TELKOMNIKA, Vol.17, No.5, October 2019, pp.2208~2217
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v17i5.8222

Hybrid distributed application in banking transaction using remote method invocation

Agus Cahyo Nugroho*[1], Mychael Maoeretz Engel[2]
[1]Information System Department, Computer Science Faculty, Soegijapranata University
Pawiyatan Luhur IV/1 Bendan Dhuwur St., Semarang 50234, Indonesia
[2]Informatics Department, Information Technology Faculty, Ciputra University
UC Town, Citraland, Made, Sambikerep, Surabaya, East Java 60219, Indonesia
*Corresponding author, e-mail: agus.nugroho@unika.ac.id[1], mychael.engel@ciputra.ac.id[2]

Abstract

Today banks have many branches in big cities of the world. System usually used a central database in a particular city. Increased of database server performance due to number of users accessing this application should not degrade performance of application. To keep database server performance optimally, application must distributed to the network. In distributed applications it takes a remote method call, that is why we are going to used Remote Method Invocation to develop this system. Based on results of analysis conducted, author can draw following conclusion of the application, which is once the client get a reference from the remote object then method of remote object is called like calling method from local object and methods that we have defined and implemented on remote object can we call or use both on desktop and web applications so we do not need to work twice. This approach makes more effective and efficient in application development, allows for better optimization, eliminates the need for processing of type information at run time and makes a light weight communication protocol possible. We have built a hybrid application, which supports both compile time and run time generation of marshallers in desktop and web application.

Keywords: *distributed application, hybrid application, remote method invocation*

1. Introduction

At this time various banks have had many branches in major cities of world. System used usually has a central database in a particular city. Application used allows for open a new savings account, handle deposits, withdrawals, transfers money between savings accounts at same bank through desktop applications and website. Prospective customers from different parts of world will access this application to submit their personal data as a condition for creating a new account at bank. This personal data must meet various bank policies, such as prospective customer must be over 18 years of age. According to Boyd et al. [1] the importance of bank selection criteria in terms of the age of the head of the household. Customers who already have account savings in bank can also transfer money between account savings at same bank. The importance of technology and speed has also been confirmed by Coyle [2] who observed "Future Bank" trade show in Minneapolis (USA) where 250 bank vendors participated. Coyle reported that the competitive bank of the future is the one which can offer speedy, technology based services (e.g. ATM, Internet) backed by an effective staff training.

Problem caused by increased database server performance due to large number of users who accessing this application should not degrade performance of application. Ideal solution of this scenario is application users access and sending information using client application. Information submitted will be validated by a component on server component. After that component on server checks whether data submitted has met bank policy, if fulfilled then server components make a connection with database server and then transmit data.

To solve above problem, we should keep database server performance optimally, the ideal solution is that application must be distributed to network. Therefore, we are going to used three-tier application [3] because three-tier applications support deployment of distributed-based applications [4] component. In distributed applications it takes a remote method call, then we are going to used Remote Method Invocation [5] to develop this system.

Received January 10, 2019; Revised April 25, 2019; Accepted 13 May, 2019

2. Research Method

This research is different from other research before because we are trying to create two different applications which are desktop and web application that are using the same methods or functions located in desktop application, so everytime we are going to changed methods or functions we do not have to work twice. We only work once to changed methods or functions in the desktop application after that both desktop and web application can used or called that methods or functions. The web application is capable to call methods or functions in desktop application through servlet. This make the development time more effective and efficient.

Earlier research [6] stated that unlike previous approaches where only application services can be invoked, Android RMI allows users to invoke system services as well as application services between devices using remote parcel format. By reducing the number of marshalling and unmarshalling steps, the time taken for remote method invocation is shortened by 148% in 4 KBytes and by up to 432% in 100 KBytes compared to distributed intent where additional marshalling and unmarshalling steps are needed.

Another earlier research [7] stated that present day's amount of computational requirements has shifted the processing of data from the regular way to parallel way of computation. Pipelined processors, array processors can be employed to construct design of parallel hardware. These systems constructed can be further extended with the help of scalar and super scalar systems. We provide an efficient way of implementing Winograd's variant of Strassen's matrix multiplication on parallel systems by making use of RMI (Remote Method Invocation) which provides us distributed object oriented programming, multithreading programming. Multithreading approach helps a lot in concurrent, dynamic and asynchronous programming.

