BUILDING A LARGE-SCALE INFORMATION SYSTEM FOR THE EDUCATION SECTOR: A PROJECT EXPERIENCE

Pawel Gruszczynski, Bernard Lange, Michal Maciejewski, Cezary Mazurek,
Krystian Nowak, Stanislaw Osinski, Maciej Stroinski, Andrzej Swedrzynski
Poznan Supercomputing and Networking Center
ul. Noskowskiego 10, 61-704 Poznan, Poland

Keywords: Large-scale, distributed system, education, public sector.

Abstract: Implementing a large-scale information system for the education sector involves a number of engineering challenges, such as high security and correctness standards imposed by the law, a large and varied group of end users, or fault-tolerance and a distributed character of processing. In this paper we report on our experiences with building and deploying a senior high school recruitment system for five major cities in Poland. We discuss system architecture and design decisions, such as thin vs. rich client, on-line vs. off-line processing, dedicated network vs. Internet environment. We also analyse potential problems our present approach may cause in the future.

1 INTRODUCTION

With more and more universal access to the broadband network infrastructure, advanced services and applications can now be available not only to scientific and business communities, but also to local administration and the society in general. The trend towards public sector applications of new technologies is clearly visible in the objectives of such programmes as eEurope (European Comission, 2002), Large Scale Networking in the U.S. (NITRD, 2004) and PIONIER in Poland (Weglarz et al., 2000).

One of the recently introduced information systems for the city od Poznan (Poland) was the High School On-line Admission System called *Nabor*[1]. Its introduction in 2003 not only significantly reduced the time and effort needed for the whole procedure, but also made the process less stressful for the candidates and their parents. Although *Nabor* is based on rules and procedures specific to Poland, we feel that both the idea of on-line admission (Capita, 2004) and the technical solutions we used are relevant in a more general context.

In this paper we report on our experiences with designing and implementing *Nabor*. Section 2 provides background information on Poznan's high school admission procedures and highlights the problems our software solved. In section 3 we describe the architecture of *Nabor* placing special emphasis on the design trade-offs we had to evaluate. Section 4 provides some implementation details and discusses the implications our design decisions may have in the future. Finally, section 5 concludes the paper.

[1]*Nabor* is the Polish word for *admission*

2 MOTIVATION

The key characteristic of senior high school admission in the city of Poznan is that the process is highly competitive. Some of over 120 Poznan's high schools are always on demand and must every year reject a great majority of their applicants, while the less popular schools can barely fill the places they offer. Therefore, each candidate is allowed to apply to more than one school, which increases the chances of getting to a school matching his or her standards. For the purposes of the admission process, all selection criteria, such as candidates' marks, sporting or artistic achievements, are aggregated to a single numerical value ranging from 0 to 200. In this way, each school orders all its applicants by the aggregated mark and fills all its places with the highest-scoring candidates.

However fair and simple the above procedure may seem at the first sight, the lack of coordination between schools can cause many problems, the most severe of them being the 'blocking problem'. This is

Gruszczynski P., Lange B., Maciejewski M., Mazurek C., Nowak K., Osinski S., Stroinski M. and Swedrzynski A. (2005).
BUILDING A LARGE-SCALE INFORMATION SYSTEM FOR THE EDUCATION SECTOR: A PROJECT EXPERIENCE.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 145-150
DOI: 10.5220/0002549301450150
Copyright © SciTePress

where the highest-scoring candidates apply and get admitted to a large number of schools, but still defer the decision as to which of them they finally want to go to. Such candidates 'block' places that could have been taken by the medium-scoring applicants, who, in turn, 'block' places for the candidates with the lowest marks. When a highest-scoring candidate makes the final decision and thus frees the 'blocked' places, the rest of the candidates may want to change their previous decisions and move to more prestigious schools. In practice, because of the 'blocking problem', the 'manual' admission would every year boil down to a series of cascading decision changes and endless updating of admission lists, which was both time-consuming and stressful.

The main objective of *Nabor* was therefore to provide high enough a level of co-ordination between all high schools in the city as to reduce the time and effort involved in the admission process. The key idea was to gather all necessary data (e.g. candidates' marks, admission limits set by schools) in a central database and design an algorithm that would generate admission lists for all schools in Poznan in one step. This, however, was to be achieved at the cost of unification and minor modifications to the admission procedures.

