# Methodologies and Technologies for Rule-Based Systems Design and Implementation. Towards Hybrid Knowledge Engineering.

Grzegorz J. Nalepa[1]

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

**Summary.** A practical design of non-trivial rule-based systems requires a systematic, structured and consistent approach. The paper focuses on selected issues in RBS knowledge engineering. Some ideas on combining knowledge engineering with software engineering are discussed. Furthermore, results of RBS design tools survey are enclosed. In the paper an original design and implementation methodology for RBS is also presented. It has been developed in the Mirella project. It is a top-down hierarchical design methodology, based on new knowledge representation methods (XTT and ARD), on-line logical system analysis in Prolog, and XML-based knowledge encoding. Basing on the experience with XTT-based methodology, as well as tools supporting it, the paper discusses an extended hierarchical design methodology for RBS. A preview of the Hekate project, which aims at developing a hybrid knowledge engineering methodology is also given.

## 1 Introduction

Knowledge-based systems (KBS) are an important class of intelligent systems originating from the field of Artificial Intelligence [17]. They can be especially useful for solving complex problems in cases where purely algorithmic or mathematical solutions are either unknown or demonstrably inefficient.

Building real-life KBS is a complex task. Since their architecture is fundamental different from classic software, classical software engineering approaches cannot be applied efficiently. Some specific development methodologies, commonly referred to as *knowledge engineering*, are required.

In AI *rules* are probably the most popular choice for building knowledge-based systems, that is the so-called rule-based expert systems [4, 5, 7]. Rule-based systems (RBS) are used extensively in practical applications, especially in domains such as automatic control, decision support, and system diagnosis. They constitute today one of the most important classes of KBS.

A rule-based expert system consists of a knowledge base and an inference engine. The knowledge engineering process aims at designing and evaluating the knowledge base, and implementing a proper inference engine. The process of building the knowledge base involves the selection of a knowledge representation method, knowledge acquisition, and possibly low-level knowledge encoding. In order to create an inference engine a reasoning technique must be selected, and the engine has to be programmed.

During the engineering process a number of problems occur. Particular problems concern the selection of a knowledge representation formalism as well as the actual design of an appropriate rule base. As the number of rules exceeds even relatively very low quantities, it is hard to keep the rule-base consistent, complete, and correct. These problems are related to knowledge-base verification, validation, and testing [21, 22]. The selection of appropriate software tools and programming languages is non-trivial either.

This paper is devoted to discussing the most important differences between knowledge engineering (see Sect. 2) and classic software approaches [19] (see Sect. 3). When it comes to practical system implementation, current RBS development is heavily dependent on software engineering tools, which enforce certain design patterns not suitable for knowledge engineering. This is why the paper aims at identifying possible areas of cooperation between software and knowledge engineering approaches in Sect. 4. Section 5 identifies the most important issues in RBS knowledge engineering process and presents possible approaches. Then in Sect. 6 an overview of selected design and implementation tools is presented. Practical design of non-trivial rule-based systems requires a systematic, structured and consistent approach. In this paper an original design and implementation methodology for rule-based systems is discussed in Sect. 7. It is a top-down hierarchical design methodology, based on new knowledge representation methods (XTT and ARD), on-line logical system analysis in Prolog, and XML-based knowledge encoding. It is supported by a prototype CASE tool called Mirella. Basing on the experience with XTT-based methodology, as well as tools supporting it, in the Sect. 8 an extended hierarchical design methodology for RBS is discussed. At the end a preview of the HEKATE project, which aims at developing a hybrid knowledge engineering methodology, is also given.

## 2 Knowledge Engineering Approach

What makes KBS distinctive is the separation of knowledge storage (the knowledge base) from the knowledge processing facilities. In order to store knowledge, KBS use various knowledge representation methods, which are *declarative* in nature. In case of RBS these are *production rules*. Specific knowledge processing facilities, suitable for specific representation method used, are then selected. In case of RBS these are logic-based inference engines.

The knowledge engineering (KE) process in case of RBS involves two main tasks: knowledge base design, and inference engine implementation. Furthermore, some other specific tasks are also required, such as: knowledge base analysis and verification, and inference engine optimization. The performance of a complete RBS should be *evaluated* and *validated*. Classic expert systems books [5] represent this process as shown in Fig. 1. While this process is specific to expert systems in general, it is usually similar in case of other KBS.

