Institute e-Austria in Timisoara

IeAT Report Series

Debugging and Verification of Expert Systems

Daniel POP, Viorel NEGRU

2004

Debugging and Verification of Expert Systems

Daniel Pop, Viorel Negru
Department of Computer Science, University of the West from Timişoara
Institute e-Austria Timisoara
4 V. Pârvan Street, RO-1900 Timişoara, Romania
E-mail: ***popica@optsol.ro,*** *vnegru@****info.uvt.ro***

Abstract. The use of expert systems in various disciplines proves an increase in human productivity, financial benefits and a better answer to users needs. There is a need for a development and integration environment that supports knowledge management and expert system construction, but also the debugging and verification as well.

Keywords. Knowledge-based debugging, automated verification, trace analysis.

1. Introduction

The use of expert systems in various disciplines proves an increase in human productivity, financial benefits and a better answer to users needs. Expert System Creator is an approach towards developing a software engineering tool for knowledge management by merging conventional CASE tool facilities with the expert system technology. Expert System Creator - created as a joint project between academic and commercial partners - assists the human designer by efficient encoding and by reusing the expert knowledge. It helps professional in the process of shifting from old informational systems implementation to modern approaches. Expert System Creator is a development and integration environment for knowledge management, expert system construction, debugging and verification.

A similar approach is represented by a family of software CREATOR expert systems has been developed [4]. Although an application of these systems is assisting the human designer when using a conventional CASE tool, they do not support the debugging, verification or profiling steps.

Another powerful knowledge management tool is Protégé-2000 project from Stanford Medical Informatics [13], which allows the construction of a domain ontology, the customization of knowledge-acquisition forms and the entering of domain knowledge. Moreover, Expert System Creator not only constructs the knowledge base but also integrates it within external projects.

Expert System Creator is an application for the development, verification and debugging, profiling and optimization of expert system applications. It represents the domain knowledge as rules set, decision tables or classification trees and automatically generates them as CLIPS, C++ or Java programs. It visually debugs the constructed expert systems using their high-level representation: rules set, decision tables or classification trees.

The next section presents the most frequent knowledge representation forms for classification problems. Section 3 presents in detail the verification, validation and debugging techniques incorporated in the Expert System Creator application. The last section presents several applications, as well as and future research directions and extensions.

2. Knowledge representation in classification problem

For the representation of knowledge in expert systems for classification problems, a number of forms are used, such as: rules set (production rules, association rules, rules with exceptions), decision tables, classification and regression trees, instance-based representations, and clusters. Table 1 synthesizes the main features of each knowledge representation form. Each representation has its advantages and drawbacks. Expert System Creator is endowed with advanced graphical manipulation tools for three of the above forms: decision table, classification tree and rules set. While a human expert can build in a straightforward way an expert system based on classification or association rules, the decision table representation allows easy automatic analysis and error and consistency checking. A classification tree can be very easily translated into any common programming language (such as C/C++, Java, Pascal etc.). As these three forms are equivalent [1][14], an expert system built in one of them can be translated into any other one. The rest of the section will briefly describe the knowledge representation forms (see [19][1] for a detailed presentation).

Rules set - A rule-based system form is a set of one or more rules. A *rule* has two parts - an antecedent and a consequent – and has the following form: **if antecedent then consequent**, where the antecedent of a rule is a conjunction of elementary constraints, and the consequent is a sequence of elementary actions. Rules may be production rules, association rules and they can have exceptions.

Decision table - A decision table consists of a two-dimensional array of cells, where the columns contain the system's constraints and each row makes a classification according to each cell's value (case of condition).

S`akd 0- Jmnvkdcfd qdoqdrdms`shnm enqlr

Knowledge Form	*Features*
Decision Tables	- automatic correctness analysis - compact, but rudimentary, form - easy to visualize and understand
Classification Trees Regression Trees	- procedural or object-oriented programming language code can be easily generated - automatically built from large datasets using data mining and statistical algorithms
Rules: Production rules Association rules Rules with exceptions	- easily built and assimilate by human experts - important, mature and large systems are already in use
Clusters	- unsupervised learning - uses data visualization techniques - often followed by a classification tree (rule set) inferring stage

Classification tree - A classification tree consists of a set of nodes and a set of arcs [15]. The set of nodes is divided into two classes: decision nodes and classification/action nodes (leaf nodes). Each decision node is associated with a constraint of the system and each leaf node (classification node) makes the classification based on the cases of the constraints from the decision nodes. Each arc has as its source a deciding node, and is associated with a case corresponding to the constraint from the source decision node, and the destination is a decision or classification node.

