Project Proposal

Project Description

In order to give the customers, the possibility to invest in an appropriate well-balanced portfolio of risk-free and risky assets, the investment strategy of CPPI (Black & Perold, 1992) should be implemented by using the existing IT infrastructure[1].
The infrastructure provides interfaces for the underlying financial transaction system and stock price system, such that the new software projects' scope is on the algorithmic implementation and integration with the infrastructure.
Furthermore, it will be required to provide a suitable user interface for inserting the customers' preferences and respective settings.

Expected Outcome

The algorithm shall be embedded within a web application, accessible for financial service employees. The portfolio re-organization shall seamlessly be processed, without any human interaction after the starting parameters were set.

List of Features

1. The user interface should prompt the user[2] to insert the following values:
    a. Portfolio floor value F_T
    b. Multiplier m, which is used for multiplying the cushion and symbolizes risk appetite
    c. Maximum risky fraction b, which stands for the limit of risky investment
    d. Initial investment value (W_t, with t=0)
    e. Time horizon T of portfolio investments
2. The application logic has to consider the following mathematical model for evaluation and investment purposes in each point in time t:

a. Setting the floor value (F_T) for the end of the planning horizon (T)	F_T
b. Calculation of the present value of the floor (F_t) at time t	$F_t = \frac{F_T}{(1+R_t)^{T_{t,T}}}$

c. Calculation of the cushion at t (C_t) via investor's wealth (W_t) and max-operator	$C_t = \max(W_t - F_t; 0)$
d. Calculation of the risky exposure ($X_{r,t}$) via min-operator, multiplier (m) and maximum risky fraction (b)	$X_{r,t} = \min(m * C_t; b * W_t)$
e. Calculation of the riskless (riskfree) exposure ($X_{f,t}$)	$X_{f,t} = W_t - X_{r,t}$
f. Calculation of Total Shareholder Return (TSR) via stock prices (S_t)	$TSR_t = \frac{S_t}{S_{t-1}} - 1$
g. Calculation of investor's wealth in current period (used in c.)	$W_t = X_{r,t-1} * (1 + TSR_t) + X_{f,t-1} * (1 + R_0)^{1/365}$

3. If the user requests the current portfolio value, it has to be presented, together with all of the previously mentioned variables, in a tabular form in the frontend. (consider also a (partial) view for historical realizations of the variables and also live updates from currently optimized portfolio values[7])
4. The evaluation of the variables has to be done, when new information is received from the stock exchanges (S_t) via the internal messaging system[1].
5. During evaluation the algorithm has to update/calculate the variables in the same order as it is described by the above mentioned equations. The variables have to be saved in a persistent storage[3] for the respective customer[2].
6. The actual investment has to be performed by calling the existing financial transaction web services[1].

