Code Jam Proposal: Electronic Stock Exchange

Contents

1 Overview

Your task is to implement an electronic stock exchange. Clients of a stock broker will be placing buy and sell orders for stocks on your exchange. For the purposes of this competition clients are “virtual” and are simulated by a desktop application that serves as an electronic broker. This broker application will be provided to you. You are responsible for implementing the exchange. Stock trade orders placed by clients are either completely or partially filled on a price time basis (to be defined later) by your exchange as market conditions allow. You are responsible for implementing the functionality to receive orders from the broker and to implement the trading algorithm for the exchange. As stock trades occur on the exchange one or more trade execution messages are sent back for an order to a client. These messages occur asynchronously as trade executions occur in the exchange. You are responsible for implementing the functionality to send these outbound messages from your exchange to the broker.

A client places orders to the exchange via a broker. The broker submits these orders to the exchange. In this competition there will be no actual clients. The broker application will be configured with “scripts” that simulate orders placed by individuals. In reality a client would receive updates as executions occur on orders placed by them. These updates are order execution messages that are sent back to the broker from the exchange. Again for competition purposes the broker application will simply track these execution messages as they are sent from the exchange and will not notify real clients. In addition, clients can receive direct updates via SMS messages sent back to his or her phone number. Your exchange will not implement any telephony related infrastructure to do this but, rather, it will communicate with clients via the Twilio telephony service using HTTP REST calls over the internet. The Twilio service is responsible for generating SMS messages to clients. Again for competition purposes there will be no real clients.

The processing flow for placing a stock trade and receiving updates is shown below. There are two scenarios: an inbound message scenario and an outbound message scenario. The inbound message scenario is the placing of a stock trade order via the broker. The outbound message scenario is the generation of trade execution messages to the broker and Twilio from the exchange.

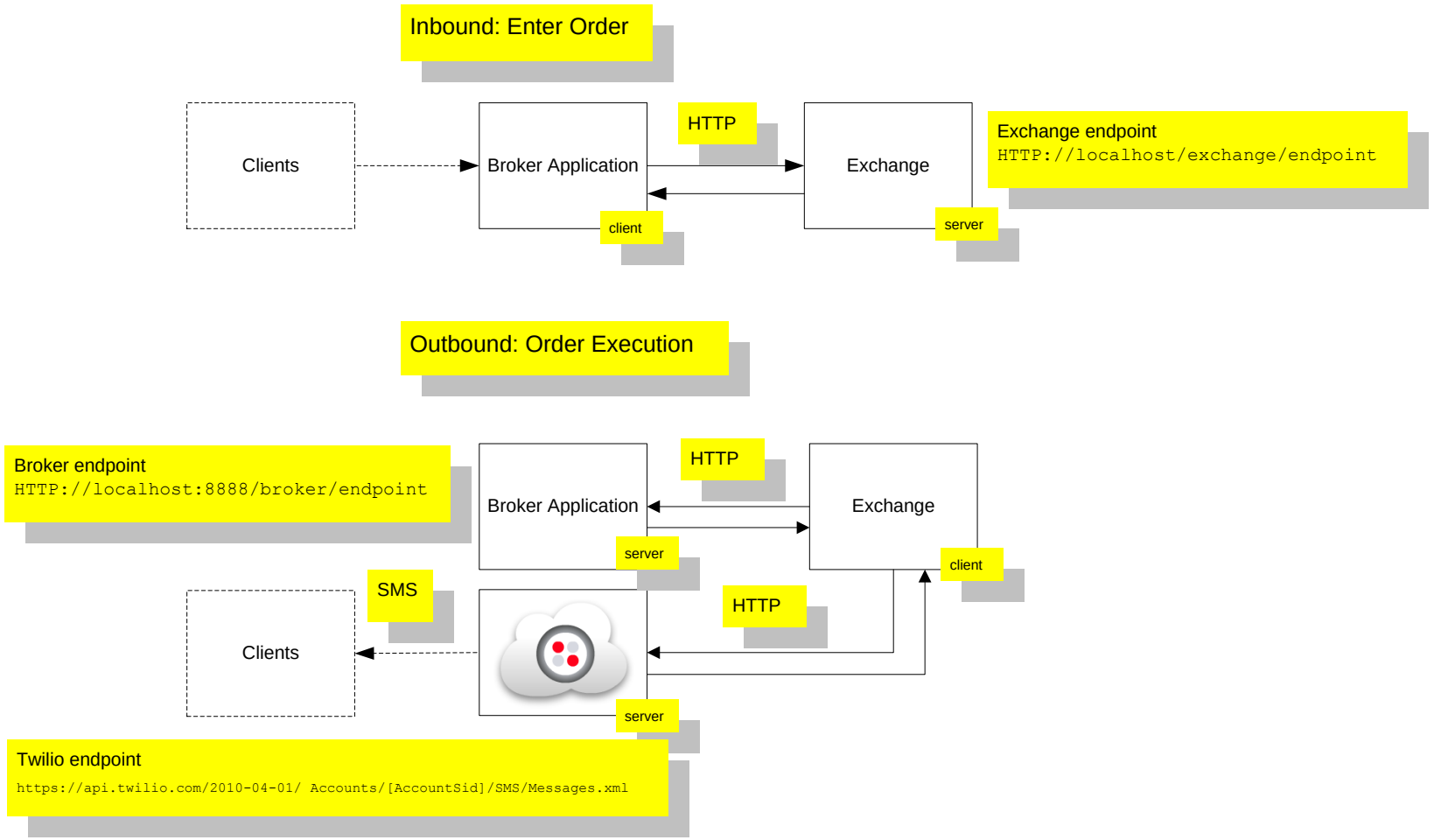


Figure 1. Inbound and outbound processing in the exchange.

Once again, your team is responsible for implementing the electronic stock exchange (the “Exchange” box in Figure 1) according to the provided specification. The broker application will be provided to you. Each competition team must have a Twilio account and a Twilio phone number. These will be used in tandem with the exchange application. You are implementing an application that is both client (during Outbound mode) and server (during Inbound mode). The application must be able to process orders via an HTTP POST to the exchange application URL from the broker application. The stock trade order parameters are encoded in the POST parameters. When a trade occurs the application must notify both

seller and buyer sides of the trade execution. This notification is done by a call back to the broker application using an HTTP POST. In addition, optional trade execution messages are sent back via a call to the Twilio service using an HTTP POST to an endpoint at Twilio.

Your application will track orders internally using an Order Book data structure. Your application must be able to generate a snap shot of all filled / unfilled orders on the Order Book as well as any trade executions that have occurred. The snapshot should be displayed in the GUI of your application. In addition, your application must be able to upload this snapshot in the form of a JSON dataset to a server located at Silanis. The Silanis server will generate an email to the judging committee's account.

