CS 51024 – Stock Trading Simulation Project

Project Description

Proposal: a trading simulation game, using a live feed, database storage and connectivity, and distributed objects. We will have a trading engine which maintains an order book, matches trades and serves market data. Users can buy or sell shares of stock from other participants via a simple trading screen.

We will implement the system to run a market for a single stock. The market prices are fully defined by the orders in the system, so this market won't have to know anything about a particular stock in order to operate.

The objects in the system--

MarketServer: it manages the order book, generates market data and fill information.

Database: it maintains a list of executed transactions (not orders). Authorized users can be set up in the database ahead of time – you don't need to implement tools to administer the database.

Client: thin clients that provide a user interface to enter orders and display market data in real time. There can be optional automated trading algorithms that can be switched on/off which listen to market data and generate orders and receive trades.

The specification for this project is remarkably similar to the Penn-Lehman Automated Trading Project, which you should check out as background for this project. This project differs from PLAT in a few important ways:

1. We will not try to integrate our prices/orders into a real market like PLAT did with Island ECN.
2. We will only implement Limit Orders – we will not implement Market Orders. Here is a good description of various order types.
3. Communication between users and the market will be done with a tag-value based protocol. This will look like a stripped down version of the FIX protocol, which is a securities industry standard protocol for order management. Here is a pretty good overview of the FIX protocol.

Highlights of the PLAT site:

- **Nice overview of the whole project (includes a description of how the order book matches trades – very important)**
- **More detailed discussion of the project – there is a lot of info in the back about performance of the traders which you can ignore.**

If you are curious about how the equity market works and want to check out a real market simulator, there are a few pretty good sites – many offer free access so you can try them out:

- **StockQuest – free, targeted to students and educators. Good educational info in the site as well.**
- **The Stock Market Game – another site targeted to students/educators.**
- **VirtualStockExchange – geared more towards people who know a bit about the market and trading. More sophisticated system and tools, affiliated with CBS MarketWatch.**
- **Hollywood Stock Exchange – yes, you can trade anything. Pretty slick implementation of a market.**

Ongoing Discussion

We are setting up a mail list for the project. This is a complex system to describe, and implementation is even more complex. I'll work to keep it as simple as possible. We will have a couple of whiteboard sessions on this (time/date tba) and keep a mail list up for day-to-day stuff. The mail list is `trading-51024@cs.uchicago.edu`. I encourage collaboration on the understanding of the problem to be solved by using the mail list to ask questions and provide solutions.

How we will assess your implementation:

1. Live demo of the story set.
2. Slugfest – set two trading agents against each other (user1 enters an order, the user2 immediately hits the outstanding order). See how many orders the system can process per second, the average response time and the standard deviation of response time.

3. Bonus Round: Standards test – point your trading agents at someone else's implementation of the MarketServer. If we got the protocol spec right, and you implemented it right, it should work.
4. Extra Credit – note that none of the story sets include the use of the `cancel` message. Implementation of `cancel`, on both the server and client side is extra credit.

Design Notes

- Clients must wait for market to acknowledge their order before subsequent action (cancel, another order,…)
- If the engine crashes, it comes back up clean – persisting outstanding orders is not necessary
- It is possible to enter an order that trades through the book – meaning that it should execute through more than the best price level. To simplify things, we will not allow this situation: simply NACK this order with reason="trade through not allowed".
- Trading with yourself is called a 'wash' – this type of trading has gotten justifiably bad press recently. You should check for this condition and NACK any order that would cause this to happen.

Protocol Specification

General Message Sequence as initiated by a Client sending an order:

1. **Limit order sent to system by Client**
2. **ACK/NACK sent to originator of order by MarketServer**

   If the MarketServer ACKs the message, the Client is assured the order is now in the order book and can be executed at any time. A NACK means the MarketServer did not accept the order and it is not in the book and will never be executed unless the Client successfully resubmits it (and gets an ACK)

   An ACK will return the ID that the MarketServer uses to identify the order (MarketServer does NOT use the ClientID, as it is not guaranteed to be unique). The MarketServer ID must be used to reference the order if you send a Cancel message.

3. **MarketServer processes order and check for match(es)**

   *The MarketServer goes through a 'match process' where the new inbound order is checked against the current working order book for potential matches. If such a match exists, then this order is termed the aggressor. There is a good discussion on **FIFO Matching** in this informational document.*

   if (match occurs)

   a) **send FILL messages to owners of crossed orders**

      This message informs the buyers/sellers of their committed transaction. It gives them the price and quantity they executed. If the filled quantity is equal to their original order quantity, then they no longer have an order in the system. If the filled quantity is less than the original order quantity, the original order is left on the book with the original price and the quantity set to the remainder: (originalQuantity – filledQuantity).

