ELSEVIER

Decision Support Systems 18 (1996) 107-115

Decision Support Systems

INTERFACER: A user interface tool for interactive expert-systems

Shigeo Kaneda [a,*], Megumi Ishii [a], Fumio Hattori [b], Tsukasa Kawaoka [a]

[a] *NTT Communication Science Laboratories, 1-2356 Take, Yokosuka-shi, Kanagawa-ken, 238-03, Japan*

[b] *NTT Information and Communication Systems Laboratories, 1-2356 Take, Yokosuka-shi, Kanagawa-ken, 238-03, Japan*

Abstract

From the user interface point of view, expert-systems are different from conventional applications in some features. First, the user query sequence highly depends upon input data up to that time. Second, any change in query sequence requires highly complicated data modification routines. Thus, user interface implementation is a bottleneck in the same manner as knowledge acquisition is the bottleneck for expert-systems. To resolve this problem, this paper proposes the user interface tool "INTERFACER" for interactive expert-systems. INTERFACER automatically generates a user interface screen according to the data input query requirement from the inference engine, and requires no user data modification routines in expert-system development. We applied the proposed INTERFACER to a practical middle-scale business system: The General Employee Affairs Expert-system. Program amount was decreased 50% compared to the conventional procedural implementation.

Keywords: User interface; Expert system; Knowledge based system; Data modification

1. Introduction

Interactive expert-systems, which generate many user queries for problem solving, are practical and are being developed for diagnostic or analytical applications [3]. This type of expert-system usually has a sophisticated user interface to get data values from the user. An inference engine generates a conclusion from the input data and the knowledge base.

In general, the user interface is a major part of the application software in terms of program size, and it is often desirable to treat the user interface as a separate structure to be designed and managed by a User Interface Management System (UIMS). This separation provides many advantages for software development: decrease in program amount, easy modification of user interface, and reuse of software modules [1,2].

From the user interface point of view, however, interactive expert-systems are different from conventional applications in some features [7,8]. In this paper, we focus on the feature that the user query sequence depends upon the input data up to that time. This means that the expert-system's query item is dynamically generated at runtime by the inference engine using the knowledge and the data input up to that time.

This dynamic sequence change results in two major difficulties in implementing a user interface of an interactive expert-system. First, the sequence of

[*] Corresponding author.

data items on a UI screen cannot be determined when designing the expert-system. Second, the dynamic change of query sequence requires highly complicated data modification routines to offer flexible data modification function on the UI screen for the user. As the result, user interface implementation is a bottleneck for expert-system development.

To resolve the bottleneck of user interface implementation, this paper proposes the user interface tool "INTERFACER" for interactive expert-systems. The INTERFACER automatically generates a UI screen using pre-defined menus/dialog-boxes in accordance with query demands, under control of the inference engine. Also, a notable feature of INTERFACER is its internal re-execution function which is triggered when an expert-system user modifies a datum on an UI screen. During re-execution, the current INTERFACER keeps the old UI screen and automatically responds to the same input requirements generated by the inference engine.

As a result, INTERFACER presents two advantages. First, an expert-system user can modify any datum on an UI screen, completely independent of expert-system status. Second, programmers need not design a UI screen and not write routines for user data modification. This means that user data modification can be completely ignored by programmers. Using INTERFACER, we developed a practical expert-system: The General Employee Affairs Expert-system. It was clarified that the program amount decreased about 50% in comparison with conventional procedural implementation.

In Section 2, the problems of interactive expert-systems are clarified. Section 3 describes INTERFACER. In Section 4, the expert-system structure for INTERFACER is discussed. In Section 5, we introduce a practical expert-system implemented by INTERFACER. In Section 6, the effect of INTERFACER is described. In Section 7, the difference between conventional techniques and INTERFACER are discussed. Section 8 concludes this paper.

2. Problems of UI building for interactive expert-systems

UI building demands a major effort when implementing an application system, even in the case of expert-systems. In this section, we focus on the difficulties of UI building for interactive expert-systems, which generate many user queries for decision-making. We clarify two problems of UI building in this type of expert-system.