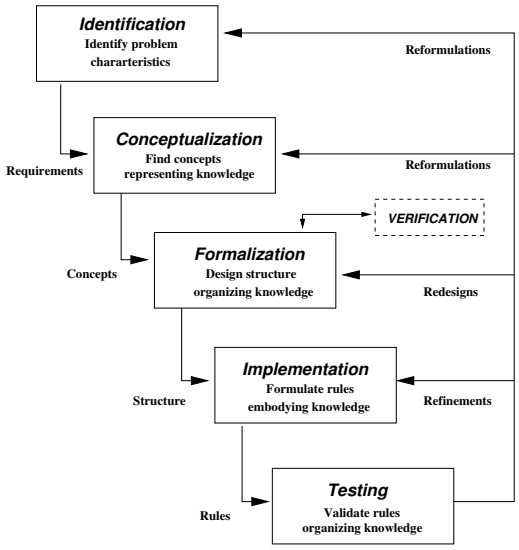


**Fig. 1.** Classic knowledge engineering process (Liebowitz 1998)

What is important about the process, is the fact that it should *capture* the expert knowledge and *represent* it in a way that is suitable for processing (this is the task for the knowledge engineer). The actual structure of a KBS does not need to be system specific – it should not „mimic" or model the structure of the real-world problem. However, the KBS should capture and contain the knowledge about the real-world system. The task of the programmers is to develop processing facilities for the knowledge representation.

It should be pointed out, that in case of KBS there is no single universal engineering approach, or universal modelling method (such as UML in software engineering). Different classes of KBS may require a specific approach, see [5, 2, 7, 20]. Having outlined the main aspects of KBS development, it can be discussed how they are related to classic software engineering methods.

## 3 Software Engineering Approach

Software engineering (SE) is the domain where a number of mature and well-proved design methods exist. They address needs of specific classes of business software. In software engineering the software development process and life cycle is represented by several models. One of the most common is called *the waterfall model* [19] and is shown in Fig. 2. In this process a number of development roles can be identified: users and/or domain experts, system analysts, programmers, testers, integrators, and end users. What makes this process different from knowledge engineering, is the fact, systems analysts in general try to *model* the *structure* of the real-world information system in the structure of computer software system. So the structure of the software corresponds to some respect to the structure of the real-world system. The task of the programmers is to encode and implement the model (which is the result of the system analysis) in some lower-level programming language.

The most important difference between software and knowledge engineering, is that the former tries to model how the system works, while the latter tries to capture and represent what is known about the system.

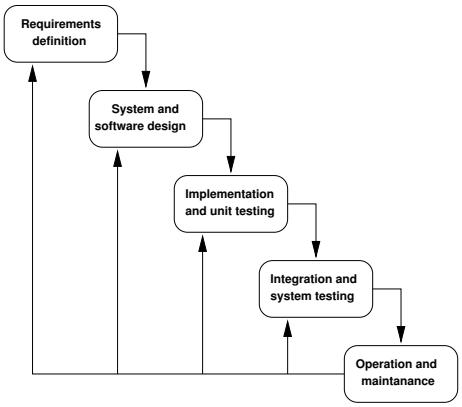


**Fig. 2.** Classic waterfall software life cycle (Sommerville 2004)

## 4 Heterogeneous Development Methodology

Historically, there has always been a strong feedback between software engineering and computer programming tools. At the same time these tools have been strongly determined by the actual architecture of computers themselves. For a number of years there has been a clear trend for the software engineering to become as implementation-independent as possible. Modern software engineering approaches tend to be abstract and conceptual.

On the other hand, knowledge engineering approaches have always been device and implementation-agnostic. The actual implementation of KBS has been based on some high level programming languages such as Lisp or Prolog. However, modern knowledge engineering tools heavily depend on some common development tools and programming languages, especially when it comes to user interfaces, network communication, etc.

It could be said, that these days software engineering becomes more knowledge-based, while knowledge engineering is more about software engineering. This opens multiple opportunities for both approaches to improve and benefit. Software engineering could adopt from knowledge engineering: advanced conceptual tools, such as declarative knowledge representation methods, knowledge transformation techniques based on existing inference strategies, as well as verification, validation and refinement methods. This trend is already visible in the *business rules* approach [16, 23]. *Model-Driven Architecture* (MDA) is a new software engineering paradigm that tries to provide a unified design and implementation method and appropriate tools for the declarative specification of business logic [9].

In order to improve and better integrate with existing software knowledge engineering could adopt: programming interfaces to existing software systems and tools, interfaces to advanced storage facilities such as databases and data warehouses, modern user interfaces, including graphical and web-based ones.

