System design document for ASTRO

Table of Contents:

Version: 1.0

Date 2012-05-23

Author:
Mattias Markehed
Daniel Malmqvist
Erik Ramqvist
Kristian Sällberg

This version overrides all previous versions.

1 Introduction

Our project has been to create a stock trading robot framework.

1.1 Design goals

The design will be divided into many separate parts, making it easy to replace for instance the parser or the Buy/Sell API.

The GUI will be loosely coupled with the underlying model.. The design must also be testable with junit.

The application will use a couple of different highly tested libraries.

The application should have few circular dependencies.

1.2 Definitions, acronyms and abbreviations

Algorithm System - Will load algorithms for a given portfolio.
Avanza parser - A parser to fetch and parse stockdata from the stocktrader Avanza.
Buy/Sell API - Our self defined API for buying and selling securities internally. Used by algorithms loaded into the system.
GUI - graphical user interface
Java - platform independent programming language.
JFreeChart - Library that provides graphs for the Swing environment
Joda Time - A library to used to replace java's Date class.
JPA (openJPA 2.0) - Java Persistence API a way to translate a database into java classes.
MVC - Model View Control, a software architectural pattern for isolating the View etc.
MYSQL - A free database system.
Parser - In this project, the parser is a stand alone program taking in data from a specific source and inserting that data into our Price List Database. Users will define their own parsers for data sources not supported by us.
Portfolio Database - The database design that is used in the Portfolio System.
Portfolio System - A system for creating a portfolio that will be coupled with one or zero algorithms, will also give the possibility to manually buy/sell stocks.
Price List Database - Database that keeps stock prices for each minute (or other time unit) the parser has been running.
SQL - Structured query language. Used for communication with the database.
Stock/Security - Part ownership of a company.

2 System design

2.1 Overview

In this section we explain the overall design choices.

2.2 Software decomposition

2.2.1 General

Package diagram. For each package an UML class diagram in
appendix

Application ASTRo Harvester
Harvester - Fetches stock data from datasource and saves it to database.
Database - Stores and serves the data received from a data source.
Parser - Parses data received from datasource.
Model - Represents the stock data.

Application ASTRo

Portfolio
Portfolio - Holds information about economic data and what algorithm to use.
PortolioHandler - Handles portfolio creation, and algorithm loading.
PortfolioSettings - Settings for the running portfolio
RobotScheduler - Starts algorithms at given intervals or when new data is available.
TraderSimulatior - handles buying and selling this has no connection to the real stock world.
ITrader - Interface for buying and selling

Database
JPAHelper - used to setup connection to the database, also used as our link to the database.

Algorithms
PluginAlgortihmLoader - Loads algorithms for use in portfolio
Algorithms - Algorithm that should be run by portfolio

Generics
Pair - Holds a tuple of the type pair.
Log - Our logging class

GUI
Graph - Handles Graph for showing different kind of graph.
Harvester - Handles the harvester.
Portfolio - Shows the portfolio information.
PortfolioHistory - The history for a given portfolio.
PortfolioWizard - Wizard for creating a new portfolio.

Simulation - Gives the user options on how to run a simulation.
Simulation Results - Displays results from a simulation.
Stock - Shows stock data.

MVC
IController - Our interface for all controllers
IView - Interface for all of our views

2.2.2 Dependency analysis

We didn't think about dependencies from the start and we tried to untangle everything halfway into the project. If we thought about it from the beginning we would probably have had less problems with it.

We created our own mvc code and applied interfaces to model, view and controller. This solved a lot of problems we had with the circular dependencies.

2.3 Concurrency issues

We will have many different threads for many functions. For example the parser will run in a separate thread. The GUI will automatically run in separate threads, using Java Swing. So the GUI will feel very responsive.

We might still have some concurrency problems. We have tried solving as many of them as we could, for example by synchronizing the critical sections of code.

The database has built-in concurrency properties. This wasn't on by default and gave us a lot of problems before we realized it.

2.4 Persistent data management

All our data will be stored using OpenJPA that in turn stores data in a backend database. The JPA system will provide memcache like functions, storing a buffer model of the database in the RAM memory for quick access to data. At certain times, the buffer will be synchronized to the hard drive located database.

