Business Rules Design and Refinement using the XTT Approach

Grzegorz J. Nalepa
Institute of Automatics,
AGH – University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
gjn@agh.edu.pl

Abstract

This paper discusses practical development of *business rules* (BR) systems. The main conceptual foundations of BR are discussed, and selected tools for BR design and implementation are described. An application of a rule-based systems design tool (Mirella Designer) for BR design is proposed. Business rules visual representation using XTT in Mirella Designer is both transparent and compact. The underlying logical formulation of XTT opens up possibility of formal analysis and refinement performed on-line, during the design.

Introduction

This paper is concerned with some practical applications of Knowledge-Based Systems (KBS) in the field of business software. KBS are an important class of the so-called intelligent systems originating from the field of Artificial Intelligence (Russell & Norvig 2003). However, building real-life KBS is a complex task. Since their architecture is fundamental different from classic software, specific development methodologies, referred to as *knowledge engineering*, are required. In AI *rules* are probably the most popular choice for building knowledge-based systems, that is the so-called rule-based expert systems (Jackson 1999; Ligęza 2006). Rule-based systems (RBS) are used extensively in practical applications, e.g. decision support. They are considered one of the most important classes of KBS.

Today the process of business software development becomes more and more complicated. Some of the reasons for this are: growing demand for versatile software, and complex, often contradictory, user requirements. In order to approach this problem, there is a growing need for flexible development approaches, and efficient computer tools.

Recently, a new approach that can aid in business software development, has been gaining an increasing popularity. It is based on the idea of using the so-called *Business Rules* (Ross 2003; von Halle 2001) to represent knowledge about the core logic of the business software. For AI researchers business rules (BR) can be considered a rediscovery, or a new application of well-established AI technology of RBS. It could be said, that these days software engineering becomes more knowledge-based. This opens some opportunities for it to benefit from knowledge engineering solutions, including advanced conceptual tools, such as declarative knowledge representation methods, knowledge transformation techniques based on existing inference strategies, as well as verification, validation and refinement methods.

This paper discusses practical development of business rules-based systems. In the second section main conceptual foundations of BR are discussed. Then, in the third section selected tools for BR design and implementation are described. The paper is written from an AI-oriented perspective. This is why in the next section an application of a RBS design tool (Mirella Designer) for BR design is proposed. The XTT design approach serves as the basis for the Hekate Project, presented in fifth section. It aims at developing a new, knowledge-based methodology for software development. The paper ends with concluding remarks; they include future directions of the Mirella project development.

Business Rules Concepts

Business Rules approach (Ross 2003; von Halle 2001) is based on concepts borrowed from knowledge engineering (KE) and rule-based systems. It is becoming an important approach in business application development, especially on the Java platform. This section deals with some conceptual foundations of this approach.

A classic description of the main principles of the approach is given in (Ross 2003). According to it, rules should be: written and made explicit, expressed in plain language, motivated by identifiable and important business factors, single sourced, specified directly by people who have relevant knowledge, managed, and built on facts, and facts should build on concepts as represented by terms. Rules should also exist independent of procedures and workflows.

There are number of *rule types* identified in the BR approach, such as: reactive rules (event-condition-action rules), transformation rules (functional-equational rules), derivation rules (implicational-inference rules), also ones restricted to facts ("premiseless" derivation rules) and queries ("conclusionless" derivation rules), as well as integrity-constraints (consistency-maintenance rules).

Business rules design uses some established *visual representations*. Depending on the design approach these are some classic tools such as simple propositional decision tables, or some high-level conceptual tools such as URML

(see the next section for more in-depth analysis).

There are attempts to officially define main aspects of the approach. A good example is the *Semantics of Business Vocabulary and Business Rules Specification*, see (OMG 2006).

Observations

From the point of view of formal knowledge engineering, some major issues can be pointed out. They are related to: logical foundations, visual representation, and formal analysis and verification of BR systems.

