Contemporary Management Research
Pages 29-52, Vol. 6, No. 1, March 2010

A Logical Approach for Implementing Dynamic Business Rules

Nasser Karami
Sharif University of Technology
E-Mail: nr.karami@sharif.ir

Junichi Iijima
Tokyo Institute of Technology
E-Mail: iijima.j.aa@m.titech.ac.jp

ABSTRACT

Business Rules are operational rules based on data that business organizations follow to perform various activities. Regarding problem domains in the organization, business rules (BR) are classified into two groups: Static and Dynamic. A static business rule is a constraint (integrity) or derivation rule that applies to each individual state of the business, taken one state at a time. Dynamic Business Rules (DBR) are concerned with the invocation of actions in response to events.

Although a lot of modeling languages and approaches for business rules modeling and different technologies and tools for business rules implementation have been proposed by researchers and practitioners in the past ten years, there is no consensus yet on technology standard and logical relationship between the modeling and the implementation of the proposed methods.

The purpose of this paper is to implement DBR based on our proposed modeling methodology. In this study, DBR System architecture is developed using Java and Prolog+CG, a CG-based logic programming language which integrates Prolog, the manipulation of conceptual graphs, Java and object-oriented constructs. The advantages of our system are demonstrated with the case study of the Locomotive Maintenance's Business (Rules).

Keywords: Business Rules Approach, Dynamic Business Rules (DBR), Conceptual Graphs (CG), Mineau's Approach, Prolog+CG

INTRODUCTION

In the last decade, business rules have received a lot of attention in the Information Systems community. The credit for this goes to many papers written by researchers about business rules modeling and methodologies (Zaniolo et al., 1997; Ross, 1997; Mens et al., 1998; Ross and Lam, 1998; Gottesdiener, 1999; Hay and Healy, 2003) and by practitioners about rule-engine tools and application development support environments (e.g. Blaze Advisor Builder, BRS RuleTrack, Business Rule Studio, Haley Technologies, ILOG Rules, Platinum Aion).

There is a new approach that divides information systems in three components, that is, data, processes and business rules (Ross, 1997). In this approach, business rules can be managed independently from system requirements and they also need special handling. Business rules create an unambiguous statement of what a business does with information to make a decision. The formal specification becomes information for processes and rules engines to run. Business rules are an important asset of any organization: they represent decisions that are made to achieve enterprise objectives and reflect the business policies of an enterprise. While a business policy is a general statement or direction for an organization, business rules are statements that are used by a body of an organization to run their activities (Hay and Healy, 2003). They are the heart of an enterprise; they guide and affect the behaviors and ways of an enterprise.

Thus, business rules are precise statements that describe, constrain, and control the structure, operations and strategies of a business. Most of them take the normal form of if [conditions] then [actions] that can be easily created and understood by anybody (Faget el at., 2003). This form is definitely the simplest expression of business rules.

According to the literatures and current researches, business rules can be classified into four basic types: fact rules (also called terms), integrity constraint rules (also 'constraint rules' or ' integrity rules'), derivation rules, and dynamic rules (also called 'action rules', 'event-action rules' or 'automation rules') (Hay and Healy, 2003; Bubenko et al., 1998; Martin and Odell, 1998; Herbst, 1996).

A “dynamic business rule” (“DBR” in abbreviation) is a transition constraint that restricts how the business may change to new states (Taveter and Wanger, 2001). This rule defines the conditions for the invocation of an operation.

DBRs have a three-part structure, consisting of a trigger, a condition, and an action. The trigger and condition describe the conditions under which a rule becomes active, whilst the action part of the rule generates messages to active operations (Terry,

2005; Oelmann, 1991). In DBR's, When indicates a Trigger, If a Precondition and Then an Action. These business rules are expressed as "When-If-Then" statements. For example, the Withdrawal_Money rule would be represented in the following way:

When *Withdrawing*
If *Withdrawable amount >= $30*
Then *Receiving the money*.

In the field-related literature, several research works have attempted to model DBR. Terry (2005) reviewed the principal concepts behind fact-orientation and focused more on static business rules. In his paper, there is no language to handle dynamic rules. In addition, the transition from analysis to implementation has not been addressed. Ross (1997) proposed one of the most comprehensive methodologies for modeling business rules. The Ross Notation is, however, largely a database oriented methodology, and does therefore not allow to model events and actions. Neither does it support the modeling business processes.

There are a number of modeling and implementation languages and approaches for business rules modeling and implementation. For example, Ross method offers sufficient methodological guidance and specific constructs for each of the rules' family together with a big number of accompanying constructs, such as special symbols, invocation values, special interpreters, and special qualifiers. However, these properties do not seem to be an advantage, as the complexity of the resulting diagrams and the vast amount of graphical symbols make the language quite complicated. Thus, there are two main shortcomings in those approaches. First, some methods, like Ross method, are quite complicated for inexperienced users and some other methods, like OCL statements, do not have any graphical notation and thus are not understandable by business people (Ross, 1997; Demuth et al., 2001). Second, no language or approach has yet been proposed for implementing business rules by researchers who have developed the modeling method. Therefore a logical approach for handling DBR is necessary, although it needs to be simple enough in both graphical and linear form to model and implement such business rules.

In order to solve these issues, we proposed a logical approach for handling DBR in our previous paper (Authors, 2007). In that approach, DBR is modeled using Conceptual Graphs (CG) and Mineau's approach shown in Figure 1.

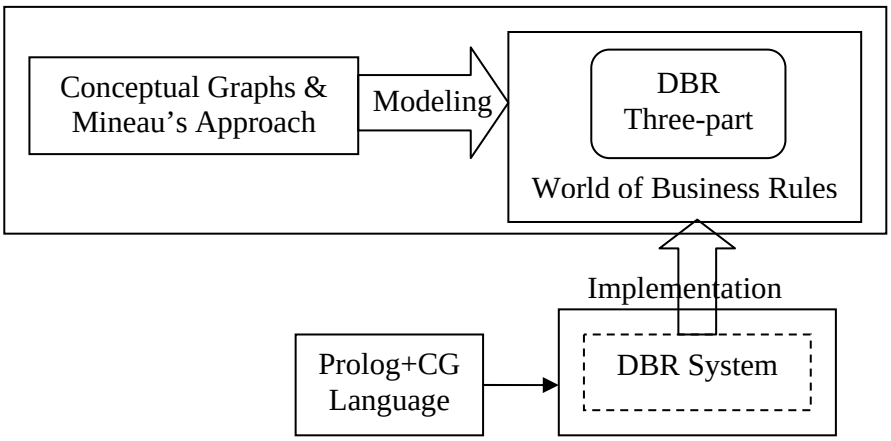

Figure 1 Framework of Research Plan

Basically, our research includes two parts (as shown in the figure). In the first part, we introduced a logical approach to model DBR. We considered together CG and Mineau's approach as a business rules modeling language because of their simple graphical and linear notations in that paper. The second part of our research focuses on formalizing and implementing these business rules. As a result, the main purpose of this paper is to present a logical approach for formalizing and implementing DBR modeled in our previous work. So, the paper has two purposes. The first purpose of our paper is to formalize DBR which were modeled by a logical approach. The second purpose of this paper is to implement the formalized DBR. In order to implement DBR, we introduce DBR System architecture which is developed using Java and Prolog+CG language.

