ソフトウェア工学 64－23
（1989. 2. 3）

交換ソフトのデバッグエキスパートシステム

赤尾由香利セシリア* 今井恵一** 土田賢省*
*ソフトウェア生産技術開発本部 **交換第一ネットワークシステム事業部
日本電気(株)

デバッグエキスパートシステムは交換系ソフトウェアのデバッグと診断を行う。本システムは、専門家およびデバッグを行う利用者との共同のもとに、両者の要求を満たし、人のやり方に沿うように、即ちシステムの振舞いが人の自然な考え方に近くなることを目指して、開発されたものである。システムでは、このような要件を満たすために、新たに導入した初期情報と不確定情報の扱いや、決定木とルールの相互関係を扱う機能を備えている。本稿では、システムの特徴および導入した概念の根拠と説明を含めて、デバッグエキスパートシステムの機構について述べる。

Debug Expert System for Switching Systems Software

*Cecilia Yukari AKAO**, *Keiichi IMAI***, *Kensei TSUCHIDA**
* Software Engineering Development Laboratory ** First Switching Network Systems Division
2-11-5, Shibaura, Minato-ku, Tokyo, 108 JAPAN 1131, Hinode, Abiko, Chiba, 270-11 JAPAN
NEC Corporation

ABSTRACT

The Debug Expert System is a system that debugs and diagnoses switching systems software. It was developed together with experts and users who debug switching systems software, in the tentative of satisfying requirements of both and of building a system that comports in the same way as humans, i.e., to approximate the system with the natural way of human thinking. For this, new concepts such as *initial information* and *unknown treatment* are introduced in this paper. There is also a correlation between decision trees and rules. This paper describes the architecture of the Debug Expert System, including a description of its features and justifications and explanations of all concepts introduced here.

1. Introduction

The objective of the *Debug Expert System (DBES)* is to debug and diagnose errors and malfunctions that occur on the testing phase of switching systems software.

In switching systems, almost all the software available is extremely large and complicated. Besides the telephone exchange systems, the coming generation of switching systems have higher expectations. They must be real time systems because immediate response time is required. They must be multi-process, allowing high concurrency to solve numerous tasks quickly. Also, they must offer continuity of services, as switching services do not allow interruptions, even when some trouble occurs. So, the structure of switching systems became complex, and concomitantly, when building a switching system, its testing phase became both complex and exhaustive. Therefore, the need for a powerful system that gives support in the testing phase of such systems arose. In response to this expectation, the Debug Expert System was developed.

In the diagnosis and debugging domains, almost all the Expert Systems (ES) available are rule-based systems based on symptom-cause rules ([Shortliffe76]). However, as knowledge of switching systems has many peculiarities and is very complex, there is a necessity of introducing features and tools to the ES, so that consultants can deal with the knowledge of ES smoothly.

In the tentative to approximate the system with the natural way of working of the user, DBES introduces new concepts, such as *unknown treatment* which deals with information which veracity the consultant does not know or which meaning the consultant does not understand, and *auxiliary information*, which gives complementary information regarding the debugging environment.

Since knowledge of the system was extracted from a decision tree, and because knowledge represented as tree structures is more easily understood by consultants not familiar with production rules, features that transform rules into decision trees and an editor of rules from a decision tree are also implemented.

Section 2 explains the difficulties of the knowledge domain. Section 3 describes the DBES. The following sections describe the new concepts, together with a detailed description of the system.

2. Knowledge Acquisition

Switching systems cover a very wide domain of knowledge. Experts are expected to know not only about switching domains, but also about OS, multi-processors, hardware, etc. And it is not rare that, given some problem to different experts, each expert interprets and solves this problem in a different way of other experts. This is understandable since there are many possible alternatives that can explain some bug. Sometimes one cannot say exactly which solution will solve the problem and sometimes two or more alternatives must be combined to find a suitable solution.

To obtain the most complete and confident knowledge data, it is obvious that knowledge should not be extracted from only one expert. Knowledge of the DBES was elaborated with the cooperation of three experts. Although many discussions have arisen around each theme that was being analyzed during the construction of the knowledge data (because each conclusion must have the consensus of all the experts, and this does not always happen), we believe that these discussions were very important because they lead to the confection of a knowledge data that has the consensus of all the experts, and so, with more consistency. But at the same time, there is always a risk of building a knowledge data that, for some bugs, does not offer the most efficient inference, because perhaps some skill of one of the experts was not accepted by the others.

