ISSN 1392 – 124X INFORMACINĖS TECHNOLOGIJOS IR VALDYMAS, 2004, Nr. 1(30)

THE REQUIREMENTS OF BUSINESS RULES MANAGING

Liudas Motiejūnas, Rimantas Butleris

Department of Information Systems, Kaunas University of Technology
Studentų St. 50, 3031 Kaunas, Lithuania

Abstract. Business rules approach is quite new and oriented at software systems in which the rules are separated, logically and physically, from other aspects of the system. There are several problems arising while managing business rules. In this paper some current software practice is discussed. The fact model and OCL (Object Constraint Language) for business rules collecting are analysed. Events that activate business rules and steps of managing them are discussed. A schema for execution of the business rules is presented.

1. Indroduction

Business rules are precise statements that describe, constrain and control the structure, operations and the strategy of a business. They can be found everywhere in a raw, unstructured form. The business rules are the most changing part of the business. We can describe a business rule as "a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business" [1].

Traditionally, business rules have been buried in the code of an application program, embedded in database structures, and coded as DBMS (Data Base Management System) triggers and stored procedures. In such case it is difficult to implement new changes in enterprises policy and to adjust the program code to real life needs. To manage these problems, we need a business rules system. Business rules systems can be defined as such applications that contain dynamic business rules and require [2]:

- The ability to quickly change rules without modifying application code;
- A mechanism for executing rules that integrates easily into application architectures;
- A high-level business rule language that can be understood by business administrators and/or users of the application;
- Tools like rule editors that support the definition and maintenance of business rules.

2. Current software practice

Nowadays there are several products that are implemented for managing business rules. Further we will discuss two of them. The first one is *Blaze Advisor* [3]. *Blaze Advisor* is the leader in the rules management technology, an industry-proven process for designing, running, and maintaining e-business applications [4].

Blaze Advisor consists from the following components [3-5]:

1. ***Blaze Advisor*** **Builder** – a development environment for creating business rules from scratch. It is aimed primarily at power users/developers and provides low-level programming utilities and graphical browsers for writing, editing, viewing, and testing rules.
2. ***Blaze Advisor*** **Innovator** – a rule management and maintenance tool for business users. Users create and modify rules through a centrally defined set of menus and controls. The "Workbench" provides a number of Web-based interfaces for structuring rules in accordance with business policies, such as allowed values and constraints. An XML-based repository provides the underlying storage facility for all rule components and changes. The source versions of rules are stored and indexed in a standard flat file structure.
3. ***Blaze Advisor*** **Rule Engine** – a scalable processing engine that determines and executes the whole control flow of rules created with *Blaze Advisor* Builder. The engine works in concert with *Blaze Advisor* Rule Server (Figure 1) to access and run the appropriate rules needed by applications. Rule processing can also be triggered by external events.
4. ***Blaze Advisor*** **Rule Server** – a dedicated rule server for enterprisewide deployment. The server supports rule execution, session management, scheduling, and dynamic load balancing. The

Deployment Manager component handles all the underlying connections to databases.

Blaze Advisor architecture is shown in Figure 1 [6]:

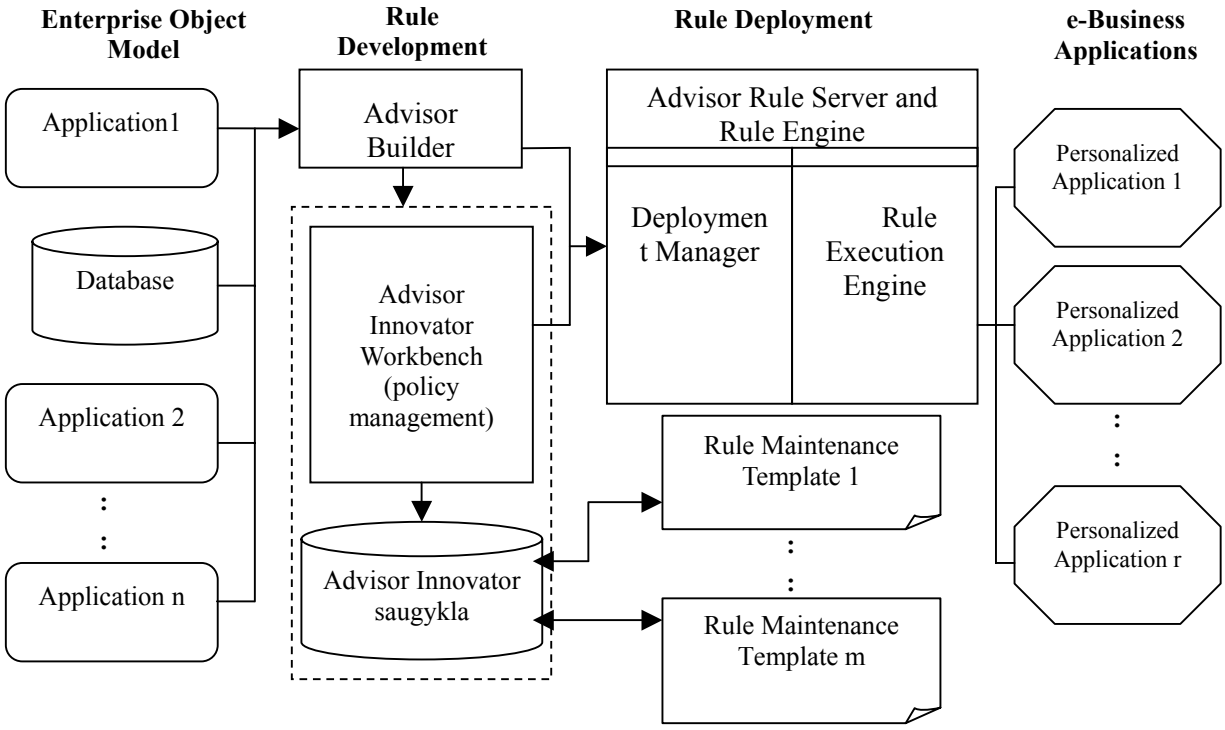

Figure 1. *Blaze Advisor* architecture

Another product is *Infrex* [7]. It is implemented in a way different than that of *Blaze Advisor*.

Infrex is designed as an embeddable rule engine (rather than a stand-alone engine) and integrates into applications written in C/C++/Java/C#. *Infrex* offers a variety of products to meet different needs. These are [5, 7]:

- **Rule Editor:** The *Infrex* rule editor allows the specification of rules using classes and variables of the application. Various types of editors are provided to make the rule definition simple and easy to use. The rule language supports high level operators that ease rule definition.
- **Rule Translator:** *Infrex* tools process the INI (Interface) and RB (Rulebase) files to generate C/C++/Java/C# code. This code is compiled and linked with the application to create an executable. The executable has the rules to be called at run-time, through the engine.
- **Rule Interpreter:** Rules can be interpreted by linking the application with the *Infrex* Interpreter library 'one-time' to create the application executable. This executable has the ability to read a rulebase file at run-time and execute the rules.
- **Rule Extractor:** The Rule Extractor is a tool for extracting rules stored in database tables into the format specified by *Infrex*. This allows applications to store rules in specific customized formats and extract them based on conversion specifications. Additionally, Decision Tables, Decision Trees or Graphs are some of the popular formats supported. The Rule Extractor converts tables into a rulebase file by scanning tables and extracting rules through a script language.
- **APIGEN:** *Infrex* facilitates dynamic linking of translated rules with an application through a utility called APIGEN. This allows rules to be changed without the need for the application to be re-linked and recreated. This is supported on the UNIX platforms by the rule translator, which compiles the rule base into a "shared object" file. A DLL (Dynamic Link Library) of the translated rules can be created on Windows-based systems. When used along with the Rule translator, APIGEN generates all wrapper functions needed to call the rulebase functions contained in the shared object.
- **Rule Debugger:** The Rule debugger facilitates debugging of the rules in a Rulebase file.
- **Rule Tracer:** The tracer keeps a log of the rules that are fired during runtime.
- **Condition Evaluator:** Allows forming of rule conditions at run time and dynamically evaluates them.