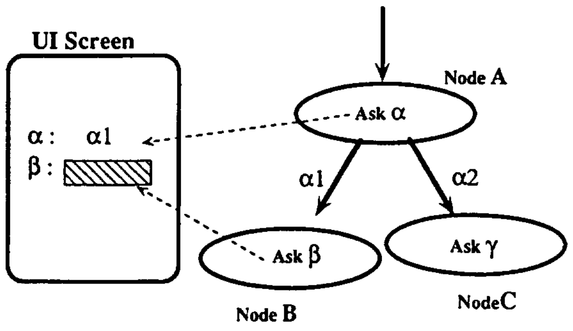

Fig. 1. Conceptual structure of decision process.

2.1. Difficulty of UI design

The decision-making process of expert-systems is usually represented by decision-trees or tabular format rules [6]. Fig. 1 shows a simplified example of this type of expert-system. In this case, the problem solving process is implemented with a decision tree. The decision-making process starts from the top node of the decision tree. First, the decision tree generates a query to get an α value. To get this value, the item name α is displayed on the CRT screen and the user must input the value. The resultant α value is sent to the decision tree, and the decision tree generates the next query, depending on the α value. That is, if α is $\alpha 1$, the next query item should be β. On the other hand, if α is $\alpha 2$, the next query item should be γ.

When node A, in Fig. 1, is invoked at the starting point, we cannot know how many nodes will be activated and cannot determine the sequence of node activation in this problem solving process. This means that the data query sequence will be determined dynamically at runtime, depending on the values input up to that time. As a result, the conventional UI builder, whose data items and sequence on the screen are fixed in the design phase, is not applicable for this type of application. Runtime dynamic generation of a UI screen is desired.

The same situation arises when decision-making is implemented with production rules, procedural

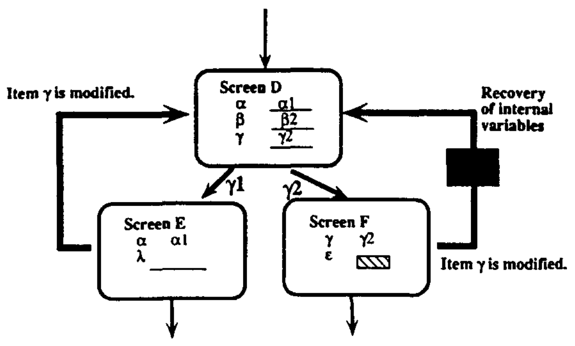

Fig. 2. Conventional table driven schema.

implementation or a decision list. Also, the Object-Oriented-Programming (OOP) paradigm is often used to implement expert-systems. This is because, in general, practical expert-system building tools provide OOP facilities; frame systems and message passing mechanisms [4–6]. The same situation can arise from OOP implementation if the object module size is small.

2.2. *Difficulty of implementing data modification routines*

In this subsection, to clarify the problem of implementing data modification routines, it is assumed that the user interface of an interactive expert-system is implemented by the conventional "Table Driven" schema. Many application systems employ this table driven control schema, especially in the business application domain.

Fig. 2 shows an example of this table driven schema. In Fig. 2, the first screen is screen D, and, prompts for data items α, β, and γ are displayed to the user. The next screen shown depends on γ value. This schema has the following problems:

1. Data modification routines, shown as the heavy lines in Fig. 2, are required. These data modification routines backtrack the normal path for problem solving, when the user wants to modify a datum. Thus, in a sense, these data modification routines duplicate the knowledge of the normal path. Clearly, the data modification routines are redundant in terms of problem solving.
2. If the modified datum had already been processed and triggered some modification of internal data, a specially designed data recovering routine, shown as the black box in Fig. 2, is required.
3. User modifiable items on each screen should be limited. The more we increase the number of modifiable items, the larger the amount of modification routines becomes.

If domain knowledge changes, programmers have to modify not only the decision-making knowledge but also these additional routines. This situation reduces the maintainability of the expert-system.

3. A user interface tool: INTERFACER

3.1. *Required function*

To solve the above problems, we developed the user interface tool: "INTERFACER". INTERFACER offers easy and simple UI implementation for interactive expert-systems. From the above discussions, the following functions are required.

1. **Dynamic Generation of UI Screens.** Programmers cannot predict which items are displayed on the UI screen and cannot determine the sequence of displayed items in the design phase of expert-system development. Thus, UI screens should be dynamically and automatically generated at run-time.
2. **Elimination of Data Modification Routines.** Usually, an expert-system user wants to correct datum-error on an UI screen as soon as he/she detects it. The implementation cost of the data modification routines are very high if any item on the screen can be selected as an erroneous item, as mentioned in Section 2.2. Thus, it is desirable for programmers to be able to implement an expert-system with no data modification routines to reduce implementation cost.

