Software Architecture Specification Document

Project:
Financial markets simulation with multiple competing algorithmic trading entities.

Client: Cortical Systems

Group:

GR∀PƎ

For every propblem, there exists a solution...

$\epsilon > 0$

Members:

Daniel Makgonta 12147100
Moeletji Semenya 12349136
Madimetja Shika 12127877

Publication Date: 22 August 2014

Version: 0.4

Change Log

Date	Name	Reason	Versio
13/05/2014	Moeletji	Creation	0.0
16/05/2014	Moeletji	Adding functional requirements	0.0
19/05/2014	Daniel	Added Architectural Requirements	0.0
21/05/2014	Madimetja	Editing	0.0
12/07/2014	Daniel	Edited Architecure requirements	0.1
15/07/2014	Moeletji	Added to functional requirements	0.1
22/07/2014	Madimetja	Edited the document	0.1
23/07/2014	Daniel	Updated Architecure requirements	0.1
23/07/2014	Moeletji	Updated functional requirements	0.1
27/07/2014	Daniel	Revised architectural requiremets	0.1
28/07/2014	Moeletji	Added function requirements	0.1
29/07/2014	Madimetja	Edited and Reviewd document	0.1
31/07/2014	Madimetja	Added vision and scope sections	0.1
01/08/2014	Moeletji	Added final images and compiled document	0.2
19/08/2014	Moeletji	Added diagram and updated functional requirements	0.3
20/08/2014	Daniel	Added section for GUI and pictures	0.3
21/08/2014	Madimetja	Updated Class Diagram document	0.3
22/08/2014	Daniel	Edited Architectural Constraints	0.3
22/08/2014	Moeletji	Finalising new version	0.4

Contents

1 Vision and Scope

1.1 Vision

The vision is to create a system that simulates the activities of a real-world financial market and the entities involved, while allowing interaction from both real-word and virtual entities. Furthermore, the aim is to observe how different multiple trading strategies behave when competing against each other and how they drive the price or value of securities.

1.2 Scope

The proposed system is a financial markets simulator system which will allow trading entities to:

- trade instruments through the system as they would in a real-world financial market while adhering to free market principles,

- observe and follow trends in the market through the system by means of observing technical indicators and strategies. This would furthermore provide entities with information on when best to enter or exit the market,

- compete with each other by assessing the profit and loss they make.

In particular, apart from allowing the participants to compete with each other in terms of the profits or loss they make, the system will allow the following specific activities for participants:

- Sellers (Offerors)

    - make offers to sell shares.
    - sell shares to buyers.

- Buyers (Bidders)

    - propose bids to buy shares.
    - search for shares on sale.

- buy shares on sale.

- Companies

  - Enter the market (system) and list stocks on the market (system).

1.3   Limitations/Exclusions

The project scope explicitly excludes and/or is limited in the following manner:

- Stock-brokers will not be explicitly active in the system as individual separate independent entities. The responsibilities of the stock-broker will be handled and performed by the matching engine and relevant market managers.

This part of document discusses the requirements around the software infrastructure within which the application functionality is to be developed.

2 Access channel requirements

The system will be accessible by human users through the following front-end channels:

- From a stand-alone Java application interface. The system will be accessible from any desktop or laptop once the appication is installed on the device.

3 Quality Requirements

3.1 Usability

- The system should be usable in that the architecture should provide the neccessary components to view and interact with the system. Furthermore, the system should be easily understood and self-affordable to novice users or users unfarmilia with financial markets.

3.2 Scalability

- If need be, the system should allow for independent entities operating from independent machines to participate in the market. i.e. The system should allow for at least two (2) participants to participate in the market from separate machines.

- If need be, the system should allow for concurrent trading between at least two (2) separate machines.

3.3 Performance

- In the case that multiple physical machines are used, the architecture should allow offers and trades to be communicated to and from the

matching engine via the network in under 50 milliseconds (1/20 of a second).

4   Integration Requirements

The system is self-contained and there is no need for integration with other external systems or components.

5   Architectural Constraints

In order to minimize cost and complexity, the system can be developed to run and operate on a single physical machine.

6 Architectural requirements

6.1 Architectural scope

- Concurrent threading execution of Market Participants for a realistic simulation.

- A relational database will be the persistent infrastructure used to record data.

- Full API of the system will be available.

6.2 Quality requirements

6.2.1 Security

- Only system administrators may tweak trading algorithms or matching engine to follow current trends within the stock market.