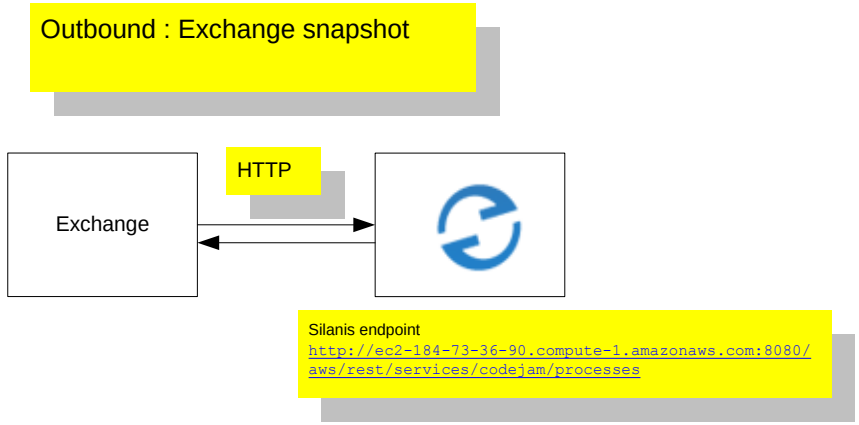


Figure 2. Data snapshot of upload to Silanis

Your application should provide a GUI. From the GUI your application should provide:

- A visual representation of the trades that have occurred in the exchange (more on this below)
- A "reset" control. This "reset" control should allow the judges to clear all orders and trade executions that are stored in the exchange.

You are free to implement the GUI using either web or desktop application paradigms.

1.1 Judging Criteria

Competitors will be judged on the following criteria (listed in order of importance below):

1. Successful implementation of the basic application functionality (i.e. send and receive HTTP calls) as described in the competition specification
2. Adherence to business rules and logic for the exchange; Correct implementation of the price time trade matching algorithm
3. The application must generate a visual data snapshot of the exchange.
4. The application must upload the data snapshot of the exchange to Silanis.
5. The application must contain a data visualization GUI of stock execution prices and amounts for a given stock traded on the exchange.
6. The ability of the application to handle large volumes of incoming HTTP requests in the form of new trade orders
7. The ability of the application to generate timely and correct outgoing HTTP calls as executions occur under heavy trading conditions
8. The application must generate HTTP calls to the Twilio service when required.

You probably will not be able to complete all eight of these deliverables. Do your best!

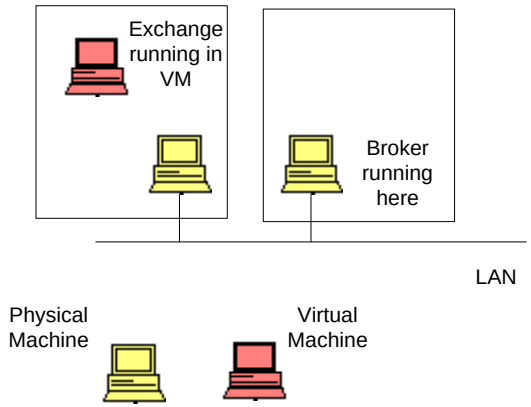
1.2 Judging Procedures

You will be provided with a virtual box image (Windows and/or Linux) on a USB key. You will install your application into this image. You will return the USB key at the end of the weekend. Make sure you provide with your image the following information:

- How to start and stop your application from inside the virtual image.
- The HTTP port and endpoint of your exchange where your exchange receives orders from the broker (e.g. http://localhost:8080/exchange/endpoint)
- If you are providing your GUI through a web based paradigm include the home page URL (e.g. http://localhost:8080/exchange/home) of your GUI.

Make sure you leave yourself plenty of time to configure the application in your virtual OS image. Last year a surprising number of submissions didn't run for the committee in the judging phase after the weekend.

Make sure that the Broker can connect to your exchange running in the virtual machine from another physical host on a local LAN. This is how we will test your exchange.



2 Business Details

The exchange application allows clients to buy and sell stocks electronically. Buyers and sellers transact with each other using orders. An order can be either a buy or sell order for an amount of shares at a fixed price. Stocks are bought and sold by matching orders for the same stock where the price of a sell order is lower or equal to the price of a buy order. If these conditions are met then a trade can be made. Making a trade is also known as an execution. When an order comes into the exchange it goes onto the Order Book. Orders are placed on the Order Book in the order they are received. The orders on the Book are matched or executed by the exchange on a price time priority basis (more details on this below).

2.1 Orders

An order consists of the following pieces of information.

- **Stock Symbol**: Symbol of the stock to be bought or sold in the order.
- **Amount**: Number of shares
- **Price of order**: Price per share. Exchanges generally have a price tick. This is the minimum increment by which a stock price can change. For example if an exchange has tick of 1 cent then share price is determined to the penny on the exchange and orders are specified to the penny.
- **Buy or Sell Indicator**
- **Telephone number.** Phone number of the person placing the order.

At the implementation level an order will have several additional pieces of information. These will be covered later.

2.2 Order Book

When a new order comes into the exchange it is matched against existing orders and then placed on to the Order Book. The Order Book is a time ordered list of orders that have previously come into the exchange. Each order is given a timestamp when it is placed on the Book. The order with the earliest timestamp at a given price is given preference when matching against a new order. Orders are matched against the new order by scanning the list of unfilled opposite orders for the same stock. This list is sorted on price and time stamp. If the incoming order is immediately executable, meaning it can be matched against one or more existing orders on the Book, one or more trade executions will occur. To be immediately executable, the new order must be:

- An order where an opposite already exists on the book (e.g. a sell for the same stock is on the book for a new buy order)
- An order to buy at a price at or above the lowest sell order on the Order Book
- An order to sell at a price at or below the highest buy order on the Order Book

An order can be filled partially. Both buy and sell orders that cannot be filled completely will partially fill and generate a residual order on to the Book for the remaining amount. Each order that comes into the exchange is assigned a unique identifier known as the “Order Reference Identifier”. This identifier is unique across all trades that take place during a market day or trading session.

2.3 Trade Executions

A stock trade is when a buyer and seller exchange money for an amount of stock. A stock trade has a buyer side and a seller side. The buyer buys an amount of shares from the seller at a price per share. Your application needs to match buyers and sellers appropriately and create records of the trades. This record of a trade is known as a trade execution. Your application needs to track all the executions that take place on the exchange during a trading session or market day. A trade execution consists of the following pieces of information

- **Stock Symbol**: Symbol of the stock that was traded.
- **Amount**: Number of shares
- **Price of order**: Price per share. The price at which the trade was executed. This will be the price of the sell order.
- **Buy Order.** The buy order of the trade. This is an order reference identifier assigned by the exchange.
- **Sell Order.** The sell order of the trade. This is an order reference identifier assigned by the exchange.
- **Time Stamp.** Time and date at which the trade occurred.
- **Match Number.** Each trade has a match number that is generated by the exchange. This number should be unique across a market session or trade day.

2.4 Example

Let's go through an example. Let’s suppose that between 9:00:00 and 9:00:03 three orders to buy stock XYZ at slightly different prices have come into the exchange. Each new order as it arrives is matched against existing orders on the book. Since there are no corresponding sell orders on the book all three buy orders are placed on the book unfilled. The orders came in at 9:00:01, 9:00:02, 9:00:03 and are for 50 shares at 91$ per share, 150 at 91$/share and 100 at 92$/share respectively. Each order has been assigned an order reference identifier. The order reference identifiers are B1, B2 and B3. At 9:00:04 an order to sell 200 shares of XYZ at $90.00 per share comes in. The four orders can now be filled. The outstanding orders on the book are filled in a price time basis. This means when the exchange looks for matches it starts with the buy order at the highest price above the sell order. If one or more buy orders have the same buy price then the oldest order *at that buy price* has priority. In this case the first matching buy order to the sell order is the B3 order because it has the highest buy price, the second matching is the B1 order (because it is the older of the two buy orders at 90.00) and the third matching order is the order B2. The first two buy orders would be filled completely and the last one (B2) would only be filled partially for 50 shares. A new residual order (O1) for the outstanding amount of 100 shares to be bought at 91$/share from parent order B2 would go on the Book.

