【Notes & Discussions】

Interface Reflecting a Hybrid of the Decision Support System and the Expert System
—Instantiation of an Object from the Class for Inference—

Hiroshi NOTO

Chapter 1

INTRODUCTION

The purpose of this article is to present an interface combining the decision support system (DSS) and the expert system (ES) to construct a small sized management support system (MSS).[1,2] In Seminar I (for Juniors) and Seminar II (for Seniors) I am in charge of, we have pursued the above theme by making an interface which is realized in an object oriented language (Visual C++)[3~7] under the Windows OS (Windows95 and/or WindowsNT). The text book[8] we have selected contains sample programs which deal with the DSS (written in C language) and the ES (written in PROLOG language) as seperate and independent modules. The DSS here means a generator or a shell for constructing a specific DSS. Therefore some model should be incorporated into the generator. Following the example of the text book we have taken up the 'finance model' for our specific DSS.

Now the first thing we must do is to understand the structure of their programs, especially their inputs and outputs. Then we examine what kind of interface or program should be required to connect the DSS and the ES and to execute the ES for theorem proving inferences[1] based on the rules and the DSS outputs. We have tried to add an interface connecting the two systems without changing the basic structure of each system to make an hybrid MSS as a user-friendly Windows applcation.

In Chapter 2 we describe combining the DSS and the ES. The format of the output from the DSS is shown in 2.1. The PROLOG program and its behavior are displayed in 2.2 which are pertaining to the merge of the DSS output into the ES and to the goal seeking (the proof search procedure). In Chapter 3 we represent an interface reflecting a hybrid of the two systems. Chapter 4 is devoted to creating a class for inference. In Chapter 5 the performance of the present MSS through the interface is exhibited. The summary and the conclusions are stated in Chapter 6.

Chapter 2

COMBINING THE DSS AND THE EXPERT SYSTEM

Here are two programs which exemplify the Decision Support System (DSS) generator or the DSS framework and the Expert System (ES) in the book written by Iijima[8]. We have tried to construct a small sized Management Support System (MSS) by combining the two programs as subsystems.

2.1 Decision Support System (DSS)

The DSS generator written in C language describes a general framework to make a specific DSS by incorporating a specific model into it. The resulting DSS outputs the data which are this time to be read into the ES for inference. The output from the DSS represents the facts which have the form of 'predicate and argument construction' of the clause in PROLOG:

⟨clause⟩ ::= ⟨*k*-place predicate⟩ (⟨argument*1*⟩, ⟨argument*2*⟩,⋯, ⟨argument*k*⟩)

which constitute the well-formed formula[1]. They are the elements of the first order logic. For example,

```
investment (1,280000.000000).
```

where the symbolic name 'investment'as a predicate is two-place relation which takes as its first argument the number of some term and as its second argument the amount of the investment. The semantics of a representation of the fact specify the interpretation of the clause as:

'The investment for the first term amounts to $280000.00'.

2.2 Expert System (ES)

The expert system (ES) written in PROLOG makes reasoning based on the rules and the facts, the latter being the output from the DSS. The rules here are considered to be the expertise transferred from the expert. The rule in the PROLOG language is represented as the Horn clause (of the first-order logic)[1,9] which is allowed to have one literal (either an atom or a negated atom) at most in the conclusion, i.e. on the left hand side of the clause (called the head of the clause). Thus a Horn clause rule has the general form:

$p :- q_1, q_2, \cdots, q_n.$

Here ':-' can be read as 'if' and commas can be read as 'and'. The interpretation reads like:

'For all values of variables occurring in the clause, the head p is true if the tail literals $q_1, q_2, \cdots, q_n$, i.e. the right hand side of the clause (called the body of the clause) are true.'

Running a PROLOG program consists mainly of a particular kind of theorem proving

?- p.

Here the goal is expressed as a literal 'p' and p is a particular theory to be proved. The goal search procedure is invoked as follows:

(1) A goal literal successfully matches (or unified) with p.

(2) The tail of the clause, $q_1, q_2, \cdots, q_n$ is instantiated with the substitution of values for variables derived from this match of (1).

