Int. J. Appl. Math. Comput. Sci., 2010, Vol. 20, No. 1, 35–53
DOI: 10.2478/v10006-010-0003-9

THE HEKATE METHODOLOGY. HYBRID ENGINEERING OF INTELLIGENT SYSTEMS

GRZEGORZ J. NALEPA, ANTONI LIGĘZA

Department of Automatics
AGH University of Science and Technology, Al. Mickiewicza 30, 30–059, Kraków, Poland
e-mail: {gjn,ligeza}@agh.edu.pl

This paper describes a new approach, the HeKatE methodology, to the design and development of complex rule-based systems for control and decision support. The main paradigm for rule representation, namely, eXtended Tabular Trees (XTT), 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 on single rules. Such groups form decision tables which are connected into a network for inference. Efficient inference is assured as only the rules necessary for achieving the goal, identified by the context of inference and partial order among tables, are fired. In the paper a new version of the language—XTT^2—is presented. It is based on ALSV(FD) logic, also described in the paper. Another distinctive feature of the presented approach is a top-down design methodology based on successive refinement of the project. It starts with Attribute Relationship Diagram (ARD) development. Such a diagram represents relationships between system variables. Based on the ARD scheme, XTT tables and links between them are generated. The tables are filled with expert-provided constraints on values of the attributes. The code for rule representation is generated in a human-readable representation called HMR and interpreted with a provided inference engine called HeaRT. A set of software tools supporting the visual design and development stages is described in brief.

Keywords: rule-based expert systems, intelligent control, system design.

1. Introduction

Rule-based systems constitute one of the most powerful and most popular knowledge representation formalisms (Liebowitz, 1998; van Harmelen *et al.*, 2007). They offer a relatively easy way of knowledge encoding and interpretation. The formalization of knowledge within a rule-based system can be based on mathematical logic (e.g., propositional, attributive, first-order, or even higher order ones) or performed on the basis of engineering intuition. Rule-based systems have found numerous applications in various domains of engineering, science, medicine, law, computer science and application software (Laffey *et al.*, 1988; Liebowitz, 1998).

The never-ending story of rule-based systems started with the very first attempts to formally codify rules of human behavior. Perhaps one of the best known sets of such rules, the *Ten Commandments*, has been with us since the times of Moses (see (Ligęza, 2006) for a more detailed historical review).

Rules are omnipresent in our everyday life, professional work, leisure and sport activities. They are a result of physical (and mathematical) laws, formed by men, tradition, culture, and civilization. The most precise rules are those found in technical and technological systems; however, many rule-based expert systems addressed the issues of medical diagnosis or business decision support as well.

In engineering, the first examples of rule-based systems are those concerning feedback control with a two-position or three-position relay; in fact, such systems can be considered simple, hard-wired rule-based systems. Further examples come from the domain of digital circuits. Actually, any combinatorial circuit can be regarded as a propositional logic rule-based system.

In computer science, rule-based systems appeared just after symbolic programming languages had been developed. First such systems were termed *production systems* or *production rules* (Brownston *et al.*, 1985; Liebowitz, 1998). The golden age for rules came in the late 1970s and 1980s with developments and practical applications of expert systems. They were dedicated to solving specific problems in narrow, well-formalized domains. A typical construction of such a system was based on two components: a *declarative rule base* encoding domain-specific knowledge and an *inference engine* of general purpose, the so-called *expert system shell* (Liebowitz, 1998; Giarratano and Riley, 2005).

Modern shells for the development of rule-based systems, such as CLIPS, Jess, Drools, Aion, Ilog Rules, or Gensym's G2, follow this classical paradigm. Rules are developed using some predefined knowledge representation framework (often close to attributive logic). The current rule-based systems and tools for their development have reached a certain level of maturity. Specialized editors that enforce a correct structure and are capable of checking the syntax of rules are in use and provide tools for computer-aided development of final applications.On the other hand, they have inherited a number of traditional features of early rule-based systems, which nowadays can be considered drawbacks.

In this paper we mainly address the following issues, which seem worth investigating and improve:

- **sparse representation:** single rules constitute items of low knowledge processing capabilities, while for practical applications a higher level of abstraction is desirable,
- **blind inference:** inference engines, especially forward-chaining ones, are highly inefficient with respect to the focus on the goal to be achieved,
- **lack of methodology:** no practical methodology for consecutive top-down design and development of rule-based systems, acceptable by engineers and ensuring quality of rules, is available.

This paper presents a new approach to the design and development of rule-based systems. More precisely, we present the state-of-the-art of the HeKatE methodology[1], for the design and development of complex rule-based systems for control and decision support. This methodology is supported with visual tools for the development of knowledge bases and a novel inference engine.

The main paradigm for rule representation, namely, eXtended Tabular Trees (XTT) (Nalepa, 2004; Nalepa and Ligęza, 2005a), 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. In this way we address the first issue of low processing capabilities of single rules. Such groups form decision tables which are connected into a network for inference.

Efficient inference is assured as only the rules necessary for achieving the goal are fired. 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 the appropriate context knowledge. Hence, only the rules working in the current context of inference are explored. The partial order between tables allows avoiding examining rules which should be fired later.

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

The rest of the paper is organized as follows: In Section 2 the perspective on the applications of rule-based systems for intelligent control is given. For intuition, a simple example concerning the relay-type controller is used to identify some of the most important issues tackled in this paper. This gives a background for the motivation for the research, as discussed in Section 3, and for the HeKatE project, which aims at providing solutions for the problems of sparse representation, blind inference, and the lack of methodology identified earlier in this section. One of the main goals of the project is to provide a new rule-based inference engine solution assuring flexible and efficient control during the inference process. In Section 4, a dynamic system state representation discussion is given in order to be able to develop a formalization for rules. Rules in HeKatE are formalized with the use of *attributive logic*, introduced in Section 5 and then discussed in an extended version in Section 6. Inference rules for the attributive logic formulae are presented in Section 7. They allow firing XTT rules grouped into tables. The table-level inference is discussed in Section 8. Practical design of XTT knowledge bases, supported by visual editors, and other tools is shortly presented in Section 9. A concise comparison with the related solutions is given in Section 10. Finally, the future challenges for the methodology are given in Section 11.

2. Rule-based intelligent control

This section is addressed to the *automatic control* audience. It briefly shows how rules can be used to provide declarative means for building controllers in a wide sense of the word *control*. In fact, the past, current, and prospective applications are allocated in numerous, diversified areas of applications, ranging from direct (digital) con-

[1]An acronym for Hybrid Knowledge Engineering, see `http://hekate.ia.agh.edu.pl`.

trol, meta-level control, decision support, business rules applications and various forms of expert systems (Laffey *et al.*, 1988; Liebowitz, 1998).

2.1. Explaining the ideas of rules in control. Consider a simple case of a direct use of rules for control. Rules can be used to specify a rely-type controller of many levels (not just two-level or three-level classical control rely). In fact, arbitrary approximation of any nonlinear characteristic of the controller can easily be specified.

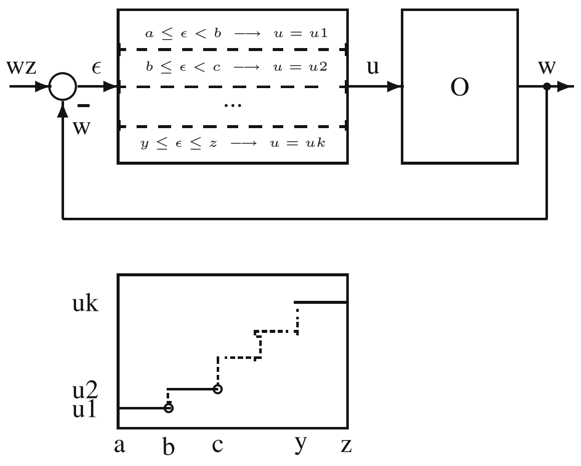

$a \leq \epsilon < b \longrightarrow u = u1$
$b \leq \epsilon < c \longrightarrow u = u2$
...
$y \leq \epsilon \leq z \longrightarrow u = uk$
wz
ϵ
-
w
u
O
w
uk
u2
u1
a
b
c
y
z

Fig. 1. Simple application of a rule-based system as a direct controller.

This type of application is illustrated in Fig. 1. In the picture, u is the input of the object under control and w is its output signal. The current value of the control error, ϵ, is calculated as $\epsilon = wz - w$, where wz is the required set point. Depending on the value of ϵ, the rule-based controller determines the level of the u signal, according to the characteristics shown in Fig. 1.

Obviously, applying a rule-based system as a direct controller is not the only way to make use of the rule-based systems technology. In fact, a rule-based approach can be applied to decision making, optimization (selection of optimal settings or trajectories), adaptation (adjusting the algorithms or parameters to new conditions), and even detetecting structural changes of the control system. These ideas are illustrated in Fig. 2.

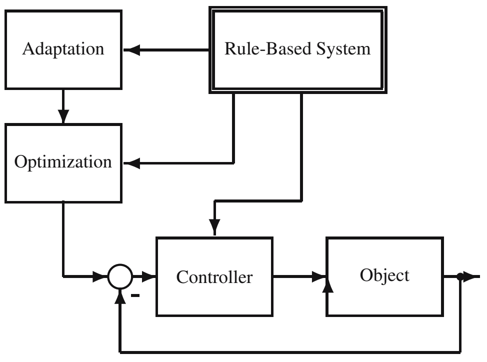


Fig. 2. Application of rule-based systems at various levels of multi-level control: direct control, optimization, and adaptation.

2.2. Rule-based control. An important step in application of rule-based systems in control was the design of the Rete algorithm (Forgy, 1982). The algorithm is based on two observations: (i) at each cycle, only few facts of the fact base change (this is referred to as *temporal redundancy*), and (ii) preconditions of rules often contain similar patterns or groups of such patterns (this is referred to as *structural similarity*) (Giarratano and Riley, 2005). Hence, instead of repeated checking of preconditions of *all* the rules at each cycle, the Rete algorithm employs a specialized network for fast indexing of changes in the fact-base and their influence on the satisfaction of the rule preconditions. Thanks to the application of Rete, the sets of rules which can be fired can be determined in an efficient way. Such sets are called *conflict sets*, since only one of the rules can be fired.