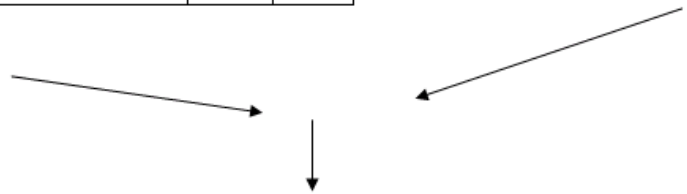
The trades are listed below the order book. Take note that there are three trades. One trade for each execution between sell order (S1) and the three buy orders (B1, B2 and B3).

Order Book

Time Stamp	Order Ref ID	Sell/Buy	Stock Amount	Parent	Order Price
9:00:01	B1	BUY	50		91.00
9:00:02	B2	BUY	150		91.00
9:00:03	B3	BUY	100		92.00

Incoming "SELL" Order

Time Stamp	Order Ref ID	Sell/Buy	Stock Amount	Parent	Order Price
9:00:04	S1	SELL	200		90.00



Order Book (after trade)

Time Stamp	Order Ref ID	Sell/Buy	Stock Amount	Parent	Order Price
9:00:01	B1	FILL	50		91.00
9:00:02	B2	FILL	150		91.00
9:00:03	B3	FILL	100		92.00
9:00:04	S1	FILL	200		90.00
9:00:04	O1	BUY	100	B2	91.00

Trades

Time Stamp	Match Number	Buy Order	Sell Order	Price	Amount
9:00:04	1	B3	S1	90.00	100
9:00:04	2	B1	S1	90.00	50
9:00:04	3	B2	S1	90.00	50

Figure 3. Order matching example.

Let's continue the scenario. Another order to sell comes into the exchange at 9:00:05 for 200 shares at 90.00. This order matches with the residual buy order O1. The new sell order will go on to the book as filled and a residual sell order O2 for 100 shares will be posted to the book.

Each residual order always has a parent order. In the case of residual order O1 the parent order is B2 and in the case of order O2 the parent of the residual order is S2. Residual orders are internal to the exchange and are used for book keeping purposes.

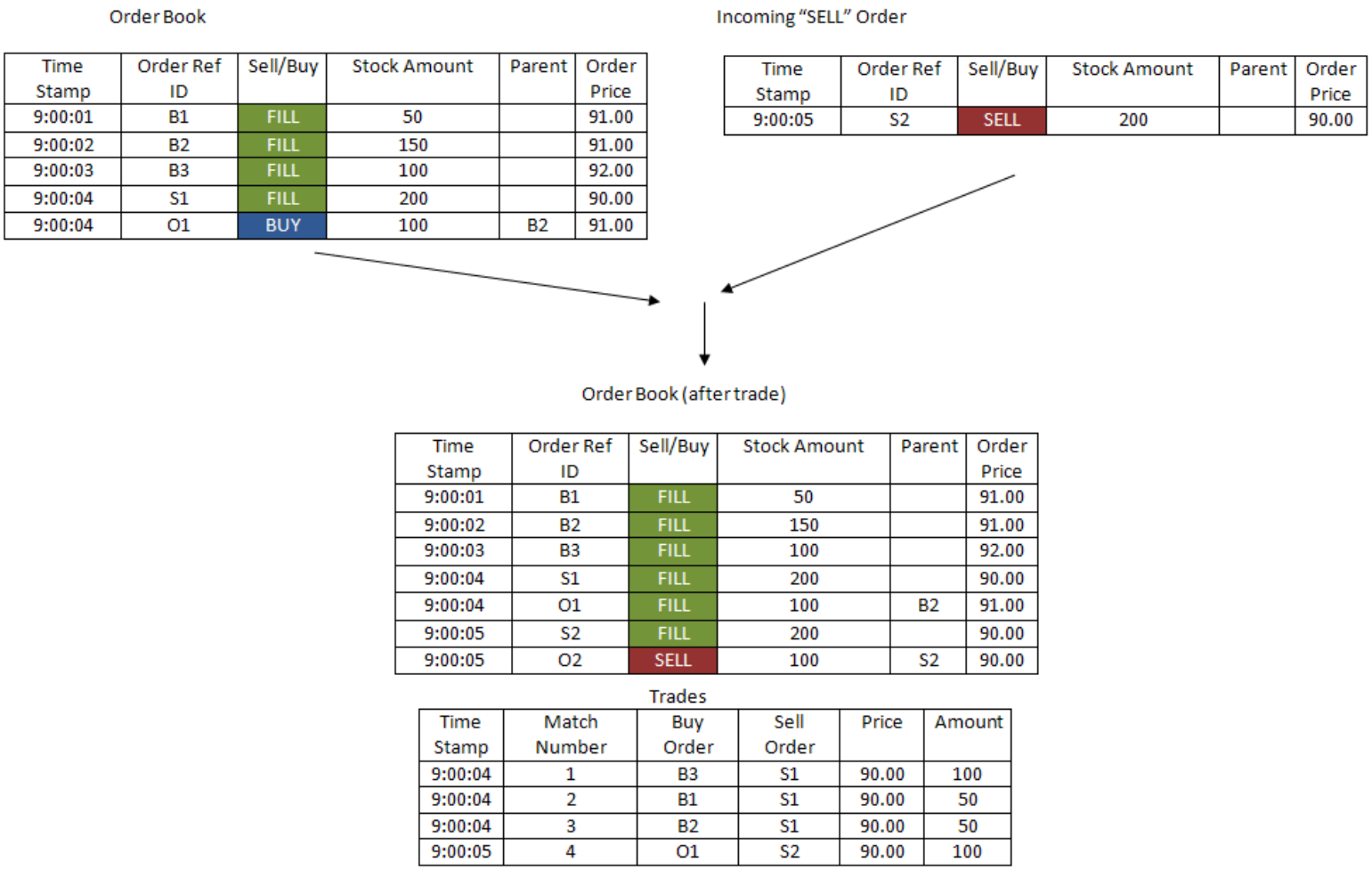


Figure 4. Order matching after second sell order (S2) has been received.

2.5 Trade Match Algorithm

After a new order has been accepted by the exchange it goes through the matching algorithm to find trade matches against unfilled orders on the book. The matching algorithm is executed prior to placing the new order on the book. A new order matches an existing order under the following conditions:

- The order on the book is unfilled.
- The two orders are for the same stock.
- The sell order price is smaller than or equal to the buy order price.
- The new order and the book order are a buy / sell or sell / buy pair.