2.1. Remote Method Invocation (RMI)

Remote Method Invocation is a specification that allows a Java Virtual Machine (JVM) [8] to call methods of that object located on another Java Virtual Machine (JVM). Both JVM's can run on a different computer or running as a separate process on same one computer. RMI is implemented in middle-tier [9] of three-tier framework architecture, this facilitates programmer to invoke distributed components over network. Sun introduced RMI as an easy to complex alternative in server-socket programming [10]. In using RMI, programmer does not need to master socket or multi-threading programming [11], as needed just concentrate on developing business logic [12].

RMI distributed application has two components: RMI Server [13] and RMI Client [14]. RMI Server consists of objects whose methods will called remotely. The server created some remote objects later create references from those objects in RMI Registry [15]. RMI Registry is service that runs on RMI Server. Remote objects created by server then listed in this registry according to unique name of the object. Client refers one or more remote objects from RMI Registry to see name of object. Then client calls methods on remote object. Once client refers to the remote object, methods on remote object called as calling method on local object. This difference can not be identified whether method is called on remote object or called on local object in client.

RMI architecture consist of three layer: Stub/Skeleton Layer [16], Remote Reference Layer [17] and Transport Layer [18]. Stub/Skeleton Layer is waiting for remote method call by client and forward it to the remote RMI Service on server. This layer consists of a Stub and a Skeleton. To call methods on remote object, request on the client side starts with Stub. Stub is a proxy on client side representing remote object. Stub is referenced as another local object by program that runs on client and provide methods of remote objects. Stub communicates method call on remote object via implementation of skeleton on server. So, Stub on client collects information consisting of: identifier of remote object to be used, description of method we will call and parameters that have been marshalled.

Skeleton is a server-side proxy that forwards communication with Stub, by reading parameters at method call, then make a call to remote service object that is implemented, receive return value, then write return value to Stub. So Skeleton on server does things below every time there is a Remote Method Invocation: unmarshals parameters, search for called object, calling desired method, catch and marshals return or exception value of calling and submitting a packet consisting of returned value marshalled to Stub on client.

Remote Reference Layer interprets and manages references of client to remote object on server. This layer is present both on client and server. RRL on client side receives a request method from Stubs are sent as marshaled streams from data to RRL on server side. Marshalling [19] is a process which are parameters passed by client converted to a format that can be sent over network. RRL on server side unmarshals parameters sent to remote method through Skeleton. Unmarshaling is a process which are marshaled parameters passed by RRL on client side via RRL on server side changed to the format which is understood by Skeleton. When returning value from Skeleton, data marshaled back and communicated to client via RRL on server side.

Transport Layer is a link between RRL on server side and RRL on client side. Transport Layer is responsible for setting up new connections and setting up an existing connection. Transport Layer is also responsible for handling remote object in address space. Below are steps that explain how client connected to server: when receiving request from RRL on client side, Transport Layer create a socket connection to server via RRL on server side. Then, Transport Layer skips connection that already connect to RRL on client side and add remote reference to connection on connection table.

Based on the RMI specification, steps in developing distributed RMI application: defines remote interface class, defines and implement various remote method in server class. Defines and implement client class. Compile source files. Source files include those files contains definition of remote interface, which server class defines the implementation of remote interface and client class. Generating Stub and Skeleton. Create a security policy [20]. Run RMI Remote Registry [21]. Run server. Run client.

In a three-tier architecture, presentation logic is on client-side, access database is controlled by server-side, and business logic lies between two other layers. Business logic layer is also known as application server or middle-tier of component-based three-tier architecture. This type of architecture, too known as server-centric [22], because it allows application components to path on middle-tier application server then implements business rule, so it separated between presentation interface and database implementation. This component can be developed using any programming language that allows component creation. These components can be created centralized to facilitate the development, maintenance, and deployment. Because middle-tier handles business logic, workload becomes balanced between client, database server and server that handles business logic. This architecture leads to efficient data access. Problem regarding limitations of connection to database is minimized because database only view business logic layer and not whole client. Unlike in two-tier application, database connection occurs at beginning and then set as long as it is still access data, while in three-tier application database connection takes place only when data access is required and released when data is returned or sent to server.