This paper is written from the knowledge engineering point of view. This is why the following sections focus on different ways of improving the KE process in case of RBS.

## 5 Rule-Based Systems Design Issues

In RBS development knowledge engineering is essentially a process of construction. As it was pointed out in Sect. 2, it involves two main tasks: knowledge base (rule base) design, and inference engine implementation.

### 5.1 Rule Base Design

The first decision that has to be made is one concerning knowledge representation method. It is widely recognized that there is no single formalism suitable to represent knowledge for all purposes. A variety of formalisms and structures is needed to represent knowledge. In the field of rule-based expert systems the *knowledge representation method* is a systematic way of "encoding" what an expert knows about some domain. However "encoding" means here rather "describing" then "encrypting" [4].

Some of the issues arising in knowledge representation are: syntax, semantics, expressive adequacy, reasoning, completeness and other consistency issues, real-world knowledge, control, flexibility. Different representations address these issues in different ways [2]. While there are numerous knowledge

representation methods, the logic-based ones are essential to the theory and practice of rule-based systems and expert systems in general.

Although propositional calculus is a simple logical system, it can serve as a practically useful language for encoding rule-based systems. Further, both analysis and design of such systems is relatively simple. The most basic logical form of *propositional rules* is as follows (see [7]): $p_1 \wedge p_2 \wedge \ldots \wedge p_n \longrightarrow h$. This form of a rule is logically equivalent to a Horn clause, provided that all the literals are positive. A more complex rule may contain conclusion part composed of several propositions. In forward-chaining systems rules are applied by checking if their preconditions are satisfied. Whenever a rule is fired, its conclusion is added to the current knowledge base. Propositional rule-based systems can take various visual forms incorporating some structural representation; most important are: *decision tables* and *decision trees* [7].

*Decision tables* are an engineering way of representing production rules. Conditions are formed into a table which also holds appropriate actions. Classical decision tables use binary logic extended with "not important" mark to express states of conditions and actions to be performed.

The main advantage of decision tables is their simple, intuitive interpretation. One of the main disadvantages is that classical tables are limited to binary logic. In some cases the use of values of attributes is more convenient. A slightly extended tables are *OAV tables* (OAT). OAV stands for Object-Attribute-Value (OAT – Object-Attribute-Value-Table, see [7]).

*Decision trees* are an important representation, since the tree-like representation is readable, easy to use and understand. The root of the tree is an entry node, under any node there are some branching links. The selection of a link is carried out with respect to a conditional statement assigned to the node. The evaluation of this condition determines the selection of the link. The tree is traversed top-down, and at the leaves final decisions are defined.

Formal *ontologies* are an important knowledge representation method, used extensively in some new implementations of Web-oriented intelligent systems. Recently ontologies gained a precise semantic interpretation with the definition of OWL DL (description logics), which is currently extended by horn-clause rules (see [3] for a current proposal for SWRL/RuleML).

In expert system practice there are several other knowledge representation methods used. Their logical interpretation is not always as direct as decision rules, tables, or trees. However, they do have many applications as a valuable conceptualization tool. These includes: graphs, and conceptual graphs, semantic networks, and frames, see [5, 20] for more details.

### 5.2 Rule Base Encoding

On the low level rules have to be encoded in a format ready for processing. Inventing a new, specific rule format, may seem the most straightforward approach. It gives developers a lot of freedom when it comes to the implementation. However, it poses problems when interfacing with existing systems.

From the KE point of view, it is desirable to adopt some general standard. However, from the SE point of view, it might be desirable to adopt specific issues of the particular application.

A more common and reasonable approach consists in choosing an expert system shell, and using a predefined rule format. It simplifies the implementation, however it determines the system architecture. It can for example enforce certain inference strategy. More on this is elaborated in Sect. 6.3.

Encoding rules in some high level logic programming language such as Prolog is – to some degree – a good combination of the two above. Prolog allows inventing any rule format, while providing high-level inference strategies, see Sect. 6.2 for more details.

The development of the Web and recent W3C Semantic Web initiative make encoding rules for web applications an important issue. Encoding rules in an XML-based format, such as RuleML (*www.ruleml.org*) is often the best solution in such a case.

### 5.3 Rule Base Analysis

Rule-based expert systems technology is being applied to critical tasks and complex problem-solving. This is why there are concerns about its dependability. A proper system development cycle, as well as a rigorous verification and validation (*V&V*) can provide an appropriate level of quality and safety [22].

