UDC: 004.42
Original scientific paper

A MODEL OF HETEROGENEOUS DISTRIBUTED SYSTEM FOR FOREIGN EXCHANGE PORTFOLIO ANALYSIS

Dragutin Kermek[1], Tomislav Jakupić[2], Neven Vrček[1]
[1]University of Zagreb,Faculty of Organization and Informatics, Varaždin, Croatia
{*dkermek | nvrcek*}*@foi.hr*
[2]Koprivnica 48000, Trg kralja Tomislava 8, Croatia
tjakupic@yahoo.com

Abstract: *The paper investigates the design of heterogeneous distributed system for foreign exchange portfolio analysis. The proposed model includes few separated and dislocated but connected parts through distributed mechanisms. Making system distributed brings new perspectives to performance busting where software based load balancer gets very important role. Desired system should spread over multiple, heterogeneous platforms in order to fulfil open platform goal. Building such a model incorporates different patterns from GOF design patterns, business patterns, J2EE patterns, integration patterns, enterprise patterns, distributed design patterns to Web services patterns. The authors try to find as much as possible appropriate patterns for planned tasks in order to capture best modelling and programming practices.*

Keywords: *foreign exchange portfolio analysis, distributed system, design pattern, refactoring, load balancer, Web service.*

1. INTRODUCTION

In the nature of Web application are unpredictable number of concurrent users and therefore unpredictable load peak moment and duration. Acceptable response time shortens from year to year [1, 2, 3] so new technologies and techniques must give an answer to response time issue. Modern information and communication technologies have significant impact on many software systems where Web applications are one of the most exposed types for common users. One of the most important issues concerning Web application is performance (latency, throughput, efficiency, scalability) [4] where expensive hardware and network infrastructures usually have been prime objects that enable feeding increased users' information "hunger". At the same time, software systems must be designed to take part in that solution to make the most of the hardware but also keeping in mind to cut down total expenses.

Building high-capability Internet-based system that must work under 24/7 scheme with constant fear of spreading over to many users that could shutdown the whole operation, is not an easy task. System architects have their cards in sleeves and according to Dyson and Longshaw [1] they are active-redundant elements, load-balanced elements, dedicated web and application servers, data replication, connection limitation, and resource pooling, just to mention few of them. Unfortunately, most of them are very expensive hardware equipment that only could afford big companies. Others must find different approach to solve performance issue.

Today is very common to plan a system based on cluster with many relatively cheep computers (personal or workstation) instead of very expensive mainframe/server. Integration on those computers into coherent system is done by some kind of distributed system hidden behind the façade put by web (or desktop) application.

The paper presents an object-oriented model of a distributed application based on two different technologies (RMI and Web services). When authors model new system they can quest for suitable well documented elements known as design patterns [5] that give them advantages to capture best known solutions to particular project parts and integrate them into new system.

The paper starts with a short description on the problem domain with architecture overview of distributed technologies. It is followed by analysis of three layers: business, data and presentation. Finally, some conclusions are given on the topic.

2. SHORT DESCRIPTION OF THE PROBLEM DOMAIN

New system should serve as a Web application that gives users a possibility to create one or more private portfolios. Each portfolio has its starting date, amount of money to invest in at least 3 foreign currencies. It is up to a user to define the starting date for a portfolio and initial division of money among chosen foreign currencies. Giving opportunity to a user to pick a date from the part has its reasons in analysing previous currencies trends and forecasting similar ones for the future in order to maximise earning. A user chooses when he/she will sell or buy some amount from his foreign currencies pool.

The functionality of proposed Web application can be divided between the main functionality, the analytical part of the application, and the background subsystem. The main functionality includes user related tasks, creation of a new currency portfolio, user's division of the starting balance among the foreign currencies in portfolio and management of transactions i.e. buying and selling. Analytical part includes various reports like the value of the foreign currency, portfolio state or profit through time interval and suggestion of the most profitable transaction in user defined time interval. The background subsystem acquires foreign currency rates from public data sources (banks) on scheduled daily base or on demand for time interval, and stores them in a local database.

