

Online Trading System Project Specifications

Distributed Objects CSPP 51024

Overview

An online trading system refers to an electronic marketplace where clients can connect and post orders to buy and sell specified products. For the purposes of this specification, the trading system is required to support a single product to exchange. The trading system deploys a gateway for clients to connect to the trading server and send and receive messages. The trading server, or trading engine, is responsible for processing new orders received, placing them in an order book and executing a matching function on the order book. These functions occur whenever the trading server receives a new order. The trading system accepts orders from clients, sending acknowledgements, ACK's, or negative acknowledgements NACK's, in response to communicate receipt of the order or rejection of the order to a given client. If the match function results in actual matches of buyers and sellers, the trading engine will communicate fill messages back to the parties involved in the transaction. Normally this would be a secure communication between the server and the individual clients involved in the transaction, but this specification only requires that the fill message contain unique identifiers for the buyer and seller (and that the buyer and seller client programs process only the fill information that is related to their unique identity). Whenever information passes into the trading engine that results in a material change in the state of the order book, the trading engine will publish market data messages. These messages are anonymous and represent aggregated data. The last message will include the price, quantity and time of a fill or trade, the book message will contain up to the top three buy and sell price levels (after all matching has resolved). Trading systems have extensive audit functionality, for this specification the only required audit capability is two administrative web pages that show 1) all messages sent and received organized by type and 2) all trades (fills) executed during the session and some aggregated measures.

The trading client is intended to be either a graphical user interface (GUI) which may take the form of a web page, java swing application, console application (run in a shell) or an applet or an automated trading system (ATS) that will consist of a console application that prints status to the screen. The client will have some audit requirements, a web page that can show the information related to messages the client has sent and received and information about fills (trades) that the client has executed in the market.

This specification also contains a series of optional functions. These are not required, but can enhance its functionality and add value to the final evaluation of the delivered product. They are diverse in nature offering opportunities to add to the project in areas of strength if so desired. Optional requirements are described at the end of each section, and of course, the developer may add their own optional features with documentation to improve performance or enhance functionality.

Client Gateway

The client gateway is responsible for managing connections with clients. Clients send messages to the gateway where they are evaluated for proper syntax and passed along to the trading engine. The gateway has the primary function of verifying messages are of the proper format, but the gateway may also determine if the contents of the message are legal and may generate the ACK's or NACK's related to the results of the process. The other option a developer may make is to allow the gateway to simply check format and transmit NACK's to clients when they send malformed messages while leaving content analysis and generation of ACK's and content-related NACK's to the trading engine.

An optional function that the gateway may have is to verify client connections using a PING/PONG message scheme. The client would send a PING message that contains content the gateway would use to determine if the client identified is already connected and if the client has the proper authorization (password) to verify the connection authenticity.

Example:

- a. A client sends in a limit order to buy 10 at 100. The client time is missing:

```
LIMIT clientID 1 clSeqID 1 quantity 10 price 100
```

The gateway generates a NACK:

```
NACK clientID 1 clSeqID 1 quantity 10 price 100 clTime mkTime 1103760000000 reason no client time
```

Trading Engine

A series of functions are executed with each new order that is entered into the market. The orders are sorted into product order book made up of a buy order book and a sell order book. The sorted book then goes through a matching process that uses a first-in-first-out (FIFO) matching algorithm to match orders. Finally any matches are reported out both as fills and changes in the order book, are reported out as anonymous market data.

The trading engine maintains an order book that sorts the client orders by buys and sells, sorting the buys first by descending price, then by the time the order was entered into the market (the market timestamp). The sells are sorted first by ascending price, then by the time the order was entered into the market.

The match process works as follows:

1. Take the highest bid and lowest offer and compare,
 - a. If bid < ask, do nothing and report any changes in the book through market data.
 - b. If bid = ask, a match occurs in the LOWER of the bid and ask quantity, each order at the matching price level is matched individually, if a

quantity of 10 sells is matched against 10, quantity of 1 buys, then there are 10 trades generated (10 fill messages, 10 market data last messages).

- c. If bid > ask, then find the market time stamp of the highest bid price and lowest ask price, if the ask has the later time stamp, then match at the bid price, otherwise match at the ask price until the highest bid < lowest ask.

The match uses a FIFO algorithm, this means that the order book is sorted in time priority and then the top bid is matched against lower or equal ask prices until the quantity of the bid is exhausted, moving then to the next bid order in the order book priority queue. Likewise, the top ask is matched against bids that have a greater or equal price until its quantity is exhausted, and so forth.

Each order match is reported in a fill message that is published through the same channels as market data. Clients are expected to use their unique ID as set by the server to determine whether they are party to the transaction.