This various advantages motivated author to use RMI in case of banking transactions. This method is considered very appropriate for banking transactions that handle large number of customers and prospective customers. Because of that reasons data access needed efficiently and minimize limitations of connections to database.

2.2. Servlet

Servlet [23] is a Java program that can be deployed on Java enabled Web server [24] to make maximum functionality of Web server. Servlet can be used to develop various web-based applications. Servlet developed using Java so we can use various advantages of Java API, this is why we can use servlet to access RMI. Therefore, author uses servlet in development of application that handle banking transactions via browser. Servlet works by client or browser to send request to server using GET or POST method. Example: servlet can be called as a result of user pressing user-interface component, such as buttons on web page form. After request processed by servlet, output is returned to client in form of html page.

Java supports servlet implementation through javax.servlet and javax.servlet.http package. Interface javax.servlet.Servlet provides general framework in manufacture of servlets. Servlet can implement equally either directly or indirectly by extend class javax.servlet.GenericServlet or javax.servlet.http.HttpServlet. GenericServlet class of javax.servlet package is used to create servlets that can work with various protocols. Package javax.servlet.htpp is used to create a HTTP servlet generate output in form of HTML pages.

Class used to make HTTP servlet called HttpServlet and derived from class GenericServlet. Complete description about this HttpServlet class/interface can be seen in Table 1.

Table 1. Class/Interface Description

Class/Interface Name	Description
HttpServlet class	Provide specific HTTP implementations from Servlet interface. This class is extended GenericServlet class that provides framework to handle different types of networks and Web service.
HttpServletRequest interface	Provides methods to process requests from clients.
HttpServletResponse interface	Responding to requests from clients sent back in HTML page format through object of HttpServletResponse interface.
ServletConfig interface	Used to store start servlets value configuration and initialization parameters. getServletConfig () method of Servlet interface used to obtain information about configuration values of servlet.

Servlet is loaded only once in memory and initialized during init () method called. After servlet is initialized, servlet is ready to accept request from client and process request through service () method until it is stopped by the destroy () method. Service () method is called every time there is a request. Life cycle of servlet is described as we can seen in Figure 1. Table 2 illustrates some of methods used in servlet creation.

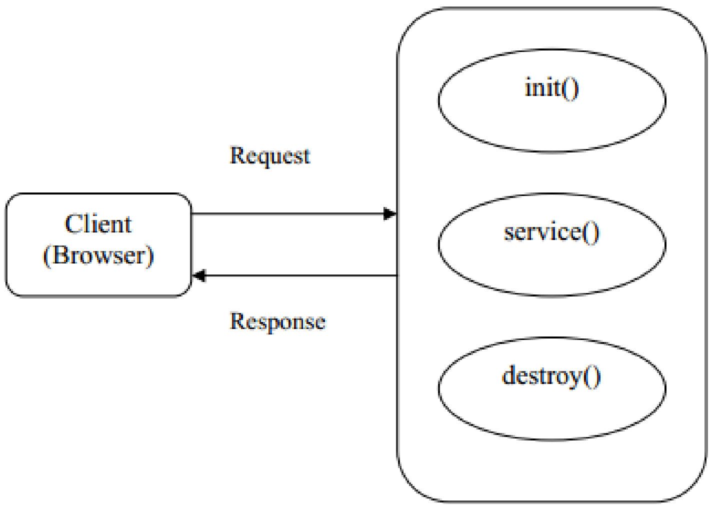


Figure 1. Servlet lifecycle

Table 2. Function Method in Servlet Creation

Method Name	Function
Servlet.init(ServletConfig config) throws ServletException	Contains all servlet initialization code and called when servlet is first loaded.
Servlet.service(Servlet Request, Servlet Response)	Receive all requests from client, identify request type, and dispatch request to doGet () or Post () for processing.
Servlet.destroy()	Executed only once when servlet is on remove from server.
ServletResponse.getWriter()	Returns reference to object PrintWriter. Class PrintWriter is used to write an object that is formatted as text-output stream in the client.
ServletResponse.setContentType (String type)	Set the type of content that is sent as a response to client. Example: setContentType ("text / html") is used to set response type as text.