Portfolio state report is one of the most time and resource challenging report offered by the Web application. This report includes daily value of deposit (which

is increased by selling some amount of one currency and decreased by buying new amount of other currency), daily values of every currency in portfolio and their sum. When user sets the interval and requests the portfolio state report, Web application would normally start processing given input data and after some amount of time the report would be presented to the user. The bigger the interval the greater is the impact on the time required for the processing and, of course, on the performance of the Web server hosting the Web application since the process involves considerable amount of calculations per day of interval. For these reasons more efficient solution is required and one option would be the distribution of work over the network.

One of the goals was to design a system that consists of subsystems with multiple applications that could be used for new projects. Typical example is foreign currencies rates data acquisition and serving subsystem. List of its tasks includes following: ease access to data, provide independence from original data sources, and offer new set of services based on single data request but what is more important on data collections request, too. For those tasks could be allocated two servers with separated functions where on of them serve as a database server for the subsystem. Due to relatively low need for processor power on data acquisition part it is very logical to use only one server for all tasks until response time is acceptable for other subsystems that rely on it.

2.1. ARCHITECTURE OVERVIEW OF DISTRIBUTED TECHNOLOGIES

The requirements analysis put some fixed points for the project and one of them is implementation in J2EE – *Java 2 Enterprise Edition* but without EJB – *Enterprise Java Beans*. To make a distributed system that fits the goal and the purpose set in the previous text the first step is to choose the proper approach. One of the simplest solutions here would be to use RMI – *Remote Method Invocation* [6], Java's simple, network-independent and portable solution for distributed computing. Moreover, since it is an all-Java solution authors can reuse code responsible for generating portfolio state reports they have developed with Web application. Distributed system is composed of the client part, in this case Web application with software-based load balancer and at least two servers. Servers have Java Virtual Machine installed and configured and RMI registry started so server-side Java application can instantiate and register remote object in RMI registry. Remote object implements the algorithm for generating portfolio state reports inside a method. A client, Web application, can with a help of server's RMI registry for obtaining reference to remote object, instantiate remote object and call this method in a way also defined by RMI specification.

Software-based load balancer has an important role here. Its primary objective is to divide big intervals for portfolio status report in equal parts and dispatch every part to a different server. Locally, every remote method call is performed in its own thread. When server returns result thread has finished and result can be picked up by load balancer. When all threads are done and results are picked up, they are merged, sorted and represented to the user. If the interval is not so big, for example less then 10 days, load balancer does not divide it or call remote methods to do the

job. It simply calls local method and whole job is done locally. This is primarily because all the network activities performed, when servers for portfolio state analysis are invoked, can result in an unnecessary overhead.

One can see that the Web application and the servers for custom analysis are pretty strongly tied together. The strong association between the two results from the fact that the analysis servers are direct extension of the Web application intended to free resources of the server that hosts the Web application and at the same time boost performance of the Web application by utilizing networking, distribution of work and parallelism.

Web service for foreign currencies is a part of the model which is also an aspect of distributed computing. It relieves the Web application and the analysis applications from the job of retrieving and managing the courses. This job is totally oblivious to the Web application because it provides the service with information about the course and expects from the Web service to return proper course value according to input information. Why use Web services here and not distributed computing with RMI? First of all, the job of retrieving and managing courses is not directly associated with the purpose of the Web application, so we can freely *outsource* it. Also, other clients could make use of such Web service because Web services are XML-based, and that directly means, they are *open* for everyone who needs them. Client of Web service can be any type of client at all, it only needs Web service's descriptor (WSDL) which is an XML document and then, using SOAP (also XML-based), can initiate communication with Web service. If we had used pure Java solution (RMI), then only Java clients could have used such *service*.

As mentioned before the job of the Web service is to retrieve courses and they can be retrieved from any source. It can be some on-line bank or some other non-XML service which has its own proprietary protocol for offering course information. Our Web service takes data from joint database that is updated by background subsystem for foreign currencies rates acquisition.