3 SYSTEM DESIGN AND ARCHITECTURE

3.1 Requirements

Being largely influenced by legal regulations, the design and architecture of *Nabor* had to meet a number of requirements. Below we summarise the most important of them:

- *High security standards*. A great majority of data processed by *Nabor* (e.g. candidate's marks) was of a confidential character and had to be carefully protected from unauthorised access and modification. Also, the system had to be designed in such a way as to minimise the possibility of manipulating the admission results and to enable tracing possible manipulation attempts.
- *Fault-tolerance*. The high school admission procedures divide the whole process into several phases and impose strict deadlines on each of them. For example, one of the final phases is entering the candidates' marks to the system, which must be completed by all schools within a time span of two or three days. Therefore, our software had to be able to operate even in case of a temporary failure of the central database or the network infrastructure.
- *Correctness of results*. The future of thousands of Poznan's pupils depended on the admission results generated by *Nabor*. For this reason, every effort had to be made to ensure that the results were 100% correct.
- *Varied user group*. An important characteristic of *Nabor* was also the fact that it required active involvement of all parties of the admission process. High school candidates used the system to find out about the offered schools, fill in and print out the application form. School administration used *Nabor* to enter the candidates' data and download the admission lists once the admission had been closed. Finally, for the local education authorities the system provided a range of analytical reports, which aided global planning.

3.2 Design decisions

Deciding on the architecture of *Nabor* required answering a number of fundamental design questions, such as whether to create a rich- or web-client application or whether to adopt the on-line or the off-line processing model. Below we evaluate the trade-offs resulting from the three design questions we found most important.

3.2.1 Rich client vs. web client

In the rich client application model (also referred to as thick or fat client model) the major part of data processing is performed by software deployed on the client's workstation. The software is usually a standalone GUI (Graphical User Interface) application, which contacts the central server only when retrieving or updating business data. Below we summarise the advantages and disadvantages of the rich client model in the context of *Nabor*.

Rich client advantages

- *Responsiveness and usability*. Compared to web applications, GUIs offer much shorter response time and are easier to usability-tune (keyboard shortcuts and navigation, accessibility options, etc.). These are very important factors for systems which like *Nabor* involve entering massive amounts of data over a short period of time.
- *Familiarity*. If the GUI application is consistent with the host Operating System's look and feel, it can be a great learnability advantage. With a varied end users group, good learnability of the software may shorten the training period.
- *Data caching*. A standalone client can more easily cache data that a web application would have to repeatedly fetch from the server. An example of such data, a so called system dictionary, can be

a list of the city's districts or schools. More importantly, having a local cache, the rich client can operate even when a temporary failure of the server or network occurs.

- *Customisable security options*. Having more control over the communication between the server and the rich client application makes it possible to implement more advanced or non-standard data security schemes.
- *Server load*. Compared to web clients, the GUI applications create much lower server load.

Rich client disadvantages

- *Data serialisation*. We feel that the most serious problem with rich client applications is the need for business data serialisation and deserialisation. With data models containing lots of master-detail relationships and high security requirements the implementation of the business data communication layer becomes a nontrivial task.
- *The need for installation*. Usually, before the rich client application can be used, it needs to be installed and configured on the user's workstation, which may be a source of external software dependencies and incompatibilities.
- *Possible platform-dependence*. With certain implementation technologies, a GUI application may become platform-dependant. This creates an additional burden of testing on all configuration the software is meant to support.
- *Software updates*. A standalone application is more difficult to update and maintain. Although there exist frameworks, such as Java Web Start (Sun Microsystems, 2004d), Rich Client Platform (Eclipse Foundation, 2004) or Smart Client (Microsoft, 2004) that automate the process of update checking and downloading, for the majority of end users the process will not be perfectly transparent and unobtrusive.
- *Secure storage*. To meet the required security standards, a standalone GUI application must implement its business data cache in such a way as to eliminate the possibility of unauthorised access or modification.

The central idea of the web application model is that all processing of business data takes place on the server side, the interaction with the client relying only on a standard web browser and the HTTP protocol. Below we highlight the advantages and disadvantages of the web model from the perspective of the *Nabor* system.

