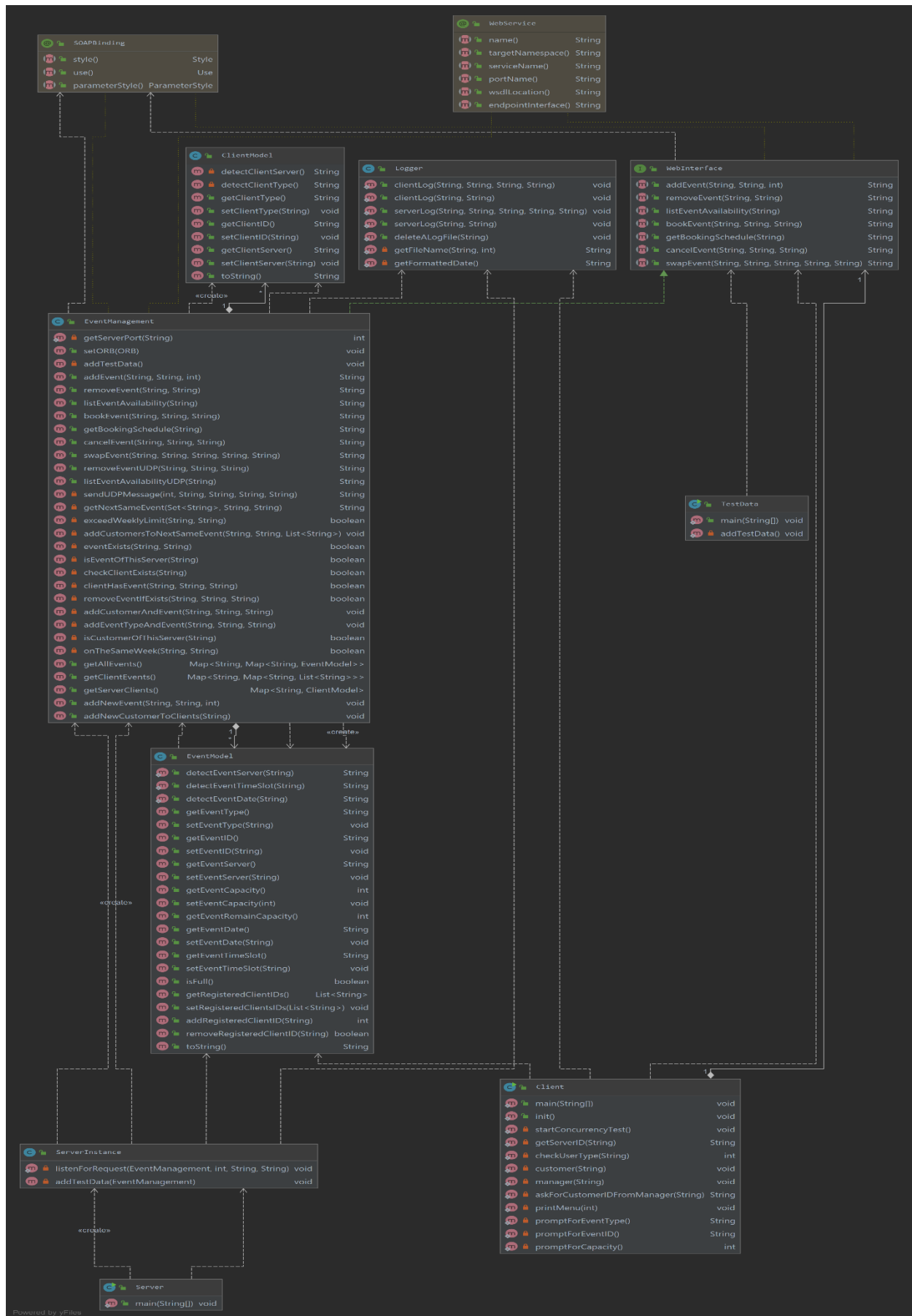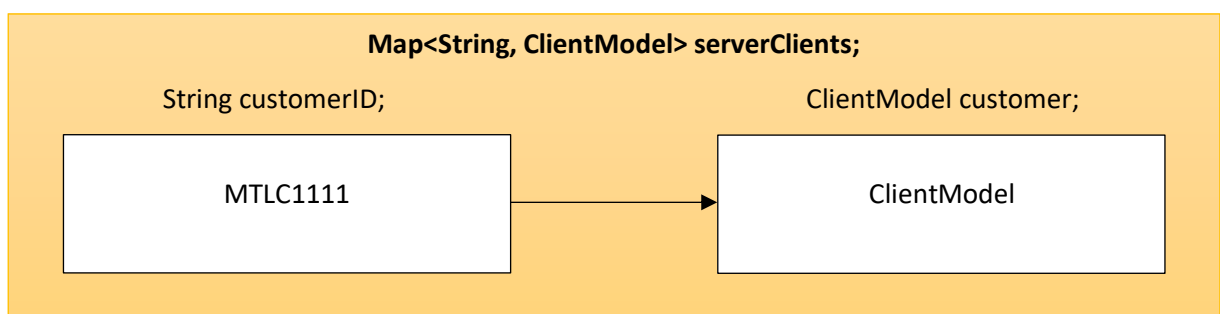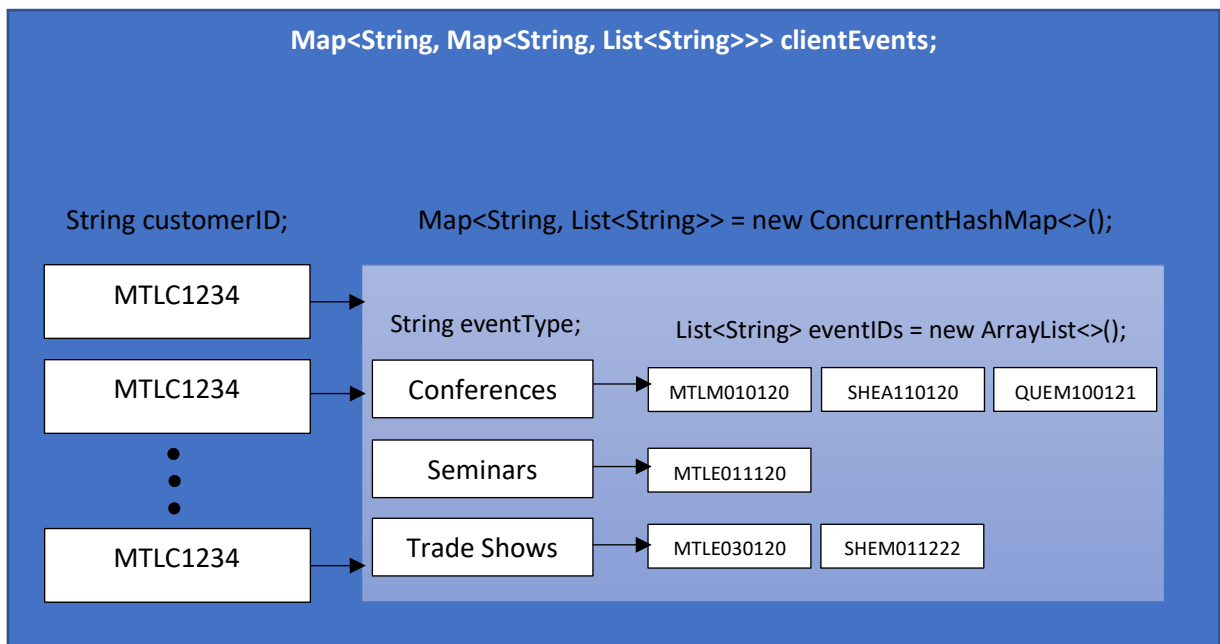