HeKatE Rule Runtime and Design Framework*

Grzegorz J. Nalepa, Antoni Ligęza,
Krzysztof Kaczor, Weronika T. Furmańska

Institute of Automatics,
AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
{gjn,ligeza,kk,wtf}@agh.edu.pl

Abstract. The HeKatE Project aims at providing a complete hierarchical design and implementation framework for rules. Principal ideas of the project include an integrated hierarchical design process covering stages from conceptual, through logical to physical design. These stages are supported by specific knowledge representation methods: ARD+, XTT2, and HMR. Practical design and implementation support using these methods is provided by the HeKatE design environment called HaDEs. A complete custom rule runtime environment HeaRT is provided to run XTT2 rule bases. The engine offers a number of rule-base quality analysis plugins.

1 Introduction

Rule-based systems [1] constitute one of the most powerful and most popular class of intelligent systems. They offer a relatively easy way of knowledge encoding and interpretation. Formalization of knowledge within a rule-based system can be based on mathematical logic or performed on the basis of engineering intuition. Practical design methodologies for intelligent systems remain a field of active development. Developing such a methodology requires an integration of accurate knowledge representation and processing methods [2], as well as practical tools supporting them. Some of the important features of such approaches are: scalable visual design, automatic code generation, support for existing programming frameworks. At the same time quality issues, as well as a formalized description of the designed systems should be considered.

In this paper a new rule runtime and design framework is presented. The *HeKatE* project (see `hekate.ia.agh.edu.pl`) aims at providing an integrated methodology for the design, implementation, and analysis of rule-based systems [1,3]. An important goal of the project is to allow for an easy integration of knowledge and software engineering methods, thus providing a *Hybrid Knowledge Engineering* methodology. The project delivers new knowledge representation methods and practical tools supporting the design process.

* The paper is supported by the HeKatE Project funded from 2007–2009 resources for science as a research project.

The main paradigm for rule representation, namely the eXtended Tabular Trees (XTT) [4], ensures high density and transparency of visual knowledge representation. Contrary to traditional, flat rule-based systems, the XTT approach is focused on *groups of similar rules* rather than single rules. Such groups form decision tables which are connected into a network for inference.

A top-down design methodology based on successive refinement of the project is introduced. It starts with development of an Attribute Relationship Diagram (ARD) which describes relationships among process variables. Based on the ARD model, a scheme of particular XTT tables and links between them are generated. The tables are filled with expert-provided definitions of constraints over the values of attributes; they are in fact the rule preconditions. The code for rules representation is generated and interpreted with provided inference engine. A set of tools supporting the design and development stages is described.

This paper provides an overview of the project, its objectives and tools in Sec. 2. The rule formulation with XTT is shortly described in Sec. 3. Then in Sec. 4 HeKatE design toolchain called HaDEs is introduced. The HeaRT inference engine described in Sec. 5. Then a short comparison to selected existing solutions is given in Sec. 6. Concluding remarks are given in the final section.

2 HeKatE Project Overview

The main principles of the HeKatE project are based on a critical analysis of the state-of-the art of the rule-based systems design (see [5]). They are:

- *Formal Language for Knowledge Representation.* It should have a precise definition of syntax, properties and inference rules. This is crucial for determining its expressive power, and solving formal analysis issues.
- *Internal Knowledge Base Structure.* Rules working within a specific context, are grouped together and form the extended decision tables. These tables are linked together forming a structure which encodes the flow of inference.
- *Systematic Hierarchical Design Procedure.* A complete, well-founded design process that covers the main phases of the system lifecycle, from the initial conceptual design, through the logical formulation, all the way to the physical implementation, is proposed. Verification of the system model w.r.t. critical formal properties, such as determinism and completeness is provided.