Experts are not interested in debugging switching systems in its coding level, i.e., in the syntax and in the analysis of the structure of the programs that compound the switching systems software. They are interested in the switching systems functions, services and interrelations between all services and functions. Therefore, DBES contains descriptions of all functions and services and how they interact, testing methods to determine where malfunctions exist and information about how to correct these malfunctions.

To the prototype, we have only implemented knowledge that debugs the *Multi-Media Service System - MMS System*, a subsystem of the switching system.

Knowledge was represented in the form of a decision tree and its fragment is shown below. Note that the original knowledge data was extracted in Japanese.

```
1- Inspect the lamp of the MMS
 1-1-HLT OFF
    Dump the number of the NCU and verify the
    terminating call line
    1-1-1-NCU number is 15
          Dump the NLC of the SCU and the trace data
       1-1-1-1-The status is 0 and there is trace
               The signal was ignored. Is the SIG
               of the trace data noticing
               terminating call?
            1-1-1-1-1-Yes...
            1-1-1-1-2-No...
       1-1-1-2-The status is 0 and there is no trace
               Check the SEND parameter of the NLC
            1-1-1-2-1-NG...
            1-1-1-2-2-OK...
       ....
       1-1-1-6-The status is not 0, 5, 6, 10 or 11.
    1-1-2-NCU number is 16 ...
    ....
    1-1-5-NCU number is not 0, 15, 16 or 17
 1-2-HLT ON
    Verify where does it fail using the trace data
    1-2-1-APL
          Where does it down?
       1-2-1-1-SVC
             1-2-1-1-1-The parameter list...
       ...
       1-2-1-3-Logic ...
    1-2-2-Driver ...
    1-2-3-OS ...
```

Fig 1: A fragment of knowledge extracted. Numbers of the nodes are altered for convenience.

3. The DBES

This section gives a brief description of the system.

The DBES tries to debug errors and malfunctions using information provided by the consultant during the session and information from the system itself. Knowledge of the DBES is stored in the form of a set of rules. The set of rules contains knowledge about the debugging environment and the inference of the system is made using the rules of this set. Information that does not interfere in the inference is stored in separated files.

The inference engine is based on the production systems. Briefly speaking, the system asks a series of questions to the consultant. Each question is followed by a series of alternatives. Each alternative is a potential answer of that question. When the consultant chooses a given alternative, this information is stored in the working memory and after a process of pattern matching, a new question is asked to the consultant. When enough information is collected by the system, it then displays a message explaining the cause of the error and suggesting what the consultant must do to correct the bug.

Besides the usual inference, the system has a special mechanism that allows the manipulation of doubtful information. Here this information is called *unknown information* and the mechanism that deals with this information, including the mechanism that backtracks the unknown information (*feedback unknown*), is called *unknown treatment mechanism.*

The inference engine and the unknown treatment mechanism are the principals of this system, and we'll call it *inference mechanism.*

The following are many of the options available that are features and tools of the system that help the consultant to use the system.

- *see dynamic memory* - displays all dynamic data available.
- *initial information* - allows the consultant to inform, before the inference starts, all information that is known as certain facts.
- *mark information* - marks some information that in the following inferences will be considered as true facts (this information is added to the *initial information*).

- *take off work memory* - erases information from the working memory.
- *more information* - displays some auxiliary information that does not belong to the inference mechanism.
- *show conclusions* - displays all the possible conclusions that can be taken.
- *tree structure* - displays the rule database(static memory) in the form of a tree structure.
- *editor of rules* - allows the alteration of the knowledge data. It can be done during the inference or separately of it.

Almost all the options described above are available by activating the *help menu* of the system.

4. Giving complementary information

It was shown that, in some cases, the consultant needs some help to interpret the system's message or wants a more detailed explanation of some question or conclusion provided by the system. In these cases, the consultant only needs to select the option *more information* and then the system displays more detailed information about the contents of the rule or about the debugging environment.

