Algo-trading market client LLD

Terminology

(Copied from the market server HLD)

**Commodity**

A traded resource, can be held by the user or by the other actors and can be sold/bought.

**Funds**

Money being held by the user, used to purchase commodities.

**Request**

An order or a query given to the market by the user. For example: A ***sell request*** is an order given to the market telling it to sell a commodity for funds. A ***buy request*** is an order given to the market telling it to buy a commodity using funds.

Sell request

A sell request is a request that when posted to the server shows the will of a user to sell some amount of a commodity for a certain price. Posting sell requests automatically reduces the amount of available commodity the user has.

A sell request contains a commodity id, amount and ask price. If the user has enough resources to make the sell, the market replies with a sell ID.

Buy request

A buy request is a request that when posted to the server shows the will of a user to buy some amount of a commodity for a certain price. Posting buy requests automatically reduces the amount of available funds the user has.

A buy request contains a commodity id, amount and bid price. If the user has enough resources to make the sell, the market replies with a buy ID.

Cancel request

A cancel request contains an ID of the sell/buy request to cancel and refunds any commodities or funds invested in that request.

Query sell/buy request

This request returns information about the buy/sell request with the given id.

The query request contains an ID of the sell/buy and returns the current information about the sell/buy

Query user request

The query request is empty (besides authentication) and returns sell/buy IDs, and for each commodity, how many the user owns (zero if the user doesn’t own it)

Query market request

This request returns information about the market state of a certain commodity.

The query contains a commodity ID and returns the best ask price and best bid price for that commodity in the market.

**Ask**

The price on a sell request existing in the market.

**Ask price**

The lowest ask for a certain commodity out of all of the sell requests.

**Bid**

The price on a buy request existing in the market.

**Bid price**

The highest bid for a certain commodity out of all of the buy requests.

3 Tier Design

The application will follow the 3-tier design structure:

* Presentation Layer
* Business Layer
* Data Layer

Each layer will have its own namespace or project. A **special PL-BL namespace** will contain shared interfaces and classes that are crucial for both presentation layer and business layer and their communication.

Presentation Layer

**GUI**

**The MainWindow is divided into 4 categories:**  
Market action:  
Includes the buy, sell and cancel queries.  
  
The buy and sell forms looks visually the same and preform each the buy or the sell query. We added a feature allowing the user to calculate the total amount of money he will spend or get when he buy or sell.  
The cancel form allowing the user to cancel a request by the ID of the request.  
  
Info:  
Include the request status, commodity status and user status queries.  
It is also include the history button and a new feature we added "test connection".  
  
The request status form allowing the user to get the a request status by the ID of the request.  
The commodity status form allowing the user to get the a commodity status by number status (the current ask and bid).  
The My status form allowing the user to get his current status (how much he got from each commodity, his funds and his requests ID's).  
The history form allowing the user to watch the history of his actions. The user can choose between show by days or dates.   
The test connection button allowing the user to check the most common and also easy to notice problem by himself.  
  
Default AMA:  
Include the run default AMA and current logics buttons.  
The run default AMA is the button that runs the AMA.   
The Current Logics button reveals to the user the current logics and he can compare it to his own.

User AMA:  
Include run user AMA, add rule, Current logics and clear all rules buttons.  
The run user AMA is the button that runs the AMA.  
The Current Logics button reveals to the user the current logics he set.  
The add rule button allow the user to add a rule by the ask of the commodity.  
The clear all rules button clear the rules the user set.

Business Layer

**IMarketResponse Interface**

This object represents a response from the market. Any message received, be it a request id or a market exception will implement this interface. The Presentation layer prints out a string from this class to the user.

**MResponse Class**

Contains several implementations for the **IMarketResponse** interface. Each implementation corresponds with each different response.

**Request Class**

A class representing a request sent by the user. These objects are sent from the Presentation Layer (via GUI) to the Business Layer (via the **InterperatorPB** class)

**Rtype**

Simply an Enum for the various types of requests and responses.

**InterperatorPB Static Class**

This class receives requests from the Presentation Layer and passes them to the Communicator class in the Business layer. It then returns an object signifying the response from the server.

This is the main pipeline for PL-BL communication.

**ICommunicator Interface**

This interface handles the communication to the server using JSON, both input and output.

The main implementation of the interface is the **Communicator** class. For testing, we use the **TestMarketCommunicator** class which handles request to the Test Market Server.

**Autonomous Market Agent**

This class hold a queue of Logic Processes and handles the activation of these processes following restrictions from the server (ie: no more than 10 requests every 10 seconds).

There are 2 extending class: **Default Market Agent** and **User Market Agent**

Logic Process

This object has an inner list of logics for each step in the process.

Each time the **LogicProcess** is run – it will activate the next inner logic in the list.

For example, the list of a Buy Process:

1. Verify that there is no existing request from this process

2. Verify that the commodity's ask price is larger or equal than the specificed price for this process

3. Send buy request

If an inner logic is successful (ie: the price of the commodity is right), it will instruct the AMA to re-insert it at the top of the queue. This is to maximize the chances of seizing an opportunity in buying or selling the commodity by acting quickly. If an inner logic is unsuccessful (ie: a request already exists), it will instruct the AMA to re-insert it at the end of the queue.

Inner Logic

This is the inner step in the list of the Logic Process.

Examples of inner steps –

* Verify if a request from the process already exists
* Verify if the ASK is lower than the threshold
* Send Buy requests

All inner logics must receive a pointer to the Logic Process to properly advanced the entire process to the next step. If a step has failed, the process should try the same step again later. If the process is successful – the process should continue to the next step.

Default Market Agent

This AMA is initialized with a pre-set queue of rules. They are designed to be extra safe and are as followed:

1. If the bid on a commodity is more than 16 – Sell.
2. If the ask on a commodity is less than 7 – Buy.

User Market Agent

This AMA is initialized with an empty queue that will be populated by the user via the GUI. The user is allowed to insert up to 30 rules under which the **User AMA** will buy and sell commodities.

Data Layer

**History**

The history is written into History folder – history.txt.

It uses Log4net to write the history logs into the file using a dedicated appender.

To read the file content the user is given 2 options:

Choose how many backward days or choose a specific range.

**Log**

In order to write all the app log, we use Log4net.

There is Rolling file appender that create a log text file. There is a separated log file for each month.