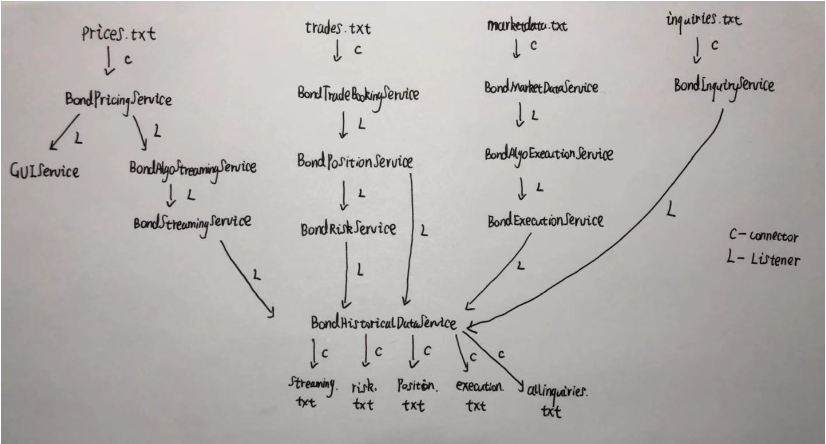
**Software Engineering final project readme**

**Bo Pang**

This project follows the frame of the sample code and can work as a basic trading system. In this doc, I will show the whole picture of this project and then explain what I did in each file. I will talk about those functions that were not in the sample code in detail.

The whole picture of the project is like this, except that I need to register another listener from BondTradeBookingService to BondExecutionService. I found this picture on the forum.

Each service is implemented with in a ‘hpp’ file. The name of the file can indicate the service within it.

**Source.cpp**

This is the file where I put the main function. Before the main function there are also some helping functions.

Create\_prices: Create ‘price.txt’, I use uniform random variables to created price data and transform them into standard strings. To make the system easier to read, instead of CUSIP, I directly use their duration to index the bond. Also for simplicity, I choose a virtual coupon and maturity date.

timestamp: create a timestamp as a string.

Create\_trades: Create ‘trades.txt’, use the same method as in create\_prices

Create\_market: Create ‘market.txt’

Create\_inquiry: Create ‘inquiry.txt

Main: instantiate every service, connect each service with listeners as the picture shows, and read in data through connectors.

**Pricingservice.hpp**

Function convert: convert the strings from files into prices as double to form the price conveniently.

Class price: I add a function Price\_to\_string to convert a Price data into string, to persist data easily. This class gives a price of a bond.

Class PricingService:

Private member: a map to save Price, a connector to subscribe data and a vector of listeners to give a signal of changes.

Key Public member: OnMessage: once the service receive a data, record it in the map and tell the listeners they should take actions.

Class PricingConnector:

Private member: a pricingservice to give data to

Key Public member: Subscribe: read the file line by line, convert each line to a Price, and send the price to the service inside the class.

**Guiservice.hpp**

Class GuiListener: A class of listener from Guiservice listening to Pricingservice.

Private member: a guiservice to give instructions to

Public member: ProcessAdd: Once it hears a change in data, it will tell the service to call OnMessage method.

Class GuiService:

Private member: a map to save data, a connector to publish data, a listener to listen to pricingservice and throttle.

Key public member: OnMessage: Control the speed of pricing stream and show the first 100 updates.

Class GuiConnector:

Key public member: Publish: show the data in the “gui.txt” file.

**Streamingalgoservice.hpp**

Class StreamingAlgoService:

Private member: a map to store data, a vector of listeners and a listener to listen to pricingstreamservice

Key public member: PublishPrice: once this service get a pricestream, then it will tell the listeners to take actions.

Class StreamingAlgoListener: Listen to pricingstreamservice.

Key public member: ProcessAdd: listen to pricingstreamservice and get the price data, transform it into pricestream data and pass it to streamingalgoservice.

**Streamingservice.hpp**

Class StreamingService: Listen to streamingalgoservice, save pricestream data to this service, and give listener signals when it receive a new data.

Class StreamingListener: Listener for streamingservice listening to streamingalgoservice.

**Historicaldataservice.hpp**

In this file, I defined 6 HistoricalDataService of 5 kinds (risk needs two). They are HistoricalDataServiceStream, HistoricalDataServicePosition, HistoricalDataServiceRisk, HistoricalDataServiceRiskB, HistoricalDataServiceExecution and HistoricalDataServiceInquiry.

They are implemented almost the same way. They each will register a listener to another service and listen to data updates. Once they hear data, they will record the data in corresponding files.

Because we need to record PV01 for bond and also for bucketed bond, I defined two services for these two cases.

Other files are implemented in the same way as this branch, so I will explain with less detail.

**Tradebookingservice.hpp**

In this file, I defined TradeBookingService and TradeBookingConnector. I use the connector to read file from the file and then pass the data to tradebookingservice. This service will save Trade data. Besides, I also defined TradeToOrderListener for this service to listen to executionservice to get ExecutionOrder data, transform it into Trade data and pass it to TradeBookingService.

**Positionservice.hpp**

In this file, I defined PositionService and PositionToBookingListener. The listener will listen to tradebookingservice and get new trade data , and pass the data to positionservice. The positionservice will update the position data stored inside it with the trade data. After updating the position data, the service will pass the new position to its listeners (riskservice and historicaldataservice).

**Riskservice.hpp**

In this file, I defined RiskService and RiskToPositionListener. The service will update the risk of the position once a new positiondata is passed in. Then, this service will pass the newly changed data to the historicaldataservice.

**Marketdataservice.hpp**

In this file, I defined MarketDataService and MarketConnector. The connector reads data from the file and transform the string into OrderBook data and pass it to the service. The service will store the orderbook data and tell the listeners (executionalgoservice) what to do.

**executionAlgoservice.hpp**

In this file, I defined ExecutionAlgoService and ExecutionAlgoListener. The listener will listen to marketdataservice for OrderBook data, if the requirements are reached, then it will transform the Order Book data into an ExecutionOrder data and pass it to executionalgoservice. The service will save this data and pass it to ExecutionService through a listener.

**Executionservice.hpp**

In this file, I defined ExecutionService and ExecutionListener. The listener will listen to executionAlgoservice for ExecutionOrder data. The service will store the executionOrder data and pass it to TradeBookingservice to book a trade. Also it will pass the data to historicaldataservice to persist the data.

**Inquiryservice.hpp**

In this file, I defined InquiryService and InquiryConnector. The connector will read the data from the file and transform it into inquiry data and send it to the service. The service will store the inquiry data and process the data depending on the inquiry states.