- The system allows anonymity in terms of no buyer or seller can view what another buyer or sellers activities.

- Matching engine is only allowed to be accessed directly by system administrators.

- The matching engine is abstracted to the buyers and sellers of the system.

- System administrators are not allowed to participate in the trading simulation.

6.2.2 Auditability

- One can query any Market Data generated within the system, this includes the user who made the change, the data that was changed, the new and old values of the data, as well as when the data was changed

6.2.3 Testability

- All services provided by the system are testable with unit tests, that the service is provided if all pre-conditions are met (i.e. that no exception is raised except if one of the pre-conditions for the service is not met), and that all post-conditions hold true once the service has been provided.

6.2.4 Usability

- System will support up to 40 Market Participants and 3 Stock Markets.

- The system will be presented in the English language, and with further development will cater for other popular languages.

- The Graphical User Interface will use design principals and usability goals from interaction design theoretical frameworks to increase the ease of using the system for the first time.

- All calculations will be abstracted for the users and the Graphical User Interface will only show averages and final results to the user.

6.2.5 Scalability

- The system will be able to use 10 trading algorithms and allow more algorithms to be plugged in at a later stage.

- The system will be able to concurrently trade with 20-40 Market Participants trading 1-3 different Market Entities effectively and accurately according to the business rules of the system.

- The system will have an API that can be used with different interfaces.

6.2.6 Performance

- The matching engine will return matches in less than 0.5 seconds

- Reporting of the market will be displayed in less than 5 seconds

- The buyers and sellers will be able the concurrently interact with the market simulation and this would take less than 0.5 second

6.3   Access Channel Requirements

The system is a stand-alone system and does not integrate with any other systems. The system will be accessible by human users through the following channel:

- From any system that contains a JVM (Java Virtual Machine).

6.4   Architectural Constraints

- System runs on a JVM

- System uses MySQL as its database

- Operating systems: Windows, Linux, Mac OS X

6.5   Architectural patterns/styles

- **Interface Segregation:** system has an API that can be used by other interfaces other than its native interface.

- **Singleton:** the system only allows an instance of one Stock Exchange to prevent overwhelming the systems resources. With improved system the system may be adapted to work with more than one stock exchange

- **Polymorphism:** All strategies inherit from a base strategy, but all sub-types have different strategies.

- **Encapsulation:** All Stocks are encapsulated by their managers to allow for further extensibility of the functionality.

- **Observer:** Market Participants will be notified of changes in Matching Engine.

- **Message Queue:** Market Participants will send Messages in the form of a Message Queue to the Matching Engine to ensure fairness.

6.6 Architectural Tactics and Strategies

- Full separate functional API allows for pluggable application interfaces.

- Multi-Threading Market Participants allow for real-time simulation.

6.7 Technologies

- **Built Environment:** Maven

- **Software:** Java

- **IDE:** Netbeans

- **Database:** db4o (Object-Orientated Database)

- **Programming Languages:** Java

- **Business logic:** Java

- **Interface:** Java (Swing library)

7 Architectural Constraints

- In order to minimize cost and complexity, the system can be developed to run and operate on a single physical machine.

8   User Interface

Main Interface allowing access to both the interface and matching engine.

Figure 1: Menu of Interface

Used for a novice trade user to understand what trading entails but views what actually happens within s matching engine.
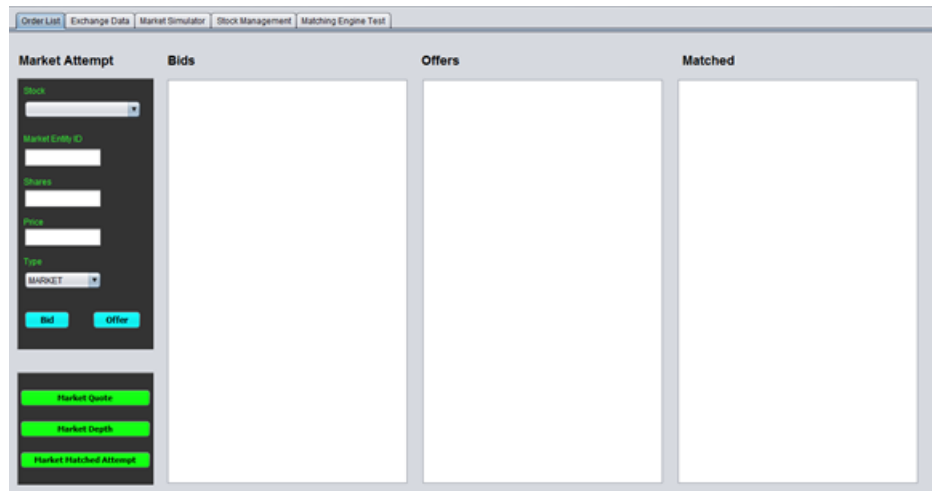

Figure 2: Matching Engine of Interface

Allows a user to create Stocks to trade in.

