Distributed System Design

COMP 6231 – Winter 2020

Concordia University

Department of Computer Science and Software Engineering

Instructor: R. Jayakumar

Distributed Event Management System – Assignment 1: Using Java RMI

By: Sepehr Jalayer

Concordia ID: 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.

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 over RMI

  - Montreal RMI Registry port: 2964

  - Quebec RMI Registry port: 2966

  - Sherbrook RMI Registry port: 2965

- 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.

- 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.
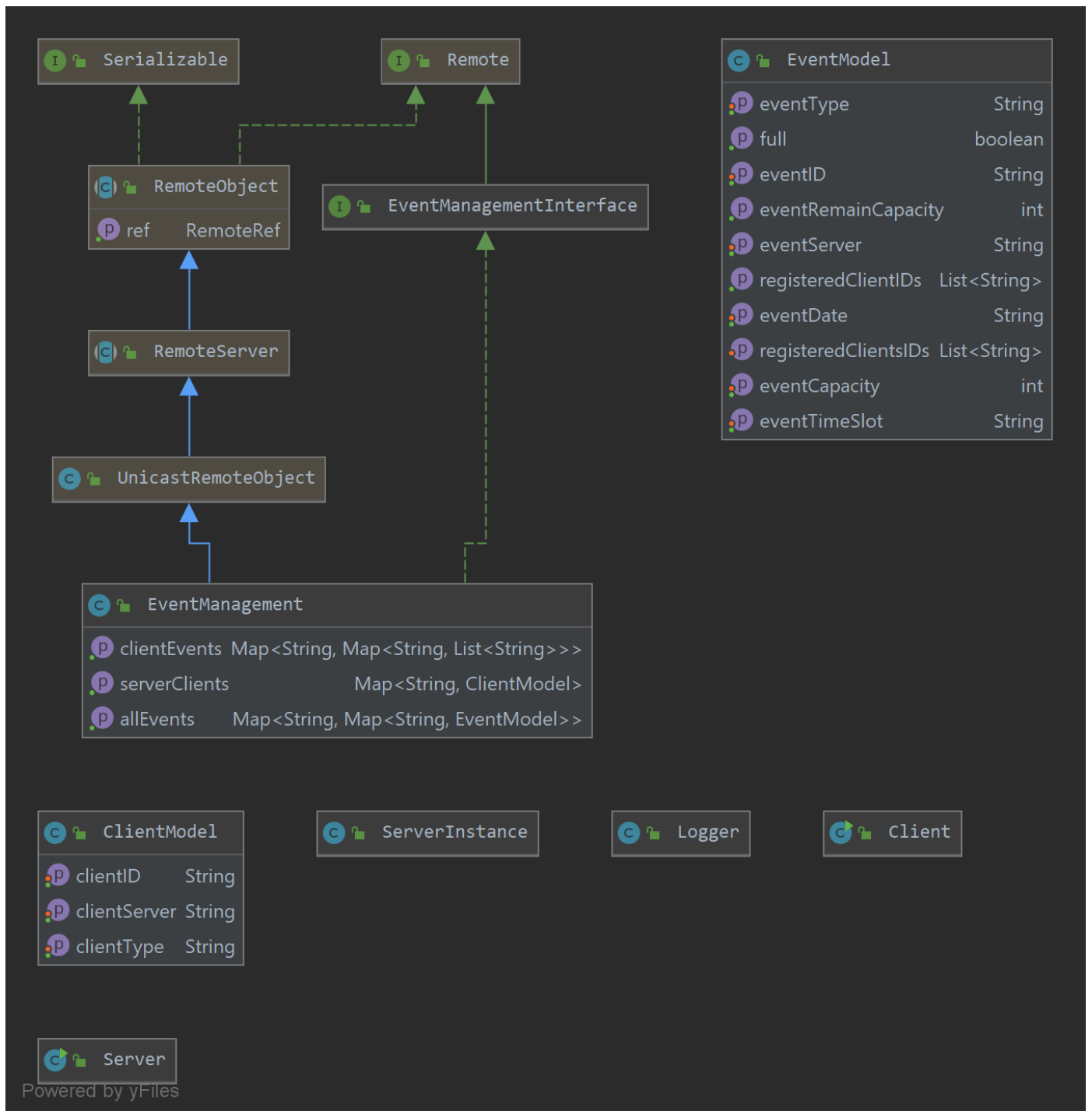
Class Diagram

