Modernization Solution for Legacy Banking System Using an Open Architecture

Constantin Marian MATEI
Faculty of Economic Cybernetics, Statistics and Informatics
Academy of Economic Studies, Bucharest, Romania
constantinmatei81@yahoo.com

Banks are still using legacy systems as the core of their business is comprised within such systems. Since the technology and client demands are changing rapidly, the banks have to adapt their systems in order to be competitive. The issue is to identify correctly what are the bank users preferences in terms of software reliability and how modern is the system For instance, there are users who enjoy working using the old screen format, and there are users who enjoy working with newer layouts, Web interfaces, and so on. We need also to know the constraints generated by the usage of legacy systems, and how these systems can be improved or replaced. The scope of the article is to present a solution of modernizing a legacy banking system by using a SOA approach. The research is based on the modernization of a legacy system developed in COBOL/400 under IBM iSeries. The modernization process uses a SOA approach using JAVA technologies.
Keywords: *Legacy Systems, Architecture, Services, Front Office, Back Office, Server, Message Queue, SOA*

1 Introduction

One of the big dilemmas of the banking CIOs is “Are we going to change the existing systems with more scalable ones?” If yes, are we able to cover the costs?”
This dilemma it has another constraint related to the fact that new players in the banking world have already new opened platforms which permits applications for 24 hours online banking, internet banking, mobile banking and so on.
The reason why the legacy systems are surviving is that the costs and risks to replace them are high enough and a system proprietary can run into a bankruptcy if the change will occur. There is also another reason related to the business logic. The applications were developed started with the 70’s and were progressively adapted to the business demand. Thus, the business logic coded into the systems is very complex and can lead to a high risk of failure in case of replacement.
In 2008 a TowerGroup analyst said that there is a pressure coming not only from the technology itself, but also from the business point of view. [1] Competitors are using new open architectures capable to do smart reports, data mining, and, why not, to integrate with platforms like tablets, notebooks and smart phones. This increases the speed of concluding a business and also offers a better system for decision making process.
Since these issues were raised by the business and technical communities, large companies have developed open architecture based on services, like SOA. Moreover, these providers are offering solutions based on cloud computing.
The question now is if the banks have to create from scratch new systems, to buy products from various vendors or to modernize existing legacy systems.
To be able to respond to this question we need to know the constraints generated by the use of the legacy systems [2]:
- legacy systems were designed for the immediate needs and they weren’t planned to be active for too many years (more than 15 years);
- these systems were built under some constraints (for instance low cost, resource availability, and so on);
- existing systems attributes such as: complexity, commercial components, business objectives [3].
Rewriting the application is expensive (time

consuming), and also risky (a lot of defects may appear during acceptance testing, and more important, the business might change during development). The advantage consists in the fact that the bank can have a software application built to meet its specific requirements. [4]

Second option consists in the acquisition of commercial off-the-shelf (COTS) systems, which are ready to use and need to be customized for the bank own business. The risks consist in the maintenance of such products, as the costs for modification are usually at higher rates. The advantages consist in the fact that the software is immediately ready to be used, SLA's (Service Level Agreements) can be established for support services. Usually SLA's are response time (the third party support team is bound to respond in a certain amount of time to the banks requirement) and resolution time (the support team has to solve the incidents or tickets in a fixed period of time). If SLA's are not respected then fines are perceived from the software provider. For COTS there is a risk of implementing the software package. In [5] there are identified two major risks of replacing existing system with COTS: the maintenance of the new system won't be as familiar as the old system; there is no guarantee that the new system will work the same as the old one.

The last approach supposes the migration of the legacy system to a SOA environment which is more flexible. Moreover the original's system data and functionality will be kept. There are multiple choices to do this migration, out of which we mention the automatic transformation with tools like IBM Rational HATS (Host Access Transformation Services), the usage of Model View Controller (MVC) patterns like SPRING, JSF (Java Server Faces), or new approaches like Model View Presenter (MVP) patterns (C#, Vaadin – open source framework with Apache license).