If there are no matching orders on the book for the new order then the new order goes onto the book as an unfilled order. Otherwise the new order is marked as filled and a list of matching orders is checked in a price and time ordered basis.
If the new order is a buy order then:

- The list of matching orders is ordered on the sell price with the lowest sell price first (i.e. the lowest sell price should be the first match). Orders that have the same sell price are ordered according to their timestamp with the earliest order having the highest priority.

If the new order is a sell order then:

- The list is ordered on the buy price with the highest buy price first. Orders that have the same buy price are ordered according to their timestamp with the earliest order having the highest priority.

Please note that when you are sorting orders that have the same price the timestamp on residual orders should **not** be used. Instead the timestamp of the oldest ancestor (i.e. the "real" client order) of the residual should be used.

Descending the list of matching orders we mark each order as filled until there are no more shares from the new order to be matched. If there are remaining shares from either the last matched book order or the new order itself then a residual order is created from the order that has residual shares. The new order is then placed on the book.

3 Implementation Details

3.1 Message Overview

Communication between the exchange, broker, and Twilio will be done using HTTP POSTs. SMS messages will be sent in conjunction with the Twilio Cloud Service. There are two categories of messages: inbound and outbound messages. Inbound messages are sent from the broker application to the exchange application. Outbound messages are sent from the exchange to the broker application, Twilio and/or Silanis. The types of messages in the two categories are listed below.

Inbound Message (Broker to Exchange)

- Enter Order message
  Client places an order for a stock trade.

Outbound Message (Exchange to Broker, Twilio or Silanis)

- Trade Execution message
  Order has been executed (either partially or completely) by the exchange. A client may receive a number of these as the order is filled on the Book)
- Send SMS message via Twilio
- Send data snap shot message to Silanis.

Inbound messages come directly from the broker application to the exchange. In an inbound message scenario the broker acts as client and the exchange acts as server. Outbound messages originate with the exchange and are sent to the broker application, Twilio or Silanis. This is done via HTTP POSTs. Messages to Twilio are sent using the Twilio REST API via an HTTP POST. In an outbound message scenario the exchange acts as client and the destination (broker, etc) acts as server.

3.1.1 Message Field Data Types

Order messages contain information about stock orders. In order to simplify discussion and standardize the formatting and content of these messages we define “data types” here. We use the term “data type” loosely here since there is no real strong typing of fields in messages.

"Data Type"	Description
Integer	Contain any numeric ASCII character: 0….9
Alpha	Contain any of the following ASCII characters A…Z, a…z, 0….9
Single Character	Can be any single ASCII character A…Z, a…z
Telephone number	E.164 format. First character will be '+' followed by a sequence for digits up to a maximum of 15 digits. Examples: +15145551212 +914466135485 +911723083177

3.2 Inbound Message Details

Inbound messages are messages that are sent to the exchange from the broker application. Incoming messages from the broker application are sent via HTTP/1.1 protocol using an HTTP POST. Your exchange application must respond to an HTTP response with `Content-Type: text/xml; charset=UTF-8`. The payload of the HTTP response is a simple XML document which will be described below.

3.2.1 Enter Order

A client places a stock trade. The exchange is notified of an order by receiving an HTTP POST on its incoming message URL from the broker. You must configure the broker application to call your incoming message URL. An example URL is given below.

```
http://localhost/exchange/incoming
```

The parameters of the stock order message will be in the body of the POST. The parameters contain the information about the order. There are ten parameters passed in the body of the HTTP POST.

Enter Order Parameters

Parameter	Length	Value	Description
MessageType	1	“O”	Enter Order identifier
From	Max 16 characters	Telephone number	The phone number that sent this trade request. The phone number will be in E.164 format. This number servers to identify the client who placed the order
BS (Buy/Sell Indicator)	1	“B” or “S”	“B” = by order “S” = sell order
Shares	Maximum 6 Minimum 1	Integer	Total number of shares entered for the order. Must be greater than zero and less than 1,000,000.
Stock	Maximum 8 Minimum 3	Alpha	Stock symbol. May not be larger than 8 ASCII characters.
Price	Minimum 1 Maximum 6	Integer	The entered price. The price is assumed to be in fixed point format with 4 whole numbers followed by 2 decimal digits. The minimum price per share is 0.01 (parameter value 1). The maximum price is 1,000.00 (parameter value 100000).
Twilio	1	“Y” or “N”	“Y” = send an SMS back to the From telephone number via Twilio for each trade execution. “N”= no SMS notification.
BrokerAddress			IP or host name of broker application. Your exchange app should send trade execution messages to this IP or hostname. Examples: localhost 198.162.2.4 exchange.mcgill.ca
BrokerPort	Minimum 2 Maximum 5		Port of broker Example: 8888 80
BrokerEndpoint		Broker URL	URL of the broker. Example: broker/endpoint

This is an example of the POST request that would be received at your exchange URL.

```
POST /exchange/endpoint HTTP/1.1
Host: localhost
User-Agent: Mozilla/4.0
Content-Length: 70
Content-Type: application/x-www-form-urlencoded

MessageType=O&From=%2B15145551212&BS=B&Shares=100&Stock=XYZ&Price=9205&Twilio=N&BrokerAddress=192%2E168%2E2%2E4&BrokerPort=88
88&BrokerEndPoint=broker%2Fendpoint
```

This is a buy order from +1-514-555-1212 for stock XYZ for 100 shares at a price of 92.05 per share. No SMS are to be sent back for executions on this order. Trade execution messages on this order should be sent to

http://192.168.2.4:8080/broker/endpoint

Please note the body of an HTTP POST is encoded according to the url encoding scheme as described in RFC 1738. This page may be of use in understanding this. For this reason the "+" character is encoded as "%2B" in the body of the POST. Other characters such as ".", "/" should be encoded as well.

Expected HTTP Response

This is the response sent by the exchange in response to an Enter Order message. There are two possible responses to the Enter Order message. The first is an accept order response and the second is a reject order response. This response is sent synchronously in response to the HTTP POST from the broker like any normal HTTP conversation. Both types of message are HTTP responses with a MIME type of "text/xml" in "UTF8". In both cases the form of the response is an XML document. The form of the response is given below as a template. The contents of the [Response] placeholder listed below in the template will vary if the response is an acceptance or a rejection of the order. The "\n" is a line feed character.

HTTP Response Template

```
<?xml version="1.0" encoding="UTF-8"?>\n
<Response>\n
<Exchange>[Response]</Exchange>\n
</Response>\n
```

Accept / Reject	**[Response]**
Accept	<Accept OrderRefId="**[Order Reference Identifier]**" />
Reject	<Reject Reason="**[Rejection Reason]**" />

The place holder [Order Reference Identifier] in the message is the *order reference identifier*. This identifier is an Alpha up to a maximum of 8 characters. This identifier is assigned by the exchange to the order. Your exchange application is responsible for generating this identifier. The identifier **must** be unique across all orders during the trading session.

Name	**Length**	**Value**	**Notes**
Order Reference Identifier	Maximum 8 Minimum 1	Alpha	May not be larger than 8 ASCII characters.

The place holder [Rejection Reason] is a single ASCII character. Possible values for this code are given below. If an order is rejected by the exchange it is **not** assigned an order ref id. The exchange is free to discard it and does not place it on the book. If a message is rejected for several reasons only one code needs be passed back in the reply.