In the HeKatE approach the control logic is expressed using forward-chaining decision rules. They form an intelligent rule-based controller or simply a business logic core. The controller logic is decomposed into multiple modules represented by attributive decision tables. The emphasis of the methodology is its possible application to a wide range of intelligent controllers. In this context two main areas have been identified in the project: control systems, in the field of intelligent control, and business rules [6] and in the field of software engineering.

HeKatE introduces a formalized language for rule representation [5]. Instead of simple propositional formulas, the language uses expressions in the so-called

attributive logic [3]. This calculus has stronger expressiveness than the propositional logic, while providing tractable inference procedures for extended decision tables [7]. The current version of the rule language is called XTT^2 [8]. The current version of the logic, adopted for the XTT^2 language, is called ALSV(FD) (*Attributive Logic with Set Values over Finite Domains*).

HeKatE also provides a complete hierarchical *design process* for the creation of the XTT-based rules.

- The main phase of the XTT rule design is called the *logical design*. This phase is supported by a CASE tool called HQed.
- The logical rule design process may be supported by a preceding *conceptual design* phase. In this phase the rule prototypes are built with the use of ARD. The principal idea is to build a graph, modelling functional dependencies between attributes on which the XTT rules are built. The version used in HeKatE is called ARD+ as discussed in [9,10]. The ARD+ design is supported by two visual tools, VARDA and HJed.
- The practical implementation on the XTT rule base is performed in the *physical design* phase. In this stage the visual XTT model is transformed into an algebraic presentation syntax called HMR. A custom inference engine, HeaRT, runs the XTT model.

Let us now shortly describe the main aspects of the XTT rule formalization.

3 Main Aspects of the XTT Rule Language Formalization

The so-called ALSV(FD) attributive logic [3,5] has been introduced with practical applications for rule languages in mind. In fact, the primary aim of the presented language is to extend the notational possibilities and expressive power of the XTT-based tabular rule-based systems [8,5]. Some main concepts of the logic are: attribute, atomic formulae, state representation and rule formulation.

After [3] it is assumed that an *attribute* A_i is a function (or partial function) of the form $A_i\colon O \rightarrow 2^{D_i}$. Here O is a set of objects and D_i is the domain of attribute A_i. As we consider dynamic systems, the values of attributes can change over time (or state of the system). We consider both *simple* attributes of the form $A_i\colon T \rightarrow D_i$ (i.e. taking a single value at any instant of time) and *generalized* ones of the form $A_i\colon T \rightarrow 2^{D_i}$ (i.e. taking a set of values at a time); here T denotes the time domain of discourse.

The *atomic formulae* can have the following four forms: $A_i = d$, $A_i = t$, $A_i \in t$, and $A_i \subseteq t$, where $d \in D$ is an atomic value from the domain D of the attribute and $t \subseteq D$, $t = \{d_1, d_2, \ldots, d_k\}$, is a (finite) set of such values. The *semantics* of $A_i = d$ is straightforward – the attribute takes a single value. The semantics of $A_i = t$ is that the attribute takes *all* the values of t (see [5]).

An important extension in ALSV(FD) over previous versions of the logic [3] consists in allowing for explicit specification of one of the relational symbols $=$,$\neq$,$\in$, $\notin$, $\subseteq$, $\supseteq$, $\sim$ and $\not\sim$ with an argument in the table.

From the logical point of view the *state* is represented by the current values of all attributes specified within the contents of the knowledge-base, as a formula:

$$(A_1 = S_1) \wedge (A_2 = S_2) \wedge \ldots \wedge (A_n = S_n) \tag{1}$$

where A_i are the attributes and S_i are their current values; note that $S_i = d_i$ ($d_i \in D_i$) for simple attributes and $S_i = V_i$, ($V_i \subseteq D_i$) for generalised ones, where D_i is the domain for attribute A_i, $i = 1, 2, \ldots, n$.