2.3. Java 2 Enterpise Edition (J2EE) Server

Java 2 Enterprise Edition (J2EE) [25] is a set of specifications that define standards in manufacture of distributed objects. J2EE also set up how this technology can be integrated to provide that complete solution. J2EE is also standard architecture that facilitate to multi-tiered programming model. J2EE server is used to deploy servlets and JSP [26] files allow users to access it by applying proper security.

2.4. Data Flow Diagram (DFD)

Data Flow Diagrams [27] are diagrams that describe data process along with flow in business system. We are going to make two Data Flow Diagrams. The first one is for Web Application, second one for Desktop Application.

2.5. Data Flow Diagram Web Application

The first one is Data Flow Diagram for Web Application, we can see in Figure 2. Complete explanation about this Data Flow Diagram for Web Application we can see below Figure 2.

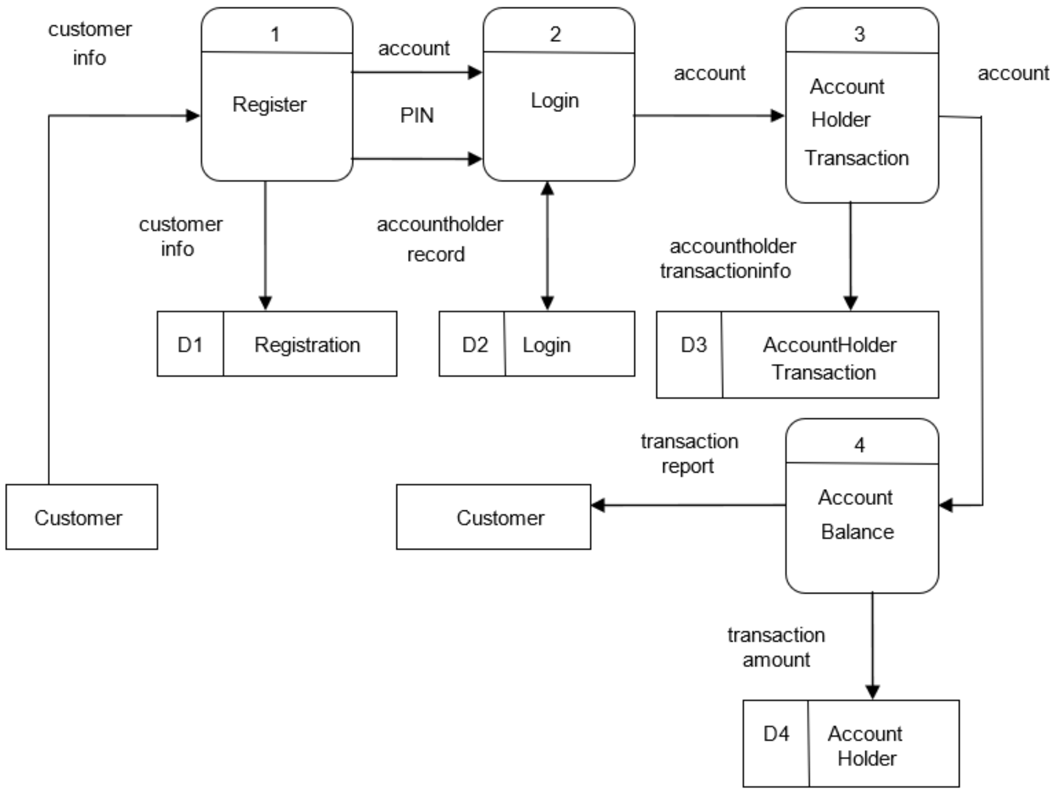


Figure 2. DFD web application

Explanation of Data Flow Diagram (DFD) Web Application above:
- Customers send their data (customer info) as a registration condition to bank customers. System stores customer’s personal data on Bank database Registration table.
- After becoming a bank customer, user obtains account number and Personal Identification Number (PIN) that can be used for login to system. System checks account number and Personal Identification Number (PIN) user to Bank database Login table, if true then user can log into system.

- On the next page user can transfer to account other customers. This transaction is recorded on Bank database Account Holder Transaction table. Then there is an adjustment (balance) both on sender's account and transfer recipient on Bank database Account Holder table.
- Customer receives report that transfer succeeded and their last balance.