The first problem concerns the *logical foundations* of BR systems. From a point of view of classical KE, a rule-based expert system consists of a knowledge base and an inference engine. The KE process aims at designing and evaluating the knowledge base, and implementing a proper inference engine. The process of building the knowledge base involves the selection of a knowledge representation method, knowledge acquisition, and possibly low-level knowledge encoding. In order to create an inference engine a reasoning technique must be selected, and the engine has to be programmed.

In the formal analysis of RBS (Ligęza 2006) some important aspects of the design and implementation are identified:

1. rulebase design, including:
   - the formal logical language of the representation,
   - formal syntax of the representation method,
   - representation expressiveness, which is often a function of the expressiveness of the underlying logic, and particular rule syntax.
2. inference engine implementation, including:
   - inference strategy,
   - interpreter model, including rule matching method,
   - conflict resolution algorithm.

Unfortunately it can be observed, that common approaches to BR tend to mix these formal aspects. The concept of "business rules types" is both misleading and imprecise. A proper formal analysis of BR should provide a more adequate classification of BR.

Formulation attempts such as (OMG 2006) are definitely not satisfactory. The main source of problems seems to be the way in which the rich semantics of business rules is being captured. In this document the problem of defining the business rules vocabulary, seems to fall into the domain of natural language processing. The other problem is the lack of explicit choice of logical calculus. The logical apparatus found in the document seems to be a mixture of first order predicate logic with some modal operators. The apparatus is described in a very vague language, far from mathematic precision. This poses severe problems when it comes to formal description of BR syntax, semantics, and inference.

The second problem is related to the *visual representation* used in the design of BR systems. Visual representations used, have scalability problems (it's easy to draw diagrams of several rules, but it becomes very difficult to cope with tens of rules). Lack of well-defined formal foundations of these representations leads to problems with automatic transformation of visual model to the logical one. When introducing a visual representation, it is important to decide what is the main reason for its introduction. Is it just some useful conceptualization tool, helpful in the design; or is it an integral part of the design process.

The third problem concerns the *formal analysis end verification* of BR systems. As the number of rules exceeds even relatively very low quantities, it is hard to keep the rule-base consistent, complete, and correct. These problems are related to knowledge-base verification, validation, and testing. The selection of appropriate software tools and programming languages is non-trivial either.

These issues are very rarely considered in the BR design. It seems that analysis (where issues such as verification, validation, and evaluation are even not properly separated) is simply considered testing. So the analysis of the *knowledge base* is implicitly substituted by testing of the *implementation*. However, in the KE approach, a proper *formal* analysis of the knowledge base minimizes the need for testing. Such an analysis, along with formal specification of the knowledge representation syntax and semantics, can eliminate not just syntactic errors, but also semantic ones.

Computer Tools for Business Rules

There has been a very active development of computer tools for BR in recent years. Today, there are number of BR-related solutions available. They fall in several different classes identified below:

- shells and inference engines – often referred to as *Business Rules Engines*, these are usually classic expert system shells extended in order to support the BR approach.
- markup languages – they are based on XML; depending on their function they support the BR representation or exchange between different systems.
- dedicated representation methods – these are dedicated conceptual tools for visual modelling of BR.
- integrated solutions – also referred to as *Business Rules Management Systems*, offer number of tools supporting the designer from the rule design phase, to the implementation in the selected target language, usually Java.

Selected examples of tools in these classes are characterized below.

Tool Examples

A good example of *shell and inference engine* is *Jess* (`www.jessrules.com`). It is a Java-based environment for creating rule-based expert systems. Jess provides a classic expert systems shell, including a rule inference engine and a language for defining the rulebase. In order to process rules Jess uses an enhanced version of the well-known Rete algorithm. The most important feature of Jess is its tight integration with Java. It also provides means to interface and process Java objects. This makes embedding Jess into larger Java applications easy. A new feature of the Jess version 7.0 is the integration with the popular Eclipse IDE.

Currently the single most important *markup language* for BR is *RuleML* (`www.ruleml.org`), an XML-based rule markup language. RuleML encompasses a hierarchy of rules, including all of the important classes of BR. It has a strong community and commercial support. However, in future its status could change, due to the efforts of *W3C Rule Interchange Format (RIF) Working Group* (see `http://www.w3.org/2005/rules/`).

