

An Extensible Environment for Expert System Development

Daniel Pop, Viorel Negru

Department of Computer Science, University of the West from Timișoara
4 V. Pârvan Street, RO-1900 Timișoara, Romania
E-mail: {popica, vnegru}@info.uvt.ro

Abstract. The use of expert systems in the speed-up of human professional work has been in two orders of magnitude with resulting increases in human productivity and financial returns. Last decade shows that a growing number of organizations shift their informational systems towards a knowledge-based approach. This fact generates the need for new tools and environments that intelligently port the legacy systems in modern, extensible and scalable knowledge-integrated systems. This paper presents an extensible, user-friendly environment for expert system development, which supports the integration of high-level knowledge “beans” into host projects, data integration from conventional database systems and system’s verification, debugging and profiling

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. The re-engineering of old, existing information systems and their transformation in modern, extensible, scalable, viable systems is a complex and tedious process involving significant costs and resources. Expert System Creator suite was created as a joint project between academic and commercial partners. Its central goal is to help professional in the process of shifting from old implementation to modern approaches, based on latest technologies. It assists the human designer by efficient encoding of expert knowledge and by reusing the available systems. Expert System Creator is a development and integration environment for knowledge management, expert system construction and validation, and database integration. It merges conventional CASE tool facilities with the expert system technology.

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 translation between different knowledge representation forms nor the debugging, verification and profiling phases, in contrast to the Expert System Creator suite.

Another powerful knowledge management tool is Protégé-2000 project from Stanford Medical Informatics [10], 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.

People involved in different phases of knowledge-based system construction or in the re-engineering of existing informational systems have different computing skills and carry out different tasks. Therefore, a viable environment must provide appropriate tools for different categories of users, ranging from domain experts to software designers and programmers.

Expert System Creator is a system for the development, verification and debugging, profiling and optimization of expert system applications. Its main facilities include:

- representation of domain knowledge using rules set, decision tables or classification trees;
- supports the system design phase using active graphical widgets;
- computer-aided system verification;
- automatic code generation for declarative, functional or object-oriented programming languages;
- integration with external systems;
- integration with relational database management systems;
- visual debugging and tracing of expert systems using the high-level representation (decision tables, classification trees).

The next section presents how Expert System Creator helps domain experts and software developers to port their old informational systems to knowledge-based approaches. In the third section are outlined techniques and tools for system's validation, verification, debugging and profiling. The last section presents some final remarks, as well as the foreseen extensions.

2 Reengineering and Building Phase

Re-engineering is the first phase in implementing a knowledge-based system from an existing informational system. In order to import the legacy code, a *Dictionary Manager* module was introduced in Expert System Creator's architecture. A dictionary is a collection of external data, user-defined data types and procedures imported from external projects, i.e.: classes, structures, interfaces, type definitions, enumerations, variables, constants and class instances, functions and procedures. The Dictionary Manager module imports C, C++, Java and CLIPS/JESS definition files. New parsers and extensions can be easily added as plug-ins.

All imported elements can be used in constructing the higher-level decision objects. The domain experts will operate with graphical representations of different knowledge forms, regardless of their implementation details, such as programming paradigm or language.

2.1 Extensions of the Standard Knowledge Forms

For the representation of knowledge in expert systems, 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. Each representation has its advantages and drawbacks. Expert System Creator is endowed with advanced graphical tools for three of the above forms: decision table, classification tree and rules set. As these three forms are equivalent [9], an expert system built in one of them can be translated into any other one. In order to overcome the limitations of the “standard” decision table and tree representation, we have introduced several extensions to the basic models: pre-actions, intermediate-actions and scoring. These will be presented in this section.

The Decision Frame Designer handles the *rule-based* expert systems. The most known shells for rule-based expert systems are CLIPS (C Language Integrated Production System) [2] and JESS (Java Expert System Shell) [5]. Both shells are integrated in the Decision Frame Designer and users can switch from one to another at any moment. The *Decision Frame Designer* [9] is a project-based, user-friendly application for the development, verification, debugging and profiling of rule-based systems. A rule-based expert system can be integrated in C++/Java host projects. The calling code is automatically generated by the Decision Frame Designer.

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). We are proposing an extended model for the decision table. Each condition has associated two new concepts: pre-actions and score. A pre-action denotes a specific action that is executed before the condition is evaluated. Each table’s condition (column) has associated a list of pre-actions that is executed prior to condition’s evaluation. A pre-action may be represented by variable initialization, user input/output handling etc. The *Decision Table Designer* let users to construct a decision table object in a visual way.