Informally, a CG is a graph or network of concepts and conceptual relations where every arc links a concept node and a conceptual relation node. CG created by Sowa is one of the suitable modeling languages. Conceptual graphs constitute an expressive logical system designed for a direct mapping to and from natural language. They allow the representation of various kinds of knowledge and offer a graphical and linear notation for human readability and a graph mathematical structure for machine computability (Sowa, 1984; Sowa, 2000). In formally, a CG is a structure of concepts and conceptual relations where every arc are links a concept node and a conceptual relation node. A simple example of CG statement is depicted in Figure 2 using linear notation. This CG statement consists of three concepts (in brackets) and two relations (in parentheses).

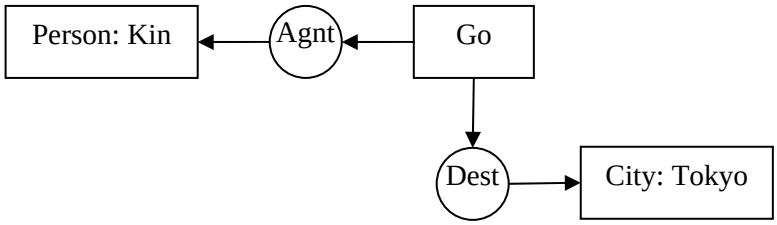

Figure 2 A CG Graphical Form for "Kin is going to Tokyo"

In order to represent dynamic processes and knowledge, Mineau (1998) proposed another approach. He proposed a representation approach for dynamic processes. This approach is more oriented toward the automatic translation of algorithms into an executable but declarative format. In his paper, Mineau uses the idea of processes to represent dynamic knowledge. Basically, Mineau's processes are one kind of executable conceptual graph formalism. Generally, Mineau's approach is an extension of Conceptual Graph theory.

After modeling DBR, these business rules are implemented by Prolog+CG language (see Figure 1). This language has been developed by A. Kabbaj in order to extend the Prolog language in two main directions: a conceptual extension allowing the representation of goals with conceptual graphs and the manipulation of simple and compounded CG (Kabbaj *et al.*, 2001; Kabbaj, 2005). The mentioned CG example has been rewritten using the Prolog+CG language as follows:

[Person: Kin]<-AGNT-[Go]-DEST->[City: Tokyo]

The purpose of this research is to present a logical approach for formalizing and implementing DBR modeled in our previous work. In this paper, we define the syntax of these business rules and implement them in connection to the prior work.

Our work makes several contributions related to the definition and implementation of business rules in the design of conceptual databases. The main contribution for implementing and defining the syntax of DBR is to consider and organize the relationships among rules in the business process so that people involved in the Information System can easily study the relationships among business rules and how business rules can be used in the business processes. They can use this study in decision making as well as in efficient processing of rules. For example, the effects of an action (Then part in DBR's structure) can be considered as a condition triggering the next business rules in a business process. One of the contributions of this paper is

to propose a logical approach which covers both modeling and implementation of DBR in the business processes area. In addition, the implementation of business rules will be easier and faster in Business Processes Management Systems (BPMS) using the DBR concept based on the main contribution and the DBR System which will be proposed and introduced later.

First we define the syntax of DBR and propose a DBR System architecture that clarifies our prior related works to consider the results of DBR's action, and second we implement these business rules using the proposed approach. In our research, business rules are treated as a central element of the DBR System.

The practicality of our approach is demonstrated with the case study of the Locomotive Maintenance's Business rules. The case study of the locomotive repairs factory at a large railway company demonstrates how the proposed approach can be applied to formalize and implement DBRs within the existing conceptual database.

The rest of the paper is organized as follows. In section 2, we review the relevant literature on business rules related to our research, and present the contributions of our system from different research area. Section 3 formalizes the syntax of DBR in BNF format. We also describe the structure of the proposed system in this section. The advantages of our system are demonstrated with the case study of the Locomotive Maintenance's Business Rules in Section 4. Finally, Section 5 concludes the paper with some observations and future issues related to the work presented here.

RELATED WORKS

A business rule dictates how the organization executes business decisions, processes and constraints essential to the company's strategy (Becerra, 2001; Vondrak, 2000). They are the specialized form of logic that expresses a constraint about the way a system or the people using it behave.

The term ‘‘business rule’’ has been used by different approaches in different ways. For example, business rules are "statements of goals, policies, or constrains on an enterprise's way of doing business" (Rosca et al., 1997) or they are defined as "statements about how the business is done, i.e. about guidelines and restrictions with respect to states and processes in an organization" (Bell et al., 1990). For the purpose of this work, we consider the definition of business rules as restrictions and conditions regarding processes in an organization. Thus, the adopted definition in this paper is Bell's definition.

There is a number of modeling languages and approaches for business rules modeling. The most popular modeling language is UML which was created through

jointing the efforts of researchers and commercial organizations. In this approach, business rules' modeling is fulfilled by the UML Object Constraint Language (OCL). Business rules are expressed in OCL statements (Eriksson and Penker, 2000). One of the most famous approaches is the report offered by the Business Rules Group (formerly the GUIDE Project on Business Rules). The GUIDE Project identifies terms and facts in natural language rule statements, and consequently, the expressiveness it allows is very high (Hay and Healy, 2003). Another method was created by Ross. The Ross Method is one of the most complete methodologies which model business rules. Ross has created the original graphical notations to represent business rules in a data model (Ross, 1997).