The links between *Infrex* components are shown in Figure 2 [8]. These links show how *Infrex* components are linked between each other.

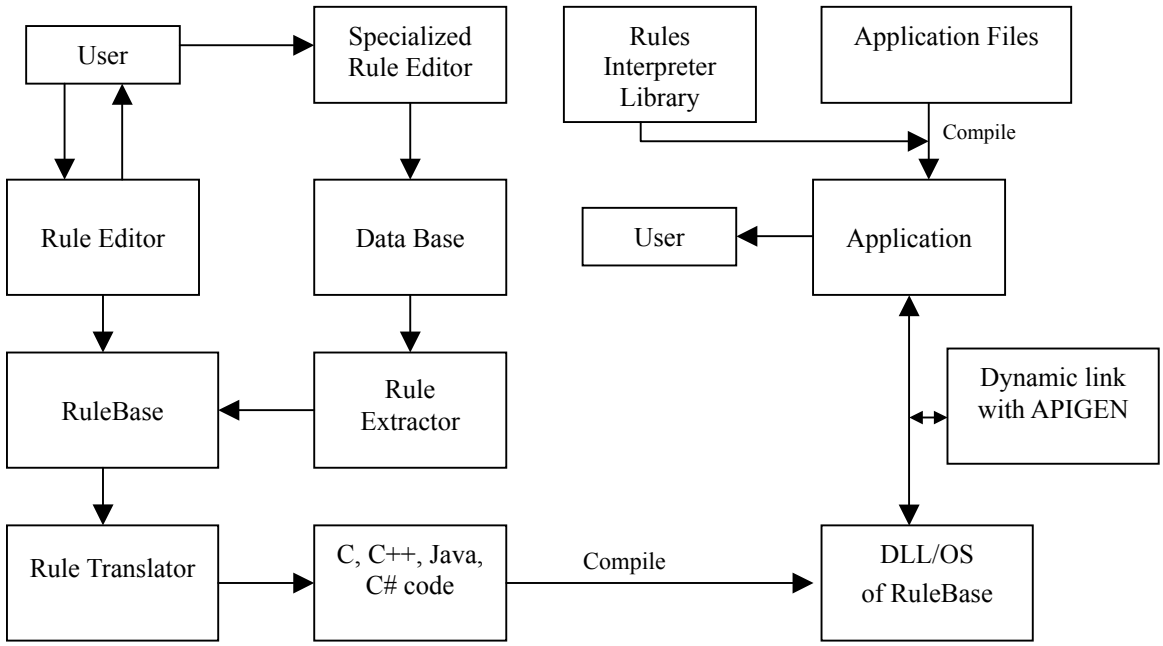

Figure 2. Links between *Infrex* components

As we can see, business rules systems can be developed in different ways. More usual is the first way, where the business rules engine manages business rules directly from the rules repository, without translating them into rulebase files. Both of these products have tools that help the user to define rules in a proper form and syntax. However, these tools can not verify the logic of business rules. No matter which type of system is chosen, first of all the business rules have to be extracted from the enterprise's business policy and these rules must be logical and consistent. Further we will discuss several suggestions for invoking and managing business rules.