Figure 1-Class Diagram with properties

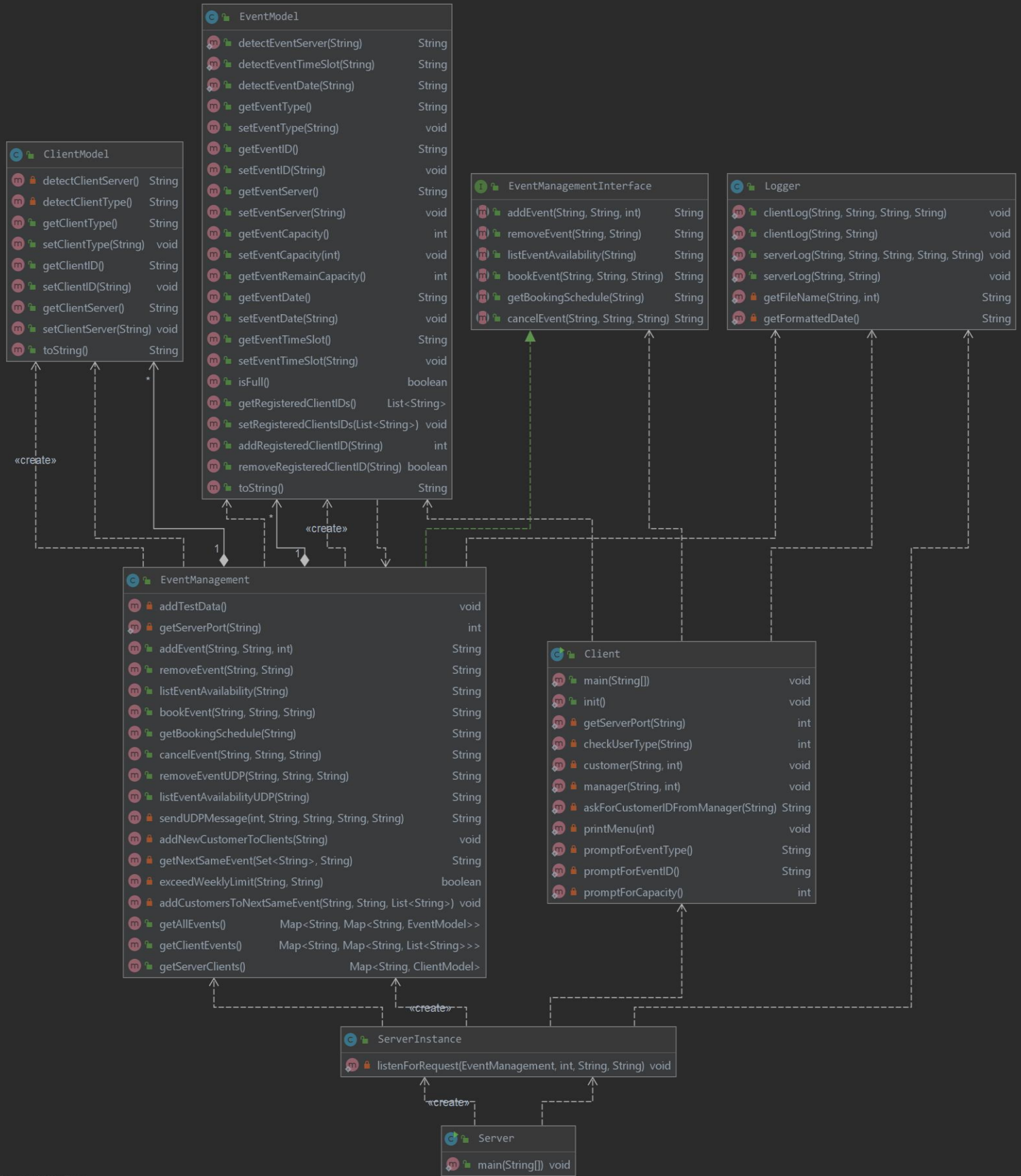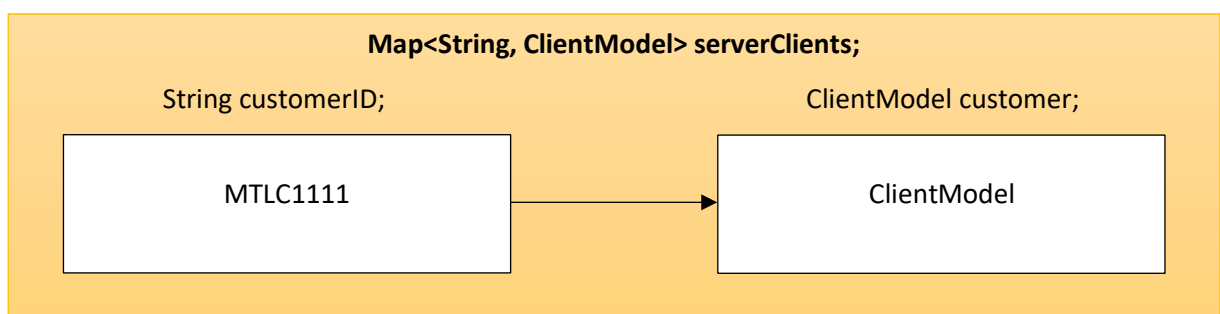**The Full dependencies and methods of each class is shown below.