The Rete-type identification of rules that can be fired is memory-consuming, since the Rete network can be quite large. On the other hand, if efficiency is of primary importance, the application of the Rete forward-chaining inference engine seems reasonable (especially in real-time control applications). Nevertheless, such forward checking procedure does not identify which rules should be fired so as to achieve the goal. As many unnecessary rules can be fired, we shall refer to the inference scheme as a *blind* one.

The next important step was the development of the CLIPS (Giarratano and Riley, 2005) (C Language Integrated Production System), a descendant of OPS5 (Brownston *et al.*, 1985), a rule-based production system written in Lisp. The CLIPS was developed by NASA and—despite being developed in C—it follows the Lisp-like style of knowledge specification (a bit clumsy, full of parentheses and hardly readable for engineers). It has become perhaps one of the most popular rule-based engines, since it is relatively fast (employing the Rete algorithm (Forgy, 1982)), simple, and free (now in the public domain). A more recent reincarnation of the CLIPS is JESS (Java Expert System Shell) (Friedman-Hill, 2003), developed in Java, but still employing the (a bit ancient)

Lisp-like style of rule encoding. Another example is Drools (Browne, 2009), also referred to as a *business rules management system*. All the systems are in principle blind, forward-checking ones, employing advanced implementations of the Rete-based inference engines.

Some overview of current developments with respect to theory, knowledge representation, tools and application areas is provided in (Liebowitz, 1998). A list of current tools is enclosed with (Ligęza, 2006). The very recent book (Giurca *et al.*, 2009) gives a good overview of some emerging technologies and tools in the area of rule-based solutions.

2.3. Rule-based control and decision support. Historical perspective of research towards the XTT approach. The authors' research in the domain of rule-based systems for control started in 1986 (Ligęza, 1986). The aim was to develop a specialized form of rules, taking into account the specific requirements of the control domain. This resulted in the development of an universal rule scheme incorporating dynamic modification of the knowledge base (through the *retract* and *assert* operations)[2] and elements of control (the concept of declaring the *next* and *else* rules). The rules were designed to work only within a certain *context*, *status* and *mode* which focused the inference on the subset of rules necessary for the current stage of inference (Tzafestas and Ligęza, 1988; 1989).

On the other hand, the application of rules for decision support in business processes was addressed in (Ligęza, 1988). It was pointed out that there are different kinds of rules for different purposes (pure deduction, dealing with numerical values, domain-independent properties, meta rules for control).

In order to ensure the reliability, safety and quality of rule-based systems, additional work on the verification of theoretical properties was carried out (Ligęza, 1993). A new inference rule, the so-called *backward dual resolution* (or *dual resolution*, for short) was invented for logical verification of completeness of rule-based systems. A taxonomy of abnormalities and approaches to deal with them was discussed in (Ligęza, 1999).

A comprehensive report on trends and directions of research was presented in (Coenen *et al.*, 2000). Some break-through ideas were introduced in (Ligęza, 1996); namely, a proposal to incorporate verification in the design stage, so that it is performed on-line and not only after the system is developed. Another idea was to group similar rules together (to form tabular components covering several rules instead of one) and to perform verification of such tables consisting of similar rules, rather than a flat set of all the rules (Ligęza, 1998).

[2]These names follow the Prolog (Bratko, 2000) language notation and correspond to removing and adding facts to the fact base.

An initial concept to organize a tabular system into a hierarchical structure was first proposed in (Ligęza, 2001) and developed as a practical test-bed implementation under the name of Tab-Trees (Ligęza *et al.*, 2001). The first *conceptual statement*, basic formalism and initial working implementation of XTT were developed as a Ph.D. project and presented in (Nalepa, 2004). They were later described in (Nalepa and Ligęza, 2005a) on the conceptual level. The Mirella tools supporting visual XTT design were introduced in (Nalepa and Ligęza, 2005c). To support logical XTT design, the ARD method was introduced later on in (Nalepa and Ligęza, 2005b). A survey in the form of a textbook presenting logical foundations and developments in the area of attribute logic, its application in XTT and elements of the design and verification methodology were the objectives of (Ligęza, 2005; 2006). A Prolog implementation for XTT was discussed in (Nalepa and Ligęza, 2006). The aforementioned developments and experience gained during the research form the foundations that led to the HeKatE methodology, which is the focus of this paper.

3. HeKatE approach

3.1. Motivation. There are certain persistent limitations of the existing approaches to the design of rule-based intelligent control systems. They are especially visible in the design of complex systems. They often make high quality design as well as the refinement of such systems very difficult. These limitations are related to the following aspects of rule design:

- *knowledge representation*—where rules are informally described and lack clear semantics,
- *transparent structure and efficient inference in the rulebase*—where the focus on the design of single rules, with no structure explicitly identified by the user, makes the hierarchization of the model very difficult; moreover, the classic massive inference mechanisms such as the Rete algorithm do not address the contextual nature of the rulebase,
- *well-founded systematic and complete design process*—preserves the quality aspects of the rule model and allows gradual system design and automated implementation of rules.

These issues are discussed in greater detail in the following subsections.

The majority of rule-based systems work according to the classical principles of *forward chaining*. They incorporate a relatively simple, blind inference engine. In order to speed up the interpretation of rules, they often employ some indexing of changes that occur in the fact base, and the way they influence the satisfaction of rule preconditions, e.g., the Rete network.

With respect to the knowledge representation language being used, the following issues may be raised:

- lack of formal relation of the knowledge representation language to classical logic, consequently,
- difficulties with understanding the expressive power of the language,
- lack of standards for knowledge representation, and, as a consequence,
- lack of knowledge portability.

With respect to the internal structure and inference mechanism, the criticism may be even stronger:

- typically, the set of rules is flat—it has no internal structure, so hundreds of different rules are considered equally important, and equally unrelated,
- the inference engine (at least potentially) tries to examine all rules in turn for firing within every cycle,
- it is unclear if the Rete network can be efficiently used in the case of knowledge representation formalism of higher expressive power[3],
- there is no definite way to decide which rules from the conflict set should be fired,
- irrelevant rules can be fired even if they do not contribute to solving the problem.

With respect to the development methodology, it should be pointed out that most of the rule-based tools are just shells providing a rule interpreter and sometimes an editor. Hence

- the knowledge acquisition task constitutes a bottleneck and is arguably the weakest point in the design and development process,
- typical rule shells are not equipped with consistent methodology and tools for efficient development of the rule base; on the other hand, general methodologies such as KADS or Common-KADS (van Harmelen, 1996) are too complex for practical engineering or business applications,
- the verification of knowledge is rarely supported, and, if it is supported,
- verification is performed only after the rule design phase (so knowledge refinement can introduce new errors).

[3]For example, in the case of first order logic one cannot avoid using a complete unification algorithm, which itself is based on a tree matching procedure. The authors are unaware of any research on the application of Rete for speeding up term unification.

3.2. Main principles. Three main principles following from the above analysis define the foundations of the approach advocated in this paper:

1. **Formal language definition**—we insist on precise definition of a formal language, its formal properties and inference rules. This is crucial for determining expressive power, defining the inference mechanism and solving verification issues.
2. **Internal knowledge structure**—we introduce an internal structure of the rule base. Similar rules, aimed at working within a specific context, are grouped together and form the XTT tables. These tables are linked together forming a partially ordered graph structure (possibly with cycles), which encodes the flow of inference.
3. **Systematic design procedure**—we argue for a complete, well-founded design process that covers all of the main phases of the system life-cycle, from the initial conceptual design, through the logical formulation, all the way to the physical implementation. We emphasize the need for a constant on-line verification of the system model with respect to critical formal properties, such as determinism and completeness.

These principles served as guidelines for the HeKatE approach we introduce in the next section.

3.3. Goals and Perspectives. The goals of the *Hybrid Knowledge Engineering* (HeKatE) methodology correspond to the principles described previously in Section 3.2. The rule-based knowledge is decomposed into multiple modules represented by attributive decision tables (the so-called *object-attribute-value* tables, see (Ligęza, 1996; Ligęza *et al.*, 2001)). The controller is designed using the HeKatE methodology. Therefore the main goals are to provide the following: an expressive formal logical calculus for rules, allowing formalized inference and analysis; a structured visual rule representation method with formally described syntax and semantics, based on decision tables; and a complete hierarchical design process based on the above, with an effective on-line verification methods as well as, automated implementation facilities.

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 (Laffey *et al.*, 1988; Cheng, 2002),
- business rules (von Halle, 2001; Ross, 2003) and business intelligence systems (Morgan, 2002).

Taking into account the first domain of application, the meaning of the term “controller” is straightforward and falls into the area discussed in this paper as *intelligent control*, a branch of automatic control.

In the case of the second domain, the term denotes a well-isolated software component implementing the so-called application logic or logical model. In business software this component is commonly integrated using a dedicated architectural pattern, such as the *Model-View-Controller* (Burbeck, 1992; Gamma *et al.*, 1995). In this pattern the components of the system are divided intro three main groups:

- *Model*—provides the main control logic,
- *View*—roughly corresponds to system interfaces, in a broad sense, and the
- *Controller*—which connects these two in a flexible way (in this case, the term “controller” is used in a purely technical meaning).

In fact, in real life applications, a number of *Views* (possibly with different *Controllers*) is provided for the same logical *Model*, to allow reusing of the same control logic in a number of applications.

Let us now present the first perspective using an example.

3.4. Intuitive example. In order to explain the underlying ideas of the presented approach, let us refer to the classical scheme of feedback control, as widely known in the domain of control theory. It is assumed that we are given a certain (dynamic) *system under control* and another system—in our case the *XTT System*, playing the role of an intelligent controller. A general scheme is presented in Fig. 3.

