

Chapter 2

Knowledge Engineering with Rules

Intelligent systems that use knowledge representation and reasoning methods described in the previous chapter are commonly referred to as *Knowledge-Based Systems* (KBS). The domain that considers building KBS is most generally referred to as *Knowledge Engineering* (KE). It involves a number of development methods and processes. A general KE process can be found in several text-books in this field [1, 2]. It involves the identification of a specific problem area, the acquisition and encoding of the knowledge using the selected language, evaluation and maintenance of the intelligent system. According to [3–5] this process can be divided into the following phases:

1. Problem identification and conceptualization,
2. Knowledge acquisition and categorization,
3. Knowledge modeling including knowledge base design,
4. Selection and application of inference mechanisms,
5. Knowledge evaluation, verification and validation,
6. System maintenance including, knowledge interoperability issues.

Each phase is dependent on the previous one. However, in practice they usually overlap considerably. Moreover, the process is highly iterative, where each iteration allows for a more precise refinement of the system model. It is common to visualize dependencies between these phases in a waterfall like manner, similar to the classic software life cycle model in *Software Engineering* (SE) [6].

In this book we are mostly concerned with selected tasks from phases 4 to 6. This chapter introduces some of these topics with respect to RBS in more detail. In Sect. 2.1 we discuss the modeling of acquired knowledge with the use of rules and other representation supporting the design. Two main groups of participants take part in these activities. The first one consists of *Domain Experts* who possess the requisite knowledge for problem solving in some specific domain. The second one consists of *knowledge engineers*, i.e. persons capable of designing, building and testing a RBS

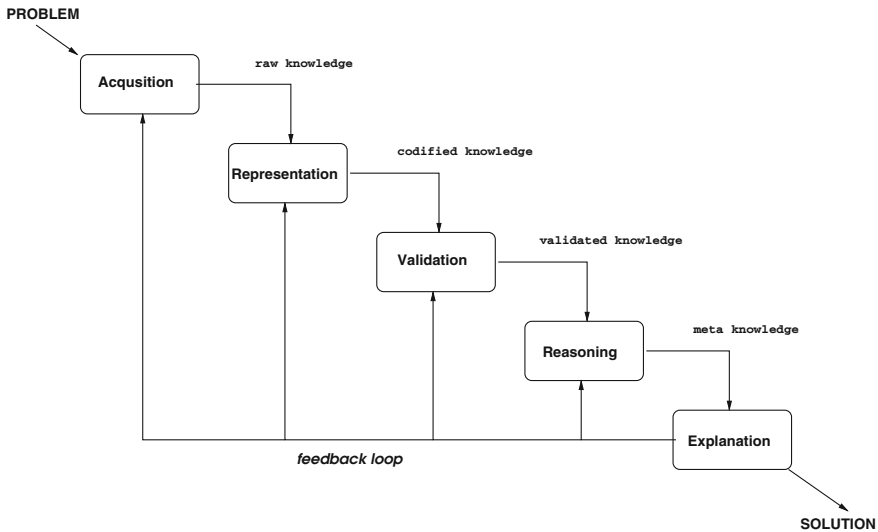


Fig. 2.1 Knowledge engineering process

using the KE methods. Once a rule set is built, an inference mechanism should be considered, see Sect. 2.2. As we discuss in Sect. 2.3, in the case of large rule sets their structure, including relations between rules have to be considered. The quality of the rule base should be analyzed during the design, or at least after it. Selected methods for this task are mentioned in Sect. 2.4. If possible, the rule base should be kept independent of the specific system implementation. To make this possible a specific rule interchange method can be used, see Sect. 2.5. Finally, as today RBS are not usually stand-alone systems, some specific architectures for their integration can be used, see Sect. 2.6. The chapter is summarized in Sect. 2.7 (Fig. 2.1).

2.1 Knowledge Acquisition and Rule Base Modeling

Challenges in Acquiring Knowledge

The knowledge engineering process includes the acquisition of new knowledge from different sources e.g. mainly from domain experts, but also different knowledge bases, available examples, etc. Knowledge acquisition is a tedious task. In fact, it is often referred to as the *Knowledge Acquisition Bottleneck*, described in several classic books e.g. one by Hayes-Roth [7, 8]. In a straightforward, if somewhat naive form, the process consists in extraction, refinement, transfer and input of the knowledge to the appropriate knowledge base. Knowledge acquisition usually starts with reviewing documents and reading books, papers and manuals related to the problem domain, as well as reviewing the old cases. Very often the knowledge is also retrieved from a domain expert, selected during the previous phase, who takes part in the guided

interviews with predefined schemas [9]. During such interviews, an expert answers a series of questions that aim to elicit knowledge. Moreover, in some cases a certain domain specific methodology can also be applied. Subsequently the acquired knowledge is studied and analyzed. Then the entire process is repeated again until no new knowledge is collected. Clearly, this is an inherently iterative process.

There are a number of problems with practical knowledge acquisition:

- *syntactic mismatch* – human expert knowledge does not easily fit into any formal language; in most cases it goes beyond the expressive power of formal languages,
- *semantic mismatch* – there is no way to represent the semantics of the human expert knowledge within the only syntactically encoded knowledge base,
- *contextual and hidden knowledge* – most of inter-human communication rely on assuming some common ontology, contextual knowledge and implicit knowledge, mostly unavailable for computers, and often tacit,
- *common sense knowledge gap* – it is challenging to express all common sense human knowledge,
- *knowledge verbalization* – human experts have difficulties with smart verbalization of knowledge,
- *knowledge vagueness* – knowledge is often uncertain, imprecise, incomplete, etc.

There are many techniques supporting knowledge acquisition, including: guided interviews, i.e. following some predefined schemas, or observations, monitoring and storing expert decisions in an automated or semi-automated way. A number of classic books have been published in this area, for example [10] provides an in depth discussion. Van Harmelen provides a working recipe to start with an analysis of several representative case studies [11]. Besides the main KE approach, the use of machine learning is commonly advised for acquiring knowledge from data. This includes rule induction from databases of sets of case-bases, learning from examples, and rule generation. They allow knowledge to be generated according to the provided model of the system or according to the processes performed within the system. Today, the automated acquisition of knowledge by the machine learning approach is still an active area of research in AI [12, 13]. However, these methods are outside of the scope of this book. As today KBS are mainly software based, the knowledge acquisition phase is very often performed using SE techniques related to requirements analysis.

Knowledge Modeling

Modeling of knowledge is an evolutionary process. During the KE process new knowledge is collected and added to the knowledge base. Thus functionality of the KBS system is improved and it gradually evolves into a final version. The problems that may occur during this phase are mainly related to the vagueness of the knowledge, as it can also be imprecise or incomplete. Other problems, which can occur during this phase, include syntax errors that may appear when the knowledge is encoded into a specific language. It is also possible that some modeling errors may be caused by the complexity of the knowledge model. Due to the large amount of rules or complex dependencies, the modeled knowledge may not reflect the acquired knowledge in

an appropriate way. Therefore, in order to make the modeling process more efficient a number of visual methods have been developed [14]. The visual (or semi-visual) languages facilitate the modeling phase making it more transparent for the knowledge engineer [15]. Therefore, the selection of an appropriate modeling technique can make this phase more efficient and improve the maintainability of the resulting KB.