Now, consider a set of n attributes $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$. Any XTT rule is assumed to be of the form:

$$(A_1 \propto_1 V_1) \wedge (A_2 \propto_2 V_2) \wedge \ldots (A_n \propto_n V_n) \longrightarrow RHS$$

where $\propto_i$ is one of the admissible relational symbols in ALSV(FD), and *RHS* is the right-hand side of the rule covering conclusions. In practise the conclusions are restricted to assigning new attribute values, thus changing the system state. State changes trigger external callbacks that allow for communication with the environment. The values that are no longer valid are removed from the state.

Based on the ALSV(FD) logic the XTT rule language is provided [4,8,5]. The language is focused not only on providing an extended syntax for single rules, but also allows for an explicit structurization of the rule base. XTT introduces explicit inference control solutions, allowing for a fine grained and more optimized rule inference than in the classic Rete-like [11] solutions. The representation has a compact and transparent visual representation suitable for visual editors.

Knowledge representation with XTT incorporates extended attributive table format. Similar rules are grouped within separated tables, and the whole system is split into such tables linked by arrows representing the control strategy. Consider a set of m rules incorporating the same attributes $A_1, A_2, \ldots, A_n$: the preconditions can be grouped together and form a regular matrix, as in Table 1.

Table 1. A general scheme of an XTT table

Rule	A_1	A_2	$\ldots$	A_n	H
1	$\propto_{11} t_{11}$	$\propto_{12} t_{12}$	$\ldots$	$\propto_{1n} t_{1n}$	h_1
2	$\propto_{21} t_{21}$	$\propto_{22} t_{22}$	$\ldots$	$\propto_{2n} t_{2n}$	h_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
m	$\propto_{m1} t_{m1}$	$\propto_{m2} t_{m2}$	$\ldots$	$\propto_{mn} t_{mn}$	h_m

In Table 1 the symbol $\propto_{ij} \in \{=, \neq, \in, \notin\}$ for simple attributes and $\propto_{ij} \in \{=, \neq, \subseteq, \supseteq, \sim, \not\sim\}$ for the generalized ones. In practical applications, however, the most frequent relations are $=$, $\in$, and $\subseteq$, i.e. the current values of attributes are *restricted* to belong to some specific subsets of the domain.

Efficient inference is assured thanks to firing only rules necessary for achieving the goal. It is achieved by selecting the desired output tables and identifying the

tables necessary to be fired first. The links representing the partial order assure that when passing from a table to another one, the latter can be fired since the former one prepares an appropriate context knowledge. Hence, only rules working in the current context of inference are explored. The partial order between tables allows to avoid examining rules which should be fired later. The details of the complete inference solution for XTT are given in [5].

Let us now move to practical issues concerning both the design and the implementation of XTT-based systems.

4 HaDEs Design Framework

The HeKatE design process is supported by a number of tools. They help with the visual design and the automated implementation of rule-based systems (see `https://ai.ia.agh.edu.pl/wiki/hekate:hades`). The complete framework including the previously discussed methods and tools is depicted in Fig. 1.