There are not many *dedicated representation methods* for BR. However, a good example is *URML* (Lukichev & Wagner 2005), the UML-Based Rule Modeling Language, which allows visual rule modeling based on UML class models. URML supports the modeling of derivation rules, production rules and reaction rules. A rule is represented graphically as a circle with a rule identifier. Incoming arrows represent rule conditions or triggering events, outgoing arrows represent rule conclusions or produced actions. It is an important effort, that tries to integrate the rule design methodology with UML and possibly MDA. However, it seems to have major representation scalability problems.

There are number of *integrated solutions* available, such as Java-based *JRules*, or number of others. *JRules* is a Java and XML-based library. It is a part of an integrated design environment made by ILOG (`www.ilog.fr`). This integrated environment supports the development and optimization of expert systems. It uses internal knowledge representation language to describe RBS. It contains multiple development tools, including ILOG JRules.

A different example is LPA *Visirule* (`www.lpa.co.uk`) tool. It is a visual design tool for developing and prototyping expert systems, including business and decision support applications. A principal idea is to support the designer by a graphical flowchart representing the underlying decision logic. The chart can be automatically translated into a lower level logic-based representation. Each element of the chart has a corresponding logical fact. The chart can be then executed in Flex, the frame-based hybrid expert systems shell. It provides various inferencing engines, including: a forward-chaining and backward-chaining rules. The environment is built upon the Win-Prolog compiler.

Excel-based tools are becoming an important class of BR tools. They are based on the idea of using popular Excel spreadsheet as a tool to create decision tables containing rules. Some good examples of tools in this class include *ARulesXL*, or *OpenRules*, and *Drools*.

Drools (*www.drools.org*) is a framework for building forward chaining inference expert systems. It is also built upon the Rete algorithm. Drools is implemented in Java, and integrated with the Java building tools. It can be seen as metaprogramming tool. It generates source in one of the selected languages, e.g. Java, from a conceptual description encoded in XML. This description includes declarative parts (rules) and embedded procedural code in the target language. Rulebase design support is provided by Excel spreadsheet-based tool. The tool is used to design simple decision tables, that can be exported via the text file, and converted to the Drools XML. Drools has been recently bought by *JBoss* and incorporated into the JBoss Java platform.

OpenRules (`www.openrules.com`) is an integrated

Debt Research Rules

IF Mortgage Holder	AND Outside Credit Score		AND Loan Holder	AND Credit Card Balance		AND Education Loan Balance		AND Internal Credit Rating			AND Internal Analyst Opinion	THEN Debt Research Recommendations	Rule Source
	Min	Max		Oper	Value	Oper	Value						
Yes												High	Mary
No	100	550										High	Joe
No	550	900	Yes	<=	0							Mid	Janet
No	550	900	Yes	>	0	>	0					High	Joe
No	550	900	Yes	>	0	<=	0	A	B	C		High	Joe
No	550	900	Yes	>	0	<=	0	D	F			Mid	Joe
No	550	900	No	>	0							Low	Joe
No	550	900	No	<=	0	<=	0					Low	Joe
No	550	900	No	<=	0	>	0	D	F			High	Joe
No	550	900	No	<=	0	>	0	A	B	C		Low	Joe
											High	High	Joe
											Mid	Mid	Joe
											Low	Low	Joe

Figure 1: Selected Loan Business Rules

environment for BR. It provides several tools, including rule learner, engine, and solver. Rule editing is provided by external spreadsheet, such as Excel, or OpenOffice. Rulebase is stored in a BR repository. It also supports the presentation layer, implemented with web forms.

ARulesXL (`www.arulesxl.com`) is a design tool developed by the well known Prolog solutions provider – *Amzi!*. What makes the tool different from others mentioned here, is that it relies on Prolog, not Java, as the main knowledge processing language. The rule design is done with use of the Excel spreadsheet too.

Observations

An overview of BR tools mentioned above allows for observing some important areas of development in the field of practical design and implementation of BR.

