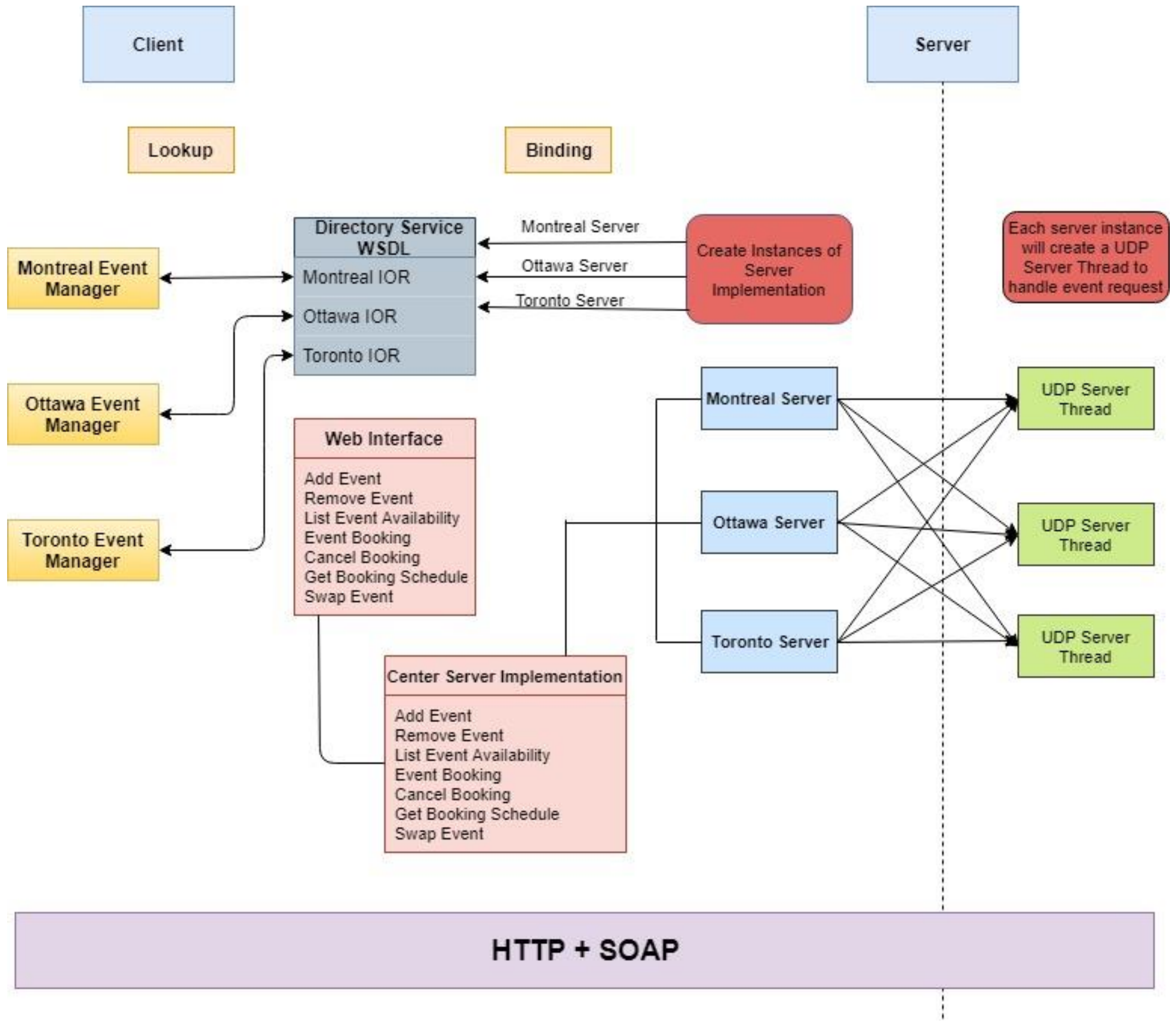
Distributed Event Management System

Prepared By: Himen Sidhpura (40091993) & Jenny Mistry (40092281)

Overall Description:

Event management is implemented as a distributed system to book and manage events across different branches of a corporate event management company. The system is published using Web Service and the users can see a single system handling user requests providing location and language transparency. It also manages simultaneous requests with adequate synchronization with the help of multithreading.

