4

Developing Software for School Administration and Management

Incorporating Flexibility

M. Bajec, M. Krisper and R. Rupnik
University of Ljubljana, Faculty of Computer & Information Science, Trzaska 25, 1000 Ljubljana,Slovenia
e-mail: marko.bajec@fri.uni-lj.si

Key words: Software Flexibility, Business Rules Technologies

Abstract: Flexibility is one of the most important characteristics of software systems for computerised school management and administration, in particular if the software has to be used in several institutions. Given that school institutions are diverse in many aspects and have their own specific needs, several problems are faced when developing unified software. In this paper we describe an approach to information system planning and development, which we believe can help to gain the required software flexibility. The main purpose of the paper is not to examine technical issues on information technologies but to emphasise the possibilities that have to be considered when developing software in support of management and administration in schools.

1. INTRODUCTION

Studies on the use of Computerised Information Systems in schools have proved the correlation between the existing variance in the type and extent of School Information System usage and the ability to adapt software to institution-specific needs (Mahnic, 1997; Wild & Fung, 1996). Flexibility is an example of a software characteristic that makes transition to new software smoother and easier. Ofcourse, there are several other indispensable features that can affect the decision whether or not to use or adapt the system. In our opinion, system developers have to be aware of all the requirements in order

to develop adequate systems. Business users, in our case school managers, also have to be aware of the possibilities offered by contemporary technologies and approaches in information system development. A few years ago it would not have been expected that a system would be able to take instructions only by voice, without using a keyboard or mouse – this was simply not possible. New technologies, however, made this dream a reality.

We will use this opportunity, therefore, to introduce a rather unknown approach to information system development. Since our domain is information science, and we have some experience in planning and developing software for school administration and management (Rupnik *et al.,* 1997; Mahnic, 1997), we believe that school management and administration is a good example of an application domain that can fit several modern development concepts. One of these is a Business Rules Approach that primarily focuses on software flexibility issues.

2. WHY FLEXIBILITY?

The need for new approaches and technologies emerged from the problems that we experienced with the development of an information system for higher-education institutions in Slovenia (Rupnik *et al*., 1997) and later with the system maintenance. The project was funded by the Ministry of Education and Sport and took place at the beginning of 1995. The objective was to develop an information system that supports centralised processing of enrolment applications, which is a common process for all Slovenian universities and independent colleges. As there were many institutions (with many different study programmes), each of which had followed its own enrolment policies and rules, we had a tremendously difficult job developing a rigorous, and at the same time flexible, enrolment policy model. Even today, five years from the launch of the Enrolment System, we still have to make changes required by faculty management.

Similar findings were discovered when developing a students records information system (Mahnic & Vilfan, 1995) for the University of Ljubljana, the largest university in Slovenia. It has 26 member institutions (twenty faculties, three academies and three colleges), more than 40,000 students, over 2,600 teaching and research personnel, and an administrative staff of about 1,250. The purpose of the development project was to support entrance examinations, enrolment, examination records, alumni records, various analyses and statistical surveys. All the applications were written in co-operation between the Faculty of Computer and Information Science and the University Computing Centre, with the support of the European Union

Tempus program (project IEP 1852 "Computerisation of Administration and Management in Higher Education", 1991-94). As Mahnic (1997) states, in Slovenia faculties have substantial autonomy within their universities, and they often have their own policy regarding the use of information technology. Thus the development of an integrated university information system was not only a difficult technical task, but required a substantial organisational effort. Among the other findings discovered through close examination of the experiences of other institutions and initiatives in foreign countries (McDonough, 1992; Powell, 1991; Frackmann, 1991, 1992; Schutte, 1991), was that the system has to offer a certain level of flexibility in order to handle all the differences among the member institutions (in organisation, administration, etc.). Even though some specific solutions required by particular institutions were not a part of an overall agreement, the project team was forced to consider them in order to retain user satisfaction. Again, the question was, how to achieve the required level of flexibility?