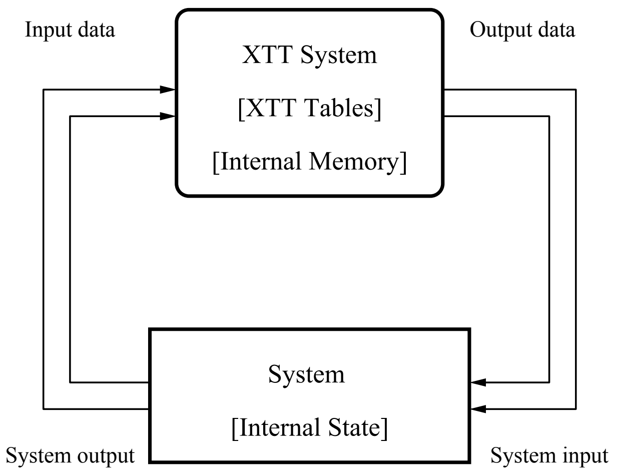


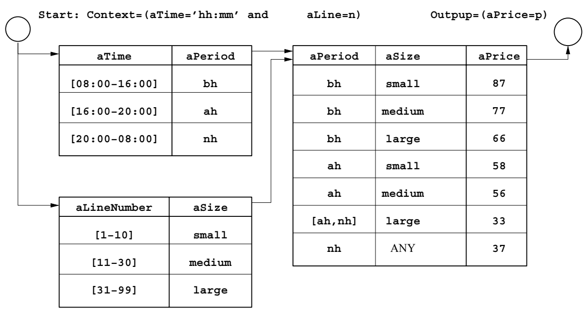
Fig. 3. Scheme of closed-loop feedback control.

The system under control has some input and some observable output. Both the input and the output can be encoded in some symbolic form (meaningful numbers, symbols, facts, logical formulae, etc.). Hence, both input and output streams can be of numerical and symbolic form. The system possesses memory, therefore one can speak about some *internal state* describing the cumulated results of the former input and allowing to model its future behavior.

The XTT system receives the output from the system under control as its input data. With the use of internal knowledge (a kind of rule base), the XTT system works out some output data (decisions, control) and sends them as the input to the system under control.

The system under control can be, in general, almost any complex system providing observable (measurable) and meaningful output. It can be a complex technological installation, computer hardware or software, a database, a knowledge-base, or a natural or artificial system with manual formation of the XTT input. The task performed by the XTT system can range from direct, closed-loop control, to monitoring, decision making or knowledge processing.

The main focus of this paper is on the XTT system, its components (tables of rules), internal structure, knowledge representation language and inference, as well as inference control and design issues. The XTT system is composed of a number of attributive decision tables, extended in comparison to simple decision tables known in the literature (Pawlak, 1991). For some intuition of how such a system is built and how it works, see Fig. 4.



aTime	aPeriod
[08:00-16:00]	bh
[16:00-20:00]	ah
[20:00-08:00]	nh

aLineNumber	aSize
[1-10]	small
[11-30]	medium
[31-99]	large

aPeriod	aSize	aPrice
bh	small	87
bh	medium	77
bh	large	66
ah	small	58
ah	medium	56
[ah,nh]	large	33
nh	ANY	37

Fig. 4. Simple intuitive example of an XTT system.

The system from Fig. 4 is devoted to inferring the minute rate for telephone calls, depending on the time of day and the size of the client company. There are three decision tables. The first one allows determining the period of day; the output attribute `aPeriod` takes one of three possible values: business hours (`bh`), afternoon hours (`ah`), or night hours (`nh`). The input attribute here is the time (`aTime`), e.g., as read from the system clock. The second decision table defines the qualitative size of the company and takes the following values: `small` (between 1 and 10 lines), `medium` (between 11 and 30 lines) and `large` (between 31 and 99 lines). After having determined the period and the size, the values of the attributes `aPeriod` and `aSize` are used to infer the prices; various combinations of period and size and the corresponding prices are defined in the third table.

The system can start its work given the values of the `aTime` and `aLineNumber` attributes as its input; in other words, the values of these attributes—when defined—determine the *context* of the work of the XTT system. The appropriate rules are activated when control is passed to a specific table. Contrary to classical solutions, rules are fired when necessary, not when just the preconditions are satisfied. The links among tables define a partial order of inference. Within a table, a single rule or a set of rules can be fired. As illustrated with this simple example, some tables can be interpreted *in parallel* (in our example—the first two tables), while the third one must be interpreted *after* the output of both of them is produced.

3.5. Review of the methods. In the HeKatE project, a formalized language for an extended rule representation is introduced. Instead of simple propositional formulae, the language uses expressions in the so-called *attributive logic* (Ligęza, 2006). This calculus has higher expressive power than propositional logic, while providing tractable inference procedures for extended decision tables (Ligęza and Nalepa, 2007; 2008). The current version of the rule language is called XTT^2 (Nalepa and Ligęza, 2008). The logical formalism, adopted for the XTT^2 language, is called ALSV(FD) (*Attributive Logic with Set Values over Finite Domains*). The details of this solution are discussed in extent in Sections 5 and 6.

Based on attributive logic, the XTT rule language is provided (Nalepa, 2004; Nalepa and Ligęza, 2005a; 2008). XTT stands for *eXtended Tabular Trees*, since the language is focused not only on providing an extended syntax for single rules, but also allows explicit structuring of the rule base. This solution allows the identification of the system contexts during rule base design. XTT introduces explicit inference control solutions, allowing fine-grained and more optimized rule inference than in the classic Rete-like solutions. XTT has been introduced with visual design support in mind. The representation is compact and transparent, thus suitable for visual editors.

The HeKatE project also provides a complete hierarchical design process for the creation of XTT-based rules. The process is based on the ideas originally introduced in (Nalepa, 2004). The main phase of the XTT rule design is called *logical design*. This phase is supported by a CASE tool called HQed (Kaczor and Nalepa, 2008; Kaczor, 2008).

The logical rule design process may be supported by a preceding *conceptual design* phase. In this phase the rule prototypes are built using the so-called *attribute relationship diagrams*. The ARD method was introduced in (Nalepa and Ligęza, 2005b) and later refined in (Ligęza, 2006).The principal idea is to build a graph modeling functional dependencies between attributes on which the XTT rules are built. The version used in HeKatE is called ARD+. ARD+ design is supported by two visual tools—VARDA and HJed.

Practical implementation of the XTT rule base is performed in the physical design phase. In this stage, the visual model built with HQed is transformed into an algebraic presentation syntax. A custom inference engine can then run the XTT model. All of these design tools are described in greater detail in Section 9.

4. State representation

When processing information, the current values of attributes form the *state of the inference process*. The values of attributes can, in general, be modified in the following three ways:

- by an independent external system,
- by the inference process itself, or
- as some time-dependent functions.

The first case concerns attributes representing some process variables, which are to be taken into account in the inference process but depend only on the environment and external systems. As such, the variables cannot be directly influenced by the XTT system. Examples of such variables may be the external temperature, the age of a client or the set of foreign languages known by a candidate. Values of those variables are obtained as a result of some *measurement* or *observation* process, and are assumed to be put into the inference system via a *blackboard* communication method; in fact, they are written directly into the internal memory whenever their values are obtained or changed.

A special group of such externally provided values of attributes are the so-called *askable* attributes. A question is posed to the user and an answer (from a limited set of possibilities) is provided. The extension of this idea consists in gathering such values in an external database accessed by the rule-based system.

The second case concerns the values of attributes obtained at a certain stage of reasoning as a result of the operations performed in the decision part (the so-called right hand side—RHS) of XTT. The new attribute values can be

- asserted to global memory (hence stored and made available for other components of the system), or
- kept as values of internal process variables.

The first solution is offered mostly for permanent changes of global values. Typically, an appropriate retract operation is to be performed before asserting new values, so as to keep a consistent state. In this way, the history (trajectory) of the system can be stored as well, provided that each value of an attribute is stored with a temporal index. The second (potential) solution may offer another method for value passing and calculations which do not require permanent storage. For example, if a calculated value is to be passed to another XTT component and it is no longer used afterwards, it is not necessary to store it in the global memory. The latter possibility is not implemented in the current version of the XTT system.

4.1. Requirements for state representation. The current state of the system is considered a complete set of values of all the attributes in use at a certain time instant. The concept of the state is similar to the one in dynamic systems and state-machines. State representation should satisfy important requirements. It should be (i) *internally consistent*, (ii) *externally consistent*, (iii) *complete*, (iv) *deterministic*, and (v) *concise*.

The first postulate says that the specification itself cannot be inconsistent at the syntactic level. For example, a *simple attribute* (one taking a single value) cannot take two different values at the same time. In general, assuming the independence of the attributes and no use of explicit negation, each value of an attribute should be specified exactly once.

A *generalized attribute* is one taking a set value (see Section 6). In the case of such attributes, the values are represented by sets (e.g., with the use of lists) and undergo the same restrictions.

The second postulate says that only *true* knowledge (with respect to the external system) can be specified in the state. In other words, facts that are syntactically correct but false cannot occur in the state formula.

The third postulate says that *all* the knowledge that is true at a certain time instant should be represented within the state. This also means that the size (with respect to the number of attributes) of state representation can vary over time, since at certain time instants the values of certain attributes can remain undefined.

The fourth postulate says that there can be no disjunctive or conditional knowledge specification.

Finally, the fifth postulate says that no unnecessary, dependent knowledge should be kept in the state. In databases (Connolly *et al.*, 1999) and most knowledge bases, this has a practical dimension: only true facts are represented explicitly. On the other hand, note that we insist that *all* the known values of attributes be present, i.e., the specification be complete. If there are some functional dependencies among attributes, the only way to represent them is to encode them with the XTT components, and once the dependent value is deduced, it is placed in the state. This also means that if some of the values are changed, we can have a temporarily inconsistent state (a transitional situation), until the new values of the dependent attributes are inferred and replace the old ones.

4.2. State specification. The current values of all attributes are specified within the contents of the knowledgebase (including current sensor readings, measurements, inputs examination, etc.). From the logical point of view, the state is represented as a logical formula:

$$(A_1 = S_1) \wedge (A_2 = S_2) \wedge \ldots \wedge (A_n = S_n), \quad (1)$$

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 complex ones, where D_i is the domain for attribute A_i, $i = 1, 2, \ldots, n$. (See Section 6 for the discussion of attribute types.)