Rejection Reason	Description
M	Message type indicator is missing or invalid.
Z	Number of shares is missing or invalid.
S	Stock identifier is missing or invalid.
X	Share price is missing or invalid.
I	Buy/Sell indicator is missing or invalid.
T	Twilio indicator is missing or is invalid
P	Broker port is missing or is invalid
A	Broker address is missing or invalid.
E	Broker endpoint is missing or is invalid
F	From telephone number is missing or invalid

Several examples of responses are given below

```
<?xml version=”1.0” encoding=”UTF-8”?>\n
<Response>\n
<Exchange><Accept OrderRefId=”S1”/></Exchange>\n
</Response>\n
```

The order has been accepted by the exchange. The assigned order reference identifier is S1.

```
<?xml version=”1.0” encoding=”UTF-8”?>\n
<Response>\n
<Exchange><Reject Reason=”Z”/></Exchange>\n
</Response>\n
```

The order has been rejected by the exchange because the maximum number of shares per trade has been exceeded.

3.3 Outbound Messages

Outbound messages are messages that are sent by the exchange to the broker application, Twilio or Silanis. Outgoing messages from the broker application are sent via HTTP/1.1 protocol using an HTTP POST. In the case of the broker it will respond with an HTTP response with `Content-Type: text/xml; charset=UTF-8.` The payload of the HTTP response is a simple XML document which will be described below.

3.3.1 Execute Notifications

As trades are matched and executed the exchange must send outbound messages to the clients to notify them of executed trades. This is done by sending messages back to the broker application. If the client requests SMS notification as well then a message must be sent to the Twilio web service which will then send an SMS to the client phone number. For each execution message an HTTP post from your exchange application back to the broker application. The address, port and endpoint for this HTTP post is made using the address, port and endpoint that was provided in the original stock order. The body of the HTTP post will contain the parameters of the order execution. As per the HTTP 1.1 standard the POST

should be have a content type of `application/x-www-form-urlencoded` and the body of the HTTP POST will contain URL-encoded data. This page may be of use in understanding this. The parameters of the body of the POST are given below.

Order Execution Parameters

The name value pairs of the POST can occur in any order within the body of the HTTP message.

Name	Len	Value	Notes
MessageType	1	“E”	Execute Order identifier.
OrderReferenceIdentifier	Maximum 8 Minimum 1	Alpha	Original Order reference identifier assigned by the exchange. In the case of a trade against a residual order this should be the parent order reference Id.
ExecutedShares	Maximum 6 Minimum 1	Integer	Incremental number of shares executed. Smaller than original Enter Order share amount for partially executed orders. Must be greater than zero and less than 1,000,000.
ExecutionPrice	Minimum 1 Maximum 6	Integer	The price at which the trade for the shares was executed. The price is assumed to be in fixed point format with 4 whole numbers followed by 2 decimal digits. The minimum price per share is 0.01 (parameter value 1). The maximum price is 1,000.00 (parameter value 100000).
MatchNumber	Minimum 1 Maximum 8	Integer	This number is an Integer number with a minimum value of 1 and a maximum value 99,999,999. Assigned by exchange to each match executed. Each match consists of one buy and one sell. The matching buy and sell executions share the same match number. Match numbers must be trade day unique.
To	Max 16 characters	Telephone number	The phone number of the buyer or seller that sent the original trade order. The phone number will be in E.164 format.

Example

Below is an example of the two HTTP POSTs that your application would be responsible for making to the broker for a trade execution.

For the sell side of the trade execution message would be

```
POST /broker/endpoint HTTP/1.1
Host: localhost
User-Agent: Mozilla/4.0
Content-Length: 111
Content-Type: application/x-www-form-urlencoded

MessageType=E&OrderReferenceIdentifier=S1&ExecutedShares=100&ExecutionPrice=9000&MatchNumber=1&To=%2B15145551215
```

For the buy side of the trade execution message would be

```
POST /broker/endpoint HTTP/1.1
Host: localhost
User-Agent: Mozilla/4.0
Content-Length: 92
Content-Type: application/x-www-form-urlencoded

MessageType=E&OrderReferenceIdentifier=B1&ExecutedShares=100&ExecutionPrice=9000&MatchNumber=1&To=%2B15145551212
```

In this example your application is sending two HTTP POSTs to http://192.168.2.16/broker/endpoint . The trade execution for the sell side of the trade is for order ref id S1 for 100 shares executed at $90.00. The phone number of the seller is +15145551212. The match identifier that links the buyer and sell orders is 1. Similarly the trade execution for the buy side of the trade is for order ref id B1 for 100 shares executed at $90.00. The phone number of the buyer is +15145551215.

HTTP Response

The broker application will respond to a valid execute message with an HTTP 201 code. The following is an example of the response you would receive from the broker.

```
HTTP/1.1 201 Created
Content-Length: nn
Connection: close
Content-Type: text/html; charset=UTF-8
<?xml version=”1.0” encoding=”UTF-8”?>\n
<Response>\n
<Broker><Accept /></Broker>\n
</Response>\n
```

If parameters are missing, improperly formatted, do not correspond to the specification data type and lengths you will receive an HTTP 400 (Bad Request) code. The following is an example of bad execution message

```
HTTP/1.1 400 Bad Request
Content-Length: nn
```

```
Connection: close
Content-Type: text/html; charset=UTF-8
<?xml version=”1.0” encoding=”UTF-8”?>\n
<Response>\n
<Broker><Reject >[Reason explaination]</Reject></Broker>\n
</Response>\n
```

Twilio SMS Notification

If a client has asked for SMS notification of trade executions then the exchange would be responsible for sending corresponding SMS messages to Twilio. These outbound SMS messages are sent via HTTP POST to the Twilio URL:

```
https://api.twilio.com/2010-04-01/ Accounts/[AccountSid]/SMS/Messages.xml
```

The [AccountSid] part of the URL is a 32 character string corresponding to your Twilio Account SID. Your application must make the POST using Basic authentication. This page may be of use in understanding how to use Basic authentication. The authentication user ID is your Twilio Account SID and the password is your Twilio Account AUTHTOKEN which can be found in your Twilio account dashboard. The HTTP POST body parameters are given below. Once Twilio receives your HTTP POST it sends an SMS to the telephone number of the client. Details on using the Twilio REST API can be found at

http://www.twilio.com/docs/api/rest/sending-sms

HTTP Post Parameters

Parameter	Description
From	Your Twilio number that has been enabled for SMS. This is a phone number that you have purchased for your Twilio account. (see notes below)
To	Destination phone number.
Body	Information about the executed or cancelled trade.

The body of the outbound Twilio message will be a simple text string formatted according to this template.

```
Your order [OrderReferenceIdentifier] has been executed on for [ExecutedShares] shares. Match number is [MatchNumber].
Execution price is [ExecutionPrice] per share.
```

The place holders [] are for the order trade parameters given in the preceding section. Again you must URL encode the POST body parameters.

Twilio HTTP Response

Twilio will respond to your message with a TwilML response. Please see the above link for more details on the TwilML language and response format.