EventModel
- detectEventServer(String) : String
- detectEventTimeSlot(String) : String
- detectEventDate(String) : String
- getEventType() : String
- setEventType(String) : void
- getEventID() : String
- setEventID(String) : void
- getEventServer() : String
- setEventServer(String) : void
- getEventCapacity() : int
- setEventCapacity(int) : void
- getEventRemainCapacity() : int
- getEventDate() : String
- setEventDate(String) : void
- getEventTimeSlot() : String
- setEventTimeSlot(String) : void
- isFull() : boolean
- getRegisteredClientIDs() : List<String>
- setRegisteredClientsIDs(List<String>) : void
- addRegisteredClientID(String) : int
- removeRegisteredClientID(String) : boolean
- toString() : String

ClientModel
- detectClientServer() : String
- detectClientType() : String
- getClientType() : String
- setClientType(String) : void
- getClientID() : String
- setClientID(String) : void
- getClientServer() : String
- setClientServer(String) : void
- toString() : String

EventManagementInterface
- 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

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

EventManagement
- addTestData() : void
- getServerPort(String) : int
- 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
- removeEventUDP(String, String, String) : String
- listEventAvailabilityUDP(String) : String
- sendUDPMessage(int, String, String, String, String) : String
- addNewCustomerToClients(String) : void
- getNextSameEvent(Set<String>, String) : String
- exceedWeeklyLimit(String, String) : boolean
- addCustomersToNextSameEvent(String, String, List<String>) : void
- getAllEvents() : Map<String, Map<String, EventModel>>
- getClientEvents() : Map<String, Map<String, List<String>>>
- getServerClients() : Map<String, ClientModel>

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

«create»

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

«create»