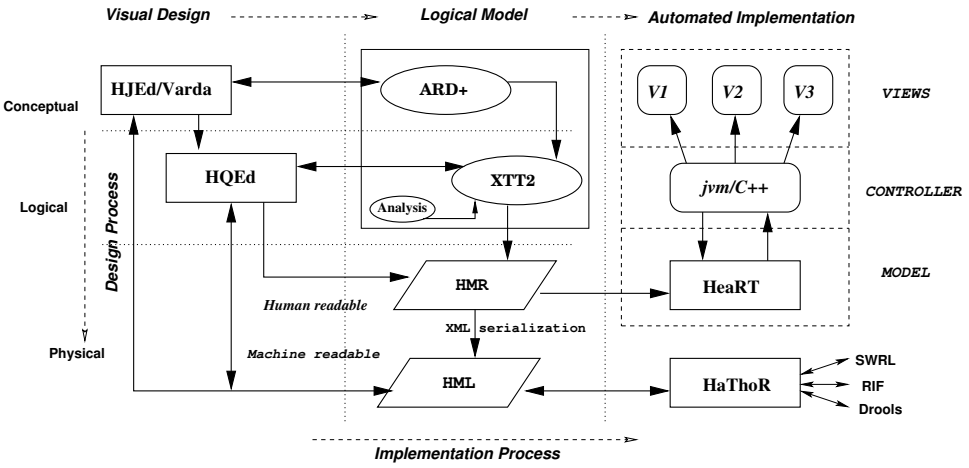

Fig. 1. The complete design and runtime framework

The ARD+ design process is supported by the HJEd visual editor. It is a cross-platform tool implemented in Java. Its main features include the ARD+ diagram creation with on-line design history available through the TPH diagram. An example of a design capturing functional dependencies between system attributes is shown in Fig. 2. Once created, the ARD+ model can be saved in a XML-based HML (HeKatE Markup Language) file. The file can be then imported by the HQEd design tools supporting the logical design.

VARDA is a prototype semivisual editor for the ARD+ diagrams implemented in Prolog, with an on-line model visualization with Graphviz. The tool also supports prototyping of the XTT model, where table headers including a de-

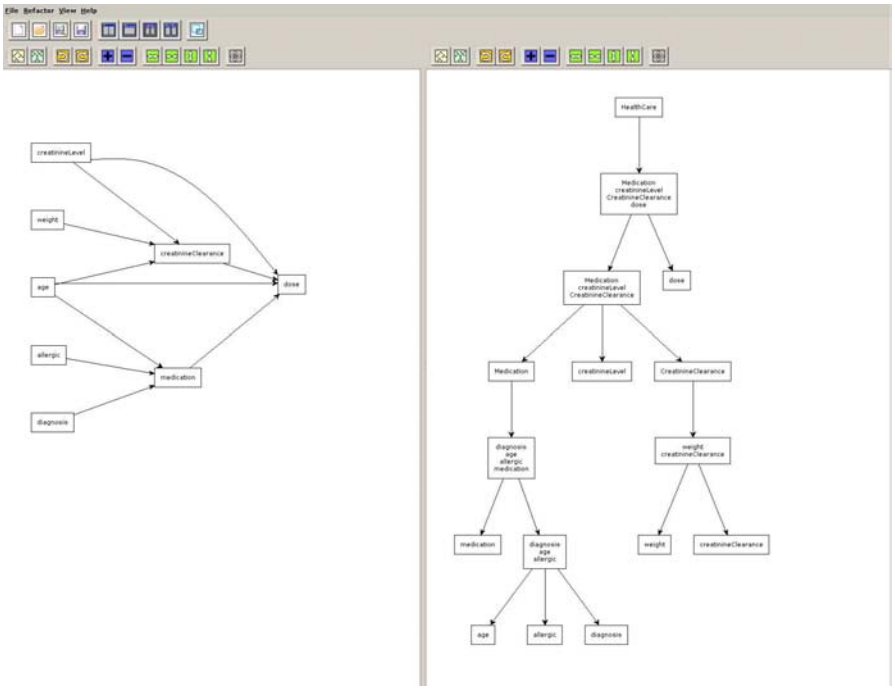

Fig. 2. ARD+ design in HJEd

fault inference structure are created, see Fig. 3. The ARD+ design is described in Prolog, and the resulting model can be stored in HML.

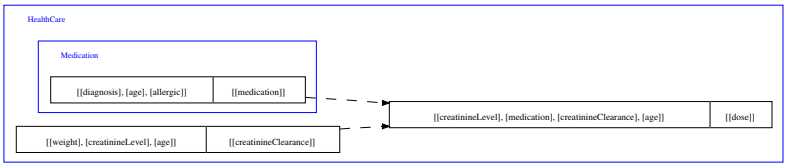

Fig. 3. XTT model generation in VARDA

HQEd provides support for the logical design with XTT, see Fig. 4. It is able to import a HML file with the ARD+ model and generate the XTT prototype. It is also possible to import the prototype generated by VARDA. HQEd allows to edit the XTT structure with on-line support for syntax checking on the table level. Attribute values entered are checked against domains and some possible anomalies are eliminated.