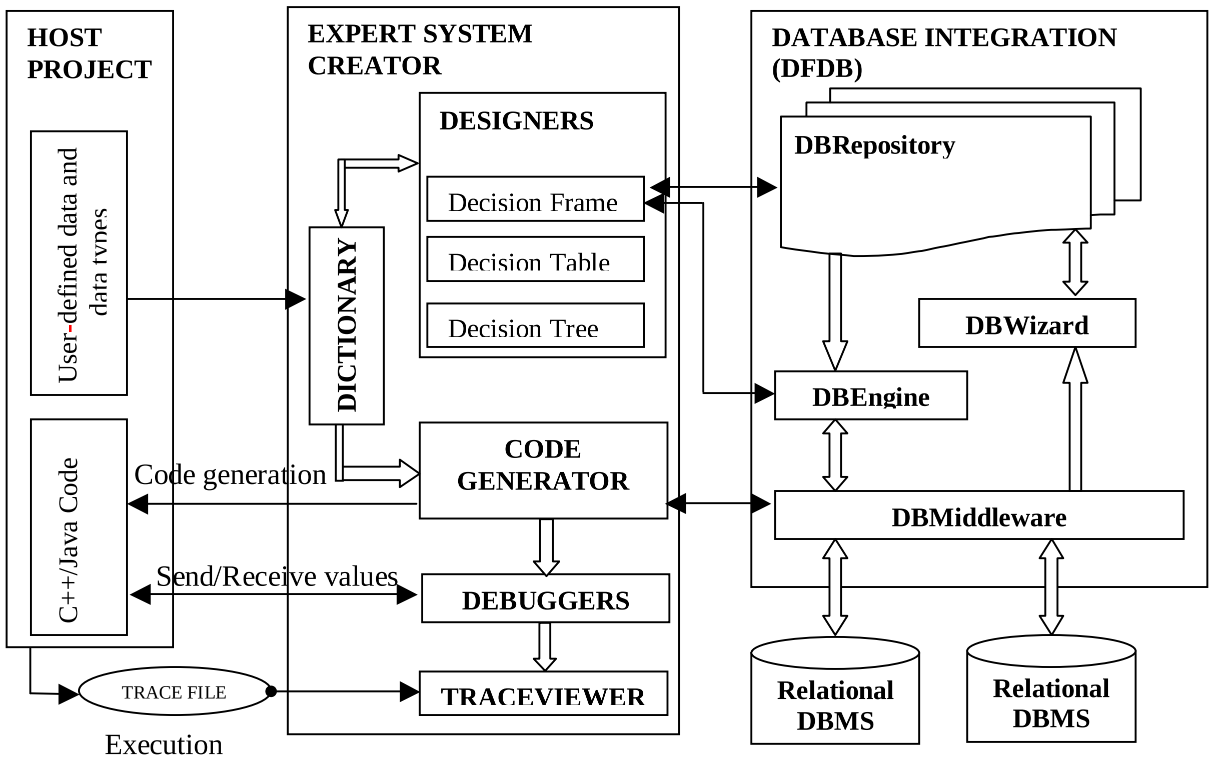
A special type of classification trees is the binary tree, which consists of a set of decision nodes and a set of leaf nodes. A decision node is associated with a boolean constraint. A boolean constraint can have two cases: true and false. There are algorithms that implement a decision table in a programming language using binary classification trees [4].

Regression tree – When it comes to predict numeric quantities, the same kind of tree representation as above can be used, but the leaf nodes of the tree would contain a numeric value which is the average of all the training set values that the leaf applies to.
Clusters – In case of clusters, the knowledge takes the form of a diagram that shows how the instances fall into clusters. In the simplest case this involves associating a cluster number with each instance. Instances can be associated probabilistic to more than one cluster. Clustering is often followed by a stage where a classification tree or rule set is inferred.