3. Business rules consistency

At first we need to ensure the consistency of business rules (rules must be concerted and do not conflict with each other) for correct performance of the business rules system. Business rules must define overall business policy of an organization. Business professionals have to be involved in business rules definition process. Because rules are built on facts, facts are built on terms [9], in a very beginning of developing of the system it is useful to create a fact model. The fact model represents the basic vocabulary for expressing its rules. A term is a basic word or phrase in English or other natural language that workers recognize and share in business. Terms are always nouns or qualified nouns. Facts are given using simple, declarative sentences that relate appropriate terms [9]. In such way original knowledge will be registered and understandable for all people that participate in developing of the system. The fact model is similar to the data model, but it isn't the same. According to the fact model it is possible to develop the data model and later compose business rules that present knowledge about the organization kept in the fact model.

For more precise definition of an organization model in the design phase additionally *Object Constraint language* (OCL) can be used. This language defines constraints for objects. These constraints later can be treated as business rules. Constraints bring us several advantages [10].

1. Constraints add to the visual models information about model elements and their relationships. Therefore, they are an excellent form of documentation. Constraints must be kept close to the model. Versioning of the constraints should be in accordance with the versioning of the model(s) to which they apply.
2. Different people can not interpret constraints different. They are unambiguous and make more precise the model or system to which they apply.
3. Models are used to communicate among users, modelers, programmers, and other people. Flawed communication is responsible for the failure of many software projects. Most models are accompanied by a natural language explanation to help the receiving party understand the model. But readers often must rely on their own interpretation. Using OCL constraints, the modeler can unambiguously communicate his intent to other parties.

Constraints that are made for objects, similarly as business rules, are expressed in a declarative manner. In a declarative language, constraints have no side effects; that is, the state of a system does not change because of evaluation of an expression [10]. There are three advantages to this interpretation.

1. The modeler need not decide how the violation of a constraint should be handled. Actions that handle violation of a constraint are taken in the business rules engine. This approach results in a clean separation between specification and implementation.
2. Constraints should be stable within their domains; that is, they should not change much over time. The actions that need to be undertaken when the constraints are not met change more often or can be different from one application to another within the same domain.
3. When constraints are checked with the aim to perform an action on them, the check must be regarded as one atomic action.

When we have developed full fact and data models and defined constraints for objects, we can specify business rules, which will be consistent and concerted (Figure 3).

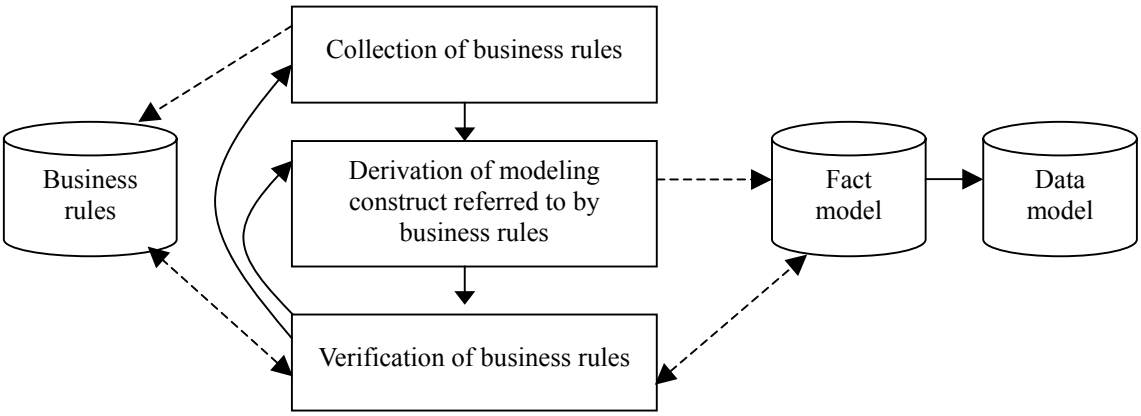

Figure 3. Business rules derivation

4. Calling business rules

Because business rules are kept in the business rules repository separate from the program code, they are declarative and implicate no control logic, they have to be called and executed by a special component – business rules engine. Every rule rejects, produces or projects some type of actions or data [9]. Also each rule is associated with particular data. Until the user does not take any action, business rules are not called from the business rules repository. But when some action occurs the business rules engine must verify business rules and evaluate that action. Generally, the business rules engine starts working on the three basic events – INSERT, DELETE and UPDATE. Business rules that respond these events are similar to data base triggers, however they are kept separately from the database. Either way when the user makes an attempt to insert, delete or update data, the business rules engine must fire business rules that are associated with data that the user wants to change. It is like a monitoring process – business rules engine is inactive and comes to action only when some changes in data are noticed. Business rules of this type must be fired at least on two separate events [9]. Let us say we have classes shown in Figure 4 [10].

