

Task08 - RMI Auction 5AHITT

Schuschnig Tobias, Lins Tobias, Auradnik Alexander, Klune Alexander,
Pölzbauer Patrick, Alexander Rieppel

Inhaltsverzeichnis

Task08 - RMI Auction	5
Aufgabenstellung	5
Aufgabenteilung (Vereinfacht)	17
Funktionsliste	18
Designüberlegung Allgemein (Probleme / Überlegungen)	19
ConcurrentHashMap	19
Exceptions	19
Offene Fragen Management Client	19
Designüberlegungen	20
UML Klassendiagramme	20
UML Activity Diagramm	24
UML Use Case Diagramm	26
Management Client	28
<i>Implementierung</i>	28
<i>Befehle um mit den Billing Server zu aggieren:</i>	28
<i>Befehle um mit den Analytics Server zu aggieren:</i>	29
<i>Remote Message Management</i>	29
<i>Dadurch ergeben sich folgende Befehle, um die Moduse zu wählen bzw. zu aggiren:</i>	29
Offene Fragen	30
Ergänzung 10.02.2014	30

Billing Server	31
<i>Exceptions</i>	32
<i>Ausgabe</i>	32
<i>login()</i>	32
Analytics Server	34
<i>Benachrichtigungen:</i>	35
Testing Component	37
Connections	43
Exceptions	45
UserInputException:	45
InvalidInputException:	45
Ebenfalls mögliche Exceptions:	46
Zeitschätzung und Tatsächliche Zeit	47
Arbeitsaufteilung und Termine	50
Abnahme der Arbeitspakete:	52
UML Klassendiagramm	52
UML Aktivitätsdiagramm	52
UML Use Case Diagramm	52
TDD (ersten Testfälle)	52
Recherche Testfälle Designüberlegung	53
Management Client Designüberlegung	53
AnalyticsServer Desingüberlegung	53
BillingServer Desingüberlegung	53
Testing Component Designüberlegung	53
Connections Designüberlegung	54

Exceptions Designüberlegung	54
Model Designüberlegung	54
Zeitaufzeichnung und Zeitschätzung	54
Review 1	55
Management Client	55
TDD fertig	55
Model	56
BillingServer	56
<i>BillingServerSecure</i>	56
Testing Component	56
Exceptions	57
Testing Component	57
Analytic Server Implementierung	57
Review 3	58
Management Client Fertig	58
Testing Restabnahmen	58
Testing Component	59
Exceptions nachgereicht	59
Connections	59
BillingServer	59
Evaluation Testing Component	60
Programmbeschreibung	61
Reviews	70
Review 29.1.14	70
<i>Negativ aufgefallen:</i>	70

<i>Positiv aufgefallen:</i>	70
Review 12.2	71
<i>Positiv Aufgefallen:</i>	71
<i>Neutral Aufgefallen:</i>	71
<i>Negativ Aufgefallen:</i>	71
Review 19.2	72
<i>Positiv Aufgefallen:</i>	72
<i>Neutral Aufgefallen:</i>	72
<i>Negativ Aufgefallen:</i>	72
Review 26.2	73
<i>Positiv Aufgefallen:</i>	73
<i>Neutral Aufgefallen:</i>	73
<i>Negativ Aufgefallen:</i>	73
Zeitaufzeichnung	74

Task08 - RMI Auction

Aufgabenstellung

General Remarks

- We suggest reading the following tutorials before you start implementing:
 - [Java RMI Tutorial](#): A short introduction into RMI.
 - [Distributed Computing for Java](#): RMI Whitepaper.
- Be sure to check the Hints & Tricky Parts section for questions!

Overview

The goal of this assignment is to extend the auction system with four additional components:

- An **analytics server** receives events from the system and computes simple statistics/analytics.
- A **billing server** which manages the bills charged by the auction provider.
- A **management client** which interacts with the analytics server and the billing server.
- A **testing component** to generate client requests for automated load testing of the system.

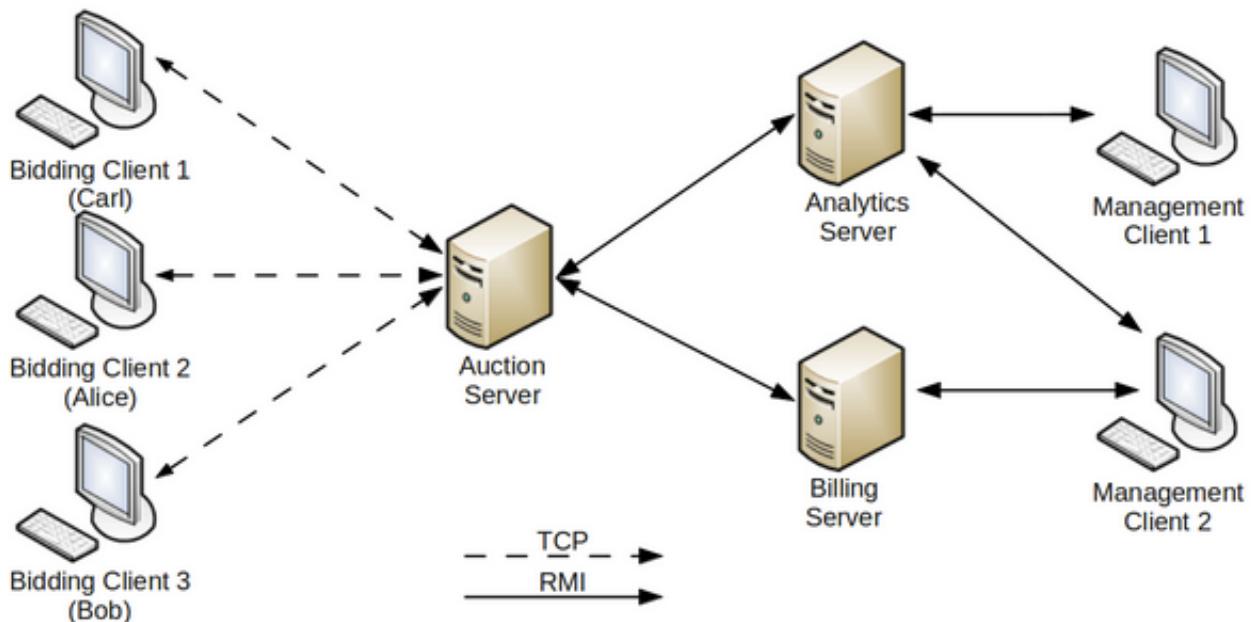


Figure 1: System Components and Interactions.

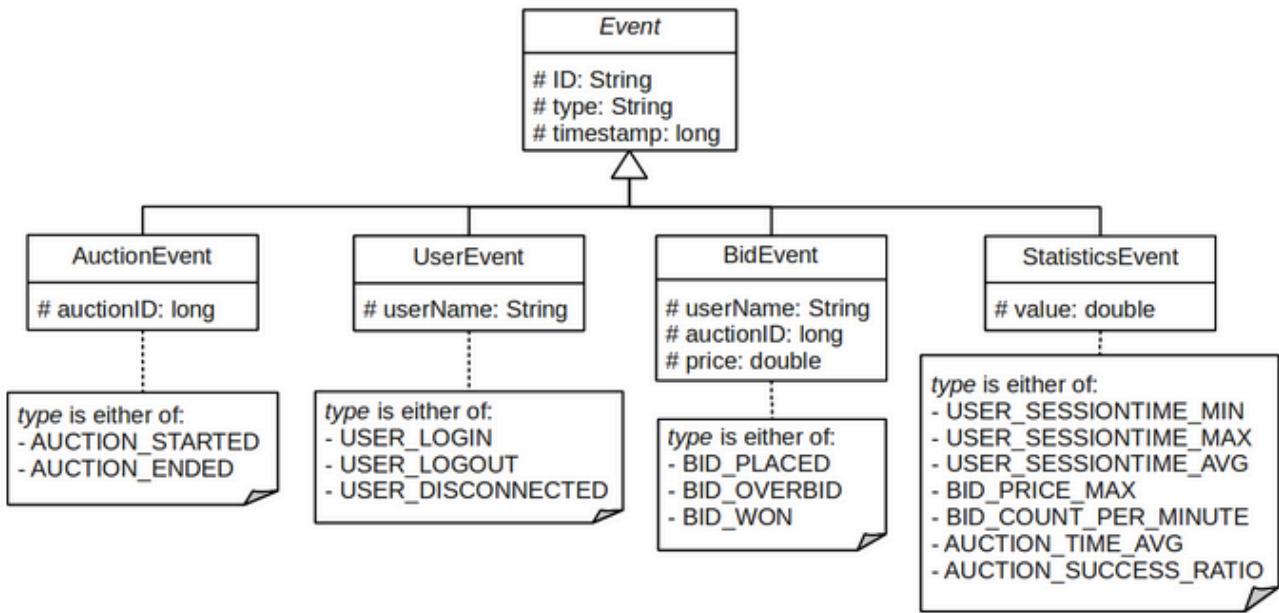
Analytics Server

The purpose of the analytics server is to monitor the execution of the auction platform and to provide simple statistics about its usage. The analytics server should receive event updates from the auction server that you have implemented in assignment 1. Moreover, the analytics server should allow one or multiple management clients to subscribe for event notifications. To that end, the analytics server provides three RMI methods:

- A **subscribe** method is invoked by the management client(s) to register for notifications. The method must allow to specify a filter (specified as a regular expression string) that determines which types of events the client is interested in (see details further below). Moreover, the subscribe method receives a callback object reference which is used to send notifications to the clients. Study the RMI callback mechanism to solve this. The method returns a unique subscription identifier string.
- A **processEvent** method is invoked by the bidding server each time a new event happens (e.g., user logged in, auction started, ...). The analytics server forwards these events to subscribed clients, and possibly generates new events (see details further below) which are also forwarded to clients with a matching subscriptions. If the server finds out that a subscription cannot be processed because a client is unavailable (e.g., connection exception), then the subscriptions is automatically removed from the analytics server.
- An **unsubscribe** method is invoked by the management client(s) to terminate an existing event subscription. The method receives the subscription identifier which has been previously received from the **subscribe** method.

Event Type Hierarchy

The processEvent method of the analytics server should receive an object of type **Event**. To transport the necessary event payload (user name, auction ID, bid price, ...), implement a simple event hierarchy which is illustrated in the figure below. The sub-types inherit from the abstract class **Event**, and the business logic inside the processEvent method should determine the concrete runtime type of the incoming events, in order to take appropriate action. Each event contains a unique identifier (ID), a type string, and a timestamp, and specialized event types contain additional data. For instance, the *AuctionEvent* defines a member variable *auctionID* that carries the auction identifier which this event applies to.

**Figure 2:** Event Hierarchy and Basic Event Properties.

Statistics Events

The analytics server is responsible for processing the raw events and creating the following aggregated statistics event types:

- `USER_SESSIONTIME_MIN` / `USER_SESSIONTIME_MAX` / `USER_SESSIONTIME_AVG`: minimum/maximum/average duration over all user sessions. A user session starts with a login and ends if the user logs out (or gets disconnected unexpectedly).
- `BID_PRICE_MAX`: maximum over all bid prices.
- `BID_COUNT_PER_MINUTE`: number of bids per minute, aggregated over the whole execution time of the system.
- `AUCTION_TIME_AVG`: average auction time, aggregated overall historical auctions logged during the execution of the system.
- `AUCTION_SUCCESS_RATIO`: ratio of successful auctions to total number of finished auctions. An auction is successful if at least one bid has been placed.

Note that an incoming event (e.g., type `AUCTION_ENDED`) may trigger the generation of multiple new events (e.g., types `AUCTION_TIME_AVG`, `AUCTION_SUCCESS_RATIO`).

Event Subscription Filter

The `subscribe` method of the statistics server should allow to set a simple subscription filter. The filter is a Java regular expression which is matched against the `type` variable of the `Event` class. For instance, to subscribe for all user-related and bid-related events, the filter "`(USER_.* | BID_.*)`" is used.

Billing Server

The billing server provides RMI methods to manage the bills of the auction system. To secure the access to this administrative service, the server is split up into a BillingServer RMI object (which basically just provides login capability) and a BillingServerSecure which provides the actual functionality.

Pricing Curve

Administrators can use the billing server to set the pricing curve in terms of fixed price and variable price steps. In a typical pricing curve, the auction fee percentage decreases with increasing auction price (price of the winning bid). For example, the price steps could look something like this:

Auction Price	Auction Fee (Fixed Part)	Auction Fee (Variable Part)
0	1.0	0.0 %
(0-100]	3.0	7.0 %
(100-200]	5.0	6.5 %
(200-500]	7.0	6.0 %
(500-1000]	10.0	5.5 %
> 1000	15.0	5.0 %

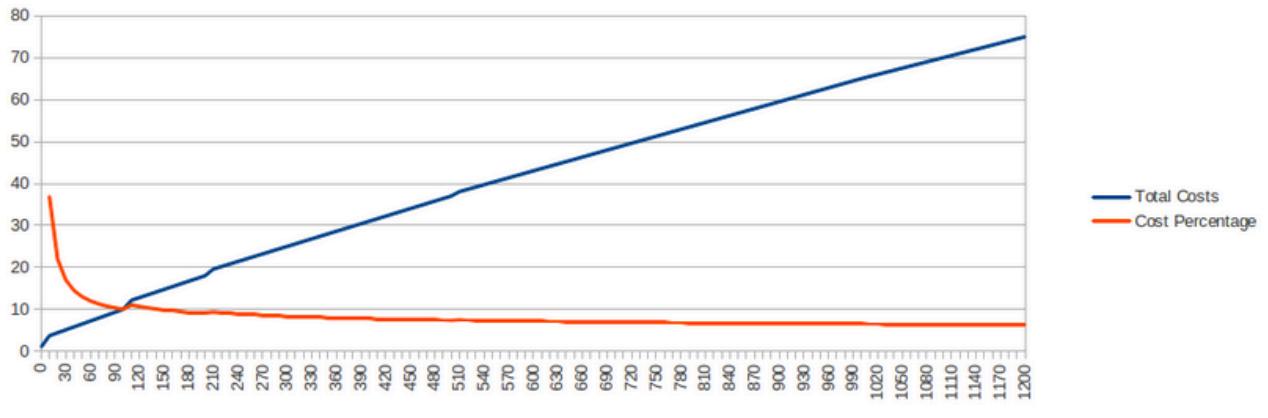


Figure 3: Pricing Curve

Billing Server RMI Methods

The *BillingServer* class provides the following RMI methods (exception declarations not included):

- `BillingServerSecure login(String username, String password):`

The access to the billing server is secured by user authentication. To keep things simple, the username/password combinations can be configured statically in a config file

`user.properties`. Each line in this file contains an entry "<username> = <password>", e.g., "john = f23c5f9779a3804d586f4e73178e4ef0". Do not put plain-text passwords into the config file, but store the MD5 hash (not very safe either, but sufficient for this assignment) of the passwords. Use the `java.security.MessageDigest` class to obtain the MD5hash of a given password. If and only if the login information is correct, the management client obtains a reference to a *SecureBillingServer* remote object, which performs the actual tasks.

The *BillingServerSecure* provides the following RMI methods (exception declarations not included):

- `PriceSteps getPriceSteps():` This method returns the current configuration of price steps. Think of a suitable way to represent the list of price step configurations inside your `PriceSteps` class.
- `void createPriceStep(double startPrice, double endPrice, double fixedPrice, double variablePricePercent):` This method allows to create a price step for a given price interval. Throw a `RemoteException` (or a subclass thereof) if any of the specified values is negative. Also throw a `RemoteException` (or a subclass thereof) if the provided price interval collides (overlaps) with an existing price step (in this case the user would have to delete the other price step first). To represent an infinite value for the `endPrice` parameter (e.g., in the example price step "> 1000") you can use the value 0.
- `void deletePriceStep(double startPrice, double endPrice):` This method allows to delete a price step for the pricing curve. Throw a `RemoteException` (or a subclass thereof) if the specified interval does not match an existing price step interval.
- `void billAuction(String user, long auctionID, double price):` This method is called by the auction server as soon as an auction has ended. The billing server stores the auction result (data do not have to be persisted, storing in memory is sufficient) and later uses this information to calculate the bill for a user. **Note:** The auction server should use the same login mechanism as the management clients to talk to the billing server. You can add pre-defined user credentials (for instance "auctionClientUser = f23c5f9779a3804d586f4e73178e4ef0") to your properties file.
- `Bill getBill(String user):` This method calculates and returns the bill for a given user, based on the price steps stored within the billing server. Implement a class `Bill` which encapsulates all billing lines of a given user (also see sample output of management client further below). You need not implement any payment mechanism - the bill should simply show the total history of all auctions created by the user.