The verification and validation of expert systems are still a maturing field, so there is no apparent consensus among researches on a single definition. The following definitions may be found in [21]:

- *Verification* checks well-defined properties of an expert system against its specification; it can focus on the knowledge base or the inference engine.
- *Validation* checks whether an expert system corresponds to the system it is supposed to represent.
- *Testing* is the examination of the behavior of a program by executing the program on sample data sets.
- *Evaluation* focuses on the accuracy of the system knowledge.

In case of mission-critical RBS applied as control systems a formal verification is essential [7] in order to provide certain level of system safety.

### 5.4 Inference Engine Development

This stage involves choosing inference strategy for rule analysis. Two most general types of inference are: forward chaining and backward chaining. Furthermore, combinations of the two types can be applied. The most typical strategy is to use forward chaining as a general control strategy, while at some stages, if detailed goals are to be inferred, backward chaining is employed.

Depending on the rule encoding chosen an inference engine may be already provided. It is the case with expert systems shells. If Prolog rules are chosen, built-in Prolog backward-chaining approach can also be used directly.

Today a number of tools are freely available for an RBS developer. They support different phases of RBS design, implementation and analysis. Selected examples are described in the following section.

## 6 Selected Development Tools Overview

The modern tools available to assist in building expert systems can be divided into several categories discussed in the following subsections.

### 6.1 Conventional Programming Languages

*Conventional* programming languages, e.g. ANSI C, do not support programming paradigm suitable for expert systems. Their *procedural* approach does not match very well the *declarative* nature of an expert system. Using these languages, a development of expert systems, while possible, is very difficult.

*Object-oriented* languages could be considered higher level languages. There is a smaller *semantic gap* between expert systems and languages such as: Java, Smalltalk or Eiffel. This is why they are sometimes chosen as expert system implementation tools. Languages such as Python, or Ruby have been gaining a growing acceptance due to their fast prototyping capabilities.

It can be concluded that it is more common to choose conventional languages as low-level implementation tools, while using higher level tool such as *expert system shell* to build a knowledge base.

*Java* is a classic object-oriented programming language. However, it has become a language of choice for many Web-related AI projects. Currently there is a number of Java-based tools for expert systems, see Sect. 6.3. It is worth noting that a standardization effort (*JSR 94: Java Rule Engine API*) is currently undertaken to formulate a standard Java Rule Language.

### 6.2 AI Programming Languages

For many years *Lisp* has been a language of choice for *symbolic* computation. Features of Lisp [4] are: programs are represented by list structures, and primitive operations are operations on lists. Lisp is the foundation of many expert systems and shells, such as *CLIPS*. In last decades it was extended in many ways, including object-oriented framework *CLOS*.

There are, however, problems with Lisp. The main problem is that lists have limited knowledge representation capabilities. Another is that no strong programming methodology has emerged from Lisp-based tools. There is a number of different dialects of Lisp language too.

*Prolog* is both flexible and powerful, with strong logical foundations. It has facilities for both knowledge representation and processing. Opposed to Lisp which is a symbolic language, Prolog is a declarative one. However, it does have dual semantics, both declarative and procedural. It is well-suited to symbolic rather than numerical problems. Since there is only a small *semantic gap* between expert-systems and Prolog, the language is an ideal tool for practical development of these systems.

The Prolog language is based on predicate logic. Prolog clauses are Horn clauses from the logical point of view. In order to find solutions (satisfy goals) Prolog uses the resolution rule and unification. Prolog is studied in detail in [1]. It has some important features to support logic-based reasoning. The Prolog inference engine uses backward-chaining with backtracking and recursion.

*Meta programs* treat other programs as data. They are used to help in both understanding and building knowledge-based systems [20]. Prolog is almost unique in the extent to which it can serve as its own meta-language. A Prolog program can create new goals, examine itself, and modify the inference engine, blurring the distinction between program and data. Prolog-based meta-interpreters are ideal to build forward-chaining inference engines.

### 6.3 Selected Expert System Shells

*CLIPS* is one of the most common expert system development tools (`www.ghg.net/clips`). It supports multiple reasoning and conflict resolution strategies. CLIPS is an expert system shell, so it does not provide any tools supporting the design of the knowledge base.

*Jess* is a *Java Expert System Shell* (`jessrules.com`). It is inspired by CLIPS but implemented in Java. Compared to CLIPS it adds several features and offers superior performance. It is easy to integrate with Java-based web-enabled applications. It plays an important role in the JSR 94 effort.