In order to cover realistic cases, an explicit notation for covering unspecified, unknown values is proposed, e.g., to deal with data containing NULL values imported from a database. Consider the case when an attribute may be applied to an object, but it takes no value. This will be denoted as $A = \emptyset$. For example, the formula `Phone_Number=`$\emptyset$ means that the person considered has no phone number. Finally, a formula of the form $A =$ `NULL` means that attribute A takes an unspecified value.

States can be identified by a key, described with some comments (key words, characteristics) and indexed with a time instant (or interval). Such states can be stored to enable access to past/historical values, the monitoring of the trajectory of the system and changes in the fact base. Recorded states are analogous to save points in databases and can be used to repeat inference and retrieve a current state in the case of a system failure. More on that can be found in Section 8.

In the next sections, logic allowing for state representation and inference is introduced.

5. Attributive logic

5.1. Basics of attributive languages. Using logics based on attributes is one of the most popular approaches to defining knowledge. Not only is it very intuitive, but it follows a simple technical way of discussion, where the behavior of physical systems is formalized by providing the values of system variables. This kind of logic is omnipresent in various applications. It constitutes the basis for the construction of relational database tables (Connolly *et al.*, 1999), attributive decision tables and trees (Klösgen and Żytkow, 2002; Pawlak, 1991; Quinlan, 1987), attributive rule-based systems (Ligęza, 2006), and it is often applied to describe the state of dynamic systems and autonomous agents. Some most typical examples include expert and decision support, rule-based control and monitoring systems, as well as diagnostic systems.

However, it is symptomatic that while a number of modern rule-based shells, such as Jess or Drools, provide new high-level features in the area of current software technologies, such as Java-integration, network services, etc., rule representation and inference methods do not evolve. The rule languages found in these tools tend to be logically trivial and conceptually simple. They mostly reuse very basic logic solutions and combine them with new programming language features, mainly borrowed from Java, building on top of classic inference approaches, such as blind, forward-checking inference engines employing the Rete-style algorithm (Forgy, 1982).

While these systems integrate well with today's business application stacks, they provide little or no improvement in the areas of formalized analysis, visual design, gradual refinement or inference control. This gives motivation to approach these problems by introducing novel knowledge representation and design tools.

It is symptomatic that although propositional and predicate logic (in the form of first-order predicate calculus) have well-elaborated syntax and semantics, presented in detail in numerous books covering logic for AI and knowledge engineering (Genesereth and Nilsson, 1987; Jackson, 1999; Torsun, 1995), logic for computer science or artificial intelligence (Ben-Ari, 2001; Liebowitz, 1998), the discussion of the syntax and semantics of attribute-based logic is omitted in such references.[4]

On the contrary, it is apparently often assumed that attributive logic is some kind of *technical language* equivalent—with respect to its *expressive power*—to propositional calculus, and as such it is not worth any more detailed discussion. Actually, it seems that some of the real reasons for the omission of presentation is that a more detailed discussion might be not so straightforward, concise and elegant as in the case of classical logic.

In fact, as follows from some first attempts presented in (Ligęza, 2006), this issue requires a more detailed study. The most typical way of thinking about attributive logic for knowledge specification may be put as follows:

- first, one has to define *facts*, typically of the form
$$A = d$$
or
$$A(o) = d,$$
where A is a certain attribute, o is the object of interest and d is the attribute value;
- second, facts are perceived as propositional logic atomic formulae;
- third, the syntax and semantics of propositional calculus are freely used.

[4]Note that even in the four-volume handbook of *Logics for Artificial Intelligence* edited by D. Gabbay *et al.* attribute logic did not earn a few pages of formal presentation and analysis of properties.

This basic approach is sometimes extended with the use of certain syntax modifications. For example, in (Klösgen and Żytkow, 2002) the discussion is extended, so that the rules take the form

$$A_1 \in V_1 \wedge A_2 \in V_2 \wedge \ldots \wedge A_n \in V_n \longrightarrow A_{n+1} = d.$$

Following this line of extended knowledge specification, various relational symbols can be introduced, e.g., $A_i > d$ (for ordered sets; this can be considered a shorthand for $A_i \in V_i \setminus V_d$, where V_d is the set of all the values of A_i less than or equal to d) or $A_i \neq d_i$ (this can be considered a shorthand for $A_i \in V_i \setminus \{d_i\}$).

Note, however, that extending the syntax in such a way preserves the limitation that an attribute can only take a single value at a time. Furthermore, without providing clearly defined semantics for the language and some formal inference rules, it may lead to severe problems. This follows from the fact that atoms do not appear to be *logically independent* any more (which is the basic, although often implicit assumption of propositional logic (Ligęza, 2006)). For example, having a rule such as

$$\mathit{Temperature} > 100 \longrightarrow \mathit{WaterState} = \mathit{boiling}$$

and a fact like $\mathit{Temperature} > 123$, we would not be able to fire the rule using classical inference rules.[5]

6. Set attributive logic development

In the recent book (Ligęza, 2006), the discussion of attributive logic is much more thorough. The added value consists in allowing attributes take *set values* and providing some formal framework of *Set Attributive Logic* (SAL) with respect to its syntax, semantics and selected inference rules. The very basic idea for further discussion is that attributes should be able to take not only *atomic* values, but *set* values as well.

After (Ligęza, 2006), it is assumed that an *attribute* A_i is a function (or partial function) of the form $A_i\colon O \rightarrow D_i$. 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 (as the state of the system changes). 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 of SAL can have the following four forms:

$$A(o) = d, \tag{2}$$

[5]Some expert system shells, such as PC-SHELL (see `http://aitech.pl/`), are capable of performing the so-called *intelligent unification* and hence succeed to carry on with this kind of inference. This has, however, nothing to do with logic. It is just a hard-wired implementation of a specialized match mechanism which works only for predefined symbols.

$$A(o) = t, \tag{3}$$

$$A(o) \in t \tag{4}$$

and

$$A(o) \subseteq t, \tag{5}$$

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. If the object o is known (or unimportant), its specification can be skipped. Hence we write $A_i = d$, $A_i = t$, $A_i \in t$ or $A_i \subseteq t$, for simplicity.

The semantics of $A_i = d$ are straightforward—the attribute takes a single value. The semantics of $A_i = t$ are that the attribute takes *all* the values of t while the semantics of $A_i \in t$ are that it takes exactly *one* value from t and, finally, $A_i \subseteq t$ means that the attribute takes *some* of the values of t (the so-called *internal disjunction*).[6]

In the case of (3) and (5), A is the so-called *generalized attribute* (Ligęza, 2006). From now on, we will refer to both types of attributes as simply attributes.

As an example for the necessity of SAL one can consider the specification of working days (denoted with $WDay$) given as $WDay = \{Monday, Tuesday, Wednesday, Thursday, Friday\}$, Now one can construct an atomic formula like $DaysOfInterest \subseteq WDay$, or a rule of the form $DaysOfInterest \subseteq WDay \longrightarrow Status(OfficeOfInterest) = open$.

SAL as introduced in (Ligęza, 2006) seems to be an important step towards the study and extension of attributive logic towards practical applications. On the other hand, it still suffers from the lack of expressive power, and the provided semantics of the atomic formulae are poor.

In this paper, an improved and extended version of SAL, namely, Attributive Logic with Set Values over Finite Domains (ALSV(FD)), first introduced in (Ligęza and Nalepa, 2007; Nalepa and Ligęza, 2008), is presented in brief. For simplicity, no objects are specified in an explicit way. The formalism is oriented towards Finite Domains (FDs), and its expressive power is increased through the introduction of new relational symbols. The semantics is also clarified. The practical representation and inference issues both at the logical and the implementation level are tackled. The main extension consists of a proposal of an extended set of relational symbols enabling definitions of atomic formulae. The values of attributes can take singular and set values over finite domains.

6.1. ALSV(FD). Attribute names and attribute values are the basic element of the language of *Attribute Logic with Set Values over Finite Domains* (ALSV(FD) for short). Let us consider

A —a finite set of attribute names,

D —a set of possible attribute values (their *domains*).

Let $\mathbf{A} = \{A_1, A_2, \ldots, A_n\}$ be all attributes such that their values define the state of the system under consideration. It is assumed that the overall domain $\mathbf{D}$ is divided into n sets (disjoint or not), $\mathbf{D} = D_1 \cup D_2 \cup \ldots \cup D_n$, where D_i is the domain related to attribute A_i, $i = 1, 2, \ldots, n$. Any domain D_i is assumed to be a finite (discrete) set. The set can be ordered, partially ordered, or unordered.

Let A_i be an attribute of $\mathbf{A}$ and D_i the sub-domain related to it. Let V_i denote an arbitrary subset of D_i, and let $d \in D_i$ be a single element of the domain. The legal atomic formulae of ALSV along with their semantics are presented in Tables 1 and 2, for simple and general attributes, respectively.

If V_i is an empty set (in fact, the attribute takes no value), we shall write $A_i = \emptyset$. In case the value of A_i is unspecified, we shall write $A_i = \texttt{NULL}$ (database convention). If we do not care about the current value of the attribute, we shall write $A = \_$ (PROLOG convention).

More complex formulae can be constructed with *conjunction* ($\wedge$) and *disjunction* ($\vee$); both of these symbols have classical meaning and interpretation.

There is no explicit use of negation. The proposed set of relations is selected for convenience, and therefore as such these are not completely independent. For example, $A_i = V_i$ can perhaps be defined as $A_i \subseteq V_i \wedge A_i \supseteq V_i$; but it is much more concise and natural to use just "=" directly. Various conventions extending the basic notation can be used. For example, with domains being ordered sets, relational symbols such as $>$, $>=$, $<$, $=<$ can be used with straightforward meaning.