The common point of these approaches is that each time a legacy system is changed the following actions are taken into consideration: assessment, transformation, refactoring, web-enablement [6]:

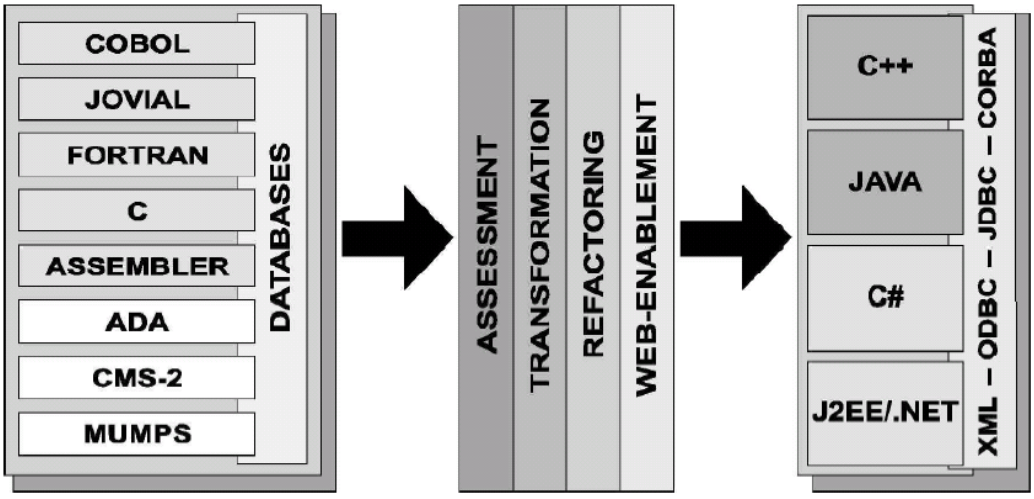

Fig. 1. Legacy system transformation actions

The present article scope is to present a modernization solution based on an open architecture, using SOA principles and MVC patterns.

2 Define the problem

In the process used to define the problem we considered the following working configuration: the legacy system is a core-banking system developed on IBM iSeries (in COBOL language); there is already a middle tier system used to connect the core-banking to different front-offices based on specific web services developed using J2EE technology.

We will use a Cash Deposit creation type of transaction. This type of transaction supposes that a bank operator based on specific data will create a cash deposit for a certain entity

(legal person or physical person).
The below picture displays the actual iSeries screen for the Create Cash Deposit option:

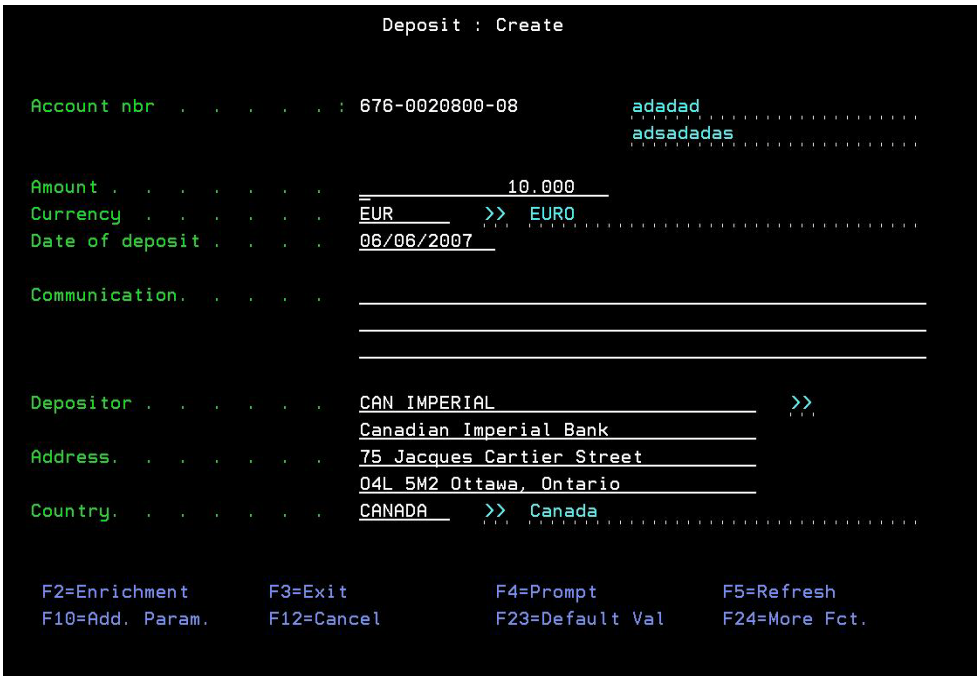

Fig. 2. Create Cash Deposit screen

This is a simple screen, but still it has the disadvantage of the limited display options.
The middle tier system mentioned before it has an architecture which permits the connectivity of various extern platforms to the back-office system. For instance SWIFT platforms or specific Online Banking applications are using the middle tier to send business requests to the back office.