One of the most important activities in business rule-based system is the business rule implementation. There are a number of different technologies and tools available to support business rule implementation and maintenance. In its simplest form, business rule implementation may involve code written in a general-purpose language such as Java. These implementations usually take the form of a series of if-then statements. Evaluating the business rules then requires that all of these statements are assessed and the associated action is implemented.

The most common way to implement business rules is to use a rules engine. A business rule engine is a software system that helps manage and automate business rules. A rule engine can significantly improve the process by separating the rule evaluations from rule invocations. Furthermore, some engines allow a simpler way to express rules, using either a GUI or an English-like language instead of expressing them using a programming language. Table 1 lists several commercial products which employ their own rule engine. Some of the products have their rule engines integrated with the Java language. The rule languages implemented by the rule engines and their mechanisms of integration differ slightly.

The table highlights the lack of uniform standards for business rules modeling languages, repository formats, and architectures. In the tools listed in Table 1, business rules engines mainly focus on outlining the use of If-Then type business rules and applying Java language, but not on further consideration in how to construct, model and implement DBR precisely based upon logic theories. In addition, the implementation and modeling of business rules are still a challenge in the business modeling area (Valatkaite and Vasilecas, 2004). There is a lack of DBR implementation using a logic approach. In order to implement the modeled DBR using CG and Mineau's approach, we introduce a logic approach which integrates Prolog, the manipulation of conceptual graphs and Java. The model resulting from CG and

Mineau's approach is more useful than current tools because it permits the description of all components of DBR, and supports the execution semantics. The significance of these graphical languages is that it describes a state transition in terms of events, conditions and actions in an explicit way thus facilitating the active behavior specification. By using this approach, a uniform and complete representation of DBR is obtained, which constitutes an important part of information systems.

Table 1 Representative Tools for Business Rules

Product	Rule Type	Modeling Language	Implementation Language	For more information
Blaze Advisor	If-Then	English-like Structured Rule Language (SRL)	XML, LDAP, JDBC	(Blaze Advisor, 2007)
JES	If-Then	Jess Language	100% Pure Java Certified	(Friedman-Hill, 2000)
Infrex	If-Then-Else/ Else If	_____	C/C++/Java/C#	(Infrex, 2004)
ILOG	If-Then-Else	BAL & TRL	C++	(ILog, 2006)
Our System	When-If-Then	CG/Mineau's Approach	Prolog+CG/Java	(Authors, 2007)

DBR SYSTEM

In this section, we first formally define the syntax of DBR in Backus-Naur Form (BNF), and then describe the DBR System which makes use of the conceptual graphs as a conceptual modeling language and employs Java/Prolog+CG language for rules execution.

Formalization of DBR

Authors (2007) proposed to use Mineau's approach, an extension of Conceptual Graph theory, to model DBR. In the previous study, we modeled these business rules using a logical approach which is reasonably readable in linear or graphical form. Since DBR can directly be mapped to first order predicate logic, they can easily be implemented for business processes. As stated in introduction, these business rules

should be implemented after modeling. In order to enforce the DBR modeled in CG format, we implement these business rules using Prolog+CG program (see Figure 1). This language is a Java implementation of Prolog with extensions implementing a subset of the Conceptual Graph (CG) theory of John Sowa.

As mentioned in Section 1, a DBR has three parts including event, precondition, and action. Hence, the DBR's form may be illustrated as a three-part format. In that form, When indicates an event, If a precondition and Then an action. Each of those parts is modeled by using CG in our approach. In order to represent these parts in the Prolog+CG environment, we have modified the If/Then template which has been defined in the Prolog+CG language for our purpose. The syntax of the mentioned parts is defined and expressed in BNF as the following clauses:

*<When-Clause>::= " [**When**=[**Proposition**=" <Concept-Clause> "]]"*

*<If-Clause>::= "[**If**=[**Proposition**=" <Concept-Clause> "]]"*

*<Then-Clause>::= "[**Then**=" <Concept-Clause> "]"*

<Concept-Clause>::= <Concept> | <Concept> <Relation> <Concept-Clause>

<Concept>::= "[" <Concept Type> ":" <Instance Name> "]"

<Relation>::= "-" <Relation Type> "->".

In the above definition, it is noted that not all of a pair of <Concept Type> and <Instant Name> is semantically allowed as a <Concept> even though it is correct in syntax.

The Concept Type, Instance Name and Relation Type statements are defined and specified based on real cases. In the above representation, the concept type *When*, *If* and *Then* are made to be subtypes of *Proposition* type in Prolog+CG.

The basic structures of these presentations are concept graphs and concept relations. These forms can be modeled by means of CG concepts and relation types. As a result, a DBR can be formalized using the above clauses as the following clause:

*<DBR-Clause>::= "[" <When-Clause> "- **COND** ->" <If-Clause > "- **PRE**->" <Then-Clause>]"."*

where *COND* and *PRE* are the reserved words.

DBR System Archite Cture

The DBR System consists of four components, a user interface with which the user interacts, a DBR Engine that applies the rules using Prolog+CG language, a DBR repository that saves the rule-related information and a DBR builder that provides users with a graphic display environment in order to easily and conveniently write rules (Figure 3).

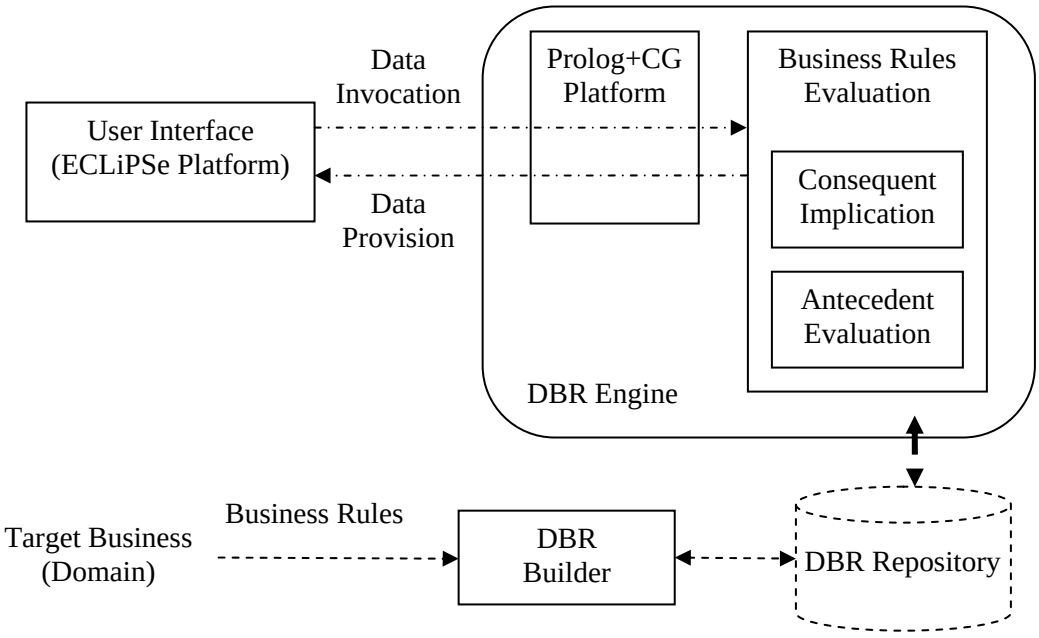

Figure 3 Overview of Components of DBR System

The major components of the proposed system are the following:

