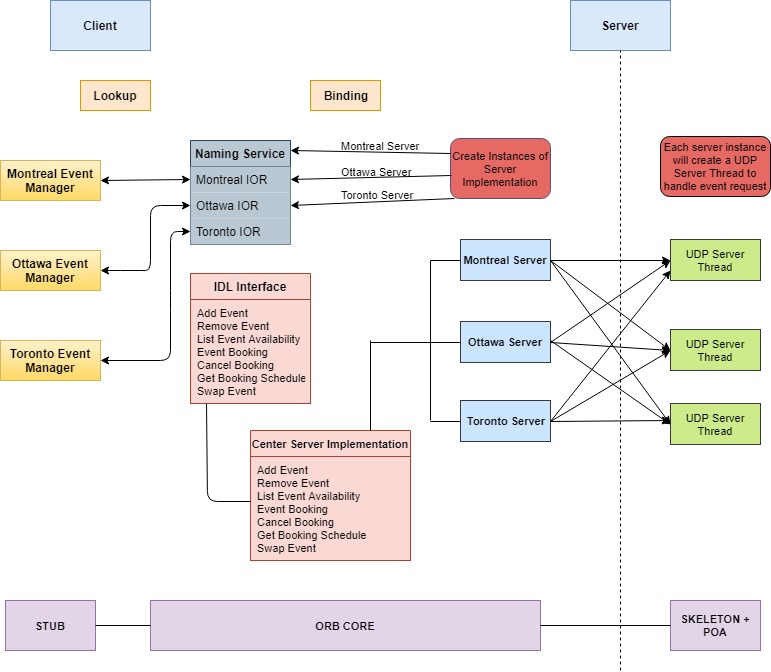
**Distributed Event Management System**

**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 built using CORBA architecture 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:**

****

**CORBA IDL Interface:**

* string addEvent(in string managerId, in string eventId, in string eventType,in string eventCapacity)
* string removeEvent(in string managerId, in string eventId, in string eventType)
* string listEventAvailability(in string managerId, in string eventType)
* string eventBooking(in string customerId, in string eventId, in string eventType)
* string cancelBooking(in string customerId, in string eventId, in string eventType)
* string getBookingSchedule(in string customerId)
* string swapEvent(in string customerId, in string newEventId, in string newEventType, in string oldEventId, in string oldEventType)

**CORBA Servant (EventManagerClient):**

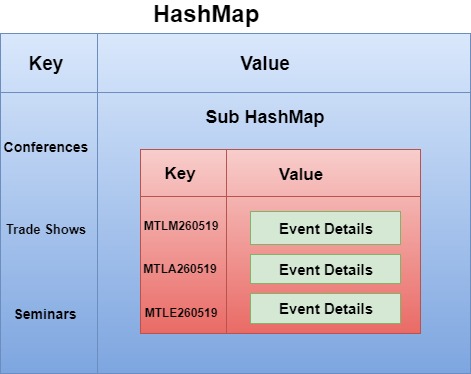
* This class implements the IDL interface.
* Three instances of CORBA Servant implementation are created. One each for branches: MTL, OTW, TOR.

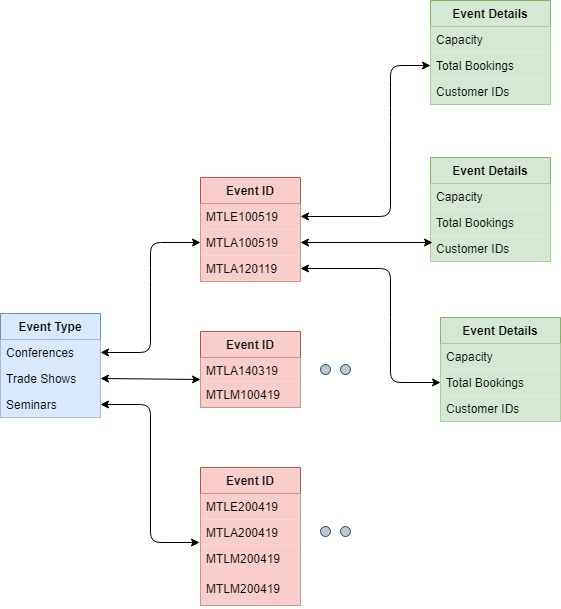
**CORBA Naming Service:**

Instances of EventManagerClient is bound to the CORBA Naming Service with three different strings to expose the objects to the client.

* ncRefTor.rebind(pathTor, hrefTor);
* ncRefMtl.rebind(pathMtl, hrefMtl);
* ncRefOtw.rebind(pathOtw, hrefOtw);

**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.log
* logs/otw.log
* logs/tor.log

These logs include:

* Event added
* Event cancelled
* Availability of events

**Client:**

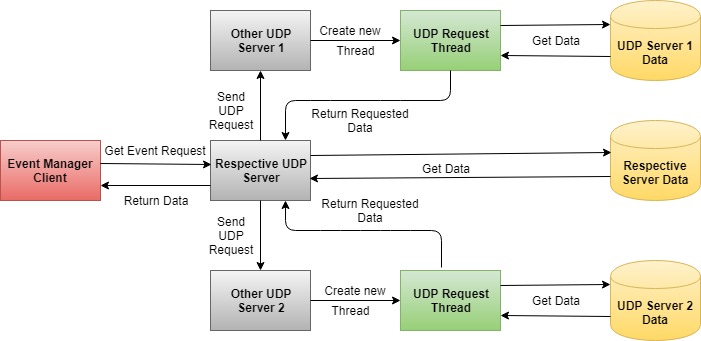
For every action performed by the client, a log file with clientID is created such as:

* 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:**

****

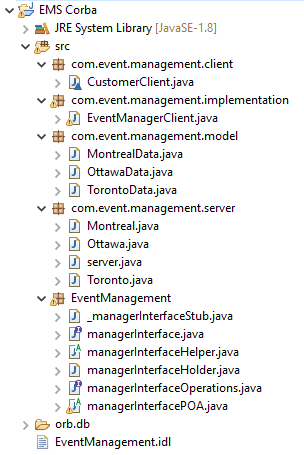
**Flow:**

* 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:**



**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/javase/7/docs/technotes/guides/idl/jidlExample.html>
* <https://www.geeksforgeeks.org/client-server-software-development-introduction-to-common-object-request-broker-architecture-corba/>
* <https://www.math.uni-hamburg.de/doc/java/tutorial/idl/hello/idltojava.html>
* <https://xennis.org/wiki/CORBA_-_Advanced_example_with_server-client_in_Java_and_C%2B%2B>
* <https://docs.oracle.com/javase/7/docs/technotes/guides/idl/POA.html>
* <https://www.geeksforgeeks.org/synchronized-in-java/>