Server
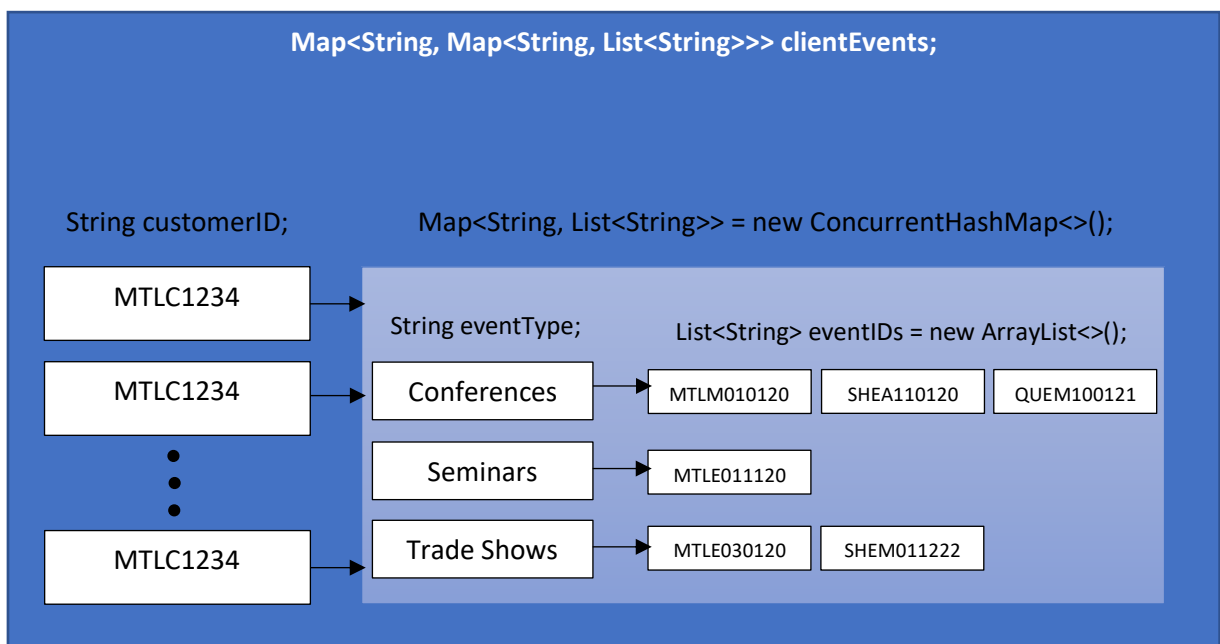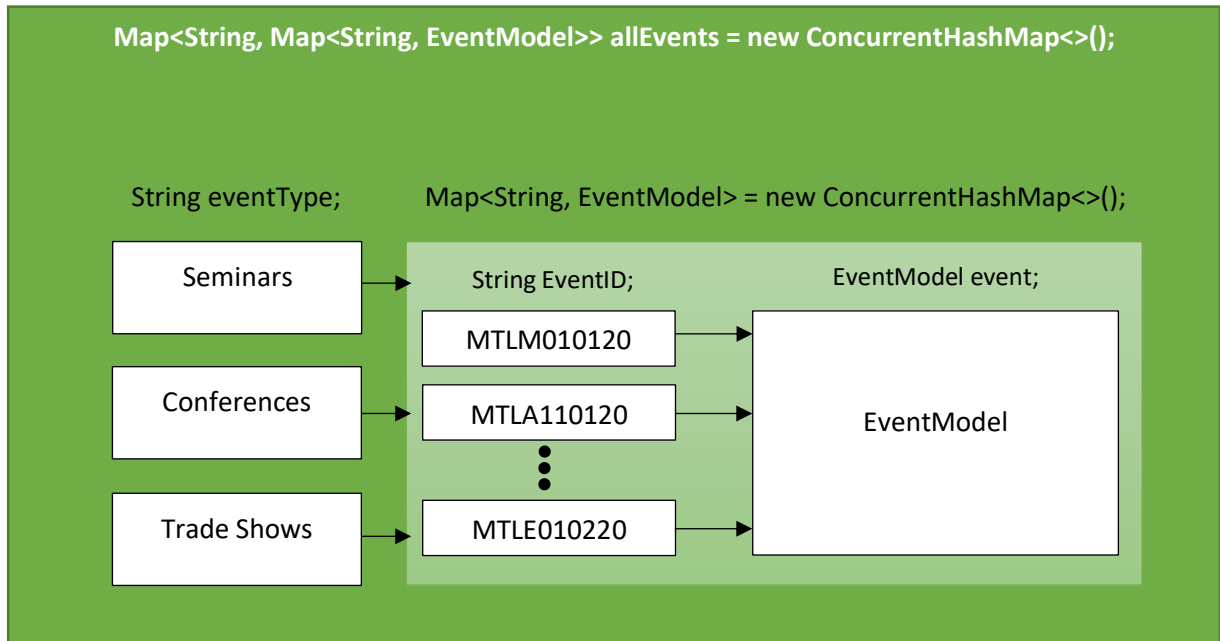- main(String[]) : void

«create»

Figure 2-Class Diagram with full dependencies

Powered by yFiles

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 |

String EventID;   EventModel event;

| MTLM010120 |

| Conferences |

| MTLA110120 |

EventModel

| Trade Shows |

| MTLE010220 |

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

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

| MTLC1234 |

| MTLC1234 |

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

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

| MTLC1234 |

| 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.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