3. Verification, validation and debugging

The knowledge base completion and correctness are key issues in designing large knowledge base systems. Expert System Creator includes appropriate mechanisms for visualizing and testing the correctness of constructed knowledge bases and for debugging the execution of expert systems. Fig. 1 shows the general architecture of the entire application. Some modules (like those of the database integration subsystem) are not described in this paper. The reader is referred to [11] for details about these modules.

The *integrated debuggers* in Decision Frame/Table/Tree Designer can be used to debug the final system in its original form (as a rules set, table or tree), instead of “classical” C++/Java/Jess debugging. This new approach helps domain users and software developers to visually test and repair the constructed expert system.



Elf t qpl 0- DwodqsRxr sdll Bqpl`snq`qbglsdbst qpl

3.1 Rules set

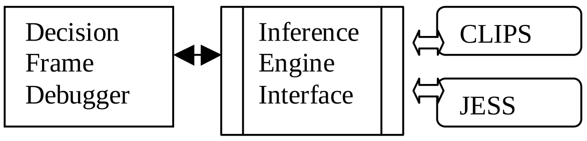
For rule-based systems, the *semantic graph (SG)* highlights the relationships between rules and templates. The semantic graph is a pair

$$SG = (N, A)$$

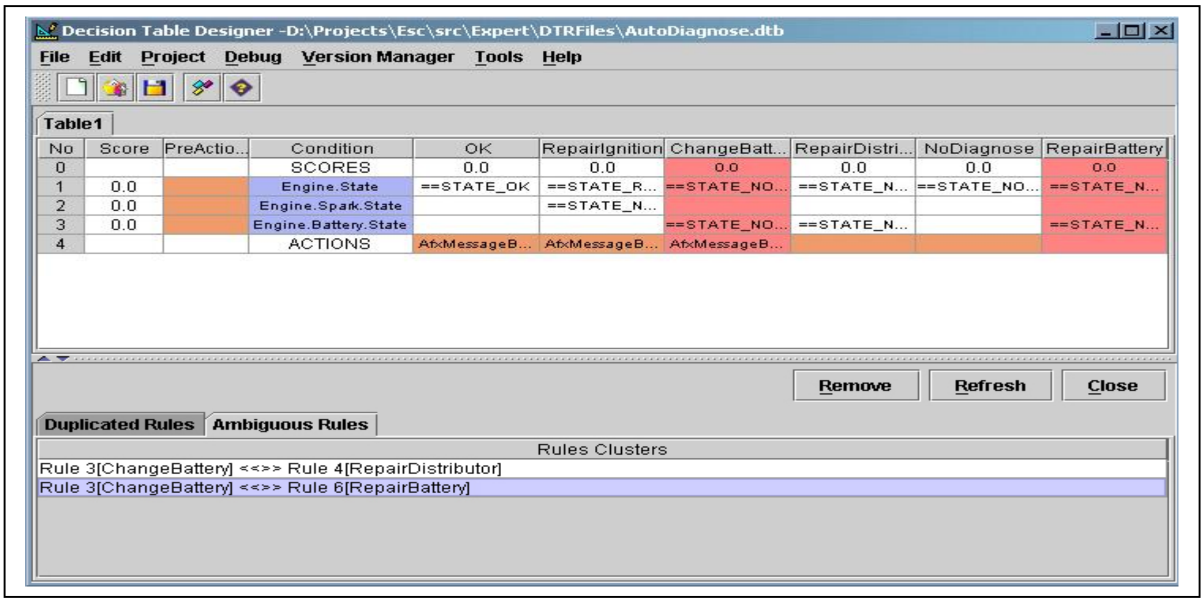
where N – the set of nodes– is represented by system's rules, and A is the set of arcs. An arc is defined with source N1 and target N2 if the consequent of the rule N1 asserts a fact that appears in the antecedent of the rule N2.