Information displayed is called *auxiliary information*, or simply *more information*. Any kind of information may be displayed. For example, the system can display messages and tables. If no auxiliary information is available to some rule, a message that there is no information is displayed.

The advantages of the *auxiliary information* are that it does not interfere in the inference mechanism and it supplies the consultant with more information, advising what the consultant must do, explaining the structure and components of switching systems and complementing consultant's knowledge.

5. The unknown treatment mechanism

Switching software contains many technical words and has many controls to manipulate. So, sometimes the consultant may not understand the meaning of some questions made by the system. And sometimes the consultant may not know the correct answer of the question being asked by the system. There will also be times when, although the consultant understands the meaning of the question and its alternatives, and although the consultant can verify which is the correct answer, the consultant judges that verifying the correctness of some answer will waste time, so that choosing the most probable answer will be more efficient

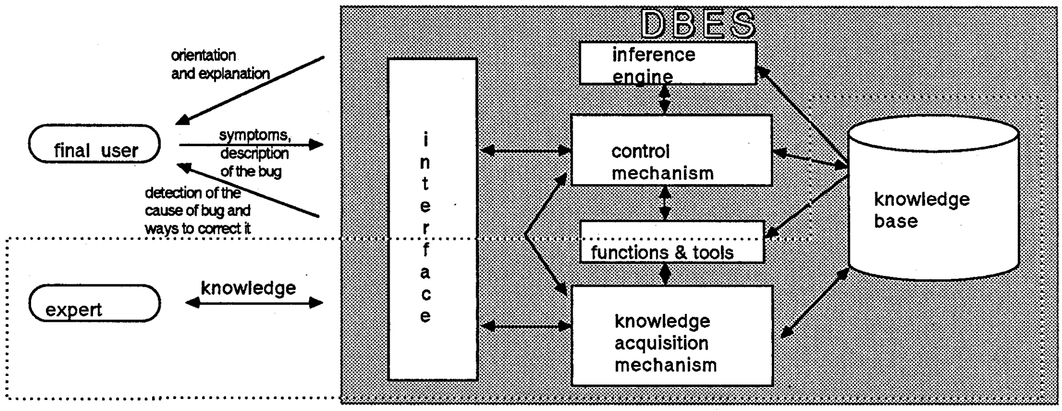
DBES
orientation and explanation
final user
symptoms, description of the bug
detection of the cause of bug and ways to correct it
expert
knowledge
interface
inference engine
control mechanism
functions & tools
knowledge acquisition mechanism
knowledge base

Fig 2: The DBES architecture

than choosing the correct one.

In these cases, the system allows the consultant to choose the option 'unknown' which informs that the answer to a question may not be correct. This question is called *unknown question.* Then the system re-asks the unknown question, asking the consultant to select the most probable answer from among the alternatives. It is called *unknown information.* This unknown information is stored in the *unknown memory* and in the working memory and the inference continues normally, considering this unknown information as a real fact. Note that if the consultant doesn't know which is the most probable alternative, then the consultant is advised to select, from top to bottom, the alternatives displayed.

When the consultant notes that inference is strange, or that inference tended to a conclusion that is not satisfactory, then the unknown information selected before is probably the cause of the deviation. So, the system allows the consultant to re-answer the unknown question, using the option *feedback unknown.* When the consultant selects this option, the unknown question is re-asked, and new information is required to be selected. This new option may or may not be *unknown information.* Once this new information is selected, then inference continues from this point.

More than one *unknown information* may be selected by the consultant. The system deals with this unknown information using a stack structure. Each unknown information is stored on the top of the unknown memory. When the option *feedback unknown* is chosen, the last unknown information stored in the unknown memory is deleted from this memory and the unknown question related to this unknown information is re-asked. And so the inference goes on.

Another memory, called *ex-unknown memory* is also used, and it contains all information that once was unknown information. So, when the option feedback of unknown information is activated, the information on the top of the unknown memory is stored in the ex-unknown memory. This memory is used to remind the consultant of all unknown information that was backtracked.

See figure 3 to understand the mechanism described here.