The semantics of the proposed language are presented below in an informal way. The semantics of $A = V$ are basically the same as those of SAL (Ligęza, 2006). If $V = \{d_1, d_2, \ldots, d_k\}$, then the attribute $A = V$ takes all the values specified with V (and nothing more). The semantics of $A \subseteq V$, $A \supseteq V$ and $A \sim V$ are defined as

$$A \subseteq V \equiv A = U$$

for some U such that $U \subseteq V$, i.e., A takes *some* of the values from V (and nothing out of V),

$$A \supseteq V \equiv A = W,$$

for some W such that $V \subseteq W$, i.e., A takes *all* of the values from V (and perhaps some more), and

$$A \sim V \equiv A = X,$$

for some X such that $V \cap X \neq \emptyset$, i.e., A takes *some* of the values from V (and perhaps some more). As can be seen, the semantics of ALSV are defined by means of the relaxation of logic to simple set algebra.

[6]For uniformity, single elements can be considered single-element sets. Hence (4) can be replaced with (5) if it is not misleading.

Table 1. Simple attribute formulae syntax.

Syntax	Interpretation: true if ...	Relation
$A_i = d$	the value is precisely defined	`eq`
$A_i \in V_i$	the current value of A_i belongs to V_i	`in`
$A_i \neq d$	shorthand for $A_i \in D_i \setminus \{d\}$.	`neq`
$A_i \notin V_i$	is a shorthand for $A_i \in D_i \setminus V_i$.	`notin`

Table 2. Generalized attribute formulae syntax.

Syntax	Interpretation: true if ...	Relation
$A_i = V_i$	equals to V_i (and nothing more)	`eq`
$A_i \neq V_i$	is different from V_i (at at least one element)	`neq`
$A_i \subseteq V_i$	is a subset of V_i	`subset`
$A_i \supseteq V_i$	is a superset of V_i	`supset`
$A \sim V$	has a non-empty intersection with V_i	`sim`
$A_i \not\sim V_i$	has an empty intersection with V_i	`notsim`

6.2. XTT rules in ALSV(FD). 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

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

where $\propto_i$ is one of the admissible relational symbols in ALSV(FD), and RHS is the right-hand side of the rule (RHS) covering conclusions. In practice, the conclusions are restricted to assigning new attribute values, thus changing the system state. The values that are no longer valid are removed from the state (for details, see (Ligęza, 2006)).

Knowledge representation with extended tabular trees incorporates extended attributive table format. Furthermore, 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. After completing the conclusion part, this can be expressed as shown in Table 3.

In Table 3, the symbol $\propto_{ij} \in \{=, \neq, \in, \notin\}$ is used 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. If this is the case, the relation symbol can be omitted (i.e., it constitutes the default relation which can be identified by the type of the attribute and its value).

Table 3. 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

The current values of all the attributes are specified within the contents of the knowledge-base (including current sensor readings, measurements, inputs examination, etc.). From the logical point of view, it is a formula of the form previously introduced by the formula (1) in Section 4.2. Having a table with defined rules, the execution mechanism searches for ones with satisfied preconditions. In short: the satisfaction of preconditions is verified in an algebraic mode, using the dependencies specified in the first row of Table 4 for simple attributes and the first row of Table 5 for complex ones (see Section 7 for more details). Rules with all preconditions satisfied can be fired. In general, rules can be fired in parallel or sequentially. For the following analysis, we assume the classical, sequential model, i.e., rules are examined in turn in top-down order and fired if the preconditions are satisfied. The details of inference with ALSV(FD) needed to implement a complete XTT inference solution are given in the following section.

7. Basic inference rules for ALSV(FD)

Since the presented language is an extension of SAL (Ligęza, 2006), its simple and intuitive semantics are consistent with SAL and clear up some points of it. The summary of the inference rules for atomic formulae with simple attributes (where an atomic formula is the logical consequence of another atomic formula) is presented in Table 4. The table is to be read as follows: if an atomic formula in the leftmost column holds, and a condition stated in the same row is true, then the corresponding atomic formula in the topmost row is also true. In other words, the formula in the topmost row is a logical consequence of the one from the leftmost column provided the condition is fulfilled.

The summary of the inference rules for atomic formulae with generalized attributes (where an atomic formula is the logical consequence of another atomic formula) is presented in Table 5. An example how the first row should be read is as follows: if $A = V$ (see the leftmost column) and provided that $V \subseteq W$ (the same row, the fourth column), we can conclude that $A \subseteq W$ (the topmost row).

In Table 4 and 5, the conditions are *satisfactory*. However, it is important to note that in the case of the first rows of the tables (the cases of $A = d_i$ and $A = V$, respectively), all the conditions are also *necessary*. The rules in Tables 4 and 5 can be used to check if the precon-

Table 5. Inference for atomic formulae, generalized attributes.

$\models$	$A = W$	$A \neq W$	$A \subseteq W$	$A \supseteq W$	$A \sim W$	$A \not\sim W$
$A = V$	$V = W$	$V \neq W$	$V \subseteq W$	$V \supseteq W$	$V \cap W \neq \emptyset$	$V \cap W = \emptyset$
$A \neq V$	_	$V = W$	$W = D$	_	$W = D$	_
$A \subseteq V$	_	$V \subset W$	$V \subseteq W$	_	$W = D$	$V \cap W = \emptyset$
$A \supseteq V$	_	$W \subset V$	$W = D$	$V \supseteq W$	$V \cap W \neq \emptyset$	_
$A \sim V$	_	$V \cap W = \emptyset$	$W = D$	_	$V = W$	_
$A \not\sim V$	_	$V \cap W \neq \emptyset$	$W = D$	_	$W = D$	$V = W$

Table 4. Inference for atomic formulae, simple attributes.

$\models$	$A = d_j$	$A \neq d_j$	$A \in V_j$	$A \notin V_j$
$A = d_i$	$d_i = d_j$	$d_i \neq d_j$	$d_i \in V_j$	$d_i \notin V_j$
$A \neq d_i$	_	$d_i = d_j$	$V_j = D \setminus \{d_i\}$	$V_j = \{d_i\}$
$A \in V_i$	$V_i = \{d_j\}$	$d_j \notin V_i$	$V_i \subseteq V_j$	$V_i \cap V_j = \emptyset$
$A \notin V_i$	$D \setminus V_i = \{d_j\}$	$V_i = \{d_j\}$	$V_j = D \setminus V_i$	$V_j \subseteq V_i$

Table 6. Inconsistency conditions for atomic formulae pairs.

$\not\models$	$A = W$	$A \subseteq W$	$A \supseteq W$	$A \sim W$
$A = V$	$W \neq V$	$V \not\subseteq W$	$W \not\subseteq V$	$V \cap W \neq \emptyset$
$A \subseteq V$	$W \not\subseteq V$	$V \cap W = \emptyset$	$W \not\subseteq V$	$W \cap V = \emptyset$
$A \supseteq V$	$V \not\subseteq W$	$V \not\subseteq W$	_	_
$A \sim V$	$V \cap W = \emptyset$	$V \cap W = \emptyset$	_	_

ditions of a formula hold or to verify subsumption among rules.

For further analysis, e.g., of the intersection (overlapping) of rule preconditions, one may be interested if two atoms cannot simultaneously be true, and if so—under what conditions. For example, formula $A \subseteq V \wedge A \subseteq W$ is inconsistent if $V \cap W = \emptyset$. Table 6 specifies the conditions for inconsistency.

The interpretation of Table 6 is straightforward: if the condition specified at the intersection of some row and column holds, then the atomic formulae labelling this row and column cannot hold simultaneously. Note, however, that this condition is just satisfactory.

Table 6 can be used for the analysis of system determinism, i.e., whether satisfying a rule precondition implies that the other rules in the same table cannot be fired.

Having the inference on the rule level established, let us now move to the discussion how the inference with the XTT tables (see Table 3) grouping XTT rules (see Section 6) operating in the same context is performed.

8. Inference strategies for XTT

Any XTT table can have one or more inputs. Let T denote a table. By $I(T)$ we shall denote the number of input links to table T. If there are k such input links, they will be denoted as $1.T, 2.T, \ldots, k.T$.

Note that all the links are, in fact, considered an AND connection. In order to fire table T, all the input tables from which the input links come must be fired to provide the necessary information to fire T.

Similar conclusions regard the output. Any table can have one or more output links (from different rules which are placed at the rows of the table), and such an output link can be directed to one or more tables. If there are m such output links, we shall denote them as $T.1, T.2, \ldots, T.m$.

If an output link $T.j$ goes to n tables $T1, T2, \ldots, Tn$, then the links can be denoted as $T.j \rightarrow T1, T.j \rightarrow T2, \ldots, T.j \rightarrow Tn$. In fact, we can have $\Sigma_{j=1}^{m} dim(j)$, where $dim(j)$ is the number of addressed tables (here n).

The XTT tables to which no connections point are referred to as *input tables*. The XTT tables with no connections pointing to other tables are referred to as *output tables*. All the other tables (ones having both input and output links) are referred to as *middle tables*.

Now, consider a network of tables connected according to the following principles:

- there is one or more input table,
- there is one or more output table,
- there is zero or more middle tables,
- all the tables are interconnected.

The problem is how to choose the inference order. The basic principle is that, before firing a table, all the immediately preceding tables must have already been fired. The structure of the network imposes a partial order with respect to the order of table firing. Below, we describe three possible algorithms for inference control.

8.1. Fixed-order approach. The simplest algorithm consists of a hard-coded order of inference, in such a way that every table is assigned an integer number; all the numbers are different from one another. The tables are fired in order from the lowest number to the highest one.

In order to ensure the executability of the inference process, the assignment of such numbers should fulfill the following minimal requirement: for any table T, the numbers assigned to tables being predecessors of T must all be lower than the one assigned to T.

After starting the inference process, the predefined order of inference is followed. The inference stops after firing the last table. In case a table contains a complete set of rules (with respect to possible outputs generated by preceding tables), the inference process should end with all the output values defined by all the output tables being produced.

8.2. Token-transfer approach. This approach is based on monitoring the partial order of inference defined by the network structure with tokens assigned to tables. A table can be fired only when there is a token at each input. Intuitively, a token at the input is a kind of a flag signalling that the necessary data generated by the preceding table are ready for use.