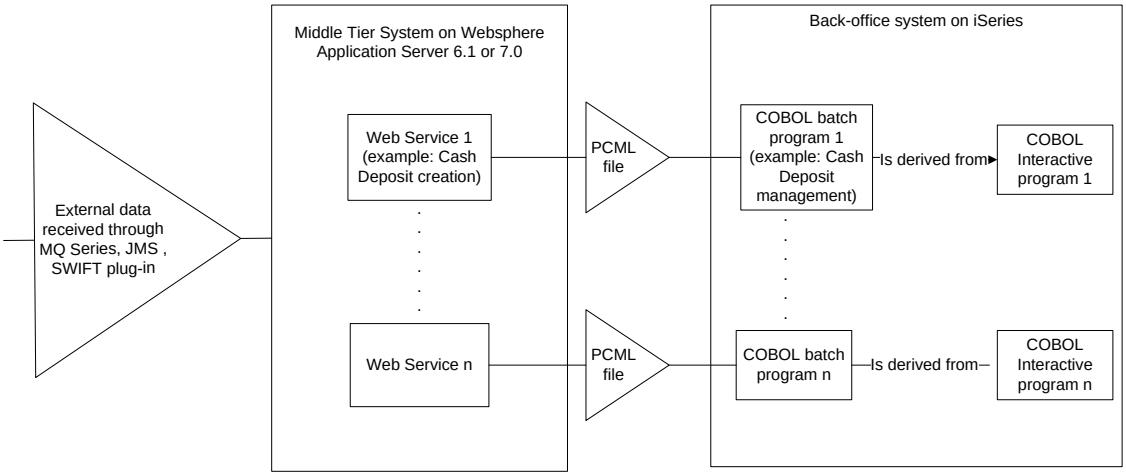

Fig. 3. Existing legacy system architecture

The main disadvantage of this kind of architecture is that the presentation layer of the back-office is still a 5250 screen for iSeries or a 3270 for Mainframe, with the same functionalities and rigid display. This type of display is no longer attractive for the clients, as we have to take in account that there is a new generation of end-users with more technological knowledge and abilities.
There are some problems of costs as well. As described in [7] we can calculate the complexity of a system based on the number of

interfaces used in building that system and the number of transactions processed. The complexity of this system would be around 166.7%, which is higher comparing to another type of architecture based on direct call to the iSeries DB2 embedded database.

There is also an advantage point of view which must be considered: the business logic of the COBOL programs is kept. The cost will be translated as the amount of time of converting an existing interactive screen into a batch one. For instance, for a screen having six input fields (like the one in Fig.1) the conversion time is one man-day. This time comprises activities such as: delete the links to the screen file; suppress the field indicators and function keys usual behaviors.

Based on above information the issue is that the system has an old display which is inflexible and is not attractive for new clients.

3 Describe the solution and the results obtained

The solution proposed further is based on a function by which we measure the performance of the system and the marketability factors.

System performance function: is a derived function from the complexity indicator [7] and the number of business transactions processed in the same amount of time and for the same technical architecture. In our case, the WebSphere 6.1 server was running on a Windows 2003 Server machine with 2 GB RAM dedicated for the application server.

We will use the notations: CP for Complexity, $t_{create\_transaction}$ for time to complete a transaction, Tx for number of transactions and t_{load_system} for time to load the application on the server.

The function is described below as:

$$System\ performance\ (SP) = CP * Tx * t_{create\_transaction/tload\_system}$$

This means that for a system with a high complexity and a large number of transactions processed in a constant time, the performance is also high.

Marketability factors take in account attributes of the solution such as:

How much flexible is the new screen in terms of introducing data, field validations and error message display?

What is the possibility to connect other applications to the system?

To what extent the new interface is easy to learn compared with the existing interface?

The solution proposed hereunder is based on the following schema:

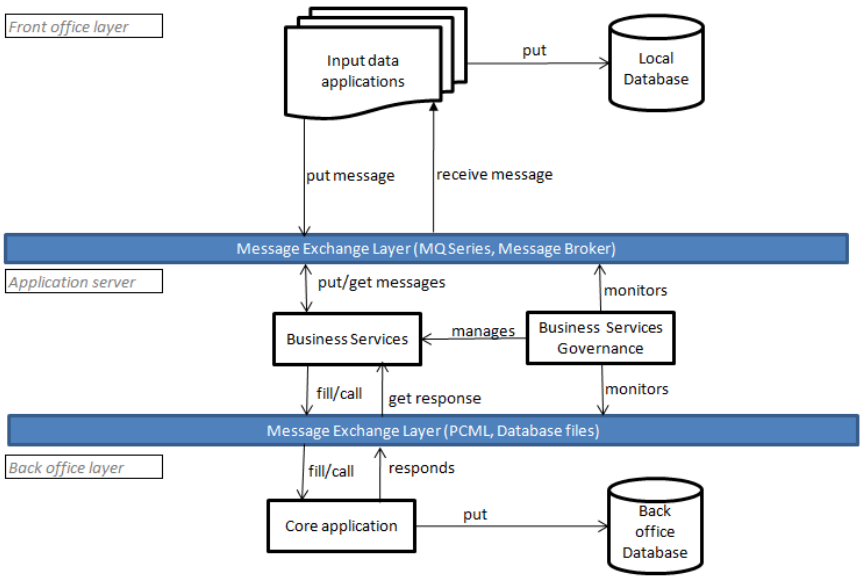

Fig. 4. Solution architecture

Each layer in the picture above has its own scope in the system:

Front office layer purpose is to collect the data

Application server layer has the services dispatcher role
Back office layer has the business logic management role.
Beside the layers presented above we have 2 message layers. These layers are used for the exchange of messages between all other business layers.
From a technical point of view the technologies involved in this solution varies based on each layer:

- front office: Java Server Faces (JSF), Java Server Pages (JSP);
- first message layer: Java Message Service architecture managed through Spring framework;
- application services layer: Java Enterprise Edition (JEE), Enterprise Service Bus (ESB), Web Services;
- second message layer: Program Call Markup Language (PCML), Program Call Beans;
- back office layer: COBOL batch programs derived from interactive screens.

Below there are presented in several images the results of implementing the solution described above. We will present the screens with the most importance in the application.
The front-office layer is actually a GUI application able to load data into the message layer and to transport them further to the back-office (via the application services layer). It has usual functionalities like: login page, authentication, transaction creation and visualization. The technology used to build the GUI is mainly based on JSF:

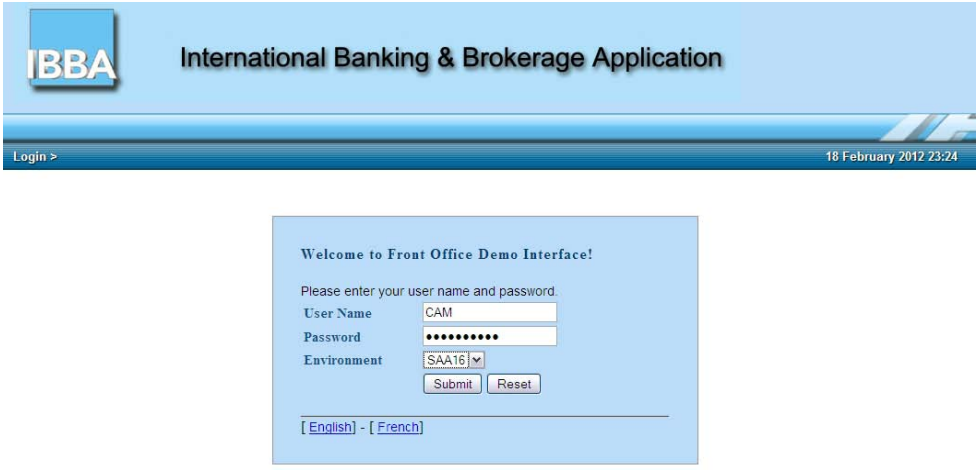

Fig. 5. Login page

For the authentication process it is implemented user group functionality. This means that the users are pre-defined in the database system based on the scope of their work. For instance some users have abilities to approve transactions for a certain department of the bank (i.e. Vault, Securities, Coupons and so on). If the authentication fails an error message is thrown back to the user. Also it is visible in the application server's log:

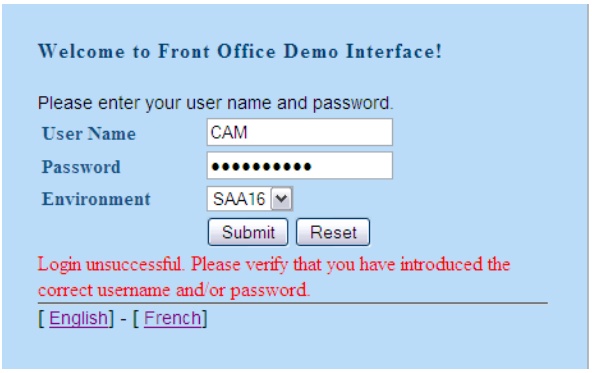

Fig. 6a. Login error

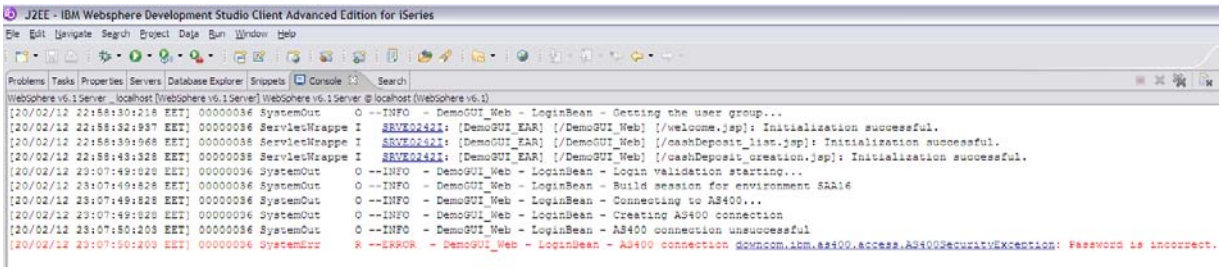

Fig. 6b. Error management on the server side

After the login succeeds, a welcome page is loaded. From the menu in the left side we can choose either to close the session or to go to the transactions directory. The user who logged in the system can access only the transaction he/she owns.

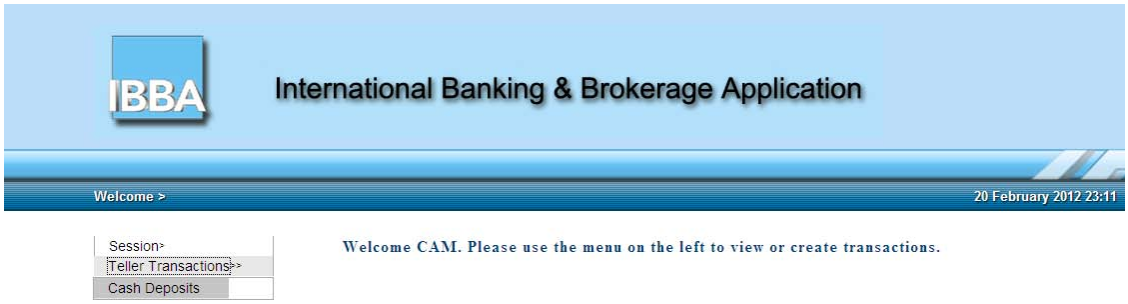

Fig. 7. Welcome page

The transaction directory contains valuable information, not only related to the business data (id, accounts, amounts, currency, depositors, creation date), but also to the technical data (status, author). All these data can be sorted. The Status column is very important at it is showing the actual processing of the transaction. For instance "Sent to JMS" means that the transaction is in the Message Queue but still not processed.

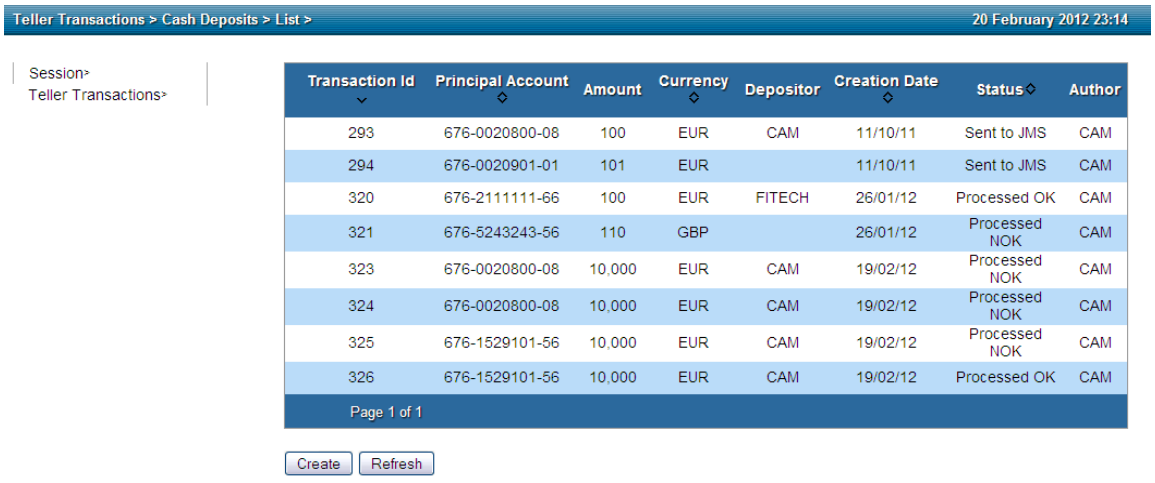

Fig. 8. Transactions directory

During the creation process the validation is made only for the fields (GUI validation). The business validation is made by the COBOL programs called by the server layer in a asynchronous mode.