The editor is integrated with a custom inference engine for XTT^2 called HeaRT. The role of the engine is twofold: run the rule logic designed with the

use of the editor, as well as provide on-line formal analysis of the rulebase. The communication uses a custom TCP-based protocol.

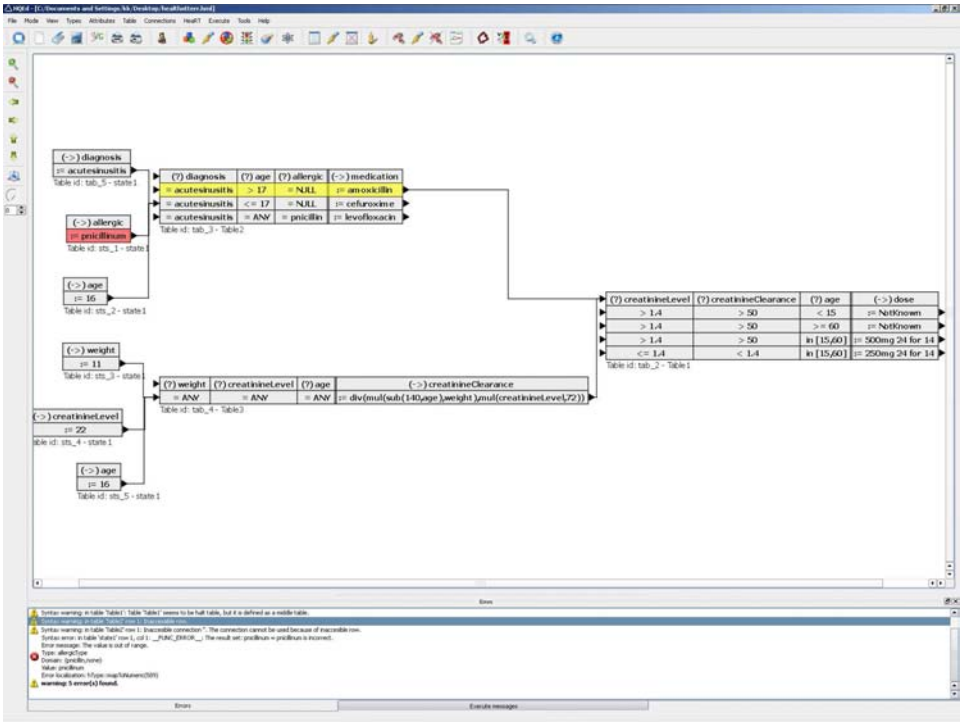

Fig. 4. XTT model edition in HQEd with anomalies detected

HaThoR is the HeKatE rule translation framework. Its goal is to provide rule import and export modules for other languages including RDF and OWL for ARD and RIF and SWRL (possibly R2ML) for XTT. It is mainly implemented in XSLT with some extra plugins integrated with HeaRT implemented in Prolog. An experimental module allows to translate visual XTT representation to a dedicated UML representation using an XMI-based serialization.

5 HeaRT Rule Runtime

HeKatE RunTime (HeaRT) is a dedicated inference engine for the XTT^2 rule bases, (see `https://ai.ia.agh.edu.pl/wiki/hekate:heart`). It is implemented in Prolog in order to directly interpret the HMR representation which is generated by HQEd. HMR (HeKatE Meta Representation) is a textual representation of the XTT^2 logic designed by HQEd. It is a human readable form, as opposed to the machine readable HML format. HeaRT allows to: store and export models in HMR files, and verify HMR syntax and logic. An example excerpt of HMR is:

```
xschm th: [today,hour] ==> [operation].
xrule th/1:
  [today eq workday, hour gt 17] ==> [operation set not_bizhours].
xrule th/4:
  [today eq workday, hour  in [9 to 17]] ==> [operation set bizhours].
```

The first line defines an XTT table scheme, or header, defining all of the attributes used in the table. Its semantics is as follows: "the XTT table *th* has two conditional attributes: *today* and *hour* and one decision attribute: *operation*". Then two examples of rules are given. The second rule can be read as: "Rule with ID *4* in the XTT table called *th*: if value of the attribute *today* equals (=) value *workday* and the value of the attribute *hour* belongs to the range ($\in$) $< 9, 17 >$ then set the value of the attribute *operation* to the value *bizhours*".