2.6. Data Flow Diagram Desktop Application

The second one is Data Flow Diagram for Desktop Application, we can see in Figure 3. Complete explanation about this Data Flow Diagram for Desktop Application we can see in Figure 3.

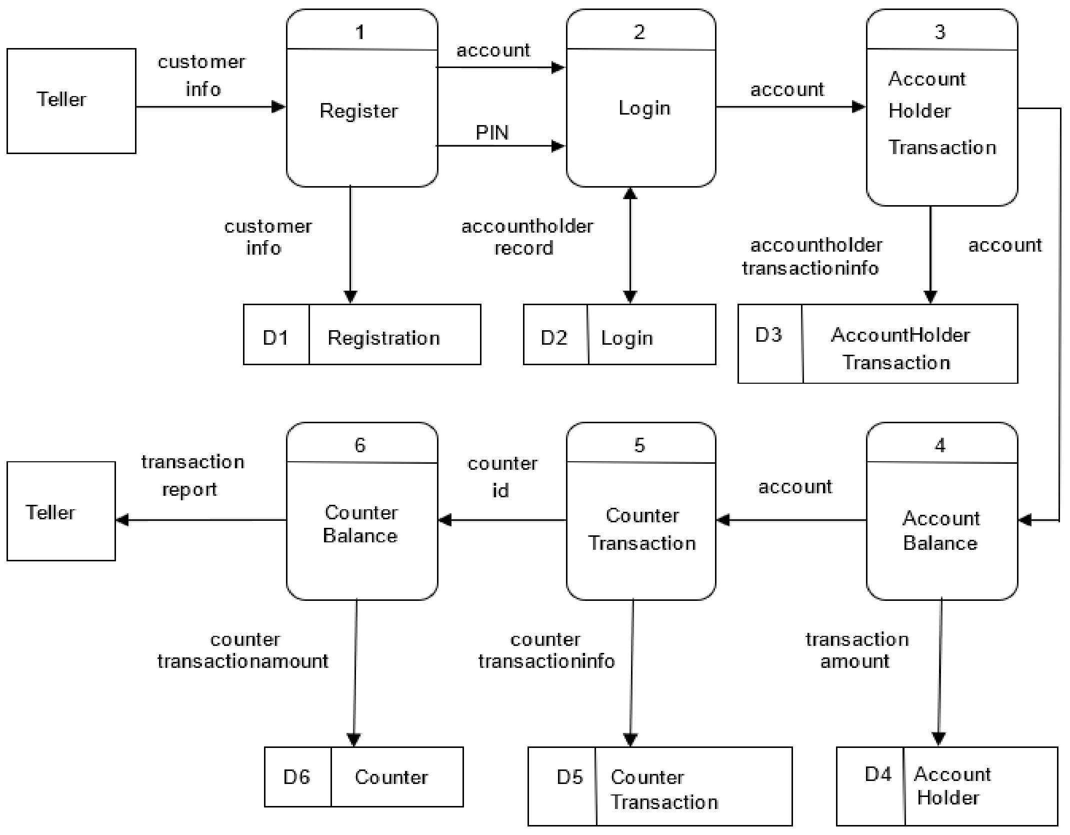


Figure 3. DFD desktop application

Explanation of Data Flow Diagram (DFD) Desktop Application above:
- Officer (Teller) to register data of prospective customers to Bank database Registration table.
- Officer (Teller) must login to system for using all system facilities.
- Officers (Teller) can perform transactions such as deposits, withdrawals and transfers, all transactions are recorded as AccountHolderTransaction Info on Bank database table AccountHolderTransaction.
- After transaction done, made account adjustment (account balance) on Bank database Account Holder table.
- All transactions that have occurred in branches are recorded at Bank database CounterTransaction table.
- Lastly adjusted branch balance (counter balance) on Bank database Counter table.
- System gives report that transaction has been successful or failed and last balance.

3. Results and Analysis

Here is the implementation of system.