Management Client

The management client communicates with the analytics server and the billing server. The client should provide all necessary commands to interact with the analytics server and the billing server. The following commands are used to interact with the billing server (please follow the output format illustrated in the examples):

- **!login <username> <password>**: Login at the billing server.

```
> !login bob BobPWD
bob successfully logged in
```

- **!steps**: List all existing price steps

```
bob> !steps
```

Min_Price	Max_Price	Fee_Fixed	Fee_Variable
0	0	1.0	0.0 %
0	100	3.0	7.0%
100	200	5.0	6.5%
200	500	7.0	6.0%
500	1000	10.0	5.5%

Note: the table columns in the output do not have to be perfectly aligned.

- **!addStep <startPrice> <endPrice> <fixedPrice> <variablePrice-Percent>**: Add a new price step.

```
bob> !addStep 1000 0 15 5
Step [1000 INFINITY] successfully added
bob> !steps
```

Min_Price	Max_Price	Fee_Fixed	Fee_Variable
0	0	1.0	0.0 %
0	100	3.0	7.0%
100	200	5.0	6.5%
200	500	7.0	6.0%
500	1000	10.0	5.5%
1000	INFINITY	15.0	5.0%

- **!removeStep <startPrice> <endPrice>**: Remove an existing price step.

```
bob> !removeStep 0 100
Price step [0 100] successfully removed
bob> !removeStep 111 222
ERROR: Price step [111 222] does not exist
```

- **!bill <userName>**: This command shows the bill for a certain user name. Print the list of finished auctions (plus auction fees) that have been created by the specified user. To de-

determine the fees of the bill, apply the current price steps configuration to the prices of each of the user's auctions.

```
bob> !bill bob
```

auction_ID	strike_price	fee_fixed	fee_variable	fee_total
1	0	1	0	1
17	700	10	38.5	48.5
19	120	5	7.8	12.8

- **!logout:** Set the client into "logged out" state and discard the local copy of the *BillingServiceSecure* remote object. After this command, users have to use the "**!login**" command again in order to interact with the billing server.

```
bob> !logout
bob successfully logged out
> !bill bob
ERROR: You are currently not logged in.
```

Provide reasonable output with all relevant information for each of the commands (as illustrated in the examples above). Also print an informative error message if an exception is received from the server (Error log messages should start with the string "ERROR:").

The following user commands are provided by the management client to communicate with the analytics server:

- **!subscribe <filterRegex>:** subscribe for events with a specified subscription filter (regular expression). A user can add multiple subscriptions.

```
bob> !subscribe '(USER_.* )|(BID_.* )'
Created subscription with ID 17 for events using filter
'(USER_.* )|(BID_.* )'
```

- **!unsubscribe <subscriptionID>:** terminate an existing subscription with a specific identifier (**subscriptionID**).

```
bob> !unsubscribe 17
subscription 17 terminated
```

Similar to the analytics server, the management client (i.e., the RMI callback object) should implement a **processEvent** RMI method. This method is invoked by the analytics server each time an event is generated that matches a subscription by this client. To display incoming events to the user, simply print the event time, event type and all additional event properties to the command line. You should support two modes of printing, namely *automatic* and *on-demand*:

- **!auto:** This command enables the automatic printing of events. Whenever an event is received by the clients, its details are immediately printed to the command line.
- **!hide:** This command disables the automatic printing of events. Incoming events are temporarily buffered and can later be printed using the "**!print**" command. This should be the default mode when the client is started.
- **!print:** Print all events that are currently in the buffer and have not been printed before (an excerpt of a possible example trace is printed below).

```

bob> !print
USER_LOGIN: 31.10.2012 20:00:01 CET - user alice logged in
USER_LOGIN: 31.10.2012 20:01:03 CET - user john logged in
BID_PLACED: 31.10.2012 20:01:50 CET - user alice placed bid
3100.0 on auction 32
BID_PRICE_MAX: 31.10.2012 20:01:50 CET - maximum bid price
seen so far is 3100.0
BID_COUNT_PER_MINUTE: 31.10.2012 20:01:50 CET - current bids
per minute is 0.563
USER_LOGOUT: 31.10.2012 20:03:11 CET - user john logged out
USER_SESSION_TIME_MIN: 31.10.2012 20:03:11 CET - minimum ses-
sion time is 62 seconds
USER_SESSION_TIME_MAX: 31.10.2012 20:03:11 CET - average ses-
sion time is 324 seconds
USER_SESSION_TIME_AVG: 31.10.2012 20:03:11 CET - maximum ses-
sion time is 778 seconds

```

Note: The value of BID_COUNT_PER_MINUTE depends on the previous events which are not visible in this trace. For the example, an arbitrary value of 0.563 has been chosen for illustration. (Arbitrary values have also been chosen for the aggregated session durations.)

In both variants, it must be ensured that each distinct event is presented to the user only *once*, even if it matches multiple subscriptions. It is up to you whether you solve this requirement server-side (never send out duplicate events to any client) or client-side (receive duplicate events, but print each event only once on the command line). If you need to temporarily store a list of already published or already printed events, make sure that the events are eventually freed and that your program does not run out of memory over time.

Load Testing Component

In practice, online auction systems are highly concurrent and should provide robustness and scalability, even for a large number of clients. Concurrency issues are hard to detect under normal operation, but generating artificial load on the system may help to detect problems and inconsistencies. Hence, you should create a load testing component to test the system's scalability and reliability. The following test parameters should be configurable in the properties file `loadtest.properties` (see template):

- *clients*: Number of concurrent bidding clients
- *auctionsPerMin*: Number of started auctions per client per minute
- *auctionDuration*: Duration of the auctions in seconds
- *updateIntervalSec*: Number of seconds that have to pass before the clients repeatedly update the current list of active auctions
- *bidsPerMin*: Number of bids placed on (random) auctions per client per minute

To place a bid, the test client selects a random auction from the list of active auctions (if any). When your test clients place bids, you need to ensure that each bid is higher than the previous bids. To achieve this, simply set the bid price that corresponds to the number of seconds that have passed since the auction was started, with decimal values (e.g., 10,342 for ten seconds and 342 milliseconds).

Moreover, the test environment should instantiate a management client with an event subscription on any event type (filter ".*"). During the tests, the management client should be in "auto" mode, i.e., print all the incoming events automatically to the command line.

Scaling up the number of clients would block too many port numbers for the UDP notifications. Hence, for assignment 2 you should disable the UDP notifications (part of assignment 1), i.e., do not open a UDP socket on the bidding clients, and do not send UDP notifications from the auction server to the clients.

Play around with different test settings and try to roughly determine the limits of your machine. By limits we mean the configuration values for which the system is still stable, but becomes unstable (e.g., runs out of memory after some time) if any of these values are further increased (or decreased). Monitor the memory usage with tools like "top" (Unix) or the process manager under Windows, and let the test program execute for a sufficiently long time (around 5-10 minutes). Obviously, the load test can also help you identify issues in your implementation (e.g., deadlocks, memory leaks, ...).

In your submission, include a file **evaluation.txt** in which you provide your machine characteristics (operating system version, number and clock frequency of CPUs, RAM capacity) and the key findings of your evaluation (at least the limit values for all configuration parameters). **Note: Make sure that your tests terminate after a short time** (say, 8 minutes)!

Implementation Details and Hints

Implementation Details

RMI uses the `java.rmi.registry.Registry` service to enable application to retrieve remote objects. This service can be used to reduce coupling between clients (looking up) and servers (binding): the real location of the server object becomes transparent. In our case, the servers (analytics server and the billing server) will use the registry for binding and the client will look up the remote objects.

One of the first things the servers need to do is to connect to the `Registry`, therefore the RMI registry needs to be set up before its first usage. This can be achieved by calling the

`LocateRegistry.createRegistry(int port)` method which creates and exports a `Registry` instance on localhost. A properties file (named `registry.properties`) should be read from the classpath (see the hint section for details) to get the port the `Registry` should accept requests on. The properties file is provided in the template. It also contains the host the `Registry` is bound to. This information is vital to the client application that needs to connect to the `Registry` using the `LocateRegistry.getRegistry(String host, int port)` method. Note that a client, in contrast to the management component, need not bind any objects to the `Registry`.

After obtaining a reference to the `Registry`, this service can be used to bind an RMI remote interface (using the `Registry.bind(String name, Remote obj)` method). Remote Interfaces are common Java Interfaces extending the `java.rmi.Remote` interface. Methods defined in such an interface may be invoked from different processes or hosts. In our case, we need to bind two objects to the registry: an instance of `BillingServer` and an instance of `AnalyticsServer`. If you prefer to bind only a single object to the registry, you may choose to implement an additional "facade" object which returns instances of the two server classes (however, this is not required).

To make your object remotely available you have to export it. This can either be accomplished by extending `java.rmi.server.UnicastRemoteObject` or by directly exporting it using the static method `UnicastRemoteObject.exportObject(Remote obj, int port)`. In the latter case, use 0 as port: This way, an available port will be selected automatically.

For managing the configurable data (e.g., user credentials) you will have to read a `.properties` file. The `user.properties` file must be located in the server's classpath. A sample user properties file is provided in the template.

Bootstrapping and Terminating the System

To allow for efficient (automated) testing of your solution, make sure that your system components can be started using both of the following command sequences:

- `ant run-billing-server`
- `ant run-analytics-server`
- `ant run-server`

or

- `ant run-analytics-server`
- `ant run-billing-server`
- `ant run-server`

That is, the billing server and the analytics server are started first (in any order), then the auction server is started. Upon instantiation of the billing server and the analytics server you should check whether the RMI registry is already available, and create a new registry instance if it does not yet exist. You can assume that there is a short delay (3 seconds) between the start of each component. To terminate any of the three components, the command `!exit` is used in the terminal in which the component is running. Terminating any of the three servers should be handled gracefully in the other components, i.e., if the auction server is unable to communicate with either the analytics server or the billing server, a warning message should be displayed, but the auction server should not terminate or raise/print an exception. Any requests targeted at an unavailable server can simply be dropped. This means that you do *not* need to store/buffer any requests or events in the auction server until the other servers become available. Simply drop requests or events (with a warning message) if they cannot be forwarded to the target server immediately.

The management clients should also be terminated using the `!exit` command. Make sure to properly clean up all resources before the client terminates.

Important Points to Consider

Make sure to synchronize the data structures you use to manage price steps, events, etc. Even though RMI hides many concurrency issues from the developer, it cannot help you at this point. You may consult the [Java Concurrency Tutorial](#) to solve this problem.

The data needs to be persisted after shutting down the server (e.g. save data to a File). You cannot assume that the management component is up first at all, so the clients have to react on unsuccessful lookup of the management components.

Ant template

As in the last assignment we provide a [template](#) build file (`build.xml`) in which you only have to adjust some class names, ports, and RMI names. Further on we give you the opportunity to use log4j. Do not use any other third-party libraries. Put your source code into the subdirectory "src", the `filesregistry.properties`, `loadtest.properties` and `user.properties` are already in the "src" directory (the ant compile task then copies this file to the build directory). Put the src directory including `registry.properties` and `build.xml` into your submission. Note that it's **absolutely required** that we are able to start your programs with the predefined commands!

Hints & Tricky Parts

- To make your object remotely available you have to **export** it. This can either be accomplished by extending `java.rmi.server.UnicastRemoteObject` or directly exporting it using the static method
`java.rmi.server.UnicastRemoteObject.exportObject(Remote obj, int port)`. Use 0 as port, so any available port is selected by the operating system.
- Before shutting down a server or client, unexport all created remote objects using the static method `UnicastRemoteObject.unexportObject(Remote obj, boolean force)` - otherwise the application may not stop.
- Since Java 5 it's not required anymore to create the stubs using the **RMI Compiler** (`rmic`). Instead java provides an automatic proxy generation facility when exporting the object.
- You should not use a Security Manager. So you do not need a policy files.
- Take care of **parameters and return values** in your remote interfaces. In RMI all parameters and return values except for remote objects are passed per value. This means that the object is transmitted to the other side using the java serialization mechanism. So it's required that all parameter and return values are serializable, primitives or remote objects, otherwise you will experience `java.rmi.UnmarshalExceptions`.
- To create a **registry**, use the static method
`java.rmi.registry.LocateRegistry.createRegistry(int port)`. For obtaining a reference in the client you can use the static method
`java.rmi.registry.LocateRegistry.getRegistry(String hostName, int port)`. Both hostname and port have to be read from the `registry.properties` file.
- We also provide a `registry.properties` file in the template. Make sure to set the port according to our policy.
- Reading in a **properties file** from the classpath (without exception handling):

```

java.io.InputStream is =
ClassLoader.getSystemResourceAsStream("registry.properties");
if (is != null) {java.util.Properties props = new
java.util.Properties();
try {props.load(is);
String registryHost = props.getProperty("registry.host");
...
}

} finally {is.close();}

}
} else {System.err.println("Properties file not found!");
}

```

Further Reading Suggestions

- **APIs:**
 - RMI: [Remote API](#), [UnicastRemoteObject API](#), [Registry API](#), [LocateRegistry API](#)
 - Properties: [Properties API](#)
 - IO: [IO Package API](#)
- **Tutorials**
 - [JavaInsel RMI Tutorial](#): German introduction into RMI programming.

Submission Guide

Submission

- Every group **must have** its own design/solution! Meta-group solutions will end in massive loss of points!
- As for group work usual, a protocol with the UML-Design, the work-sharing, the timetable and test documentation is mandatory!
- Upload your solution as a **ZIP** file. Please submit only the sources of your solution and the build.xml file (not the compiled class files and only approved third-party libraries).
- Your submission must compile and run! Use and complete the provided ant template.
- Before the submission deadline, you can upload your solution as often as you like. Note that any existing submission will be **replaced** by uploading a new one.

Interviews

- During the implementation there will be review interviews with the teams. Please be aware that the continuous implementation will be overseen and evaluated!
- After the submission deadline, there will be a mandatory interview.
- The interview will take place in the lesson. During the interview, every group member will be asked about the solution that everyone has uploaded (i.e.,**changes after the deadline will not be taken into account! There will be only extrapoints for nice and stable solutions!**). In the interview you need to explain the code, design and architecture in detail.

Points

Following listing shows you, how many points you can achieve:

- Design(10): usecase, uml-class, activity diagrams
- Documentation(10): timetable, explanation, description(JavaDoc), protocol/test documentation
- Implementation(40): exception handling, concurrency, resource handling, performance, functional requirements
- Testing(15): unit-testing, coverage, system-testing, user-acceptance

published by: Vienna University of Technology
Institute of Information Systems 184/1
Distributed System Group

Aufgabenteilung (Vereinfacht)

Für eine genaue Beschreibung bitte im Kapitel „Aufgabenteilung und Termine“ nachschauen.

Schuschnig:

- UML Klassendiagramm
- Analytics Server
- Connections
- Protokoll / Reviews
- Team Kontrolling

Lins:

- UML Klassendiagramm
- Billing Server
- Connections
- Technical Architect

Auradnik:

- UML Aktivitätsdiagramm
- Management Client

Klune:

- UML Usecasediagramm
- Billing Server
- Testing Component

Pöelzbauer:

- UML Aktivitätsdiagramm
- Testing Component
- Exceptions

Rieppel:

- UML Use Case Diagramm
- Testing (TDD)
- Management Client
- Model

Funktionsliste

Analytics Server:

- Speichern aller Events
- Statistische Auswertung in calculate()
- Benachrichtigen mittels processEvent()
- Anmelden und Abmelden für Benachrichtigungen suscribe und unsuscribe()

Billing Server:

- Festlegen der Price Steps (Hinzufügen/Entfernen)
- Ausgeben der Price Steps
- Erstellt für jeden Benutzer Bills
- Bills ausgeben

Management Client:

- Greift auf die Funktionen von BillingServer und AnalyticsServer zu
- Benachrichtigungen mit print() oder automatisch ausgeben

Testing Component:

- Testen des gesamten Systems durch automatisches erstellen von Auktionen und Usern und automatischem bieten
- Verschiedenen Testing Einstellungen siehe Kapitel: „Testing Component“

Designüberlegung Allgemein (Probleme / Überlegungen)

ConcurrentHashMap

Ab jetzt werden überall ConcurrentHashMaps verwendet. Diese haben eine bessere Performance und lösen die Probleme mit Threads die in dieses Programm oft verwendet werden.

Key: Sollte dabei immer das Attribut sein nach dem meistens identifiziert wird. Zum Beispiel: Username.

Probleme: Key muss unique sein?, Wenn nach anderem Attribut identifiziert wird?

Exceptions

Wenn es sinnvoll ist sollen Exceptions definiert werden diese werden müssen auch im UML Klassendiagramm vermerkt werden.