      Note also that FILL messages can go to more than 2 Clients (client1 and client2) – for instance there might be two clients both with orders in the system to buy 10 shares at a price of $100. If an order is received by the MarketServer from client3 to sell 20 shares at $100, then 3 FILL messages are sent:

MktServer->Client1 buys 10 at $100
MktServer->Client2 buys 10 at $100
MktServer->Client3 sells 20 at $100

The MarketServer needs to archive all matches to a trade history database

b) send LAST message to all market data listeners
The LAST message informs all market participants that a trade has just occurred. We send the price and quantity, but do not reveal who were the parties to the trade.

The MarketServer needs to archive all LAST messages to a time-and-sales database.

c) **send BOOK message to all market data listeners**
If a match occurs then the book has somehow changed – send out the top 3 prices and aggregate quantities at each price level to all market participants. See section on ***Market Data*** *for more in-depth discussion of this topic.*

else if (top 3 of book changed in price or qty but no match)

d) **send BOOK message to all market data listeners**
If a match occurs then the book has somehow changed – send out the top 3 prices and aggregate quantities at each price level to all market participants.

else

e) **no external messages**
If the top 3 prices/aggregate quantities in the book have not changed as a result of the order just processed, then there is no need to push market data out to the market participants.

General Message Sequence as initiated by a Client canceling an order:

4. **Cancel message sent to MarketServer by Client**

   Client will initiate a cancel with the mktID provided by the MarketServer when it ACKed the original limit order. MarketServer will ensure that this connection was the originator of the order and cancel it from the book

5. **ACK/NACK sent to originator of Cancel by MarketServer**

   *If the MarketServer ACKs the message, the Client is assured the order is now cancelled out of the order book and cannot be executed. A NACK means the MarketServer did not accept the cancel and it the original order may still be in the order book (see `reason` for explanation – there are some error conditions where the order will still be on the books, others where it will not [order not found, for instance]) **.** If the book structure changes as a result of this cancel, then a new market data (BOOK) message will be sent.*

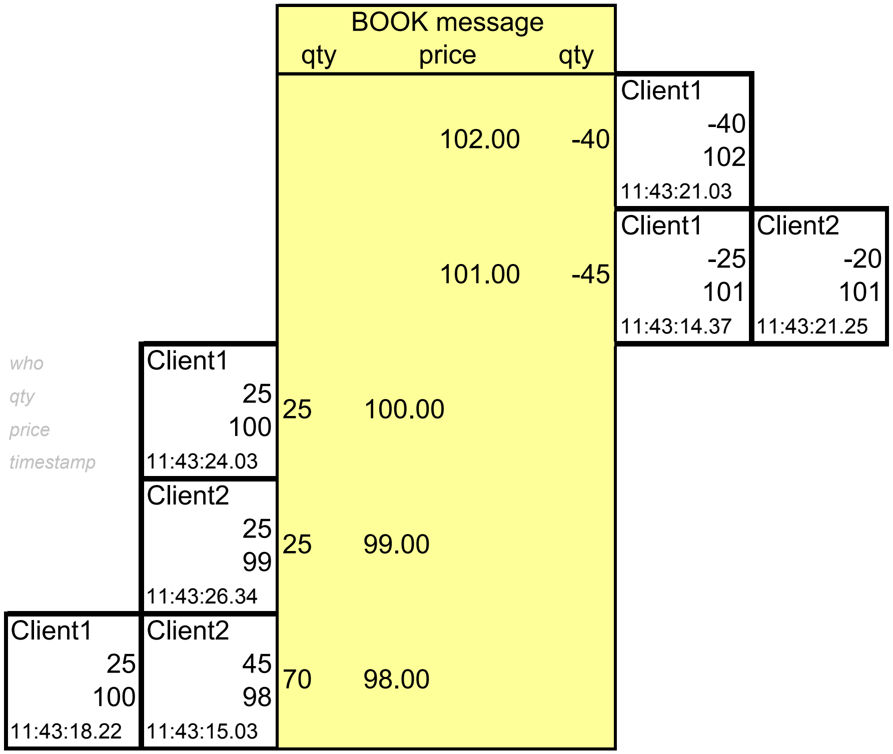
Market Data Message

Let’s say we have the order book depicted in the diagram to the right – In order to generate our market data message we must determine the aggregate quantity at each price level – which is simply the sum of quantities for orders at a give price level. Let’s consider what will go into the BOOK message, represented as the yellow portion of the diagram.

In this example we display the book with bids on the left (positive quantities, the ‘best’ bid has the highest price) and offers on the right (negative quantities, the ‘best’ offer has the lowest price). We have three unique bids based on price and two unique offers. Our BOOK message will therefore have 5 price/quantity pairs representing the 5 price levels.

If a particular price has zero quantity, it is not reported and does not count as a price level.

Note that in the BOOK message we do NOT report who has the orders in the system – the identities are kept secret.