As the number of rules identified in the system is increasing, it may be difficult to model and manage them. Thus, in complex systems having rule sets consisting of thousands of rules, various forms of rule set representations are used. Such forms as tables or trees are logically equivalent to a set of rules, but they are easier to understand and maintain. Therefore, below we discuss how these two knowledge representations can be used during the modeling process to represent the rule base.

Decision Tables for Rule Modeling

Decision tables are used to group sets of rules that are similar with respect to a set of formulas present in the preconditions and conclusions or actions [16]. Here, we start with the most basic logical form of a propositional rule (Horn clause [17]) which can be expressed as follows:

$$\text{rule: } p_1 \wedge p_2 \wedge \cdots \wedge p_n \rightarrow h$$

The canonical set of rules is one that satisfies the following assumptions [16]:

- all rules use the same propositional symbols in the same order, and
- the rules differ only with respect to using the negation symbol before the propositional symbol.

A complete canonical set of rules is a set containing all the possible combinations of using the negation sign, and it can be expressed as follows (# means either nothing or the negation symbol) [16]:

$$\begin{aligned} \text{rule}_1 : \#p_1 \wedge \#p_2 \wedge \cdots \wedge \#p_n &\longrightarrow \#h_1 \\ \text{rule}_2 : \#p_1 \wedge \#p_2 \wedge \cdots \wedge \#p_n &\longrightarrow \#h_2 \\ &\vdots \\ \text{rule}_m : \#p_1 \wedge \#p_2 \wedge \cdots \wedge \#p_n &\longrightarrow \#h_m \end{aligned}$$

A set of rules which is not in the canonical form, can always be transformed into an equivalent canonical set. Typically, such canonical sets of rules are used for creating decision tables.

The binary decision table corresponding to the above schema is presented in Table 2.1, where *vs* and *ws* are values of conditional and decision attributes. In the presented table, each rule is specified in a single row, in which the first *n* columns specify the conditions under which a specific conclusion is fulfilled (e.g. specific actions can be executed or some conclusion statements can be inferred). As conditions are limited to binary logic, this is often too limited for knowledge encoding.

To enhance the expressive power and knowledge representation capabilities Attributive Logic can be used, see [18] for Attribute-Value Pair Table (AV-Pair Table)

Table 2.1 Decision table

p_1	p_2	\dots	p_n	h
v_{11}	v_{12}	\dots	v_{1n}	w_1
v_{21}	v_{22}	\dots	v_{2n}	w_2
\vdots	\vdots	\vdots	\vdots	\vdots
v_{k1}	v_{k2}	\dots	v_{kn}	w_k

Table 2.2 Attributive decision table

Rule	p_1	p_2	\dots	p_j	\dots	p_n	h_1	h_2	\dots	h_m
r_1	v_{11}	v_{12}	\dots	v_{1j}	\dots	v_{1n}	w_{11}	w_{12}	\dots	w_{1m}
r_2	v_{21}	v_{22}	\dots	v_{2j}	\dots	v_{2n}	w_{21}	w_{22}	\dots	w_{2m}
\vdots	\vdots	\vdots		\vdots		\vdots	\vdots	\vdots		\vdots
r_i	v_{i1}	v_{i2}	\dots	v_{ij}	\dots	v_{in}	w_{i1}	w_{i2}	\dots	w_{im}
\vdots	\vdots	\vdots		\vdots		\vdots	\vdots	\vdots		\vdots
r_k	v_{k1}	v_{k2}	\dots	v_{kj}	\dots	v_{kn}	w_{k1}	w_{k2}	\dots	w_{km}

or [16] for Attributive Decision Table (AD-Table). An example of such a decision table is shown in Table 2.2. A row of an AD-Table represents a rule, expressed as follows:

$$r_i : (p_1 = v_{i1}) \wedge (p_2 = v_{i2}) \wedge \dots \wedge (p_n = v_{in}) \rightarrow h_1 = w_{i1} \wedge h_2 = w_{i2} \wedge \dots \wedge h_m = w_{im}$$

The conditions of such a rule can take several values from a specified domain. Moreover, this approach can be extended in order to allow for specifying an attribute value as an interval or a subset of the domain [16]. An example of such a table determining a rented car category¹ based upon the driver's age and driving license holding period is presented in Table 2.3. A second rule of this table corresponds to the following rule:

$$r_2 : (driver_age < 21) \wedge (driving_license_hp < 2) \rightarrow rented_car_cat = A$$

More extensions to such decision tables, e.g. allowing for specifying an attribute value as an interval or a subset of the domain, can be found in [16]. Clearly the tabular knowledge representation has a very intuitive interpretation. Furthermore, through the examination of the table it is possible to obtain a better understanding of the contents of the rule base. It also allows for the identification of potential problems,

¹The rented car category corresponds to Euro Car Segment classification, see: http://en.wikipedia.org/wiki/Euro_Car_Segment.

Table 2.3 Example of decision table

Driver age	Driving license holding period	Rented car category
<18	Any	None
<21	<2	A
<21	>=2	{ A, B }
>=21	<2	{ A, B, C }
>=21	>=2	{ A, B, C, D }

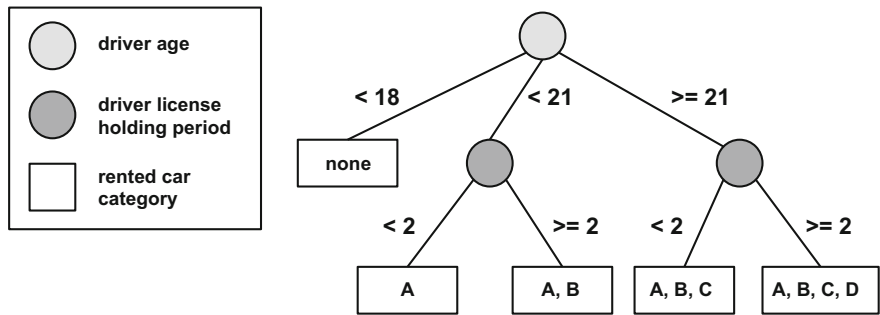


Fig. 2.2 Example of a decision tree corresponding to the decision table

such as missing values. In a way, decision tables allow for structuring rule bases. On the other hand, decision trees are useful for modeling how the knowledge is actually used in the inference process.²

Decision Trees for Rule Modeling

Decision trees allow for organizing rules in a hierarchical manner. As they show the dependencies between conditions and decisions, this clarifies the thinking about the consequences of certain decisions being made [19]. A decision tree has a flowchart-like structure in which a node represents an attribute and branches from such a node represent the attribute values. The end nodes (leaves) represent the final decision values. An example of a decision tree corresponding to the decision table from Table 2.3 is presented in Fig. 2.2.

Trees, especially Binary Decision Trees (BDTs), are very popular visual decision structures. They are used both in computer science and in other disciplines [16]. BDT consists of simple tests. Each test is an evaluation of a propositional statement which can be either true or false. Thus, a BDT leads through a series of tests to arrive eventually at some decision. Such a form of knowledge representation allows for the clear presentation of the decision process. Unfortunately, decision trees become much more complex if each attribute has a large number of different values because of the redundancy of nodes.

²Furthermore, they are useful to be integrated with user dialogs.

Both trees and tables are also useful from a practical point of view, as they provide a visual representation of the rule base. In this book, we assume after [16] that the transformation between rules, tables, and trees is always possible (having some syntactic restrictions). However, it is not always straightforward. These visual models help during the design process. After it they are translated to a set of rules that can then be processed using some of the common techniques described below. Similar to the acquisition phase, today a number of SE techniques are used in rule modeling. While we do not mention them here, they will be briefly described in Sect. 3.5.

