5AHIT

Distributed Auction System

Tobias Lins, Michaela Lipovits, Daniel Dimitrijevic, Nanak Tattyrek

12/11/2013

Tasks

Assignee	Issue	End Date	Status
DIMD	Socket Research (Client Prototypen) + TEST + GIT	11.12.2013	done
LIPM	DatagramSocket Research (Client Prototypen) + TEST + GIT	11.12.2013	done
TATN	Konsolenbefehle samt exception handling (überlegen auf was zu achten is) + TEST + GIT	11.12.2013	done
LINT	ServerSocket Research (Server Prototypen) + TEST + GIT	11.12.2013	done
TATN	Mock Objekte zum Testen anschauen + GIT	11.12.2013	~
DIMD	Usecase überarbeiten, File Prototyp (Obj. To file)	18.12.2013	nearly done
LIPM	Aktivitätsdiagram überarbeiten, Patterns (Factory, Command Pattern)	18.12.2013	done
TATN	Konsolenbefehle, Befehlauswertung	18.12.2013	done
LINT	UML + Patterns (Factory, Command Pattern)	18.12.2013	done

Planned Tasks

Assignee	Issue	Estimated Time	Actual Time
Tobi	ServerSocket implementierung + Testing	1 h 30 min	
Michi	Server - Anfragen in Threads bearbeiten (Threadpool, ThreadSafe) + Testing	4 h	
Nanak	Server. Auction Management implementieren + Testing	4 h	
Daniel	Server - Ressourcenfreundlichkeit (GarbageCollector, clean shutdown) + Testing	3 h	
Tobi	Server - Kontrolle der Befehle vom Client + Testing	2 h	
Tobi/Nanak	Server - asynchrone Benachrichtigungen + Testing	3 h	
Daniel	Client - Connection to Server + Testing	1 h	
Nanak	Client - Konsoleneingaben + senden an Server + Testing	1 h 30 min	
Michi	Client - DatagramSocket + Testing	1 h	
	Summe	18 h	

Time Planning

Assignee	Issue	Estimated Time	Actual Time
DIMD	Socket Research (Client Prototypen) + TEST + GIT	1h	30min
LIPM	DatagramSocket Research (Client Prototypen) + TEST + GIT	1h	30min
TATN	Konsolenbefehle samt exception handling (überlegen auf was zu achten is) + TEST + GIT	30min	0min
LINT	ServerSocket Research (Server Prototypen) + TEST + GIT	30min	40min
TATN	Mock Objekte zum Testen anschauen + GIT	30min	30min
	Week 2		
DIMD	Usecase überarbeiten, File Prototyp (Obj. To file)	3h	1h 30min
LIPM	Aktivitätsdiagram überarbeiten, Patterns (Factory, Command Pattern)	2h	1h 20min
TATN	Konsolenbefehle, Befehlauswertung	2h	2h 40min
LINT	UML + Patterns (Factory, Command Pattern)	2h	2h

Planning

Client

- mehrere Clients sind möglich
- `client(host, tcpport, udpport)`
    - `falls was fehlt/falsch message & exit`
- Eingabe über konsole (system.in)
- Senden über enter
- Soll Befehle vom Server schoen darstellen
    - e.g. alice> The auction 'Super small notebook' has ended. dave won with 250.00.
- via Socket mit Server verbinden, Verbindung bis Server oder Client beendet werden
- Antworten vom Server werden von eigenem Thread verarbeitet
- DatagramSocket(udpport) beim login dem Server senden (eigener Thread für DatagramSocket)

Client Befehle

Client kann anonym gestartet werden

- nur `!list`

man kann sich einloggen

- `!login <username>`
    - `Successfully logged in as <username>!`
- `!list`
    - Wenn noch kein Gebot “0.00” und hoechstbietender “none”
- `!bid <auction-id> <amount>`
    - `You successfully bid with <amount> on <description>.`
- `!create <duration> <description>`
    - `!create 25200 Super small notebook`
    - duration in sekunden
- `!end`
    - beendet Client
    - bei Conn zu Server erst Threads beenden
    - Ressourcen freigeben