1. User Interface (UI)

   The interface connects users with the DBR engine. This component enables the user to interact with the second component of the architecture. Using the UI component, the user can enter and request information in graphical form. The user interface handles only display and input issues. The user selects the necessary information, and the system will run the user's query. This component is also used to display the results of the query.

2. DBR Engine

   The second component is the logical level or rule engine, which is responsible for computation and evaluation of the business rules according to the user's invocation and request. This component acts as a software system that helps manage and automate business rules. The DBR engine is a central component, for

it determines the representational power of the rule set that can be used. It is a system for executing a set of When-If-Then statements. In addition, Prolog+CG language acts as the role of platform in the second component. The DBR Engine has a two-layer structure including the Prolog+CG Platform and Business Rules Evaluation.

The first layer is a link between the User Interface and the Business Rule Evaluation layer. Here, Prolog+CG acts as the platform in our system. This language is an extended version of Prolog that supports Conceptual Graphs (CG). We also developed our system using the ECLiPSe platform in order to call the Prolog+CG modules when it is appropriate. ECLiPSe is an effective Constraint Logic Programming (CLP) System and is mainly backward compatible with Prolog+CG.

The second layer acts as a rule engine. This layer is composed of logic programs based on a business domain that aims at defining and specifying business rules. The Business Rule Evaluation layer is performed by Prolog+CG. In fact, the DBRs which have been modeled using Conceptual Graphs language are executed in the Business Rules Evaluation.

3. DBR Repository

The DBR repository is a repository that stores the rule-related information and supports the flexibility of rule expression. The DBR repository allows for the maintenance and management of business rules throughout their life cycle. This component represents only a part of the business rules repository used to store the information of business rules represented in CG form. The stored business rules in the repository component are determined based on the target system's specifications. In this case, the repository acts as a specific-domain repository.

4. DBR Builder

The Business Rule Builder has a default template and allows a developer to create business rules based on our definition. As displayed in Figure 4, the user or business analyst can add, edit and delete DBR stored in the rule repository based on the proposed format (When-If–Then construct) using this component .

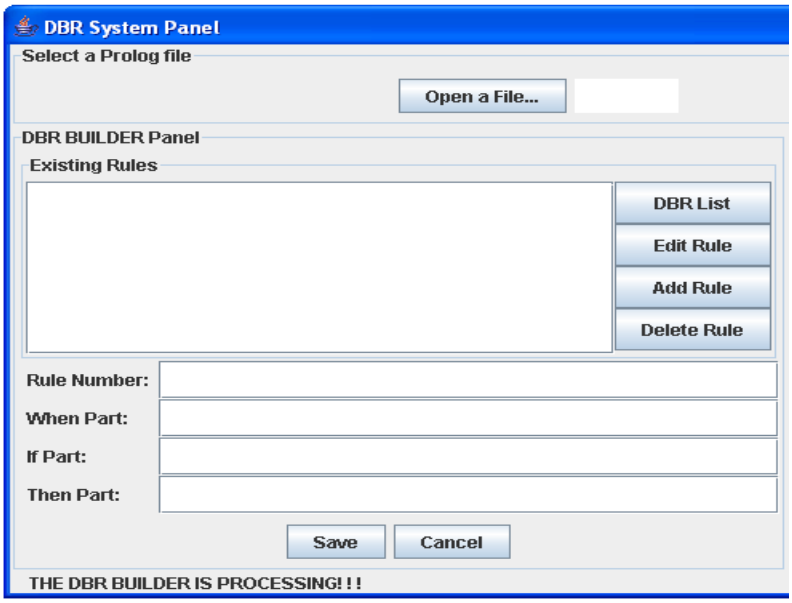

Figure 4 The DBR Builder Panel

In our approach, the DBR System receives and accepts the specifications and parameters from a target system and reads and processes the inputs which are stored in the repository component through the business rules builder component. The system creates and constructs a specific-domain DBR engine based on the target system's parameters at the end. In our approach, the input specifications and parameters of the system rely on the <Concept Type>, <Instance Name> and <Relation Type> statements in the DBR's format and are semantically specified regarding the target system.

CASE STUDY: LOCOMOTIVE MAINTENANCE AND REPAIRS

In order to illustrate our approach, we have applied the DBR System to the Maintenance & Repair organization of the Iranian Railway Company, the only railway system in Iran. The Iran Railway Company has a huge responsibility for transporting large number of goods and passengers. We have built a restricted scope (a small module) version of the running application where all the business rules are stated in the DBR System described in Section 3. We introduce a subset of business rules which may be relevant to the Locomotive Maintenance & Repair's Business Rules in the Iranian Railway Company.

The repair activities deal with overhauls and repairs, scheduled and unscheduled maintenance of General Electrics (GE), General Motors (GM), Alstom, and Hitachi locomotives. In this case, we try to incorporate our proposed approach into existing

business solutions. We are able to separate business rules as DBRs and place them into the DBR System. We consider the example of Locomotive repairs, where the locomotives can be fixed based on different kinds of repairs. The case study demonstrates how the proposed approach can be used to apply the DBR System within the existing conceptual database.

A locomotive is a diesel traction vehicle that pulls a train. It is repaired after the occurrence of a defect. Locomotive repairs are generally of four kinds in our case: "*slight*", "*minor*", "*casual*", or "*overhaul*" repairs. For example, overhaul repairs involve all parts of the locomotive being brought up to near new standards, while casual repairs only require, normally, the repair of one major component on the locomotive or defective part so it can be returned back to service. Such repairs are executed according to the locomotive manufacturer's recommendations and their related technical instructions. A diesel locomotive has five main parts including engine, main and assistant generator, compressor, bogy and engine (Traction Department of Iranian railway, 2006).