The engine implements the inference based on ALSV(FD). It supports four types of inference process, Data and Goal Driven, Fixed Order, and Token Driven [5]. Inference is based on assumption, that the system is deterministic. Conflicts should be handled during design process or detected by verification.

HeaRT also provides a modularized verification framework, also known as HalVA (HeKatE Verification and Analysis). Verification and analysis module implements: simple debugging mechanism that allows tracking system's work, logical verification of models (several plugins are available, including completeness, determinism and redundancy checks), and syntactic analysis of HMR files using a DCG grammar of HMR. The verification plugins can be run from the interpreter or indirectly from HQEd using the communication protocol.

The engine has communication and integration facilities. HeaRT supports Java integration based on callbacks mechanism and Prolog JPL library. It allows for a direct interaction via Prolog console based on callbacks mechanism. HeaRT can operate in two modes, stand-alone and as TCP/IP server, offering TCP/IP integration mechanism with other applications. It is possible to create console or graphical user interface build on Model-View-Controler design pattern.

There are two types of callbacks related to attributes in HMR files. 1) input used to get attribute value from user. This can be done by console or graphical user interface. 2) output used to present an attribute value to user. Callbacks can be use to create GUI with JPL and SWING in Java.

To make HeaRT integration easier, there are three integration libraries, JHeroic, PHeroic or YHeroic. *JHeroic* library was written in Java. Based on JHeroic one can build applets, desktop application or even JSP services. It is also possible to integrate HeaRT with database using ODBC, or Hibernate. *YHeroic* is a library created in Python. It has the same functionality as JHeroic but is easier to use because of Python language nature. *PHeroic* is the same library but created in PHP5. It can be used in a dynamic web page based on PHP.

6 Related Solutions

Here, the focus is on two important solutions: CLIPS and its Java-based incarnation – Jess, as well as Drools, which inherits some of the important CLIPS

features, while providing a number of high-level integration features. Other environments include LPA VisiRule.

XTT provides an expressive, formally defined language to describe rules. The language allows for formally described inference, property analysis, and code generation. Additional callbacks in rule decision provide means to invoke external functions or methods in any language. This feature is superior to those found in both CLIPS/Jess and Drools. On the other hand, the main limitation of the HeKatE approach is the state-base description of the system, where the state is understood as the set of attribute values.

The implicit rule base structure is another feature of XTT. Rules are grouped into decision tables during the design, and the inference control is designed during the conceptual design, and later on refined during the logical design. Therefore, the XTT representation is highly optimized towards rulebase structurization. This feature makes the visual design much more transparent and scalable.

In fact all the Rete-based solutions seek some kind of structurization. In the case of CLIPS it is possible to modularize the rulebase (see chapter 9 in [1]). It is possible to group rules in modules operating in given contexts, and then provide a context switching logic. Drools 5 offers Drools Flow that allows to define rule set and simple control structure determining their execution. In fact this is similar to the XTT-based solution. However, it is a weaker mechanism that does not correspond to table-based solution.

A complete design process seems to be in practice the most important issue. Both CLIPS and Jess are classic expert system shells, providing rule languages, and runtimes. They are not directly connected to any design methodology. The rule language does not have any visual representation, so no complete visual editors are available. Implementation for these systems can be supported by a number of external environments (e.g. Eclipse). However, it is worth emphasizing, that these tools do not visualize the knowledge contained in the rule base.