3.2. *Scheme of the proposed INTERFACER*

INTERFACER intermediates between the **user** of an expert-system and the **"main-routine"** of the expert-system (ES) as shown in Fig. 3. In this paper, the term "main-routine" means an expert-system

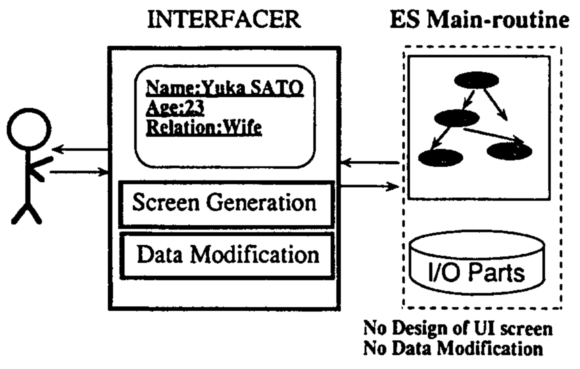

Fig. 3. Schema of INTERFACER.

routines other than a user interface. The main-routine is concretely constructed from an inference engine, knowledge base and input data values up to that time.

In Fig. 3, two types of messages should be sent from the main-routine: A *Display Message* to display a datum value or guidance text, and a *Requirement Message* to require a user datum. INTERFACER displays a window and locates each datum on this screen. The proposed INTERFACER usually generates a main window, a window for guidance, and a menu/a dialog-box for control. A help-window is also available. The main-routine never knows the display position of each datum on the screen. The position is decided by INTERFACER. Menus, dialog-boxes or guidance texts should be pre-defined as Input/Output (I/O) parts by programmers in the system development phase.

By using INTERFACER, an expert-system user can modify an arbitrary datum on an UI screen, independent of the expert-system's status. Also, the expert-system needs no data modification routines. It is clear that this scheme doesn't depend upon the structure of the inference engine. Thus, the expert-system can adopt any type of inference engine: decision trees, production rules, procedural implementation, decision list, or OOP paradigm.

3.3. Details of the proposed INTERFACER

This section describes the two major functions of INTERFACER: dynamic generation of a UI screen and elimination of data modification routines.

3.3.1. Dynamic generation of a UI screen

INTERFACER displays a window and locates items on the window from the top to the bottom. Thus, a UI screen is dynamically generated in accordance with the display or input-requirement demands from the expert-system main-routines. Fig. 4 image-of-action outlines INTERFACER. In Fig. 4 image-of-action, the expert-system main-routine is a decision tree of simplified Japanese Tax-rules.

INTERFACER has two types of messages:

1. Display message to display a value (including text).
2. Requirement message to query user datum and display the input value.

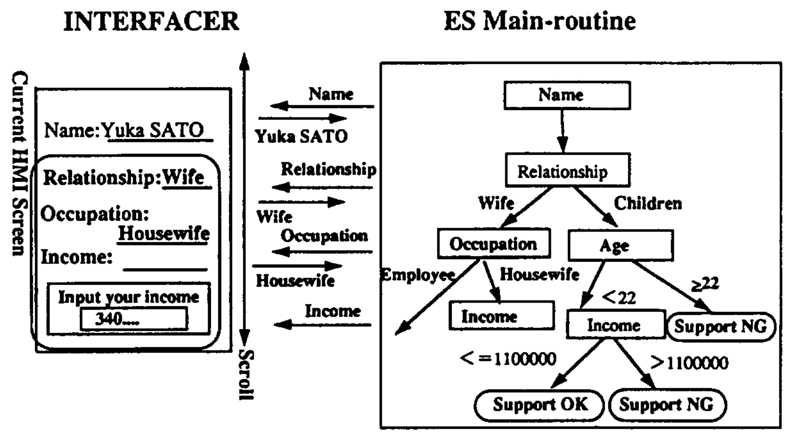

Fig. 4. Image of INTERFACER actions.

When the expert-system sends a message of display or requirement, the message has the following parameters:

1. Name of data-item (concretely, object-name + slot-name).
2. Name of I/O part (a menu or a dialog-box).
3. Datum value (only for display requirement).