Using Twilio and sending an SMS

Sending an SMS using Twilio requires a "To" and a "From" number. The “To” number is the “To” phone number in the corresponding trade execution message that was sent to the broker. The “From” number in the Twilio message will be a phone number that is associated with your Twilio account. In order to get a phone number you will need to do two things:

- Upgrade your Twilio account using a credit card number. You can avoid charges to your card by using the promotion code “TWILIOLOVESCODEJAM” to give you a $40 credit on your account.
- Purchase a phone number that can send SMS texts (get a phone number with a US area code). Your account will be charged $1/month after this.

To help you with testing sending an SMS through Twilio you should use the following number as the “from” in your order messages :
+19176395561
This number is a Twilio phone number associated with a judging account. You can view SMS messages received at this number at the following URL : http://sms.mcgillcodejam.com/ in a web browser. Make sure to refresh the page after sending an SMS.

3.4 Message Examples

Let's go through the example given in the previous example section. We look at the sequence of inbound and outbound messages sent to and from the exchange. We will go through the messages with a high level logical view forgetting about most of the protocol level-details. As can be seen from the table below the exchange receives orders from four different phone numbers between 09:00:01 and 09:00:04. The first three orders are buy orders for stock XYZ. The exchange accepts each buyer's order. At 09:00:04 it receives a sell order from the first seller (+15145551215). After accepting the order from the first seller (+15145551215) the exchange matches up the trades and sends back order execution messages to all three buyers and the seller. Note that the first seller will get three order execution messages: one for each match. At 09:00:05 a second seller sends in a trade order to sell 200 shares of XYZ. The third buyer and the second seller will then receive trade execution messages for this trade. Note that the trade is a match between the residual order O1 and the sell order S2. The notification however is sent back to buyer 3 by looking up the parent order from the residual order. Note that the second seller had asked for Twilio notification on trade executions therefore an additional HTTP POST message to Twilio from the exchange will be sent.

Time	Broker		Exchange	Enter Order Message						
				Buy/Sell Indicator	Order Ref ID	Price	Shares	Stock	From	Twilio
9:00:01 AM	Enter Order	→		BUY	B1	91.00	50	XYZ	+15145551212	N
		←	Accept							
9:00:02 AM	Enter Order	→		BUY	B2	91.00	150	XYZ	+15145551213	N
		←	Accept							
9:00:03 AM	Enter Order	→		BUY	B3	92.00	100	XYZ	+15145551214	N
		←	Accept							
9:00:04 AM	Enter Order	→		SELL	S1	90.00	200	XYZ	+15145551215	N
		←	Accept							
Time	Broker		Exchange	Execution Notification Message						
					Order Ref ID	Execution Price	Executed Shares	Match Ref Number	To	Twilio
9:00:04 AM		←	Execute Notification		B3	90.00	100	1	+15145551214	N
	HTTP 201 / Accept	→								
9:00:04 AM		←	Execute Notification		S1	90.00	100	1	+15145551215	N
	HTTP 201 / Accept	→								
9:00:04 AM		←	Execute Notification		B1	90.00	50	2	+15145551212	N
	HTTP 201 / Accept	→								
9:00:04 AM		←	Execute Notification		S1	90.00	50	2	+15145551215	N
	HTTP 201 / Accept	→								
9:00:04 AM		←	Execute Notification		B2	90.00	50	3	+15145551213	N
	HTTP 201 / Accept	→								
9:00:04 AM		←	Execute Notification		S1	90.00	50	3	+15145551215	N
	HTTP 201 / Accept	→								

Time	Broker		Exchange	Enter Order Message						
				Buy/Sell Indicator	Order Ref ID	Price	Shares	Stock	From	Twilio
9:00:05 AM	Enter Order	→		SELL	S2	90.00	200	XYZ	+15145551216	Y
		←	Accept							
Time	Broker		Exchange	Execution Notification Message						
					Order Ref ID	Execution Price	Executed Shares	Match Ref Number	To	Twilio
9:00:05 AM		←	Execute Notification		B2	90.00	100	4	+15145551214	N
	HTTP 201 / Accept	→								
9:00:05 AM		←	Execute Notification		S2	90.00	100	4	+15145551216	Y
	HTTP 201 / Accept	→								

3.5 Data Snapshot

Your application must provide the ability to generate a snapshot of all the orders on the Order Book as well as any trade executions that have occurred. You should provide the ability to take a snapshot using a button or similar visual control in the GUI of your application. The judge should be able to click on the control in the GUI of your application to take the snapshot.

Initiating the snapshot will open a window or webpage that will give the data in the exchange in tabular format. In the window or webpage there must an additional button or control that will upload a JSON dataset from the exchange application to the Silanis web service sandbox.

The following information should be provided in the snapshot in a flat row-based format. Some of the fields are left blank depending on the type of record in the data snapshot. The data should be sorted on the “Time Stamp” field.

Field	Value	Notes
Time Stamp of the order or execution	YYYY-MM-DDThh:mm:ss.s Example 2011-10-06T09:04:01.9	Should be formatted in ISO 8601 format
Buy/Sell/Execute	“B” “S” or “E”	Indicates the type of record
Order Reference ID	Alpha	Ignored or empty for trade executions. As per spec for Order messages.
Execution Match Number	Integer	Ignored or empty for trade orders. As per spec for Execution messages.
Stock Amount	Integer	As per spec for Order messages.
Stock Symbol	Alpha	As per spec for Order messages.
Sell Order Reference ID	Order Reference Identifier	As per spec for Order messages. Can be a residual order.
Buy Order Reference ID	Order Reference Identifier	As per spec for Order messages. Can be a residual order.
Parent Order Reference ID	Order Reference Identifier	As per spec for Order messages.
Price	Integer	As per spec for Order messages.
State (FILL/UNFILLED)	“F” or “U”	
Client Telephone Number	E.164 format +16175551212	As per Twilio REST API. Ignored for trade Executions.

An example of this data snapshot is shown below. This is based on the example from previous sections (after the exchange receives order B1, B2, B3, S1 and S2).

Time Stamp	Buy Sell or Execute	Order Reference ID	Execution Match Number	Stock Amount	Stock Symbol	Sell Order Reference ID	Buy Order Reference ID	Parent Order Reference ID	Price	State (FILL or UNFILLED)	Client Telephone Number
2011-10-06T09:00:01.2	B	B1		50	XYZ				91.00	F	+15145551212
2011-10-06T09:00:02.3	B	B2		150	XYZ				91.00	F	+15145551213
2011-10-06T09:00:03.1	B	B3		100	XYZ				92.00	F	+15145551214
2011-10-06T09:00:04.0	S	S1		200	XYZ				90.00	F	+15145551215
2011-10-06T09:00:04.1	E		1	100	XYZ	S1	B3		90.00		
2011-10-06T09:00:04.2	E		2	50	XYZ	S1	B1		90.00		
2011-10-06T09:00:04.3	E		3	50	XYZ	S1	B2		90.00		
2011-10-06T09:00:04.4	B	O1		100	XYZ			B2	91.00	F	+15145551213
2011-10-06T09:00:05.0	S	S2		200	XYZ				90.00	F	+15145551216
2011-10-06T09:00:05.1	E		4	100	XYZ	S2	O1		90.00		
2011-10-06T09:00:05.2	S	O2		100	YXZ			S2	90.00	U	+15145551216

As stated earlier the data displayed in the GUI should be uploaded to Silanis. This is done using an HTTP POST with Basic authentication. The authentication user id is *codejam* and the authentication password is *secret*. This authentication information can be included in your HTTP request by adding the following HTTP header to your request:

```
Authorization: Basic Y29kZWphbTpzZWNyZXQ=
```

The content-type header in your message should be set to: `Content-Type:application/json`

The endpoint for the HTTP message is an Amazon cloud instance dedicated to the competition at:

`http://ec2-184-73-166-185.compute-1.amazonaws.com/aws/rest/services/codejam/processes`

on port 80.