The visual representation of this graph reveals main or isolated rules and templates. The graphical representation is a better approach to computing various numeric metrics that measure the quality of system's design. For users confident to numbers, a set of base metrics is also computed.

Despite CLIPS's age, there are no integrated development environments offering "standard" debugging techniques, like step-by-step execution or breakpoints management for it. Decision Frame Debugger implements these debugging techniques by means of an easy-to-use, visual user interface. The Decision Frame Debugger includes the following features: rule-by-rule execution, breakpoints management, step into rule RHS's actions (procedural debugging), variables inspection, display of facts and agenda memories. It supports both CLIPS and JESS inference engines. In debugging mode, the system is automatically executed in a rule-by-rule manner, stopping on each breakpoint. In case of using CLIPS inference engine, Java Native Interface (JNI) technology is used for bridging between Decision Frame Debugger (Java environment) and CLIPS engine (native environment). The communication between the Debugger and the inference engine is described by some general interfaces. For the moment, CLIPS and JESS interface instances are implemented, but more inference engines can be easily plugged in (see Fig. 2).



Elf t qd 1- Cdbhr hmmEq` l d Cdat f f dq


Decision Table Designer -D:\Projects\Esc\src\Expert\DTRFiles\AutoDiagnose.dtb
File Edit Project Debug Version Manager Tools Help
Table1
Remove
Refresh
Close
Duplicated Rules
Ambiguous Rules

No	Score	PreActio...	Condition	OK	RepairIgnition	ChangeBatt...	RepairDistri...	NoDiagnose	RepairBattery
0			SCORES	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0		Engine.State	==STATE_OK	==STATE_R...	==STATE_NO...	==STATE_N...	==STATE_NO...	==STATE_N...
2	0.0		Engine.Spark.State		==STATE_N...				
3	0.0		Engine.Battery.State			==STATE_NO...	==STATE_N...		==STATE_N...
4			ACTIONS	AfxMessageB...	AfxMessageB...	AfxMessageB...			

Rules Clusters
Rule 3[ChangeBattery] <<>> Rule 4[RepairDistributor]
Rule 3[ChangeBattery] <<>> Rule 6[RepairBattery]

Elf t qd 2- Cdbhr hmmS` ald @m` kxydq

In order to find the time-consuming rules, *rule-based profiler* traces the system's execution . The system records the execution context in the trace files and using the Trace Viewer these files are visualized. In case of rules set system, the execution context is formed by the antecedent and the consequent of the executed rule.

3.2 Decision table

The automatically check of the correctness and completeness of the decision table is carried out by the Table Analyzer tool (embedded in Decision Table Designer), which highlights the duplicated and ambiguous rules that exist in the table. Fig. 3 shows two ambiguous rules (the 3rd and 6th rules from left to right) in the sample decision table. To measure the completeness of a decision table, the *completion ratio (CR)* is computed as follows:

CR = [Possible Rules] / [Actual Rules]

where [Possible Rules] is the number of possible rules and [Actual Rules] is the number of rules of the decision table. The number of possible rules is computed as the product of cardinalities of all attribute domains.

The Decision Table Debugger lets you debug the system as a decision table. You can set breakpoints on table's cell that will stop the execution of the system. While the execution is paused, you can inspect variables status. To stop the system's execution on a breakpoint, the Code Generator module generates additional Java/C++ code for each table's cell. Before the execution of a Java/C++ statement in the "host" project, the Decision Table Debugger is interrogated whether or not a breakpoint is hit. If a breakpoint is hit, the execution control is passed to the Expert System Creator thread and the current values of all watched variables are updated. When the user continues the program execution (from Decision Table Debugger), the control is regained by the host project thread and the watch variables' values (possibly modified by the user during the debugging session in Decision Table Debugger) are sent back to the host project.

3.3 Classification tree

For a better highlighting of the rules induced by a classification tree, the Decision Tree Designer displays all the rules encapsulated in the tree in a distinct panel. It also offers statistics regarding the number of nodes in tree, the number of nodes in each category (decision nodes, action nodes, leaf nodes), the number of induced rules, the number of incomplete decision nodes etc.