The tables ready to be fired (with all tokens at the input) are placed in an FIFO queue. The outline of this algorithm is as follows:

- Since input tables have 0 inputs, they automatically have all the tokens they need.
- All the input tables are placed in an FIFO queue (in arbitrary order).
- Then the following procedure is repeated; the first table from the queue is fired and removed from the queue, the token is removed from its input and placed at the active output link and passed to all following tables. Simultaneously, if a token is passed to a table, the table is immediately checked if it has tokens at all the inputs; if so, it is appended to the end of the queue.
- The process stops when the queue is empty (no further tables to fire).

Note that this model of inference execution covers the case of possible loops in the network. For example, if there is a loop and a table should be fired several times in turn, the token is passed from its output to its input, and it is analyzed if it can be fired; if so, it is placed in the queue.

8.3. Goal-driven approach. The presented models of inference control can be considered to be *blind* procedures since they do not take into consideration the goal of inference. Hence, it may happen that numerous tables are fired without purpose—the results they produce are of no interest. This, in fact, is a deficiency of most of the forward-chaining rule-based inference control strategies.

A *goal-driven approach* works backwards with respect to *selecting* the tables necessary for a specific task, and then fires the tables forwards so as to achieve the goal. The principles of backward search procedures are as follows:

- one or more output tables are identified as the ones that can generate the desired goal values: these are the tables that must be fired;
- these tables are stored on a stack (LIFO queue) in an arbitrary order;
- the following procedure is repeated: the list of tables from the queue is examined and all the input tables are identified; they are placed on the stack, while the analyzed table is marked as "needed" and removed from the queue;
- only unvisited tables are examined;
- for input tables no analysis is necessary; all the input tables necessary to start the process are identified.

The execution is performed forward using the token-transfer approach. Tables which are not marked as "needed" on the stack are not fired—they are not necessary to achieve the goal.

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

9. XTT rule runtime and design tools

The XTT approach introduces a new knowledge representation as well as inference algorithms. This makes the use of the existing rule design and implementations impractical and often impossible. The HeKatE project aims at developing not just the conceptual methods but also practical computer tools to support them. Within the project, a number of tools to support the design and implementation of XTT-based systems has been developed. These include the following:

- visual design tools for rule design (HQEd) and prototyping (HJEd),
- rule runtime environment (HeaRT), and
- rule translation facilities (HaThoR).

All of these tools are built around the XTT knowledge model.

The visual XTT model is represented by means of a human readable, algebraic notation, also called the XTT presentation syntax, or HeKatE Meta Representation (HMR). An example excerpt of HMR is given below:

```
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 are as follows: "the XTT table *th* has two conditional attributes: *today* and *hour*, and one decision attribute: *operation*". This information is determined from the conceptual design phase using the ARD method. 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 the 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*". For a more complete and up-to-date description of HMR, see the HeKatE wiki.[7]

The above representation can be directly run by the HeKatE RunTime environment (HeaRT) (Nalepa *et al.*, 2009). An HMR file is, in fact, a legal Prolog code that can be interpreted directly (a number of custom operators are defined). Therefore, the HeaRT prototype is implemented in Prolog (Bratko, 2000) and provides implementation for a number of inference solutions, including the three described before. The engine can then be embedded into a complete application, including the interfaces and the presentation layer, using the MVC pattern (see Section 3). In this case, HeaRT provides the logical *Model*. The control logic is built in a declarative way, using structured XTT representation. The engine provides a flexible communication mechanism using callbacks that can be run on the attribute value updates. The callbacks can be implemented in any language, using a Prolog interface; currently, Java, Python and PHP callback interfaces are available.[8]

In order to design the XTT model, visual editors introduced in Section 3 are used. In the ARD rule prototyping phase, two tools are currently available: VARDA and HJEd. They support the visual design of ARD diagrams, modeling functional dependencies between XTT rule attributes. VARDA is a proof of a concept prototype tool written in Prolog. HJEd is a portable visual editor implemented in Java. An example session with HJEd is presented in Fig. 5. The output from this phase allows the generation of XTT table schemes.

One of the main features of the XTT method is the compact visual representation. From the designer's point of view, it needs to be supported by a CASE tool in order to be effective. The *HQed* (Kaczor, 2008; Kaczor and Nalepa, 2008) tool (Fig. 6) uses the rule prototypes generated in the conceptual design, and supports the actual visual process of the logical design of XTT tables. It is a cross-platform tool written in C++ and the Qt library. One of the most important editor features is the support for XTT; it concerns *rulebase quality assurance* and refers to several specific issues: condition specification constraints, structured rulebase syntax, gradual model refinement, with partial simulation logical rule model quality. The first issue is tackled by providing a number of editing features, enforcing strict user data verification. Every attribute value entered into XTT cells (corresponding to ALSV(FD) formulae) is checked against the attribute domain. On the other hand, the user is hinted during the editing process with feasible attribute values.

The rulebase syntax may be checked against anomalies, e.g., incomplete rule specification, malformed inference specification, including missing table links. The editor allows gradual rule refinement, with online checking of attribute domains, as well as simple table properties, such as inference-related dead rules. With simple tables, it is possible to emulate and visualize the inference process.

However, the main quality feature being developed is a plugin framework, allowing the integration of Prolog-based components for rule analysis (being part of the inference engine) to check the formal properties of the XTT rule base, such as completeness, redundancy, or determinism (Ligęza and Nalepa, 2005). Here, analysis is performed on the logical level, where the rows of the XTT tables are interpreted and analyzed as ALSV(FD) formulae.

Another important group of HeKatE tools are knowledge translators (HaThoR). They use HMR *serialization* to an XML-based format known as the HML (*HeKatE Markup Language*). This format is used by editing tools to store the model together with some technical information, such as diagram location in the editor, etc. HML translators are implemented with the use of XSLT (Clark, 1999), which provides an XML-based syntax for defining XML translators. The HaThoR framework aims at allowing exchange between HeKatE knowledge representation and other important rule formats. These are mainly being developed by the Semantic Web initiative[9] and additionally include RIF[10], which provides a generic rule interchange format, and the SWRL (*A Semantic Web Rule Language*) (Horrocks *et al.*, 2004), which builds on ontologies.

[7]`https://ai.ia.agh.edu.pl/wiki/hekate:hmr`.

[8]See `https://ai.ia.agh.edu.pl/wiki/hekate:heart` for a more up-to-date description of HeaRT.

[9]`http://www.w3.org/2001/sw`.

[10]`http://www.w3.org/2005/rules/wiki/RIF_Working_Group`.

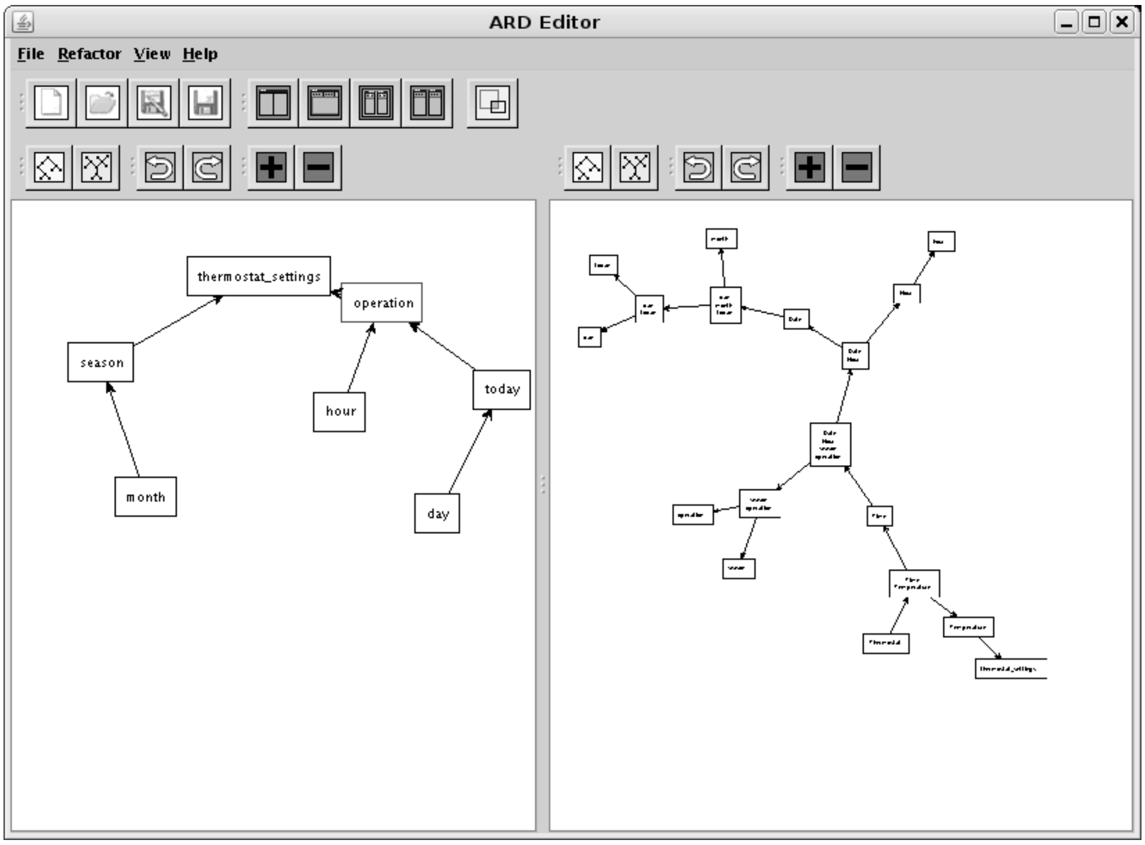
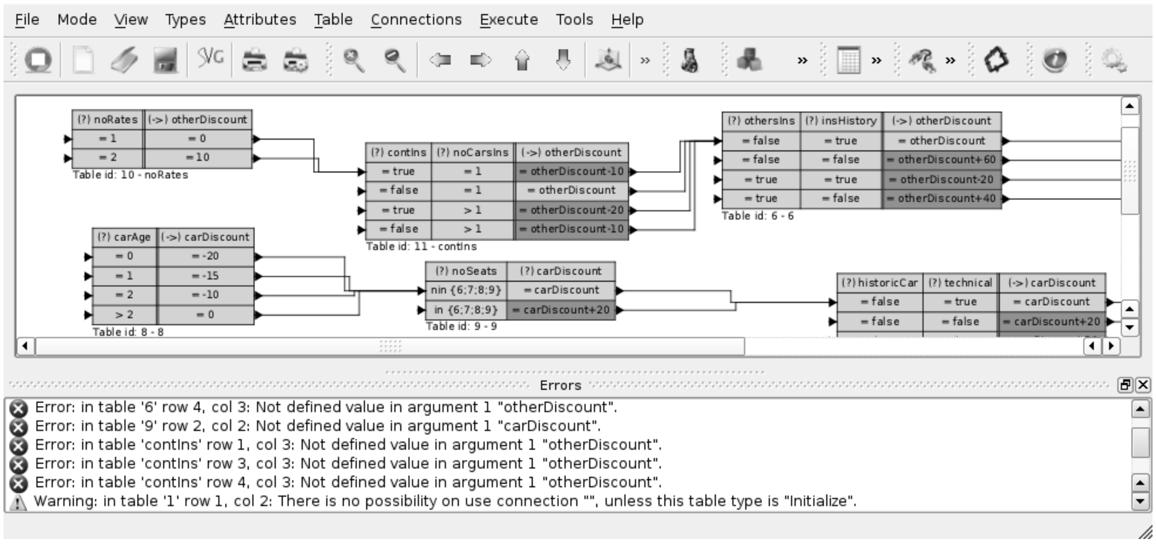


Fig. 5. ARD diagram editing with HJEd.


File Mode View Types Attributes Table Connections Execute Tools Help
Errors

(?) noRates	(->) otherDiscount
= 1	= 0
= 2	= 10

Table id: 10 - noRates

(?) contIns	(?) noCarsIns	(->) otherDiscount
= true	= 1	= otherDiscount-10
= false	= 1	= otherDiscount
= true	> 1	= otherDiscount-20
= false	> 1	= otherDiscount-10

Table id: 11 - contIns

(?) othersIns	(?) insHistory	(->) otherDiscount
= false	= true	= otherDiscount
= false	= false	= otherDiscount+60
= true	= true	= otherDiscount-20
= true	= false	= otherDiscount+40

Table id: 6 - 6

(?) carAge	(->) carDiscount
= 0	= -20
= 1	= -15
= 2	= -10
> 2	= 0

Table id: 8 - 8

(?) noSeats	(?) carDiscount
nin {6;7;8;9}	= carDiscount
in {6;7;8;9}	= carDiscount+20

Table id: 9 - 9

(?) historicCar	(?) technical	(->) carDiscount
= false	= true	= carDiscount
= false	= false	= carDiscount+20