Drools 5 is decomposed into four main parts: Guvnor, Expert, Flow, Fusion. It offers several support tools, including an Eclipse-based environment. A "design support" feature, is the ability to read Excel files with simple decision tables. While this is a valuable feature, it does not provide constant syntax checking.

It is crucial to emphasize, that there is a fundamental difference between a graphical user interface like the one provided by generic Eclipse-based solutions, and *visual design support and specification* provided by languages such as XTT for rules, and in software engineering by UML. Other dedicated visual rule design languages include URML [12] that provides a UML-based representation for rules. Here focus is on single rules, not on decision tables, like in XTT.

7 Conclusions

The primary area of interest of this paper is to introduce the main concepts of the HeKatE project, its methods and tools. The main motivation behind the project is to speed up and simplify the rule-based systems design process, while assuring the formal quality of the model. The HeKatE design process

is practically supported by a number of tools presented in the paper. These include the HeKatE design environment called HaDEs and the rule runtime called HeaRT. The up-to-date results of the project, as well all the relevant papers are available at the project website see `http://hekate.ia.agh.edu.pl`.

HeKatE project ends in November 2009. Therefore, future work includes a tighter tool integration, as well as modeling complex cases in order to identify possible limitations of the methodology. Providing a comparative studies modelling the same cases in XTT, CLIPS and Drools is planned.

References

1. Giarratano, J.C., Riley, G.D.: Expert Systems. Thomson (2005)
2. van Harmelen, F., Lifschitz, V., Porter, B., eds.: Handbook of Knowledge Representation. Elsevier Science (2007)
3. Ligęza, A.: Logical Foundations for Rule-Based Systems. Springer-Verlag, Berlin, Heidelberg (2006)
4. Nalepa, G.J., Ligęza, A.: A graphical tabular model for rule-based logic programming and verification. Systems Science **31**(2) (2005) 89–95
5. Nalepa, G.J., Ligęza, A.: Hekate methodology, hybrid engineering of intelligent systems. International Journal of Applied Mathematics and Computer Science (2009) accepted for publication.
6. Ross, R.G.: Principles of the Business Rule Approach. 1 edn. Addison-Wesley Professional (2003)
7. Ligęza, A., Nalepa, G.J.: Knowledge representation with granular attributive logic for XTT-based expert systems. In Wilson, D.C., Sutcliffe, G.C.J., FLAIRS, eds.: FLAIRS-20 : Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference : Key West, Florida, May 7-9, 2007, Menlo Park, California, Florida Artificial Intelligence Research Society, AAAI Press (may 2007) 530–535
8. Nalepa, G.J., Ligęza, A.: Xtt+ rule design using the alsv(fd). In Giurca, A., Analyti, A., Wagner, G., eds.: ECAI 2008: 18th European Conference on Artificial Intelligence: 2nd East European Workshop on Rule-based applications, RuleApps2008: Patras, 22 July 2008, Patras, University of Patras (2008) 11–15
9. Nalepa, G.J., Ligęza, A.: Conceptual modelling and automated implementation of rule-based systems. In: Software engineering : evolution and emerging technologies. Volume 130 of Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam (2005) 330–340
10. Nalepa, G.J., Wojnicki, I.: Towards formalization of ARD+ conceptual design and refinement method. In Wilson, D.C., Lane, H.C., eds.: FLAIRS-21: Proceedings of the twenty-first international Florida Artificial Intelligence Research Society conference: 15–17 may 2008, Coconut Grove, Florida, USA, Menlo Park, California, AAAI Press (2008) 353–358
11. Forgy, C.: Rete: A fast algorithm for the many patterns/many objects match problem. Artif. Intell. **19**(1) (1982) 17–37
12. Lukichev, S., Wagner, G.: Visual rules modeling. In: Sixth International Andrei Ershov Memorial Conference Perspectives of System Informatics, Novosibirsk, Russia, June 2006. LNCS, Springer (2005)