User kann sich ausloggen

- `!logout`
    - `Successfully logged out as <username>!`

Ausgaben sollen ca so wie in Angabe aussehen, kleine Unterschiede sind erlaubt

Server

- 1 Server
- Ueberpruefen der Kommandos
- Spricht Clients per Username an, nicht IP (socket.getInetAddress())
- Verweigert Verbindung wenn dieser User schon eingeloggt ist
- jede Auktion kriegt eine ID zugewiesen vom Server
- Konstruktor Server:
    - `server(tcpPort)`
    - wenn falsch/fehlend → Message & exit
- ServerSocket
    - `accept()`
    - `getInputStream()`
    - `getOutputStream()`
    - schließen von Conn wenn nicht mehr nötig
- pro Anfrage ein Thread
- Threadpool (`java.util.concurrent.ExecutorService`)
- Threads sollen wenig Ressourcen verbrauchen und sauber beendet werden
- Auction Management
    - bieten auf abgelaufene Auktionen verbieten
    - abgelaufene Auktion aus Liste löschen
    - java.util.Timer in combination with a java.util.TimerTask
- Synchronization, ThreadSafe
- Garbage Collector zum Freigeben von Ressourcen
- BufferedStreams
    - e.g. java.io.BufferedReader
    - streams schließen
    - Ressourcen sparen
- Server Enter → Shutdown (alle conn schließen, alle Ressourcen freigeben!!)

Benachrichtigungen

- asynchron (UDP - DatagramSocket)
    - nur wenn User online
- wenn sie ueberboten wurden(der mit dem highestbid zuvor)
- wenn auktion endet

    - owner benachrichtigen
    - highestbidder benachrichtigen
- `!new-bid <description>`
    - vom auktionsserver an client
- `!auction-ended <winner> <amount> <description>`
    - vom auktionsserver an client

Befehle

`auction(id,description,owner,enddate,highestbid,highestbidder)`

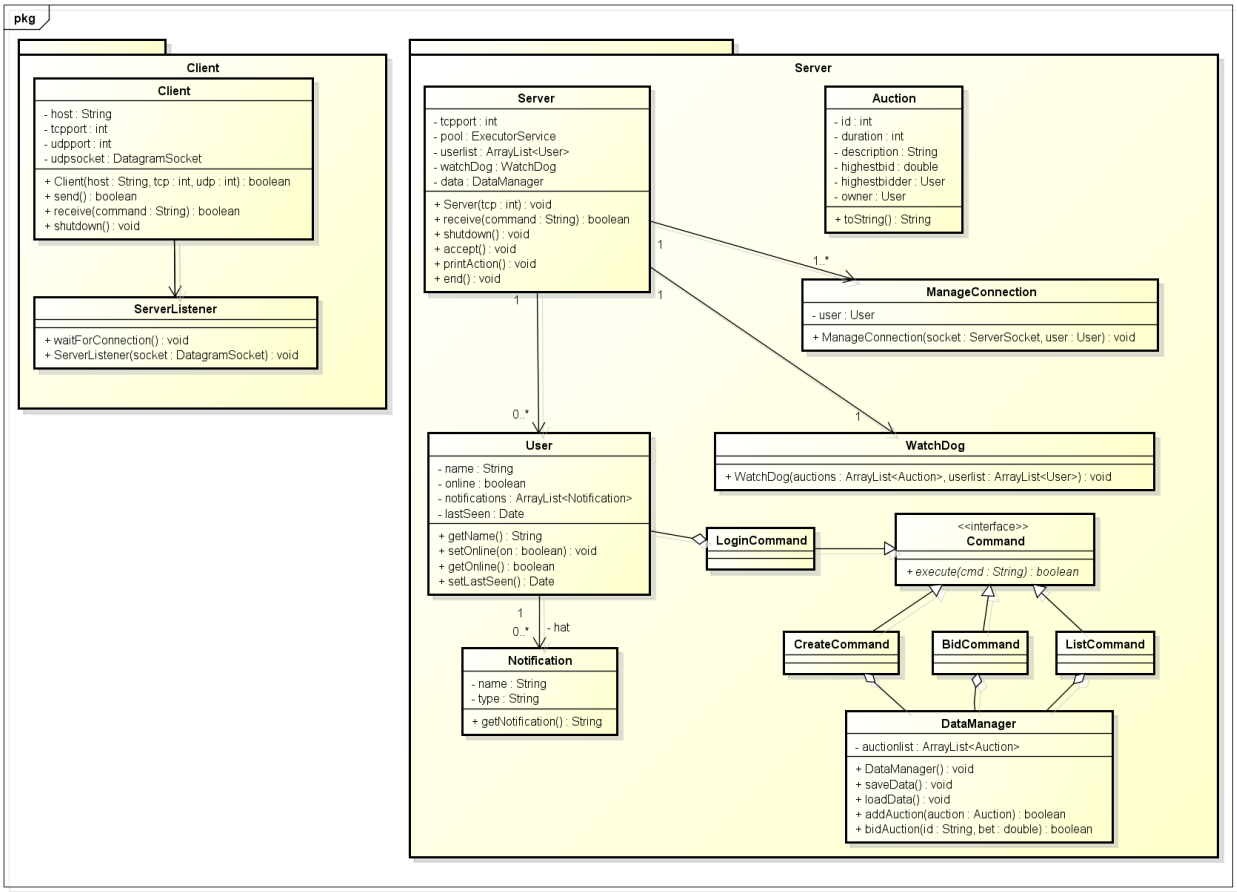
```
!create 25200 Super small notebook
An auction ‘Super small notebook’ with id 3 has been created
an will end on 04.10.2012 18:00 CET.
```

```
!bid 3 200
You successfully bid with 200.00 on ‘Super samall notebook’.
```