			BOOK message			
		qty	price	qty		
			102.00	-40	Client1 -40 102 11:43:21.03	
			101.00	-45	Client1 -25 101 11:43:14.37	Client2 -20 101 11:43:21.25
who qty price timestamp	Client1 25 100 11:43:24.03	25	100.00			
	Client2 25 99 11:43:26.34	25	99.00			
Client1 25 100 11:43:18.22	Client2 45 98 11:43:15.03	70	98.00			

Let’s say we just received the 99 bid for 25 shares: our BOOK message will look like this:

```
BOOK mktTime 11:43:26.34 qty 25 price 100 qty 25 price 99 qty 70 price 98 qty -45 price 101 qty -40 price 102
```

CS 51024 – Stock Trading Simulation Project

Sample Message   tag/value pairs

```
>hello  clientID       bob0001 clientName     bob     clientTime     11:23:45.81
<ACK    clientID       bob0001 mktTime 11:23:45.92
<NACK   clientID       bob0001 mktTime 11:23:45.92    reason  text
```

this sequence is the initiation from the Client and either the ACK or NACK from MarketServer (it will send either ACK or NACK, never both) We can get a good idea of clock differences between Client and MarketServer from this sequence.

```
>limit  clientID       bob0002 qty     100     price   60.51   clientTime     11:23:47.02
<ACK    clientID       bob0002 mktID   mkt1000 mktTime 11:23:47.09
<NACK   clientID       bob0002 mktTime 11:23:47.09    reason  text
```

this sequence is an order being sent from the Client and either the ACK or NACK from MarketServer. We need to save the mktID as this is the reference we need to send to MarketServer if we want to cancel the order later. Note we return MarketServer timestamps here so we can check latencies between Client and MarketServer if we are so inclined. Remember that we cannot send another order until we have received an ACK from the last message.

```
<FILL   mktID   mkt1000 mktTime 11:23:47.11    qty     100     price   60.51
```

Here is a fill message - MarketServer gives me the ID of the original order (MarketServer internal ID, not the ClientID) - plus the quantity and price of the fill.

```
<LAST   mktTime 11:23:47.11    qty     100     price   60.51   totalQty       50000   totalMsgs      467     totalTx 17
```

The LAST message goes out to all market participants. Included at the end of the message are some overall market statistics that are updated with every trade:

```
○   totalQty    total number of shares traded during this session
○   totalMsgs   total number of messages from clients (hello, limit,cancel)
○   totalTx     total number of FILL messages sent from MarketServer
○
```

```
>cancel mktID   mkt1000 clientTime     11:23:47.87
<ACK    mktID   mkt1000 mktTime 11:23:47.99
<NACK   mktID   mkt1000 mktTime 11:23:47.99    reason  text
```

Cancel sequence - note again that we send the MarketServer ID to access the trade. The MarketServer will check to be sure that the ID you are canceling is actually your order.

```
<BOOK   mktTime 11:23:47.11    qty     100     price   60.51   [up to 5 price/qty pairs per side]
```

BOOK message give you the current timestamp of MarketServer, and up to 6 price/quantity pairs which represent the top 3 aggregate bids and top 3 aggregate offers. (interesting note, at the CME, we show the top 5 bids and offers)

Rules for Client/Server interaction, message traffic etc.

1. **Client Session begins with hello**
2. **All client side orders are guaranteed to be responded to with ACK/NACK. No response means market is down.**
3. **FILL/LAST/BOOK are sent to clients on unsolicited basis.**
4. **MarketServer processes one message at a time, performs all responses to appropriate Clients, then begins processing the next inbound client message.**

CS 51024 – Stock Trading Simulation Project

Story Set

Story #1

			BOOK message	
		qty	price	qty
who	Client1			
qty	25	25	100.00	
price	100			
timestamp	11:43:24.03			

Client1 initiates order to buy 25 shares at a price of 100

		Market Server		
	qty	price	qty	
Client1				Client1
25	25	100.00	-25	-25
100				100
11:43:24.03				11:43:14.37

Client 2, after receiving the BOOK message which describes the bid, sends in an offer for 25 shares at a price of 100. This order will immediately **cross** with the bid and a transaction will occurr. FILL messages are sent to Client1, Client2, LAST messages are sent to everyone showing 25 shares traded at a price of 100

	BOOK message	
qty	price	qty

Since the quantities of 25 matched, we have no remaining quantity and therefore both orders are now removed from the book. The subsequent BOOK message reflects that there are no outstanding orders.