*jDREW* (`www.jdrew.org`) is a deductive reasoning engine for clausal first order logic written in Java and well integrated with the Web. Knowledge-based systems can use jDREW as an embedded reasoning engine through its various APIs. jDREW can be easily deployed as part of a larger Java system.

The *Algernon* (`algernon-j.sf.net`) rule-based inference system is implemented in Java and interfaced with Protege ontology editor. It performs forward and backward rule-based processing of frame-based knowledge bases, and stores and retrieves information in ontologies and knowledge bases. It is aimed at integration with Semantic Web projects.

### 6.4 Selected Design Environments

*Sphinx* [8] is an integrated development environment for expert systems development. It uses backward-chaining inference engine, contains a shell (PC-Shell) and several design tools, such as CAKE, which supports the process of knowledge base design and simple verification.

*KbBuilder* [18] is an integrated environment for designing and verifying Sphinx knowledge bases. The approach is oriented towards backward-chaining systems based on simple attributive language. Furthermore, its verification capabilities are limited to local properties of the so-called decision units.

*Mandarax* (`mandarax.sf.net`) is an open source Java class library for deduction rules. It provides an infrastructure for defining, managing and querying rule bases. Mandarax includes open APIs to interface with relational databases and XML, in particular RuleML. *Oryx* is a graphical user interface application to design and maintain Mandarax knowledge bases.

*XpertRule* (`www.attar.com`) supports developing rule-based systems. It uses a simple visual knowledge builder which maps knowledge modules to decision trees, which are main knowledge representation units. It also provides additional features, such as fuzzy reasoning.

*VisiRule* (`www.lpa.co.uk`) is a visual design tool for developing expert systems. A principal idea is to support the designer by a graphical flowchart representing the decision logic. The chart can be automatically translated into a lower level logic-based representation, processed in Prolog. The most important feature is the support for the visual design of the knowledge base; it is however, limited to decision trees. VisiRule does not provides means to validate or evaluate the knowledge base.

*Drools* (`www.drools.org`) is a framework for building forward-chaining expert systems, with the use of the Rete algorithm. It is implemented and Java, and integrated with Java building tools. It generates source in a selected language, from a conceptual description encoded in XML. This description includes declarative parts (rules) and embedded procedural code in the target language. The tool does not offer any verification or evaluation facilities.

## 7 Mirella Project

In [10] results of a research and evaluation of multiple RBS design methods, supported by development tools have been presented. A conclusion has been drawn, that existing methods and tools have some serious limitations located in the following areas: knowledge representation, formal analysis and verification, and design support tools. Most important limitations concerning the knowledge representation methods consist in using system-specific knowledge representation formalisms. This results in restricted application area, and scalability problems. With respect to the practical analysis approaches, the following problems have been identified: late verification problem, inefficient development cycle, and lack of integrated software framework.

Available design approaches do not offer integrated computer development tools (*CASE*) supporting the RBS building process at *all* stages – from the design to implementation. Such methods support mainly subsequent stages of the *conceptual design*, while direct technical support for the logical design

and during the implementation phase is mostly limited to providing a context-sensitive, syntax checking editors, or simple wizards that support the design.

Practical design of non-trivial RBS requires a systematic, structured and consistent approach. Such an approach is usually referred to as a *design methodology*. To overcome limitations outlined above, a new approach to RBS design process, supported by an integrated CASE tool, has been proposed [10].

It is a top-down hierarchical design methodology, based on the idea of meta-level approach to the design process. It includes three phases: conceptual, logical, and physical. It provides a clear separation of logical and physical (implementation) design phases. It offers equivalence of logical design specification and prototype implementation, and employs XTT, a new hybrid knowledge representation. The methodology is supported by a CASE tool.

The main goal of the methodology is to move the design procedure to a logical level, where knowledge specification is based on the use of abstract rule representation. The design specification can be automatically translated into a low-level code, including Prolog and XML, so that the designer can focus on logical specification of safety and reliability. On the other hand, selected system properties can be automatically analyzed on-line during the design, so that its characteristics are preserved. The generated Prolog code constitutes a prototype implementation of the system. Since it is equivalent to the visual design specification it can be considered an executable specification.