Design Architecture:



Web Service Interface:

- String addEvent(String managerId, String eventId, String eventType, String eventCapacity)
- String removeEvent(String managerId, String eventId, String eventType)
- String listEventAvailability(String managerId, String eventType)
- String eventBooking(String customerId, String eventId, String eventType)
- String cancelBooking(String customerId, String eventId, String eventType)
- String getBookingSchedule(String customerId)
- String swapEvent(String customerID, String newEventID, String newEventType, String oldEventID, String oldEventType)

Web Service Implementation (EventManagerClient):

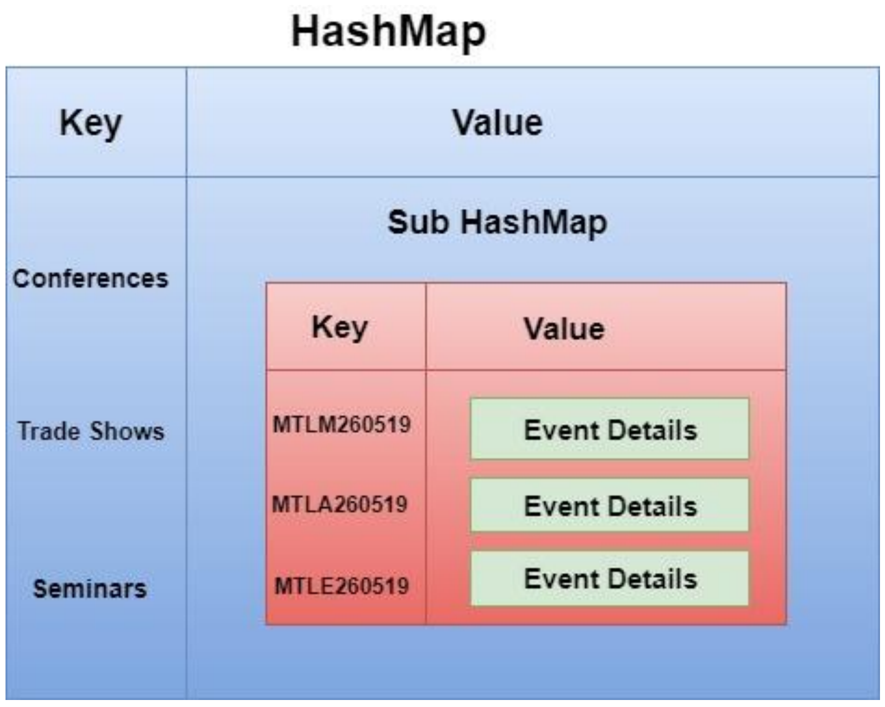
- This class implements the Web interface.
- Three instances of Web Service implementation are created. One each for branches: MTL, OTW, TOR.