The body of the request is a JSON-formatted message with the following name/value pairs. This page may be of use in understanding the JSON format.

JSON Name	Description
Name	Descriptive name for your upload message. Example: `"name": "Test Signing Process 1"`
description	Description name/value pair. Example: `"description": "Trading snapshot 1"`
owner	This is the initiator of the signing process. It consists of name and email name/value pairs. This information should identify your team. Example: `"owner": { "name": "Student X Team", "email": "student@mcgill.ca" }`
signer	This is the competition judge email. It also consists of name and email pairs. When testing you should use your own email (make it different from the owner). Example: `"signer": { "name": "Judge Judy", "email": "codejamjudge@mcgill.ca" }`

transaction	This is the trading transaction snapshot. It consists of all elements found in section 3.5: timestamp , action , orderRef , matchNumber , amount , symbol , sellOrderRef , buyOrderRef , parentOrderRef , price , state , phone
	Please note this is an array of JSON objects. There is one object for each record in the snapshot. Example:
	`"transactions" : [`
	`{ "timestamp" : "2011-10-06T09:00:01.2", "action" : "B", "orderRef" : "B1", "matchNumber" : "", "amount" : "50", "symbol" : "XYZ", "sellOrderRef" : "", "buyOrderRef" : "", "parentOrderRef" : "", "price" : "91.00", "state" : "F", "phone" : "+5141112222" },`
	`{ "timestamp" : "2011-10-06T09:00:02.3", "action" : "B", "orderRef" : "B2", "matchNumber" : "", "amount" : "150", "symbol" : "XYZ", "sellOrderRef" : "", "buyOrderRef" : "", "parentOrderRef" : "", "price" : "91.00", "state" : "F", "phone" : "+5141113333" }`
	`]`

The body of the data snapshot consists of the name, description, owner, signer and a list of transactions.

Example:

```
{
  "name": "Test Signing Process 1",

  "description": "Trading snapshot 1",

  "owner": { "name": "Student X", "email": "student@mcgill.ca" },

  "signer": { "name": "Judge Judy", "email": "codejamjudge@mcgill.ca" },

  "transactions" : [
    { "timestamp" : "2011-10-06T09:00:01.2", "action" : "B", "orderRef" : "B1", "matchNumber" : "", "amount" : "50", "symbol" : "XYZ",
"sellOrderRef" : "", "buyOrderRef" : "", "parentOrderRef" : "", "price" : "91.00", "state" : "F", "phone" : "+5141112222" },
    { "timestamp" : "2011-10-06T09:00:02.3", "action" : "B", "orderRef" : "B2", "matchNumber" : "", "amount" : "150", "symbol" :
"XYZ", "sellOrderRef" : "", "buyOrderRef" : "", "parentOrderRef" : "", "price" : "91.00", "state" : "F", "phone" : "+5141113333" }
  ]
}
```

Example HTTP Request using curl:

```
#! /bin/sh

# create signing process
`curl -X "POST" -H "Authorization: Basic Y29kZWphbTpzZWNyZXQ=" -H "Content-Type:application/json" --data-binary @cjProcess.json
"http://ec2-184-73-166-185.compute-1.amazonaws.com/aws/rest/services/codejam/processes"`
```

3.6 Graphical User Interface

Your exchange application must provide a graphical user interface. The GUI must provide the following functionality:

- A reset control. Clicking on the reset control resets the stock exchange. All orders are cleared from the book. All executions are cleared from the exchange.
- A data snapshot control. Clicking on this control opens the data snap view. Clicking on another button in the view sends the data snap shot of the exchange to Silanis as per earlier sections of this specification.
- A visualization of movements of stock price and amount for a selected stock. The user interface must allow the end user to select one of the stocks that has been traded on the exchange. Selecting the stock should show two graphs. The first graph should show executed trades at the prices the executions occurred at. This should be shown as a function of when the execution occurred. The second graph should show the execution amounts as a function of execution time in the exchange. An example of this window is shown below. For large volumes of trades the GUI should summarize by “binning” the data appropriately as a function of time tick on the x-axis and display the average the volumes and prices within each bin.