Figure 3: Create Stock

Allows a user to create custom Market Participants and having their own trading strategy.

Figure 4: Create Market Participant

Used to Start, Pause and Stop the simulator.

Figure 5: Simulation

Allows the user to view Market Data for both Stocks and Participants.

Figure 6: Market Data

View the Stock Prices.

Figure 7: Stock Prices

View the Technical Indicators.

Figure 8: Indicators

9 Functional requirements and application design

9.1 Introduction

This part of the document outlines the functional requirements of the *Financial market simulation* at the different levels of granularity. In the deisgn of the system we followed the SOLID design principles.
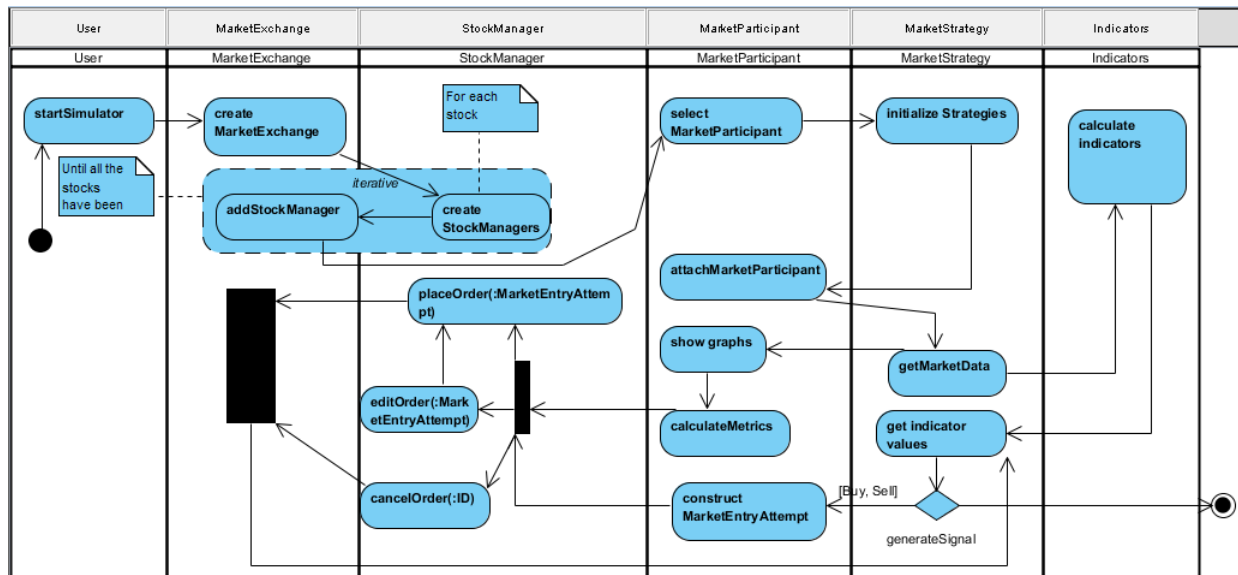

Figure 9: A high level activity diagram of the Financial Market Simulator

9.2   Required functionality

9.2.1   Matching Engine

- Requirement Prioritization: Critical

- Requirement Source: Client

- Requirement Level: High

Pre-condition :If no MarketEntryAttempts(bids and offers) are entered.
Post-conditions : A MatchedMarketEntryAttempt of a trade.
Request Data Structure : MarketEntryAttempt object.
Result Data Structure : MatchedMarketEntryAttempt object.

This is the core of the whole system. The following functionality should be provided:

- Provide a mechanism for accepting and maintaining market depth of bids and offers.

- Match the different types of market orders(e.g. primarily bids and offers).

- Notifying the relevant market participants of the matches involved in(if any).

- Report market data back to market participants.