3.1. Implementation of Servlet

We are using Servlet to access same method that desktop application used. In Figure 4 is web login page that call method in the firstservlet. We are using Account Server server=(Account Server)Naming.lookup ("rmi://localhost/ AccountServer"); in Figure 5 to get reference from remote object implementation. Method lookup using name that we are registered to the server as parameter. Beside that, we are calling menuservlet not because of button click but by Request Dispatcher interface. We are sharing account number attribute in Figure 6 using Servlet Context context=getServletContext(); Object obj=context.getAttribute("accountnumber"); Because account number data type is integer then we need to parsed it to String so we can compared it. After that checked the session.

```
<Form onSubmit="return valid(this)" method=post
action="http://127.0.0.1:8000/RMIContext/firstservlet">
<td>Enter ...</td><td><input type= text name= accnum></td>
<td>Enter ...</td><td><input type= password name=pinnum></td>
<center><input type=submit value=submit></center>
</form>
```

Figure 4. Web login page

```
public class firstservlet extends HttpServlet
{
...
AccountServer server = (AccountServer)Naming.lookup
("rmi://localhost/AccountServer");
str = server.cekLogin(userName,password);
if(str.equals("fail"))
{...}
else
{
...
//Jika valid panggil second servlet
RequestDispatcher dispatcher =
getServletContext().getRequestDispatcher("/menuservlet");
...
}
```

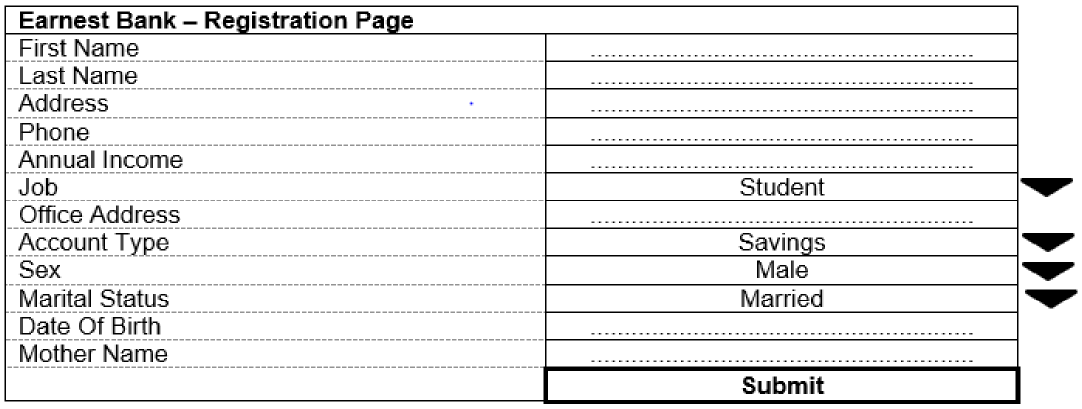
Figure 5. First servlet

```
public class menuservlet extends HttpServlet
{
public void service(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException
{
...
ServletContext context = getServletContext();
Object obj = context.getAttribute("accountnumber");
String value = obj.toString();
if(value.equals(session.getAttribute("uName")))
{
out.println("<Form method=post action=http://
127.0.0.1:8000/RMIContext/validasimainmenu>");
out.println("<center>
<input type=submit name=setorButton value=Penyetoran>
<input type=submit name=transferButton value=Transfer>
...</center>");
out.println("</form>");
}
else{}
}
}
```

Figure 6. Menu servlet

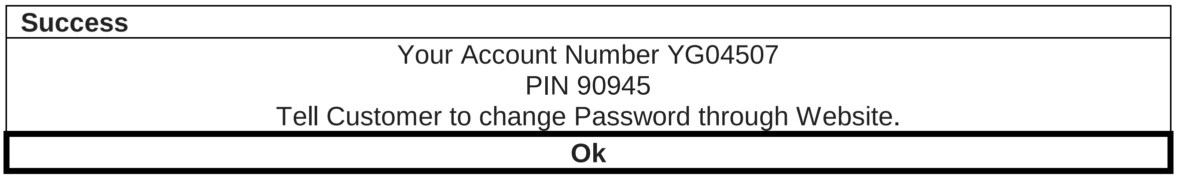
3.2. Desktop Application

Registration menu is menu that appears when teller input customer information for first time like in Figure 7. If registration is successful system will give output that registration is successful and inform customer account number as well as initial PIN to login via web application. A window informing you that registration was successful can be seen in Figure 8. In Figure 9 we can see login page of desktop application. Teller must insert username and password for using menus in system.