Figure 5 shows how the information is processed between the two repair shops. As shown in the figure, the repairs of locomotives are performed in two repair shops called the *running* and the *overhauling shop*. The running shop is the first place for locomotive repairs and services. The overhauling shop is in charge of overhaul and casual repairs. Locomotive repairs begin after the locomotive has arrived at the first repair shop. If the initial inspection of the locomotive deems it irreparable by the mentioned workshop, the locomotive is sent to the second repair shop for overhaul or casual repair ("*cold*" situation). The cold situation means that the locomotive is stopped for repairs and maintenances. After all necessary repairs have been completed, the locomotive is operational and ready to work ("*warm*" situation) for pulling the trains which carry goods and passengers.

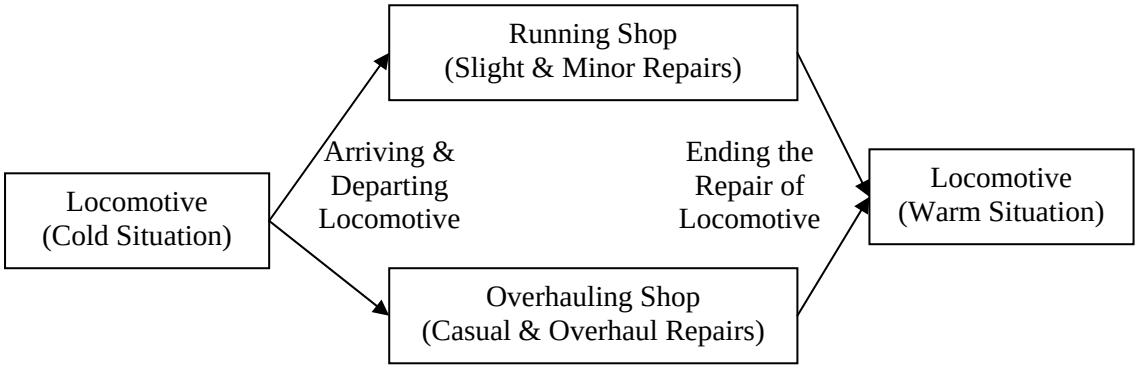

Figure 5 The Information Processing of Locomotives

Using the advantages of the proposed approach, we can track locomotive situations at the workshops and make better decisions to manage and control locomotive repairs based on the locations and repairs. In addition, the DBRs related to the new locomotives and unexpected defects can be updated and changed by using DBR builder

.

Instantiating Parameters

In our case study, we have extracted business rules from the existing system. Based on the Locomotive Repairs' Business Rules, there are four main DBRs that can be transformed into DBR form, including rule 1, 2, 3 and 4. These business rules specify the location, the situation of the locomotive based on the defect types and the waiting list for repairs. The structure of these rules related to the repairs shop is illustrated in Table 2.

Table 2 The Structure of Locomotive Repairs' DBR

DBR-Type / **DBR Parts**	**Rule 1**	**Rule 2**	**Rule 3**	**Rule 4**
When	Arriving locomotive	Departure of the locomotive to the related workshop.	Determine the repair type based on the defect.	The locomotive situation is cold.
If	The defects are related to a workshop	The defect type is related to a repair type.	The defect type is related to a workshop	The specified repair is performed at the workshop.
Then	Departure of the locomotive to the related workshop.	Determine the repairs type based on the defect.	Determine the repair type based on the defect.	The locomotive situation is warm.

The main DBRs are developed into sub-rules based on location, defect types (engine system, main generator, wiring system …), repair types (minor, slight, casual, and overhaul), situation (warm and cold) and locomotive number.

Based on the above explanation and the DBR System in Section 3, the parameters of the DBR System which are the *Concept Type*, *Instance Name* and *Relation Type* statements can semantically be instantiated as below:

Concept Type:

Act | Locomotive | Situation | Location | Repairs | LocaSitu | DefectType

Instance Name:

Act= {Arrive, Perform, Need, Depart, Select}

Locomotive= {GT26_900, GT26_901 … GE_900 … GE_910 … ALSTOM_900 …}

Situation= {Warm, Cold}

Location= {Running_Shop, Overhauling_Shop}

Repairs= {Overhaul, Casual, Minor, Slight}

LocaSitu= {Arrived, Depart, RepairWait}

DefectType= {OilServices, WaterServices, LightsInspection, BuggyServices, Bogy, WaterRadiator, OilCooler, WiringSystem, DriverCabin, TractionMotorSystem, TractionMotor, AuxiliaryGenerator, TurbochargedTurbine, EngineSystem, MainGenerator}

Relation Type:

LOC | IS | ON | AGNT | RSLT

where

LOC *= Location; relation type related to the place,*

IS *=Is; relation type "is",*

AGNT*= Agent; relation type related to an active animate entity which voluntarily initiates an act,*

THME *= Theme; relation type to a participant that may be moved, said, or experience, but is not structurally changed,*

RSLT*=Result; relation type related to an animate goal of an act, and*

ON*= On; relation type "on".*

Locomotive Repairs' DBR System

The architecture of Locomotive repairs' DBR System incorporating the proposed system for implementing DBR is presented in Figure 6. The target business (Domain) contains two repair shops. The system includes the DBRs which exist in the two repair shops, which have their own business rules specific to repair types. These business rules are then modeled using a logic language, conceptual graphs and Mineau's approach. As we mentioned in Section 3, the modeled business rules are executed by the DBR System, which consists of two major components: Antecedent Evaluation and Consequent Implication. Users can select some options and submit their data invocation to the system via their interfaces. The system sends output information to the user's interface after analyzing information using the DBR System.