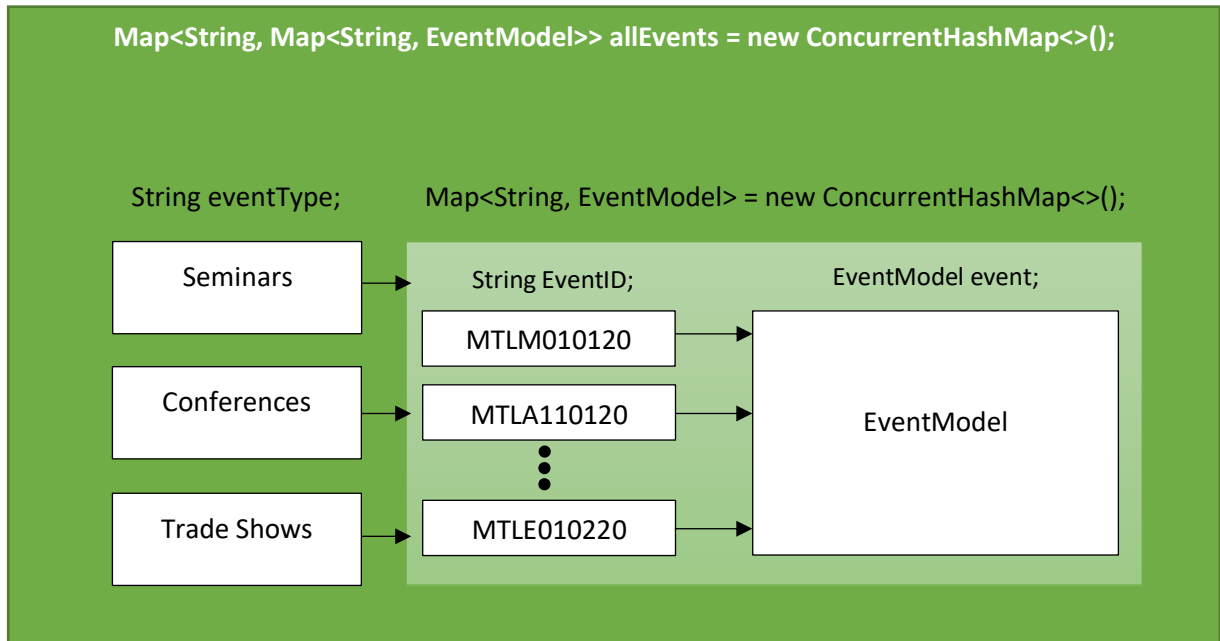**The Full dependencies and methods of each class is shown below.