Offene Fragen Management Client

- Verbindungsaufbau + Eventmanagement
- Management von gleichen Ausgaben, bei mehreren Abonneten (subscriptions). Server oder Clientseitig.
- Implementierung Command Pattern
- Zweck des TcpConnectors

Designüberlegungen

UML Klassendiagramme

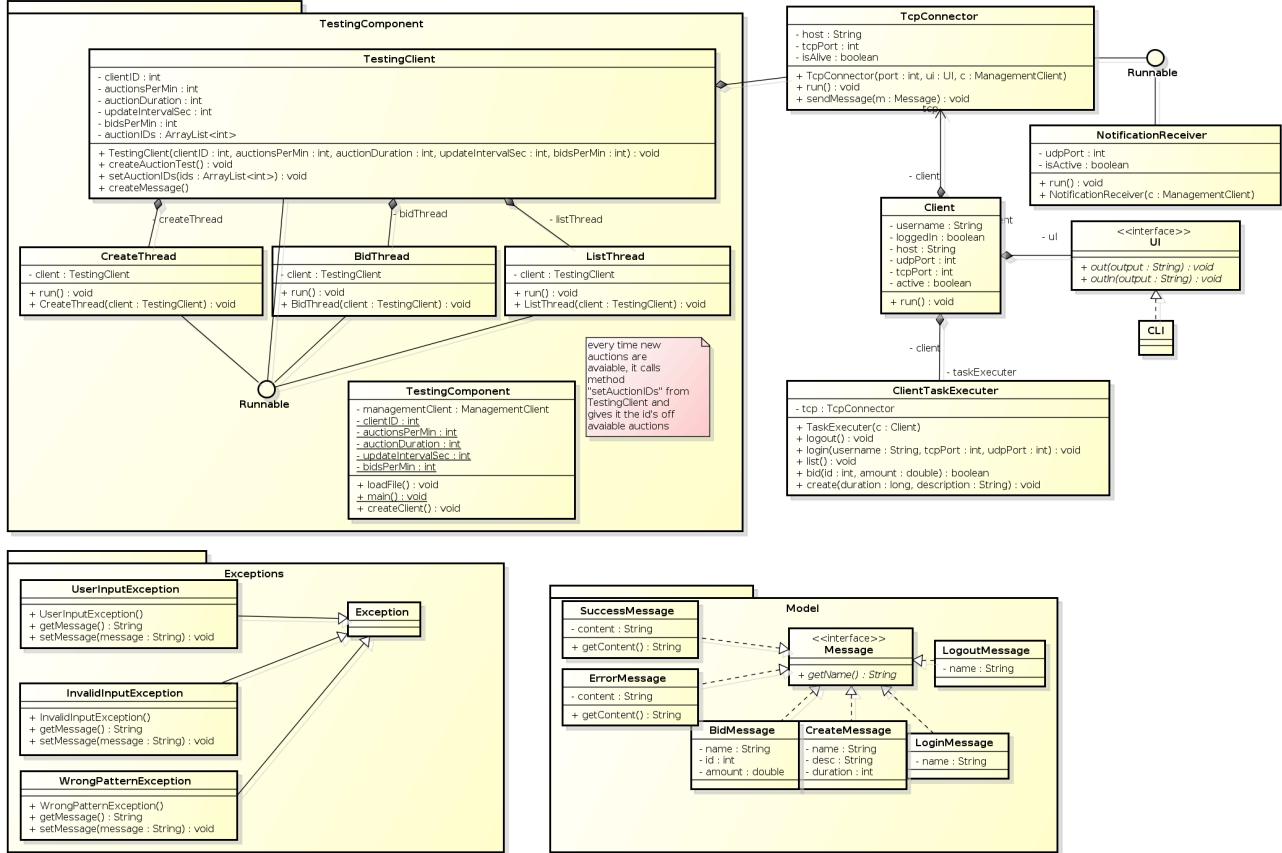


Abbildung 1: Hier sieht man den Client und die TestingComponent

Der Weitgehend unveränderte Teil aus Aufgabe07. Hier sieht man den Client wie er damals schon funktioniert hat allerdings wurden die UDP Benachrichtigungsteile gelöscht.

Des Weiteren sieht man die Erweiterung des TestingComponents dieser wird im Kapitel Testing Komponenten genau beschrieben.

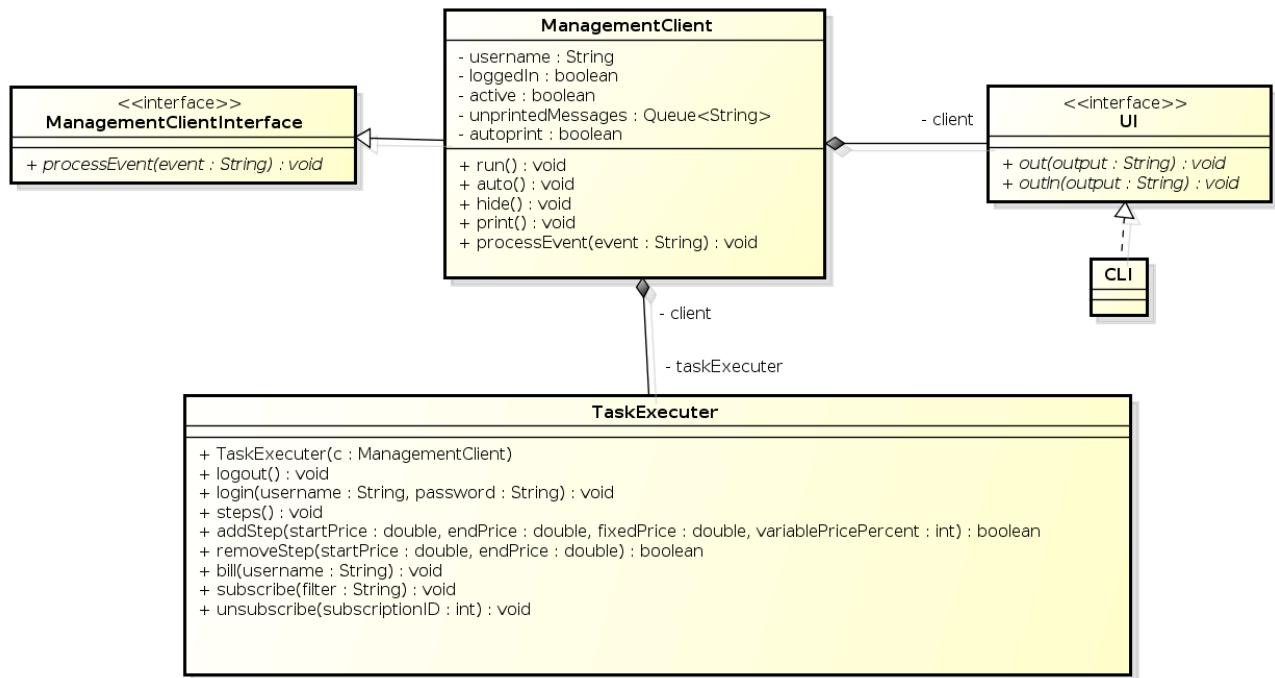


Abbildung 2: Das UML Diagramm des Management Clients

Dieser wurde im UML dem normalen Client nachempfunden. Es soll dabei an das bewährte Design des Clients festgehalten werden und dieser ähnlich programmiert werden.

Der TaskExcuter wird dabei immer aufgerufen wenn eine Methode nicht lokal ausgeführt wird sondern auf einem der Server aufgerufen wird. Aus diesem Grund verfügt der TaskExcuter für jede Funktion des ManagementClients eine Methode die mittels RMI dann die gewünschte Methode auf den Servern invoked.

Die Eingabe findet weiterhin ganz normal über die CLI statt. Dabei soll wieder das gleiche Konzept wie beim normalen Client verwendet werden

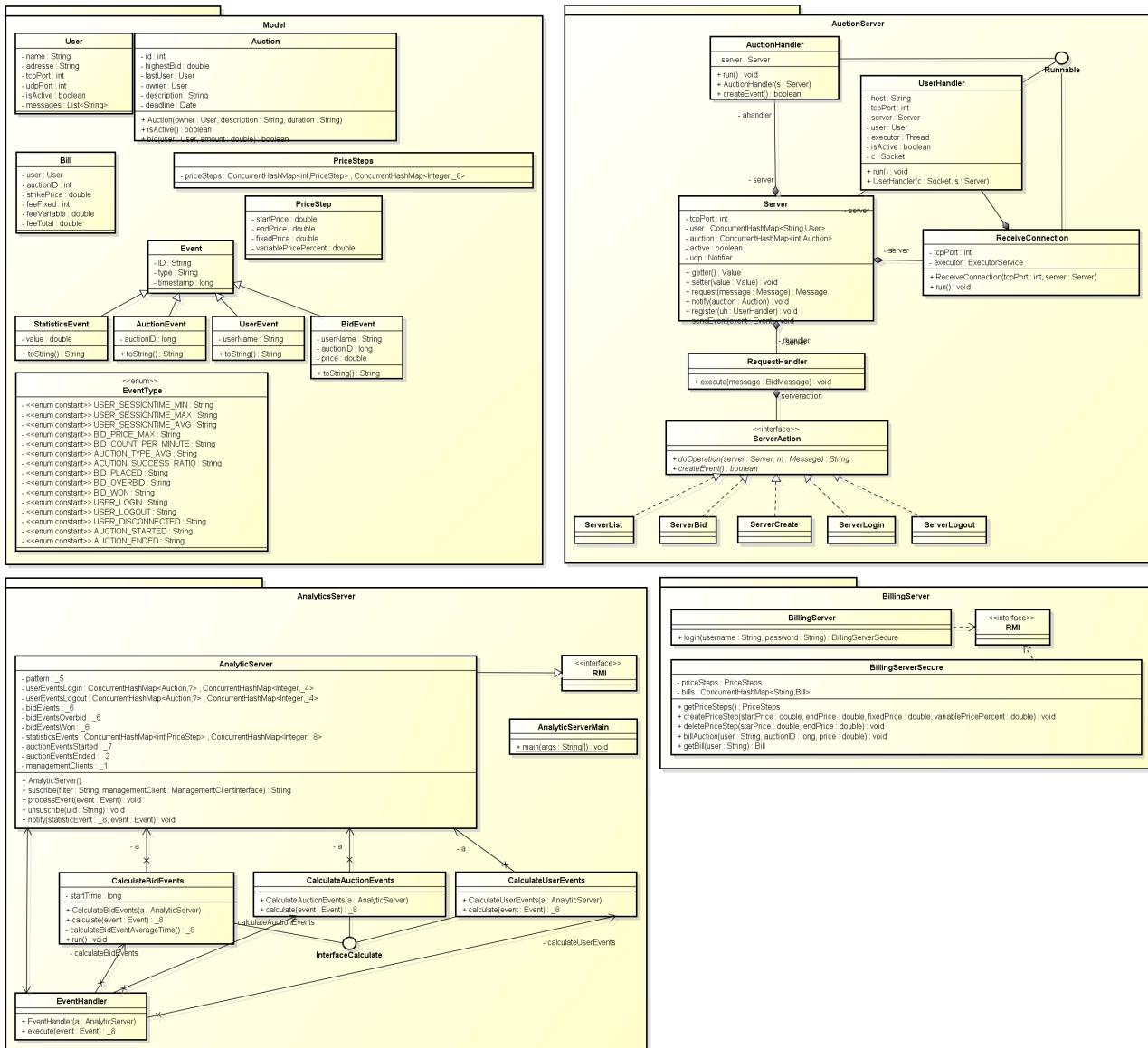


Abbildung 3: Hier sieht man alle Server im Überblick

Der Hauptserver auf dem die Auktionen laufen wurde dabei weitgehend unverändert gelassen bis auf den Unterschied, dass es keine UDP Benachrichtigungen mehr gibt. Deshalb musste sowohl der Server als auch der AuctionHandler leicht modifiziert werden. Der Server verfügt des weiteren über zwei neue Methoden. Die Methode createEvent() erzeugt wann immer eine Aktion geschieht ein Event (siehe Package Model) und schickt diese mittels der Methode sendEvent() über RMI an den AnalyticsServer. Dieser speichert die Events und wertet sie statistisch aus.

Die restlichen Teile des Servers bleiben weitgehend unverändert da sich dieser bewährt hat.

Neu dazu kommen eine AnalyticsServer und ein BillingServer diese sind jeweils über RMI mit dem Server verbunden und eine ManagementClient kann sich ebenfalls über RMI mit diesen verbinden. (Allerdings kann er sich nicht direkt mit dem Server verbinden).

Der AnalyticServer und der BillingServer werden in den folgenden Kapiteln genauer beschrieben.