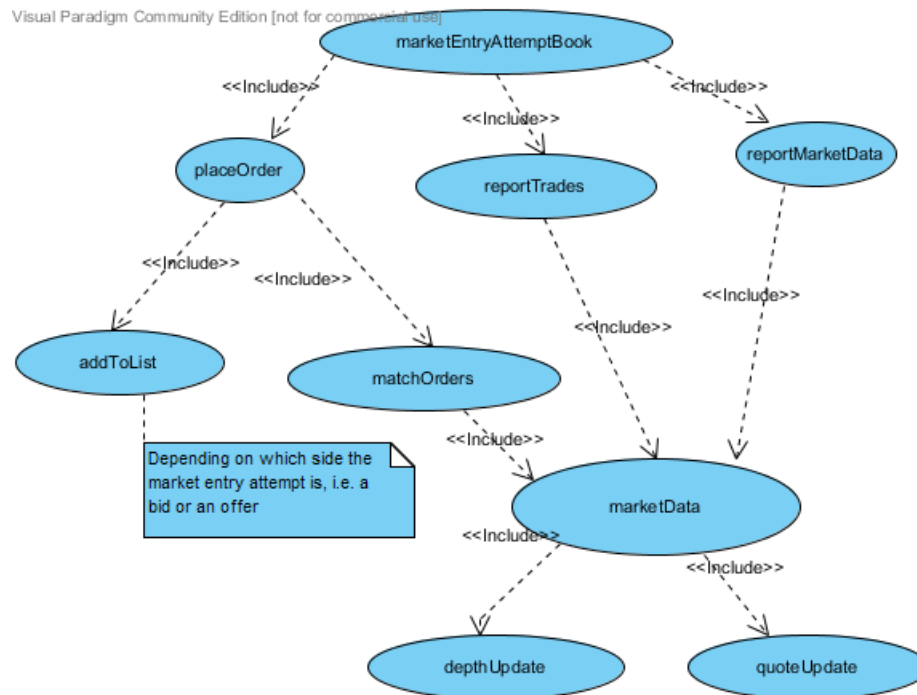So all bids and offers will be dealt with by the matching engine.

Figure 10: Use case for the matching engine

9.2.2 Keep track of the bids and offers

- Requirement Prioritization: Critical

- Requirement Source: Client

- Requirement Level: High

Pre-condition :If no MarketEntryAttempts(bids and offers) are entered. If the price or quantity of shares is zero.
Post-conditions : An ordered bids and offers stacks.
Request Data Structure : MarketEntryAttempt object.

The bids and offers should be stored in different stacks in the appropriate order. The offers will be stored in ascending order and the bids will be stored in descending order in terms of the price per share value. The market entry attempt will be stored if and only if their is no matching order(in terms of price) on the opposite stack.

This is an important part of the system as the correct order of the bids and offers must be stored because the order of the MarketEntryAttempts(bids and offers) is crucial to the functioning of the system.

Figure 11: The class with the market depth

9.2.3 Provide persistent storage for the market data generated by the system

- Requirement Prioritization: Critical

- Requirement Source: Client

- Requirement Level: High

Pre-condition :If no trades occur.
Post-conditions :Records of the trades that occured stored in the database.
Request Data Structure : MatchedMarketEntryAttempt object.

The financial market simulator is going to generate its own historical data so a mechanism needs to be put in place to store the market data of trades and other information used in generating trading events. These trading events will be using technical indicators and chart patterns.

A relational database will be used for the persistent storage of market data using the MySql server and a MySql connector to connect to the system. With this in place all the market data can be gathered in order to be analysed later to see how the financial market is behaving in certain circumstances.

9.2.4 User Interfaces for the market participants and the whole financial market

- Requirement Prioritization: Critical

- Requirement Source: Client

- Requirement Level: High

Pre-condition :If no trades occur.
Post-conditions :Records of the trades that occured stored in the database.
Request Data Structure : MatchedMarketEntryAttempt object.

This requirement should provide the following user interfaces:

- For displaying the various metrics of the performance of any selected market participant for comparing the different algorithms' performance.

- For showing the full market depth and matching orders by the matching engine.

For any participant, their interface should display all the participant's market data and whether the participant is making a profit or a loss. The other part of the user interface will display all the activity going on in the market to see how the market it performing at that point in time.

9.2.5 Calculating technical indicators for trading strategies

- Requirement Prioritization: Critical

- Requirement Source: Client

- Requirement Level: High

Pre-condition :If there is no data available(as in trades) .
Post-conditions : A value(s) calculated by the technical indicator.