2.1 Business Rules and Flexibility

Recently, much effort has been put into developing applications that are flexible and easier to change and adapt. Unfortunately, most of today's applications do not apply to these characteristics, as changes require the modification of low-level program code. Of course, not all changes are equally difficult. If a customer wants to have an additional dialogue box incorporated into the user interface, we can (usually) do that without any substantial effort. But what if, for example, a process of entrance examinations has been changed due to some additional rules that have been put into operation? Or what if a current system allows a student to select four course offerings for the coming semester, but now management would like to allow students to select four course offerings plus two alternative choices, in case the student cannot be assigned to a primary selection? From an organisational point of view, these changes are rather trivial, but in terms of a software change, they are difficult and time consuming. In fact, this is commonly the case: the rules change constantly at a policy level, while we cannot keep up with the software that is used to implement them.

These kinds of rules are known as "Business Rules". Although the name is rather confusing[1], it has become a widely accepted term within information science. Since there is no common definition that would clearly explain the concept of a business rule, various inconsistencies can be noticed

[1] The adjective "business" only causes confusion, as it forces us to think that business rules can apply to the development of business applications only. In fact, they apply to all kinds of applications.

when reading papers on business rules or using tools that claim to support them. For the purposes of this paper, a business rule will represent a statement that defines or constrains some aspect of the organisation's behaviour. Here are some examples:

- A candidate can apply to a maximum of three study programmes at the same or different institutions.
- Each study programme is composed of several courses.
- A date must be specified for an examination and cannot be changed after it has been published.
- A student must register for an exam at least three workdays before the examination date. After that, any registration is rejected.
- If a candidate is not a full-time student, her/his registration to an exam is automatically cancelled, unless she/he has preliminarily paid for the examination.
- If a student has failed an examination more than three times, the board of examiners must be convened. In addition, the student must pay for the examination.

According to various discussions (GUIDE, 1995; Barnes & Kelly, 1997; Ross, 1997; Hurwitz, 1997), business rules have a significant impact on software flexibility and scalability. If not presented properly, for instance if buried in the program code, they can be very difficult to manage and maintain. The most common problems that arise as a result are:

- Every change of business rules requires programming.
- Business rules are distributed across the application logic; thus the place where the change has to be made is hard to find.
- Business rules are dependent and interrelated chunks of logic. Therefore they have to be modified carefully, considering the possible effect on the other rules.
- It is very difficult to control business rules, as there is no common place where they are stored.
- Since the need for changes to rules usually arises from organisation requirements with which developers are not necessarily familiar, there is a risk the requirements will be misunderstood.

These are only a few of the problems that stimulated the development of a new strategy that is primarily focused on business rules. The main idea of this approach is to conceptually, logically and physically separate business rules from the other parts of application, data and functionality, while

making them easy to access, view, modify and manage. Achieving those goals results in improved application flexibility and scalability.

Before discussing the concept of a business rules approach and its associated technologies, we will first examine how the rules are managed within the traditional application development life cycle and traditional applications.

3. TRADITIONAL APPROACHES TO FLEXIBILITY AND ADAPTABILITY

In order to make applications adaptable and flexible, developers have been using several different approaches for a long time. The most common traditional methods include parameterisation and using database mechanisms.

3.1 Parameterisation

One method of adding flexibility and adaptability to an application is to parameterise the application and its components. These parameters may be then set in a configuration file or in a database, and can be managed through a configuration utility. In doing so, the application can be adapted to different environments and situations just by parameter settings, without any programming effort.

The parameterisation technique proves useful when used to provide parameters for the business rules. Although the business rules remain hidden in the application logic, they can be modified through parameters, without any need for changing the program code. In addition, end users can make business rules modifications, if they are provided with simple and user-friendly configuration utilities.

However, this approach presupposes that the development team can foresee all kinds of changes that are likely to be required. Moreover, it presupposes that the development team able to programme and parameterise all additional cases. This requires the parameterisation of all logical decisions (decision logic has to be used or bypassed, and the variables used have to be stored as modifiable parameters). Consequently, this is an extreme burden on the application developer. In addition, application testing suffers from a combinatorial explosion effect that sometimes requires an additional application in order to configure the parameters correctly. These parameters themselves will often be interrelated, requiring assumptions and rules to be encoded in the parameter-modifying application, with similar problems.