Web client advantages

- *No need for installation*. The only required software on the client side is a standard-compliant web browser. This eliminates the need for software distribution, installation and the possible dependencies.
- *On-the-fly updates*. A web application is updated only on the server side, which makes the process totally transparent and unobtrusive for the end users.
- *No need for data serialisation*. In a web application all processing takes place on the server side and the only data exchanged with the client are HTML/XML/HTTP streams.
- *Platform-independence*. Because the only required software on the client side is a web browser, it is much easier to achieve platform-independence of a web application.

Web client disadvantages

- *Poor responsiveness*. As all business data processing occurs on the server side, and because the user interface is based on HTML pages and forms, compared to the rich client, responsiveness of a web application is rather poor.
- *Usability issues*. Compared to GUI, an HTML-based user interface is more difficult to work with for the end users, because of e.g. only a limited support for keyboard shortcuts.
- *On-line operation*. A web application requires that the server connection be present at all times. In case of a failure of the server or the network infrastructure, the web application cannot function at all.
- *Server load*. Compared to the rich client model, a web application generates a much higher load on the server side.

3.2.2 Off-line processing vs. on-line processing

In the off-line processing model, the rich client application[2] normally stays disconnected from the server and performs all operations on the locally cached data. A connection is established, usually for a limited period of time, only to synchronise the stored data with the central database. This architecture had the following advantages and disadvantages from the perspective of the *Nabor* system.

[2]implementing a web application in the off-line processing model seems impractical

Off-line processing advantages

- *Permanent server link not required*. For the end users still using dial-up connections, the off-line processing model will be a less costly option.
- *Fault-tolerance*. Because it is very natural for an off-line application to operate without server connection, the tolerance to server and network failures is in a sense built-in into its architecture.

Off-line processing disadvantages

- *Data synchronisation*. Part of the off-line processing model must be an algorithm for synchronising business data and resolving possible conflicts, e.g. concurrent modifications of the same data. On many occasions, conflict resolution will require the end user's active involvement and decisions.
- *Security infrastructure*. Secure storage of cached data will require additional development and testing effort.

The primary assumption behind the on-line processing model is that the client application (implemented using either the rich or the web model) for normal operation requires a permanent connection with the server. Below we analyse the implications of the on-line architecture for the *Nabor* software.

On-line processing advantages

- *No need for data synchronisation*. Changes made to the data are immediately reflected on the server and therefore the problems of conflict resolution are avoided.
- *Up-to-date data*. In the on-line model the global and up-to-date state of the data is always available to all clients. This greatly simplifies the implementation of operations that depend on the global state of the database.

On-line processing disadvantages

- *Permanent server link required*. The on-line processing model requires that a permanent and possibly broadband connection to the server be available to all clients, which may turn out costly for some end users.
- *Possible security threats*. An on-line rich client application would usually require direct access to the system's database, e.g. through some Object to Relational Model mapping layer. This would greatly simplify the development, but also increase the security risks.

3.2.3 Dedicated network vs. Internet

In order to use the system in the dedicated network setting, all clients would need to connect to a separate network using e.g. a dial-up service. Clearly, this solution incurs additional costs related to setting up dial-up access points on the server side and providing modems for the end users. On the other hand, in a dedicated network it is much easier to protect the server and the client software from external attacks. An alternative to the dedicated network is the Internet, where the additional infrastructure expenses are avoided at the cost of higher security risks.

3.3 SYSTEM ARCHITECTURE

Having analysed all available options we have decided that the architecture of *Nabor* should be based around the off-line rich client model and Internet communication. We have therefore traded off the increased complexity and possibly higher development cost for better usability and fault-tolerance of the system as a whole.

The reasons for choosing the off-line rich client model were twofold. First of all, this model would allow us to meet the most important design requirements — the high standard of data security and fault-tolerance. To this end, we have implemented a sophisticated off-line business data transfer layer with encryption and digital signature support (see section 4). Another incentive to use the off-line processing model was the fact that due to the specific character of Poznan's high school admission procedures, many of the general data synchronisation problems would not at all occur in *Nabor*[3].