The Decision Tree Debugger module offers the possibility to debug the expert system in the classification tree form. Similar to the decision table debugging, the Code Generator module generates additional code for each tree's node. See the section 3.2 for details regarding the communication between host project and Decision Tree Debugger and for control of the execution as well.

In order to find out the time-consuming rules, the Code Generator module optionally generates addition code for tracing the host program execution. During a "traced" execution of the host program a *trace file* is created. It contains the execution context for each visited node. The execution context is composed of the timestamp and variables' values. The trace file is visualized by the Trace Viewer module that embosses the "bottlenecks" nodes.

4. Final remarks and future extensions

Expert System Creator is a cross-platform development environment that significantly cuts the cost of knowledge base systems development.

The system is entirely implemented in Java using Java2™ SDK 1.3. The Decision Frame module supports CLIPS 6.1 and JESS 6 expert system shells [5] that perform the knowledge-based reasoning process. The Code Generator outputs C++ and Java code for decision tables and trees, whilst the rule-based systems are generated using CLIPS/JESS syntax.

Further extensions are planned to integrate fuzzy logic engines such as FuzzyCLIPS [8] and FuzzyJ [9]. The automatic growth of classification trees from large data sets [6] is another important feature to be added in the near future.

Acknowledgements

This work was partially supported by: the Romanian Government's CNCSIS grant no. 564/2003, by the Austrian Ministries BMBWK project no. GZ 45.527/1-VI/B/7a/02 and BMWA project GZ no. 98.244/1-I/18/02. Our industry partner was represented by Optimal Solution Software [7] a leader of the expert systems development market.

References

[1] Colomb R.M.Representation of Propositional Expert Systems as Partial Functions, Artificial Intelligence 1999, 109: 187-209.

[2] Culbert C, Riley G, Donnell B. CLIPS Reference Manual, Volume 1-3, Johnson Space Center NASA; 1993.

[3] Dial R.B. Decision Table Translation, Collected algorithms from CACM, 1970.

[4] Far B.H, Takizawa T, Koono Z. Software Creation: An SDL-Based Expert System for Automatic Software Design. In Faergemand O, Sarma A. editors. Proceedings of SDL '93; Elsevier Publishing Co, North-Holland, 1993; p. 399-410.

[5] Friedman-Hill E. JESS: The Rule Engine for the Java Platform. http://herzberg.ca.sandia.gov/jess [3/11/2002]

[6] Gehrke J, Ganti V, Ramakrishnan R, Wei-Yin Loh. BOAT-Optimistic Decision Tree Construction, Proceedings of SIGMOD Conference 1999, 169-180.

[7] Optimal Solution web site. http://www.optsol.at [3/21/2002]

[8] Orchard R.A. FuzzyCLIPS User's Guide, Integrated Reasoing Institute for Information Technology National Research Council Canada; 1998.

[9] Orchard R.A. NRC FuzzyJ Toolkit for the Java™ Platform. User's Guide; 2001. http://www.iit.nrc.ca/IR_public/fuzzy/fuzzyJDocs [3/11/2002]

[10] Pop D. and Negru V. Intertranslability of Representation Forms in Classification, Proceedings of 10th SINTES Conference; 2000 May; Craiova, Romania.

[11] Pop D. and Negru V. Expert System Creator: A Visual Tool For Expert Systems Construction. Proceedings of the 24th Intl. Conference on Information Technology Interfaces ITI; 2002 June; Cavtat, Croatia

[12] Pop D. and Negru V. Knowledge Management in Expert System Creator. LNCS/LNAI 2443, Springer, 2002, pp. 233 – 242

[13] Protégé-2000. http://protégé.stanford.edu/index.html [2/15/2003]

[14] Quinlan J.R. Introduction of Decision Trees, Machine Learning 1; 1984.

[15] Riley G, Donnell B. CLIPS Arhitecture Manual, Johnson Space Center NASA; 1993.

[16] Shwayder K. Conversion of limited-entry decision tables to computer programs - a proposed modification to Pollack's algorithm. Communications of the ACM 14, pp. 69-73, 1971.

[17] Witten I.H, Frank E. Data Mining. Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufman Publishers; 2000.