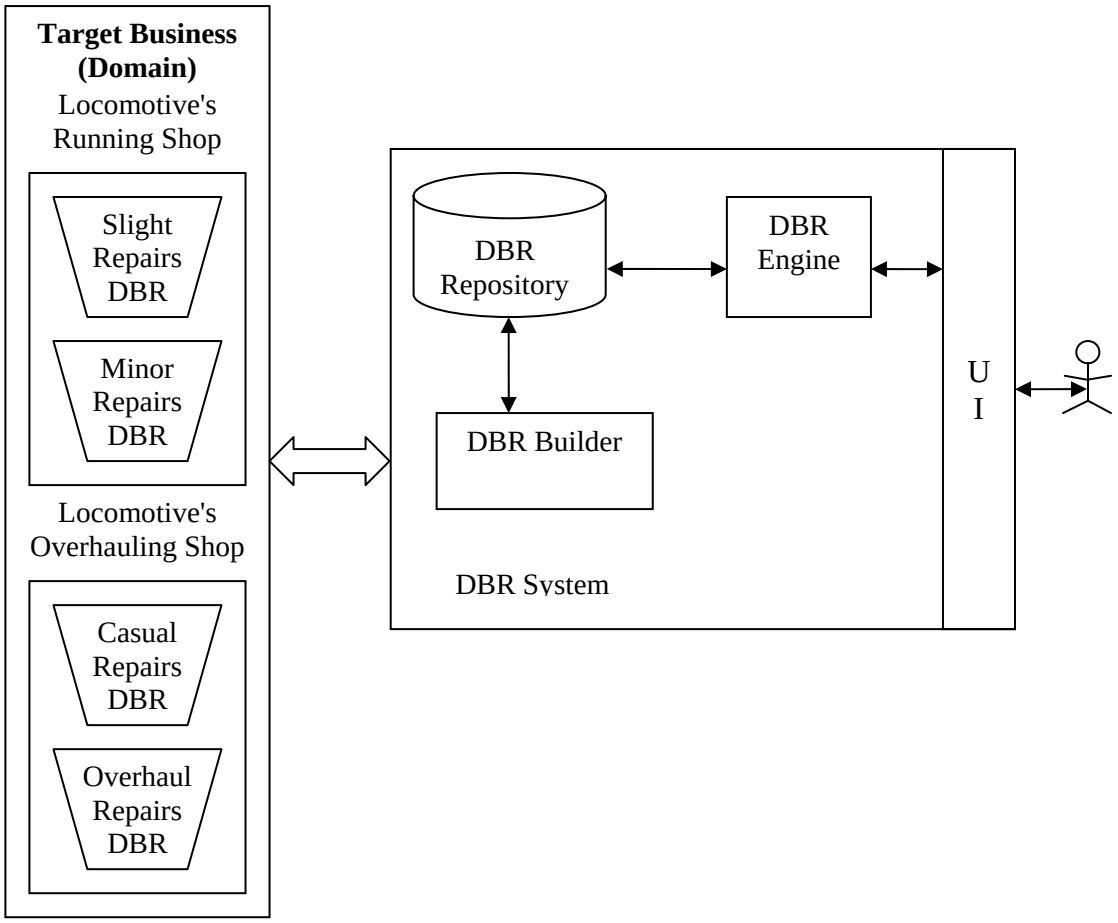

Figure 6 Locomotive Repairs' DBR System Architecture

After instantiating the parameters, the DBR's parts are defined based on the proposed format. For instance, assume locomotive ALSTOM_900 has arrived and the defect type is the engine motor. The defect type is related to overhaul repairs. Thus, the locomotive should be sent to the overhauling shop, where its situation is labeled as cold. The *When-Clause*, *If-Clause* and *Then-Clause* would be represented as the following clause:

When-Clause:

*[**When** = [**Proposition** = [Locomotive: ALSTOM_900]-*

*-**IS**-> [LocoSitu: Arrived]]]*

If-Clause:

*[**If** = [**Proposition** = [Locomotive: ALSTOM_900]-*

*-**IS**-> [DefectType: EngineSystem]]]*

Then-Clause:

*[**Then** = [Locomotive: ALSTOM_900]-*

*-**LOC**->[Location: Overhauling_Shop],*

*<-**AGNT**- [Act: Depart]]*

Consequently, the DBR which clarifies the situation of locomotive ALSTOM_900 in the Prolog+CG environment may be formalized in the following form:

DBR-Clause:

*[[**When**= [**Proposition**= [Locomotive: ALSTOM_900]-*

*-**IS**->[LocoSitu: Arrived]]]*

*-**COND**-> [**If** = [**Proposition**= [Locomotive: ALSTOM_900]-*

*-**IS**-> [DefectType: EngineSystem]]]*

*-**PRE**-> [**Then**= [Locomotive: ALSTOM_900]-*

*-**LOC**->[Location: Overhauling_Shop],*

*<-**AGNT**- [Act: Depart]].*

The underlined terms in the above statements indicate instance names.

As stated in Section 4.1, there are four main DBRs in the case study based on the location, the situation, the repair types and the defect types, and 150 sub-rules are

specified and defined. In the end, 450 sub-rules are extracted with regard to the locomotive number in the Prolog+CG program. For instance, the following DBRs are related to Locomotive GT26_901 with an Engine System defect type:

[[When = [Proposition = [Locomotive: GT26_901]-IS->[LocaSitu: Arrived]]

-COND->[If = [Proposition = [Locomotive: GT26_901]-IS->[DefectType: EngineSystem]

]-PRE->[Then= [Act: Depart]-AGNT->[Locomotive: GT26_901]-LOC->[Location: Overhauling_Shop]]].

[[When = [Proposition = [Locomotive: GT26_901]-ON->[Situation: Cold]]

-COND->[If = [Proposition = [Locomotive: GT26_901]-IS->[DefectType : EngineSystem]]-PRE->[Then = [Act: Need]-AGNT->[Locomotive: GT26_901]-RSLT->[Repairs: Overhaul]]].

[[When = [Proposition = [Locomotive: GT26_901] -

-ON->[LocaSitu: RepairWait],

-IN->[Location: Overhauling_Shop]]

-COND->[If = [Proposition = [Act: Select]-AGNT->[Locomotive: GT26_901]-THME->[Repairs: Overhaul]]

-PRE->[Then= [Act: Perform]-AGNT->[Locomotive: GT26_901]-RSLT->[DefectType: EngineSystem]]].

The purpose of the case study is to illustrate how existing DBRs are to be handled by the DBR System in a real scenario. In order to demonstrate our proposed approach, these business rules can easily be modeled and formalized using the mentioned format in Section 3. Therefore, the locomotive repairs' DBR System improves the existing system and works through the system’s physical process. It allows the system user to recognize the physical system shortcomings with a reliable approach and make proper decisions in solving problems in order to increase the benefits of the existing system.

The Usage of Locomotive Repairs’ DBR System

As described in subsection 4.1, repairs begin when the locomotive arrives for repairs. The user selects and enters the information upon the arrival of locomotives using the input information panel based on the locomotive specifications including

locomotive type (GT26, ALSTOM…), locomotive number (900, 901…) and locomotive defects (see Figure 7). The input screen offers lists and slider bars and it actually enables the writing in the Prolog+CG program of new assertions related to the arrival of locomotives. Therefore, this information is added as part of the system's knowledge base into the Prolog+CG file.