3.2 Using Database Mechanisms

Most applications rely to a greater or lesser extent on a persistent data store that is usually based on relational database technology. Relational Database Management Systems (RDBMS) provide the so-called "triggers" that have the ability to execute actions immediately before or after particular events occur. This is when a record is inserted into a database, when a record is modified, and when a record is deleted from the database. Actions executed by triggers are written in Structured Query Language (SQL) and are stored either as the trigger body, or as a database procedure.

Because of those attributes, both triggers and database procedures can be very useful to implement business rules. For example, by using a "before-insert" trigger the record can be checked (and refused if it violates any business rule) before it is actually inserted. Since the implementation of the business rules resides in the database itself, modifications are application independent and can be performed without accessing the application logic. The advantages of such an approach are as follows:

- The formal representation of business rules is based on the SQL language, which is close to natural language.
- As business rules remain stored in the database, they are independent of the application logic.
- Business rules are not dispersed over a number of clients - they are stored on the database server.
- Modifications of business rules can be performed remotely (remote accessibility is one of the most important features of database servers).
- The place where the particular business rules are implemented is easier to find, since we know the actions that initiate them.
- Adding a new business rule does not necessary require changes to the application code.

Of course, there are some disadvantages as well. First, the approach suffers from being data- and database-dependent. Second, it cannot easily provide rule support for client or middle-tier applications that deal with complex business objects and business processes. And finally, such an approach forces the data store to be a process engine, a task that database engines are not optimised for.

4. BUSINESS RULES APPROACH

Till now we have only discussed problems regarding the implementation and execution of rules. However, the shortcomings of the traditional development life cycle come from other phases as well. According to proponents of a business rules approach, there is no support in conventional development processes for business rules acquisition, modelling and design (Hurwitz, 1997). In their opinion, business rules should be represented as a distinct concept within application analysis, design and implementation, and should not be treated implicitly through other perspectives that primarily focus on information resources like data, functions, processes, etc.

The business rules approach therefore suggests additional activities to be carried out in support of business rules philosophy:

- Business rules discovery;
- Business rules classification;
- Information resource object mapping;
- Business rules implementation; and
- Business rules change management (dependency, efficiency, versioning, traceability).

From all the activities listed above, we will only focus on the implementation phase. As we mentioned at the beginning, our intention is only to emphasise the possibilities of modern technologies (the technologies that end users work with). The scope, of course, is software flexibility.

5. BUSINESS RULES ASSOCIATED TECHNOLOGIES

According to the findings of the GIGA Information Group (Rymer, 1997), business rules have begun to get appropriate attention as a modelling approach (Gottesdiener, 1997), as well as an important meta-element addressed by software development tools. It is expected that developers building event-driven relational database applications will be the largest market segment for business rules tools. Besides the fact that the major database vendors are targeting this opportunity (Rymer, 1997), the appearance of some new business-oriented development tools confirms these predictions. Considering the business rules representation, implementation and execution, these tools can be classified as follows (Barnes & Kelly, 1997):

- Database-independent tools;
- Server-based tools; or
- Rule-based systems.

5.1 Database-Independent Tools

Database-independent tools use database mechanisms to enforce business rules, which seems rather confusing at first. However, different from the typical RDBMS[2] that we mentioned in previous sections, these tools support codes for triggers and procedures to be created automatically, and managed within the tool. This effectively moves the creation and management of data-centred business rules up one level to an application development tool rather than a database-specific tool.

Development tools that are based on such an approach presuppose that business rules form sufficient information (together with other rules) for developing any business application of a classical type. Through interviews and business area analysis, the analysts begin to form structure for the data elements required by the end user. The analyst identifies tables, columns, data types, data relationships and other structure-based rules (data rules). Instead of requiring the end user to verify the data model, the tool creates components for the database and a window interface to review the structure by working with the application (based on those rules). If the application does not match the rules of the business, the analyst can immediately change the structure rules. The user-interface is then specified further by using implementation rules (presentation rules). Finally, the specification of all other business rules takes place. The tool usually provides interfaces to capture, describe, formalise, and implement business rules.