Name of data-item is an identifier and address at which to store user input value. Each I/O part has many slots, such as title, list of selection items, list of return values, display position, datum type, name of check functions, default value, format of display. Details of these slots are omitted in this paper. These I/O parts are designed using the OOP concept. Programmers can easily customize existing classes.

3.3.2. Elimination of data modification routines

Data displayed on an UI screen should be easy to modify at the user's decision. In this case, data sequence on the screen may change. In particular, if the modified datum was input on the previous screen, the data sequence up to the current display position, may be changed. In this case, the expert-system has to backtrack and re-write the UI screens. These codes decrease expert-system productivity or maintainability as mentioned in Section 2.2.

To avoid the above problem, we think that the expert-system should be returned to the starting point and re-executed again, if the user modifies a datum on UI screen. This re-execution is triggered by a return value from INTERFACER. As shown in Fig. 5, when INTERFACER receives "Income" input requirement from the expert-system, we assume that the user wants to modify the old "Relationship" datum on the screen. In this case, INTERFACER sends a special "Modification symbol", as a return value of the "Income" requirement message (See (2) Return of Modification Symbol in Fig. 5.).

When the "modification symbol" is detected, the expert-system returns to the starting point. This is a relatively easy task, because all information is backtracked. For example, we can use the "Sub-cancel" function of Expert-system-building-tools KBMS [6] (Note: KBMS is a Japanese registered trademark of Nippon Telegraph and Telephone Corporation.) Most of practical Expert-system-building-tools have this type of function to reproduce the old status of objects/frames.

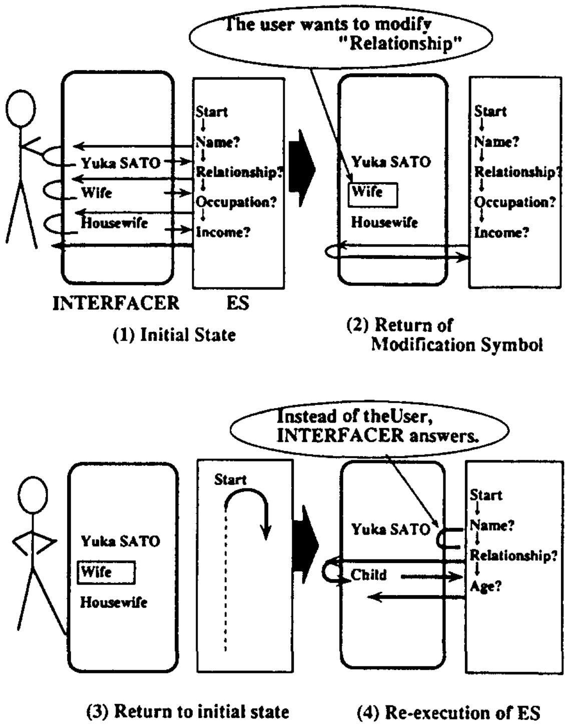
The user wants to modify "Relationship"
Yuka SATO
Wife
Housewife
Start
Name?
Relationship?
Occupation?
Income?
INTERFACER
ES
(1) Initial State
(2) Return of Modification Symbol
Instead of theUser, INTERFACER answers.
Child
Age?
(3) Return to initial state
(4) Re-execution of ES

Fig. 5. Detailed action of the proposed INTERFACER.

To eliminate data modification routines, INTERFACER provides the following functions.

Retention of current CRT screen. During restarting, INTERFACER must keep the current UI screen, unless the data display sequence doesn't change ((3) in Fig. 5). If the data display sequence changes, the changing point is displayed as the current screen. After re-execution, the user can input the new value of the modification item.

Memorization of user input data. Expert-system internal status backtracks to the initial state. Thus, the expert-system generates the same queries again, but the user would never accept this imposition. To avoid this phenomenon, INTERFACER memorizes the user input values of each data-item name. If the expert-system asks for an item having same name, INTERFACER answers this query. Note that the user is not aware of this INTERFACER's internal response ((4) in Fig. 5).

4. Expert-system program structure

By adaptation of INTERFACER, only a little additional routines are required. We consider two types of expert-system as follow.