Earnest Bank – Registration Page	
First Name	....
Last Name	....
Address	....
Phone	....
Annual Income	....
Job	Student ▼
Office Address	....
Account Type	Savings ▼
Sex	Male ▼
Marital Status	Married ▼
Date Of Birth	....
Mother Name	....
	Submit

Figure 7. Registration menu

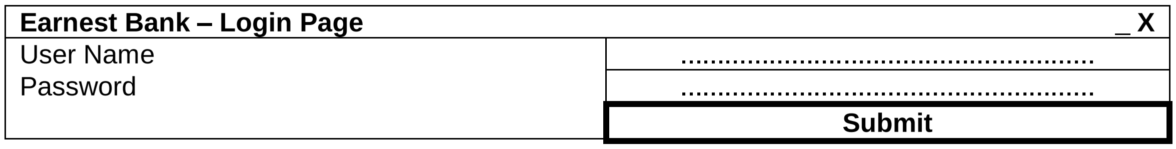


Success

Your Account Number YG04507
PIN 90945
Tell Customer to change Password through Website.

Ok

Figure 8. Successful information window

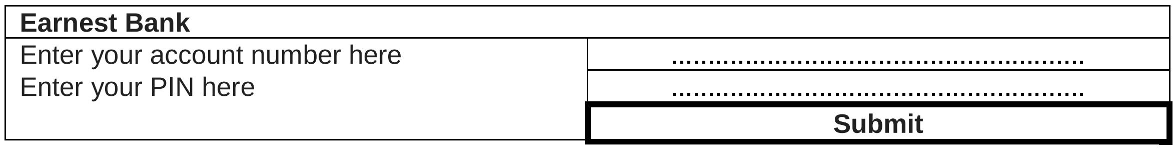


Earnest Bank – Login Page	_ X
User Name	....
Password	....
	Submit

Figure 9. Desktop login page

3.3. Web Application

Login web page is page that appears when customer already registration through teller (desktop application) and get PIN to login through web page like in Figure 10. In Figure 11 we can see main menu of web system. In Figure 12 we can see transfer menu of web system.



Earnest Bank	
Enter your account number here	....
Enter your PIN here	....
	Submit

Figure 10. Web login page

Earnest Bank – Main Menu			
Deposit	**Transfer**	**Update PIN**	**Logout**

Figure 11. Web main menu

Earnest Bank – Transfer	
Your Account Number	YG01507
Recipient Account Number	....
Transfer Amount	....
Description	....
Branches	Web
	Submit

Figure 12. Web transfer menu

4. Conclusion

This paper presented a hybrid application to support both compile time and run time generation of marshallers in desktop and web application. This application can be run as separate processes on one same computer or run on different computer. For optimization and light weight communication protocol once client gets reference from remote object then method of remote object is called as calling method from local object. We have defined and implemented on remote object can we call or use both in desktop and web application so we do not need to work twice. If we are going to developed web application, we can used Servlet to call remote method that we are implemented in desktop application. This make the application development more effective and efficient.

References

[1] W Boyd, M Leonard, C White. Customer preferences for financial services: an analysis. *International Journal of Bank Marketing*. 1994; 12(1): 9-15.

[2] T Coyle. The bank of tomorrow. *Americans Community Banker.* 1999; 8(7): 16-18.

[3] DL White, et al. *The Intelligent River©: Implementation of Sensor Web Enablement technologies across three tiers of system architecture: Fabric, middleware, and application.* 2010 International Symposium on Collaborative Technologies and Systems, Chicago, IL. 2010: 340-348.

[4] RA Kendall, E Aprà, DE Bernholdt, EJ Bylaska, M Dupuis, GI Fann, RJ Harrison, J Ju, JA Nichols, J Nieplocha, TP Straatsma. High performance computational chemistry: An overview of NWChem a distributed parallel application. *Computer Physics Communications.* 2000; 128(1-2): 260-283.

[5] PA Hartmann, P Ittershagen, K Grüttner, F Oppenheimer, A Rettberg. A framework for generic HW/SW communication using remote method invocation. *Analysis*. 2011; 3(T5): S0.

[6] H Kang, K Jeong, K Lee, S Park, Y Kim. Android RMI: a user-level remote method invocation mechanism between Android devices. *The Journal of Supercomputing.* 2016; 72(7): 2471-2487.