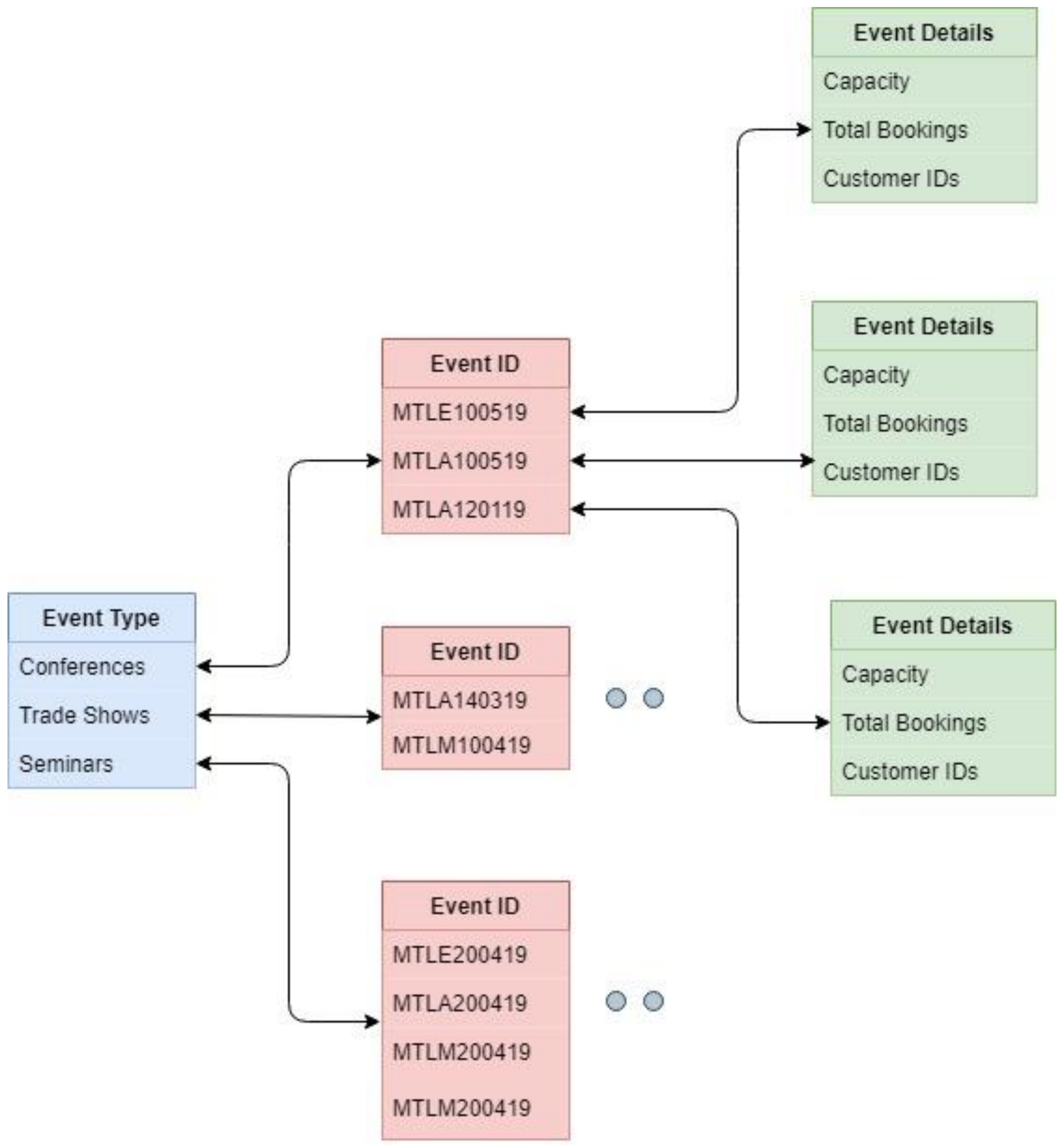
Web Service Directory Service:

Instances of EventManagerClient is published on localhost with three different strings to expose the WSDL objects to the client.

- Endpoint.publish("http://localhost:8080/EMS/TOR",toronto);
- Endpoint.publish("http://localhost:8080/EMS/MTL",montreal);
- Endpoint.publish("http://localhost:8080/EMS/OTW", ottawa);

Data Models:





Logs:

To perform logging for troubleshooting on both server and client end, we have utilized the logger functionality of Java (java.util.logging).

Log Format:

Each log data comprises of the below mentioned details:

- Date and time the request was sent.
- Request type (book an event, cancel an event, etc.).
- Request parameters (clientID, eventID, etc.).
- Request successfully completed/failed.
- Server response for the particular request.