Object-Oriented Expert-system Programs. Additional routines for INTERFACER need be added only to the message passing mechanism of the object-oriented language processor. If the user selects "Modification" of a datum on the CRT screen, INTERFACER sends "Modification Symbol" to the source object. The message passing mechanism detects this symbol and returns the expert-system to the starting point (Fig. 6).

However, it is inefficient to restart a large expert-system often. Thus, INTERFACER allows expert-system programmers to set "Break Points". Each break point is distinguished by a sequential number. When the user selects data modification, INTERFACER returns the appropriate break point number. The Message passing mechanism backtracks the expert-system to the adequate break point.

Fortunately, many expert-system-building-tools have the break-point-restart function. By using this function, the message passing mechanism can easily return the object status to the break point (Note: Usually, business expert-systems read and write DBMS. In this paper, we assume that the expert-system never modifies DB values during its operation. DB modification should be done after INTERFACER operation.).

Procedural Expert-system Programs. If the expert-system is written using procedural paradigm, programmers have to write the back-tracking routines by themselves. Concretely, each module that transmits datum-display or datum-input requirements must have a modification-detecting-routine and a return-path to the break point.

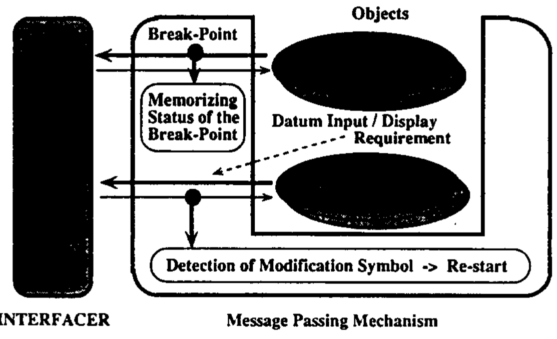
Objects
Break-Point
Memorizing Status of the Break-Point
Datum Input / Display Requirement
Detection of Modification Symbol -> Re-start
INTERFACER
Message Passing Mechanism

Fig. 6. Message passing mechanism for INTERFACER.

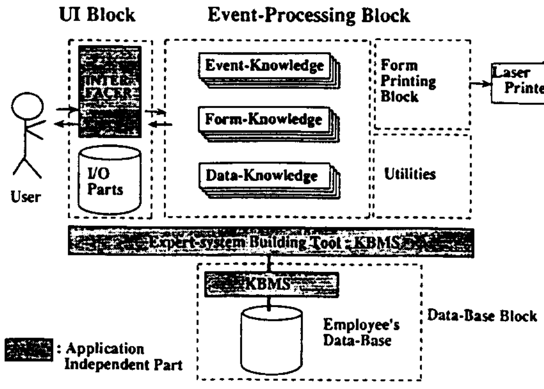

Fig. 7. Block diagram of the general affairs expert-system.

5. A practical expert-system using INTERFACER

We briefly introduce a practical expert-system developed using proposed INTERFACER, The General Employee Affairs Expert-System [9].

5.1. *Purpose of the general employee affairs expert-system*

This expert-system supports NTT employees in selecting and filling out application forms concerning a mutual aid association or public welfare. Japanese companies have many responsibilities for employees, such as payment of commutation allowance, providence of company house, giving special holidays, and applications for health-insurance. For these applications, a large amount of domain knowledge is required. This expert-system selects and fills out about 40 kinds of sophisticated application forms depending on the employee's situation and his/her requirements.

5.2. *System block diagram*

Fig. 7 shows a block diagram of The General Employee Affairs Expert-system. The user must know his or her "Event", that is, what happened on himself or herself. We prepared about 20 kinds of "Events", such as "Birth of baby", "Moving of address", "Death in his/her family". First, the user must select "Event" on a starting screen. Next, this user must answer questions from the system one by one. Finally, the user can select and complete the application forms.

The General Affairs Expert-system was implemented by INTERFACER and the Expert-system-building-tool KBMS [6]. This expert-system has about 600 kinds of user input data and domain knowledge. The expert-system routines are completely written as OOP paradigms. As a result, we can easily maintain this expert-system when application form format or domain knowledge changes.

In Fig. 7, menus, dialog-boxes, and guidance texts are pre-defined. The Event-processing block selects application forms, prompts for data-input, and fills out selected forms. Data-base block and Form-printing block are not mentioned in this paper. The Event-processing block has three layered objects: Event-Knowledge, Form-Knowledge and Data-Knowledge.