2.2 Automated Inference with Rules

In rule-based systems the inference process requires two main elements. The first is the knowledge base, i.e. a rule base encoded in the format appropriate for the inference engine. The second is the inference engine itself. It uses specific inference algorithms to analyze the contents of the rules base, identify rules that can be fired, and fires them. It is generally assumed that the engine and algorithm are independent from the encoded knowledge and allow for the processing of knowledge from any domain. Important aspects that determine the operations of the inference engine include the inference mode and tasks.

The *inference mode* defines how the knowledge contained in the rule base is processed. In RBS two main inference modes are used [20]:

- *Forward chaining* is the *data-driven* (or bottom-up) reasoning. This mode of reasoning starts from the existing knowledge stored as facts and continues until no further conclusions can be drawn. The engine checks the preconditions (LHS) of rules to see which ones can be fired. Then actions in RHS are executed. Therefore, the main drawback of this inference mode is that many rules may be executed that have nothing to do with the established goal.
- *Backward chaining* is a reverse process to forward chaining and is called *goal-driven* reasoning. In this mode the system has a goal (a hypothetical solution) and the inference engine attempts to find the evidence to prove it with the help of the facts stored within the fact base. Inference engine processes rules in a backward manner (in comparison to the previous mode) i.e. from RHS to LHS, and search for such rules that RHS matches to the desired goal. If LHS of a certain rule can be proven to be true in terms of the facts then the goal is also proven. If not, then the elements of LHS go to the list of goals and the entire process repeats.

In fact, sometimes a *mixed* mode allows for the combination of forward and backward inference modes.

Both of the inference modes can be applied to different kinds of problems. However, according to [21] forward chaining is a natural way to design expert systems for analysis and interpretation. For example, DENDRAL, an expert system for determining the molecular structure of unknown soil, used forward chaining [22]. In turn, the backward chaining expert systems are mostly used for diagnostic purposes. For

instance, MYCIN, a medical expert system for diagnosing infectious blood diseases, used backward chaining [23].

An *inference task* is a scenario of using rules that is performed by an inference engine working in a given inference mode. Thus, it is important to distinguish between inference modes, like forward and backward chaining, and inference tasks. Therefore, a given inference task can be performed in different inference modes. One can introduce several inference tasks, such as:

- The *final consequence* inference task determines the evaluation of a given set of rules in order to infer all possible conclusions based upon the existing facts and the facts drawn during inference. Performing this task, the inference engine must take the changes of the fact base (state) into account and reevaluate rules that can be affected by these changes [21].
- The *single-pass consequence* inference task is similar to final consequence, but in contrast to it, the inference engine does not track changes of the system's state. This ensures that the rules get evaluated against existing facts only one time and then prevent them from being reevaluated.
- The *specific-pass consequence* inference task is, in turn, similar to a single-pass consequence, but additionally it explicitly defines the order of rules in which they must be executed.
- The *consequence reduction* inference task forces an inference engine to answer a question if a given hypothesis can be proved to be true according to existing facts. An inference engine tries to find such a sequence of rules that allows for the expression of the hypothesis by means of the existing facts.

Now let us consider the most important inference algorithms.

A typical forward chaining inference process performed in RBS is an iterative process consisting of the steps described below.

1. *Match* – search for all rules having satisfied their LHS.
2. *Conflict set resolution* – selection of rule for executing.
3. *Action* – invoking actions from the consequent part of the selected rule.
4. *Return* – performing the next iteration (return to the step 1).

The same process for backward chaining is analogous but it differs in direction of rule processing [21]. Among these four steps, the first step is the bottleneck of inference because it requires to match facts stored within the fact base to rules in order to check if a given rule has satisfied their LHS or not. The implementation of the naive algorithm involves iteration over all matching of the all combinations of facts to all rules. Nevertheless, such an approach is inefficient and thus other algorithms have been developed.

An important and now commonly used inference algorithm is RETE [24]. This algorithm allows the naive approach to be avoided and makes the *match* step much more efficient. It is based upon the two major ideas. The first one is related to knowledge compilation where each knowledge base is compiled and the set of all rules is transformed into so-called *discrimination network* that represents all the rules in the form of directed and acyclic graph. The second idea is to store the information

concerning facts satisfying a certain condition within a corresponding node. As a result of this, operations performed during this step are limited to monitoring only changes (adding or removing) made in the fact base. When such a change is observed, it is passed through the network in order to identify rules having satisfied their LHS.

The idea of a discrimination network can also be found in other inference algorithms like TREAT [25] or GATOR [26]. In comparison to RETE, TREAT provides a more efficient memory management because it does not store any redundant information. In turn, GATOR is the most general one, as it uses several optimization methods for building a discrimination network. In particular cases they may take the forms of networks used by RETE or TREAT. Therefore, both of them are considered to be special cases of GATOR. More details on the comparisons of these algorithms can be found in [27–29].

2.3 Structure of Rule Bases

A basic discussion of rule-based systems is most commonly focused on building single rules, or constructing relatively small sets of rules. This is justified in simple cases, or studies of rule extraction algorithms. However, in engineering practice the size, structure, properties, and quality of such rule sets are very important. In fact, larger rule sets should be more commonly referred to as “rule bases” as rules in such a set are most often interrelated.

Structure on the Design Level

Relations of rules in a rule base can be expressed explicitly in rules. Examples include rules for decision control, where the execution of certain rules explicitly calls other rules. Moreover, there are cases of rewriting systems, where some rules can be modified by others. Furthermore, rule sets can be explicitly partitioned into groups operating in or in relation to given situations. There might be implicit relations between rules. Probably the most common and important cases are rules that share the same attributes. Even if such rules are not grouped together they could be related to similar situations, or objects. What makes such cases even more complicated are logical relations between such rules. This can result in contradicting or excluding rules which can lead to an unexpected operation of the system. There are different solutions to address these issues.

A simple solution, common in rule-based shells is the introduction of modularization of the rule base. CLIPS offers functionality for organizing rules into so-called modules. They allow for the restriction of access to their elements from other modules, and can be compared to global and local scoping in programming languages. In CLIPS each module has its own pattern-matching network for rules and its own agenda (rules to be fired). Jess provides a similar module mechanism that helps to manage large rule bases. Modules provide a control mechanism: rules in a module will fire only when this module has the focus, and only one module can have focus at a time. In general, although any Jess rule can be activated at any time, only rules

in a module having focus will fire. Although this is a step towards introducing the structure of the rule base, still all rules are checked against the facts. In certain cases this mechanism can improve management of the rule base, as the large set of rules can be partitioned into smaller ones. It can also have a positive impact upon the performance of the inference process as not all of the rules need to be analyzed. A similar approach was employed in the Drools system. However, as Drools moved away from a CLIPS-like inference in large rule bases to a dedicated process engine, it will be described in the subsequent section on inference.