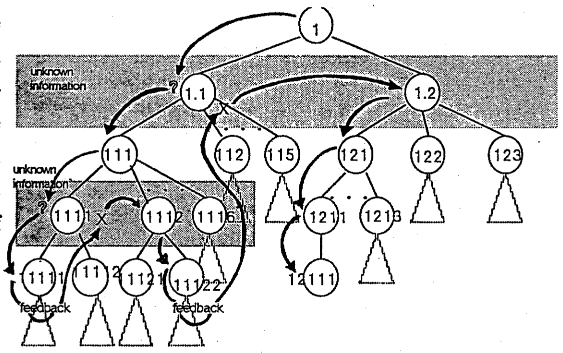

Fig 3: For clarity, we'll explain the unknown mechanism *using a tree represented in Fig 1. The arrows symbolize the inference. Question marks symbolize* unknown information. *X marks represent information erased from the working memory when option* feedback of unknown *is activated.*

Note that facts selected by the consultant after some unknown option is chosen are considered as certain facts, except when the option unknown is again selected, and they are not deleted from the working memory when the option *feedback unknown* is activated. Therefore, these facts will be certain facts during the whole consultation. If the consultant is not sure whether or not the information that comes after some unknown information is correct, the consultant must select the option 'unknown' again, and identify it as unknown information.

6. 'Initial information' and 'mark information'

There are times when the consultant already knows some facts about the debugging environment. And there are times when some facts do not alter while debugging some specific field of the debugging environment. These facts are called here *initial information* and they can be initialized before the inference starts.

When beginning the consultation, the DBES first shows a menu of options, called here simply as *first table*. All the data in the first table is stored in the *initial information data base*. Each option is a description of some status or is some information regarding the debugging environment. The consultant is asked to select any number of options from this table that are to be considered as true facts. All facts selected are stored in the *initial information memory* and when beginning each consultation, the facts stored in this memory are loaded into the working memory. Then the system starts the inference. So, information selected from the first table is used in the inference.

In many cases, however, the consultant will only realize options that should be *initial information* when they appear during the inference, i.e., when the consultant is requested to inform some fact during the analysis he/she is doing, he/she recognizes that this fact will be always true. So, the consultant can *mark* this information so that it will be stored in the *initial information memory* and be used as an *initial information* during subsequent consultations.

The *initial information* can be used in subsequent consultations, until alterations are required. This solves the annoyance of having to input consecutively information that does not vary while debugging a specific field.

7. Seeing the routes of the inference

During the inference, there will be times when the consultant wants to see, without continuing the inference, the possible steps that follow the inference done till now, i.e., using a tree, the consultant wants to see the descendants of some node in order to certify that the consultant is not deviating from the real cause of the bug or to analyze the possible alternatives to decide which answer is the best for some question of the DBES. Two options are available in the DBES that deal with this problem. Let us see them now.

7.1. Displaying Final Conclusions

There are times when the consultant is not interested in the inference but in the conclusions that some bug may lead to. This is very often done by experienced users of switching systems software, and from the conclusions displayed, the user selects the one that fits the bug (or the one that the user judges so) and corrects it. Therefore, the steps of inference are ignored here. Obviously, the user may select a false conclusion. But in almost all cases, experienced users can, looking at all the final conclusions, determine what the most probable cause of the bug is.

The *show conclusions* option displays all the possible conclusions that can be achieved from the point of inference where this option is invoked.

7.2. Rule-based data represented in Decision Tree structure

It is clearly obvious that interpretation of a decision tree is easier than interpretation of a set of rules, especially when the number of nodes or rules is large.

Given some rule, the *tree structure* option displays all the productions that appears in the derivation of this rule, represented as a decision tree. The consultant can see, using this option, all the possible derivations from the rule, and judge whether or not to continue the inference.

This option is also important while editing rules, because it can organize knowledge inputed as rules. For example, looking at the tree, one can determine which rules are missing and/or which rules cannot be derived because of the lack of information, because in the tree these rules are displayed in different colors.

8. The 'see dynamic memory' option

If the consultant wants to know all the information already available, i.e., if the consultant wants to see the dynamic data, the consultant only needs to choose the option *see dynamic memory* from the Help Menu.

Note that the working memory contains certain facts and unknown facts. Below is a brief explanation of each of them.