After the arrival, the wait for repairs, and the end of the repairs, some new facts are added in the program based on the DBR's Locomotive Repairs business rules. For example, the Prolog+CG program contains the following facts when Locomotive GT26_901 has arrived at the workshop:

[Locomotive: GT26_901]-ON->[Situation: Cold]

[Locomotive:GT26_901]-ON->[LocaSitu:RepairWait]

[Locomotive: GT26_901] -IN->[Location : Overhauling_Shop]

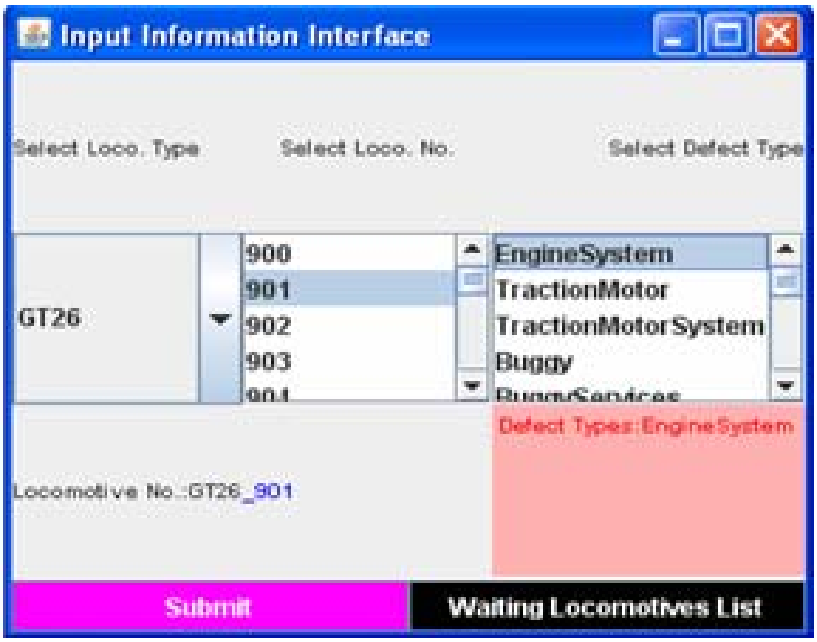

Figure 7 User Interface of the Locomotive Repairs System

The new locomotives are added to the waiting repairs' list after completing and asserting the entire locomotive information. In fact, this list is the first output of the system and is created using DBR related to the new locomotives. If a locomotive is selected from the list and then repaired, the business rules that specify the repairs of the locomotive are removed and then the locomotive becomes ready to work (Warm

situation). The new facts and DBR which are related to a warm locomotive are created in the Prolog+CG program.

After running the logical program and the analysis process in the output phase, correlative analyzed results can be displayed based on the user request. The user can select and request the information which has been explained in the output interface. A results-screen then presents the output list which can be created for any user's request. The results depend on the type of workshop and repairs. The screen also displays information related to locomotives situation and location.

For example, if the user selects the lists of arrived locomotives and overhaul repairs, the request lists would be as displayed in Figure 8. As shown in the figure, the stopped locomotives for the overhaul repairs are GT26_907, GT26_900, HITACHI_900, GL22_916, and GT26_919. Furthermore, the existing information of locomotive ALSTOM_908 is shown in this figure.

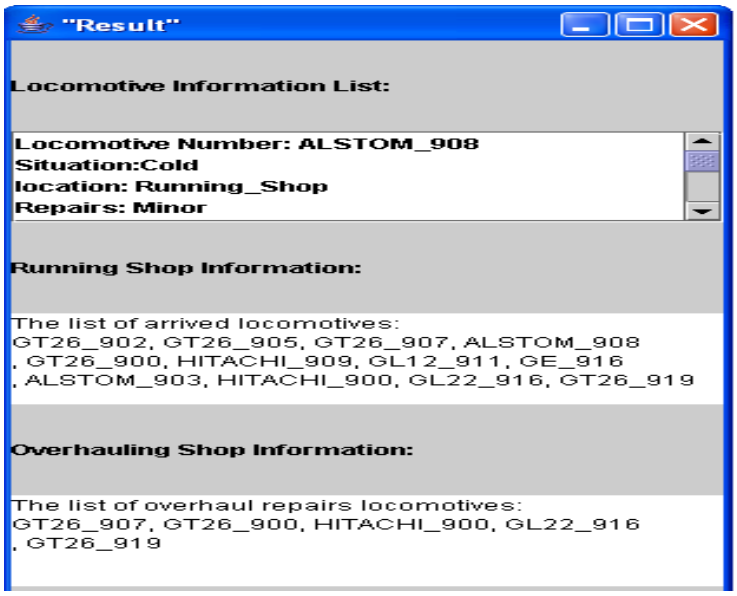

Figure 8 Result-Screen for the Locomotive Repairs System

CONCONCLUSION AND FUTURE WORKS

In order to reflect activities or dynamic behaviors of business processes, this paper has introduced a logical approach for formalizing and implementing Dynamic Business Rules (DBR). We formalized DBRs in BNF format and proposed a DBR System which supports our approach for implementing such business rules. The proposed system has a four-part structure, consisting of a user interface, a DBR engine, a DBR repository, and a DBR builder. In this system, DBR included in a business

domain are represented in CG format, and then are embedded together with data in Prolog+CG program. As a result, the system requires only Prolog+CG language as an inference engine to enforce business rules. As the business rules are represented declaratively in Prolog+CG, the rules can be immediately executed without any complex system implementation.

To illustrate the DBR System, the case study of the Iranian Railway locomotive Maintenance & Repairs organization has been presented. All modeling constructs referenced in the business rules (e.g. entity types and instance types) and the responses linked to user queries were created using the capability of Prolog+CG language. The results of this research show the considerable potential of the proposed logical approach as one possible alternative for the implementation of business rules in business processes.

The proposed approach has a number of limitations which also point out directions for future research. One for instance is the limited scope of the case study. Given the complexity of the issue, we can speculate that some practical upper limit exists in terms of systems size above which the proposed approach becomes inapplicable. Future work is required to explore this question. Another limitation of this research lies in the fact that not all types of business rules could be captured.