[7] H Kaur, S Bagga, A Arora. *October. RMI approach to cluster based Winograd's variant of Strassen's method.* MOOCs, Innovation and Technology in Education (MITE), 2015 IEEE 3rd International Conference on. 2015: 156-162.

[8] T Lindholm, F Yellin, G Bracha, A Buckley. The Java virtual machine specification. Pearson Education. 2014.

[9] C Guo, G Lu, HJ Wang, S Yang, C Kong, P Sun, W Wu, Y Zhang. *Secondnet: a data center network virtualization architecture with bandwidth guarantees*. Proceedings of the 6th International Conference. 2010: 15.

[10] M Xue, C Zhu. *The socket programming and software design for communication based on client/server*. Circuits, Communications and Systems, 2009. PACCS'09. Pacific-Asia Conference on. 2009: 775-777.

[11] AHS Data. Multi-thread Programming. 2009.

[12] P Ain, R Kothari. Mapping software code to business logic. U.S. Patent 7,640,532. International Business Machines Corp. 2009.

[13] KJ Chou, MIW Huang, T Lee, BN Soetarman, RN Summers, MPT Vo. Architecture and implementation of a dynamic RMI server configuration hierarchy to support federated search and update across heterogeneous datastores. U.S. Patent 7,197,491. International Business Machines Corp. 2007.
[14] D Hou, H Xia. *Design of distributed architecture based on java remote method invocation technology*. Environmental Science and Information Application Technology, 2009. ESIAT 2009. International Conference on. 2009; 2: 618-621.
[15] K Kang, J Lee, H Choi. *Extended service registry for distributed computing support in osgi architecture*. Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference. 2006; 3: 1631-1634.
[16] MB Juric, I Rozman, B Brumen, M Colnaric, M Hericko. Comparison of performance of Web services, WS-Security, RMI, and RMI–SSL. *Journal of Systems and Software.* 2006; 79(5): 689-700.
[17] EH Page, RL Moose Jr, SP Griffin. *Web-based simulation in Simjava using remote method invocation*. Proceedings of the 29th conference on Winter simulation. IEEE Computer Society. 1997: 468-474.
[18] P Bajpai, VK Jain, AS Akella. Method and node for employing network connections over a connectionless transport layer protocol. U.S. Patent 8,750,112. Echo Star Technologies LLC. 2014.
[19] AS Huang, E Olson, DC Moore. *LCM: Lightweight communications and marshalling*. Intelligent robots and systems (IROS), 2010 IEEE/RSJ international conference on. 2010: 4057-4062.
[20] J Aarnos, P Pentikainen, Nokia Oy AB. Method for enforcing a Java security policy in a multi virtual machine system. U.S. Patent Application 11/126,651. 2006.
[21] MB Juric, I Rozman, B Brumen, M Colnaric, M Hericko. Comparison of performance of Web services, WS-Security, RMI, and RMI–SSL. *Journal of Systems and Software.* 2006; 79(5): 689-700.
[22] C Guo, G Lu, D Li, H Wu, X Zhang, Y Shi, C Tian, Y Zhang, S Lu. BCube: a high performance, server-centric network architecture for modular data centers. *ACM SIGCOMM Computer Communication Review*. 2009; 39(4): 63-74.
[23] H Cai, W Lu, B Yang, LH Tang. Method for accessing and collaborating between servlets located on different Java virtual machines. U.S. Patent 7,543,289. International Business Machines Corp. 2009.
[24] N Karapanos. Strengthening Authentication and Integrity in Web Applications (Doctoral dissertation, ETH Zurich). 2018.
[25] J Pachouly, V Dange. Automating live update for J2EE applications over distributed environment. *International Journal of Advanced Technology and Engineering Exploration*. 2018; 5(43): 99-106.
[26] TD Samaranayake, WPJ Pemarathane, B Hettige. *Solution for event-planning using multi-agent technology*. Advances in ICT for Emerging Regions (ICTer), 2017 Seventeenth International Conference on. 2017: 1-6.
[27] K Tiwari, A Tripathi, S Sharma, V Dubey. Merging of Data Flow Diagram with Unified Modeling. *International Journal of Scientific and Research Publications*. 2012: 2(8): 1-6.