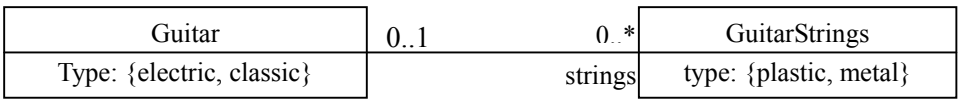

Figure 4. Class model

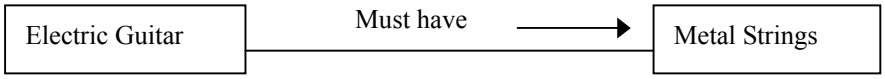

Figure 5. Terms and facts for *Electric Guitar* rule

There is a rule for these classes: *Electric guitar must have metal strings.* This rule can be represented by terms and facts like it is shown in Figure 5.

Such a notation of a rule is quite understandable, but sometimes there can be a need to define rules in a more formal way. In such case we can use *Object Constraints Language*, which is more understandable for programmers:

Guitar
Type = #electric implies strings –>
forAll (type = #metal)

The first event when this rule has to be evaluated is visual – as a new *Guitar* record is created (event INSERT), *metal Guitarstrings* have to be assigned to it. If a type of the string is not *metal*, then the business rule is violated and an error message has to be

supplied to the user. But there is another event when the business rule can be violated – as someone tries to edit (event UPDATE) the *Guitar* record. In such way the business rule has to be evaluated on two events to ensure the consistency of the systems data. A general schema of the business rule, which is called on INSERT, DELETE and UPDATE events, is shown in Figure 6.

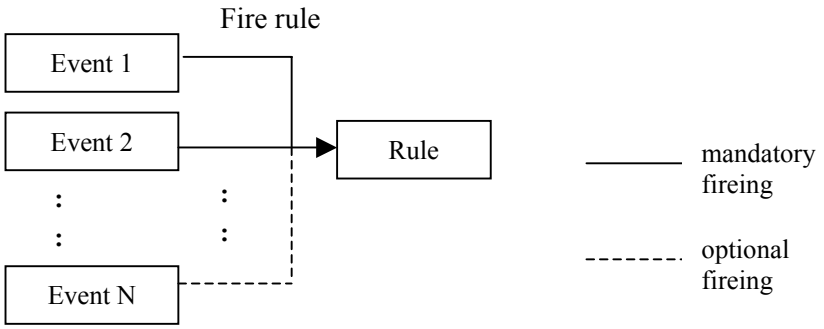

Figure 6. Rule fireing on INSERT, DELETE and UPDATE events

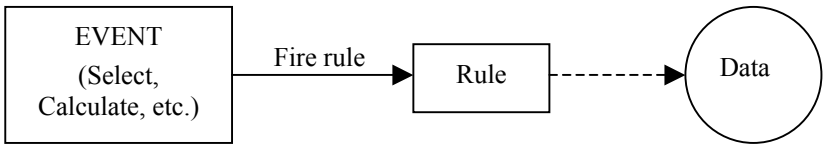

Figure 7. Rule firing on specific events

As we can see from Figure 6, each rule *must* be fired at least at two events, but of course there can be more events that can violate the rule.