Technical indicators are used for analysing and interpreting market behaviour. This ranges from analysing the price of a stock or the number of shares traded or the trend of a certain share. The indicators generate value(s)

and are interpreted so that they can be used by the strategies which will decide whether to enter(bid) or exit(offer) the market.

Some of the indicators use each other to compute more complex technical indicators. So far 13 technical indicators have been implemented(*A library will be used if/when a suitable one is found).

They are:

- Simple Moving Average(SMA)

- Exponential Moving Average(EMA)

- Negative Directional Index(NDI)

- Positive Directional Index(PDI)

- Negative Directional Movement(NDM)

- Positive Directional Movement(PDM)

- Relative Strength Index(RSI)

- Moving Average Convergence Divergence(MACD)

- Average Directional Index(ADX)

- Average True Range(ATR)

- Volatility(Standard deviation)

- Bollinger Bands

- Stochastic Oscillator(SO)

9.2.6 Allow for at least 20 market participants

- Requirement Prioritization: Critical

- Requirement Source: Client

- Requirement Level: High

Enable multiple market participants to be active in the market and allow for concurrent bids and offers to be placed. With this functionality, we can see how the market performs under certain(e.g. conditions when a stock is trending or when the market is liquid).

Participants will use the trading strategies implemented in order to compete with each other in the financial market to see which ones perform better or not in terms of profit and loss.
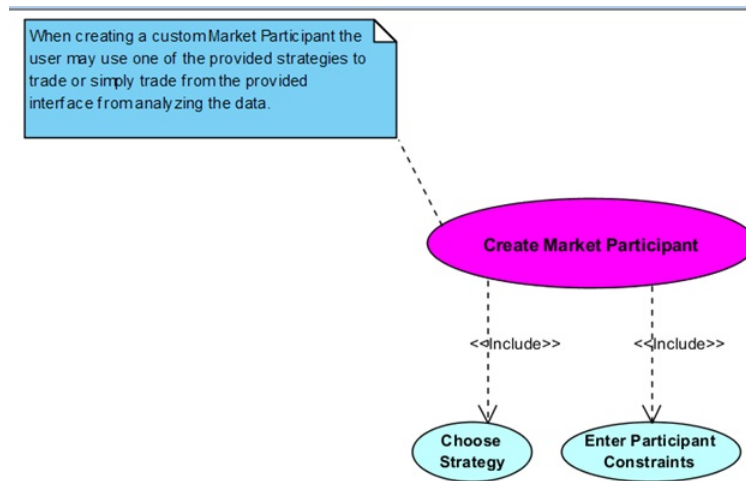

Figure 12: A use case for creating a market participant.

9.2.7   Algorithmic trading entities(strategies)

- Requirement Prioritization: Critical

- Requirement Source: Client

- Requirement Level: High

Since there are multiple participants who will be competing in the market, we need to see how trading stategies perform in different market conditions.

Each entity will need to do the following:

- Process market data and trading events from the matching engine

- Decide the entry or exit point into the market using data from the matching engine.

These are the different number of trading strategies that would be implemented by the developers. Different variations of trading strategies will be implemented(with the help of technical indicators). Each trading strategy will be able to process market data in order to use the information generated from the process and will generate a trading event(either a bid or offer). The trading algorithms will be able to remove bids and offers.

So far 5 trading strategies have been implemented, namely: Crossover, Moving Average Crossover, Price EMA Crossover, Price SMA Crossover and the Moving Average Envelope.

9.2.8 Generate trading events

- Requirement Prioritization: Critical

- Requirement Source: Client

- Requirement Level: High

The system should be able to generate bids and offers. The participant will have a mechanism(trading strategies) to decide how much to offer or bid and the quantity of shares. This requirement is strongly linked with previous requirement.