2.5 Access control and security

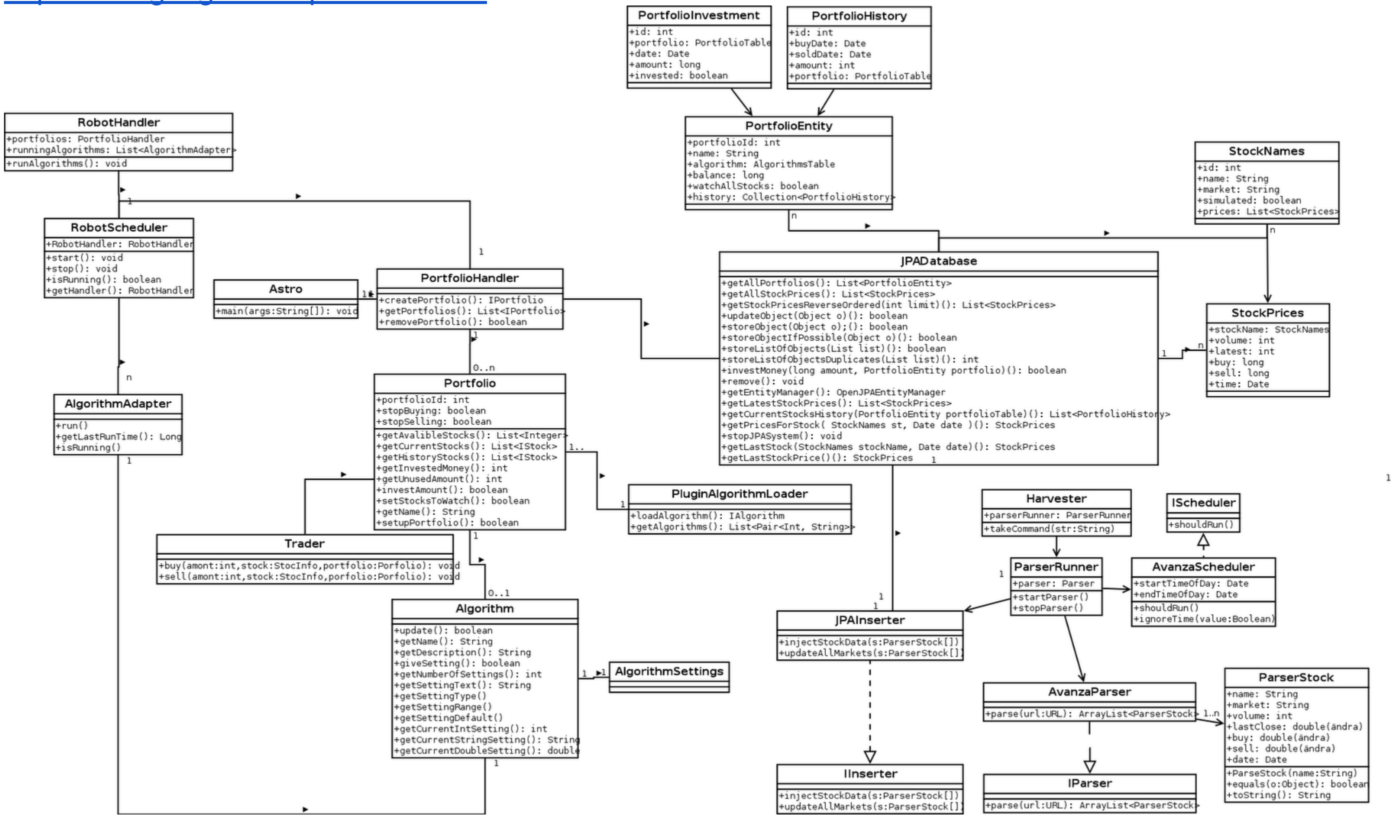
We have no access to the users data. Nor do we have any control who uses this application or of which purpose. Our own database server has “password” protection so unauthorized users can’t modify the data.

2.6 Boundary conditions

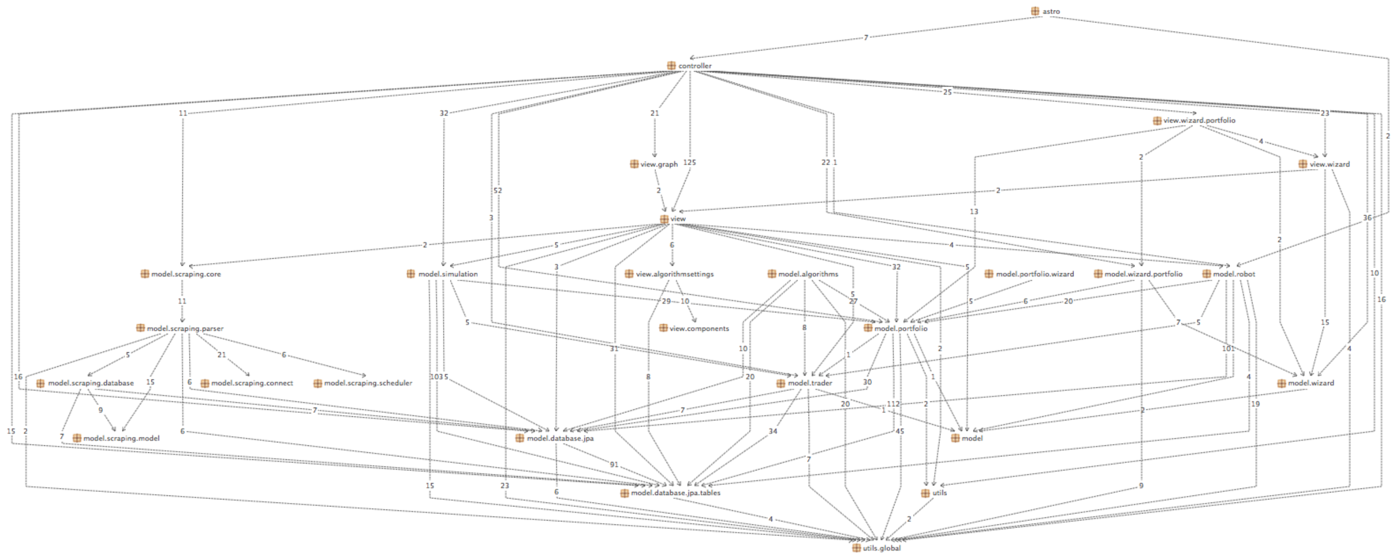
Application launched and exited as normal desktop application

