Migration to Multitier Application Architecture: A Case Study

Mirta Baranović, Ladislav Mačkala
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, HR-10000 Zagreb, Croatia
mirta.baranovic@fer.hr, ladislav.mackala@fer.hr
and
Denis Kranjčec
University Computing Centre (SRCE), Zagreb
Marohnićeva bb, HR-10000 Zagreb, Croatia
denis.kranjcec@srce.hr

Abstract. *During the migration of an application to a new system, it is usually desirable to start using new tools while keeping existing models. However, challenges brought by new technologies often lead to modifications in architecture of the existing system.*

Two similar applications with different architecture not sharing the same code base present a possible point of improvement.

Development of a new multi-tier application based on open standards was an attempt to solve problems brought by old architecture. Special attention was given to implementation of specific feature of existing applications. Quality solution was achieved using stable platform and design patterns.

Keywords. Multitier Architecture, Application Migration, Design Patterns.

1. Introduction

Computerization of the system of high education in the Republic of Croatia [2] is a project that started at the end of 1998. It is large in size and complexity and requires a number of years to be completed. The project is based upon the experience gathered since 1992 on a similar project implemented on the Faculty of Electrical Engineering and Computing (FER) of the University in Zagreb [1]. This implementation covers a complete evidence of syllabuses, the educational program fulfillment, course scheduling and all the students' activities, starting from the entrance examination, education and examinations to the graduation thesis and issuing of graduation certificate (diploma) and its supplement with the evidence of the student's performance.

Devices like kiosks with touch screen monitors (Studomat), Internet access (Studonet) and voice service (Studophone) have been introduced to facilitate the administrative procedures to all the involved parties. The system was based on the Informix RDBMS with applications written in INFORMIX-4GL. Software for Studomat was written in Visual Basic, while software for Studonet was created later using CGI, HTML and JavaScript technologies.

Migration to modern technologies is achieved by design of a completely new system. Character based user interfaces are replaced with graphical ones, Java programming language is used, and so is JavaScript in combination with HTML and XML technologies. Although, the first goal of migration to new technologies is to keep existing architecture and to add a small number of new features, most often it is impossible to resist the challenges that new technology brings. Modified architecture of this new system combined with aforementioned technologies will allow, among other things, integration of Studomat and Studonet applications that will simplify the code maintenance and improve the scalability of the system.

2. Old Studomat/Studonet application

Applications that had been developed in the old system enabled the students' interaction with the Faculty administration. Information became more available to the students and the administrative tasks became easier to achieve.

2.1. Functionality

The basic idea was to install the application on a kiosk-like device with a touch screen and a printer and to put such a kiosk in a place with public access at faculties, thus granting the

students 24 hours availability and consequently greater independence of Student Service working hours.

The Studomat offered following options to the students:

- Applying for an exam or canceling that application, viewing the results of written exams
- Enrolling in a new academic year: choosing courses for senior students and choosing the mentor to guide the graduation thesis
- Printing of various documents and grade lists
- Notifying students about matters of interest via e-mail
- Presenting the summary of all relevant information pertaining to the student

An improvement of the system was the development of a web application named Studonet that allowed students to accomplish simple, but often required tasks using Internet from the comfort of their own room, without having to come to their faculties.

2.2. Model

Entire information system was based on a relational DBMS. Decisions about application architecture design were made by analyzing requirements placed on application.

One part of mentioned functions of the system such as enrolling in a new academic year and choosing courses encompasses very demanding interaction between a user and the application. This interaction consists of navigation through a complex menu tree because it is based on an extremely complicated business rules system. Transaction complexity level is high and includes interaction with 30 or more database tables.

One Studomat (kiosk with installed application) can be connected to several printers. Then it controls them, checks their status and evenly distributes the workload during printing. Documents to be printed are dynamically created because they consist of database data and static text.

Considering all mentioned above, the only choice for Studomat application was two-tier application architecture (also known as client-server). Known flaws of two-tier architecture had little significance because of the rather small number of Studomats.

Requirements placed on Studonet were very different from those placed on Studomat because intended functionality consisted of simple menus with a minimal set of business rules on the client with emphasis on its availability and on small requirements while connecting the client through web interface. Hence, Studonet architecture is a variation of three-tier architecture most commonly used by web applications.

Studomat and Studonet shared some of their functionality so it made sense to base them on the same part of business rules.