Another approach is to introduce structure into the model of the knowledge base during the design. Using visual representation methods such as decision tables can simplify the grouping of rules sharing the same attributes [30]. A decision tree can also be used to represent a group of rules but emphasizing the inference process. There exist hybrid representations such as XTT2 (eXtended Tabular Trees) that combine tables with trees [31]. While tables group rules with the same attributes, a high level inference network allows the inference process to be controlled. The tables can be connected during the design process to denote relations between groups of rules. These connections can be further used by the inference engine to optimize the inference process. For more details see [29]. In [32] a complete design and integration approach for formalized rule-based systems was introduced. It is called *Semantic Knowledge Engineering* (SKE) as it put emphasis on the proper interpretation of rule based knowledge as well as on its integration with other software engineering paradigms. In the subsequent parts of this chapter we will briefly discuss how inference control and integration are handled in SKE.

Improving Inference in Structured Rule Bases

The structuring of the rule base can be reflected during the inference process. CLIPS modules allow for the restriction of access to their elements from other modules, and can be compared to global and local scoping in other programming languages. The modularization of the knowledge base helps managing rules, and improves the efficiency of rule-based system execution. In CLIPS each module has its own pattern-matching network for its rules and its own agenda. When a *run* command is given, the agenda of the module which is the current focus is executed. Rule execution continues until another module becomes the current focus, no rules are left on the agenda, or the return function is used from the RHS of a rule. Whenever a module that was focused on runs out of rules on its agenda, the current focus is removed from the focus stack and the next module on the focus stack becomes the current focus. Before a rule executes, the current module is changed to the module in which the executing rule is defined (the current focus). The current focus can be dynamically switched in RHS of the rule with a *focus* command. A similar mechanism is present in Jess.

The Drools platform introduced RuleFlow tool. It is a workflow and process engine that allows for the advanced integration of processes and rules. It provides a graphical interface for processes and rules modeling. Drools have a built-in functionality to define the structure of the rule base which can determine the order of the rules evaluation and execution. Rules can be grouped into a ruleflow-groups which defines

the subset of rules that are evaluated and executed. The ruleflow-groups have a graphical representation as the nodes on the *ruleflow* diagram. The ruleflow-groups are connected with the links what determines the order of its evaluation. A *ruleflow* diagram is a graphical description of a sequence of steps that the rule engine needs to take, where the order is important.

More recently, Drools 5 moved from a dedicated flow control engine into the integration of a rule-based reasoning system with a complete *Business Process Management Systems* (BPMS). In this case rule-based modules or subsystems can be called arbitrarily by a high-level flow control mechanism. In this case it is a Business Process engine jBPM. This approach to controlling the rule-based inference will be described in the section regarding integration of RBS with other systems.

2.4 Knowledge Base Analysis

Knowledge-based systems are widely used in areas where high performance and reliability are important. In some cases a failure of a such system may have serious consequences. Therefore, it is crucial to ensure that the system will work correctly in every possible situation.

Verification and validation have been discussed by many authors, including [33–42]. Verification concerns proving correctness of the set of rules in terms of some verifiable characteristics [43]. In fact features such as consistency, completeness, and various features of correctness may be efficiently checked only by using formal methods [15]. In turn, validation is related to checking if the system provides correct answers to specific inputs. In other words, validation consists in assuring that the system is sound and fits user requirements. Boehm [43] characterized the difference as follows: “Verification is building the system right. Validation is building the right system.”

Here, it is assumed that:

- *Verification* is a process aimed at checking if the system meets its constraints and requirements (after Tepandi [41], Andert [33], and Nazareth [35]).
- *Testing* is a process which seeks to analyze system operation by comparing system responses to known responses for special input data (after Tepandi [41]).
- *Validation* is a case of testing, aimed at checking if the system meets user requirements (after Tepandi [41]).

A comprehensive overview of this field has been presented by Vermesan in [44]. Moreover, a summary result of analysis techniques and tools is presented in an online report by Wentworth et al. [45].

According to [46] verification and validation procedures of the system can be understood as one of the following: anomaly detection, formal verification, parallel use, rule base visualization to aid review, code review, testing. That study shows, that verification of the rule based systems is dominated by testing and code review. This

approach highly depends on human skills, since an incorrectly written test may produce the wrong results. Formal verification and anomaly detection are not so widely used despite the fact that these methods usually have strong logical foundations and in most cases exceed the testing and debugging approach. These methods are able to check the following characteristics of the rule base [47]:

- *Redundancy* – is knowledge specified in an efficient way, so that redundancy is avoided, it includes identical rules, subsumed rules, equivalent rules, unusable rules (those that are never fired).
- *Inconsistency* – is the knowledge specified within a rule base consistent, where both internal, and external logical consistency, referring to the consistency of the real world (model) can be considered, it includes ambiguous rules, conflicting, ambivalent rules, logical inconsistency.
- *Minimal representation* – is specified knowledge in some minimal form, i.e. it cannot be replaced by another, more efficient and more concise representation: it includes reduction of rules, canonical reduction of rules, specific reduction of rules, the elimination of unnecessary attributes.
- *Completeness* – is knowledge complete, can the system always produce the output, it includes logical completeness, specific (physical) completeness, detection of incompleteness, identification of missing rules.
- *Determinism* – is provided output unique and repeatable under the same input conditions, it includes pairwise rule determinism, and complete system determinism,
- *Optimality* – is knowledge sufficient for producing the best available solutions. It includes optimal form of individual rules, locally optimal solutions, optimal knowledge covering, and optimal solutions.

According to a comparison of existing verification tools in [48] one can draw a conclusion, that the main reason why formal verification is not widely used among expert system developers is that it requires *formal knowledge representation*. In fact, most of these tools are usually based upon propositional or predicate logic. MELODIA [49] used propositional logic and a flat rule base. CLINT [50], COVER [51], PREPARE [52], SACCO [53] used predicate logic and a flat rule base. Moreover, IN-DEPTH [39] introduces a hierarchical representation, COVADIS [54] used simple production rules language with a flat rule base, whereas KRUST [55] used frames.

However, common expert system shells such as CLIPS, JESS or DROOLS do not provide formal knowledge representation, so it is not possible to apply formal methods to these tools. Although there are some analysis tools that are dedicated to aforementioned shells like CRSV-CLIPS [48, 56] for CLIPS, DROOLS Verifier, their aim is not to provide formal verification, but to offer a framework for writing tests.

Verification of knowledge in RBS is typically considered the last stage of the design procedure [5]. It is assumed that it will be performed on a complete, specified knowledge base; see the important works of Preece and Vermesan [44, 57, 58]. As such it is costly and difficult. In this book we advocate approaches that introduce a formalized description of rule base. As a result of these, formal verification of the rule base is possible during its design.

Once designed and analyzed the rule base needs to be maintained. An important and challenging aspect of maintenance includes the use of the same rule base in a different system. This involves the possible re-coding of rules in another rule language and it might be a challenging process. As it is one of the areas addressed in this book, we will analyze it in much more detail in the following section.

2.5 Rule Interchange and Interoperability

Having a complete, possibly verified and validated system, it is desirable to ensure that there is a method for sharing knowledge with other systems, representations and tools. Such methods bring many significant advantages especially for knowledge maintenance that becomes easier and more efficient. Some of them may be summarized in the following way:

- *Rapid development* – as a result of efficient interoperability methods, a new knowledge base may be supplemented by already existing knowledge that can even be modeled in different representation.
- *Cross-representation knowledge engineering* – knowledge can be modeled and verified by tools dedicated for different representation. This allows for the wider reuse of already existing tools and prevents from the unnecessary development of new ones.
- *Heterogeneous knowledge bases* – an intelligent system may use different sources/pieces of knowledge that are translated into required representation on the fly. This creates a great opportunity for collaborative methods for knowledge engineering.