A *classification tree* consists of a set of nodes and a set of arcs [11]. The set of nodes is divided into three classes: decision nodes, intermediate-action nodes and classification 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. The standard decision tree model was extended with a new type of nodes – intermediate-action nodes – that let users to specify actions to be executed before the decision node’s evaluation. The semantic of intermediate-action is similar to decision table’s pre-action. The *Decision Tree Designer* offers the possibility to develop classification trees, providing a structured designer interface that let users to group tree’s nodes at each level, thus saving design space.

To support inexact reasoning, the decision table/tree were enhanced with scores. A *score*, represented by a real number between 0 and 1, is attached to each condition of a decision table/tree. The score is a measure of user’s confidence in the specific condition. The final rule’s score (confidence) is obtained by combining the

conditions' scores. Various combination functions can be implemented, but for now, experiments using cumulative and multiplicative functions have been conducted.

2.2 Database Integration

A large number of legacy systems underlies on relational databases systems. To access and transform available data of this systems, Expert System Creator offers direct database access for all three forms: rules set, decision table and classification tree.

In the case of decision tables and classification trees, users can use their preferred database access libraries by importing them in the dictionary component. Using database access functions from within the constructed decision table or classification tree is straightforward and requires no specific handling.

In the case of rules set (or decision frames), the problem of reasoning on facts residing in conventional relational database systems requires more attention. A major objective of database integration is to provide independence of both inference engine and DBMS. The Decision Frame DataBase [9] is an independent subsystem that acts as a communication channel between one or more database systems and a rule-based expert system.

3 Verification, Validation and Debugging Phases

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.

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.

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

$$SG = (N, A) \quad (1)$$

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.

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. To measure the completeness of a decision table, the *completion ratio (CR)* is computed as follows:

$$CR = [\text{Possible Rules}] / [\text{Actual Rules}] \quad (2)$$

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 portable development environment that significantly cuts the cost of porting legacy systems to new technologies. The system is entirely implemented in Java using Java2™ SDK 1.3. The Decision Frame module works together with CLIPS [2][12] or JESS [5] expert system shells that perform the knowledge-based reasoning process. The Code Generator outputs C/C++ and Java code for decision tables and trees, whilst the rule-based systems are generated using CLIPS/JESS syntax.

Two main directions are foreseen for further developments: enhanced knowledge representation and support for automatic project documentation generation. The first direction will be supported by several enhancements in Expert System Creator, such as: the integration of fuzzy logic engines (like FuzzyCLIPS [7] or FuzzyJ [8]), employing advanced widgets supporting the knowledge acquisition phase, and automatic tree growing from large data sets. The second direction will be supported by integrating intelligent reporting tools available on the market. Its aim is to generate the system documentation based on the overall architecture of the developed expert system, using the user's comments and implementation code inserted in frames, tables or trees projects.

Acknowledgements

This project is a joint effort of both academia and industry partners. The University of the West from Timisoara supported this project through the Romanian Government's

INFOSOC grant no. 61/2002 and CNCSIS grant no. 564/2002. Our industry partner is represented by Optimal Solution Software [6].

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, Vol. 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) 399-410
5. Friedman-Hill, E.: JESS: The Rule Engine for the Java Platform. <http://herzberg.ca.sandia.gov/jess> [3/11/2002]
6. Optimal Solution web site. <http://www.optsol.at> [3/21/2002]
7. Orchard, R., A.: FuzzyCLIPS User's Guide, Integrated Reasoning Institute for Information Technology National Research Council Canada (1998)
8. 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]
9. Pop, D., Negru, V.: Knowledge Management in Expert System Creator. LNCS/LNAI 2443, Springer (2002) 233 – 242
10. Protégé-2000. <http://protégé.stanford.edu/index.html> [2/15/2003]
11. Quinlan, J., R.: Introduction of Decision Trees. *Machine Learning* 1 (1984)
12. Riley, G., Donnell, B.: CLIPS Architecture Manual. Johnson Space Center NASA (1993)
13. Shwayder, K.: Conversion of limited-entry decision tables to computer programs - a proposed modification to Pollack's algorithm. *Communications of the ACM* 14 (1971) 69-73
14. Witten, I., H., Frank, E.: Data Mining. *Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufman Publishers (2000)