[2] RDBMS – Relational Database Management Systems.

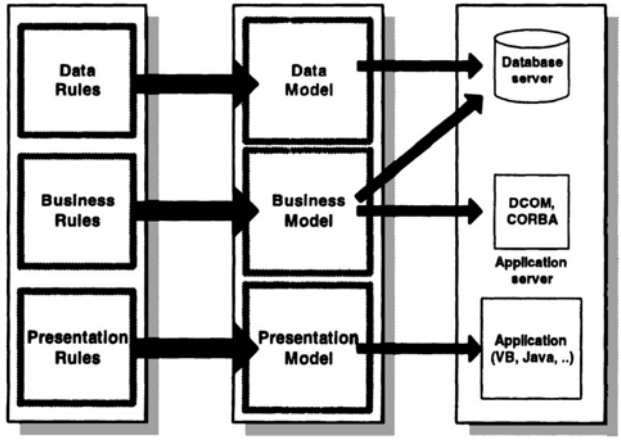

Figure 1. From rules to application

A good example of this type of product is Vision Software's *Vision Builder*[3], which automatically generates the appropriate stored procedures and triggers that reside in the target database.

The most important benefit of using such an approach to develop business applications is when the application needs to be changed and maintained. Because the complete information is stored in one place - in the repository – modifications can be easily made and the components can be automatically regenerated. However, this presupposes that the application is, and can be, 100% generated from the repository information. (The question of 100% generation goes beyond the scope of our discussion; it relies on sophisticated libraries and template sets of classical business applications modules.)

5.2 Server-Based Tools

Another way of ensuring business rules is to implement them as an application service. An example of a product associated with this type of approach is Usoft's *Developer*[4], which, like previous database-independent tools, assures that the application is automatically derived from the business rules (including implementation rules). The peculiarity here is that once the business rules are captured and stored in the repository, no further programming is required to fire and process those rules. The business rules processor does it all automatically. The tool also creates both the database

[3] Vision Software, http://www.vision-soft.com
[4] USOFT, Inc., http://www.usoft.com

and the other required components of three-tier architecture. Client-side applications then invoke objects, methods or functions on the server that contain business rules. Thus the main application and data-related logic reside on the application server and not in the database itself.

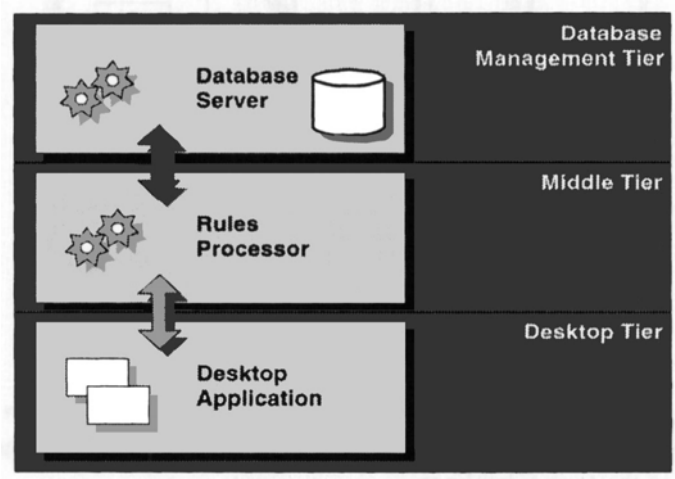

Figure 2. Rules processor: middle-tier service

Such an approach brings additional advantages. Developers do not need to worry about when the rules need to be fired and in what order they should be fired, because the rules processor handles all this. Another important advantage is technological independence. While the application architecture follows the three-tier architecture industry standard (known as ANSI/SPARC three-tier architecture), the different components of a business application remain independent. In this way, as better technologies are developed to implement the separate tiers, these new technologies can be "plugged in" to supersede less efficient methods.