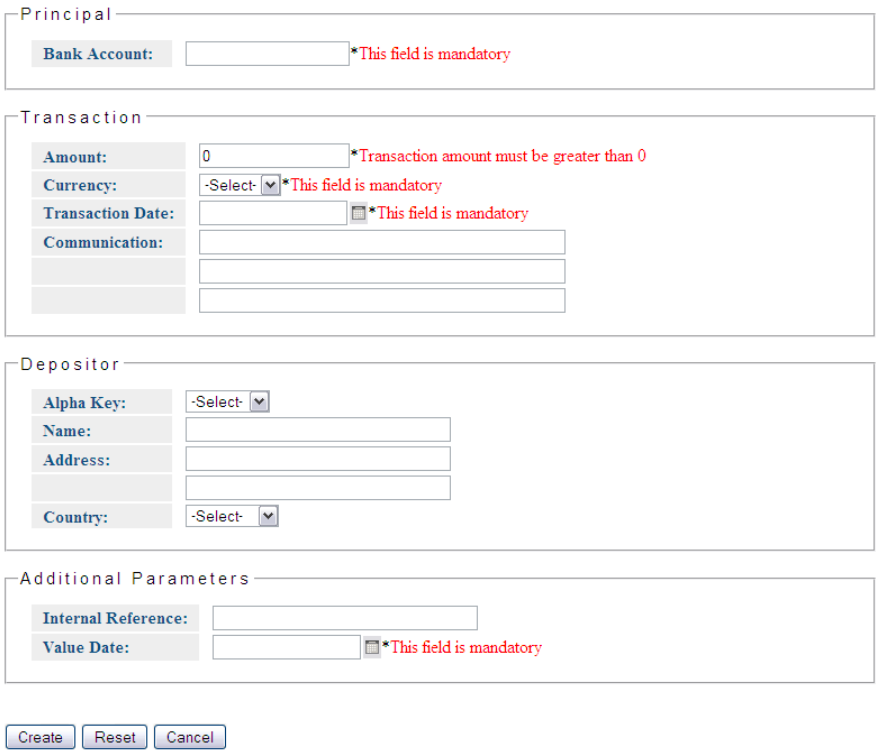

Fig. 9. GUI validation

If we introduce the data displayed below, we will obtain a business error message saying that the date should be less or equal with the back-office date:

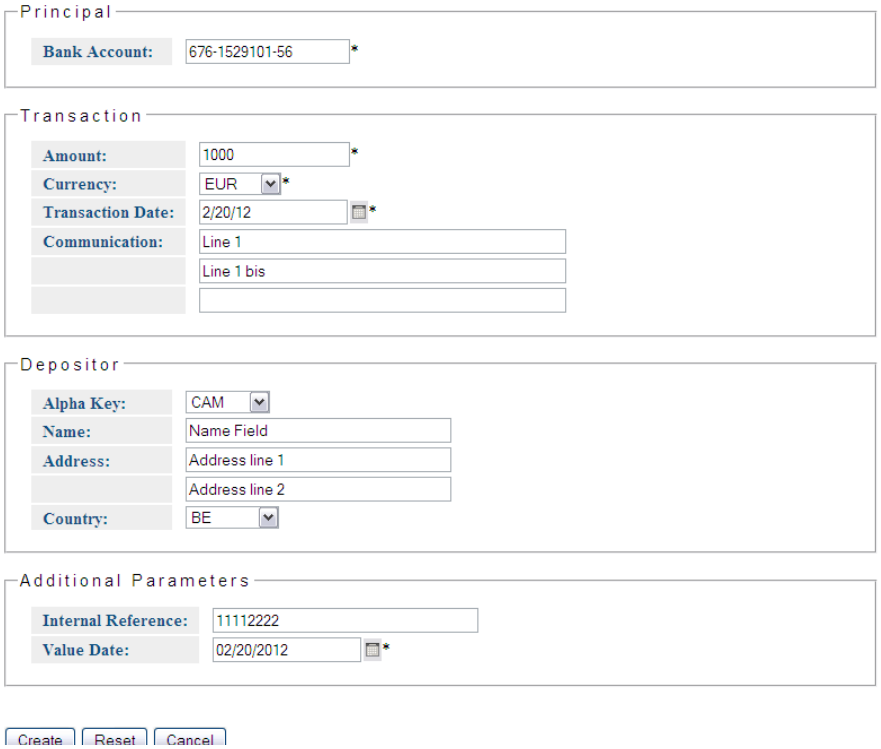

Fig. 10a. Creation data

Audit Data

Transaction Id:	327
User:	CAM
Creation Date:	20/02/12
Creation Time:	23:30
Status:	Processed NOK
Details:	Date must be < or = to the day date - TransactionDate = "20022012"

Additional Parameters

Internal Reference:	11112222
Value Date:	20/02/12

Fig. 10b. Audit data for the transaction launched in creation process

The server response contains XML messages which actually will be displayed into the GUI application. The results of the transaction creation process are to be found on the iSeries machine. For the transaction created above, below there are the correspondence within the system developed in COBOL on iSeries. Figure 11 is showing the directory of all transactions new created or already existing.