In the classic approach, a RBS was considered as standalone i.e. the implemented system was self-contained and was usually separated from other elements in its environment. Today, the rule-based knowledge representation found application in several areas that are oriented towards collaboration and integration with external technologies. Together with the increasing number of rules application areas, the number of different rule representations is also growing. The differences between these representations ensure that rule-based knowledge cannot be easily shared among different rule bases. Usually, the naive translation methods do not take rule base semantics into account which leads to a semantic mismatch before and after translation. This persistent problem is called *rule interoperability problem* and it has been known since classic expert systems [21].

In general, the methodology of interoperability must take two aspects into account: syntax that is used for knowledge encoding and semantics. On each of these two levels some problems can be identified, including ambiguous semantics, different expressiveness, and syntactic power. Over time, many different methods and approaches to the knowledge interoperability problem were developed. Some of them are general-purpose, i.e. aim to provide a framework for translation between many different representations. Others are dedicated for a certain set of representations that share similar assumptions and thus have similar semantics. The most important of these are described below.

KIF

Knowledge Interchange Framework (KIF) [59] constitutes one of the first implementations of the formal knowledge interoperability approach that uses unified intermediate representation providing declarative semantics. KIF was intended to be a formal language for the translation of knowledge among disparate computer programs providing the possibility of precise definition of semantics. It was not limited only to rules but also supports other representation techniques like frames, graphs, natural language, etc. It is important to note that KIF was not intended as a primary language for interaction with human users (though it can be used for it). Different programs could interact with their users in whatever forms that are most appropriate to their applications. The formal definition (specification) of KIF is complicated. It provides very complex meta-model consisting of a large number of classes. Moreover, its complexity led to very weak tool support and currently there are few tools that support KIF even partially. In the past, KIF was used in the context of classic expert systems. A new implementation of KIF appeared as one of the dialects of CL called *Common Logic Interchange Format* (CLIF) [60].

RIF

Rule Interchange Format (RIF) [61, 62] is a result of research conducted by Rule Interchange Format Working Group. This group was established by the World Wide Web Consortium (W3C) in 2005 to create a standard for exchanging rules among rule systems, in particular among web rule engines. Although originally envisioned by many as a *rule layer* for the Semantic Web, in reality the design of RIF is based upon the observation that there are many rule languages in existence, and what it is important to exchange rules between them. The approach taken by the group was to design a family of languages, called dialects with rigorously specified syntax and semantics of different rule systems. Currently RIF is part of the infrastructure for the Semantic Web, along with SPARQL, RDF and OWL.

Unlike CL, RIF dialects do not share the same expressiveness and semantics. Nevertheless, it is assumed that all of them include RIF-Core dialect [63] which defines an XML syntax for definite Horn rules without function symbols, i.e. DATALOG, with standard first-order semantics. Until now, the development of the RIF framework was focused on two kinds of dialects: (1) Logic-based dialects – *Basic Logic Dialect* (RIF-BLD) [64] and a subset of the RIF *Core Dialect*, they include languages that employ some kind of logic, such as FOL (often restricted to Horn logic) or non-FOL underlying the various logic programming languages; (2) Dialects for rules with actions – the *Production Rule Dialect* (RIF-PRD) [65]: it includes production rule systems, such as JESS, DROOLS and JRules, as well as reactive (or event-condition-action) rules, such as Reaction RULEML and XChange. Unfortunately, the RIF specification provided by the W3C working group has several limitations and deficiencies that makes RIF application difficult.³

³Lukichev shows that RIF does not specify guidelines regarding how to implement a transformation from a source rule language into the RIF and, what is more, how to verify the correctness of already

PRR

Production Rule Representation (PRR) [68] is an OMG standard for production rule representation that was developed as a response for Request For Proposals from 2002–2003 to address the need for a representation of production rules in UML models (i.e. business rule modeling as part of a modeling process). It adopts the rule classification scheme (supplied by the RULEML Initiative) and supports only production ones. It provides the MOF-based meta-model and profile that are composed of a core structure referred to as PRR Core and a non-normative abstract OCL-based syntax for the expressions, defined as an extended PRR Core meta-model referred to as PRR OCL [68]. PRR Core is a set of classes that enable the production of rules and rule sets to be defined in a purely platform independent way without having to specify OCL, in order to represent conditions and actions. As conditions and actions are “opaque” and simply strings, it is suitable to support rule expressions in both formal and informal way in order to be useful also for people without KE skills. This expressiveness was reached by very general definition of production rule semantics. This representation defines rule as the dynamic elements that allows for the chaining state of the rule-based systems and provides operational semantics for rules, the forward-chaining of production rules and rule sets.

RuleML

Rule Markup Language (RULEML) [69, 70] is defined by the RULEML Initiative.⁴ This initiative aims to develop an open, vendor neutral XML/RDF-based rule language allowing for the exchange of rules between various systems including: distributed software components on the web, heterogeneous client-server systems found within large corporations, etc. RULEML is intended to be used in Semantic Web and this is why it offers XML-based language syntax for rules. In turn, the abstract syntax of this language is specified by means of a version of Extended BNF, very similar to EBNF notation used for XML. The foundation for the kernel of RULEML is the DATALOG (constructor-function-free) sub-language of Horn logic. Its expressiveness allows for the expression of both forward and backward-chaining rules in XML. It also supports different kinds of rules: derivation rules, transformation rules, reaction rules and production rules. The formal model of RULEML is comprehensively described in [71]. RULEML, as a general rule interoperability format, can be customized for various semantics of underlying rule languages that should be represented and translated. Although specific default semantics are always predefined for each RULEML language, the intended semantics of a rule base can override it by using explicit values for corresponding semantic attributes.

(Footnote 3 continued)

made translations [66]. In turn, in [67] Wang et al. indicate that, from the perspective of modeling, RIF does not have meta-model to describe features of rules.

⁴See: <http://wiki.ruleml.org>.

R2ML

REVERSE Rule Markup Language R2ML [72] was developed by the *REVERSE Project Working Group II*.⁵ It is an interoperability format for rules integrating RULEML with the SWRL as well as the OCL. The main goal of R2ML is to provide a method for: rule translation between different systems and tools; enriching ontologies by rules; connecting different rule systems with R2ML-based tools for visualization, verbalization, verification and validation. It supports four rule categories: derivation rules, production rules, integrity rules and ECA/reaction rules. The concepts that are used within rules can be defined in MOF/UML. R2ML was modeled using Model-Driven Approach (MDA) and it is required to accommodate Semantic Web by: web naming concepts, such as URIs and XML namespaces; the ontological distinction between objects and data values; the datatype concepts of RDF and user-defined datatypes. On the other hand, R2ML can be used as a concrete XML syntax for Extended RDF, which extends the RDF(S) semantics. Similarly to RIF, R2ML is based upon partial logic [73]. However, R2ML is supposed to be used as an intermediate representation for rules and not as a reasoning formalism. Thus, there is no efficient tool support for R2ML.

2.6 Architectures for Integration

Historically, rule-based systems were considered as stand alone tools. This meant such a system was an independent software component (sometimes integrated into hardware system). As such, it was fully responsible for processing input data, performing processing, and then making the appropriate decision and ultimately producing output data, or carrying out control actions. Therefore, with time in classic RBS systems such as CLIPS, a number of additional libraries were created to support such an environment. However, today such an approach seems redundant, and is rather rare. RBS are considered as software components, that have to be integrated into a larger software environment using well-defined software engineering approaches [6]. Therefore, here we provide a short account of different architectures to integrate rule-based systems into a larger software environment.