Center Server:

Each server log (Montreal, Ottawa, Toronto) will be saved in their respective folder

- logs/MTL.txt
- logs/OTW.txt
- logs/TOR.txt

These logs include:

- Event added
- Event cancelled
- Availability of events
- Events swapped

Client:

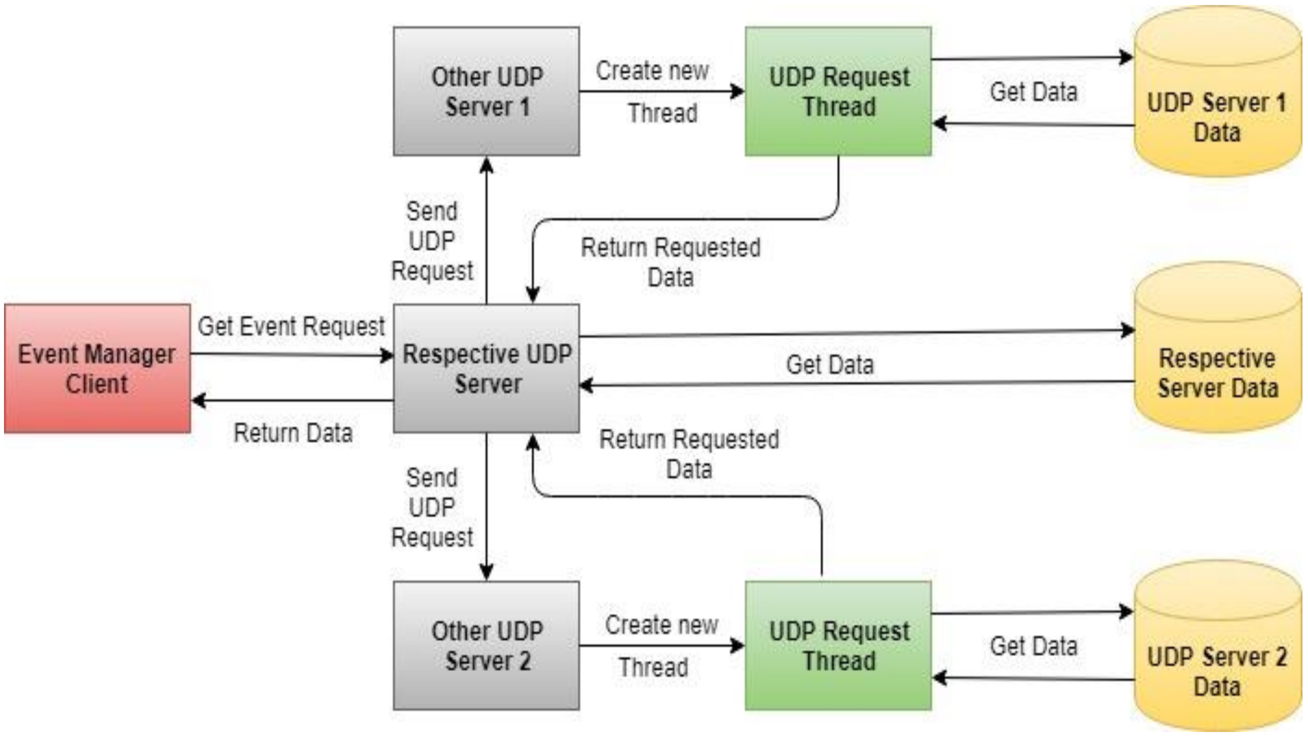
For every action performed by the client, a log file with clientID is created such as:

- Booking an event
- Canceling an event
- Retrieving booking schedule

Implementation:

- We have created a separate logger file for each of the three servers.
- To save contents of the corresponding log file, we have used a file handler.
- Various server responses are recorded using levels like WARNING, ERROR etc.