As we can see all components of the proposed model are platform-independent which one of the basic characteristics of Java platform is. The authors use it for development of Web application and its supporting distributed system, but also for the Web service. More important fact about the Web service is its mentioned availability to all kind of clients thanks to XML. Also, if authors are going to completely change the platform and implement service in some other programming language, this wouldn't influence clients too much. The only thing relevant to the outside world that could change would be the Web service's descriptor. Clients would then have to update the way they access service using SOAP according to the new WSDL and that's it.

The architecture of the model is shown in Figure 1. Internal data flows between subsystems are shown with thick double arrows.

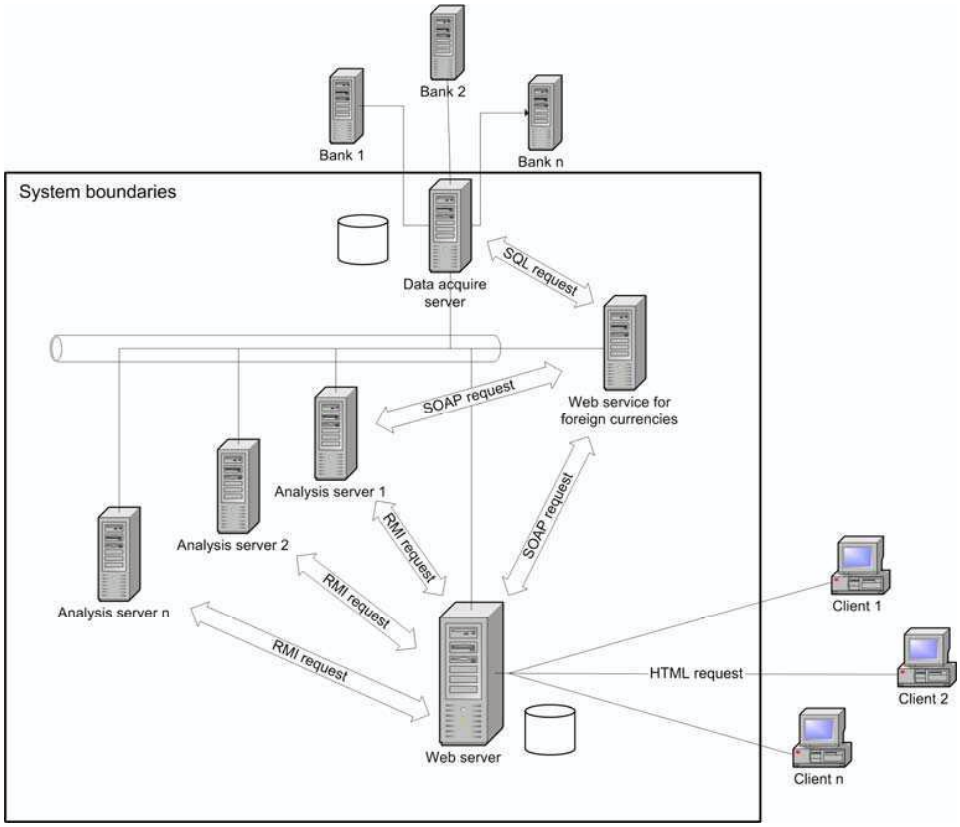

Figure 1. Architecture of the model

3. BUSINESS LAYER ANALYSIS

Buying and selling foreign currencies is a typical trading activity where exchange rates depend on changing market conditions on daily bases. According to Fowler's trading system for a bank [7] each trade is described by *Contract* pattern, a simplest kind of financial deal of buying some *Instrument* (foreign exchange rate) from another *Party* (usually bank is the problem domain). Eriksson and Penker [8] list a contract pattern but as a business pattern namely in the category resource and rule patterns.

A contract is useful for businesses that rely on tracing directions of deals, especially when deep analysis should be performed. This two-way pricing behaviour (one price where we buy and another when we sell) is captured by a *Quote*. Application's clients need to invest in at least three foreign currencies and they are captured by a *Portfolio*, which is a collection of contracts. The goal for a party on a financial market is gaining some profit. The profit in this problem domain is expressed by a difference in a current value of the portfolio (or at some date) and its initial value. The value of the portfolio is the sum of the values of underlying contracts that are priced according to some *Scenarios* as representation of the state of the market, either real or hypothetical. The problem domain could

have many different scenarios but authors are mainly oriented to hypothetical ones with historical data. Figure 2 shows simple class diagram of the domain model.