The already mentioned classic approach with standalone systems can be considered a *homogeneous* one. As in such a case the RBS should be able to provide not just the decision making, but also a vital part of the interfaces on the software runtime level. An important aspect is in fact related to the rule language level. In this case, the rule language should be powerful enough to program all of these features, as it is the only language available for the system designer. This results in the design of expressive rule languages like in the case of CLIPS with additional programming libraries, or language extensions such as COOL [74].

An alternative approach is to restrict the role of the RBS only to decision making. In this case, the remaining functionality is delegated to other systems or components.

⁵See: <http://oxygen.informatik.tu-cottbus.de/reverse-ii/>.

The RBS only need to possess interfaces allowing for lower-level integration. It also operates as intelligent middleware, not a stand-alone system. This kind of architecture can be described as *heterogeneous*.

Heterogeneous Integration Using MVC Design Pattern

The rule-based component can be then integrated with into larger software system using some of the common software design patterns [75]. An example of such an approach was previously proposed in [32]. It is related to bridging knowledge engineering with software engineering (SE). Today software engineering faces a number of challenges related to efficient and integrated design and implementation of complex systems. Historically, when systems became more complex, the engineering process became more and more declarative in order to model the systems in a more comprehensive way. It made the design stage independent of programming languages, which resulted in a number of approaches. One of the best examples is the MDA (Model-Driven Architecture) approach [76]. Since there is no direct “bridge” between declarative design and sequential implementation, a substantial amount of work is needed to turn a design into a running application. This problem is often referred to as a *semantic gap* between a design and its implementation as has been discussed by Mellor in [77]. It is worth noting that while the conceptual design can sometimes be partially formally analyzed, the full formal analysis is impossible in most cases. However, there is no way of assuring that even a fully formally correct model would translate into a correct code into a programming language. Moreover, if an application is automatically generated from a designed conceptual model then any changes in the generated code have to be synchronized with the design. Another issue is the common lack of separation between core software logic, interfaces, and presentation layers.

Some of the methodologies e.g. the MDA, and the design approaches e.g. the MVC (Model-View-Controller) [78] try to address this issue. The main goal is to avoid semantic gaps, mainly the gap between the design and implementation. In order to do so, the following elements should be developed: a rich and expressive design method, a high-level runtime environment, and an effective design process. Methodologies which embody all of these elements should eventually shorten the development time, improve software quality, and transform the “implementation” into the runtime-integration and introduce so-called “executable design”.

Using these ideas the *heterogeneous integration* of a RBS may be considered on several levels:

- *Runtime level*: the application is composed of the rule-based model run by the inference engine integrated into the external interfaces.
- *Service level*: the rule-based core is exposed to external applications using a network-based protocol. This allows for an SOA (Service-Oriented Architecture)-like integration [79] where the rule-based logic is designed and deployed using an embedded inference engine.
- *Design level*: integration considers a scenario, where the application has a clearly identified logic-related part, which is designed using a visual design method for

rules (such as decision table, or decision trees), and then translated into a domain-specific representation.

- *Rule language level*: in this case rule expressions can be mixed with another programming language (most often Java). In this case both syntax and semantics is mixed. However, this allows for an easy integration of rule-based code to be easily integrated into the rich features of another programming environment (e.g. Java).

An example of integration on the first three levels will be given in Sect. 9.3.

Integration of Rules and Business Process System

A specific case of heterogeneous integration of RBS with BPMS can also be considered. The Drools 5 platform encompasses several integrated modules including *Drools Expert* and *jBPM*. The former is a business rules execution engine. The latter is a fully-fledged BP execution engine. It executes business process models encoded in Business Process Model and Notation (BPMN) [80]. This notation includes dedicated syntactic constructs, so-called rule tasks. As a result of these it is possible to delegate the execution of the details of business process logic to a rule-based system. Practically it can be any system implemented in Drools. However, from the design transparency perspective a reasonable approach is to connect only restricted well-defined subsystems, or even single modules (tables).

Following the previously defined levels, such a scenario for integration is mainly runtime-oriented. While proper design tools are currently not available for Drools, with some extensions this integration can also be reflected on the design level. Preliminary work in this direction was presented in [81], where a web design framework for business processes with rules were presented. The Drools approach also allows for service-level integration, as the whole runtime environment is web-enabled. Drools supports the orchestration of web services using rules. The execution of such solutions is supported by the runtime environment.

We provided new results in the area of integration, including a formalized model for business process models with rules in [82]. This model will be discussed in Chap. 5. It allows for a more complete integration of rules and processes, both during design and execution, as presented in Chap. 5.

2.7 Summary

In this short chapter we provided an overview of important topics related to the development of rule-based systems, commonly simply referred to as knowledge engineering. Following classic textbooks [1, 2] we discussed the general phases of the KE process [3–5]. We emphasized aspects that are non-trivial and are explored later in this book. This includes knowledge base modeling with the use of visual methods. During this phase formalization of the representation methods can be introduced. Moreover, we discussed structuring of the rule base as an important aspect of complex

real-life systems. Furthermore, we identified knowledge interoperability as playing a key role in the long-term maintenance of once acquired and modeled knowledge. Finally we explored opportunities for integration of the RBS with other systems.

In the next chapter we will discuss selected applications of RBS. Thus we will conclude the presentation of the important aspects of the state of the art in RBS needed for the discussion of original results presented later. As today's KBS are mostly software based [83] we try to identify possible bridges with KE and SE. This will provide a foundation for the subsequent parts of the book.

References

1. Guida, G., Tasso, C.: *Design and Development of Knowledge-Based Systems: From Life Cycle to Methodology*. Wiley, New York (1995)
2. Gonzalez, A.J., Dankel, D.D.: *The Engineering of Knowledge-Based Systems: Theory and Practice*. Prentice-Hall Inc, Upper Saddle River (1993)
3. Durkin, J.: *Expert Systems: Design and Development*. Macmillan, New York (1994)
4. Waterman, D.A.: *A Guide to Expert Systems*. Addison-Wesley Longman Publishing Co., Inc, Boston (1985)
5. Liebowitz, J. (ed.): *The Handbook of Applied Expert Systems*. CRC Press, Boca Raton (1998)
6. Sommerville, I.: *Software Engineering*. International Computer Science, 7th edn. Pearson Education Limited, Harlow (2004)
7. Hayes-Roth, F., Waterman, D.A., Lenat, D.B.: *Building Expert Systems*. Addison-Wesley Longman Publishing Co., Inc, Boston (1983)
8. Buchanan, B.G., Shortliffe, E.H. (eds.): *Rule-Based Expert Systems*. Addison-Wesley Publishing Company, Reading (1985)
9. von Halle, B.: *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. Wiley, New York (2001)
10. Scott, A.C.: *A Practical Guide to Knowledge Acquisition*, 1st edn. Addison-Wesley Longman Publishing Co., Inc, Boston (1991)
11. van Harmelen, F., Lifschitz, V., Porter, B. (eds.) *Handbook of Knowledge Representation*. Elsevier Science, Amsterdam (2007)
12. Stefik, M.: *Introduction to Knowledge Systems*. Morgan Kaufmann Publishers, San Francisco (1995)
13. Buchanan, B.G., Wilkins, D.C.: *Readings in Knowledge Acquisition and Learning: Automating the Construction and Improvement of Expert Systems*. M. Kaufmann Publishers, San Francisco (1993)
14. Debenham, J.: *Knowledge Engineering: Unifying Knowledge Base and Database Design*, 1st edn. Springer Publishing Company, Incorporated, Berlin (2012)
15. Nalepa, G.J.: Languages and tools for rule modeling. In: Giurca, A., Gašević, D., Taveter, K. (eds.) *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*, pp. 596–624. IGI Global, Hershey (2009)
16. Ligeza, A.: *Logical Foundations for Rule-Based Systems*. Springer, Berlin (2006)
17. Ben-Ari, M.: *Mathematical Logic for Computer Science*. Springer, London (2001)
18. Ignizio, J.P.: *An Introduction to Model Driven Architecture. Applying MDA to Enterprise Computing*. David S. Frankel, Wiley Publishing, Inc., Indianapolis 2003 *Expert Systems. The Development and Implementation of Rule-Based Expert Systems*. McGraw-Hill, New York (1991)
19. Graham, I.: *Business Rules Management and Service Oriented Architecture*. Wiley, New York (2006)