- *Visual knowledge representation* is important, in order to support the design process, and provide improved transparency of the rulebase. This is why, many tools provide some kind of rule visualization, from simple decision trees, such as LPA, to complex representation given by URML.
- *Machine readable rule encoding* provided by the XML-based representations is becoming a must. RuleML seems to emerge as a standard language here.
- *Automatic code generation* using business rules as the high level prototype becomes a common approach, e.g. Drools, or OpenRules.
- *Use of some well established tools* such as Excel aims at attracting large user base, especially people that do not necessarily come from the field of computer science.

While these developments aim at improving the design process of BR applications, it seems that they fail to provide effective solutions. There seem to be two main problems.

The first one is unsuitable knowledge representation used during the design. The basic representation, such as simple decision tables, is then used in a design tool which is often inefficient (such as Excel). What is even more important is the fact, that these tools do not try to overcome the so-called *semantic gap* (Merrit 2004) between declarative design and procedural implementation. Tools like Drools provide a kind of rule meta-language embedded in the implementation lan-

guage (e.g. Java). This does not seem to be an efficient solution, since it mixes two different semantics.

The second one concerns analysis and verification of BR. It is bizarre that there are virtually no specialized evaluation tools. Some simple testing or syntax checking features during the design (which are not even present in case of Excel-based solutions) are a step backwards, compared to some rule-based evaluation and analysis tools developed several years ago (e.g. see (Gerrits & Spreeuwenberg 2000)).

In the following section a BR example design is presented. The design is discussed using the *OpenRules*, since it is a typical Excel-based environment.

A Business Rule Base Example

In this section a simple but illustrative example of a business rulebase is discussed. It is one of the "benchmark" examples from the *OpenRules* environment. The full example in the XLS format can be freely downloaded and examined, see `www.openrules.com/docs/xls/Loan2.xls`.

The example describes a simple decision support system, used in a bank, for determining whether a client should be granted a loan. As it was mentioned in the previous section, the *OpenRules* environment uses Excel spreadsheet as the "design" tool for business rules, so the system is contained in a multipage spreadsheet. It contains: a glossary (attributes description), some test data, rules, inference control information, and UI information.

The system has 16 simple rules and 15 attributes. It uses standard forward-chaining rules. The basic idea is to determine whether a customer is eligible to get a requested loan, taking into account his financial history and the experience of the bank. During the decision making process the system takes into account customers income, and current debt. The final loan recommendation is then authorized by a bank manager. An example sheet, containing rules for debt research, is shown in Fig. 1.

Using this example several observations can be made. First of all there is a clear *structure* in the rule base, related to different phases or contexts of the decision making process. However, the "design" tool has no facilities to properly model this structure, especially when it comes to the relations between different contexts. In the example the conceptual, rule-related parts, are mixed with pseudo-code, or parts of Java code. The "model" itself can be considered barely readable, especially to a non-programmer, since it uses many programming conventions implicitly.

The next section discusses how the system can be designed using the solutions provided by the prototype MIRELLA Designer.

Application of the Mirella Designer

MIRELLA Designer is based on several concepts:

- an integrated design and implementation process, containing: conceptual, logical, and physical design,
- the XTT (*eXtended Tabular Trees*) knowledge representation, that supports the logical design (Nalepa 2004),