These ideas are the basis for the MIRELLA Project, `mirella.ia.agh.edu.pl`. The goals of the project are: to fully develop and refine the design process outlined above, extend its' application areas onto different real-life RBS, and provide computer tools and methods supporting this process. So far the following elements have been developed: the XTT knowledge representation method, the concept of an integrated design process, a prototype Mirella CASE tool. They have been all described in detail in [10]. Some of the applications of these ideas were presented in [11, 12]. Further developments include ARD conceptual design [13, 7]. All of these are shortly introduced below.

### 7.1 EXtended Tabular Trees

The main idea behind the new visual knowledge representation language called *Extended Tabular-Trees* [10] aims at combining some of the existing approaches, namely decision trees and decision tables building a special hierarchy of Object-Attribute-Tables [6, 7]. It allows for a hierarchical visual representation of the OAT tables linked into tree-like structure, according to the control specification provided. XTT as a design and knowledge representation method offers transparent, high density knowledge representation as well as a formally defined logical, Prolog-based interpretation, while preserving flexibility with respect to knowledge manipulation. On the *machine readable level* XTT can be represented in an XML-based *XTTML* (*XTT Markup Language*) suitable for import and export operations; it can also be translated to XML-based rule markup formats such as *RuleML*.

### 7.2 Integrated Design Process

The *eXtended Tabular Trees*-based design method introduces possibility of on-line system properties analysis and verification, during the system design phase. Using XTT as a core, in [10] an integrated design process, covering the following phases has been presented:

1. *Conceptual modeling*, in which system attributes and their functional relationships are identified; during this design phase the ARD modelling method is used. ARD stands for *Attribute-Relationship Diagrams* [13, 7]. It allows for specification of functional dependencies of system attributes using a visual representation. Using this model the logical XTT structure can be designed. ARD can be represented in an XML-based *ARDML* (*ARD Markup Language*) suitable for data exchange operations, as well as possibly transformations to other diagram formats.
2. *Logical design with on-line verification*, during which system structure is represented as XTT hierarchy, which can be instantly analyzed, verified (and corrected, if necessary) and even optimized on-line. The XTT hierarchy can also be represented in XML, using the XTTML format.
3. *Physical design*, in which a preliminary Prolog-based *implementation* is carried out. A RuleML translation of the XTT rule base is also available.

Using the predefined XTT translation it is possible to automatically build a prototype. It uses Prolog-based meta-language for representing XTT knowledge base and rule inference (also referred to as XTT-Prolog).

### 7.3 Mirella CASE Tool

A prototype CASE tool for the XTT method called Mirella [10] has been developed. It supports XTT-based visual design methodology, with an integrated, incremental design and implementation process, providing the possibility of the on-line, incremental, verification of formal properties. Logical specification is directly translated into Prolog-based representation providing an executable prototype, so that system operational semantics is well-defined. In the Mirella Editor the specification looks as in Fig. 3.

### 7.4 Meta-Level Features

The approach is based on the idea of a knowledge representation method which offers the *design and implementation equivalence* by a direct XTT $\rightarrow$ Prolog mapping. Using a visual design method the designer can focus on building the system structure, since the prototype implementation can be *dynamically generated* and *automatically analyzed*. The approach discussed herein offers strict, formal description of system attributes and structure, creates a framework for integrating the design and verification process, and supports the design and verification process by an integrated CASE tool.

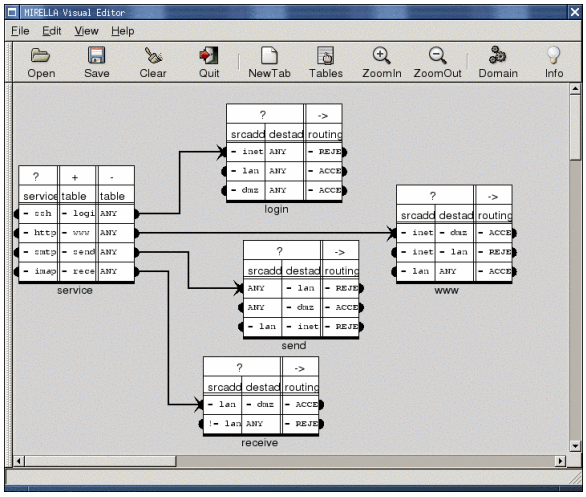


**Fig. 3.** RBS design session in Mirella

In this way, it is possible to assure that some safety-critical system properties such as attribute domains, and basic system structural logical constraints are preserved during the design process.