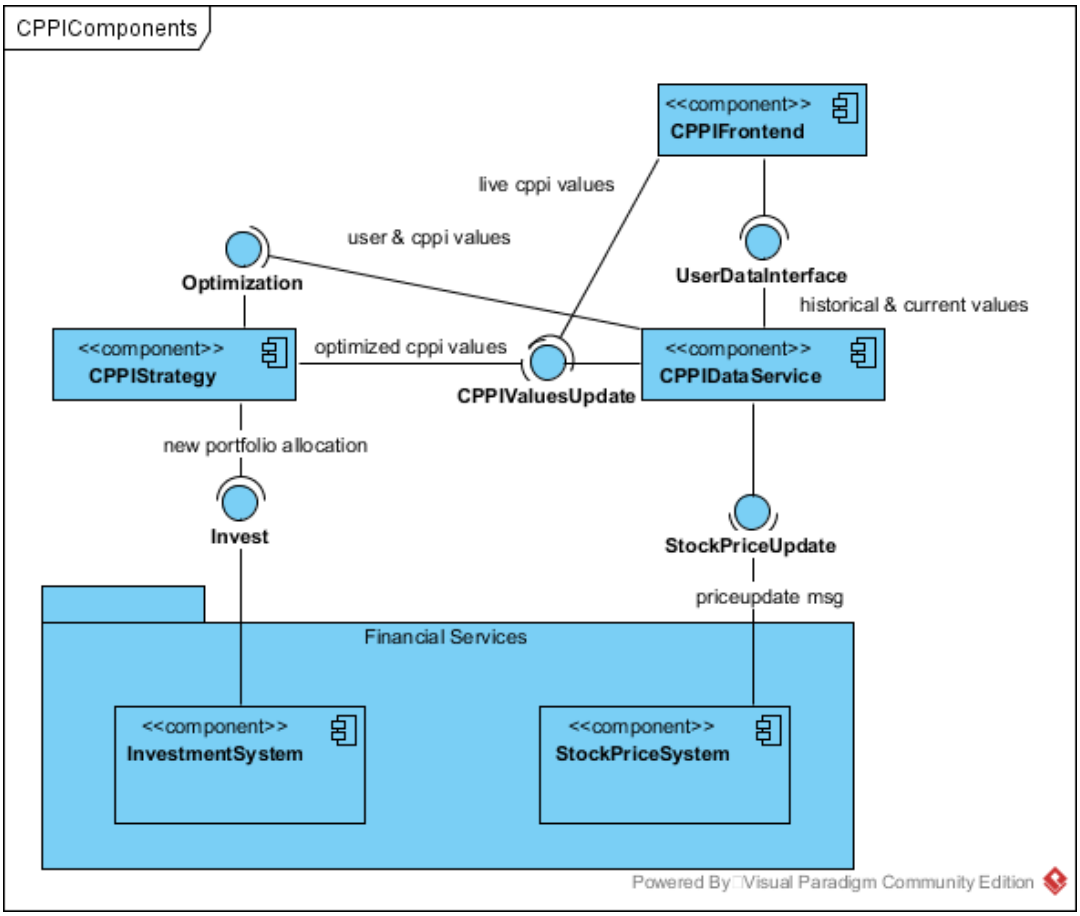
Initial High-level Design

As indicated in the component diagram below, the system consists of three main components: frontend (CPPIFrontend), CPPI algorithm implementation (CPPIStrategy) and a data service (CPPIDataService[4]). The package "Financial Services" can be considered as given, with an

investment interface and a stock update messaging. The data service persists the current and historical CPPI values for all clients. The CPPIFrontend will be realized as web interface, which retrieves values and provides the possibility of inserting initial planning parameters. CPPIStrategy starts its computation of the new portfolio after new CPPI values came in for processing the clients' portfolio investment **according to the PDCA framework**. The portfolio shift is actually performed by the InvestmentSystem component, accessible via the Invest interface.

The communication from StockPriceSystem to CPPIDataService and further to CPPIStrategy is performed **asynchronously**[5]. The CPPIFrontend is requesting the CPPIDataService **synchronously**[6], which also holds between CPPIStrategy and the InvestmentSystem and from CPPIStrategy to CPPIDataService. The CPPIFrontend is listening to live updates of the optimized CPPI values and receives the content as soon as it is available[7].

The overall system can be considered as distributed system, where each component is a stand-alone microservice[8] with a separate state (if necessary also a separate data storage), which is communicating over the network with other microservices.



Clarifications, Hints, and Simplifications for the Assignment

[1] The financial infrastructure (investment system + stock price system) **has to be developed** with the necessary functionality as **dummy** services.

- Investment System: simple web service, which receives the investment information (e.g., via REST interface), after the CPPI algorithm has performed one optimization iteration. The invested risky and riskless amount have to be logged to stdout.
- Stock Price System: simple service which generates the stock price updates every 3sec (better to be configurable). It creates a message with the price information and pushes it in a queue/topic of the used message broker (e.g., RabbitMQ, ActiveMQ), where it can be consumed by the actual CPPI implementation.
  We consider just one risky asset which realizes the following prices over time (for testing purposes): 100, 102, 105, 110, 115, 115, 115, 117, 120, 119, 116, 116, 116, 114, 118, 120, 125, 130, 123, 119, 116, 115, 114, 113, 120.

[2] Assume only **one** user (employee from the finance department who performs the transaction) and only **one** customer (person, whose money is invested by the employee) for this assignment prototype.

[3] You may use a relational in-memory database. (Nevertheless, you are also allowed to host a proper stand-alone instance)

[4] This service is in charge of managing the access to the data, stored in the database.

[5] Use asynchronous messaging techniques/technologies for this machine-to-machine communication (e.g., Message Oriented Middleware as RabbitMQ or ActiveMQ). You may also use this for communication purposes within components between independently operating objects as e.g., CheckProcess.java and ActProcess.java)
(see: http://www.enterpriseintegrationpatterns.com/patterns/messaging/Messaging.html and https://en.wikipedia.org/wiki/Message-oriented_middleware and asynchronous messaging

between microservices: https://capgemini.github.io/architecture/is-rest-best-microservices/#asynchronous-messaging)
(hint: you can make use of out-of-the-box ready RabbitMQ docker container from dockerhub.com: https://hub.docker.com/_/rabbitmq/ and integrate it in a spring boot application [microservice]: https://spring.io/guides/gs/messaging-rabbitmq/)

[6] e.g., use a typical REST-like HTTP communication or other synchronous remoting techniques. (see: https://capgemini.github.io/architecture/is-rest-best-microservices/#rest)

[7] Live updates of CPPI portfolio values, which are delivered to the user interface without requesting it, is an **optional** part for the assignment.

[8] see:

- concept of microservices: https://en.wikipedia.org/wiki/Microservices.
- How to build a microsevice in Java with Spring Boot, Spring Cloud, Eureka: https://github.com/paulc4/microservices-demo and the corresponding github repository: https://github.com/paulc4/microservices-demo (Note: if you don't want to, you don't need to introduce eureka as a registration server. It may be enough for assignment purposes to use the links in the next point)
- Suggested implementation variant considering the spring technology:
  - CPPIDataService, InvestmentSystem: https://spring.io/guides/gs/rest-service/
  - CPPIDataService: data access according to https://spring.io/guides/gs/accessing-data-rest/
  - CPPIStrategy, CPPIDataService, StockPriceSystem: https://spring.io/guides/gs/messaging-rabbitmq/ for asynchronously receiving and sending messages.
  - Client-side handling of REST requests: https://spring.io/guides/gs/consuming-rest/
- If you use spring you may also have a look at a more comprehensive spring boot tutorial: https://spring.io/guides/tutorials/bookmarks/