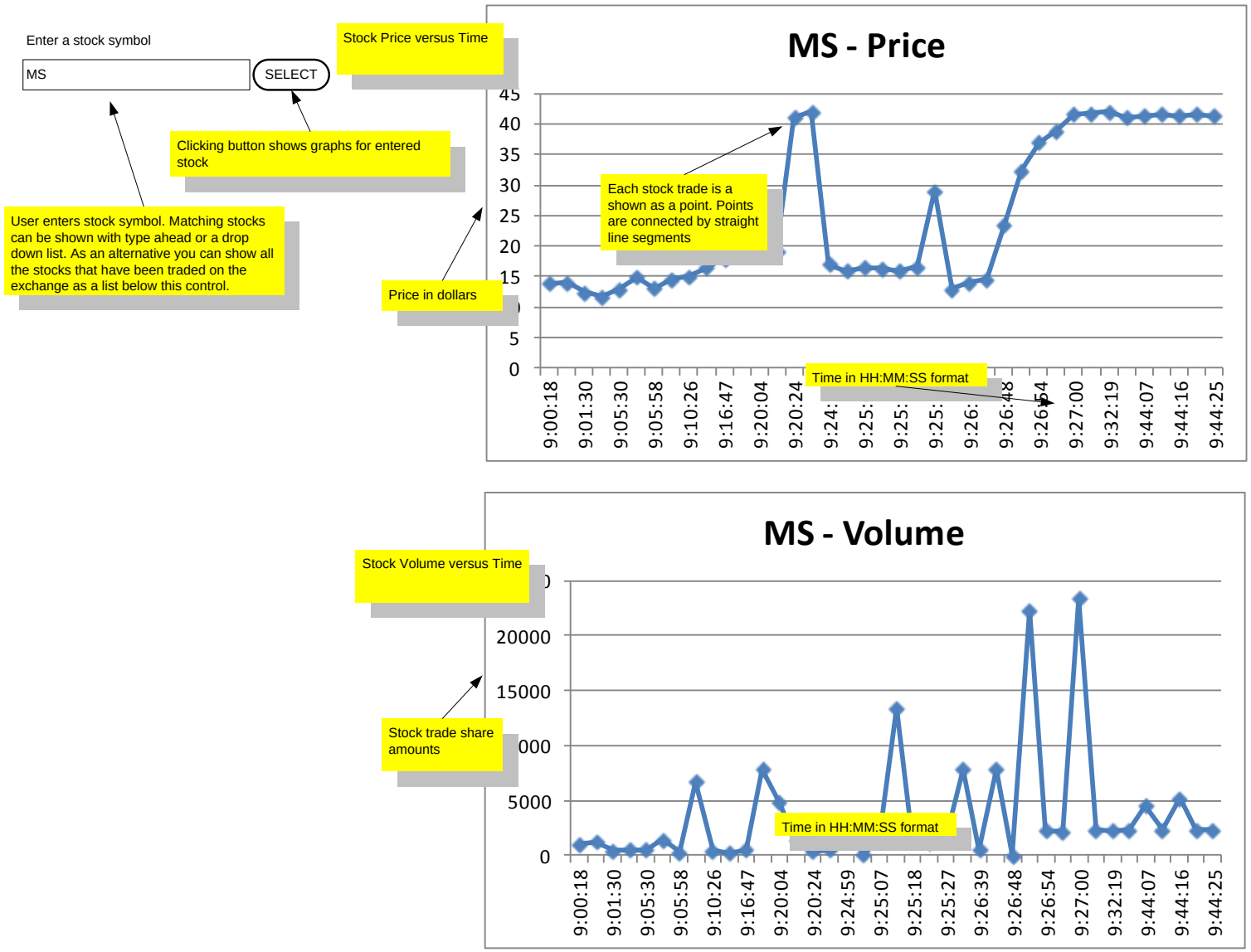


Figure 5. Example of stock data visualization screen.

4 Trade Order / Execution Data

Below is a sample trading session that extends the examples discussed earlier. It may be of use to you to compare the results from your implementation of the matching algorithm with the sample given below.

```
TIME STAMP: 900001
Comment:    Order B1 Buy 50 @ 91.00
 -------------- ORDER BOOK -----------------------
 Order Ref Id          Time     Stock Sym        Filled      Buy/Sell        Amount         Price        Parent
           B1        900001           XYZ             U           BUY            50          9100          NULL
 -------------- Executions -----------------------
 Match Number          Time     Stock Sym           Buy          Sell        Amount         Price

TIME STAMP: 900002
Comment:    Order B2 Buy 150 @ 91.00
 -------------- ORDER BOOK -----------------------
 Order Ref Id          Time     Stock Sym        Filled      Buy/Sell        Amount         Price        Parent
           B1        900001           XYZ             U           BUY            50          9100          NULL
           B2        900002           XYZ             U           BUY           150          9100          NULL
 -------------- Executions -----------------------
 Match Number          Time     Stock Sym           Buy          Sell        Amount         Price

TIME STAMP: 900003
Comment:    Order B3 Buy 100 @ 92.00
 -------------- ORDER BOOK -----------------------
 Order Ref Id          Time     Stock Sym        Filled      Buy/Sell        Amount         Price        Parent
           B1        900001           XYZ             U           BUY            50          9100          NULL
           B2        900002           XYZ             U           BUY           150          9100          NULL
           B3        900003           XYZ             U           BUY           100          9200          NULL
 -------------- Executions -----------------------
 Match Number          Time     Stock Sym           Buy          Sell        Amount         Price
```

TIME STAMP: 900004
Comment: Order S1 Sell 200 @ 90.00 Generate three executions and residual for B2. Note the sell price is lower than the buy price.
We still have a match. The converse is not true however: a sell price higher than a buy price is not a match !!

---------- ORDER BOOK ----------

Order Ref Id	Time	Stock Sym	Filled	Buy/Sell	Amount	Price	Parent
B1	900001	XYZ	F	BUY	50	9100	NULL
B2	900002	XYZ	F	BUY	150	9100	NULL
B3	900003	XYZ	F	BUY	100	9200	NULL
S1	900004	XYZ	F	SELL	200	9000	NULL
3a3b68	900004	XYZ	U	BUY	100	9100	B2

---------- Executions ----------

Match Number	Time	Stock Sym	Buy	Sell	Amount	Price
0	900004	XYZ	B3	S1	100	9000
1	900004	XYZ	B1	S1	50	9000
2	900004	XYZ	B2	S1	50	9000

TIME STAMP: 900005
Comment: Order S2 Sell 200 @ 90.00 Generate 1 execution and a residual from this order

---------- ORDER BOOK ----------

Order Ref Id	Time	Stock Sym	Filled	Buy/Sell	Amount	Price	Parent
B1	900001	XYZ	F	BUY	50	9100	NULL
B2	900002	XYZ	F	BUY	150	9100	NULL
B3	900003	XYZ	F	BUY	100	9200	NULL
S1	900004	XYZ	F	SELL	200	9000	NULL
3a3b68	900004	XYZ	F	BUY	100	9100	B2
S2	900005	XYZ	F	SELL	200	9000	NULL
3a3e30	900005	XYZ	U	SELL	100	9000	S2

---------- Executions ----------

Match Number	Time	Stock Sym	Buy	Sell	Amount	Price
0	900004	XYZ	B3	S1	100	9000
1	900004	XYZ	B1	S1	50	9000
2	900004	XYZ	B2	S1	50	9000
3	900005	XYZ	3a3b68	S2	100	9000

TIME STAMP: 900006
Comment: Order B4 Buy 100 @ 92.00 This order will fill the residual from S2.

---------- ORDER BOOK ----------

Order Ref Id	Time	Stock Sym	Filled	Buy/Sell	Amount	Price	Parent
B1	900001	XYZ	F	BUY	50	9100	NULL
B2	900002	XYZ	F	BUY	150	9100	NULL
B3	900003	XYZ	F	BUY	100	9200	NULL
S1	900004	XYZ	F	SELL	200	9000	NULL
3a3b68	900004	XYZ	F	BUY	100	9100	B2
S2	900005	XYZ	F	SELL	200	9000	NULL
3a3e30	900005	XYZ	F	SELL	100	9000	S2
B4	900006	XYZ	F	BUY	100	9200	NULL

---------- Executions ----------

Match Number	Time	Stock Sym	Buy	Sell	Amount	Price
0	900004	XYZ	B3	S1	100	9000
1	900004	XYZ	B1	S1	50	9000
2	900004	XYZ	B2	S1	50	9000
3	900005	XYZ	3a3b68	S2	100	9000
4	900006	XYZ	B4	3a3e30	100	9000

5 Web Frameworks

Some of these links might help you in terms of building your application

Java

Apache Tomcat, Jetty
http://www.ibm.com/developerworks/web/library/x-servlets-and-xml/
http://www.cs.usfca.edu/~parrt/course/601/lectures/servlets.html
http://www.seas.upenn.edu/~cis330/jetty.html
http://stackoverflow.com/questions/5069433/static-content-with-maven-jetty-plugin

PHP

http://www.the-art-of-web.com/php/ajax/
http://www.tonymarston.co.uk/php-mysql/dom.html

CGI

http://www.jmarshall.com/easy/cgi/

Python

Web.py (http://webpy.org)
http://webpy.org/cookbook/xmlfiles

PERL

HTTP::Engine
http://search.cpan.org/~miyagawa/Starman-0.2014/lib/Starman.pm
http://plackperl.org/
http://en.wikibooks.org/wiki/Perl_Programming/HTTP::Engine

VB or C# (ASP.NET / IIS)

HTTPWebRequest

http://www.4guysfromrolla.com/articles/122204-1.aspx

HTTP Handlers

MVC