UML Activity Diagramm

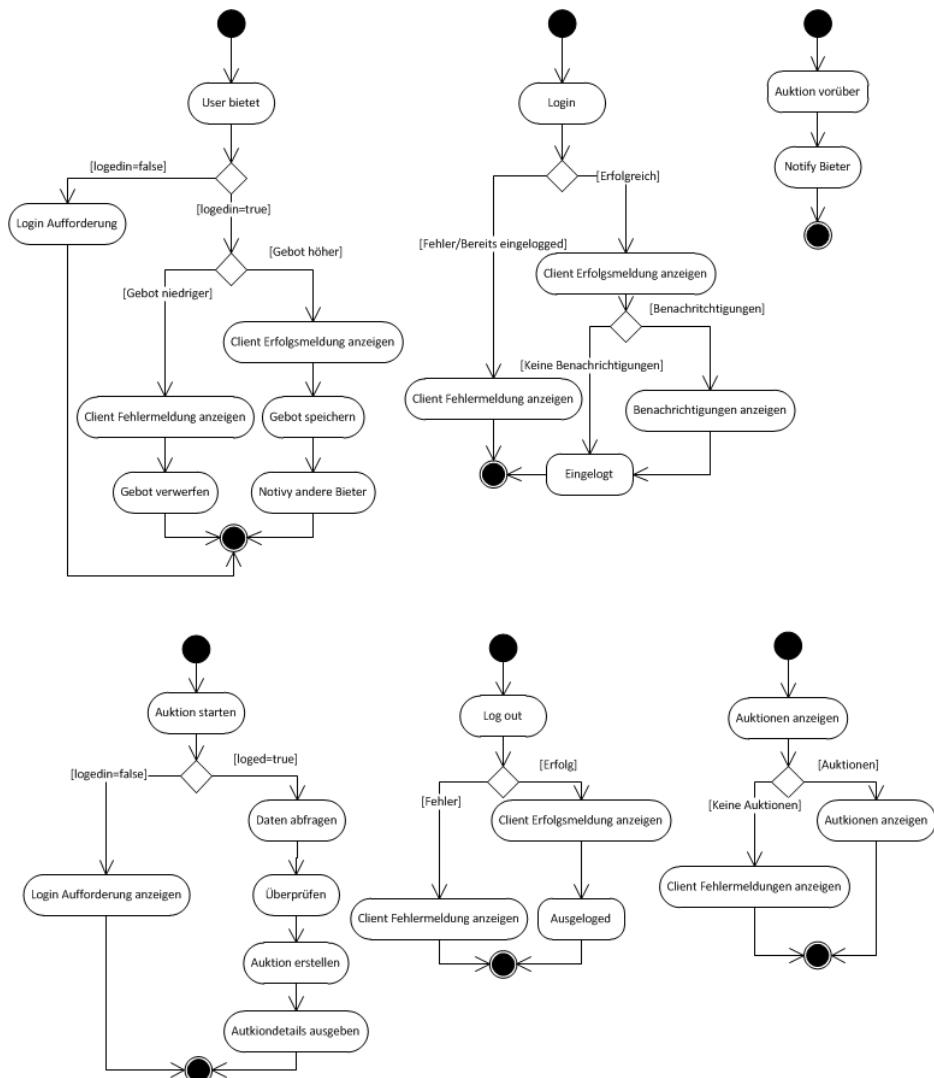


Abbildung 4: Zeigt das Activity Diagramm von Task07 mit allen Funktionen

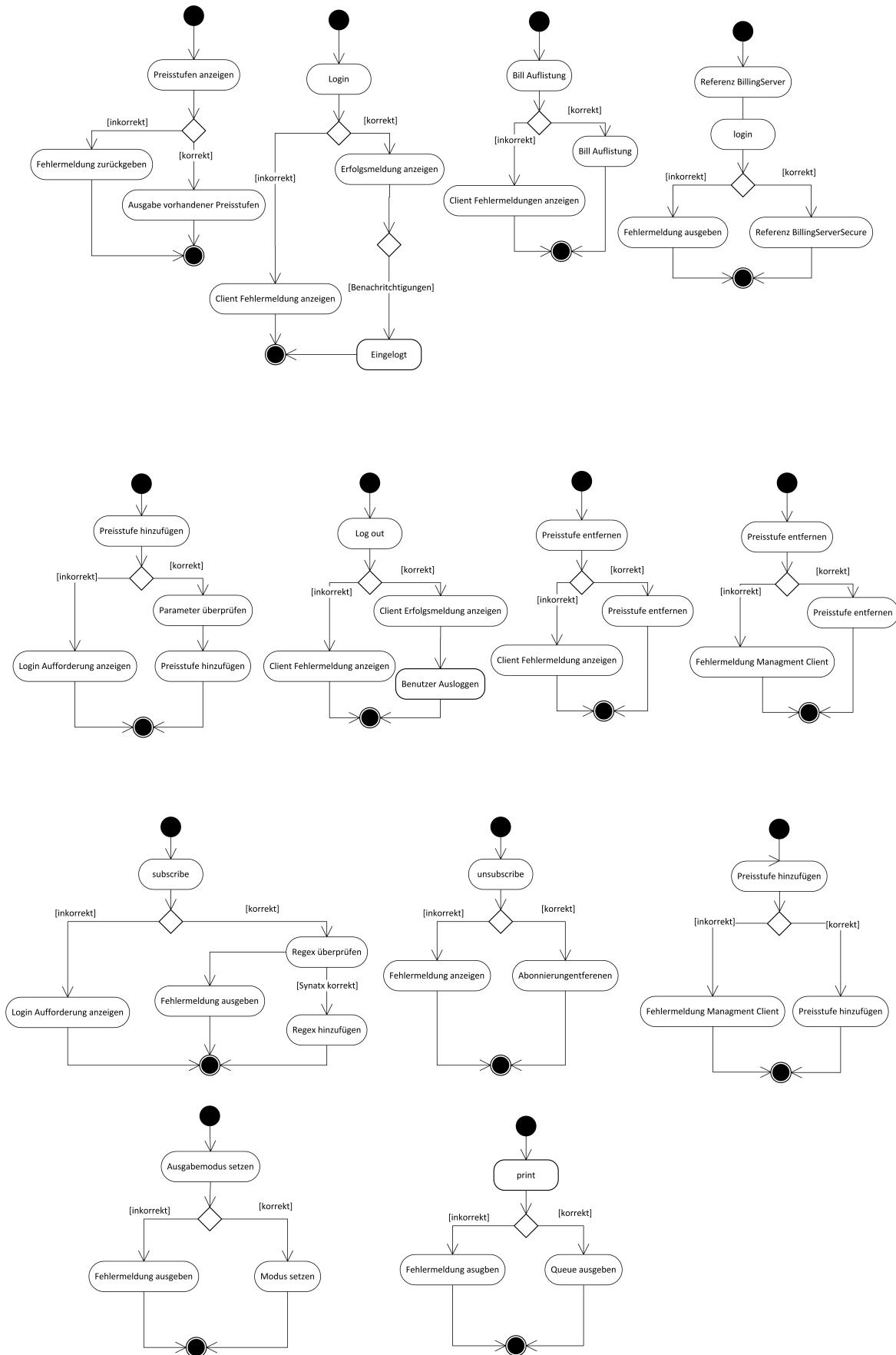


Abbildung 5: Zeigt einen Teil der neuen Funktionen aus Task08.

UML Use Case Diagramm

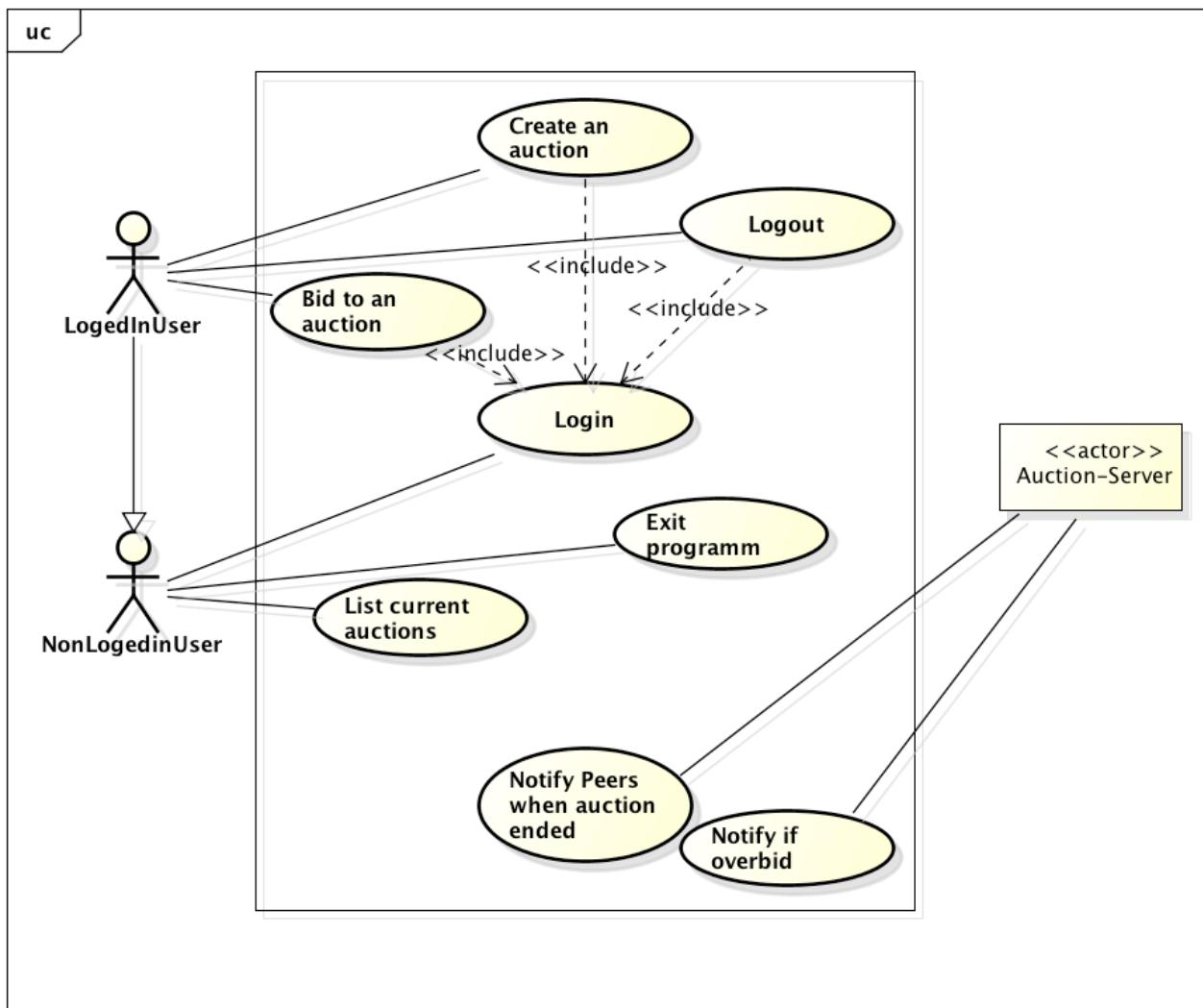


Abbildung U.1: Das Use Case Diagramm aus Task 07

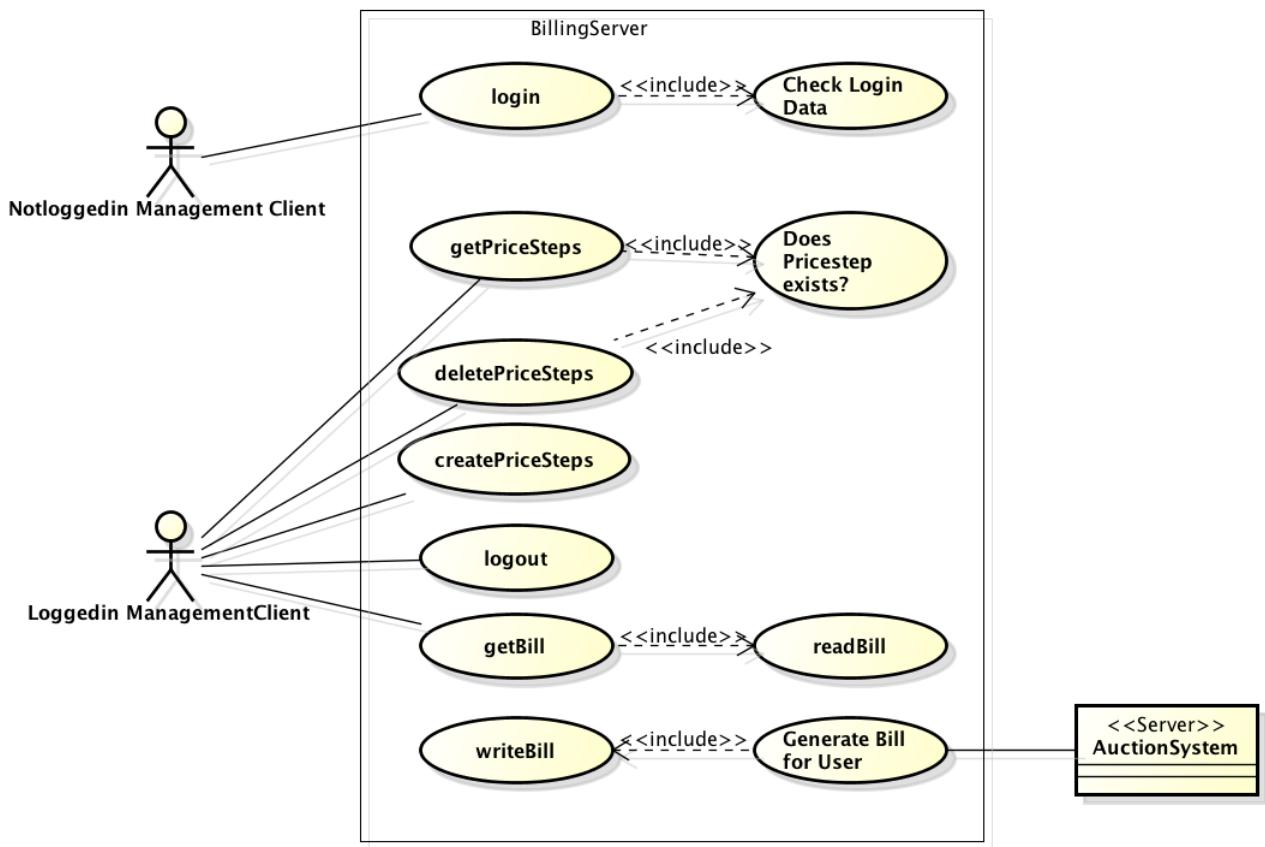


Abbildung U.2: Zeigt das Use Case Diagramm des Billing Servers

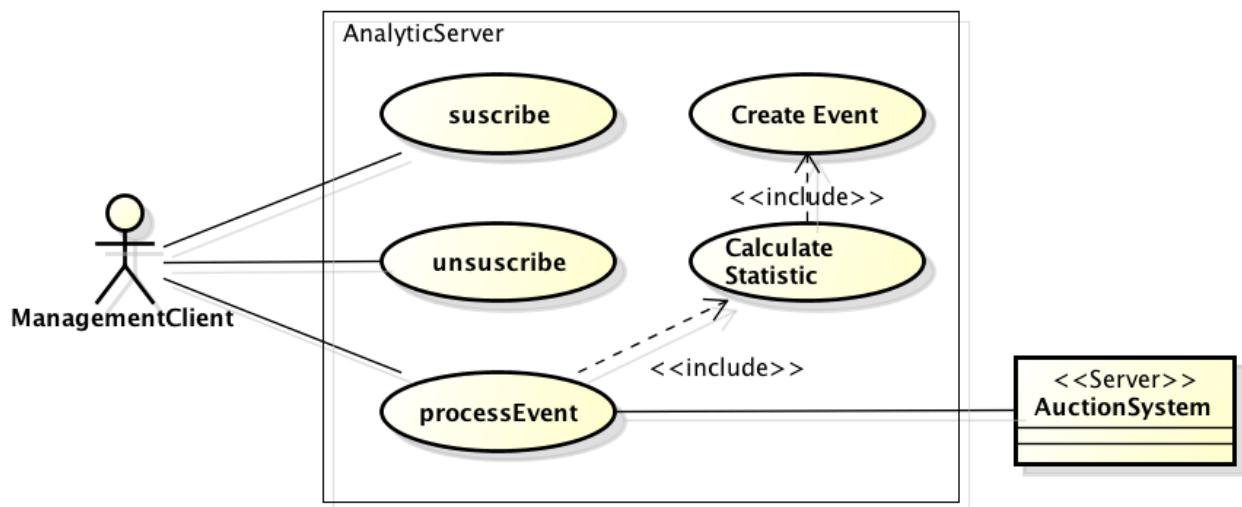


Abbildung U.3: Zeigt das Use Case Diagramm des Analytic Servers

Management Client

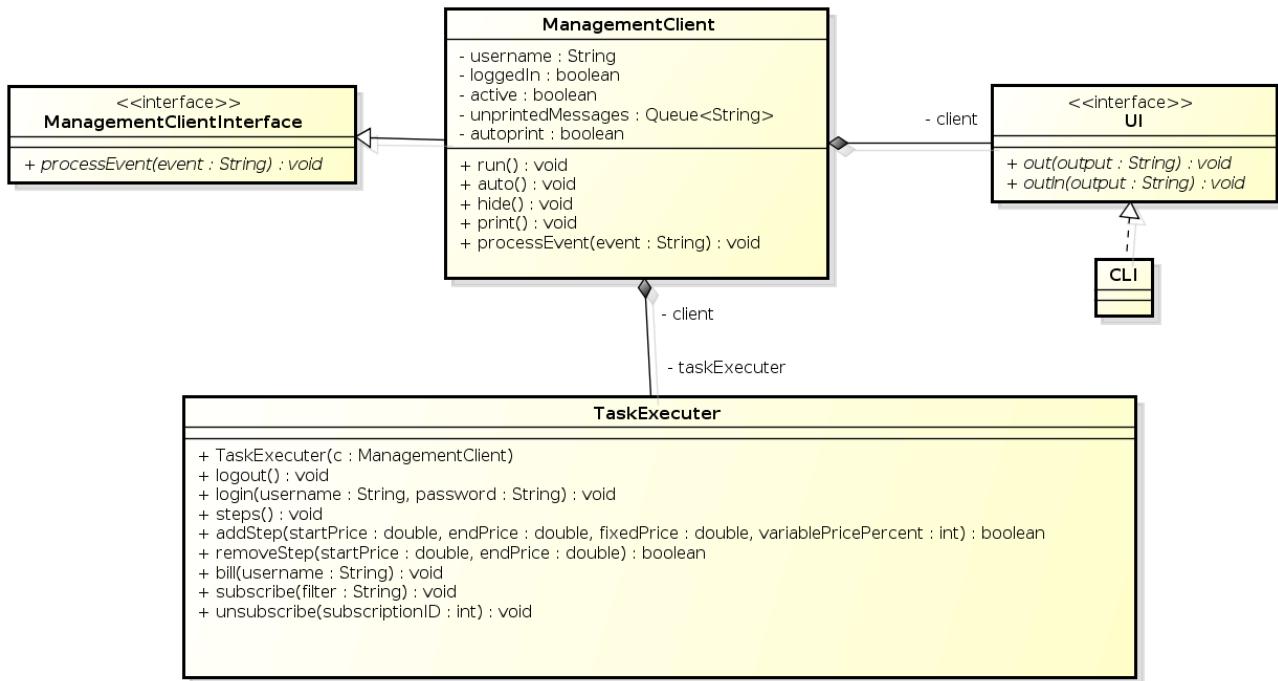


Abbildung 7.: Entwickeltes Klassenkonzept

Implementierung

Der Management Client hat die Aufgabe, mit Billing und Analytics Server zu kommunizieren und alle nötigen Befehle zu übermitteln.

Befehle um mit den Billing Server zu aggieren:

- `!login <username> <password>`: Anmelden mit spezifischen Daten
- `!steps`: Auflistung aller Preisstufen
- `! <startPrice> <endPrice> <fixedPrice> <variablePricePercent>`: hinzufügen einer neuen Preisstufe
- `!removeStep <startPrice> <endPrice>`: Entfernen einer Preisstufe
- `!bill <user_name>`: Zeigt die Gebote des gewählten User. Die Anzeige begrenzt sich auf Auktionen die bereits abgeschlossen worden sind. Zusätzlich werden die Auktions Abgaben angegeben, welche mittels der Abgabeliste ermittelt wird.