5.2.1. *Event-knowledge*

This knowledge memorizes which forms are related to each "Event". To select Form-Knowledge, Event-Knowledge sends a datum input requirement message to Data-Knowledge. If Data-Knowledge doesn't have the datum, Data-knowledge sends a datum input requirement message to INTERFACER. Finally, Event-Knowledge selects some Form-Knowledge from the input data and sends activation messages to the selected Form-Knowledge.

5.2.2. *Form-knowledge*

This Form-Knowledge has many slots that correspond to application form items, as shown in Fig. 8. Also, Form-Knowledge has "methods" that express decision knowledge. If Form-Knowledge is activated, it sends many messages to Data-Knowledge and completes the application form.

5.2.3. *Data-knowledge*

Data-Knowledge holds the employee's and his/her family's data as shown in Fig. 9. We prepared two types of data: "New" and "Old". "Old" values are copied from NTT's Employee Database

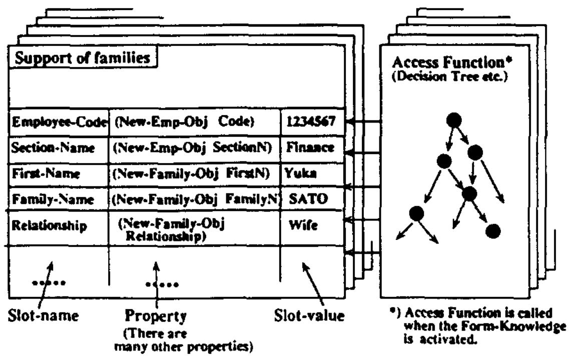

Fig. 8. Form-knowledge.

when this expert-system is activated. If Form-Knowledge or Event-Knowledge requires datum to Data-Knowledge, Data-Knowledge acts as follows.

- When "New" has a value: The value is already confirmed by the user. Thus, Data-Knowledge return this value to the appropriate source object.
- When "New" has no value and "Old" has a value: The old value is not confirmed by the user. Thus, Data-Knowledge transmits the old value to INTERFACER. INTERFACER displays the value on the CRT screen and the user confirms it or modifies it. The confirmed value is stored in "NEW" slot and Data-Knowledge sends it to the source object. If the user changes the old value, the new value is stored.
- When "New" and "Old" have no value: Data-Knowledge transmits a datum requirement to INTERFACER. INTERFACER asks the user and displays the input value on the CRT screen. The value is stored in "New" slot and transmitted to the source object.

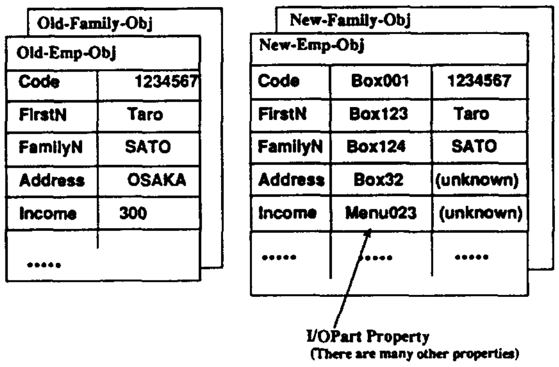

Fig. 9. Data-knowledge.

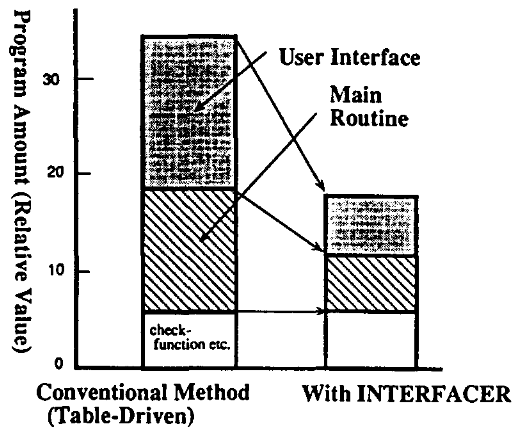

Fig. 10. Program amount comparison between two paradigms.

As shown before, data display or requirement messages are transmitted by Data-Knowledge. Event-Knowledge and Form-Knowledge needs some user input data. These values are usually received from Data-Knowledge. This scheme guarantees that, if some datum is input by a user, the same datum is never required again.