3 References

http://openjpa.apache.org
http://www.jfree.org/jfreechart/
http://code.google.com/p/stock-robot/


PortfolioInvestment
+id: int
+portfolio: PortfolioTable
+date: Date
+amount: long
+invested: boolean
PortfolioHistory
+id: int
+buyDate: Date
+soldDate: Date
+amount: int
+portfolio: PortfolioTable
PortfolioEntity
+portfolioId: int
+name: String
+algorithm: AlgorithmsTable
+balance: long
+watchAllStocks: boolean
+history: Collection<PortfolioHistory>
StockNames
+id: int
+name: String
+market: String
+simulated: boolean
+prices: List<StockPrices>
RobotHandler
+portfolios: PortfolioHandler
+runningAlgorithms: List<AlgorithmAdapter>
+runAlgorithms(): void
RobotScheduler
+RobotHandler: RobotHandler
+start(): void
+stop(): void
+isRunning(): boolean
+getHandler(): RobotHandler
Astro
+main(args:String[]): void
PortfolioHandler
+createPortfolio(): IPortfolio
+getPortfolios(): List<IPortfolio>
+removePortfolio(): boolean
JPADatabase
+getAllPortfolios(): List<PortfolioEntity>
+getAllStockPrices(): List<StockPrices>
+getStockPricesReverseOrdered(int limit)(): List<StockPrices>
+updateObject(Object o)(): boolean
+storeObject(Object o);(): boolean
+storeObjectIfPossible(Object o)(): boolean
+storeListOfObjects(List list)(): boolean
+storeListOfObjectsDuplicates(List list)(): int
+investMoney(long amount, PortfolioEntity portfolio)(): boolean
+remove(): void
+getEntityManager(): OpenJPAEntityManager
+getLatestStockPrices(): List<StockPrices>
+getCurrentStocksHistory(PortfolioEntity portfolioTable)(): List<PortfolioHistory>
+getPricesForStock(StockNames st, Date date)(): StockPrices
+stopJPASystem(): void
+getLastStock(StockNames stockName, Date date)(): StockPrices
+getLastStockPrice()(): StockPrices
StockPrices
+stockName: StockNames
+volume: int
+latest: int
+buy: long
+sell: long
+time: Date
AlgorithmAdapter
+run()
+getLastRunTime(): Long
+isRunning()
Portfolio
+portfolioId: int
+stopBuying: boolean
+stopSelling: boolean
+getAvalibleStocks(): List<Integer>
+getCurrentStocks(): List<IStock>
+getHistoryStocks(): List<IStock>
+getInvestedMoney(): int
+getUnusedAmount(): int
+investAmount(): boolean
+setStocksToWatch(): boolean
+getName(): String
+setupPortfolio(): boolean
PluginAlgorithmLoader
+loadAlgorithm(): IAlgorithm
+getAlgorithms(): List<Pair<Int, String>>
Harvester
+parserRunner: ParserRunner
+takeCommand(str:String)
IScheduler
+shouldRun()
Trader
+buy(amont:int,stock:StocInfo,portfolio:Porfolio): void
+sell(amont:int,stock:StocInfo,porfolio:Porfolio): void
ParserRunner
+parser: Parser
+startParser()
+stopParser()
AvanzaScheduler
+startTimeOfDay: Date
+endTimeOfDay: Date
+shouldRun()
+ignoreTime(value:Boolean)
Algorithm
+update(): boolean
+getName(): String
+getDescription(): String
+giveSetting(): boolean
+getNumberOfSettings(): int
+getSettingText(): String
+getSettingType()
+getSettingRange()
+getSettingDefault()
+getCurrentIntSetting(): int
+getCurrentStringSetting(): String
+getCurrentDoubleSetting(): double
JPAInserter
+injectStockData(s:ParserStock[])
+updateAllMarkets(s:ParserStock[])
AlgorithmSettings
AvanzaParser
+parse(url:URL): ArrayList<ParserStock>
ParserStock
+name: String
+market: String
+volume: int
+lastClose: double(ändra)
+buy: double(ändra)
+sell: double(ändra)
+date: Date
+ParseStock(name:String)
+equals(o:Object): boolean
+toString(): String
IInserter
+injectStockData(s:ParserStock[])
+updateAllMarkets(s:ParserStock[])
IParser
+parse(url:URL): ArrayList<ParserStock>

Stan: Screenshot of current state



Domain Model: http://stock-robot.googlecode.com/files/domainmodel.png
Design Model: http://stock-robot.googlecode.com/files/designmodel.png