20. Jackson, P.: Introduction to Expert Systems, 3rd edn. Addison–Wesley, Boston (1999). ISBN 0-201-87686-8
21. Giarratano, J., Riley, G.: Expert Systems. Principles and Programming, 4th edn. Thomson Course Technology, Boston (2005). ISBN 0-534-38447-1
22. Sutherland, G.R.: DENDRAL, a computer program for generating and filtering chemical structures. Report Memo AI-49, Stanford University, Department of Computer Science, Stanford, California (1967)
23. Buchanan, B.G., Shortliffe, E.H.: Rule Based Expert Systems: The Mycin Experiments of the Stanford Heuristic Programming Project. The Addison-Wesley Series in Artificial Intelligence. Addison-Wesley Longman Publishing Co. Inc., Boston (1984)
24. Forgy, C.: Rete: a fast algorithm for the many patterns/many objects match problem. *Artif. Intell.* **19**(1), 17–37 (1982)
25. Miranker, D.P.: TREAT: a better match algorithm for AI production systems; long version. Technical report 87-58, University of Texas (1987)
26. Hanson, E.N., Hasan, M.S.: Gator: an optimized discrimination network for active database rule condition testing. Technical report 93-036, CIS Department University of Florida (1993)
27. Kaczor, K., Nalepa, G.J., Bobek, S.: Rule modularization and inference solutions – a synthetic overview. In: Schneider, A. (ed.) Crossing Borders within ABC. Automation, Biomedical Engineering and Computer Science: 55 IWK Internationales Wissenschaftliches Kolloquium: International Scientific Colloquium, Illmenau, Germany, pp. 555–560 (2010)
28. Bobek, S., Kaczor, K., Nalepa, G.J.: Overview of rule inference algorithms for structured rule bases. *Gdansk Univ. Technol. Fac. ETI Ann.* **18**(8), 57–62 (2010)
29. Nalepa, G., Bobek, S., Ligeza, A., Kaczor, K.: Algorithms for rule inference in modularized rule bases. In: Bassiliades, N., Governatori, G., Paschke, A. (eds.) Rule-Based Reasoning, Programming, and Applications. Lecture Notes in Computer Science, vol. 6826, pp. 305–312. Springer, Berlin (2011)
30. Vanthienen, J., Dries, E., Keppens, J.: Clustering knowledge in tabular knowledge bases. In: *ICTAI*, pp. 88–95 (1996)
31. Nalepa, G.J., Ligeza, A., Kaczor, K.: Formalization and modeling of rules using the XTT2 method. *Int. J. Artif. Intell. Tools* **20**(6), 1107–1125 (2011)
32. Nalepa, G.J.: Semantic Knowledge Engineering. A Rule-Based Approach. Wydawnictwa AGH, Kraków (2011)
33. Andert, E.P.: Integrated knowledge-based system design and validation for solving problems in uncertain environments. *Int. J. Man-Mach. Stud.* **36**(2), 357–373 (1992)
34. Nguyen, T.A., Perkins, W.A., Laffey, T.J., Pecora, D.: Checking an expert systems knowledge base for consistency and completeness. In: *IJCAI*, pp. 375–378 (1985)
35. Nazareth, D.L.: Issues in the verification of knowledge in rule-based systems. *Int. J. Man-Mach. Stud.* **30**(3), 255–271 (1989)
36. Preece, A.D.: Verification, validation, and test of knowledge-based systems. *AI Mag.* **13**(4), 77 (1992)
37. Preece, A.D.: A new approach to detecting missing knowledge in expert system rule bases. *Int. J. Man-Mach. Stud.* **38**(4), 661–688 (1993)
38. Preece, A.D., Shinghal, R.: Foundation and application of knowledge base verification. *Int. J. Intell. Syst.* **9**(8), 249–269 (1994)
39. Meseguer, P.: Incremental verification of rule-based expert systems. In: *Proceedings of the 10th European Conference on Artificial Intelligence. ECAI'92*. Wiley, New York, NY, USA, pp. 840–844 (1992)
40. Suwa, M., Scott, C.A., Shortliffe, E.H.: Completeness and consistency in rule-based expert system. *Rule-Based Expert Systems*, pp. 159–170. Addison-Wesley Publishing Company, Reading (1985)
41. Tepandi, J.: Verification, testing, and validation of rule-based expert systems. In: *Proceedings of the 11-th IFAC World Congress*, pp. 162–167 (1990)