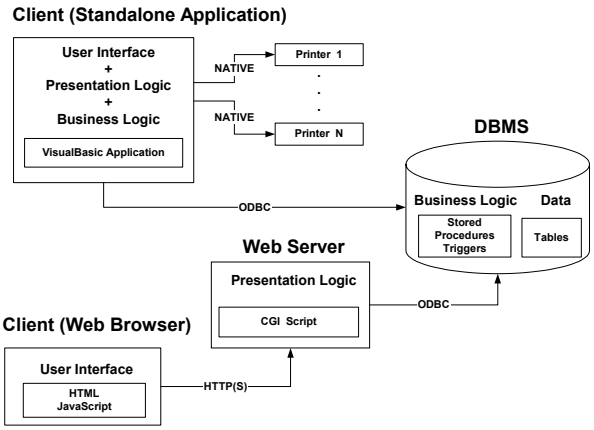

Figure 1. Studomat and Studonet

2.3. Implementation

Studomat is a standalone application written in MS Visual Basic using ODBC for database access and native printer access. Visual Basic and its development environment made the design of rich GUI possible, and full integration with Windows operating system allowed good printer control. Documents were in MS Word format.

Analysis yielded that one Studomat covered the needs of roughly 1000 students, which, for FER, meant 3-4 Studomats and 2 printers. Overload occurred several times a year when a large number of students wanted to accomplish similar actions in the same time. The server part of Studonet was created using CGI technology that was the only available server-side technology at the time of development. The client part of Studonet was created using a combination of HTML and JavaScript language which is sufficient for design of a simple user interface.

Studonet later appeared to be rather sensitive to workload because of limited scalability of CGI technology because it starts separate processes for each new client.

3. New Studomat application

During the development of the new information system that is expected to include all institutions of higher education in Croatia, a new version of Studomat had to be developed as well. In the analysis phase all problems discovered during the development and exploitation of the old system and requirements placed on the new system were taken into account.

3.1. Motivation

The new application is supposed to contain all of Studonet's and Studomat's functionality and also some additional enhancements. Similarly to the old system, the new application should have two different "allotropic modifications" based on old Studomat and Studonet with similar set of functions and requirements.

The first idea was to take the complete code base of the old Studomat/Studonet and, after some modifications, use it in the new system. Another solution was to write the whole programming code from scratch using the insight and experience gained during development of the old Studomat.

From performed analysis it was concluded that the greatest potential problem of this new system would be the different code base and low scalability.

The part of business rules that was located in the database was a solid basis, but each form of the application contained a part of business logic and the complete presentation logic. That made the debugging and adding of new features a difficult and error prone task. Adding support for new technologies, that are yet to come (mobile clients, web services), would be expensive and the problem of code maintenance would constantly grow.

Scalability has become important because to the contrary of the old system (4000 users), the new system has to be able to handle continuous growth of the number of users up to about 100000 when all institutions enter the system. Several Studomat-kiosks, realized as typical fat clients that keep connection to the database open all the time would be substituted by about 150 of Studomat-kiosks that continually overload the database. Web load would also increase significantly and more than CGI technology could support.

Maximum data security is an aspect that cannot be neglected because user's confidential data are transferred over web.

Since a number of applications were already developed for the new system, it would be desirable, if possible, to use the existing programming code. Existing applications were developed using our own framework [6] designed for development of standalone applications, which could also be used.

Intensive use of design patterns would make the interchange of ideas within the development team easier and the maintenance of programming code less demanding.

The choice of technologies should be, as much as possible, based on open systems and industrial standards rather than on proprietary technologies in order to maximally avoid vendor lock-in.

3.2. Choice of technologies

Starting from the requirements for open architecture, industrial standards and object-oriented programming language, Java appeared to be the optimal choice, taking into account that applications for the new system and the framework they are based on were written in Java 2 Standard Edition.

As a logical extension Java 2 Enterprise Edition was considered. The Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multi-tier enterprise applications. J2EE simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically, without complex programming.

All J2EE technologies have proved themselves on the market as stable, reliable and complete which was also an important factor in choosing the development platform.

The choice of technology for user interface design was between HTML/JavaScript combination and Java applets. Although Java applets appears as a natural combination with Java as server technology, still HTML/ JavaScript seems to be a better choice, mainly because of small requirements on the client, but also because of maturity of this standard. HTML definitely brings a certain trade-off because it is inappropriate for design of complex user interfaces.

Document printing introduced an additional dose of complexity because the old solution was completely inadequate for the new platform. XML standard together with belonging technologies (XSL, XSL-FO) and PDF as an end document format is a good choice.

3.3. Model

The choice of application architecture is a strategic decision in the application development process. Based on all available parameters, multi-tier architecture consisting of a thin client, a middle-tier with presentation logic, a middle-tier with business rules and a database was chosen [4].

The client controls data presentation for the user and the capturing of user's actions.

The presentation layer was realized as a set of components that are executed on a web server. The standard design pattern for this layer is a Model-View-Controller design pattern [5].

The component (Java Servlet) that represents the front controller is the only input into the application. When a request arrives, as first, the security checks are completed and then a second controller component to process the received request is chosen. The controller component takes over the request and determines which methods of the model and with what arguments should be called.