5.3 Rule-Based Systems

All procedural languages are built upon the foundational concept of procedural flow, sometimes called the control flow. Programs written in languages like C, C++, Java, Basic, etc. are all based on control flow. The text of such programs is executed left to right, top to bottom, as the source code would be read. If we know the statement of procedural code being executed, we can always determine which statement will be executed next.

The benefits of procedural languages are predictability and straightforward mapping to today's serial, Von Neuman architectures, which

make them very popular and commonly used program languages. However, their drawback is that they are difficult to use. The fact is that if we want our computer to compute something, we must first take all the specifications and constraints out of the problem and then organise them into a set of loops and conditional sentences that correctly describe how the computer should behave in the face of every problem it might encounter. This, of course, is not an easy job, otherwise there would be no shortage of software engineers, and maintenance costs would be probably much lower.

One example of a program language that follows to a rather different philosophy is a rule-based language. The most important characteristic that distinguishes it from the procedural language (beside the fact that there is no control flow in the rule-based language) is that with a rule-based architecture the system, not a software engineer, is responsible for mapping specifications and constraints into executable code. Unlike the procedural languages, rule-based languages take independent statements of if-then-else rules and automatically integrate their 'if' parts into a single, integrated and efficient conditional logic, which is directly executable by an inference engine. The inference engine is the core of every rule-based system. It monitors the applicability of the conditions of rules against a database. Whenever the database changes, whether by adding, deleting, or modifying a record in the database, all the rules are checked and fired if necessary.

Rule-based languages have been used for a long time, since they are the essence of every expert and knowledge-based system. They can be very useful if used as a language for specifying conditional or declarative knowledge. Moreover, they can be useful to specify business rules as they represent a special class of rules only. In fact, according to the findings of the GIGA Information Group (Rymer, 1997), "the interest in business rules will fuel expanded use of rule-based systems in commercial applications. Business rules products that are easily integrated with a variety of languages, platforms, and legacy systems will be the most successful." The ability of integration and coexistence is actually the essential difference between the so-called expert systems products of the 1980s and contemporary rules products. The classic rule-based system architecture is shown in Figure 3.

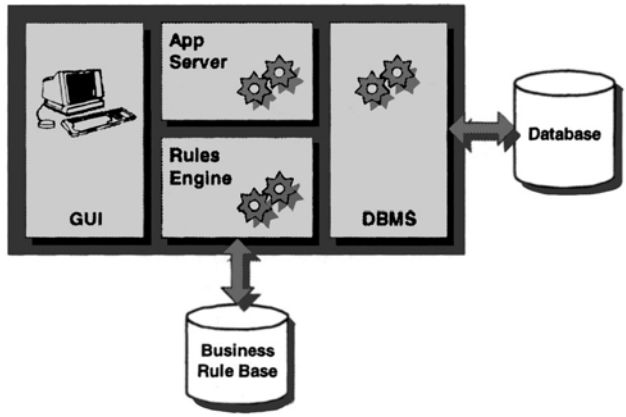

Figure 3. Rule-based system

Today's rules products can be easily integrated with different procedural program languages, various databases, and even with leading GUI tools. Neuron Data's *Elements Expert*[5] and ILOG'S *Ilog Rules*[6] are good examples of such products.

There are many benefits of using rule-based systems instead of conventional development tools, when engineering systems for change. The most important are gathered below.

The key benefits of rule-based systems are (Barnes & Kelly, 1997):

- *Incremental development and rapid prototyping.* The rules can be run and tested the moment they are added to the system. Unlike traditional programming tools such as C++ or C, changes to the rules do not require recompilation, re-linking and re-deploying.
- *Understandable units of business practice* Rules in the rule-base are self-contained chunks of logic, representing single concepts. This helps their readability and understandability.
- *No control flow.* Unlike a conventional program that usually has a single starting point and a set sequence of execution, there is no control flow in a rule-based approach. Rules can start to execute from any point in the rule-base.
- *Consistency.* In comparison to conventional code, incomplete, incorrect, irrelevant or redundant rules are much easier to find, since they stick out from the system.