CS 51024 – Stock Trading Simulation Project

Story 2

		BOOK message		
		qty	price	qty
who qty price timestamp	Client1 25 100 11:43:24.03	25	100.00	
	Client2 25 99 11:43:26.34	25	99.00	

We begin with 2 orders on the book as depicted

	Market Server			
	qty	price	qty	
Client1 25 100 11:43:24.03	25	100.00	-35	Client3 -35 100 11:43:27.03
Client2 25 99 11:43:26.34	25	99.00		

Client3 then sends an order to sell 35@100. This will cross with the bid for 25@100. FILL messages go to Client1 and Client3, LAST messages go to all.

	BOOK message			
	qty	price	qty	
	0	0.00	-10	Client3 -10 100 11:43:27.03
Client2 25 99 11:43:26.34	25	99.00		

BOOK message reflects the remaining quantity from the original order (-35@100) on the offer side, there is no bid at 100; the bid for 25 shares at 99 remains in the book and is reflected in the BOOK message.

Story 3

		BOOK message		
		qty	price	qty
who	Client1	25	100.00	0
qty	25			
price	100			
timestamp	11:43:24.03			

Client1 initiates order to buy 25 shares at a price of 100

		BOOK message		
		qty	price	qty
Client2	Client1	50	100.00	0
25	25			
100	100			
11:43:24.05	11:43:24.03			

Client2 sends an order to buy 25 shares at a price of 100. Note this order arrives after the order from Client1, so it is 2nd in line to be filled. The BOOK message reflects a total of 50 shares bid at a price of 100

		Market Server			
		qty	price	qty	
Client2	Client1	50	100.00	-35	Client3
25	25				-35
100	100				101
11:43:24.05	11:43:24.03				11:43:25.37

Client3 then sends in an offer to sell 35 shares at 100. This will cross with all of Client1's order and 10 shares of Client2's order. Client1 gets a FILL message for 25@100 (and knows they have no remaining quantity in the book), Client2 gets a FILL message for 10@100, so they know they still have a bid in for 15@100. LAST message sent to all with 35@100

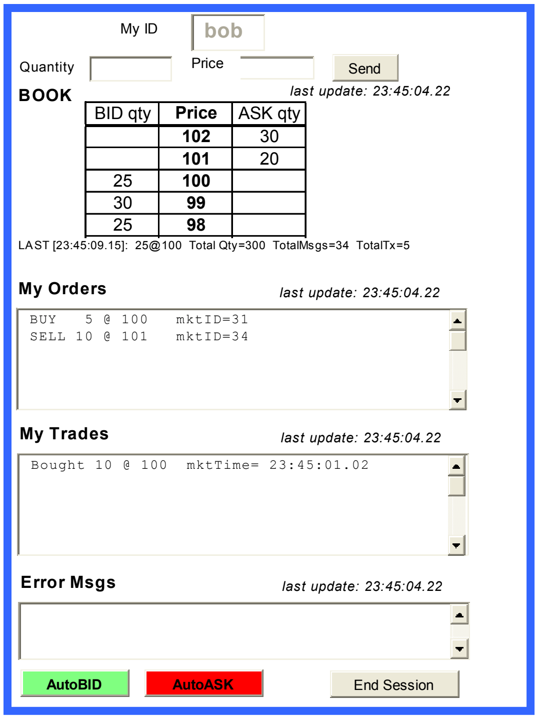
	BOOK message		
	qty	price	qty
Client2	15	100.00	0
15			
100			
11:43:24.05			

The subsequent BOOK message reflects the remaining bid for 15 shares at a price of 100.

Example Trading Screen

Note: this is an example layout and functionality – there are many other ways to do this, it's up to you!

Our screen is started with command line args to point to the MarketServer, and to set our identity for the session.



My ID bob

Quantity

Price

BOOK

last update: 23:45:04.22

BID qty	**Price**	ASK qty
	102	30
	101	20
25	**100**	
30	**99**	
25	**98**	

LAST [23:45:09.15]: 25@100 Total Qty=300 TotalMsgs=34 TotalTx=5

My Orders

last update: 23:45:04.22

```
BUY   5 @ 100   mktID=31
SELL 10 @ 101   mktID=34
```

My Trades

last update: 23:45:04.22

```
Bought 10 @ 100  mktTime= 23:45:01.02
```

Error Msgs

last update: 23:45:04.22

The screen implements a simple order entry facility (quantity,price, SEND)

The **BOOK** section handles BOOK messages – remember these arrive unsolicited in real time.

The **Orders** section keeps track of all outstanding orders for this Client. If orders are filled, they come off the screen. If they are partially filled, their outstanding quantities are updated.

The **Trades** section simply logs trades which have been reported from MarketServer.

The **Error Msgs** section logs the original message and the NACK that was received from MarketServer.

AutoBID – starts the automatic sending of orders to MarketServer. Remember that we expect this bid to be hit, so you should probably wait until you see the correct LAST before sending another bid!

AutoASK – watches the market until it sees a prescribed bid (you can hardcode whatever price/qty you want here). We then send offers in to hit bids.

End Session – kills the client cleanly.