But there can be other kinds of business rules in a system. That means, business rules are fired in other situations, not only when the user attempts to change data. This kind of rules is not associated with the data control; they can create data themselves. As an example, business rules of SELECT or CALCULATION type could be pointed out. The result of such business rules is derivative data that can be stored in a file, shown on the screen or printed in a report. These rules usually are called by specific events, which can depend on user's actions (button click) or simply on the timer (the last day of the month). Business rules of this type do not have to be fired at least on two events, because they do not ensure the consistency of data, they can create data themselves (Figure 7).

5. Execution of the business rules

As it was mentioned above, business rules can be of two general types – those, which secure the integrity of data and those, which are not directly associated with the data control and can create derivative data. Rules of the first type can be violated and must be fired when the user attempts to change data. Then the business rules engine has to take appropriate actions and to ensure the consistency and the integrity of the data. Actions are as follows:

1. When an appropriate event DELETE or UPDATE occurs, then the business rules engine makes a copy of data (if event INSERT has occurred a copy of the data is not made) that the user tries to change (it can be a single field, a record or some other data structure).
2. Data changes that were indicated by the user are performed (for example, the user has updated one field in a record).
3. The business rules engine fires the rule that is associated with that field and performs a test, restriction or the same action as it is defined in that rule.
4. Depending on whether the rule was violated or not, the following actions are initiated:
   a) If conditions described in the rule were satisfied, then the actions defined by the user are accepted (in this case the updated field is accepted) and no message is shown.
   b) If conditions described in the rule were violated, then the actions defined by the user are rejected, initial data, using a copy that was made in step 1, are restored and an error message is shown to the user.

These are the actions that have to be performed by the business rules engine to verify one single rule. But in a large organization there are hundreds and thousands of business rules and often the same data restrict several different rules. In such case, the business rules engine has to verify all the rules that are associated with the data the user wants to change.

The action sequence diagram for managing single or multiple rules is shown in Figure 8.

Business rules of the second type are managed much more simply. Because they create new data by data kept in the database or perform selection of data and are not related to the data control, managing them

does not require a lot of actions. When some specific action is performed or appropriate circumstances occur, then the business rule is simply fired (as it is shown in Figure 6) and actions described in the business rule are performed. However, rules of the first and the second type can be associated with the same data. For that reason, we can use specific labels to indicate the type a rule belongs to and there will be no a reason to perform rules of the second type, when the data are changed.