- !logout: Setzt den Verbindungsstatus zurück. Um auf Befehle, wie ‚bill‘ zugreifen zu können, muss sich der Management Client User wieder über das BillingServiceSecure remote Object mit dem Befehl ‚login‘ Verbinden.

Befehle um mit den Analytics Server zu aggieren:

- !subscribe <filterRegex>: Den Management ein bestimmtes Event abonnieren lassen. Mehrfache Abonnierungen möglich.

Beispiel:

```
!subscribe '(USER_.*)|(BID_.*)'
Created subscription with ID 17 for events using filter '(USER_.*)|(BID_.*)'
```

- !unsubscribe <subscriptionID>: Eine Abonnierung eines Events mittls ID wieder abmelden.

Befehlseingaben werden geprüft. Bei unbekannten Befehlen oder falschen Syntax wird der Client über das UI mit vordefinierten Errormessages verständigt.

Remote Message Management

Der Management Client implementiert die RMI Methode ‚processEvent‘. Die Methode wird von Analytic Server aufgerufen, wenn eine bestehende Event-Abonnierung (subscription) mit einem von Server generierten Event übereintrifft.

Es gibt zwei Arten die eingehenden Nachrichten anzuzeigen:

- *Automatic*
- *on-demand*

Dadurch ergeben sich folgende Befehle, um die Moduse zu wählen bzw. zu aggiren:

- !auto: Events werden automatisch, sofort ausgegeben. (none default)
- !hide: Events werden nicht ausgegeben und im Hintergrund gespeichert.
- !print: Gespeicherte Nachrichten (durch Befehl !hide), werden ausgegeben (Reihenfolge)

Offene Fragen

Verbindungsaubau + Eventmanagement

Management von gleichen Ausgaben, bei mehreren Abonneten (subscriptions). Server oder Clientseitig.

Implementierung Command Pattern

Zweck des TcpConnectors

Ergänzung 10.02.2014

-BillingServer fehlt eine logout Methode. Ist nötig, da der ManagementClient die Option hat ein logout aufzurufen.

-Die ceatePriceStep Methode hat bei den BillingServerSecure Parameter: double,double,double, double. Die Methode

addStep im Management Client hat aber Parameter: double,double,double, int.

-Die login methode vom BillingServer gibt das Object BillingServerSecure zurück, Ich weis leider nicht was ich

damit im Management Client tun soll. Einfach nur Client Aufrufen und Rückgabe von BillingServerSecure ignorieren?

-Ich musste leider mit mehr Attributen als im UML aufgezeichnet arbeiten, da es anders nicht möglich war(zb StringRemote-Adresse). Klassen und Methoden-Strukturen wurden eingehalten.

-Management Client muss die RMI Interfaces von BillingServer und Analytics Server implementieren. Fehlt im UML

-Management Client benötigt ebenfalls ein RMI Interface (für die processEvent Methode). Fehlt im UML.

Ressourcen

Billing Server

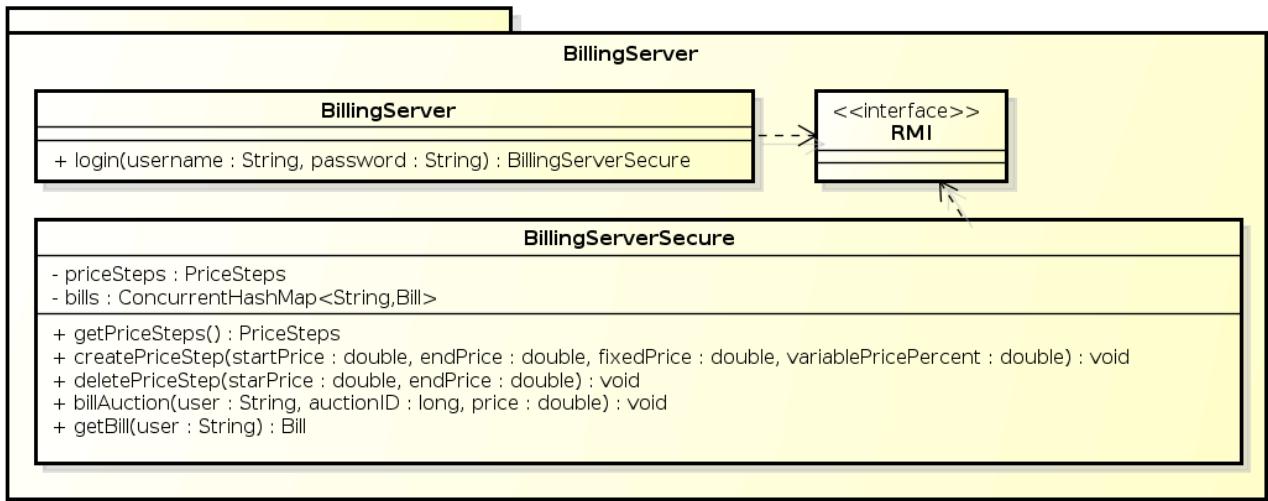


Abbildung 8: Genau veranschaulicht die für BillingServer relevanten Teile des UML's.

Der Billing Server ist für die Rechnungen des Auktion Systems verantwortlich.

Damit eine Sicherheit gegeben ist, wird es zwei RMI Objekte geben.

Das Objekt „BillingServer“ stellt uns nur eine Funktion zum login da, welche ein Management Client aufrufen kann. Falls dieser Login korrekt ist, wird die Referenz zum eigentlichen Billing Server, namens „BillingServerSecure“ weitergegeben.

Pro Auktion gibt es Gebüren. Diese Gebüren können von einem ManagementClient festgelegt werden.

Eine Gebür wird für einen bestimmten Preis-Bereich festgelegt.

Auction Price	Auction Fee (Fixed Part)	Auction Fee (Variable Part)
0	1.0	0.0 %
(0-100]	3.0	7.0 %
(100-200]	5.0	6.5 %
(200-500]	7.0	6.0 %
(500-1000]	10.0	5.5 %
> 1000	15.0	5.0 %

Abbildung 9: Auktionspreise

Auction Price: Kosten von bis

Fixed Part: Auktionskosten die fix festgelegt sind (zb 50€ -> 3€ kosten)

Variable Part: Hängt Prozentuell mit den Kosten zusammen

Je teurer die Auktion, desto weniger % zahlt man für den Variable Part.

Der BillingServer hat eine Lokale Datei namens user.properties welche die Userdaten enthält.

Das Passwort soll aus Sicherheitsgründen mit md5 gehasht sein (welches auch nicht sehr sicher ist)

Ein Eintrag sollte dann so ausschauen:

Admin=f23c5f9779a3804d586f4e73178e4ef0

Der BillingServerSecure bietet uns die Möglichkeit neue „PriceSteps“ wie oben beschrieben zu erstellen und diese zu löschen.

Desweiteren soll die Methode „billAuction“ die Relevanten informationen einer beendeten Auktion speichern, welche später für die Rechnungen verwendet werden sollen.

Eine solche Rechnung, kann dann mit der Methode „getBill(String username)“ abgerufen werden.

Exceptions

Exceptions die hier auftreten können:

- InvalidInputException ... diese tritt auf wenn die Username Passwort Kombination falsch ist
- UserInputException ... diese tritt auf wenn kein gültiger Befehl übergeben wird.

Ausgabe

Ausgabe der steps nach folgendem Schema:

Min_Price	Max_Price	Fee_Fixed	Fee_Variable
0	0	1.0	0.0 %
0	100	3.0	7.0 %
100	200	5.0	6.5 %
200	500	7.0	6.0 %
500	1000	10.0	5.5 %

Abbildung 10: Schema für ausgabe der Steps allerdings dann mit Ascii Zeichen gelöst.

login()

Die Sicherheit wird durch den Rückgabewert bei login() erreicht.

Wenn die Username / Passwort Kombination stimmt wird die gültige Referenz auf den BillingServerSecure zurückgegeben und der ManagementClient kann arbeiten.

Andernfalls als Rückgabewert *null* und es wird eine Exception erzeugt. (InvalidInputException)

Probleme:

PriceSteps hatte Konstruktor Parameter, die aber nicht nötig waren

PriceSteps wird nun ohne Parameter initialisiert

Kommentare und Ausgaben wurden von Herrn Klune in Deutsch verfasst

RMI:

RMI war nicht mehr sehr verständlich.

Wurde Anhand eines neuen Beispiels (Bank System) erneut angeschaut.

Ist nun verständlich im Vergleich zum CalculatePI

Sonst keine Probleme

Analytics Server

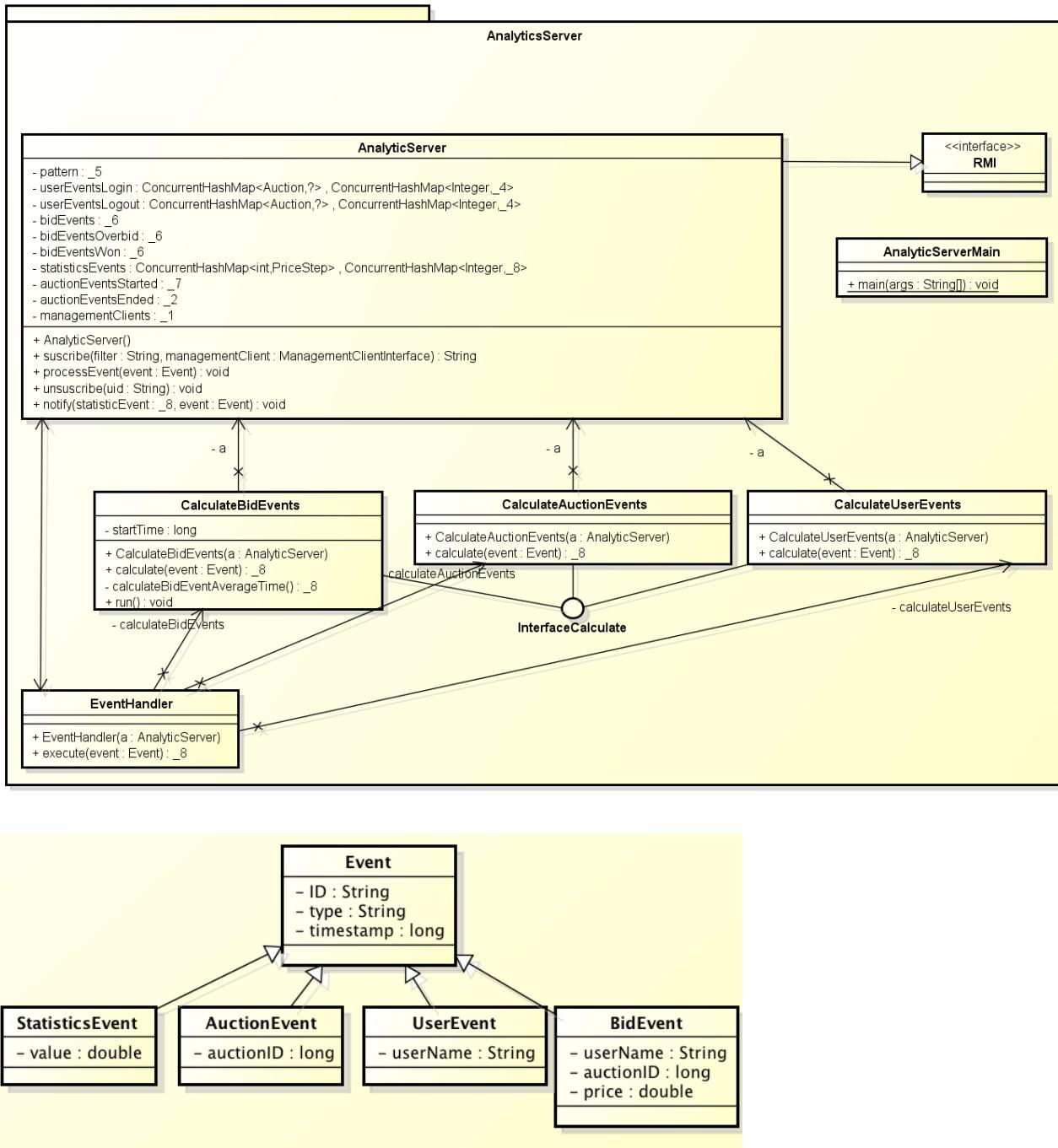


Abbildung 11: Analytics Server eventuell mit einem EnumTyp und das Model für Events

Der AnalyticsServer soll alle Aktionen die auf Auctionen passieren statistisch auswerten. Dazu werden alle Aktionen die geschehen in einem vorgefertigtem Model an den AnalyticsServer übertragen. Übertragen werden sie in sogenannten Events.

Dazu speichert er alle vom Server übertragenen Events in ConcurrentHashMap's. (Verwendung dieser siehe Designüberlegung Allgemein).

Unterschieden wird in:

- AuctionEvent: Werden bei Beginn und Ende einer Auktion erstellt
type:
 - AUCTION_STARTED
 - AUCTION_ENDED
- UserEvent: Werden bei jeder User Aktivität erstellt
type:
 - USER_LOGIN
 - USER_LOGOUT
 - USER_DISCONNECTE
- BidEvent: Bei jedem bieten auf eine Auktion erstellt
type:
 - BID_PLACED
 - BID_OVERBID
 - BID_WON
- StatisticEvent: Wird auf dem AnalyticsServer erstellt aufgrund statistischer Auswertung
type:
 - USER_SESSIONTIME_MIN
 - USER_SESSIONTIME_MAX
 - USER_SESSIONTIME_AVG
 - BID_PRICE_MAX
 - BID_COUNT_PER_MINUTE
 - AUCTION_TIME_AVG
 - AUCTIONS_SUCCESS_RATO

Die einzelnen Events die am Server erstellt werden, werden an die Methode processEvent weitergeleitet diese speichert die Events in der entsprechenden Liste und ruft dementsprechend die calculate() Methode auf diese überprüft ob sich eines der statistischen Events ändert und wenn ja wird dieses neu erstellt und wieder in die Liste statisticsEvent gespeichert.

Event: Datenelement: type

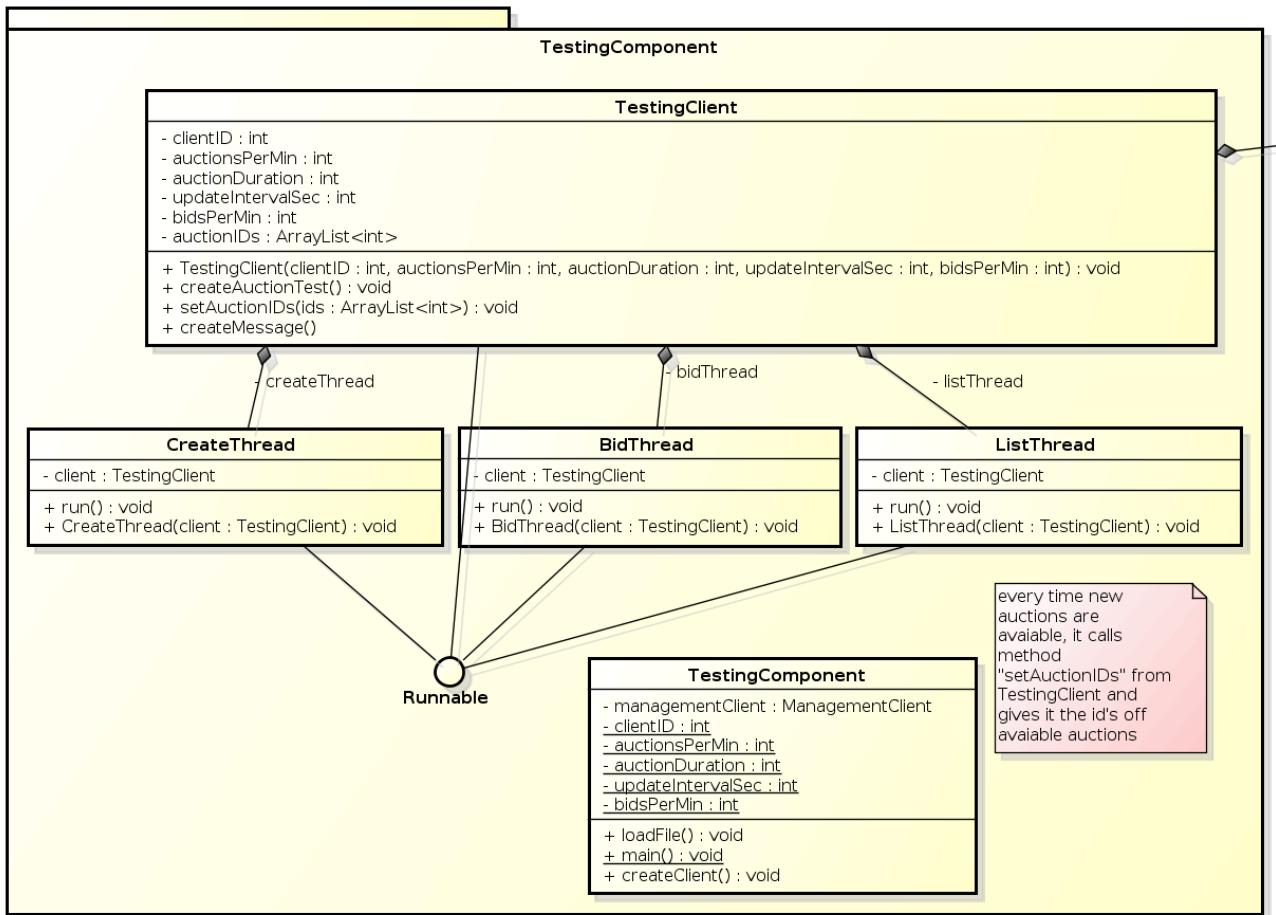
Eventuell als Enum definieren da es dann nur die vorgegebenen Werte haben kann und dieses leichter eingetragen werden können. (Frage ist ob dieses als String implementiert werden muss)

Benachrichtigungen:

Damit die Benachrichtigungen richtig weitergeleitet werden gibt es eine Liste mit allen ManagementClients die sich mit der subscribe Methode für Neuigkeiten angemeldet haben.