### 7.5 Lessons learned from Mirella

*Expressive* knowledge representation is needed in order to truly support the design. Mirella is focused around XTT, which proved to be a valuable tool, allowing for designing different classes of RBS. Addressing *all of the design phases* is an important issue addressed in Mirella. In order to successfully build real-life systems it is necessary to formulate a complete design methodology, covering design stages from conceptual analysis to the implementation, including verification. *On-line* formal verification allows for assuring system characteristics during the design, and keeping them up the the implementation. *Integration* of design phases is needed in order to truly support the designer, and preserve system characteristics during the design process. Easy to use *visual CASE tool* not only is important for the design support but also is necessary for the adoption of the new design methodology.

Mirella in its current state was successful as a proof of concept. However, after more than two years of development some possible areas of extension and improvement have been identified, such as: business rules support, automatic knowledge acquisition facilities, optional backward-chaining, possibly fuzzy rules support, and even more extended verification capabilities [14].

## 8 Towards Hybrid Knowledge Engineering

Basing on the experiences with the MIRELLA project a refined RBS design methodology is put forward. It addresses three design phases described in Sect. 7.2, that is: conceptual, logical, and physical design. It also addresses three important aspects of the design models used, that is:

- *visual representation*, which is valuable for both the design support and the human interaction,
- *knowledge encoding*, which is based on XML and is useful for automatic models transformations,
- *executable code*, which is based on Prolog representation of RBS.

An outline of this approach is shown in Fig. 4.

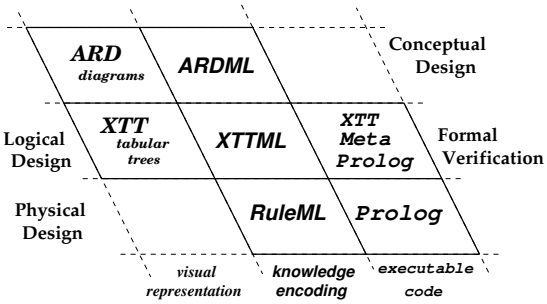


**Fig. 4.** Hierarchical design methodology

It is hoped, that if refined, this methodology could provide universal modelling methods for RBS design and implementation.The HEKATE projects aims at applying this methodology to practical design and analysis of intelligent systems. Main goals of the HEKATE project are to:

- develop an extended, hierarchical methodology for practical design, analysis and implementation of selected software classes,
- build computer CASE tools package supporting this methodology,
- test the approach on illustrative software examples.

The projects focuses on wide class of software, namely two very different classes, that is:

- general business software based on the so called *business logic*,
- low-level control software, possibly for the embedded control systems, based on a *control logic* [15].

A principal idea in this approach is to describe the logic behind the software using advanced knowledge representation methods. The logic would be expressed with use of a Prolog-based representation. The logical, Prolog-based

core would be then embedded into a business application, or embedded control system. The business or control applications can be developed with some classic programming languages such as Java or C. The HEKATE project should eventually provide a coherent runtime environment for running the combined Prolog and Java/C code.

Hekate is currently (fall 2006) in a very early development stage. See the project webpage at `hekate.ia.agh.edu.pl` for more up to date information on the project progress, tools and technologies.

## 9 Concluding Remarks

In the paper RBS knowledge engineering issues, methodologies and selected tools have been discussed. They have been contrasted with some aspects of software engineering. The paper also discusses and advanced design methodology developed in MIRELLA project which aims at combining classic knowledge engineering methods with software engineering approach. While MIRELLA is work in progress it already created some valuable results such as XTT knowledge representation method, along with on-line Prolog verification approach. It is hoped that the HEKATE will develop these concepts into a complete hierarchical design and implementation methodology for both business software and rule-based systems.

## References

1. Ivan Bratko. *Prolog Programming for Artificial Intelligence*. Addison Wesley, 3rd edition, 2000.
2. Adrain A. Hopgood. *Intelligent Systems for Engineers and Scientists*. CRC Press, Boca Raton London New York Washington, D.C., 2nd edition, 2001. ISBN 0849304563.
3. Ian Horrocks, Peter F. Patel-Schneider, Sean Bechhofer, and Dmitry Tsarkov. Owl rules: A proposal and prototype implementation. *Journal of Web Semantics*, 3(1):23–40, 2005.
4. Peter Jackson. *Introduction to Expert Systems*. Addison–Wesley, 3rd edition, 1999. ISBN 0-201-87686-8.
5. Jay Liebowitz, editor. *The Handbook of Applied Expert Systems*. CRC Press, Boca Raton, 1998. ISBN 0-8493-3106-4.
6. A. Ligęza, I. Wojnicki, and G.J. Nalepa. Tab-trees: a case tool for design of extended tabular systems. In H.C. Mayr et al., editor, *Database and Expert Systems Applications*, volume 2113 of *Lecture Notes in Computer Sciences*, pages 422–431. Springer-Verlag, Berlin, 2001.
7. Antoni Ligęza. *Logical Foundations for Rule-Based Systems*. Springer-Verlag, Berlin, Heidelberg, 2006.
8. K. Michalik. *Zintegrowany Pakiet Sztucznej Inteligencji Sphinx 4.0*. AITech Artificial Intelligence Laboratory, Katowice, Poland, 2003.
9. Joaquin Miller and Jishnu Mukerji. *MDA Guide Version 1.0.1*. OMG, 2003.