Figure 3 - Class Diagram with full dependencies

Data Structures

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

Map<String, Map<String, EventModel>> allEvents = new ConcurrentHashMap<>();

String eventType;   Map<String, EventModel> = new ConcurrentHashMap<>();

| Seminars |

| Conferences |

| Trade Shows |

String EventID;   EventModel event;

| MTLM010120 |

| MTLA110120 |

| MTLE010220 |

| EventModel |

Map<String, Map<String, List<String>>> clientEvents;

String customerID;   Map<String, List<String>> = new ConcurrentHashMap<>();

| MTLC1234 |

| MTLC1234 |

| MTLC1234 |

String eventType;   List<String> eventIDs = new ArrayList<>();

| Conferences | | MTLM010120 | | SHEA110120 | | QUEM100121 |

| Seminars | | MTLE011120 |

| Trade Shows | | MTLE030120 | | SHEM011222 |

Map<String, ClientModel> serverClients;

String customerID;   ClientModel customer;

| MTLC1111 | | ClientModel |

Test Scenarios

#	Type of Test	Scenario	Cases
1	Login	UserName	1.Event Manager ID
2	Menu Items		2.Customer ID
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.new event has no capacity - Status: false
		2.Old event doesn't exists, and given new Event ID exists - Status: false
		3.Old event exists, and given new Event ID doesn't exists - Status: false
		4.old eventID city equals to users city new eventID city equals to users city happening in same week - Status: true
		5.old eventID city equals to users city new eventID city equals to users city Not happening in same week - Status: true
		6.old eventID city not equals to users city new eventID city equals to users city happening in same week - Status: true
		7.old eventID city not equals to users city new eventID city equals to users city not happening in same week - Status:true
		8.old eventID city equals to users city new eventID city not equals to users city happening in the same week. Limit == 3 - status: false
		9.old eventID city equals to users city new eventID city not equals to users city happening in the same week

		Limit < 3 - Status:true
		10. old eventID city equals to users city new eventID city not equals to users city not happening in same week. limit == 3 - Status:false
		11. old eventID city equals to users city new eventID city not equals to users city not happening in same week. limit <3 - Status: true
		12. old eventID city not equals to users city new eventID city not equals to users city happening in the same week. limit < 3 - Status:true
		13. old eventID city not equals to users city new eventID city not equals to users city happening in the same week. limit == 3 - Status:true
		14. old eventID city not equals to users city new eventID city not equals to users city not happening in the same week. limit < 3 - Status:true
		15. old eventID city not equals to users city new eventID city not equals to users city not happening in the same week. limit == 3 - Status:false