Figure 13 shows an activity diagram of how an order would be placed in the Financial Market Simulator.
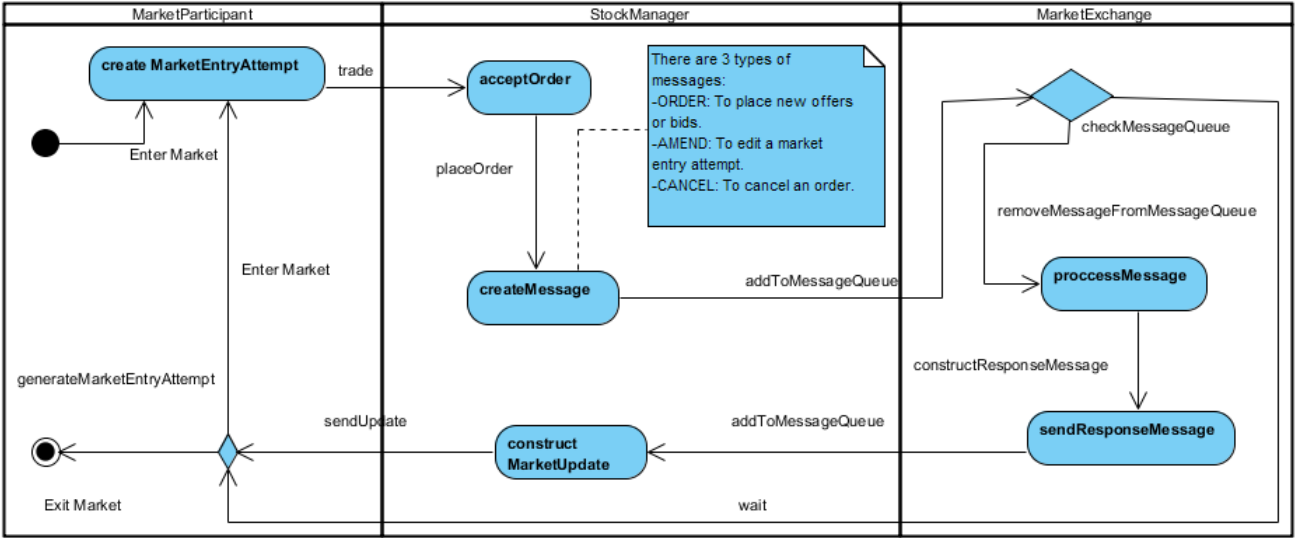
Figure 13: An activity diagram for placing an order.

9.2.9 A mechanism for delivering messages about market events

- Requirement Prioritization: Critical

- Requirement Source: Client

- Requirement Level: High

Messages about the last traded price need to be delivered to the other relevant market participants in order for them to have the latest information on whats happening in the market. The stock manager will deliver the messages to market participants and other components of the system that will need the latest data for computation purposes. All the market participants which are trading a particular instrument will be informed of what the other participants are doing.

9.2.10 Simulate the financial market

- Requirement Prioritization: Critical

- Requirement Source: Client

- Requirement Level: High

The system needs to be able to simulate(without input from the user) in order to evaluate the trading algorithms(strategies) and market participants performances. The range for the prices and shares will be specified so that the financial market is controlled and trends can be spotted.

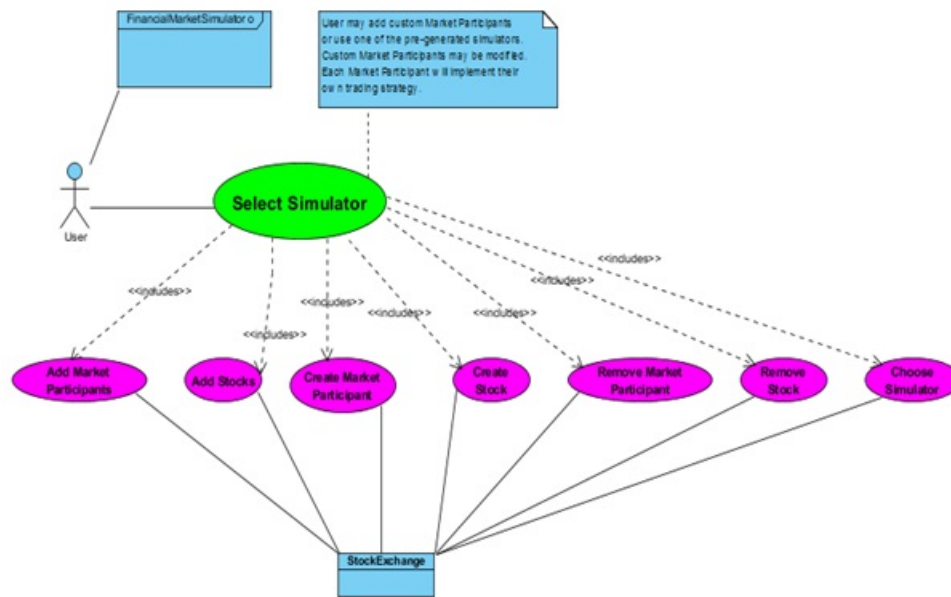A user can manually envoke market events via the user interface into the financial market.