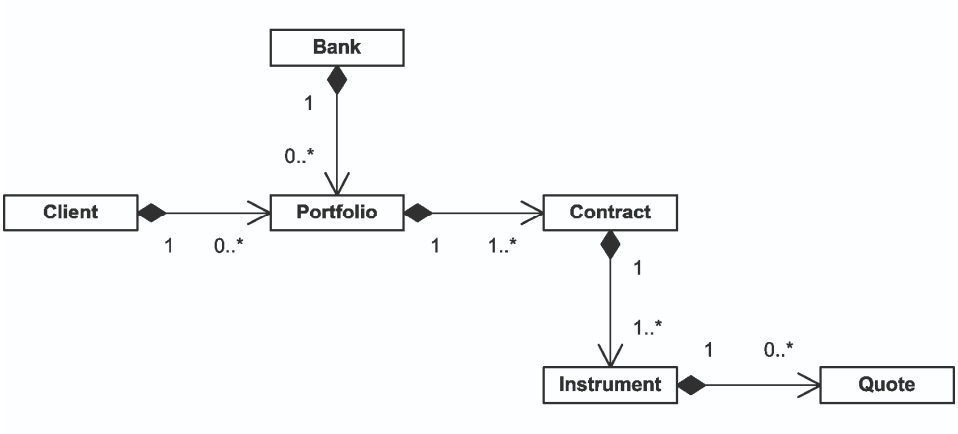

Figure 2. Class diagram of the domain model

Arlow and Neustadt [9] use *Money* archetype pattern as a focal point for trade affairs. It should be enough to say how complex it is that its class diagram consists of more than twenty classes. The authors extracted some elements from *Money* archetype pattern and used them in the domain model. *Currency*, *Locale*, *ISOCountryCode*, *ExchangeRate*, *CurrencyConverter* archetypes, and their subclasses are main targets. They have better expressive power for the domain model then relationship among *Instrument*, *Quote*, and *Timepoint*. Figure 2 shows an excerpt from the *Money* archetype pattern where classes gather around *Currency*. Finally, to get more usable domain model *Currency* could be put instead of *Instrument*.

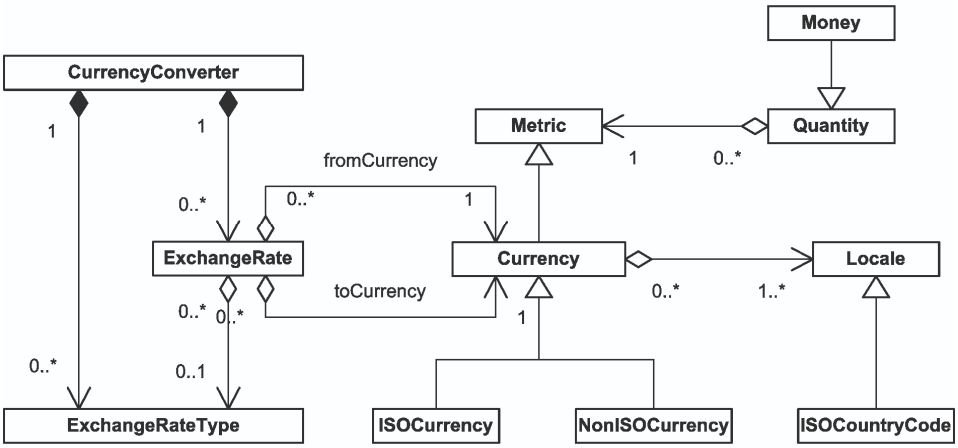

Figure 3. Class diagram of an excerpt of the *Money* archetype pattern [9]

The portfolio analysis is a perfect place where many design patterns from one of the most important design pattern book known as a GOF (Gang-of-five) book [5] could have their place in the model. For instance, *Strategy* pattern [5] is used to encapsulate each different portfolio analysis algorithms, and make them interchangeable. As said, the load balancer has very important role in the system. Its main functionality is bases upon *Composite* pattern [5] to compose distributed objects in two level tree structures to represent path-whole hierarchy for analysis. It is also very useful when one want to improve performance by caching some data. Objects could be created using *Factory Method* pattern [5] when one need to integrate classes specialized for some type of analysis. Load balancer must be implemented as *Singleton* pattern [5] in order to centralize dispatching subtasks accordingly to analysis servers and their freedom to serve new requests. On some rare situation one might use *Chain of Responsibility* pattern [5] to pass request along the chain (analysis servers) until one of them handles it.