[5]Neuron Data, Inc., http://www.neurondata.com
[6]ILOG Inc., http://www.ilog.com

- Ability to work with incomplete and missing information. In many business situations, it is not always possible to provide complete and verifiable data. Rule-based systems can deal with such cases of incomplete information – the rule engine is able to work with the special values "unknown" (not relevant in the case of a particular calculation) and "not-known" (the value is not known).

6. SUMMARY

The business rules approach that has been introduced in this paper follows a new philosophy that forces application developers to deal with business rules explicitly. The tools based on such an approach efficiently separate the business rules apart from the other applications components, describing them in both business and formal language. Even though there is no tool that would support the intuitive path leading from definition to implementation of business rules (Gottesdiener, 1997), we can get several advantages out of it.

End users of school information systems would find the business rules approach and related technologies very valuable. Programmers are able to:

- Develop applications that are easier to adapt to different environments;
- Involve end-users in application development;
- Let the end-user control the rules;
- Let the end-user maintain the rules; and
- Let the end-user simulate (test) different rule scenarios (how would the system behave if some rules are changed, added, removed, etc.).

Despite the fact that a business rules approach can be used in a variety of situations, it may not be applicable to all application domains. However, when engineering systems that have to offer a certain level of flexibility (and school information systems are a good example), such an approach is valuable.

REFERENCES

Barnes, M., & Kelly, D. (1997). Play by the rules. *Byte* (Special Report), *22* (6), 98-102.

Frackmann, E. (1991). Information for institutional administration and management in German higher education. *CRE-action, 3,* 29-46.

Frackmann, E. (1992). HIS: The German initiative for administrative computing in higher education, Information for institutional administration and management in German higher education. *Higher Education Management, 4* (3), 317-328.

Gottesdiener, E. (1997). Business Rules show Power, Promise. *Application Development Trends, 4* (3), 36-42.

Hay, D. & Healy, K.A. (1997). *GUIDE Business Rules Project, Final Report – revision 1.2.* Chicago: GUIDE International Corporation.

Hoffer, J.A., George, J.F., & Valacich, J.S. (1999). *Modern systems analysis and design (2nd* ed.). Reading, MA: Addison-Wesley Publishing Company.

Hurwitz, J. (1997). When rules meet development. *Database Programming & Design, 10* (1), 12-14.

Lawrence, B. (1998). Designers must do the modeling! *IEEE Software, 15* (2), 30-33.

Mahnic, V. (1997). Towards the re-integration of the University of Ljubljana information system. *Proceedings of European co-operation in higher education information systems.* Grenoble, France, September 1997, pp. 250-258

Mahnic, V., & Vilfan, B. (1995). Design of the student records information system at the University of Ljubljana. *Trends in academic information systems in Europe, Proceedings* of the *EUNIS'95 Congress.* Dusseldorf, Germany, 1995, pp. 207-220

McDonough, R. (1992). The management and administrative computing (MAC) initiative. *Higher Education Management, 4* (3), 284-292.

Powell, I.H.C. (1991). The MAC initiative in the United Kingdom. *CRE-action, 3,* 19-28.

Ross, R.G. (1997). *The business rule book: Classifying, defining and modelling rules.* Boston, MA: Database Research Group, Inc.

Rupnik, R., Bajec, M., & Krisper, M. (1997). Information system for application procedure for registration in higher education in Slovenia. *Proceedings of European co-operation in higher education information systems.* Grenoble, France, September 1997, pp.221-226.

Rymer, J.R. (1997). *Business rules: A promising technique, but not a new paradigm.* GIGA Information Group. [http://www.ilog.co.jp/html/products/infrastructure/rules_ga.htm]

Schutte, F. (1991). Information policy of Dutch university management. *CRE-action, 3,* 53-82.

Wild, P., & Fung, A. (1996). Evaluation of information technology in educational management for proactive development. *Education and Information Technologies,* 239-249.