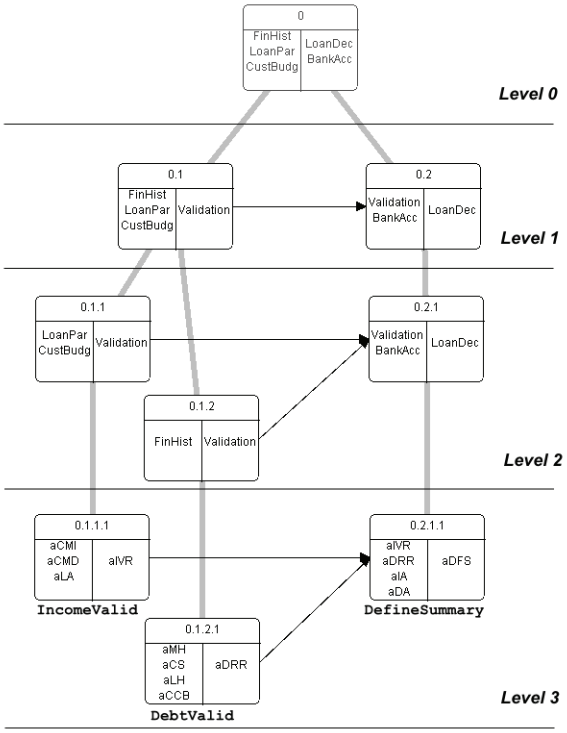

Figure 2: ARD Conceptual Design.

- the ARD (*Attribute Relationship Diagrams*) knowledge representation, supporting the conceptual design (Nalepa & Ligęza 2005),
- a direct translation to a formal, Prolog-based representation equivalent to the XTT knowledge base (Nalepa & Ligęza 2006),
- an *on-line* knowledge analysis, including formal verification, and refinement of the knowledge base (Ligęza & Nalepa 2005).

These concepts have been previously discussed, so here the main parts of the analysis of the example will be given.

The Conceptual Design

In the first design phase a complete specification of system attributes has been formulated. The *physical* attribute specification in the XTT/ARD contains:

- attribute names e.g. `aDebtResearchRecommend`,
- abbreviated attribute names, suitable for Prolog implementation, e.g. `aDRR`,
- attribute types, e.g. `symbolic`, and
- specification of attribute value domains, e.g. `[Low,Mid,High]`.

During the the ARD phase, the so-called *conceptual* attributes are also used. These are the *generalized* attributes, that are specified, during the conceptual design, into the physical ones.

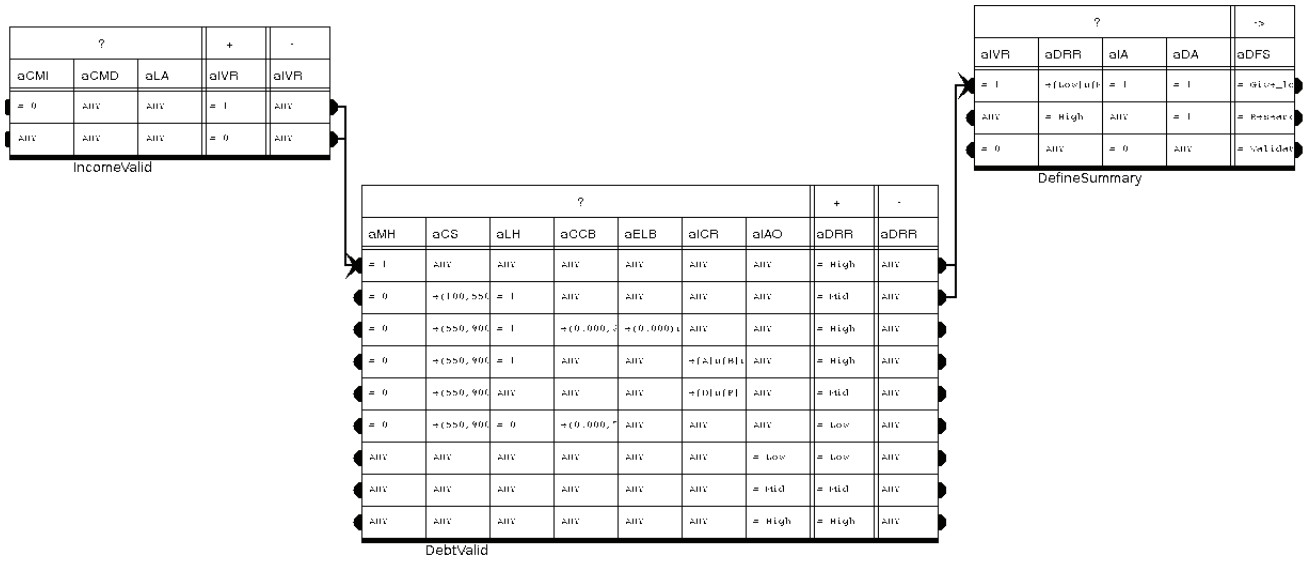

Figure 3: XTT Logical Design.