(3)The instantiated literals in the tail can be thought of as subgoals that further invoke subgoals search procedures (i.e. searching for other rules). And all the subgoals need to be resolved.

This procedural invocation shows a backward (from the goal) reasoning strategy. The pattern matching here in PROLOG is what is called unification in logic. In the syntax of theorem proving in the predicate calculus, it is rather advantageous to negate the goal p first and next all the remaining subgoals are resolved away by pattern matching to derive the empty clause, signifying contradiction. Then the goal p is succeeded or satisfiable. The method of this proving theorem is called resolution refutation which the PROLOG exploits in its theorem proving.[1]

In Fig. 1, part of an example of the ES program written in PROLOG is shown. Here the following sentence in the Fig.1 asks the PROLOG to search the goal 'analyze' i.e. to resolve away all the clauses in the program by unification.

```
?- analyze. /* GOAL */
```

In the program shown in Fig.1, 'asserta (clause)' is a built-in predicate which appends a clause to the run-time database in its beginning.

```
liab_equ(1,1754671.000000).
liab_equ(2,2018274.000000).
liab_equ(3,2256718.000000).
/* cfinl_end noto */

/* EXPERT SYSTEM  'main'*/

/* Goal */
?- analyze.
/* Goal end */

analyze:-
preprocessing,
check,
diagnosis(X),
write("Therefore,"),
advice(X), nl.

/* EXPERT SYSTEM  'main' END*/

/* Expert System */
preprocessing:-
T is 3,
sousihon_keizyourieki_ritsu(T,X1),
asserta(sousihon_keizyourieki_ritsu(T,X1)),
uriagedaka_keizyourieki_ritsu(T,X2),
asserta(uriagedaka_keizyourieki_ritsu(T,X2)),
sousihon_kaiten_ritsu(T,X3),
asserta(sousihon_kaiten_ritsu(T,X3)),
T1 is T - 1,
sousihon_keizyourieki_ritsu(T1,Y1),
asserta(sousihon_keizyourieki_ritsu(T1,Y1)),
uriagedaka_keizyourieki_ritsu(T1,Y2),
asserta(uriagedaka_keizyourieki_ritsu(T1,Y2)),
sousihon_kaiten_ritsu(T1,Y3),
asserta(sousihon_kaiten_ritsu(T1,Y3)).
```

Fig. 1 Part of an example of the ES program written in PROLOG language.

Chapter 3

INTERFACE REFLECTING THE HYBRID OF THE TWO SYSTEMS

Now we have to combine the two subsystems (or modules) DSS and ES to make the management support system (MSS). We present here an interface connecting the two subsystems without changing the basic structure of the two modules.

The interface should satisfy the following requirements. The interface enables us

1) to execute the DSS and then

2) to consult the facts the DSS outputs, which means 'to read them into the ES' and finally

3) to perform inference invoking the rules stored in the ES.

The example of the interface is given in Fig. 2, where there are eight buttons and two edit boxes. The function of each button appears on its surface.

The upper edit box(Edit box1) monitors the facts the DSS system gets out and the lower edit box(Edit box2) displays the merged PROLOG program which the ES is to execute.