In our future work, we will extend and improve our approach in order to model and implement all types of business rules including static, fact and derivation rules. Since we focused on DBR in this paper, other classifications of business rules will be considered and modeled using existing logical approaches. In particular, we will focus on the ambiguous and vague terms and facts used in business rules that may be represented in CG format associated with a logic approach such as fuzzy logic.

REFERENCES

Authors (2007) Modeling Dynamic Business Rules using A Dynamic Knowledge Approach. *Industrial Engineering & Management System Journal (IEMS)*, 6 (1), 72-82.

Becerra, P. (2001). The living transaction. *In Intelligent Enterprise Magazine,* 4(8), 299-311.

Bell, J., Brooks, D., Goldbloom, E., Sarro, R., & Wood, J. (1990). *Re-engineering case study analysis of business rules and recommendations for treatment of rules in a relational database environment.* US West Information Technologies Group: Bellevue Golden.

Blaze Advisor. (2007). The Blaze Advisor Business Rules Management System: How it Works. Retrieved April, 2007, from http://www.fairisaac.com/NR/rdonlyres/ C3817720-3C36-4B43-9F65-3300B0B9AA29/0/advisorhow.pdf.

Bubenko, J. A., Brash, D., & Stirna, J. (1998). EKD user guide. Technical report, Kista, Dept. of Computer and Systems Science. *Royal Institute of Technology (KTH) and Stockholm University, Stockholm, Sweden*, Retrieved February 11, 1998, from ftp://ftp.dsv.su.se/users/js/ekd_user_guide.pdf.

Demuth, B., Hussmann, H., & Loecher, S. (2001). OCL as a specification language for business rules in database applications. *Proceedings of the 4th International Conference on the Unified Modeling Language, Modeling Languages, Concepts, and Tools*, Toronto, Canada, 2185, 104-117.

Eriksson, H-E & Penker, M. (2000). *Business modeling with UML: Business patterns at work*. New York: OMG Group, Wiley Computer Publishing.

Faget, J., Marin, M., Megard, P., Owens, V., & Train, L. (2003). *Business processes and business rules: Business agility becoming real*. Workflow Handbook, Lighthouse point, Florida: Future Strategies Inc.

Friedman-Hill, E. (2000). Jess, the Java expert system shell. SAND98-8206, Version 5.1 Distributed Computing Systems, Sandia National Laboratories, Livermore, CA.. Retrieved April 24, 2000, from http://web.njit.edu/all_topics/Prog_Lang_ Docs/html/jess/

Gottesdiener, E. (1999). Discovering an organization's knowledge: Facilitating business rules workshops. Williamsburg, Virginia, USA：Annual Meeting of the International Association of Facilitators.

Hay, D. & Healy, K.A. (2003). *Defining Business Rules, What are they* really?. Business Rules Group. Retrieved March 23, 2003, from http://www.businessrulesgroup.org/firstpaper/br01c01.htm.

Business Rules Group Hay, D. & Healy, K.A. (2003). *Defining Business Rules, What are they really?* Retrieved March 23, 2003, from http://www.businessrulesgroup.org/firstpaper/br01c01.htm

Herbst, H. (1996). Business rules in systems analysis: A meta-model and repository system. *Information Systems*, 21(2), 147-166.

ILog, ILOG RULES. (2006). Retrieved October, 2006, from http://www.ilog.com/products/rules/

Infrex. (2004). The business rules engine. Technical overview, Retrieved 2004, from http://www.tcs.com/0_products/infrex/downloads/ Infrex_Brochure.pdf

Kabbaj A., Moulin B., Gancet J., Nadeau D., & Rouleau O. (2001). Uses, improvements and extensions of Prolog+CG: Case studies. *Proceedings of ICCS'01*, 346-359.

Kabbaj, A. (2005). *Prolog+CG: User's manual, Version 2*. Retrieved December 21, 2005, from http://www.insea.ac.ma/CGTools/PROLOG+CG.htm.

Martin, J. & Odell, J. (1998). *Object-oriented methods: A foundation (UML edition)*. Upper Saddle River, NJ: Prentice-Hall.

Mens, K., Wuyts, R., Bontridder, D., & Grijseels, A. (1998). Tools and environments for business rules. Brussels： *ECOOP98*.

Mineau, G. (1998). From actors to processes: The representation of dynamic knowledge using conceptual graphs. *Proceedings of the 6th International Conference on Conceptual Structures*, Montpellier, France, LNAI 1453, 65–79.

Oelmann, A. (1991). Representing a system specification with a temporal dimension in an object-oriented language. *Proceedings of the 3rd International Conference on Conceptual Structures*, Montpellier, France, 498, 540-560.

Rosca, D., Greenspan, S., Feblowitz, M., & Wild, C. (1997). *A decision support methodology in support of the business rules lifecycle*. Annapolis, MD, USA： Proceeding of the International Symposium on Requirements Engineering (ISRE_97).

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

Ross, R.G. & Lam, G. (1998). *Putting business rules to work: A tutorial and workshop on business rules, business tactics and policies*. Chicago：Business Rule Forum, Technology Transfer Institute.

Sowa, J.F. (1984). *Conceptual structures: Information processing in mind and machine*. Boston, MA ：Addison-Wesley Longman Publishing Co..

Sowa, J.F. (2000). *Knowledge representation: Logical, philosophical, and computational foundations*. California ：Books Cole Publishing Co..

Taveter., K. & Wagner, G. (2001). Agent-oriented enterprise modeling based on business rules. *Proceeding of 20th International Conference on Conceptual Modeling (ER2001)*, 2224, 527-540.

Terry, H. (2005). Fact-orientation meets agent-orientation. *Proceeding of 6th International Bi-Conference Workshop (AOIS 2004)*, New York, USA, 97-109.

Traction Department of Iranian railway. (2006). *GM locomotive manual*. Retrieved April, 2006, from http://keshesh.rai.ir/eng/Site.aspx.

Valatkaite, I. & Vasilecas, O. (2004). Automatic enforcement of business rules as ADBMS triggers from Conceptual Graphs model. *Information Technology and Control*, Kaunas, Technologija, 2(31), 36 – 42.

Vondrak, I. (2000). Business process modeling and simulation for quality management. *Proceeding of 14th European Simulation Multi-conference (ESM 2000)*, 375-379.

Zaniolo, C., Ceri, S., Faloutsos, C., Snodgrass, R., Subrahmanian, V.S., & Zicari, R. (1997). *Advanced database systems*. San Francisco, CA: Morgan Kaufmann.