The only exception to the off-line rich client model was the web site for high school candidates, where they could fill in the application form and check the admission results after the admission lists had been released. For this service we wanted to avoid the overhead of installation and configuration of the rich client application. For the security reasons, the web server operated on a separate read-only data source that was periodically synchronised with the system's main database and contained results only for those candidates who decided to open the electronic data access channels (about 87% of the candidates decided to do so).

Figure 1 summarises our discussion on the overall architecture of *Nabor*. High school administration used rich client GUI applications to enter candidates'

[3]For example, only one of the schools a candidate is applying to has the right to enter and further modify the candidate's data. Having noticed and exploited such assumption, we can avoid the majority of conflict resolution problems.

details, marks etc. to the system and to retrieve admission lists, while local education authorities used rich client GUI applications to retrieve analytical reports. Finally, high school candidates could find out about the admission results using a web application or a text message sent to a mobile phone.

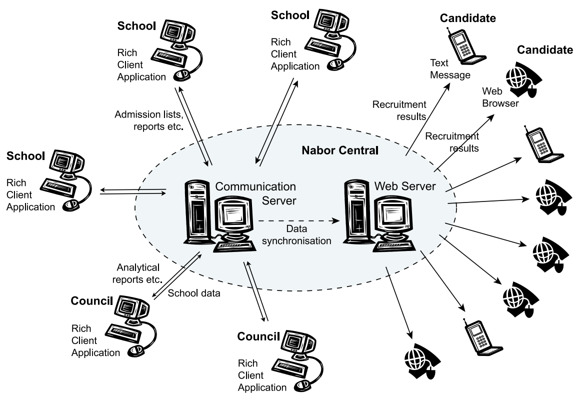

Figure 1: The architecture of *Nabor of* Nabor

4 IMPLEMENTATION AND EVALUATION

4.1 Implementation

We have decided to implement the *Nabor* system using Java technology. Employing Java Swing (Sun Microsystems, 2004b) on the client side would make the GUI application platform-independent, while using non-proprietary Java technologies on the server side would reduce the cost of the whole undertaking.

Noteworthy is how we implemented the secure offline storage and transport layer. To this end we developed the Offline Business Objects (OBO) framework, whose primary motivation was to avoid duplication of code and logic among various layers of the system. Because of the high security and fault-tolerance requirements, every operation performed by the off-line client application had to be:

- packaged into encrypted and digitally signed units called packets,
- logged both on the client and server side to enable automated recovery after a failure
- authenticated and authorized,
- cached on the client side in an encrypted form,
- incrementally synchronized with the server.

The key observation here is that the above requirements specify some mechanisms underlying the communication between client and server rather than the application business logic. Manual implementation of all these mechanisms in each of 25 different types of business objects would be tedious, error-prone and would severely duplicate code. On the other hand, none of the existing Object to Relational Model mappings, such as Hibernate (Bauer and King, 2004), provided off-line operation mode at that time[4]. We have therefore decided to create the Offline Business Objects Framework, which would streamline and automate the process of generating *Nabor's* business object transport layer.

Table 1: *Nabor* production source code metrics

	Hand-coded	*Generated*	*TOTAL*
KLOC	92	89	181
Classes	1956	650	2606

An essential part of OBO is a code generator. It takes an XML-based declarative specification of business classes and generates code which implements all the required security mechanisms and additionally serves as a persistence layer for business objects on both client[5] and server side. The generated code has many extension points which can be used by a developer to implement the specific behaviour of different business objects. One advantage of the extension points approach is that the automatically generated code need not be modified and can be regenerated without losing or overwriting the customisations (Herrington, 2003).

The Java implementation of the Offline Business Objects framework was based on a number of non-proprietary technologies, such as Torque (Apache Foundation, 2004a), Velocity (Apache Foundation, 2004b), Java Cryptography Extension (Sun Microsystems, 2004a) and Java Secure Socket Extension (Sun Microsystems, 2004c).

In table 1 we provide a set of basic software metrics related to the *Nabor* project source code, and in table 2 we show *Nabor* software statistics from the user's point of view.