The model is represented by a Java Enterprise Bean component which is located in the business logic layer and on whom different operations are performed.

After the state of the model has been changed, the controller receives data and passes them to the View which is represented by a JavaServer Pages component, which displays the new state of the model to the user. The JavaServer Pages component receives only the necessary data for user interface generation through custom tag library in HTML and JavaScript.

In Enterprise JavaBean components a part of the business rules and all of the database access code are implemented. The presentation layer controller uses them as models, calling their methods thus indirectly manipulating data in the data layer.

The business logic layer was created using Session Façade design pattern [3]. This provides great independence of the presentation layer from business logic. Calling of just one remote method performs complete business logic and it consequently minimizes both, the network traffic and the number of network calls that introduce latency. This solution conceals internal model structure from the client, and changes in internal structure of the model do not imply changes in the client.

Data exchanged between the presentation layer and the business logic layer frequently are not of simple data types but rather complex structures because they represent data extracted from multiple tables in the database. To avoid remote procedure call overhead during access to and manipulation of data, Data Transfer Objects design pattern [3] was used. When a client requests data, it calls a method in the business logic layer who then fetches all the required data from the data layer and creates a Data Transfer Object, Java class which contains and encapsulates bulk data in one network transportable bundle. The same principle is used when a client wants to create, modify or erase data, except in that case the client creates a Data Transfer Object and sends it to the EJB layer using one network call.

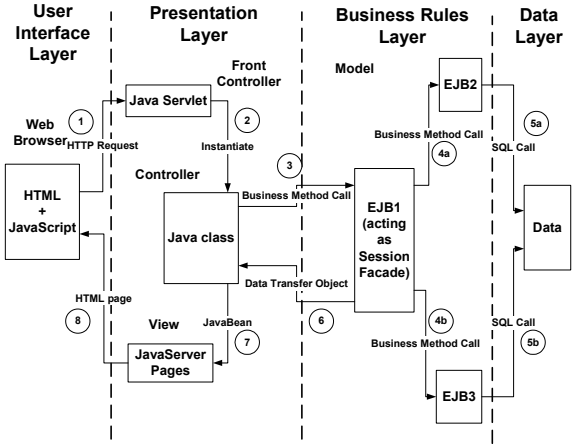

Figure 2. Architecture details

The data layer is represented by relational database.

The issue of document printing is solved using a separate Java application acting as the server to which Enterprise JavaBeans are connected as clients and are sending their printing requests. These requests contain only the basic data for the PrinterServer application to receive the requests from the EJB component using RMI protocol. The PrinterServer then extracts data from the database, generates an XML document, by a series of transformations brings it into the final form (PDF) and prints it on one of the connected printers, performing

automatically load-balancing between connected printers.

E-mail sending is realized using a message queue. If a student performs a certain action on the Studomat, causing results about which s/he wants to be notified, the EJB sends a message to the application for notification using the JMS which then generates an e-mail and sends it to the user (student). The Studomat does not wait for the message to be sent because message sending proceeds asynchronously.

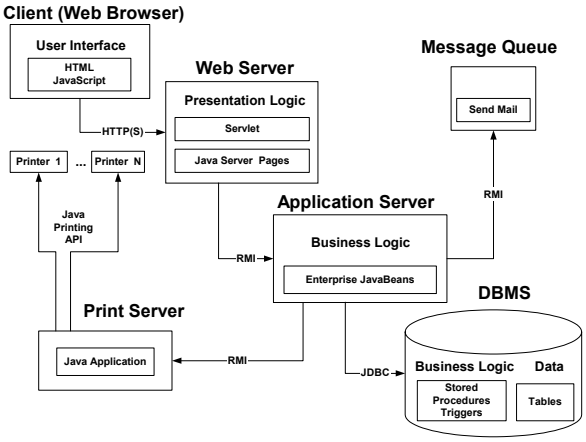

Figure 3. Architecture overview

3.4. Implementation

Throughout the implementation of each of the aforementioned application layers, the choices made were constantly questioned and better solutions to the encountered problems were searched for. The general idea was to use as many proved concepts as possible.

The choice of HTML and JavaScript in user interface design brought some anticipated problems with different implementations of these standards in different web browsers. Differences between HTML implementations are minimal, while the story with JavaScript is much worse, so in the end, a separate JavaScript code had to be written for each supported web browser.

Special attention was given to the design of user interface with the ability to display data in a clear and unambiguous way and to enable meaningful and precise interaction with the application.

Each segment of this user interface is completely suitable for the touch screen on Studomat kiosks: the menus are placed on one side to avoid moving hands over the screen, and the button size corresponds to the size of the fingers. During logging on to the system, virtual keyboard is displayed.

There are some small differences in the interface and some big differences in the functionality when the application is not used from an Internet kiosk. All options demanding interaction with other systems are disabled, e.g. documents are printed on a printer situated at the faculty and the student must be there to pick it up.