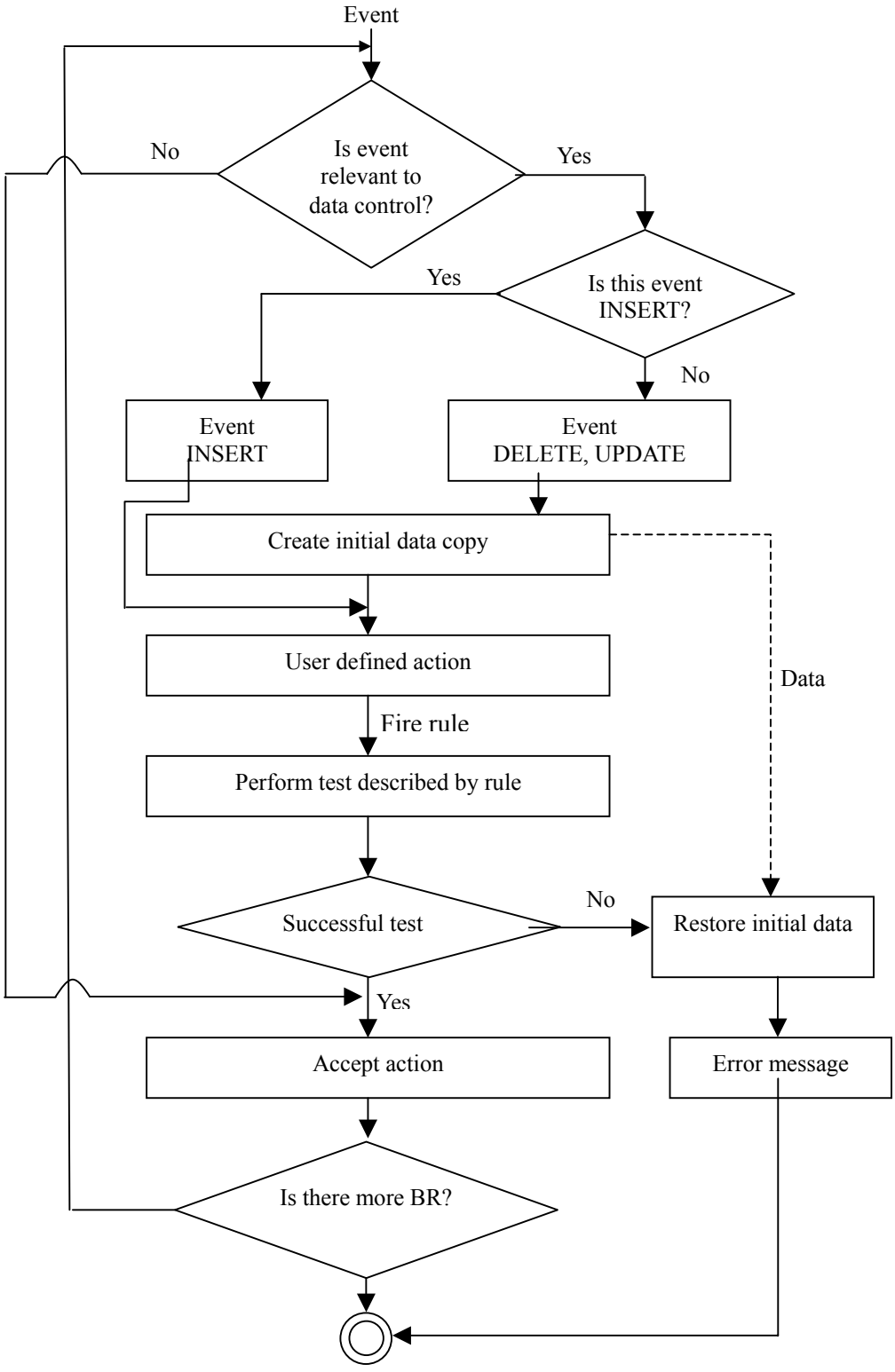

Figure 8. Action sequence diagram for managing business rules

6. Example of the rule execution

For example, let us consider the object specified in Figure 9 and a single rule for it.

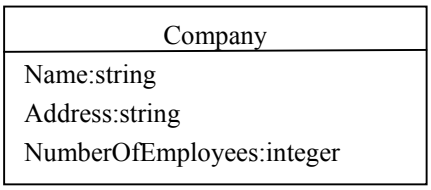

Figure 9. *Company* object

Rule: *Number of employees in a company must higher than* 20.

The formal expression of the rule in OCL is as follows:

Company
Self.NumberOfEmployees > 20

Let us say that the user attempts to insert a new company. Actions that have to be performed are shown in Figure 10.