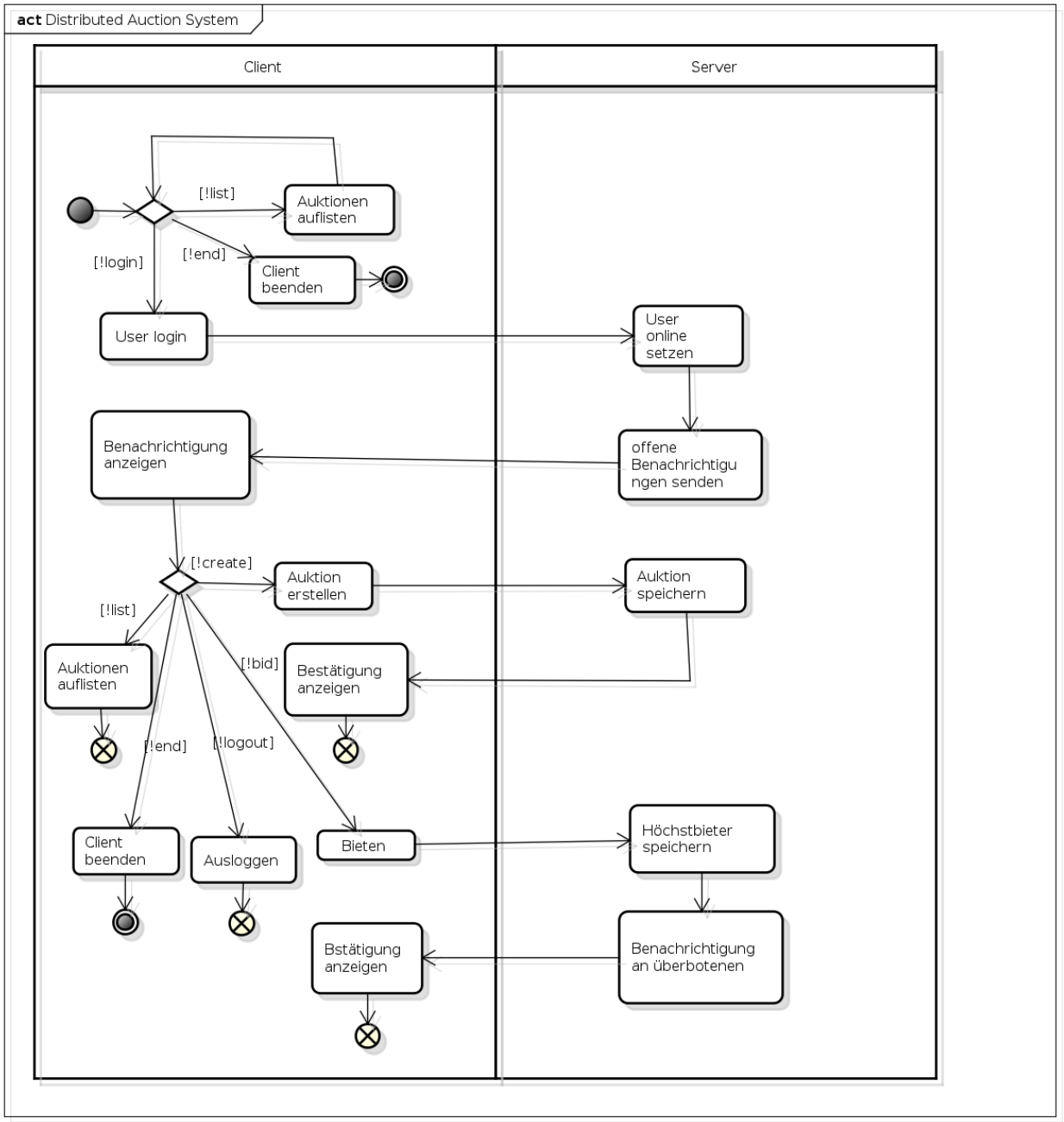
wichtige sachen:

- Exceptoion/errormessages niiiie vergessen
- Userdaten können in einer db sein, können beim sauberen beenden einfach in files geschreieben werden
- Thread muss sauber beendet werden!!
- Change Request per mail rechtzeitig anfragen, designkonzept darf nur bei annahme des change requests geänder werden
- Concurrend Collections

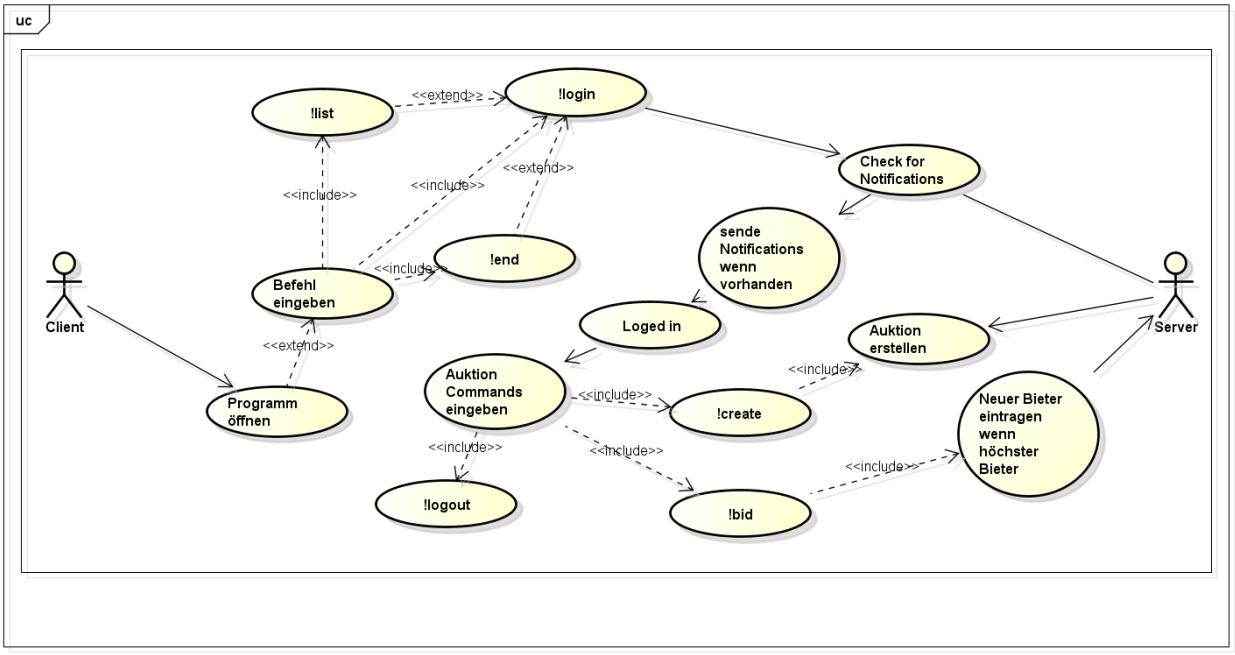
UML

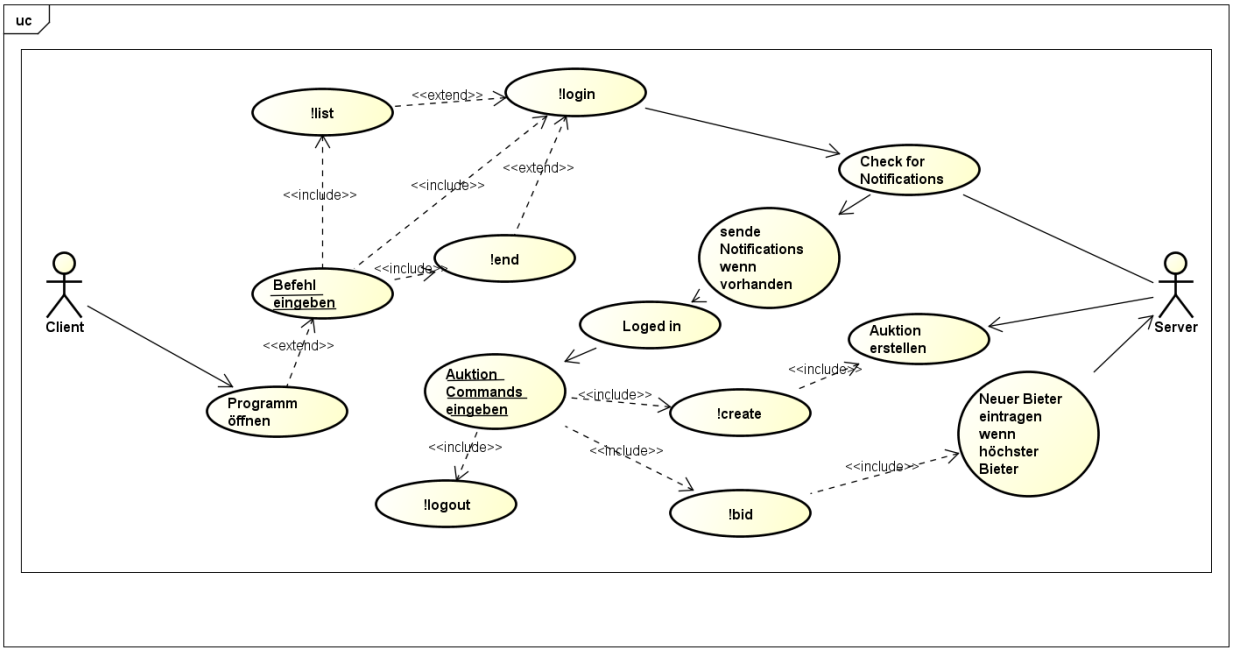

pkg
Client
Client
- host : String
- tcpport : int
- udpport : int
- udpsocket : DatagramSocket
+ Client(host : String, tcp : int, udp : int) : boolean
+ send() : boolean
+ receive(command : String) : boolean
+ shutdown() : void
ServerListener
+ waitForConnection() : void
+ ServerListener(socket : DatagramSocket) : void
Server
Server
- tcpport : int
- pool : ExecutorService
- userlist : ArrayList<User>
- watchDog : WatchDog
- data : DataManager
+ Server(tcp : int) : void
+ receive(command : String) : boolean
+ shutdown() : void
+ accept() : void
+ printAction() : void
+ end() : void
Auction
- id : int
- duration : int
- description : String
- highestbid : double
- highestbidder : User
- owner : User
+ toString() : String
ManageConnection
- user : User
+ ManageConnection(socket : ServerSocket, user : User) : void
User
- name : String
- online : boolean
- notifications : ArrayList<Notification>
- lastSeen : Date
+ getName() : String