4.2 Evaluation

In 2004 *Nabor* was deployed in five cities in Poland on over 200 independent client workstations and handled almost 30.000 high school candidates. Throughout the operation period, the system performed

[4]Only recently the Service Data Objects standard (IBM and BEA, 2004) has been proposed, which includes support for off-line operation mode, defined there as the 'disconnected programming model'

[5]The design of the client-side persistence layer was to a large extent inspired by the Prevayler (Prevayler, 2004) framework.

Table 2: *Nabor* user interface statistics

UI element	*Size*
GUI windows	>100
WWW pages	>20
PDF reports	22
Excel reports	60

smoothly and reliably, and to the users' great satisfaction[6] all results were correct and delivered before the required deadlines. No attempts to manipulate the admission results were reported, no major usability issues were discovered. We therefore feel that the requirements of security, fault-tolerance, correctness and usability were fully met, which proves the viability of the off-line rich client model we decided to adopt.

Security, usability and fault-tolerance of *Nabor*, however, came at a cost. The cost was not only the increased development and testing effort, but also limited extensibility of the whole system. Adding new capabilities, such as a continuously operating education management application, would break the assumption on which we based our decision to use the off-line processing model. Without the postulate that one piece of data (e.g. pupil's personal data) can be accessed and modified only by one client (e.g. one school), data synchronisation and conflict resolution become unmanageably complex. Major functionality enhancements would therefore require abandoning the off-line rich client paradigm in favour of the on-line web model.

5 CONCLUSION

In this paper we have reported on our experiences with building a large-scale distributed information system for the education sector. We described the requirements the system was to meet and how these requirements influenced the design and architecture of the software. We also highlighted selected implementation issues and evaluated *Nabor* from the performance and extensibility standpoint.

Our future plans include extending *Nabor* with nursery, primary and secondary school admission tools, as well as a continuously operating on-line education management application. Although in two consecutive editions of *Nabor*, the off-line rich client paradigm proved a perfectly viable approach, with the shift from occasional towards continuous operation of the system, we will most likely abandon the present model in favour of the more scalable on-line web client approach.

[6]According to the survey we carried out after the admission had finished, of all the users 53% were generally satisfied and 45% were fully satisfied with *Nabor*.

REFERENCES

Apache Foundation (2004a). *Torque: Persistence Layer*. http://db.apache.org/torque/.

Apache Foundation (2004b). *Velocity Template Engine*. http://jakarta.apache.org/velocity/.

Bauer, C. and King, G. (2004). *Hibernate in Action*. Manning Publications.

Capita (2004). *Education Management System: On-line Admissions and Transfers*. http://home.capitaes.co.uk/EMS/Modules/ Admissions_and_Transfers.asp.

Eclipse Foundation (2004). *Rich Client Platform*. http://dev.eclipse.org/viewcvs/index.cgi/ ∼checkout∼/platform-ui-home/rcp/index.html.

European Comission (2002). *eEurope 2005: An information society for all*. http://europa.eu.int/information_society/eeurope/ 2005/index_en.htm.

Herrington, J. (2003). *Code Generation in Action*. Manning Publications.

IBM and BEA (2004). *Service Data Objects specification*. ftp://www6.software.ibm.com/software/developer/ library/j-commonj-sdowmt/Commonj-SDO-Specification-v1.0.doc.

Microsoft (2004). *Smart Clients: Combining the Power of the PC with the Reach of the Web*. http://www.microsoft.com/net/products/client.asp.

NITRD (2004). *Large Scale Networking*. http://www.itrd.gov/iwg/lsn.html.

Prevayler (2004). *Prevayler: Free-software prevalence layer for Java*. http://www.prevayler.org/.

Sun Microsystems (2004a). *Java Cryptography Extensions*. http://java.sun.com/products/jce/.

Sun Microsystems (2004b). *Java Foundation Classes*. http://java.sun.com/products/jfc/.

Sun Microsystems (2004c). *Java Secure Socket Extension*. http://java.sun.com/products/jsse/.

Sun Microsystems (2004d). *Java Web Start Technology*. http://java.sun.com/products/javawebstart/.

Weglarz, J., Rychlewski, J., Starzak, S., Stroinski, M., and Nakonieczny, M. (2000). PIONIER — Optical Internet in Poland. ISThmus, Poznan.