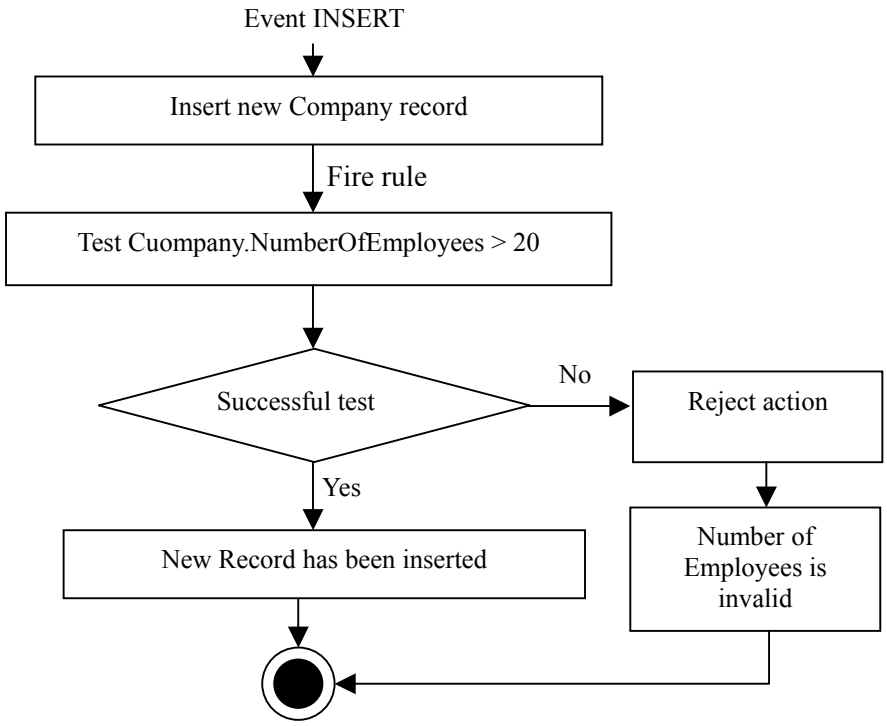

Figure 10. Example of the rule execution

7. Conclusion

In this paper, the main requirements for business rules collecting are presented. We need to ensure the consistency of the business rules for correct functioning of the business rules system. Business rules must define overall business policy of an organization. Also they must be concerted and do not conflict with each other. For this reason we can use the fact model and the OCL language.

Business rules can be of the following two major types: those, which are related to the data control and those, which are related to specific events. Events that activate business rules were discussed. There are three main events that activate business rules of the first type – INSERT, UPDATE and DELETE. Business rules that react to these events must be fired at least on two of these events. These business rules control data. Other business rules can create or derive data themselves and they are fired on specific events.

If the rule was related to the data, the business rules engine has to make a copy of the data (if event INSERT has occurred a copy of the data is not made), perform data changes and fire the rule. If the rule was violated, then the action is rejected and initial data are restored. In the other case data changes are accepted.

If the rule is activated on specific events, it is simply fired and actions described in the business rule are performed. Usually, one rule activates another one or several different rules that control the same data. For that purpose the action sequence diagram for managing single or multiple rules was presented.

On the basis of this article the next step is to create a business rules storage model and to develop an architecture of the business rules engine to manage them.

References

[1] Guide Business Rules Project, Final Report, 95/11. *http://www.businessrulesgroup.org/brgactv.htm.*

[2] Business Rules. Powering Business and e-Business. White Paper. *http://www.ilog.com/products/rules/wp_businessrules.pdf.*

[3] Developing Real-World Java Applications with Blaze Advisor. Technical White Paper. *http://www.blazesoft.com/brwhitepapers/advisor_java_apps.pdf.*

[4] Detailed Solution report. Blaze Advisor. *http://www.blazesoft.com/support_training/tech_support.html*.

[5] **L. Motiejūnas, R. Butleris**. Veiklos taisyklių manipuliavimo mechanizmų analizė. *Informacinės Technologijos* 2003. *Kaunas, Technologija,* 2003, XIV-82 - XIV-90.

[6] The Power of Rules Driven Processing. *http://www.blazesoft.com/International/UK/white_papers/index.html.*

[7] Infrex – where business rules. *http://www.tcs.com/0_products/infrex/index.htm.*

[8] Infrex. The business rules engine. Technical overview. *http://www.tcs.com/0_products/infrex/downloads/Infrex_Brochure.pdf.*

[9] **R. G. Rosss**. Principles of the Business Rule Approach. *Addison-Wesley*, 2003.

[10] **J. B. Warmer, K. Anneke**. The object constraint language: precise modeling with UML. *Addison-Wesley*, 2000.

DOI: 10.5755/j01.itc.30.1.11800