```
Error: in table '6' row 4, col 3: Not defined value in argument 1 "otherDiscount".
Error: in table '9' row 2, col 2: Not defined value in argument 1 "carDiscount".
Error: in table 'contIns' row 1, col 3: Not defined value in argument 1 "otherDiscount".
Error: in table 'contIns' row 3, col 3: Not defined value in argument 1 "otherDiscount".
Error: in table 'contIns' row 4, col 3: Not defined value in argument 1 "otherDiscount".
Warning: in table '1' row 1, col 2: There is no possibility on use connection "", unless this table type is "Initialize".
```

Fig. 6. HQEd editing session, the XTT rulebase structure with anomalies detected.

All of the tools are released as *free software*[11] under the terms of the GNU GPL licence from the project website, see `http://hekate.ia.agh.edu.pl`.

[11]`http://www.gnu.org/philosophy/free-sw.html`.

10. Related tools

The primary idea behind XTT as a knowledge representation and HeKatE as a design methodology was to overcome selected important limitations of well-established approaches (see Section 3.1). Considering that fact, it is important to briefly evaluate the achieved results. Here, the focus is on two most important solutions that have become *de facto* standards and are openly available (see Section 2). The first is CLIPS and its new Java-based incarnation—Jess. The other one is Drools, which inherits some of the important CLIPS features, such as the Rete-based inference, while providing a number of high-level integration features on the Java Enterprise Edition platform.

Previously, in Section 3.1, three main problems areas where identified, that is, (i) knowledge representation, (ii) a transparent structure and efficient inference in the rulebase, (iii) a well-founded systematic and complete design process.

As of the first issue, XTT provides an expressive, formally defined language to describe rules. The language allows 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 a set of attribute values. This solution fits well in a number of well defined control or decision precesses. However, it might prove insufficient in more general applications. In that case, CLIPS/Jess offers several other programming paradigms, including an object-oriented one. Drools is integrated with the Java stack, opening means to implement almost any behavior. One should keep in mind that such heterogeneous multi-paradigm programs cannot be considered plain "rule-based systems" anymore. It should also be emphasized that, in a general case, formal analysis of such systems is hardly possible.

The implicit rule base structure is another important feature of XTT. Rules are grouped into decision tables during design, and inference control is designed during conceptual design, and later on refined during logical design. Therefore, XTT representation is highly optimized towards rulebase structuring. This is different from all Rete-based solutions, including all three previously mentioned, that is, CLIPS, Jess, and Drools. This feature makes visual design much more transparent and scalable. It also greatly improves the inference process.

It is worth noting that, in fact, all the Rete-based solutions seek some kind of structuring. In the case of CLIPS, it is possible to modularize the rulebase (see Chapter 9 in (Giarratano and Riley, 2005)). It is possible to group rules into modules operating in given contexts, and then to provide context switching logic. Such a modularized structure can correspond to phases of the decision process. Another solution is the hierarchization of the rule set. Drools 5 offers Drools Flow, which allows defining the rule set and a 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 solutions. In XTT, rules having the same attributes are grouped by design. This opens up possibilities for inference optimization and strong formal analysis on the table level. A group (subtree) of linked XTT tables can also be defined to work in a given context.

The last issue concerning a complete design process seems to be in practice the most important one. 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. In fact, the implementation process for these systems can be supported by a number of external environments, with Eclipse[12] being the best example. Jess 7 provides JessDE, and an Eclipse plugin-based environment. However, it is worth emphasizing that these tools do not visualize the knowledge contained in the rule base. In fact, they simplify syntax checking, runtime debugging (including the Rete network view) and rule management.

Drools 5 is decomposed into four main parts: Guvnor, Expert, Flow, Fusion. It offers several support tools, namely, the Eclipse-based environment, similar but more robust than the one Jess provides. One of the "design support" features is the ability to read Excel files containing simple decision tables, with the basic control structure. While this is a valuable feature, it does not provide any on-line syntax checking.

In both cases, 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, in software engineering by UML, or by Petri nets for parallel systems.

11. Conclusions

The primary area of interest of this paper is the design of rule-based intelligent control and decision support systems. In the paper, a review of the state-of-the-art in rule-based systems is given. Based on it, some practical issues in the design of rule-based controllers are discussed. Major drawbacks in the existing solutions are identified, including limitations in the formal language definition, the internal knowledge structure, and the systematic design procedure.

This analysis provides guidelines for the HeKatE approach. It aims at solving these limitations by the introduction of a custom formalized rule language called XTT (in fact, the version considered in this paper is called XTT^2). The formalization is based on ALSV(FD) logic. In the described approach, a systematic design process for

[12] `http://www.eclipse.org.`

rule-based systems is considered. It is practically supported by a number of tools presented in the paper.

The original contribution of the presented approach covers the following issues:

- a clear definition of formal logical language with an intuitive interpretation for knowledge representation,
- a modular structure of the rule base, oriented towards modular development methodology,
- goal-directed inference and flexible inference modes,
- formal verification of XTT knowledge modules,
- consistent top-down design methodology supported by software tools.

Some future extensions of the presented formalism are also considered, including fuzzy rules. However, these are out of scope of this paper. For more details, see (Ligęza and Nalepa, 2008).

The up-to-date results of the project, as well all the relevant papers, are available on the project website (`http://hekate.ia.agh.edu.pl`). A repository of tools available for download is also open for the community.

Acknowledgment

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

The authors wish to thank the anonymous reviewers for their valuable comments and remarks that greatly helped in improving the final version of the paper.

Special thanks are due Dr. Marcin Szpyrka, Dr. Sławomir Nowaczyk, Claudia Obermaier, and Weronika T. Furmańska for reading the preliminary versions of the paper and providing us with valuable comments, remarks and criticism that helped us to improve the article as well as gain a broader perspective on our work.

We would like to kindly acknowledge an important contribution from our Master's degree students Krzysztof Kaczor, Michał Gawędzki, Szymon Bobek, and Szymon Książek, who implemented the tools mentioned in this paper.

References

Ben-Ari, M. (2001). *Mathematical Logic for Computer Science*, Springer-Verlag, London.

Bratko, I. (2000). *Prolog Programming for Artificial Intelligence, 3rd Edn*, Addison Wesley, Harlow.

Browne, P. (2009). *JBoss Drools Business Rules*, Packt Publishing, Birmingham.

Brownston, L., Farrell, R., Kant, E. and Martin, N. (1985). *Programming Expert Systems in OPS5*, Addison-Wesley, Reading, MA/Menlo Park, CA.

Burbeck, S. (1992). Applications programming in smalltalk-80(tm): How to use model-view-controller (MVC), *Technical report*, Department of Computer Science, University of Illinois, Urbana-Champaign, IL.

Cheng, A. M. K. (2002). *Real-Time Systems. Scheduling, Analysis and Verification*, John Wiley & Sons, Inc., Hoboken, NJ.

Clark, J. (1999). Xsl transformations (xslt) version 1.0 w3c recommendation 16 November 1999, *Technical report*, World Wide Web Consortium (W3C).

Connolly, T., Begg, C. and Strechan, A. (1999). *Database Systems. A Practical Approach to Design, Implementation, and Management, 2nd Edn*, Addison-Wesley, Harlow/Reading, MA.

Coenen, F., *et al.* (2000). Validation and verification of knowledge-based systems: Report on eurovav99, *The Knowledge Engineering Review* **15**(2): 187–196.

Forgy, C. (1982). Rete: A fast algorithm for the many patterns/many objects match problem, *Artificial Intelligence* **19**(1): 17–37.

Friedman-Hill, E. (2003). *Jess in Action, Rule Based Systems in Java*, Manning, Greenwich, CT.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns, 1st Edn*, Addison-Wesley Pub Co, Reading, MA.

Genesereth, M. R. and Nilsson, N. J. (1987). *Logical Foundations for Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., Los Altos, CA.