4. DATE LAYER ANALYSIS

Date sources have very important role in most application thus one must take care of a way he/she organizes data access and manipulation in persistent storages. According to architecture of the model (figure 1) two data sources exist, first one serves only client transactions and second one collects foreign exchange data from other resources and serves requests for foreign exchange rate on a particular date or time interval. Although their purposes are quite different their internal logic could have the same underlying mechanism based on *Data Access Object* pattern. According to [11] one should use data access objects when want to decouple the persistent storage implementation from the rest of application, provide a uniform data access API for a persistent mechanism to various types of data sources, organize data access logic and encapsulate proprietary feature to facilitate maintainability and portability.

Very often one tries to boost efficiency or usability of some software module or just wants to avoid most-known bad program practices. Refactoring looks like proper solution to needs like these. Fowler [10] defines refactoring as a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour. General refactoring policy concerning data source usage proposes implementing a *connection pool* in order to [11] pre-initialize multiple connections, improving scalability and performance.

Up to now it should be clear that the most data-oriented traffic inside the system will be around Web service. Target platform J2EE provides *Web Service Broker* pattern [11] that is a member of integration tier patterns and serves as a broker to one or more services. In the system these services deal with a plain old java object (POJO) because we want to implement system without EJB container.

Each client's request to create portfolio must begin with uploading a list of currencies from Web service server or Web application server must previously replicate currency table to local database. The authors prefer first approach although it is considerable slower then second one but it does not have redundant

data and does not need any additional module for data replication. Target Web service mainly deals with *Business Object Pattern* [11, 12] in order to capture best of object-oriented approach and to use the same intrinsic logic throughout the whole system. There are few important hints when Web service a business object represents. Some objects might have very complex structure with components that contain additional complex data type and so on. Such tree could have few layers of inclusion and pool unpredictable amount of data. Inheritance have very important role in object-oriented modelling but Web service designer should be aware that Web service clients could be many programming platforms and not all of them are capable of translating inheritance into similar form.

Some tasks need relatively small amount of data (e.g. one or two currency objects) so they could call the Web service as many times as they need. Other tasks (exchange rate trend analysis) that are oriented towards broader time interval need different approach to minimize unnecessary data traffic between a Web service client and a server. Common logic says that the Web service should have additional parameters to be able to capture data collection and return it to a caller instead of performing multiple calls for single data. The obvious solution is *Business Object Collection Pattern* [12] that consists of many single business objects. Its core functionality must provide access methods to fetch single business object based on its position in a collection or key.

Second part of distributed environment (RMI) is just an extended arm of the Web application that operates on other computer. Good thing is that the number of these distributed objects depends only on available number of computers to host them and demand to instantiate and run single object on chosen computer. The RMI module presents *Remote Facade* [13] pattern due to its aim to provide coarse-grained facade on fine-grained object to improve efficiency over a network. The authors use it mainly to perform portfolio analysis for broader time interval that returns serialized bulk object containing all important calculations as lower layer objects. Almost each calculation inside portfolio analysis needs foreign exchange rate so very intensive traffic between the analysis server and the Web service server should be expected.