+ setOnline(on : boolean) : void
+ getOnline() : boolean
+ setLastSeen() : Date
WatchDog
+ WatchDog(auctions : ArrayList<Auction>, userlist : ArrayList<User>) : void
LoginCommand
<<interface>>
Command
+ execute(cmd : String) : boolean
CreateCommand
BidCommand
ListCommand
Notification
- name : String
- type : String
+ getNotification() : String
DataManager
- auctionlist : ArrayList<Auction>
+ DataManager() : void
+ saveData() : void
+ loadData() : void
+ addAuction(auction : Auction) : boolean
+ bidAuction(id : String, bet : double) : boolean
1
1..*
1
1
0..*
1
1
0..*
- hat

Activity Diagram



Usecase


uc
!list
<<extend>>
!login
Check for Notifications
<<include>>
<<include>>
<<extend>>
sende Notifications wenn vorhanden
!end
<<include>>
Befehl eingeben
Client
Server
Loged in
<<extend>>
Auktion erstellen
<<include>>
Auktion Commands eingeben
<<include>>
!create
Neuer Bieter eintragen wenn höchster Bieter
Programm öffnen
<<include>>
<<include>>
!logout
!bid
<<include>>


uc
!list
<<extend>>
!login
Check for Notifications
<<include>>
<<include>>
<<extend>>
sende Notifications wenn vorhanden
!end
<<include>>
Befehl eingeben
Client
Server
Loged in
<<extend>>
Auktion erstellen
<<include>>
Auktion Commands eingeben
<<include>>
!create
Neuer Bieter eintragen wenn höchster Bieter
Programm öffnen
<<include>>
<<include>>
!logout
!bid
<<include>>