UDP Server Design:



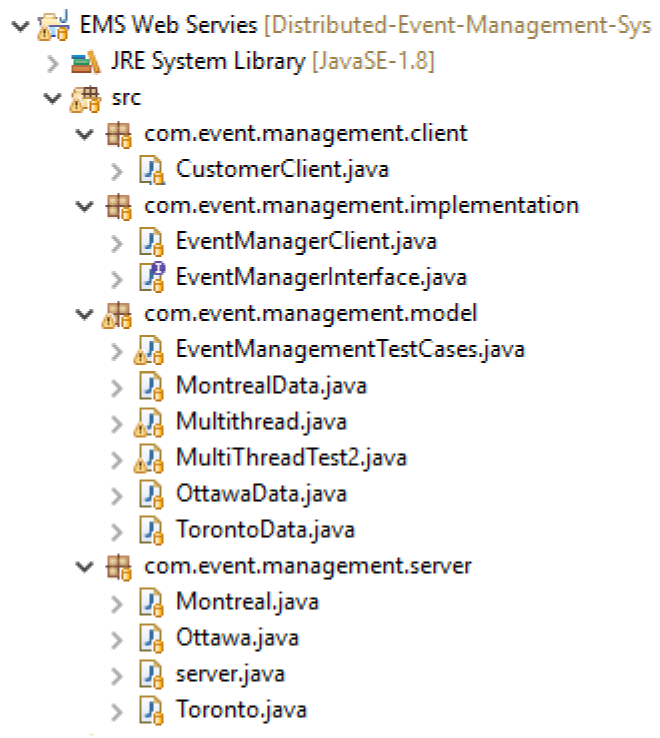
Flow:

- The event manager client sends event request to the respective server.
- The server fetches the requested data.
- It forks new requests to send the event request to the other servers located at various locations.
- The UDP servers at these locations receives the request and creates new threads to process the request.
- The newly created threads fetches the respective data and responds to the request.
- The server which received the request responds to the manager client with appropriate data.

Concurrency:

The manager client creates new thread to communicate to each of servers to handle requests for same or different events at the same time.

Code Structure:



```
EMS Web Servies [Distributed-Event-Management-Sys
  JRE System Library [JavaSE-1.8]
  src
    com.event.management.client
      CustomerClient.java
    com.event.management.implementation
      EventManagerClient.java
      EventManagerInterface.java
    com.event.management.model
      EventManagementTestCases.java
      MontrealData.java
      Multithread.java
      MultiThreadTest2.java
      OttawaData.java
      TorontoData.java
    com.event.management.server
      Montreal.java
      Ottawa.java
      server.java
      Toronto.java
```

Challenges:

Implementation of synchronization while managing multiple event requests at the same time has been challenging.

Test Scenarios:

- If the availability of an event is full, more customers cannot book the event.
- A customer can book as many events in his/her own city, but only at most 3 events from other cities overall in a month.
- A customer can perform only customer operation and cannot perform any event manager operation but an event manager can perform all operations for its own branches.
- If the user tries to add an event with an event id already added, then event details get updated.
- The user gets an error message "No events available", if he/she tries to add an event which is not created by manager.
- All the user and manager event requests have been synchronized to handle multiple concurrent event requests for the same/different branches.
- The swap event is successful only if old event remove and new event add operation are successful.
- If old/new event does not exist for swap event, an error message is shown.
- The swap event throws an error if new event add operation exceeds the month's max limit.

References:

- https://www.geeksforgeeks.org/multithreading-in-java/
- https://docs.oracle.com/javaee/5/tutorial/doc/bnayn.html
- https://docs.oracle.com/cd/E17802_01/webservices/webservices/reference/tutorials/wsit/doc/Examples_glassfish4.html
- https://www.journaldev.com/9191/java-web-services-tutorial
- https://www.theserverside.com/video/Step-by-step-SOAP-web-services-example-in-Java-using-Eclipse
- https://stackoverflow.com/questions/14451322/how-to-publish-wsdl-for-java/14451776#14451776