Designkonzept

# AnalyticsServer

Der AnalyticsServer bekommt Events vom Auction-Server und verarbeitet diese, wenn notwendig, zu weiteren Events. ManagementClients können sich mittels einer „Subscribe“-Methode beim Analyse-Server für Benachrichtigungen für bestimmte Events anmelden. Dies geschieht mit sogenanntent Regex, diese Überprüfen, ob sie mit einem Event-Typen übereinstimmen, wenn ja, bekommt der Client Benachrichtigungen darüber. Diese Benachrichtigungen werden mittels RMI Call-Back gelöst, das heißt, dass der Client beim Anmelden beim Analyse-Server sein eigenes Remote-Objekt mit der Methode „notify“ übergibt. Wird ein entsprechendes Event getriggert, so werden alle Clients für dieses Event benachrichtigt.

Intern werden die Clients in einer Map gespeichert. Diese speichert für jedes Event eine weitere Map, welche jeweils Subscription-ID und Clientinterface speichert.

Alle Events die vom Auktions-Server an den Analyse-Server geschickt werden, werden in einer ConcurrentQueue gespeichert. Die Events werden anschließend von einem Thread „EventHandler“ bearbeitet. Dieser wartet mit der Methode „queue.take()“ solange, bis in der Liste ein Event steht. Kommt ein Event rein, so wird überprüft ob es zu einem der 3 Typen gehört:

* AuctionEvent
* UserEvent
* BidEvent

Für jedes dieser Event-Arten gibt es unterschiedliche Behandlungsmöglichkeiten:

**AuctionEvent:**

Wenn es vom Typ Auction\_Started ist, so wird das Event in einer Liste gespeichert, damit bei einem eintreffenden Event (Auction\_Ended) die Dauer berechnet werden kann.

Ist es vom Typ Auction\_Ended, so wird aus der Liste das Start-Event gesucht und die Dauer berechnet. Des Weiteren wird aus einer Liste mit gebotenen Auktionen gesucht, ob auf diese Auktion geboten wurde, wenn ja, dann wird ein erfolgreiches Event hinzugefügt. Danach werden folgende, weitere Events getriggert:

* AuctionSuccessRatio
* AuctionTimeAVG

**UserEvent:**

Ist es vom Typ User\_LOGIN, dann wird das Event in einer Liste gespeichert.

Kommt ein Event USER\_LOGOUT oder USER\_DISCONNECTED, so wird das entsprechende USER\_LOGIN Event gesucht, und folgende Events berechnet:

* USER\_TIME\_MIN
* USER\_TIME\_MAX
* USER\_TIME\_AVG

**BidEvent:**

Hier werden nur die Bid\_Placed Events behandelt. Wenn ein Bid platziert wurde, so wird die Auktion ID der Liste der gebotenen Auktionen hinzugefügt, um später die erfolgreichen Auktionen zu berechnen.

Das Ursprungsevent und die berechneten Events werden in einer Queue gespeichert, die von der notify-Methode abgearbeitet wird. Diese Methode prüft auch, dass ein User nur ein Event einmal bekommt. Dies geschieht, indem man alle User die das Event schon haben in ein Set speichert. Ist der User bereits vorhanden, so bekommt er das Event nicht nochmal.

**User nicht mehr vorhanden:**

Wenn ein User beim notify nichtmehr vorhanden ist, so wirft die Methode eine RemoteException. Diese wird gefangen. Wenn diese Exception gefangen wird, dann wird der User aus allen Subscriptions entfernt.

**Subscription-ID:**

Eindeutige ID, die einer Anmeldung eines Client zugehörig ist. Mit dieser ID kann sich der Client wieder von Benachrichtigungen anmelden. Diese ist ein Integer der für jeden User hochgezählt wird.

**BidCountPerMinute**

Um die Bids pro Minute zu berehnen wird ein extra Timer implementiert. Dieser greift auf die Variable „bidcount“ vom EventHandler zu und berechnet durch die bereits abgelaufene Zeit (die Anzahl der Durchläufe wird als Integer gespeichert) und erstellt das entsprechende BidCountPerMinute Event.

Testbeschreibung

# EventHandlerTest

In dieser Testklasse wird die Funktionsweise des EventHandler getestet. Das bedeutet, ob alle Eingangs Events die entsprechenden Ausgangsevents liefern.

## Setup

Am Anfang wird ein neuer AnalyticsServer gestartet. Danach wird ein Mock-Client gestartet, welcher alle Events über die er benachrichtigt wird in einer Liste speichert. Diese können sich die Testfälle dann mittels getter + iterator holen.

## TestUserLoginEvent

Ein USER\_LOGIN Event wird an den EventHandler geschickt. Das erwartete Ergebnis ist ein Event USER\_LOGIN, welches sich in der Queue des Clients befindet. Dieser Test ist sehr trivial und dient nur zum initialen Testen.

## testUserLoginLogoutSessionEvent

In diesem Test wird ein User eingeloggt und ein User ausgeloggt. Das passiert durch das Senden eines USER\_LOGIN und eines USER\_LOGOUT Event. Der Timestamp wird so gesetzt, dass die SessionTime 50 ist. Nach der Verarbeitung werden folgende Events erwartet: USER\_LOGIN, USER\_LOGOUT, USER\_SESSIONTIME\_MIN (mit dem Wert 50), USER\_SESSIONTIME\_MAX (mit dem Wert 50) und USER\_SESSIONTIME\_AVG (mit dem Wert 50).

## testAuctionStartedEndedNoBid

In diesem Test wird eine Auction gestartet, beendet und KEIN Bid darauf platziert. Erwartet wird eine Success\_Ratio von 0.

Input: AUCTION\_STARTED, AUCTION\_ENDED

Output: AUCTION\_STARTED, AUCTION\_ENDED, AUCTION\_TIME\_AVG (Wert 50, da der Timestamp so gesetzt wird, AUCTION\_SUCCESS\_RATIO( Wert 0))

### testBidOnAuctionChangesRatio

In diesem Fall wird eine Auktion gestarted, ein Bid platziert und die Auktion beendet. Erwartet wird eine SuccessRatio von 1.

Input: AUCTION\_STARTED, AUCTION\_ENDED

Output: AUCTION\_STARTED, AUCTION\_ENDED, AUCTION\_TIME\_AVG (Wert 50, da der Timestamp so gesetzt wird, AUCTION\_SUCCESS\_RATIO( Wert 1))

## testBidCountPerMinute

Es wird eine Auktion gestartet, ein Bid platziert und 1 Minute gewartet. Danach wird der BidCountPerMinuteWatcher ein Event senden. Erwartet wird ein Wert 1.0