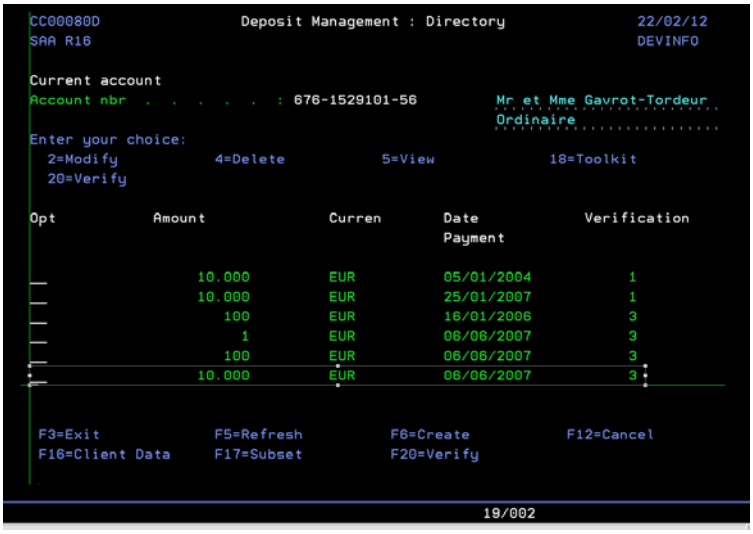

Fig. 11. Transaction directory

The Figures 12 and 13 are displaying the transaction's technical details, respectively the interface details (contains information related to the data received from Java).

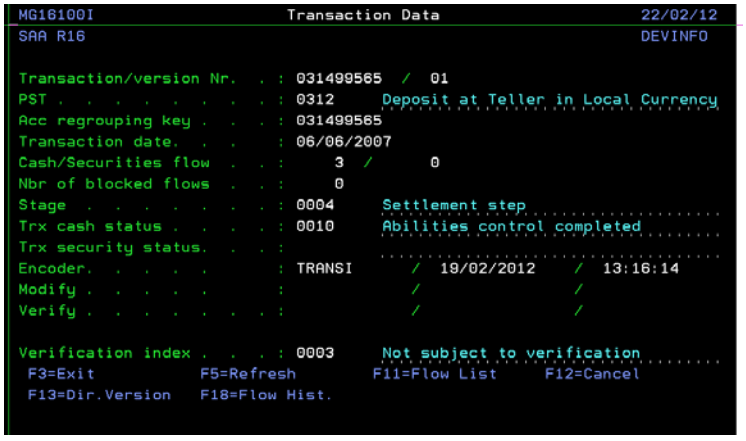

Fig. 13. Transaction data

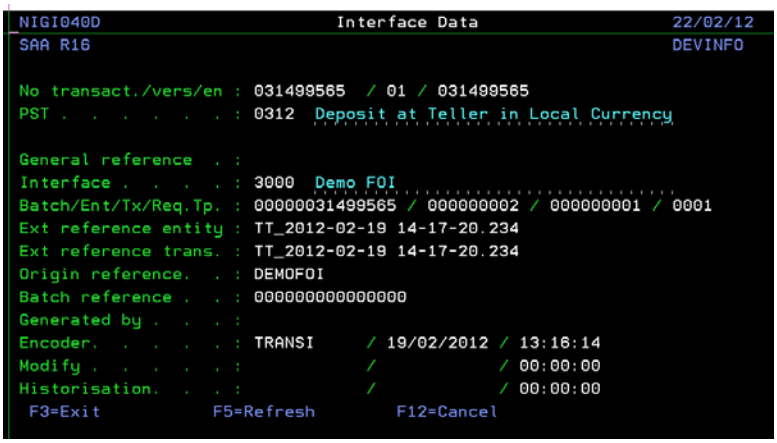

Fig. 14. Interface details

When calculating the *System performance (SP)* indicator we took in consideration the time used to create one transaction (from the moment the transaction was confirmed and the status changed from "Sent to JMS" into "Processed OK"). This lead to around 30 seconds duration. For the Cash Deposit product there will be only one transaction created at a time. The time to load the system is around 10 minutes for the iSeries session and the application server (alone 7 minutes to load). All these data are leading to an SP off 8,3%. This poor compared with the same indicator calculated for a system having only the iSeries display. It takes around 2 minutes to load the iSeries interface. The time to create a transaction is similar when using the JAVA interface. This leads to an SP off 33%. Out of calculation it seems that a wise idea is to keep the old interface.

Then we have to answer the question: what are other particularities which will require a change of the existing interface? The answer would be the marketability gain. As mentioned before we identified three attributes of a banking system which make it more attractive for the business: flexibility, connectivity and learning ability.