Due to security and nondisclosure of confidential data, special care had to be taken to prevent that the presented web pages are cached on the computer using Studomat application. This was done to avoid situations where a student uses the Studomat in some Internet cafe and all viewed pages remain cached on the Internet cafe computer. Real life experience has shown that web browsers do not comply with settings selected in the HTTP protocol and the viewed web pages, despite explicitly instructing otherwise, can be cached on local hard disks. The only solution that really disables caching is to direct all requests through servlets.

At first, a complete implementation of business rules within the EJB layer was planned. However, after initial testing, it became clear that using the Entity EJBs, due to the extremely complex database structure, is sometimes very demanding on computer resources, what results in long response times and it was decided that this solution is not acceptable.

The next introduced and tested solution was a partial implementation of business rules in data layer in the form of stored procedures. Due to the mentioned transaction complexity, these procedures were written in a modular fashion to avoid code duplication and to simplify maintenance. At first, this solution seemed as a good one, but after being tested with a bigger workload, it resulted in DBMS overload and congestion because of intensive use of stored procedures. In a complex, real world example, a call of procedure that determines which courses a student can choose resulted with 1500 successive procedure calls. For a simulation of 15 students choosing courses concurrently and repeatedly, average response time was 109 s. A simulation with 30 students fully congested the database. Afterwards it was concluded that it was necessary to reduce the modularity in stored procedures. Now, in the same example, the number of procedure calls was reduced to 400 and average time decreased to 10 s (for 15 students). Increasing of number of students didn't increase average time significantly. Other operations performed on the Studomat aren't so

demandable and can be easily ignored in performance measuring. The maximum number of students choosing courses simultaneously is estimated to be about 150 (when system is fully implemented).

Also some temporary results from stored procedures are cached in EJB layer for later use.

After these alterations had been made, the response time dropped significantly, and the application was able to bear a substantially larger number of users without DBMS congestion.

3.5. Achieved goals

Our aspiration to unique code base demanded abandoning the existing model of "allotropic modifications" with different architecture in favor of a single unified application with all business rules and all the features, achieving different functionality by different (or at least modified) user interface and presentation logic. The introducing of a new type of client is much easier because it is only necessary to program a lesser part of the presentation logic for a desired type of client, while most of the existing presentation logic and the complete business logic are reusable. This minimizes and centralizes code maintenance, makes addition of new features easier, decreases the number of bugs and consequently decreases the cost of the entire system.

The scalability and performance problems are very well solved using distributed application concept which is the basis for multi-tier application architecture. The presentation logic and the business rules are distinctly separated in this model and are located in separate components.

Using the PrintServer concept for printer control completely accomplishes the goal, because it enables the necessary control over printers, and raises its scalability by allowing printers and print servers to be added depending on workload.

4. Conclusion

Strategic decisions made during development of a new application reflect in the choice of application architecture and technology. By migrating from an application having two forms of similar functionality but different code base and proprietary technology to the application with scalable, multi-tier architecture having a unique code base, based on open technology and industrial standards, all the set goals have been achieved, but with some compromises.

Scalability was accomplished using multi-tier architecture; the unified code base decreases maintenance costs and bug problems and makes the addition of new features easier. The choice of Java platform and Linux operating system resulted in a stable system, while the quality and readability of the programming code and the possibility of verification of founded concepts was achieved by applying various design patterns. The security was realized using several elements in all the layers. Implementation of all the necessary features has been accomplished.

During the system development, certain concepts had to be altered and adapted due to poor performance caused by a distinctive situation. The database appeared to be the greatest problem due to very complex transactions and occasional very complex data extraction. Much of the time was spent finding the optimal model of implementation of business rules, but, in the end, the results were more than satisfactory.

5. References

[1] D. Kalpić, J. Anzil, V. Dumanić, H. Zoković: Student Administration System, Proceedings of the 15th International Conference on Information Technology Interfaces, Pula, Croatia, June 15-18, 1993, pp. 123-128

[2] D. Kalpić, M. Baranović, V. Mornar, S. Krajcar: Development of an Integral University Management System, Proceedings of the International Conference on Systems Engineering, Communications and Information Technologies, Punta Arenas, Chile, April 16-19, 2001-05-23

[3] F. Marinescu: EJB Design Patterns, Wiley Computer Publishing, 2002

[4] N. Kassem and the Enterprise Team: Designing Enterprise Applications with Java 2 Platform, Enterprise Edition, Sun Microsystems, 2000

[5] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Design Patterns, Elements of Reusable Object-Oriented Systems, Addison-Wesley, 1998

[6] M. Baranović, S. Zakošek, L. Mačkala: Creating Database-aware Java Classes Based on Extended Data Dictionary and Abstract Classes, SCI2001/ISAS 2001, Orlando, 2001