Software Architecture Specification Document

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

Client: Cortical Systems

Group:

Members:

Daniel Makgonta 12147100
Moeletji Semenya 12349136
Madimetja Shika 12127877

Publication Date: 21 May 2014

Version: 0.0

Change Log

Date	Name	Reason	Version
13/05/2014	Moeletji	Creation	0.0
16/05/2014	Moeletji	Adding functional requirements	0.0
19/05/2014	Daniel	Adding architectural requrements	0.0
21/05/2014	Madimetja	Editing	0.0

Contents

1 Architectural requirements

1.1 Architectural scope

- The Java programming language will be used for main process execution and C/C++ languages will be used as dll references within the Java classes to
- JDBC and MySQL are the database that will be used for the persistent infrastructure.
- All reporting functions will be use JReport(Java Report).
- Graphs will be displayed using Java libraries Graphics2D, AWT and Swings.

1.2 Quality requirements

1.2.1 Security

- Only system administrators may view the statistical analysis of the system.
- Only system administrators may tweak trading algorithms or matching engine to follow current trends within the stock market.
- Only registered users may be able to buy/sell shares on the system.
- 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.

1.2.2 Auditability

- One should be able to query any entity 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.

1.2.3 Testability

- All services provided by the system must be testable with unit tests, that 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. JUnitEE will be used for unit testing.

1.2.4 Usability

- All registered users(participants) of the system will be able to buy and sell shares concurrently.
- 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.
- The user interface will be simplified in order to speed up the amount of trading and increase concurrency.

1.2.5 Scalability

- The system will be able to use multiple competing trading algorithms and allow more algorithms to be plugged in at a later stage by using inheritance, polymorphism and the Template design pattern to achieve this.

- The system will be able to concurrently trade with 20-40 traders effectively and accurately according to the business rules of the system.
- The system will be implemented using JAVA EEs multi-tier layer architecture where the top layer will be applications that can be added and removed for presenting the data from the EIS tier layer and being processed by the business tier layer (where the matching engine is situated).

1.2.6 Performance

- The matching engine will return matches in less than 1 second.
- Reporting of the market will be displayed in less than 10 seconds.
- The buyers and sellers will be able the concurrently interact with the market simulation and this would take less than 1 second.

1.3 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 a web browser through a simple user interface. The system must be accessible from any of the widely used web browsers including all recent versions of Mozilla Firefox, Google Chrome, Apple Safari and Microsoft Internet Explorer.
- From Android mobile application (at a later stage).

1.4 Architectural constraints

- System uses the JAVA EE as its enterprise software .
- System uses Java Server Pages for Web application framework.
- System displays HTML5 web pages.

- HTTP/HTTPS, SMTP and POP3 protocols for emails and communication between wen server and web client.
- System uses JDBC and MySQL as its Databases.
- Operating systems: Windows, Linux, Android(mobile).

1.5 Architectural patterns/styles

The enterprise software of choice Java EE uses a 4 tier layering architecture.

Layer 1 Fundamental Services: JDBC database for a persistent infrastructure **Layer 2** Business Domain Tier: Application Server, Java EE EJB **Layer 3** Java Servlets/JSP **Layer 4** Client Tier: Presentation Tier: Thin Client HTML5 pages on supported Web Browsers, Java and XML on Android capable devices

4-tier architecture pattern allows separation of concerns and allows one service provided from layer 2 to be displayed on different applications without each application getting only the abstracted data back.

Matching situated on layer 2 will manipulate the data in layer 4 and display the results using one of the applications on layer 4. Layer 3 will dynamically display the data on to the application.

1.6 Use of frameworks

Java Servlets is the web framework used to create dynamic web pages because the system runs in real time and requires it to quickly update the applications graphical interface

1.7 Technologies

Enterprise software: Java EE (enterprise software used to wrap all aspects of a complex system into one package).
Database: JDBC (Database).
Programming Languages: Java, JavaScript, JSP, C/C++.
Business logic: EJB (Entity Java Beans)

2 Functional requirements and application design

2.1 Introduction

This part of the document outlines the functional requirements of the *Financial market simulation* at the different levels of granularity.

2.2 Required functionality

2.2.1 Matching Engine

- Requirement Prioritization: Critical
- Requirement Source: Client
- Requirement Level: High

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.