Bei dieser Map ist der Key der Filter damit leicht nachgeschaut werden kann bei wem der Filter zutrifft dieser wird dann als Unterscheidung verwendet wer die Nachriten bekommen soll.

Testing Component



Online-Auktionssysteme sollten in der Praxis gleichzeitig Robustheit und Skalierbarkeit bieten. Dies sollten sie auch für eine große Anzahl von Kunden gewährleistet werden.

Concurrency Probleme sind schwer zu erkennen, im Normalbetrieb, aber die Erzeugung von künstlicher Last auf dem System kann helfen, Probleme und Widersprüche zu erkennen.

Daher wird eine Lasttest-Komponente erstellt, um die Skalierbarkeit und Zuverlässigkeit des Systems zu testen.

Die folgenden Funktionen werden im Zuge dieser Lasttest-Komponente implementiert:

- Clients: Anzahl der gleichzeitig bietenden Kunden.
- auctionsPerMin: Anzahl von gestarteten Auktionen per Client per Minute.
- auctionDuration: Dauer der Auktion in Sekunden.
- updateIntervalSec: Anzahl der Sekunden die vergehen müssen bevor die Clients die Liste der aktiven Auktionen updaten.
- bidsPerMin: Anzahl der Gebote auf (zufällige) Auktionen die platziert werden, pro Client por Minute.

Um dies zu implementieren werden die Klassen CreateThread, BidThread und ListThread erstellt.

Diese werden eine Aggregation zur Klasse TestingClient haben, aus welcher die Methode setAuctionsIDs aufgerufen wird wenn neue Auktionen verfügbar sind.

Natürlich müssen diese Klassen auch Runnable implementieren.

Weiters besteht eine Aggregation zwischen TestingClient und TcpConnector.

Genaueres ist dem UML-Klassendiagramm zu entnehmen.

Die Methode readFile aus der Klasse TestingComponent liest die Date aus dem properties-File. Danach werden die TestingClients (Objekte der Klasse TestingClient) anhand dieser Daten erstellt. Diese Clients erstellen dann Auktionen und bieten auf diese, anhand der Daten die sie aus dem properties-File übergeben bekommen haben. Um die Befehle auszuführen dienen die Klassen BidThread, CreateThread und ListThread

Durch die Methode createMessage werden die Messages aus dem Model-Package erstellt und ausgeführt.

Änderungsvorschläge:

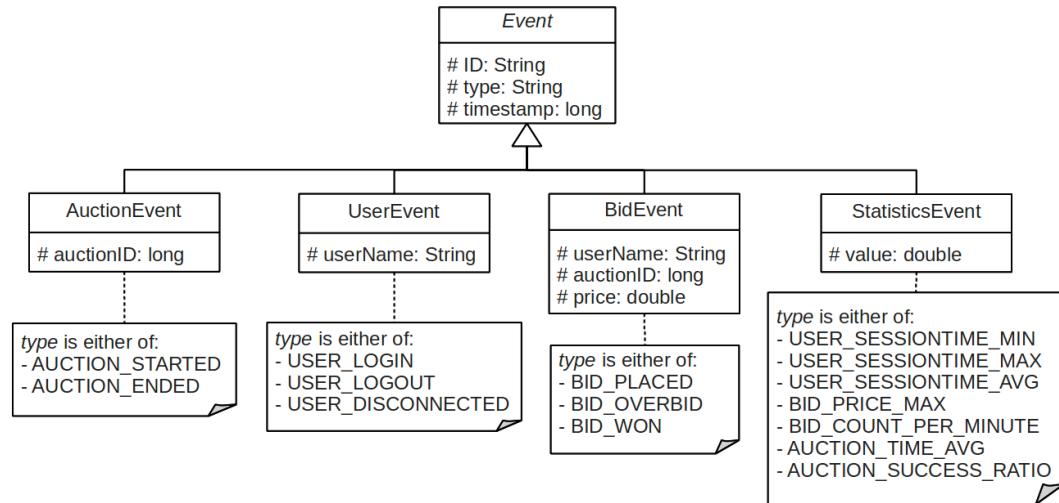
1. In den Klassen UserInputException und InvalidInputException sollten die getter und setter Methoden, für die Message, verworfen werden. Da diese meiner Meinung nach keinen Zweck erfüllen. Denn einen Exception sollte and ein except angepasst werden und nicht umgekehrt.
--> Verbesserungsvorschlag Schuschnig: einen weiteren Konst. mit dieser Message als param (auch nicht viel besser aber OK)
2. Die Methode createMessage aus der Klasse TestingComponent muss in die Klasse TestingClient verschoben werden da diese in eigentlich in TestingClient aufgerufen werden sollte.
3. Die Attribute aus der Klasse TestingClient (jene die aus dem properties-File ausgelesen werden) müssen ebenfalls in der Klasse TestingComponent hinzugefügt werden. Ansonsten wird es nicht möglich sein die einzelnen TestingClients zu erstellen.

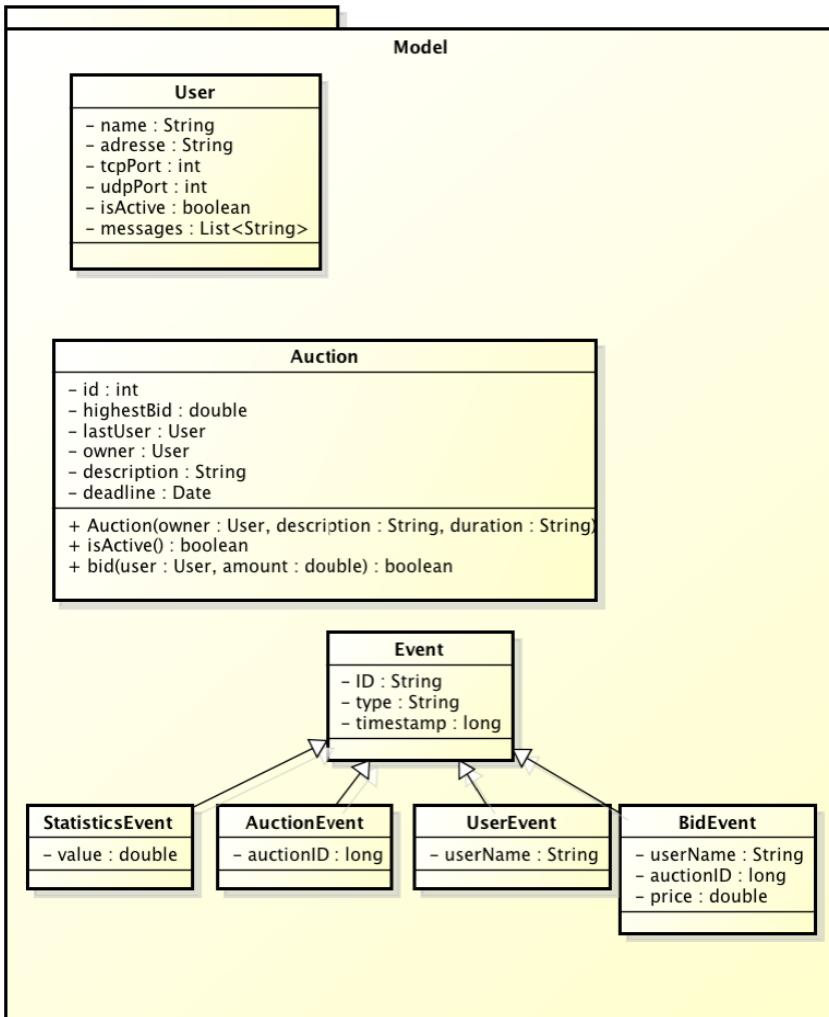
4. Im UML fehlen die Parameter der Methode createClient aus der Klasse TestingComponent.
5. In der Klasse TestingComponent wird eine ConcurrentHashMap benötigt um die TestingClients zu erstellen. (muss im UML-Klassendiagramm als Attribut hinzugefügt werden)
6. Die createClient Methode aus der Klasse TestingComonent sollte verworfen werden da diese Methode aus genau einem Konstruktor aufruf bestehen würde. Die Funktionalität kann in die main Methode verschoben werden.
7. Da sich in der Klasse TestingComponent eine main Methode befindet müssen diese static sein und ebenfalls alle Attribute und die Methode readFile da diese in der Main Mehtde verwendet werden.

Model (Nachtrag)

Neben den Klassen Auction, User und dem Interface Messages, die bereits in Task07 beschrieben sind, zählen laut UML noch die verschiedenen Events zum Model welche vom Auction Server an den Analytics Server gesendet werden sollen, um Aufzeichnungen zu machen. Dazu gibt es eine Hauptklasse Event und zusätzlich noch die verschiedenen Typen von Events, die von der Hauptklasse erben. Je nachdem was am Auction Server passiert, wird das entsprechende Event an den Analytics Server geschickt. Die einzelnen Eventtypen enthalten alle relevanten Daten, welche später vom Analytics Server aufgezeichnet werden.

Unten sind die entsprechenden Gründe für die Events aufgelistet.





Messages (alt)

Das Model beschreibt alle Messages die vom Client gesendet werden. Es wird durch ein Interface und den jeweiligen Implementierungen dieses Interfaces repräsentiert. Das Model besteht aus den Objekten SuccessMessage, ErrorMessage, BidMessage, CreateMessage, LoginMessage, LogoutMessage. Das Objekt SuccessMessage wird an den User bei einem erfolgreichen Vorgang gesendet. Sie besitzt das Attribut Content in Form eines Strings, für den Inhalt der Message und eine Getter Methode um diesen Content zu erhalten. Das Objekt ErrorMessage wird dem User bei einem fehlgeschlagenen Vorgang gesendet. Ansonsten ist es vom Aufbau her genau gleich wie die SuccessMessage. Das Objekt BidMessage wird dem Auction Server gesendet um auf eine Auktion zu bieten. Es besitzt den Namen des Clients, die ID der Auktion und die Höhe des Betrags welcher geboten werden soll als Attribut. Das Objekt CreateMessage wird dem Server gesendet um eine neue Auktion zu erstellen. Sie besitzt wiederum den Namen des Users als Attribut, allerdings auch noch die Beschreibung der Auktion und die Dauer der Auktion. Die Objekte LoginMessage und LogoutMessage werden an den Server gesendet um sich einzuloggen bzw. auszuloggen. Mitgesendet wird der Name des Users der eingeloggt werden soll als Attribut.

Alle Messages besitzen die Methode getName() welche vom Interface bereitgestellt wird. Diese Methode ist dazu da um sich die Namen der Messages zu holen.

Connections

Alle Verbindungen zwischen den neuen Servern werden über RMI gelöst. Die alten Verbindungen über TCP bleiben dabei erhalten.

UDP Verbindungen und Nachrichten aus Task07 werden aus dem Programm entfernt.

Die Verbindung zwischen ManagementClient und AnalyticServer wurde mit Callbacks gemacht.

Dies ist möglich in dem man ein Interface des ManagementClients speichert mit einer subscribe Methode:

```
public String suscribe(String filter, ManagementClientInterface managementClient)
```

Anschließend kann man sich auch unsubscribe:

```
public void unsuscribe(String uid) {
```

In dieser Methode wird nur der ManagementClient gelöscht.

Die restlichen Verbindungen wie sie zu sehen sind in der Architektur wurden so gelöst:

```
public interface AnalyticServerInterface extends Remote {
    /**
     * Suscribes with a regular expression for events
     * @param filter The regular Expression
     * @param managementClientInterface The Interface for the callback
     * @return rturns the String of the suscription
     * @throws RemoteException throws a Remote exception
     * @throws PatternSyntaxException When the Pattern is invalid this is thrown
     * @throws InvalidFilterException When the Pattern is invalid this is thrown
     */
    public String suscribe(String filter, ManagementClientInterface managementClientInterface)
throws RemoteException, PatternSyntaxException, InvalidFilterException;;
    /**
     * Receives events from the system and computes simple statistics/analytics.
     * processes the events to Taskexecuter
     * @param event the processed event
     * @throws RemoteException throws a Remote exception
     */
    public void processEvent(Event event) throws RemoteException;;
    /**
     * Unsuscribe from events
     * @param uid the id of the cliet who has suscribed
     * @throws RemoteException throws a Remote exception
     */
    public void unsuscribe(String uid) throws RemoteException;;
}
```

Danach kann man den Server binden wenn dieser das Interface implementiert:

```
public class AnalyticServer implements AnalyticServerInterface{
```

Hier wird der Server dann gebindet:

```
public static void main(String args[]){
    if (args.length != 1) {
        System.err.println("Wrong Arguments!\nCorrect: AnalyticServer Name");
        System.exit(0);
    }
    Properties p=new Properties("registry.properties");
    int port=Integer.parseInt(p.getProperty("registry.port"));
    try {
        try{
            r=LocateRegistry.createRegistry(port);
        }catch (Exception e) {
            r=LocateRegistry.getRegistry(port);
        }

        AnalyticServer analyticServer = new AnalyticServer();
        AnalyticServerInterface analyticServerInterface =
        (AnalyticServerInterface)UnicastRemoteObject.exportObject(analyticServer, 0);
        r.rebind(args[0], analyticServerInterface);
        System.out.println("AnalyticServer bound");
    }
    catch (Exception e) {
        System.err.println("Wrong Arguments!\nCorrect: AnalyticServer Name");
    }
}
```

Der Lookup:

```
try {
    obja = (AnalyticServerInterface)
Naming.lookup("rmi://"+host+":"+port+"/AnalyticServer");
} catch (Exception ex) {
    System.err.println("Server exit: Cannot connect to AnalyticsServer: AnalyticServer");
    return;
}
```

Die Verbindung mit dem BillingServer wurde genauso gelöst.

Exceptions

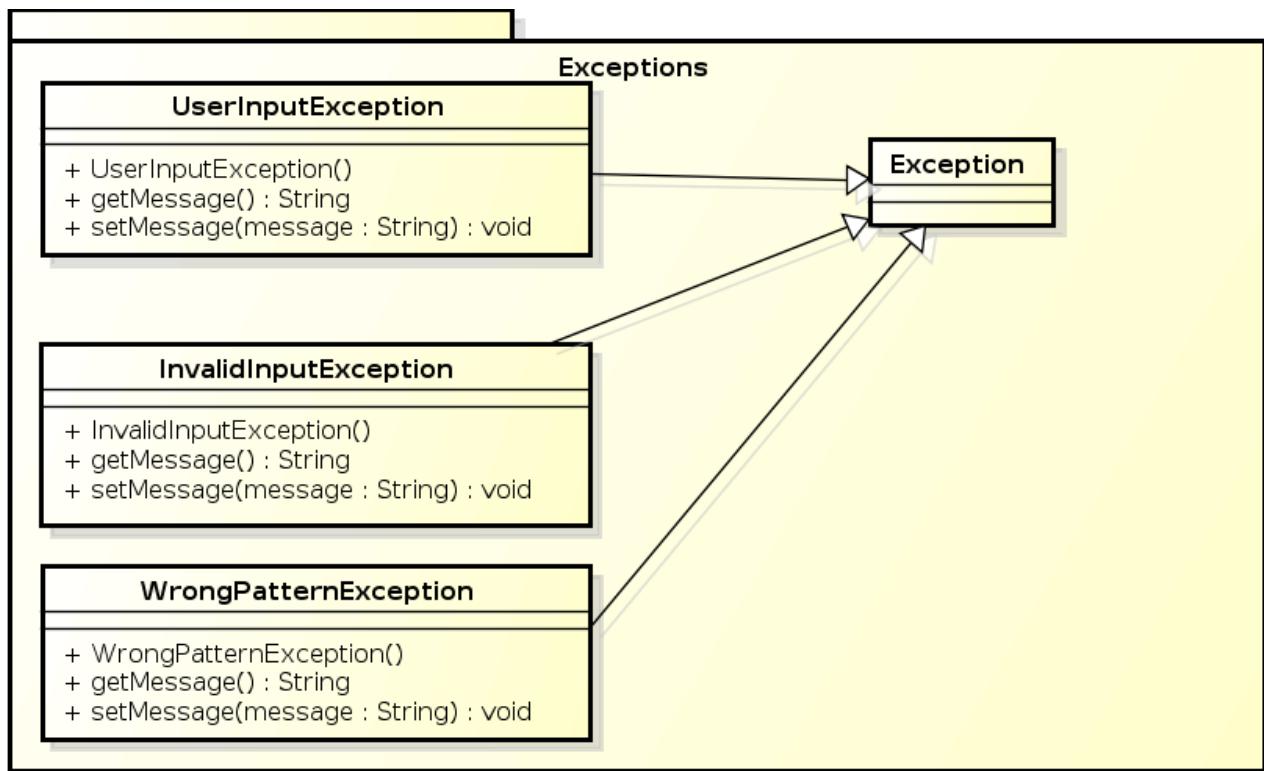


Abbildung E.1: Die bereits definierten Exceptions.