The complete design can be observed in Fig. 2. ARD provides a hierarchical model. At the top-most *Level 0*, the most basic relation between some general input (such as `CustomerBudget`) and output (such as `LoanDecision`) attributes is visualized. At every subsequent level, this relation is become more specific. At the last, bottom level, the diagram gives full specification of relations between physical system attributes. For more details on syntax and semantics of ARD see (Nalepa & Ligęza 2005).

The last level of the ARD is a *table scheme specification* for the XTT logical design method. During this phase, rules are built using specific attribute values.

The Logical Design

Using the results of the conceptual analysis, the actual design of the rule base is put forward, using the XTT representation method (Nalepa 2004; Nalepa & Ligęza 2006). In this method the rule base is visualized using tables grouping rules operating in the same context (on the same attributes) into a tree-like structure. What is important, the representation is based on an expressive *attributive* language, with use of *non-atomic* attribute values. This allows for a more compact representation.

The XTT logical design is presented in Fig. 3. In the figure three tables are visible. The first table, the so-called *root table* is a start point for the inference process. In a general case, the inference process is non-monotonic, since XTT allows for dynamic modification of the knowledge base, using Prolog-like *assert/retract* statements in the rule decision part. The tables are slightly minimized compared to the original ones, thanks to the expressiveness of the XTT language. The figure is somehow visually compact, due to certain limitations of current version of the CASE tool (e.g. not all attribute values are fully visualized). However, it does offer transparent visualization of the decision making process along with rule inference.

There are number of specific issues that need to be taken into consideration when handling the vague semantics of business rules. Some of important problems include: implementing calculations on attribute values (including functions), testing some functional dependencies between attributes. In the current version of XTT these are not handled directly and explicitly. This is due to the fact, that introducing such features largely complicates the verification issue. However, they are quite easily implemented, using additional Prolog procedures, that can be triggered by fired rules. This solution will be replaced in future versions of XTT with knowledge representation enhancements.

The Physical Design

The XTT representation is automatically transformed into a Prolog-based representation (Nalepa 2004; Nalepa & Ligęza 2006). Pure Prolog clauses are not used here, since Prolog uses backward chaining only. A meta-interpreter for forward chaining rules is provided. The Prolog-encoded rule-base is an executable prototype of the system.

The On-Line Analysis and Refinement

The automatic transformation of XTT to Prolog, which can be done at any stage of the logical design, allows for an *on-line* evaluation of the rule base. In the Mirella Designer number of Prolog based verification plugins are provided. They verify some important formal properties of the system, such as redundancy, completeness, or determinism. The details of the verification procedures have been presented in (Ligęza & Nalepa 2005; Nalepa & Ligęza 2006).

What is important in this case, is the possibility of *formal* analysis, *during* the design. The possibility of gradually designing the rule base, at the logical level, while having the

possibility of dynamic generation of an executable Prolog-based prototype, allows for conducting a refinement of the rulebase.

Business Rules in the Hekate Project

The HEKATE Project (see `hekate.ia.agh.edu.pl`) aims at incorporating some well established KE tools and paradigms into the Software Engineering (SE) domain. The project is based on experiences with the MIRELLA project but it extends its RBS perspective towards SE. The HEKATE design process offers important capabilities of formal verification, and gradual refinement of software from the conceptual model stage to an executable prototype.

A principal idea in this approach is to model, represent, and store the *business logic* using advanced knowledge representation methods taken from KE. The logic is then encoded with use of a Prolog-based representation. The logical, Prolog-based core (the *logic core*) would be then embedded into a business application. The remaining parts of the business or control applications, such as interfaces, or presentation aspects, would be developed with a classic procedural programming or object-oriented languages e.g. Java.

The HEKATE project aims at applying this methodology to practical design and analysis of real-life software. The projects focuses on wide class of software, namely two very different "benchmark" classes, that is: general business software based on the *business logic*, often expressed by business rules, low-level control software, possibly for the embedded control systems, based on a *control logic*.