10. Grzegorz J. Nalepa. *Meta-Level Approach to Integrated Process of Design and Implementation of Rule-Based Systems*. PhD thesis, AGH University of Science and Technology, AGH Institute of Automatics, Cracow, Poland, September 2004.
11. Grzegorz J. Nalepa and Antoni Ligęza. Designing reliable web security systems using rule-based systems approach. In Ernestina Menasalvas, Javier Segovia, and Piotr S. Szczepaniak, editors, *Advances in Web Intelligence. First International Atlantic Web Intelligence Conference AWIC 2003, Madrid, Spain, May 5-6, 2003*, volume LNAI 2663 of *Lecture Notes in Artificial Intelligence*, pages 124–133, Berlin, Heidelberg, New York, 2003. Springer-Verlag.
12. Grzegorz J. Nalepa and Antoni Ligęza. Markup-languages-based approach to knowledge management and representation. In Małgorzata Nycz and Mieczysław Lech Owoc, editors, *Pozyskiwanie Wiedzy i Zarządzanie Wiedzą*, number 1011 in Prace Naukowe Akademii Ekonomicznej im. Oskara Langego we Wrocławiu, pages 332–339, Wrocław, 2004. Akademia Ekonomiczna im. Oskara Langego we Wrocławiu.
13. Grzegorz J. Nalepa and Antoni Ligęza. Conceptual modelling and automated implementation of rule-based systems. In Tomasz Szmuc Krzysztof Zieliński, editor, *Software engineering : evolution and emerging technologies*, volume 130 of *Frontiers in Artificial Intelligence and Applications*, pages 330–340. IOS Press, 2005.
14. Grzegorz J. Nalepa and Antoni Ligęza. Prolog-based analysis of tabular rule-based systems with the xtt approach. In Geoffrey C. J. Sutcliffe and Randy G. Goebel, editors, *FLAIRS 2006 : proceedings of the nineteenth international Florida Artificial Intelligence Research Society conference : [Melbourne Beach, Florida, May 11–13, 2006]*, pages 426–431, FLAIRS. - Menlo Park, 2006. Florida Artificial Intelligence Research Society, AAAI Press.
15. Grzegorz J. Nalepa and Piotr Zięcik. Integrated embedded prolog platform for rule-based control systems. In Andrzej Napieralski, editor, *MIXDES 2006 : MIXed DESign of integrated circuits and systems : proceedings of the international conference : Gdynia, Poland 22–24 June 2006*, pages 716–721, Łodź, 2006.
16. Ronald G. Ross. *Principles of the Business Rule Approach*. Addison-Wesley Professional, 2003.
17. Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 2nd edition, 2002.
18. Roman Simiński. *Dynamiczna weryfikacja poprawności baz wiedzy w procesie ich weryfikacji*. PhD thesis, Instytut Podstaw Informatyki PAN, Warszawa, 2002.
19. Ian Sommerville. *Software Engineering*. International Computer Science. Pearson Education Limited, 7th edition, 2004.
20. I. S. Torsun. *Foundations of Intelligent Knowledge-Based Systems*. Academic Press, London, San Diego, New York, Boston, Sydney, Tokyo, Toronto, 1995.
21. A. Vermesan. *The Handbook of Applied Expert Systems*, chapter Foundation and Application of Expert System Verification and Validation. CRC Press, 1998.
22. A. Vermesan and F. Coenen, editors. *Validation and Verification of Knowledge Based Systems. Theory, Tools and Practice*. Kluwer Academic Publisher, Boston, 1999.
23. Barbara von Halle. *Business Rules Applied: Building Better Systems Using the Business Rules Approach*. Wiley, 2001.