UserInputException:

Diese Exception wird aufgefangen wenn ein User beim Versuch sich einzuloggen die falsche Syntax für den !login-Befehl verwendet.

InvalidInputException:

Diese Exception wird aufgefangen wenn ein User beim Versuch sich einzuloggen die falsche Syntax für den:

!addStep-Befehl

!removeStep-Befehl

!bill-Befehl

!subscribe-Befehl

!unsubscribe-Befehl

(!create-Befehl)

(!bid-Befehl)

verwendet.

Ebenfalls mögliche Exceptions:

InvalidCommandException:

Die Exception könnte geworfen werden falls der User versucht einen Befehl einzugeben der nicht implementiert wurde.

Zeitschätzung und Tatsächliche Zeit

Tobias Schuschnig		
Arbeitspakete	Geschätzte Zeit	Tatsächliche Zeit
Teamkoordination	1,5	2
UML Klassendiagramm	2	4
Analytics Server Designüberlegung	1	0,5
Conections Designüberlegung	1	0,3
Analytics Server Implementierung	2	2,5
Conections Implementierung	1,5	0,5
Analytics Server Fertig	3	5
Conections Fertig	2	1,5
Protokoll	1	1,5
Anpassungen und Abstimmungen	1,5	2,5
User Acceptance Testing	1	1,5
Testing Component		4,5
Summe	16,5	26,3

Tobias Lins		
Arbeitspakete	Geschätzte Zeit	Tatsächliche Zeit
UML Klassendiagramm	3	6
Protokoll	1,50	
Billing Server Designüberlegung	1	1
Conections Designüberlegung	1	1,5
Billing Server Implementierung	5	5
Conections Implementierung	2,50	2
Billing Server Fertig	1	1,5
Connections Fertig	1	0,5
Anpassungen und Abstimmungen	1	1
Ant	1	1,5
User Acceptance Testing	1	1
Summe	19	21

Alexander Auradnik		
Arbeitspakete	Geschätzte Zeit	Tatsächliche Zeit
UML Aktivitätsdiagramm	2	2,5
Management Client Designüberlegung	2,5	1,2
Management Client Implementierung	5	4,5
TDD		1,5
Management Client Fertig	3	4
Anpassungen und Abstimmungen	3	3
User Acceptance Testing	2,5	2,5
Summe	18	19,2

Alexander Klune		
Arbeitspakete	Geschätzte Zeit	Tatsächliche Zeit
Use Case	5	2,5
Billing Srv Design	4	0
Testing Comp. Design	4	0,5
Billing Srv. Impl	10	7
Testing Comp. Impl	10	5
Anpassen & Abstimmen	4	
Summe	37	15

Patrick Pölzbauer		
Arbeitspakete	Geschätzte Zeit	Tatsächliche Zeit
UML Aktivitätsdiagramm	1	0,1
Testing Component Designüberlegung	1	1,2
Exceptions Designüberlegung	1	1
Testing Component Implementierung	6	4,5
Exceptions	1	2
User Acceptance Testing	2	2
Summe	12	10,8

Alexander Rieppel		
Arbeitspakete	Geschätzte Zeit	Tatsächliche Zeit
UML Use Case Diagramm	2	2
TDD (ersten Testfälle)	1	2
Recherche Testfälle Designüberlegung	1	1
Management Client Designüberlegung	1	0
Model Designüberlegung	1	1
TDD fertig	5	5
Management Client Implementierung	2	2
Model fertig	1	1
Unit Testing	5	8
Beschreibung Testing	1	3
Anpassungen	2	2
User Acceptance	2	0
Summe	24	27

Gesamt		
Teammitglieder	Geschätzte Zeit	Tatsächliche Zeit
Tobias Schuschnig	16,5	26,3
Tobias Lins	19	21
Alexander Auradnik	18	19,2
Alexander Klune	37	15
Patrick Pölzelbauer	12	10,8
Alexander Rieppel	24	27
Summe	126,5	119,3

Arbeitsaufteilung und Termine

Arbeitspakete	Schuschnig	Lins	Auradnik	Klune	Pölzlbauer	Rieppel	Termin	Erledigt (Mängel)
UML Klassendiagramm	x	x					27.01.14	x
UML Aktivitätsdiagramm			x		x		27.01.14	x
UML Use Case Diagramm		x		x		x	27.01.14	nachgereicht
TDD (ersten Testfälle)			x			x	27.01.14	nein
Recherche Testfälle			x			x	27.01.14	nein
Designüberlegung								
Management Client			x			x	27.01.14	x
Designüberlegung								
Billing Server Designüberlegung		x		x			27.01.14	x
Analytics Server	x						27.01.14	x
Designüberlegung								
Testing Component				x	x		27.01.14	x
Designüberlegung								(Mängel!)
Conections Designüberlegung	x	x					27.01.14	x
Exceptions Designüberlegung					x		27.01.14	nachgereicht (Mängel)
Model Designüberlegung						x	27.01.14	nachgereicht (Mängel)
Zeitaufzeichnung und Zeitschätzung	x	x	x	x	x	x	27.01.14	x
Review 1 mit Protokoll	x						29.01.14	
TDD Testing Fertig			x			x	10.02.14	x (laufend)
Management Client Implementierung			x			x	10.02.14	x
Billing Server Implementierung		x		x			10.02.14	x
Analytics Server Implementierung	x						10.02.14	x

Testing Component Implementierung				x	x		10.02.14	x (pushen)
Conections Implementierung	x	x					10.02.14	
Exceptions					x		10.02.14	x (Mängel)
Model Fertig						x	10.02.14	x
Review 2 mit Protokoll	x						12.02.14	
Management Client Fertig			x			x	17.01.14	x (Bug fixing)
Billing Server Fertig		x		x			17.01.14	x (Bug fixing)
Analytics Server Fertig	x						17.01.14	x (Bug fixing)
Testing Component Fertig				x	x		17.01.14	nein
Conections Fertig	x	x					17.01.14	x
Exeptions Fertig					x		17.01.14	nein
Review 3 mit Protokoll	x						19.02.14	
Uni Testing			x			x	22.02.14	x
Beschreibung Testing			x			x	22.02.14	x
Beschreibung Applikation	x						23.02.14	
Anpassungen und Abstimmungen	x	x	x	x	x	x	23.02.14	
Ant		x					24.02.14	
User Acceptance Testing	x	x	x	x	x	x	24.02.14	
Abgabe	x	x	x	x	x	x	25.02.14	
Finanl Review mit Protokoll	x						26.02.14	

Abnahme der Arbeitspakete:

UML Klassendiagramm

Von Tobias Schuschnig und Tobias Lins durchgeführt keine Mängel festgestellt.

Vor dem Termin abgegeben.

UML Aktivitätsdiagramm

Durchgeführt von Alexander Auradnik und Patrick Poelzbauer. Vorhanden mit Mängel. Schreibfehler. Nicht als .jpg sonder als .png. Des Weiteren ohne großem weißem Rand auf der Seite. Bitte nachbessern. (Bitte an Abbildung 4 richten, hier sieht man das gewünschte Format aus Task08)

Nachgereicht zur vollsten Zufriedenheit.

UML Use Case Diagramm

Durchgeführt von Alexander Klune und Alexander Rieppel. Nicht vorhanden.

Etwas nachgereicht.

TDD (ersten Testfälle)

Durchgeführt von Alexander Rieppel. Die kreierten Testfälle entsprechen leider nicht der Übungsvorgabe. Es ist nicht Sinn der Sache selbstverständliche Testfälle wie getter() und setter() Methoden zu testen. Des Weiteren ist es nicht zweckmäßig TDD Testfälle für ein Package Model zu schreiben dessen Code bereits existiert und zu einer Coverage von 72.63(!) getestet wurde. Bitte überleg dir nocheinmal was du zu testen hast.

Nachgereicht ebenfalls nicht zufriedenstellend. (Gleichen Gründe wie oben)

Recherche Testfälle Designüberlegung

Durchgeführt von Alexander Rieppel. Nicht vorhanden.

Nachgereicht: Unzureichend nicht genau beschrieben an welcher stelle was.

Management Client Designüberlegung

Durchgeführt von Alexander Auradnik. Gut überlegt und auf den Punkt gebracht das gesamte Arbeitspaket erledigt.

AnalyticsServer Desingüberlegung

Durchgeführt von Tobias Schuschnig. Es wurden keine Mängel festgestellt.

BillingServer Desingüberlegung

Durchgeführt von Tobias Lins. Kleine Mängel festgestellt allerdings schon ausgebessert.

Testing Component Designüberlegung

Durchgeführt von Patrick Poelzbauer. Vorhanden allerdings fehlen wesentliche Teile des Arbeitspaketes. Der für diesen Teil relevante Teil des UML's sollte hier nocheinmal angeführt werden. Des Weiteren fehlen Probleme die hier auftreten können. (Genauere Beschreibungen sind nicht dem UML zu entnehmen sondern sollen hier beschrieben werden). Bitte nachbessern. (Mögliche Vorlagen: AnalyticsServer Designüberlegung, ManagementClient Designüberlegung)

Connections Designüberlegung

Durchgeführt von Tobias Lins und Tobias Schuschnig.

Exceptions Designüberlegung

Durchgeführt von Patrick Poelzbauer. Nicht vorhanden.

Nachgereicht nicht zufriedenstellend.

Model Designüberlegung

Durchgeführt von Alexander Rieppel. Leider Falsch es ist nicht gefragt das Model aus Task07 zu beschreiben sonder das aktuelle. Das Model besteht auch aus weit mehr als nur den Messages. Bitte nachbessern.

Nachgereicht mit weiterhin bestehenden Mängeln.

Zeitaufzeichnung und Zeitschätzung

Es ist ein Zumutung ein so leichtes Arbeitspaket ungenügend abzuschliessen. Die Formatierung dieser Datei ist nicht meine Aufgabe und wäre für jeden einzelnen nicht weiter schwer gewesen. Weiters ist es auch keine Herausforderung auch in die Gesamttabelle bei seinem Namen die Gesamtzeit einzutragen.

Des Weiteren nicht erledigt von: Alexander Klune, Patrick Poelzbauer

Alexander Rieppel: Reihenfolge beachten, Summe

Nachgereicht: Immer noch Mängel. (Beim leichtesten Arbeitspaket)

Review 1

Management Client

Passt. Schaut vernünftig aus. Kleine Fehler noch zu beheben. Muss noch getestet werden.

Strukturierung Management Client.

Arbeitsaufteilung fair.

Problemstellungen ins Protokoll für nächste Woche.

TDD fertig

Eine AllTest Klasse.

AnalyticServerInterfaceTest passt soweit allerdings jetzt wirklich Test durchlaufen lassen und Erfolg/Misserfolg dokumentieren.

AnalyticServerTest:

- Coverage nicht ausreichend mit diesen Tests
- Fehler mit Exceptions
- Überall ein Assert
- Dokumentieren (Javadoc)

BillingServerSecureTest:

- Getter und Setter sind vorhanden bitte Fehler ausbessern
- Laufend die Entwicklung verfolgen um die Coverage zu optimieren
- Ansonsten sehr gut
- Testablauf im Protokoll dokumentieren

BillingServerTest:

- Sehr gut
- Testablauf im Protokoll dokumentieren

ManagementClientTest:

- Sehr gut
- Coverage verfolgen bitte
- Fehler rausbringen

- Testablauf im Protokoll dokumentieren
- Arbeitsaufteilung fair.

ModelTest:

- Wichtig für Coverage bitte nachreichen
- Kann man generieren lassen

Model

Erledigt (Hilfe durch Lins und Schuschnig)

BillingServer

Schaut vernünftig aus.

Arbeitsaufteilung: Klune mehr.

Testen aufrufen fehlt. Funktionsbeweise bitte.

BillingServerSecure

Schaut vernünftig aus.

Arbeitsaufteilung: Klune mehr.

Testen aufrufen fehlt. Funktionsbeweise bitte.

Javadoc

Zeilenkommentare

Testing Component

Arbeitsaufteilung: Pölzlauer mehr.

Exceptions

Hängt mit Arbeitspaket Exceptions Designüberlegung zusammen bitte mehrere da dieses Programm mehrere erfordert.

Zweiten Konstruktor bei dem man Message übergibt.

Testing Component

Schaut vernünftig aus.

Funktionsbeweis liefern / Bugfixing für nächste Wochen.

Alle Probleme dokumentieren Mails an Schuschnig und Lins

Analytic Server Implementierung

Gut hätte mehr sein können.

Fragen klären.

Review 3

Management Client Fertig

Durchgeführt von Alexander Auradnik alleine. (Interne Absprache)

Fertig. Bug fixing bei Eingaben!!!

Rieppel an Testfällen gearbeitet.

Testing Restabnahmen

Durchgeführt von Alexander Rieppel alleine diese Woche. (Interne Absprache)

Eine AllTest Klasse.

AnalyticServerInterfaceTest passt soweit allerdings jetzt wirklich Test durchlaufen lassen und Erfolg/Misserfolg dokumentieren.

AnalyticServerTest:

- Coverage nicht ausreichend mit diesen Tests
- Fehler mit Exceptions
- Dokumentieren (Javadoc)

BillingServerSecureTest:

- Coverage mit 62,5 zu wenig
- Fehler abklären mit Lins ob Code oder Test falsch

BillingServerTest:

- Coverage mit 35,2 zu wenig
- Fehler mit user.properties beheben!

ManagementClientTest:

- Funktioniert nicht

ModelTest:

- Nicht vorhanden

Arbeitsaufteilung fair.

Auradnik an Management Client gearbeitet.

Testing Component

Pölzlbauer zuständig nur für die Klasse: TestingComponent und Teile von TestingClient

Diese Funktionieren.

Klune zu spät angefangen Fehler in der DesingÜberlegung (Mo 17.2.14 angefangen)

Fehlt alles bei TestingClient, CreateThread, BidThread, ListThread

Exceptions nachgereicht

Es wurde nichts nachgereicht.

Connections

Durchgeführt bei Lins und Schuschnig funktioniert.

BillingServer

Immer noch Fehler.

Bitte am 18.2 nachreichen.

Evaluation Testing Component

Operating System: Mac OsX Version 10.8.5

CPU: 2.3 GHz Intel QuadCore i7

RAM: 8 GB 1600 MHz DDR3

PID	Prozesname	Benutzer	%CPU	Threads	Physika.	Speicher	Speicher
3626	java	tobi	101,9	126	4890,1 MB	Intel (64-Bit)	

CPU%:

Benutzer 92,16

System 7,84

clients = 1000

auctionsPerMin = 10

auctionDuration = 2000

updateIntervalSec: 20

bidsPerMin = 2000

After about 7 minutes the system crashes.

Programmbeschreibung

Das Programm kann entweder über die ant targets gestartet werden. Hierzu siehe das build.xml File.

Ansonsten kann das Programm so gestartet werden:

AnalyticServerStart.java

BillingServer.java

StartServer.java

ManagementClientStart.java

MainClient.java

Additional: TestingComponent.java

Die ersten zwei Targets können vertauscht werden.

Anschließend können alle Funktionen wie unter ManagementClient Manual beschrieben verwendet werden.

Management Client Manual

Der Management Client dient zur Verwaltung von Billing- und Analytics Server.