An optional function would be to publish a CLOSE message when the trading engine is halted (closed). This message would be published through the same market data function as fills. It would be expected that any client could use this message to gracefully cease.

Examples

- a. A client places an order to buy 10 at 100

```
LIMIT clientID 1 clSeqID 1 quantity 10 price 100 clTime 1103760000000
```

- b. Results in an ACK from the server

```
ACK clientID 1 clSeqID 1 mktSeqID 11 clTime 1103760000000 mktTime 1103760000050
```

c. The order to buy 10 at 100 goes into the book and a match process kicks off:

Book

Bids	Price	Ask
	102	11
	101	100
	100	7
6	99	
55	98	

Book after order to buy 10 at 100

Bids	Price	Ask
	102	11
	101	100
10	100	7
6	99	
55	98	

Assume there are two orders to sell at 100 one for 5 and one for 2

Matching occurs

The sell order of 2 was placed first

Match the 2 first

Bids	Price	Ask
	102	11
	101	100
8	100	5
6	99	
55	98	

Matching occurs, part 2

Now the sell order for 5 is matched

Bids	Price	Ask
	102	11
	101	100
3	100	
6	99	
55	98	

MATCH 2 at 100

Send FILL message

Print market data for trade

MATCH 5 at 100

Send FILL message

Print market data for trade

d. Assume clientID 2 has the order to sell 2, mrkSeqID 4 and clientID 3 has the order to sell 5, mrkSeqID 6, the FILL messages look like:

FILL buyerID 1 sellerID 2 quantity 2 price 100 buyerMrkSeqID 11 sellerMrkSeqID 4 mrkTime 1103760000100

FILL buyerID 1 sellerID 3 quantity 5 price 100 buyerMrkSeqID 11 sellerMrkSeqID 6 mrkTime 1103760000105

e. A book market data message is published after the match process is completed. (Message is described in following section).

Market Data

Market data is the anonymous messages published for general consumption. There are two required messages, the last message and the book message. The last message is published whenever there is an order match. The last message also has the total volume traded and the total number of trades. The book message is published whenever a new order is processed and a match function occurs. The book message consists of a repetitive optional structure, that requires only the first bid with aggregate quantity and first ask with aggregate quantity to be passed, but can allow up to the top 5 bids with corresponding aggregate quantity and top 5 asks with corresponding aggregate quantity. Aggregate quantity refers to the total quantity associated with bids or asks at a given price level.

Examples

- a. Remembering the example in the last section, here are the two trade market data messages:

The trade of 2 at 100 (assume the trade is the first of the session):

```
last price 100 quantity 2 mktTime 1103760000100 mktVolume 2 mktTrades 1
```

The trade of 5 at 100 (assume the trade is the second of the session):

```
last price 100 quantity 5 mktTime 1103760000105 mktVolume 7 mktTrades 2
```

- b. Again referring to example in the last section here are two book messages:

Before the match occurred:

```
book bid1 99 bqty1 6 ask1 100 aqty1 7 bid2 98 bqty2 55 ask2 101 aqty2 100 ask3 102 aqty3 11
```

After the match occurred:

```
book bid1 100 bqty1 3 ask1 101 aqty1 100 bid2 99 bqty2 6 ask2 102 aqty2 11 bid3 98 bqty3 55
```

Note the book message may have up to 5 price levels per the message specification, but the passed message may only have the non-zero price levels defined.

System Audit

The system audit function must be able to list messages sent and received by the server sorted by internal identifying sequence number and message type. Another list provided will show trades including the buyer and seller system ID's, quantity, price, time and a running total of Volume and trades for each trade that is done.

The structure of the pages are left to the developer.

Requirements Summary

The online trading engine (server) has a set of primary functional requirements:

- Managing client connectivity to the exchange and organizing client orders in an order book.
- Executing a first-in-first-out matching algorithm when new orders enter the market.
- Transmitting order fills to clients and broadcasting market data to clients.
- Recording all incoming and outgoing messages and provide an interface that can show the data in an organized fashion including important market statistics.

The online trading client also has a set of primary functional requirements:

- Establishing a connection to the trading engine (server).
- Handling order acknowledgements, negative acknowledgments and fill (trade). messages from the trading engine (server).
- Handling market data broadcast from the trading engine (server).

There are a set of "optional" functions that may be implemented related to:

- On the trading engine,
 - Client connectivity, implementing an authentication process when a client first connects.
 - Elegant handling of multiple clients, for example, an implementation where no two clients may enter orders with the same identification signature.
 - Enhanced statistical information stored or reported.
 - A process that notifies clients of market close.
- On the trading client,
 - A well designed GUI for order entry.
 - A well designed "intelligent" automated trading system (ATS).
 - Recording and accessing relevant data related to orders sent to the market and fills (trades) in the market.