Giarratano, J. C. and Riley, G. D. (2005). *Expert Systems*, Thomson, Boston, MA.

Giurca, A., Gasevic, D. and Taveter, K. (Eds) (2009). *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*, Information Science Reference, Hershey, New York, NY.

Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosof, B. and Dean, M. (2004). SWRL: A semantic web rule language combining OWL and RuleML: W3C member submission 21 may 2004, *Technical report*, W3C.

Jackson, P. (1999). *Introduction to Expert Systems, 3rd Edn*, Addison-Wesley, Harlow.

Kaczor, K. (2008). *Design and implementation of a unified rule base editor*, M.Sc. thesis, AGH University of Science and Technology, Cracow.

Kaczor, K. and Nalepa, G. J. (2008). Design and implementation of hqed, the visual editor for the xtt+ rule design method, *Technical Report CSLTR 02/2008*, AGH University of Science and Technology, Cracow.

Klösgen, W. and Żytkow, J. M. (Eds) (2002). *Handbook of Data Mining and Knowledge Discovery*, Oxford University Press, New York, NY.

Laffey *etal*, T. (1988). Real-time knowledge-based systems, *AI Magazine* **Spring**: 27–45.

Liebowitz, J. (Ed.) (1998). *The Handbook of Applied Expert Systems*, CRC Press, Boca Raton, FL.

Ligęza, A. (1986). An expert systems approach to analysis and control in certain complex systems, *Preprints of the 4-th IFAC/IFIP Symposium on Software for Computer Control SOCOCO'86, Graz, Austria*, pp. 147–152.

Ligęza, A. (1988). Expert systems approach to decision support, *European Journal of Operational Research* **37**(1): 100–110.

Ligęza, A. (1993). Logical foundations for knowledge-based control systems—Knowledge representation, reasoning and theoretical properties, *Scientific Bulletins of AGH: Automatics* **63**(1529): 144.

Ligęza, A. (1996). Logical support for design of rule-based systems. Reliability and quality issues, *in* M. Rousset (Ed.), *ECAI-96 Workshop on Validation, Verification and Refinment of Knowledge-based Systems*, Vol. W2, ECCAI (European Coordination Committee for Artificial Intelligence), Budapest, pp. 28–34.

Ligęza, A. (1998). Towards logical analysis of tabular rule-based systems, *Proceedings of the Ninth European International Workshop on Database and Expert Systems Applications, Vienna, Austria,* pp. 30–35.

Ligęza, A. (1999). *Validation and Verification of Knowledge Based Systems: Theory, Tools and Practice*, Kluwer Academic Publishers, Boston, MA/Dordrecht/London, pp. 313–325.

Ligęza, A. (2001). Toward logical analysis of tabular rule-based systems, *International Journal of Intelligent Systems* **16**(3): 333–360.

Ligęza, A. (2005). *Logical Foundations for Rule-Based Systems*, AGH University of Science and Technology Press, Cracow.

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

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

Ligęza, A. and Nalepa, G. J. (2007). Knowledge representation with granular attributive logic for XTT-based expert systems, *in* D. C. Wilson, G. C. J. Sutcliffe and FLAIRS (Eds), *FLAIRS-20: Proceedings of the 20th International Florida Artificial Intelligence Research Society Conference: Key West, Florida, May 7-9, 2007*, Florida Artificial Intelligence Research Society, AAAI Press, Menlo Park, CA, pp. 530–535.

Ligęza, A. and Nalepa, G. J. (2008). Granular logic with variables for implementation of extended tabular trees, *in* D. C. Wilson and H. C. Lane (Eds), *FLAIRS-21: Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society conference: 15–17 May 2008, Coconut Grove, Florida, USA*, AAAI Press, Menlo Park, CA, pp. 341–346.

Ligęza, A., Wojnicki, I. and Nalepa, G. J. (2001). Tab-trees: A case tool for design of extended tabular systems, *in* H.C. Mayr, J. Lazansky, G. Quirchmayr and P. Vogel (Eds), *Database and Expert Systems Applications*, Lecture Notes in Computer Sciences, Vol. 2113, Springer-Verlag, Berlin, pp. 422–431.

Ligęza, A. (1996). *Logical Support for Design of Rule-Based Systems. Reliability and Quality Issues* LAAS, Report No. 96170, Toulouse.

Morgan, T. (2002). *Business Rules and Information Systems. Aligning IT with Business Goals*, Addison Wesley, Boston, MA.

Nalepa, G. J. (2004). *Meta-Level Approach to Integrated Process of Design and Implementation of Rule-Based Systems*, PhD thesis, AGH University of Science and Technology, Institute of Automatics, Cracow.

Nalepa, G. J. and Ligęza, A. (2005a). A graphical tabular model for rule-based logic programming and verification, *Systems Science* **31**(2): 89–95.

Nalepa, G. J. and Ligęza, A. (2005b). *Software Engineering: Evolution and Emerging Technologies*, Frontiers in Artificial Intelligence and Applications, Vol. 130, IOS Press, Amsterdam, pp. 330–340.

Nalepa, G. J. and Ligęza, A. (2005c). A visual edition tool for design and verification of knowledge in rule-based systems, *Systems Science* **31**(3): 103–109.

Nalepa, G. J. and Ligęza, A. (2006). Prolog-based analysis of tabular rule-based systems with the "xtt" approach, *in* G. C. J. Sutcliffe and R. G. Goebel (Eds), *FLAIRS 2006: Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference: Melbourne Beach, Florida, May 11–13, 2006*, Florida Artificial Intelligence Research Society, AAAI Press, Menlo Park, CA, pp. 426–431.

Nalepa, G. J. and Ligęza, A. (2008). Xtt+ rule design using the alsv(fd), *in* A. Giurca, A. Analyti and G. Wagner (Eds), *ECAI 2008: 18th European Conference on Artificial Intelligence: 2nd East European Workshop on Rule-based Applications, RuleApps2008: Patras, 22 July 2008*, University of Patras, Patras, pp. 11–15.

Nalepa, G. J., Ligęza, A., Kaczor, K. and Furmańska, W. T. (2009). Hekate rule runtime and design framework, *in* G. W. Adrian Giurca and G.J. Nalepa (Eds), *Proceedings of the 3rd East European Workshop on Rule-Based Applications (RuleApps 2009), Cottbus, Germany, September 21, 2009*, BTU Cottbus, Cottbus, pp. 21–30.

Pawlak, Z. (1991). *Rough Sets. Theoretical Aspects of Reasoning about Data*, Kluwer Academic Publishers, Dordrecht/Boston, MA/London.

Quinlan, J. R. (1987). Simplifying decision trees, *International Journal of Man-Machine Studies* **27**(3): 221–234.

Ross, R. G. (2003). *Principles of the Business Rule Approach, 1st Edn*, Addison-Wesley Professional, Reading, MA.

Torsun, I. S. (1995). *Foundations of Intelligent Knowledge-Based Systems*, Academic Press, London/San Diego, CA/New York, NY/Boston, MA/Sydney/Tokyo/Toronto.

Tzafestas, S. and Ligęza, A. (1988). Expert control through decision making, *Foundations of Control Engineering* **13**(1): 43–51.

Tzafestas, S. and Ligęza, A. (1989). Expert control through decision making, *Journal of Intelligent and Robotic Systems* **1**(4): 407–425.

van Harmelen, F., Lifschitz, V. andPorter, B. (Eds) (2007). *Handbook of Knowledge Representation*, Elsevier Science, Amsterdam.

van Harmelen, F. (1996). Applying rule-based anomalies to kads inference structures, *ECAI'96 Workshop on Validation, Verification and Refinement of Knowledge-Based Systems, Budapest, Hungary,* pp. 41–46.

von Halle, B. (2001). *Business Rules Applied: Building Better Systems Using the Business Rules Approach*, Wiley, New York, NY.

Grzegorz J. Nalepa holds the position of an assistant professor at the AGH University of Science and Technology in Cracow, Poland, Department of Automatics. Since 1995 he has been actively involved in a number of research projects, including Regulus, Mirella, Adder, HeKatE, and recently INDECT. His Ph.D., received from the AGH UST in 2004, concerned new design methods for intelligent systems. Since 2007 he has been coordinating the HeKatE project, led by Prof. Antoni Ligęza. He is the author of a number of publications from the domains of knowledge engineering, intelligent systems and software engineering. His fields of interest also include computer security and operating systems. He has formulated a new design and implementation approach for intelligent rule-based systems called XTT (eXtended Tabular Trees). He is involved in several conferences and workshops, including *DERIS* at *FLAIRS, RuleApps, AISEW*, and *KESE*. Since 2008 he has been co-organizing the Knowledge and Software Engineering Workshop (*KESE*) at *KI*, the German AI conference.

Antoni Ligęza is a full professor in the domain of computer science at the AGH University of Science and Technology in Cracow, Poland. His principal areas of investigation are artificial intelligence and knowledge engineering. He lectures on knowledge engineering, databases, Prolog, automated diagnosis, discrete mathematics and logics. He is a member of the ACM and the IEEE Computer Society. He is an author and co-author of over 200 research publications, including international conference papers, journals, chapters in books. His recent book *Logical Foundations for Rule-Based Systems* was published by Springer in 2006, and it covers issues ranging from logic foundations, propositional, attributive and first-order logics, through various forms of rule-based systems to design and verification issues. He has been a visiting scholar and worked in Denmark (Technical University of Lyngby) for four months, France (LAAS of CNRS, Toulouse; University of Nancy I, Nancy; CRIL Lens; University of Caen, Caen) for about two years in total, and Spain (University of Balearic Islands, Palma de Mallorca; University of Girona, Girona) for about one year.

Received: 1 February 2009
Revised: 20 July 2009