1) Working memory - contains all facts already

available(known + unknown).

2) Unknown memory - contains only unknown information.
3) Ex_unknown memory - contains information that was unknown, i.e., in the process of feedback, unknown information deleted from the unknown memory is stored in the ex-unknown memory.
4) Deleted_memory - contains information that was removed intentionally by the consultant from the working memory.
5) Initial_memory - contains *initial information*. It can be thought as a subset of the *working memory.*

9. The editor of rules

This editor accepts knowledge represented in decision trees and transforms it into knowledge represented as a set of rules. Note that this editor may also accept knowledge already represented as a set of rules.

The principal objective of this editor is to facilitate the experts' work in representing their knowledge. For this, many studies about which knowledge is used and how knowledge is used have been done to built a more powerful editor that helps experts' tasks in elaborating knowledge data.

10. Conclusion

The prototype system is already developed and members of the First Switching Network Systems Division are using this system experimentally. Experts and non-experts are using the system. Knowledge data covers only some specific fields of the MMS Systems, and, in this area, the system has proved satisfactory, but there is a need for more rules to cover all fields of switching systems software.

The concepts of *unknown treatment* and *initial information* have proved very useful, as well as the mechanisms that display *final conclusions* and rules represented as tree structures. Besides these features, experts have now asked for a feature that creates a history of all bugs detected, showing all bugs that occurred, their frequency and the ways used to correct these bugs.

One characteristic of the DBES is its simplicity of manipulation. The consultant has no trouble in learning how to manipulate the DBES. A new consultant will need about 10-30 minutes to understand the inference mechanism of the DBES and a few hours to understand and use fully all the mechanisms and the tools of the DBES.

Based on the prototype developed and on their experience, experts estimate that when more rules are added to the knowledge data, DBES will be able to detect 60-70% of the bugs that occur in switching systems software. Bugs not covered by the DBES are those that have never occurred and those which are so complex that even experts can not predict them.

Time spent in debugging tasks will be 40-50% shorter when using the DBES. And by using the DBES, novices will also be able to debug errors, and so, the complex task of debugging will be shared among experts, experienced users and novices alike, thus reducing the experts' overload.

Experts agree that the DBES is not only a system that performs debugging tasks. DBES may also be used in other domains, such as in switching systems design, giving support in the process of the design of such systems.

Our future work is to enhance the DBES, allowing an efficient manipulation of a large amount of knowledge data. We would also like to construct a more powerful editor that orients and aids experts' work in transferring their knowledge to the DBES and that allows the modification of knowledge data in real time.

Acknowledgement. We would like to thank all persons who helped us in the elaboration of the expert knowledge data, specially *Kousei KAKIHARA* and *Yasushi MITSUGI.* We are also grateful to all other members of the Software Engineering Development Laboratory and the First Switching Network Systems Division.

REFERENCES

[1] Brownston L., Kant E., Farrell R., Martin N., *Programming Expert Systems in OPS5 - An introduction to Rule-based Programming*, Addison-Wesley, Reading, MA, 1985.

[2] Cunningham, P., Brady, M., *Qualitative Reasoning in Electronic Fault Diagnosis*, IJCAI'87, pp443-445, 1987.

[3] Davis, R., Lenat, D.B., *Knowledge-based systems in Artificial Intelligence* , McGraw-Hill, 1982.

[4] Hayes-Roth F., Waterman D.A., Lenat D.B., *Building Expert Systems*, Addison-Wesley, Reading, MA, 1983.

[5] Pepper, J., Kahn, G.S., *Repair Strategies in a Diagnostic Expert System*, IJCAI'87, pp531-534, 1987.

[6] Quinlan, J.R., Generating Production Rules from Decision Trees, Knowledge Acquisition, IJCAI'87, pp304-307, 1987.

[7] Shortliffe E.H., Buchanan B.G., Rule-based Expert Systems, Addison-Wesley, Reading, MA, 1984.

[8] Shortliffe E.H., *MYCIN: Computer-based Medical Consultations*, Elsevier, NY, 1976.

[9] Winston, Patrick Henry, *Artificial Intelligence*, Second Edition, Addison-Wesley, Reading, MA, 1984.