42. Szpyrka, M.: Design and analysis of rule-based systems with adder designer. In: Cotta, C., Reich, S., Schaefer, R., Ligeza, A. (eds.) *Knowledge-Driven Computing: Knowledge Engineering and Intelligent Computations*. Studies in Computational Intelligence, vol. 102, pp. 255–271. Springer, Berlin (2008)
43. Boehm, B.W.: Verifying and validating software requirements and design specifications. *IEEE Softw.* **1**(1), 75–88 (1984)
44. Vermesan, A.I., Coenen, F. (eds.) *Validation and Verification of Knowledge Based Systems. Theory, Tools and Practice*. Kluwer Academic Publisher, Boston (1999)
45. Wentworth, J.A., Knaus, R., Aougab, H.: Verification, Validation and Evaluation of Expert Systems. World Wide Web Electronic Publication. <http://www.tfhr.gov/advanc/vve/cover.htm>
46. Zacharias, V.: Development and verification of rule based systems – a survey of developers. In: *Proceedings of the International Symposium on Rule Representation, Interchange and Reasoning on the Web. RuleML'08*. Springer, Berlin, pp. 6–16 (2008)
47. Ligeza, A., Nalepa, G.J.: Rules verification and validation. In: Giurca, A., Gašević, D., Taveter, K. (eds.) *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*, pp. 273–301. IGI Global, Hershey (2009)
48. Tsai, W.T., Vishnuvajjala, R., Zhang, D.: Verification and validation of knowledge-based systems. *IEEE Trans. Knowl. Data Eng.* **11**, 202–212 (1999)
49. Charles, E., Dubois, O.: Melodia: logical methods for checking knowledge bases. In: Ayel, M., Laurent, J.P. (eds.) *Validation, Verification and Test of Knowledge-Based Systems*, pp. 95–105. Wiley, New York (1991)
50. De Raedt, L., Sablon, G., Bruynooghe, M.: Using interactive concept-learning for knowledge base validation and verification. In: Ayel, M., Laurent, J. (eds.) *Validation, Verification and Testing of Knowledge Based Systems*, pp. 177–190. Wiley, New York (1991)
51. Preece, A.D., Shinghal, R., Batarekh, A.: Principles and practice in verifying rule-based systems. *Knowl. Eng. Rev.* **7**(02), 115–141 (1992)
52. Zhang, D., Nguyen, D.: Prepare: a tool for knowledge base verification. *IEEE Trans. Knowl. Data Eng.* **6**(6), 983–989 (1994)
53. Ayel, M., Laurent, J.P.: Sacco-Sycojet: two different ways of verifying knowledge-based systems. In: Ayel, M., Laurent, J.P. (eds.) *Validation, Verification and Test of Knowledge-Based Systems*, pp. 63–76. Wiley, New York (1991)
54. Rousset, M.C.: On the consistency of knowledge bases: the covadis system. In: *ECAI*, pp. 79–84 (1988)
55. Craw, S., Sleeman, D.H.: Automating the refinement of knowledge-based systems. In: *ECAI*, pp. 167–172 (1990)
56. Culbert, S.: Expert system verifications and validation. In: *Proceedings of First AAAI Workshop on V,V and Testing* (1988)
57. Preece, A.D.: A new approach to detecting missing knowledge in expert system rule bases. *Int. J. Man-Mach. Stud.* **38**, 161–181 (1993)
58. Vermesan, A.: Foundation and application of expert system verification and validation. *The Handbook of Applied Expert Systems*. CRC Press, Boca Raton (1997)
59. Genesereth, M.R., Fikes, R.E.: *Knowledge interchange format version 3.0 reference manual* (1992)
60. Delugach, H.: ISO/IEC 24707 information technology–common logic (CL) – A framework for a family of logic-based languages. The formally adopted ISO specification (2007)
61. Kifer, M., Boley, H.: RIF overview. W3C working draft, W3C (2009). <http://www.w3.org/TR/rif-overview>
62. Kifer, M.: Rule interchange format: the framework. In: Calvanese, D., Lausen, G. (eds.) *Web Reasoning and Rule Systems, Second International Conference, RR 2008, Karlsruhe, Germany, October 31–November 1, 2008*. Proceedings. Lecture Notes in Computer Science, vol. 5341, pp. 1–11. Springer (2008)
63. Paschke, A., Reynolds, D., Hallmark, G., Boley, H., Kifer, M., Polleres, A.: RIF core dialect. Candidate recommendation, W3C (2009). <http://www.w3.org/TR/2009/CR-rif-core-20091001/>

64. Kifer, M., Boley, H.: RIF basic logic dialect. Candidate recommendation, W3C (2009). <http://www.w3.org/TR/2009/CR-rif-bld-20091001/>
65. Hallmark, G., Paschke, A., de Sainte Marie, C.: RIF production rule dialect. Candidate recommendation, W3C (2009). <http://www.w3.org/TR/2009/CR-rif-prd-20091001/>
66. Lukichev, S.: Towards rule interchange and rule verification. Ph.D. thesis, Brandenburg University of Technology (2010) urn:nbn:de:kobv:co1-opus-20772. <http://d-nb.info/1013547209>
67. Wang, X., Ma, Z.M., Zhang, F., Yan, L.: RIF centered rule interchange in the semantic web. In: Bringas, P.G., Hameurlain, A., Quirchmayr, G. (eds.) Database and Expert Systems Applications, 21st International Conference, DEXA 2010, Bilbao, Spain, August 30–September 3, 2010, Proceedings, Part I. Lecture Notes in Computer Science, vol. 6261, pp. 478–486. Springer (2010)
68. OMG: Production rule representation (OMG PRR) version 1.0 specification. Technical report formal/2009-12-01, Object Management Group (2009). <http://www.omg.org/spec/PRR/1.0>
69. Boley, H., Paschke, A., Shafiq, O.: RuleML 1.0: the overarching specification of web rules. In: Dean, M., Hall, J., Rotolo, A., Tabet, S. (eds.) Semantic Web Rules - International Symposium, RuleML 2010, Washington, DC, USA, 21–23 October 2010. Proceedings. Lecture Notes in Computer Science, vol. 6403, pp. 162–178. Springer (2010)
70. Boley, H., Tabet, S., Wagner, G.: Design rationale for RuleML: a markup language for semantic web rules. In: Cruz, I.F., Decker, S., Euzenat, J., McGuinness, D.L. (eds.) Proceedings of SWWS'01, The first Semantic Web Working Symposium, Stanford University, California, USA, July 30–August 1, 2001, pp. 381–401 (2001)
71. Wagner, G., Antoniou, G., Tabet, S., Boley, H.: The abstract syntax of RuleML - towards a general web rule language framework. Web Intell. IEEE Comput. Soc. 628–631 (2004)
72. Wagner, G., Giurca, A., Lukichev, S.: R2ml: a general approach for marking up rules. In: Bry, F., Fages, F., Marchiori, M., Ohlbach, H. (eds.) Principles and Practices of Semantic Web Reasoning, Dagstuhl Seminar Proceedings 05371 (2005)
73. Herre, H., Jaspars, J.O.M., Wagner, G.: Partial logics with two kinds of negation as a foundation for knowledge-based reasoning. Centrum voor Wiskunde en Informatica (CWI) **158**, 35 (1995)
74. Giarratano, J.C., Riley, G.D.: Expert Systems. Thomson, Toronto (2005)
75. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns, 1st edn. Addison-Wesley Publishing Company, Reading (1995)
76. Miller, J., Mukerji, J.: MDA guide version 1.0.1. OMG (2003)
77. Mellor, S.J., Balcer, M.J.: Executable UML: A Foundation for Model Driven Architecture, 1st edn. Addison-Wesley Professional, Boston (2002)
78. Burbeck, S.: Applications programming in smalltalk-80(TM): How to use model-view-controller (MVC). Technical report, Department of Computer Science, University of Illinois, Urbana-Champaign (1992)
79. Bieberstein, N., Bose, S., Fiammante, M., Jones, K., Shah, R.: Service-Oriented Architecture (SOA) Compass: Business Value, Planning, and Enterprise Roadmap. IBM Press, Indianapolis (2006)
80. OMG: Business process model and notation (BPMN): version 2.0 specification. Technical report formal/2011-01-03, Object Management Group (2011)
81. Kluza, K., Kaczor, K., Nalepa, G.J.: Enriching business processes with rules using the Oryx BPMN editor. In: Rutkowski, L., et al. (eds.) Artificial Intelligence and Soft Computing: 11th International Conference, ICAISC 2012: Zakopane, Poland, April 29–May 3, 2012. Lecture Notes in Artificial Intelligence, vol. 7268, pp. 573–581. Springer (2012)
82. Kluza, K., Nalepa, G.J.: Business processes integrated with business rules, F.M. Inf. Syst. Front. (2016) submitted
83. Guida, G., Lamperti, G., Zanella, M.: Software Prototyping in Data and Knowledge Engineering. Kluwer Academic Publishers, Norwell (1999)

Modeling with Rules Using Semantic Knowledge
Engineering

Nalepa, G.J.

2018, XXVI, 435 p. 130 illus., 35 illus. in color.,

Hardcover

ISBN: 978-3-319-66654-9