5. PRESENTATION LAYER ANALYSIS

Last but not the least important part of the model is a presentation layer. It has very significant role for many systems because it is the first (and probably only) visible part of the system for users. Therefore, developers should devote very significant attention to user interface in order to provide users with different data presentations. A typical situation in the system is browsing portfolio value or foreign exchange rate over some time interval. Some users prefer data presented as a chart while other as a table. It should be very wise to apply one of the most used architectural patterns named *Model-View-Controller* (MVC) [5, 13, 14]. The authors plan to use Java applet to visualize data so a controller will capture user's action (e.g. selection of desired visual presentation type, change of time interval, change of scale, etc), and translate it into request for the model or associated view. The View will present data to the user, so one can say that the user sees only view

component from MVC. The model will request data from the analysis server or the Web service server and serve to the view as data provider. Any change in the model will trigger appropriate action in the view.

The rest part of the presentation layer is HTML oriented with implementation in the Java servlet or Java Server Page (JSP) technologies where JSP is more suitable when one wants to separate business logic from the presentation. It must be point out that J2EE platform promotes separation of concerns [11] therefore its system is stack based where user interface is covered in two tiers: Client and Presentation. A client can be Web browser, earlier mentioned Java applet, or some device. Presentation tier deals with presentation logic required to service clients that access the system. The proposed system uses few presentation tier patterns among many of them.

There are many reasons to tract users' activities on the system. All users' requests must be logged in some file or database in order to perform some analyses. *Intercepting Filter* pattern [11] is very useful for this kind of task because J2EE framework calls filter class method *doFilter(...)* on every user request. Inside the method one can do some request data processing, measure execution time, log data, etc. HTML formatting could be very simple and straightforward but also very complex that integrates many modular parts. Simple visual design includes at least header, footer, menu on the left side, and content in central part of window. The solution is described in *Composite View* pattern [11] and it's up to developer to select appropriate strategy among proposed.

6. CONCLUSION

The first objective of designing a model of heterogeneous distributed system for foreign exchange portfolio analysis was to make it fit a wide range of environments and to scale it according to user demand. The authors planned to use software components to perform critical tasks instead of expensive hardware. The second objective was to find as much as possible appropriate patterns for planned tasks in order to capture best modelling and programming practices. The authors used different types of patterns like GOF design patterns, business patterns, J2EE patterns, integration patterns, enterprise patterns, distributed design patterns to Web services patterns. The paper presents just some of the most interesting patterns used in the model. The number of pattern categories shows how deep and wide a pattern community is involved in the process of software developing. It is up to software developers to educate themselves to use patterns instead of reinventing the wheel or repeat commonly occurring bad practices that are documented as antipatterns [4].

REFERENCES :

[1] Dyson, P., Longshaw, A. *Architecting Enterprise Solutions, Patterns for High-Capability Internet Based Systems*, John Wyles & Sons, 2004.

[2] ProactiveNet. *Speed is King!* http://www.proactivenet.com/Library/Speed_Is_King.pdf

[3] Nelson, M. G. *Fast Is No Longer Fast Enough*. http://www.informationweek.com/789/web.htm

[4] Greenfield, J., Short, K. *Software Factories: Assembling Application with Patterns, Models, Frameworks, and Tools*, Wiley Publishing, 2004.

[5] Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns, Elements of Reusable Object/Oriented Software*, Addison-Wesley, USA, 1995.

[6] Farley, J. *Java Distributed Computing*, O'Reilly, 1998.

[7] Fowler, M. *Analysis Patterns: Reusable Object Models*, Addison Wesley Professional, 1997.

[8] Eriksson, H-E., Penker, M. *Business Modelling with UML*, Business Patterns at Work, 2000.

[9] Arlow, J., Neustadt, I. *Enterprise Patterns and MDA, Building Better Software with Archetype Patterns and UML*, Addison Wesley, 2004.

[10] Fowler, M. *Refactoring: Improving the Design of Existing Code*, Publisher: Addison Wesley Professional, 1999

[11] Alur, D., Crupi, J., Malks, D. *Core J2EE Patterns, Best Practices and Design Strategies, 2nd Edition*, Sun Microsystems Press, 2003.

[12] Monday, P.B. *Web Service Patterns: Java Edition*, Apress, 2003.

[13] Fowler, M. *Patterns of Enterprise Application Architecture*, Addison Wesley Professional, 2003.

[14] Buschmann, F, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Addison-Wesley, Reading, MA, 1996.

Received: 31 October 2005
Accepted: 30 June 2006