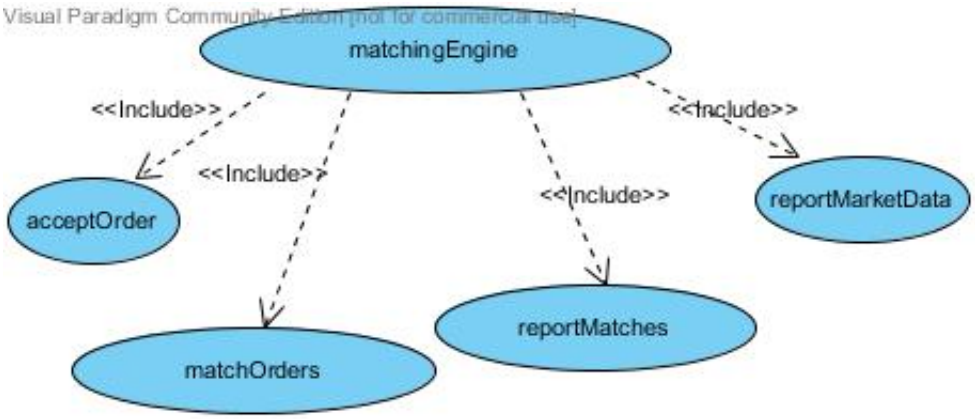

Figure 1: Use case for the matching engine

2.2.2 Allow for multiple 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).

2.2.3 Algorithmic trading entities

- 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.

2.2.4 User Interfaces

- Requirement Prioritization: Critical
- Requirement Source: Client
- Requirement Level: High

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.

2.3 Domain Objects

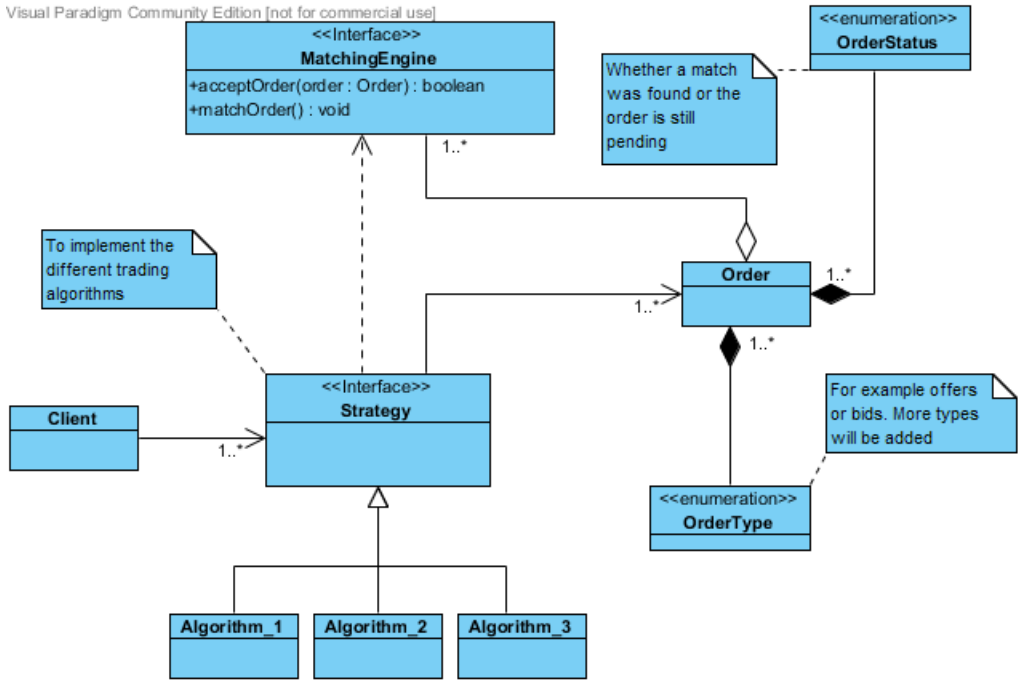

Figure 2: Core domain objects

The strategy design pattern will be used for implementing the different trading algorithms to provide pluggability.

3 Glossary

- **Order**: An instruction from a participant to either put in an offer or a bid.
- **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.