Figure 14: A use case for showing how the simulator will be set up.

9.2.11 Provide a concurrent message queue between market participants and the matching engine

- Requirement Prioritization: Important

- Requirement Source: Client
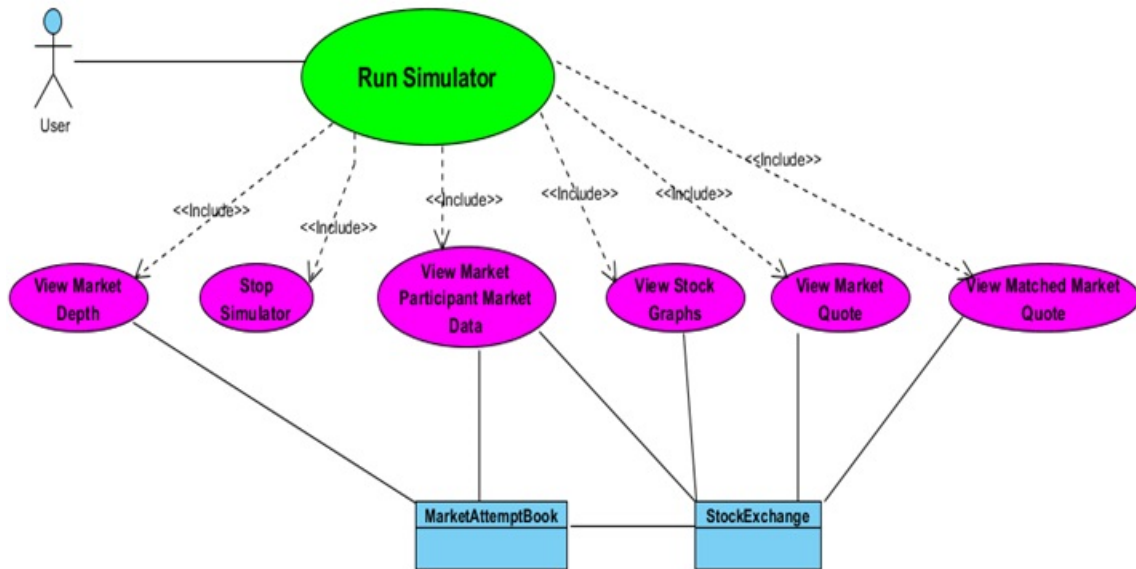
- Requirement Level: Medium

Figure 15: A use case for showing what services will be available once the simulator is running.

In order to have market participants decoupled from the matching engine, the participants will communicate with the matching engine using a message queue interface. All the participants would do is place an order onto the message queue and the matching engine will be responsible for taking the message and process the request.

9.2.12 Allow the matching engine to broadcast market updates to all market participants using a message queue

- Requirement Prioritization: Important

- Requirement Source: Client

- Requirement Level: Medium

The matching engine will be interacting with the message queue interface. After every state(a new order or a trade has occured or an order has been edited) change the matching engine will broadcast this change to all the market particpants by placing a message on the meassage queue.

9.2.13 Load market participants from persistent storage

- Requirement Prioritization: Nice to have

- Requirement Source: Grape Team

- Requirement Level: Medium

Enable users to load market participants from persistent storage with their preferences as it can get tiring to enter a new market everytime one runs the system.

9.2.14 Provide graphical representation of the data generated

- Requirement Prioritization: Nice to have

- Requirement Source: Grape Team

- Requirement Level: Medium

Allow users who use the system to be able to see graphs(e.g. line graphs) of the calculated data(i.e technical iindicators) in order to see how the financial market and individual instruments are performing.

9.3 Domain Objects

See the last page.

The strategy design pattern will be used for implementing the different trading algorithms to provide pluggability. And the observer design pattern will be used for the event handling of delivering messages about the activity in the financial market.

Figure 16: Class diagram of the system so far

28

10 Glossary

- **Order**: An instruction from a participant to either put in an offer or a bid.(Note-The term market entry attempt has the same meaning in this context)

- **Market data**: Data reflecting current trading information to include pricing and volume and other additional information related to the trade.

- **Match**: When an offer and a bid have the same price.

- **Trading Event** Entering the market(placing an bid) or exiting the market(placing an offer).

- **Volume**: A measure of activity i.e. the total number of shares one(a participant) has.