The three layers solution compared with the old interface is more flexible. For instance we are able to have more fields in one screen, while on the iSeries interface you are limited to maximum 27 rows per screen. On the GUI application we are able to add CSS (Cascading Style Sheets) in order to improve the visual effect of the application, while the COBOL screens have mostly the same formats.

From the connectivity point of view the GUI application is a Web application. Thus, it can be accessed from anywhere within the Internet (of course based on credentials managed within the Login module). The iSeries application can be accessed only from the iSeries server and using specific tools for connectivity security clearance (like TOX and SOCKS clients).

Since the Internet developed so rapidly, the Web applications are used at a large scale. This makes more easy the job for the solution we proposed, as it is developed taking in account the current technological features and tendencies.

4 Conclusions

Due to the rapid development of software and hardware technology the banking business tries to be in line with the new tendencies. Banks focuses on changing the legacy applications with newer software products. But how much will this process cost the banks?

The present article proposes a solution based on interconnecting legacy system with new interfaces through Web Services. Of course, we have to keep in mind the performance of the new system and also the marketability factors. These aspects influence the decision making process of the banks.

As we saw in the article, from the performance point of view a modernization solution can be slower than an old interface. But how long can we consider this aspect as an advantage? In banks the end-users are either people who worked for decades on legacy platforms, or employees with junior skills who are more used with new interfaces (like Web applications, even more with socialization networks). From this point of view a modern application has more value added for the bank business. Moreover, using the modern technologies we are able to do baking by Internet, Cloud computing or even more from a tablet PC or a smart phone. The presentation layer is such important as it never had been before the years 2000.

The solution proposed is based on an open architecture which makes the application to be scalable, flexible, easy to maintain. The business logic is maintained within the legacy system and only the presentation layer is changed. Thinking on a long term basis, the presented approach is helpful as the maintenance effort is lower due to the fact that the business logic is separated from the presentation layer. The presentation layer is enough flexible, thus, if we want to change the page style in order to look more attractive, it would be easier to do it. In this manner we don't have to change the business logic. The reverse is also true, meaning that when we change the business logic, we don't have to change the presentation layer.

The research will continue in the idea of modernizing legacy systems, by involving specific tools capable to do transformations of iSeries screens into Web pages. The result will be compared with the solution presented within this article in order to establish if we can use a combined architecture of these, or to use a single solution to do a generic modernization.

References

[1] J. Goldsmith, *Bank CIOs face legacy systems dilemma*, June 2007 http://www.silicon.com/management/cio-insights/2007/06/06/bank-cios-face-legacy-systems-dilemma-39167376/.

[2] M. W. Chowdhury and M. Z. Iqbal, "Integration of Legacy Systems in Software Architecture," *SAVCBS 2004 Specification and Verification of Component-Based Systems*, http://www.eecs.ucf.edu/SAVCBS/2004/ .

[3] R. C. Seacord, D. Plakosh and G. A. Lewis, "Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices," *Carnegie Mellon Software Engineering Institute*, ISBN 0-321-11884-7, February 2003.

[4] A. A. Almonaies, J. R. Cordy, T. R. Dean, *Legacy System Evolution towards Service-Oriented Architecture*, http://research.cs.queensu.ca/home/cordy/Papers/ACD_MigToSOA_SOAME10.pdf .

[5] S. C. Dorda, K.C. Wallnau, R.C. Seacord and J.E. Robert, "A Survey of Black-Box Modernization Approaches for Information Systems," *Proceeding ICSM '00 Proceedings of the International Conference on Software Maintenance (ICSM'00)*, IEEE Computer Society Washington, DC, USA ©2000, ISBN:0-7695-0753-0

[6] P. Newcomb and R. A. Doblar, "Automated Transformation of Legacy Systems," *CrossTalk: The Journal of Defense Software Engineering*, December 2001, https://www.softwarerevolution.com/v2/fyles

[7] C. M. Matei, "Services Used to Integrate Banking Front Office Applications," *Open Source Science Journal*, Vol. 2, No. 3, 2010 http://opensourcejournal.ro/2010-Volume02/number03/paper002-fullpaper.pdf

Constantin Marian MATEI has graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 2005. He holds a Masters degree in IT Systems for Managing Economy Process (SIMPRE). He held positions as technical team leader, software configuration controller and software developer. Currently, he is project manager in banking field and Mainframe Prac-

tice Area Leader within IBM Romania. Also, he is PhD student enrolled in a PhD programme on Computer Science in Economics. His key skills refer to the following: project management, leadership, business management, banking, coaching and mentoring, IT&C specialist in business oriented software development and maintenance.