Ausführung

Argumente:

<AnalyticsServer Remote Adress> <BillingServer Remote Adress>

Beispiel:

Java -jar AnalyticsServer BillingServer

Bei verwendung des build.xml müssen Namen des Analytics und Billings Server in der Properties eingetragen werden.

Befehle

Befehle um mit den Billing Server zu aggieren:

- !login <username> <password>: Anmelden mit spezifischen Daten
- !steps: Auflistung aller Preisstufen
- ! <startPrice> <endPrice> <fixedPrice> <variablePricePercent>: hinzufügen einer neuen Preisstufe
- !removeStep <startPrice> <endPrice>: Entfernen einer Preisstufe
- !bill <userName>: Zeigt die Gebote des gewählten User. Die Anzeige begrenzt sich auf Auktionen die bereits abgeschlossen worden sind. Zusätzlich werden die Auktions Abgaben angegeben, welche mittels der Abgabeliste ermittelt wird.
- !logout: Setzt den Verbindungsstatus zurück. Um auf Befehle, wie ‚bill‘ zugreifen zu können, muss sich der Managment Client User wieder über das BillingServiceSecure remote Object mit den Befehl ‚login‘ Verbinden.

Befehle um mit den Analytics Server zu aggieren:

- !subscribe <filterRegex>: Den Management ein bestimmtes Event abonnieren lassen. Mehrfache Abonnierungen möglich.

Beispiel:

```
!subscribe '(USER_.*)|(BID_.*)'
Created subscription with ID 17 for events using filter '(USER_.*)|(BID_.*)'
```

- !unsubscribe <subscriptionID>: Eine Abonnierung eines Events mittls ID wieder abmelden.

Befehlseingaben werden geprüft. Bei unbekannten Befehlen oder falschen Syntax wird der Client über das UI mit vordefinierten Errormessages verständigt.

Analytics Server Event-Message Management

Der Management hat zwei Möglichkeiten Nachrichten, die durch aufgetretene Event entstehen, zu erhalten

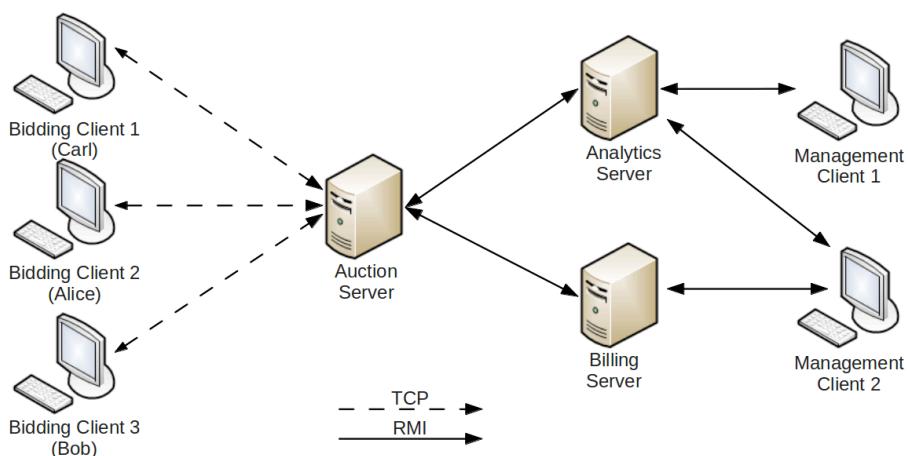
- *Automatic*
Sofortige Ausgabe der Nachrichten
- *on-demand*
Ausgabe nur bei Befehl

Dadurch ergeben sich folgende Befehle, um die Moduse zu wählen bzw. zu aggiren:

- !auto: Events werden automatisch, sofort ausgegeben. (none default)
- !hide: Events werden nicht ausgegeben und im Hintergrund gespeichert.
- !print: Gespeicherte Nachrichten (durch Befehl !hide), werden ausgegeben (Reihenfolge)

Architektur

Grafische Darstellung des Auction System:



Die Test Driven Developement Testcases wurden so geschrieben, dass alle wesentlichen Gesichtspunkte, die aus der Aufgabenstellung und aus den bisher geschriebenen Codeteilen hervorgehen, sinnvoll auf ihre jetzige und zukünftige Funktionalität getestet werden. Derzeit existieren Testcases zum AnalyticServer, BillingServe, Model und ManagementClient. Weiters existieren noch Testfälle aus der vorherigen Aufgabe Task07 welche gegebenenfalls noch erweitert werden müssen wenn dies erforderlich ist. In der folgenden Liste sind alle Tests einzeln, pro Package aufgelistet. Angegeben ist pro Test jeweils am Anfang der getesteten Methodenname und eventuelle Zusatzschritte die unternommen wurden in Spezialfällen. Am Ende befindet sich eine Liste mit Problemen die während dem Testen festgestellt wurden und ob sie noch ausständig sind oder bereits behoben.

Wichtig: Vor dem Starten der UnitTests ist es zwingend erforderlich die beiden Server manuell zu starten, da sonst ein Teil der Testfälle fehlschlägt und Coverage verloren geht.

Beim BillingServer im Package BillingServer werden folgende Methoden getestet:

- Login mit richtigen Usereingaben und Direktaufruf
- Login mit falschen Usereingaben
- createMD5

Beim BillingServerSecure im Package BillingServer werden folgende Methoden getestet:

- getPriceSteps
- createPriceStep mit akzeptierten Eingaben
- createPriceStep zweimal hintereinander und akzeptierten Eingaben
- createPriceStep zweimal hintereinander und überlappenden Eingaben
- createPriceStep zweimal hintereinander und gleichen Eingaben
- createPriceStep mit Startpreis größer Endpreis
- createPriceSep mit startPrice negativ
- createPriceSep mit EndPrice negativ
- createPriceSep mit FixedPrice negativ
- createPriceSep mit VariablePricePercent negativ
- deletePriceStep mit existierenden Eingaben
- deletePriceStep mit falschem Startpreis
- deletePriceStep mit nicht existierenden Eingaben
- billAuction mit akzeptierten Eingaben
- billAuction mit mehreren und gleichen Eingaben
- getBill

Overall Coverage von Package BillingServer 59.0% (Testzeitpunkt: 25.02.2014 18:06)

Beim Management Client im Package ManagementClient werden folgende Methoden getestet:

- auto
- hide
- processEvent mit autoprint true
- processEvent mit autoprint false
- Getter- und Setter -Methoden

Zusätzlich werden noch folgende Eingaben am ManagementClient getestet

- Keine Eingabe
- Falsche Eingabe
- !login mit richtigen Benutzereingaben
- !login zweimal hintereinander mit richtigen Benutzereingaben
- !login mit falscher Verwendung
- !login mit falschen Benutzeingaben
- !logout mit Login
- !logout ohne Login
- !steps mit Login
- !addstep mit Login und richtiger Verwendung
- !removestep mit Login und richtiger Verwendung
- !addstep ohne Login
- !addstep mit falschem Format
- !addstep mit falscher Anzahl von Argumenten
- !addstep mit falschen Parametern
- !bill ohne Login

- !bill ohne Usernamen
- !subscribe mit richtiger Verwendung
- !unsubscribe mit richtiger Verwendung
- !unsubscribe ohne ID
- !auto
- !hide
- !print
- !bill mit richtiger Verwendung
- Eingabe mit Leerzeichen am Anfang
- !steps ohne Login
- !subscribe ohne Login
- !subscribe mit falscher Anzahl von Argumenten
- !removestep mit falscherAnzahl an Argumenten
- !removestep ohne Login
- !removestep mit falschem Parameterformat
- !removestep mit nicht existierendem PriceStep

Overall Coverage von Package ManagementClient: 78.8% (Testzeitpunkt: 25.02.2014 18:12)

Beim AnalyticsServer werden folgende Methoden getestet:

- processEvent
- notify
- Getter und Setter Methoden

Zusätzlich wurde noch die korrekte Objekterstellung und ein Vergleich zweier Objekte überprüft. Zu beachten ist hier, dass die Methoden des AnalyticServer durch die ManagementClient Testfälle mitgetestet werden.

Beim EventHandler im Package AnalyticsServer werden folgende Methoden getestet:

- Execute mit Event AUCTION_STARTED
- Execute mit Event AUCTION_ENDED

- Execute mit Event BID_PLACED
- Execute mit Event BID_PLACED zweimal
- Execute mit Event USER_LOGIN
- Execute mit Event USER_LOGIN zweimal
- Execute mit Event USER_LOGOUT
- Execute mit Event USER_LOGOUT zweimal
- Execute mit Event USER_DISCONNECTED
- Execute mit Event BID_OVERBID
- Execute mit Event BID_OVERBID zweimal
- Execute mit Event BID_WON
- Execute mit Event BID_WON zweimal

Overall Coverage von Package AnalyticsServer 77.7% (Testzeitpunkt: 25.02.2014 18:15)

Beim Package Model werden alle Methoden erfolgreich getestet. Da dies nur getter- und Setter Methoden sind, wird diesen Testfällen nicht weiter Beachtung geschenkt. Eine Ausnahme im Package Model stellt die Klasse PriceSteps dar.

Bei PriceSteps im Package Model werden folgende Methoden getestet:

- Getter und Setter-Methoden
- ToString mit leerer Liste
- toString mit gefüllter Liste

Es existieren noch weitere Testfälle in dieser Testklasse, allerdings sind diese mit den Testfällen aus dem BillingServerSecure ident.

Overall Coverage von Package Model 95.7% (Testzeitpunkt: 25.02.2014 18:15)**Overall Coverage zum gesamten Projekt 73.6% (Testzeitpunkt: 25.02.2014 18:15)****Probleme die anhand der Testfälle festgestellt wurden**

1. Zwei PriceSteps hintereinander abspeichern nicht möglich (behoben von Lins)
2. Inkorrekte Verarbeitung überlappender PriceSteps (behoben von Lins)

3. Ausführung der Methoden billAuctions und getBill, ohne Fehler nicht möglich, Exceptions werden geworfen aufgrund Punkt 2 (behoben von Lins)
4. User.properties fehlt, deshalb keine ausreichende Testung des BillingServers (behoben von Lins)
5. Gleichzeitiger Start von Billing- und AnalyticServer auf einem Gerät nicht möglich, da beide auf den selben Port eine Registry erstellen wollen (behoben von Lins)
6. Erstellen von PriceSteps mit endPrice=0 für unendlichen EndPrice nicht möglich, da die Bedingung dafür fehlt (behoben von Klune)
7. Entsprechende Stelle für den BillingServer (wegen Punkt 5) auskommentiert, deshalb keine ausreichende Testung des ManagementClient möglich (behoben von Auradnik)
8. Methode für Unsubscribe funktioniert nicht ordnungsgemäß, liegt (laut Auradnik) am AnalyticServer (behoben von Schuschnig)
9. Zum gleichen User kann nur eine einzige Bill gleichzeitig gespeichert werden, sollen aber mehrere möglich sein (behoben von Lins)

Fazit

Insgesamt konnte eine Coverage von 73.6% mit 226 Testfällen erreicht werden. Dabei wurde jedes Package getestet.

Die wichtigsten Pakete in Überblick:

Package Model wurde fast vollständig getestet, da fast alle anderen Packages darauf aufbauen. Fehler in der Implementierung des Models können zu Konflikten und zusätzlicher Arbeitszeit führen. Die meisten Tests des Packages Models wurden vor den ersten Implementierungen durchgeführt.

Grundfunktionen und besondere Szenarien von Billing- und Analyticsserver wurden ebenfalls ausführlich vor dem Zusammenführen getestet. Es konnten mehrere Bugs behoben und Zeit in der weiteren Implementierung durch Fehlersuche gespart werden. Durch Designfehler konnten einzelne Bereiche (z.B. große Teile der Main-Methode der Packages Billing- und Analyticsserver nicht komplett getestet werden. Hierbei ist zu beachten, dass dies keine Relevanz auf Fehleranfälligkeit des vorhandenen Codes hat, da alle Funktionen der Server Packages ausführlich getestet werden konnten.

Tests des Package ManagementClient wurden nach einer Teilimplementierung bearbeitet und hinzugefügt. Es kam hier zu kleinen Änderungen in anderen Packages, was dazu führte, dass sich Teile des Management Clients geändert haben. Trotz einer Coverage von 78% konnten alle Grundfunktionen und besondere Szenarien getestet werden.

Package	Gesamte Coverage in Prozent
Model	95.7%
BillingServer	59.0%
ManagementClient	78.8%
AnalyticsServer	77.7%
...	
	73.6%

Weitere getestete Packages sind:

- Client
- Server
- Testing Component

Tests des Package Server und Client, welche bereits vorhanden waren, wurden überarbeitet und teilweise erweitert.

Package Testing Component wurde auf Gültigkeit der Testing Szenarien getestet. Ein höherer Coverage Anteil war schlussendlich nicht nötig. Alle Tests konnten erfolgreich implementiert werden.

Reviews

Review 29.1.14

- Mehr Zeit als erwartet für das UML Klassendiagramm benötigt (Schuschnig/Lins)
- Auch nach dreimaliger Aufforderung falsche Gitnamen vorhanden. (Patrick Poelzbauer)
- Formatierung der Zeitschätzungstabelle ist wünschenswert und nicht meine Aufgabe.
- Patrick Poelzbauer ich währe dir dankbar wenn du einen ordentliches Programm verwendest das beim öffnen der .xlsx Dateien nicht die Formatierung zerstört oder bitte dich diese nach dem öffnen nicht zu speichern ich bin nicht gewillt diese wieder zu formatieren.
- Keine Fortschritt bei TDD Testfällen diese sind nicht zureichend. (Alexander Rieppel)
- Model Designüberlegung falsch. (Alexander Rieppel)
- Keine Zeitschaetzung (Alexander Klune; Patrick Poelzbauer) (**Nachgereicht**)
- Falsche Zeitschaetzung (Alexander Rieppel) (**Nachgereicht**)
- Kein Use Case Diagramm (Alexander Klune; Alexander Rieppel) (**Nachgereicht**)
- Falsche Arbeitspakete Alexander Rieppel: TDD (ersten Testfälle); Recherche Testfälle Designüberlegung; Model Designüberlegung
- Fehlende Arbeitspakete Patrick Poelzbauer: Exceptions Designüberlegung

Negativ aufgefallen:

Alexander Rieppel, Alexander Klune, Patrick Poelzbauer

Positiv aufgefallen:

Tobias Schuschnig, Tobias Lins, Alexander Auradnik

Review 12.2

- Arbeitspakete Auradnik erfüllt aber noch Korrekturen Notwendig
- Arbeitspakete Rieppel erfüllt aber noch Dokumentation der Testabläufe im Protokoll notwendig
- Arbeitspakete Klune BillingServer auf einem guten Weg. TestingComponent mehr bei Pölzlbauer
- Arbeitspakete Pölzlbauer auf einem guten Weg Testing Komponent muss noch getestet werden; Exceptions verbessern

Positiv Aufgefallen:

Alexander Auradnik, Alexander Rieppel, Tobias Lins, Tobias Schuschnig

Neutral Aufgefallen:

Alexander Klune, Patrick Pölzlbauer

Negativ Aufgefallen:

Review 19.2

- Arbeitspakete Auradnik erfüllt. (Fehlend ausgiebiges Testing)
- Arbeitspakete Rieppel nicht erfüllt siehe Feedback
- Arbeitspakete Klune. TestingComponent nicht weitergearbeitet
- Arbeitspakete Pölzlbauer andere Arbeitspakete kein Fortschritt möglich da ManagementClient und die Server nicht fertig; Exceptions verbessern fehlt
- Arbeitspakete Lins: Fehler Bugfixing
- Arbeitspakete Schuschnig: Kleine Bugfixing arbeiten

Positiv Aufgefallen:

Alexander Auradnik, Tobias Lins, Tobias Schuschnig

Neutral Aufgefallen:

Patrick Poelzlbauer, Alexander Rieppel

Negativ Aufgefallen:

Alexander Klune

Review 26.2

- Arbeitspakete Auradnik: erfüllt.
- Arbeitspakete Rieppel: erfüllt
- Arbeitspakete Klune: nicht erfüllt durch Schuschnig
- Arbeitspakete Pölzlbauer: teilweise erfüllt Rest durch Schuschnig
- Arbeitspakete Lins: erfüllt
- Arbeitspakete Schuschnig: erfüllt

Positiv Aufgefallen:

Alexander Auradnik, Tobias Lins, Tobias Schuschnig

Neutral Aufgefallen:

Alexander Rieppel

Negativ Aufgefallen:

Alexander Klune, Patrick Poelzlbauer

Zeitaufzeichnung

Der Übersichtlichkeitshalber der angehängten Datei Zeitaufzeichnung.xlsx zu entnehmen