This General Affairs Expert-system has been put into service in Nippon Telegraph and Telephone Corporation. It performs the work of about one person in a big branch office and decreases human error.

6. Evaluation of INTERFACER

Initially, this expert-system was developed by a conventional static UI screen-builder and table driven control scheme. However, it was very hard to construct and maintain this expert-system. Thus, we rewrote this expert-system using INTERFACER. Fig. 10 compares the conventional table driven version to the INTERFACER version. The INTERFACER version has no data modification routine and no UI routine except for the definition of I/O parts. Fig. 10 doesn't include utilities, such as a tel-communication module, data-base access module, security module [10], and system maintenance utility. These utility modules have relatively large program amount, nearly equal to the amount shown in Fig. 10.

The combination of INTERFACER and OOP greatly simplifies the maintenance of this system. In office applications, domain knowledge is often changed. Usually, domain knowledge changes mainly result in application form changes. Since the KBMS provides an interactive modification environment of objects, no program re-writing or program compilation is required for form customization. Only one or two hours are required for item addition or deletion.

7. Related techniques

The control scheme of INTERFACER is analogous to the break-point-restart scheme of DBMS. For example, if a DBMS failure occurs, the DBMS status is recovered by transaction re-execution from that break point. Also, the INTERFACER scheme is similar to the re-execution performed by screen editors, for example, the recovering function of EMACS editor.

In both cases, re-execution generates exactly the same version as the old version. On the other hand, re-execution of INTERFACER not always generates the same screen. If a modification changes the sequence, the new screen will not duplicate the old screen. In this case its change point should be displayed to the user. After this change point, the old input values are shown as "Default Values".

8. Conclusion

From the user interface point of view, expert-systems are different from conventional applications in some features. First, the user query sequence strongly depends upon data input up to that time. Second, query sequence change requires highly complicated data modification routines.

To solve these problems, this paper has demonstrated the general purpose user interface tool "INTERFACER", which is suitable for interactive expert-systems. INTERFACER dynamically generates UI screens using pre-defined I/O parts, such as menus, dialog-boxes or guidance texts, in accordance with the demands of the expert-system. A special feature of INTERFACER is expert-system re-execution from break points when the user modifies a datum displayed on an UI screen. The current UI screen is kept during re-execution. Also, INTER-

FACER automatically answers the same input requirements, generated by re-execution.

By using INTERFACER, expert-system users can modify any datum on an UI screen, independent of the expert-system's status. Also, programmers do not need to design UI screens nor implement data modification routines. INTERFACER has been applied to a practical middle-scale business system: The General Employee Affairs Expert-system. Its program amount decreased about 50% in comparison with conventional table driven implementation.

Acknowledgements

The authors thank Teiji Emori for his kind and suggestive discussions. Also, the authors thank Katsuyuki Nakano and Fumihiro Kimura for their discussion and support.

References

[1] M. Green, Report on dialogue specification tools, in: G. Pfaff, Ed., User Interface Management Systems (Springer Verlag, Berlin, 1985).

[2] M. Green, A Survey of three dialogue models, ACM Transactions on Graphics 5, No. 3 (1986).

[3] B. Guchanan and E. Shortliffe, Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project (Addison-Wesley Publishing Company, 1984).

[4] J. Kempf et al., Experience with CommonLoops, Proceedings of OOPSLA'87 (1987).

[5] J. Kempf et al., Teaching Object-Oriented Programming with the KEE System, Proceedings of OOPSLA'87 (1987).

[6] KBMS Reference Manual (NTT Software Corporation, Yokohama, Japan, 1992).

[7] A. Kidd, The consultative role of an expert system, in: P.Johnson and S. Cook, Eds., People and Computers: Designing the Interface (Cambridge University Press, Cambridge, 1985).

[8] A. Kidd and M. Cooper, Man-machine interface issues in the construction and use of an expert system, International Journal of Man-machine Studies 22 (1985).

[9] K. Nakano, H. Kojima, and S. Kaneda, A General Employee Affairs Expert-system having Easy Knowledge Update Facility, IEICE of Japan, Reports of Technical-Group for Knowledge-processing and Artificial-Intelligence (in Japanese) (1992).

[10] S. Miyaguchi, S. Kurihara, S. Ohta and H. Morita, Expansion of FEAL Cipher, NTT Review 2, No. 6 (1990).