In HEKATE a business rules rulebase is designed with XTT, and embedded into the application as the declarative *logic core*. The rulebase is then used by the HEKATE runtime engine. The core is integrated on runtime with other components e.g. providing interfaces.

Concluding Remarks

In the paper some important aspects of practical design and implementation of business rules (BR) have been discussed. Some common design and implementation tools have been presented, and contrasted with the Mirella Designer. The original contribution of this paper is the demonstration how a RBS design tool (Mirella Designer) can improve the design of BR and allow for formal analysis. Mirella is based on the concept of XTT knowledge representation and design. Business rules visual representation using XTT in Mirella Designer is both transparent and compact. The underlying logical formulation of XTT opens up possibility of formal analysis performed on-line, during the design. Rule processing in Prolog is flexible and allows for automatically generating an executable prototype.

However, it is important to emphasize, that Mirella Designer is still a prototype tool, built as a proof of concept for the XTT knowledge representation and design. Currently it cannot be used as a replacement for mature business rules tools. A new, improved version of the Designer is in the works. Mirella in its current state was successful as a proof of concept. Planned areas of extension and improvement have been identified, such as: robust business rules support, automatic knowledge acquisition facilities, optional backward-chaining, possibly fuzzy rules support, and even more extended verification capabilities, incorporation of system validation is also under consideration.

References

Gerrits, R., and Spreeuwenberg, S. 2000. Valens: A knowledge based tool to validate and verify an aion knowledge base. In *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000*, 731–738.

Jackson, P. 1999. *Introduction to Expert Systems*. Addison–Wesley, 3rd edition. ISBN 0-201-87686-8.

Ligęza, A., and Nalepa, G. J. 2005. Visual design and on-line verification of tabular rule-based systems with xtt. In Jantke, K. P.; Fähnrich, K.-P.; and Wittig, W. S., eds., *Marktplatz Internet: Von e-Learning bis e-Payment : 13. Leipziger Informatik-Tage, LIT 2005*, Lecture Notes in Informatics (LNI), 303–312. Bonn: Gesellschaft fur Informatik.

Ligęza, A. 2006. *Logical Foundations for Rule-Based Systems*. Berlin, Heidelberg: Springer-Verlag.

Lukichev, S., and Wagner, G. 2005. Visual rules modeling. In *Sixth International Andrei Ershov Memorial Conference PERSPECTIVES OF SYSTEM INFORMATICS, Novosibirsk, Russia, June 2006*, LNCS. Springer.

Merrit, D. 2004. Best practices for rule-based application development. *Microsoft Architects JOURNAL* 1.

Nalepa, G. J., and Ligęza, A. 2005. Conceptual modelling and automated implementation of rule-based systems. In Krzysztof Zieliński, T. S., ed., *Software engineering : evolution and emerging technologies*, volume 130 of *Frontiers in Artificial Intelligence and Applications*, 330–340. Amsterdam: IOS Press.

Nalepa, G. J., and Ligęza, A. 2006. Prolog-based analysis of tabular rule-based systems with the xtt approach. In Sutcliffe, G. C. J., and Goebel, R. G., eds., *FLAIRS 2006 : proceedings of the nineteenth international Florida Artificial Intelligence Research Society conference : [Melbourne Beach, Florida, May 11–13, 2006]*, 426–431. FLAIRS. - Menlo Park: Florida Artificial Intelligence Research Society.

Nalepa, G. J. 2004. *Meta-Level Approach to Integrated Process of Design and Implementation of Rule-Based Systems*. Ph.D. Dissertation, AGH University of Science and Technology, AGH Institute of Automatics, Cracow, Poland.

OMG. 2006. Semantics of business vocabulary and business rules (sbvr). Technical Report dtc/06-03-02, Object Management Group.

Ross, R. G. 2003. *Principles of the Business Rule Approach*. Addison-Wesley Professional, 1 edition.

Russell, S., and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition.

von Halle, B. 2001. *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. Wiley.