Concluding Tips

- There are a lot of ways to design and engineer the solution, there is no expected coding or technology for any component. At the end of the day, the customer will have to decide whether you met the required functionality described and how you achieved it is secondary to functionality.
- Implement the system using the technology with which you are most comfortable, or at least get the parts of the system using familiar technology finished first.
- You can always add improvements, so concentrate on required functionality in development process while leaving yourself room to add the optional stuff if you have time.

Appendix – Message Specification

Order Entry			
TAG	Tag Type	Value Type	Comment
LIMIT	Header	None	Header Tag for limit order message
ClientID	Data	Long Int	Assumed unique identifier maintained at server for client
ClSeqID	Data	Long Int	Assumed unique sequence number for client messages passed to server
Quantity	Data	Integer	Non-zero integer value; positive indicates bid, negative ask
Price	Data	Integer	Non-zero integer value; positive indicates new order, negative cancel of a existing order
ClTime	Data	Long Int	UTC milliseconds

ACK	Header	None	
ClientID	Data	Long Int	Assumed unique identifier maintained at server for client
ClSeqID	Data	Long Int	Assumed unique sequence number for client messages passed to server
MktSeqID	Data	Long Int	Sequence number assigned to the order to identify the order at the server
ClTime	Data	Long Int	UTC milliseconds
MktTime	Data	Long Int	UTC milliseconds

NACK	Header	None	
ClientID	Data	Long Int	Assumed unique identifier maintained at server for client
CISeqID	Data	Long Int	Assumed unique sequence number for client messages passed to server
Quantity	Data	Integer	Non-zero integer value; positive indicates bid, negative ask
Price	Data	Integer	Non-zero integer value; positive indicates new order, negative cancel of a existing order
CITime	Data	Long Int	UTC milliseconds
MktTime	Data	Long Int	UTC milliseconds
Reason	Text	String	Reason for rejection

FILL	Header	None	
BuyerID	Data	Long Int	Assumed unique identifier maintained at server for client
SellerID	Data	Long Int	Assumed unique identifier maintained at server for client
Quantity	Data	Integer	Non-zero integer value; positive indicates bid, negative ask
Price	Data	Integer	Non-zero integer value; positive indicates new order, negative cancel of a existing order
BuyerMktSeqID	Data	Long Int	Sequence number assigned to the order to identify the order at the server
SellerMktSeqID	Data	Long Int	Sequence number assigned to the order to identify the order at the server
MktTime	Data	Long Int	UTC milliseconds

Market Data			
TAG	Tag Type	Value Type	Comment
last	Header	None	
Price	Data	Integer	Price of trade
Quantity	Data	Integer	Quantity of trade
MktTime	Data	Long Int	UTC milliseconds
MktVolume	Data	Long Int	Total quantity traded in session
MktTrades	Data	Integer	Total number of trades in session

book	Header	None	
Bid1	Data	Integer	Top Bid Price
BQty1	Data	Integer	Aggregate Qty at top bid price
Ask1	Data	Integer	Top Ask Price
AQty1	Data	Integer	Aggregate Qty at top ask price
Bid2	Data	Integer	Optional
BQty2	Data	Integer	Optional, Aggregate Qty
Ask2	Data	Integer	Optional
AQty2	Data	Integer	Optional, Aggregate Qty
Bid3	Data	Integer	Optional
BQty3	Data	Integer	Optional, Aggregate Qty
Ask3	Data	Integer	Optional
AQty3	Data	Integer	Optional, Aggregate Qty
Bid4	Data	Integer	Optional
BQty4	Data	Integer	Optional, Aggregate Qty
Ask4	Data	Integer	Optional
AQty4	Data	Integer	Optional, Aggregate Qty
Bid5	Data	Integer	Optional
BQty5	Data	Integer	Optional, Aggregate Qty
Ask5	Data	Integer	Optional
AQty5	Data	Integer	Optional, Aggregate Qty
MktTime	Data	Long Int	UTC milliseconds

Optional Messages

Market State			
TAG	Tag Type	Value Type	Comment
PING	Header	None	
ClientID	Data	Long Int	Assumed unique identifier maintained at server for client
CITime	Data	Long Int	UTC milliseconds

PONG	Header	None	
ClientID	Data	Long Int	Assumed unique identifier maintained at server for client
CITime	Data	Long Int	UTC milliseconds
MktTime	Data	Long Int	UTC milliseconds

NPONG	Header	None	
ClientID	Data	Long Int	Assumed unique identifier maintained at server for client
CITime	Data	Long Int	UTC milliseconds
MktTime	Data	Long Int	UTC milliseconds
Reason	Text	String	Reason for rejection

CLOSE	Header	None	
MktTime	Data	Long Int	UTC milliseconds