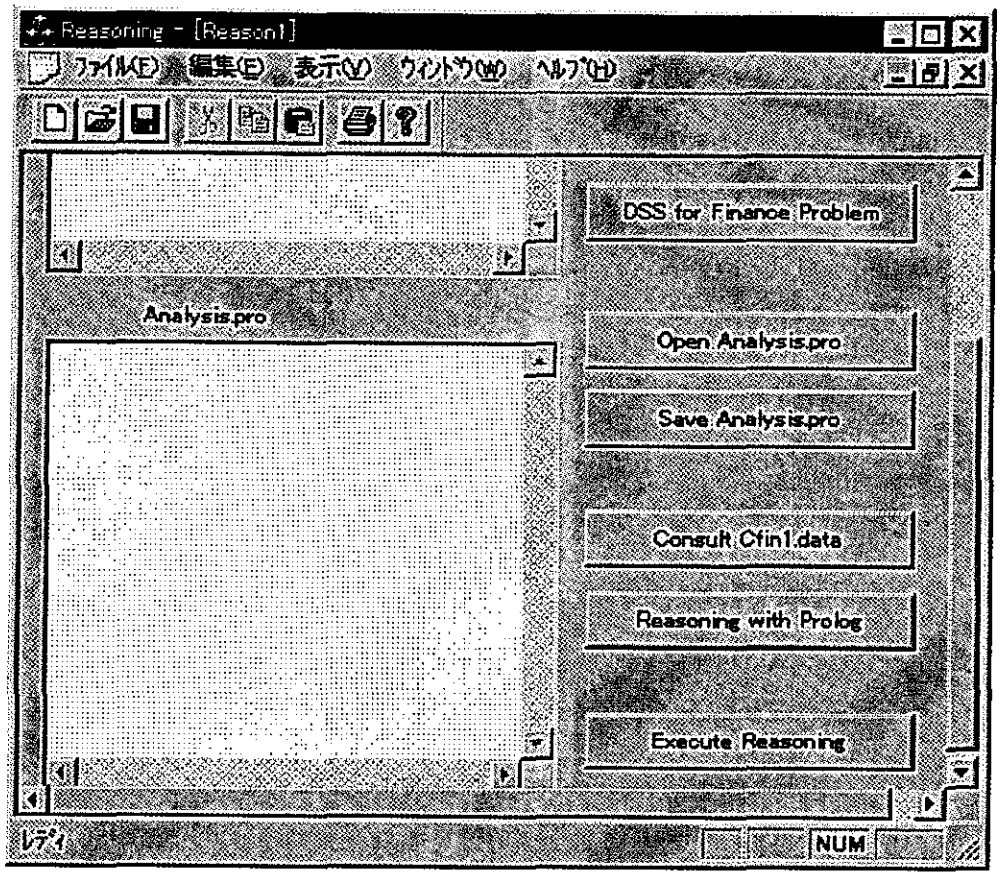

Fig. 2 Interface dialog box combining the DSS and the ES.

Chapter 4

CREATING THE CLASS FOR INFERENCE

It is very convenient to create a new class whose functions are to deal with inference. The new class (named 'CInference') has two member variables 'Cfin1' and 'Analysis_pro' which hold the text strings of the two files, one for the facts the DSS outputs and the other for the ES program written in PROLOG, respectively. We define three member functions 'Consult'', 'Execute_Inference' and 'Project_Inference', one for reading the facts from the DSS and adding them to the ES module and another for performing inference with the use of the facts and the rules and the other for executing the former two functions consecutively. Fig. 3 gives the definition of the class 'CInference', from which a new object is instantiated when the interface program runs, i.e. the ES gets executed.

```
//The definition of a new class CInference

class CInference : public CView
{
public:
CInference();
DECLARE_DYNCREATE(CInference)

private:
CString Analysis_pro, Cfin1;
CString begin_mark, end_mark;
CString m_filename2;

public:
void Project_Inference(CString,const CString&);
void Execute_Inference();
void Consult(CString, const CString);
public:
virtual ~CInference();
}
```

Fig. 3 Declaration of the class 'CInference'.

CInference() and ~CInference() constructs and destructs the class 'CInference', respectively. A member variable 'm_filename2' specifies the file name of the PROLOG program. Member variables 'begin_mark' and 'end_mark' designate the place in the PROLOG program where to put in the facts from the DSS. All the member variables are declared 'private', since they are referenced only within the 'CInference' class. Whereas the member functions are declared 'public', since they are all referenced from the 'CView' class which defines the interface dialog box and the 'view' of the window in Fig. 3.

Chapter 5

EXECUTING THE MSS THROUGH THE INTERFACE

Fig. 4 exhibits the result of performing the MSS which consecutively runs the DSS and the ES. After the execution of DSS is over, the facts are displayed in Edit box1 and the 'Consult' member function directs the facts into the PROLOG program in the ES. Edit box2 shows the PROLOG program after the facts are merged into the program.

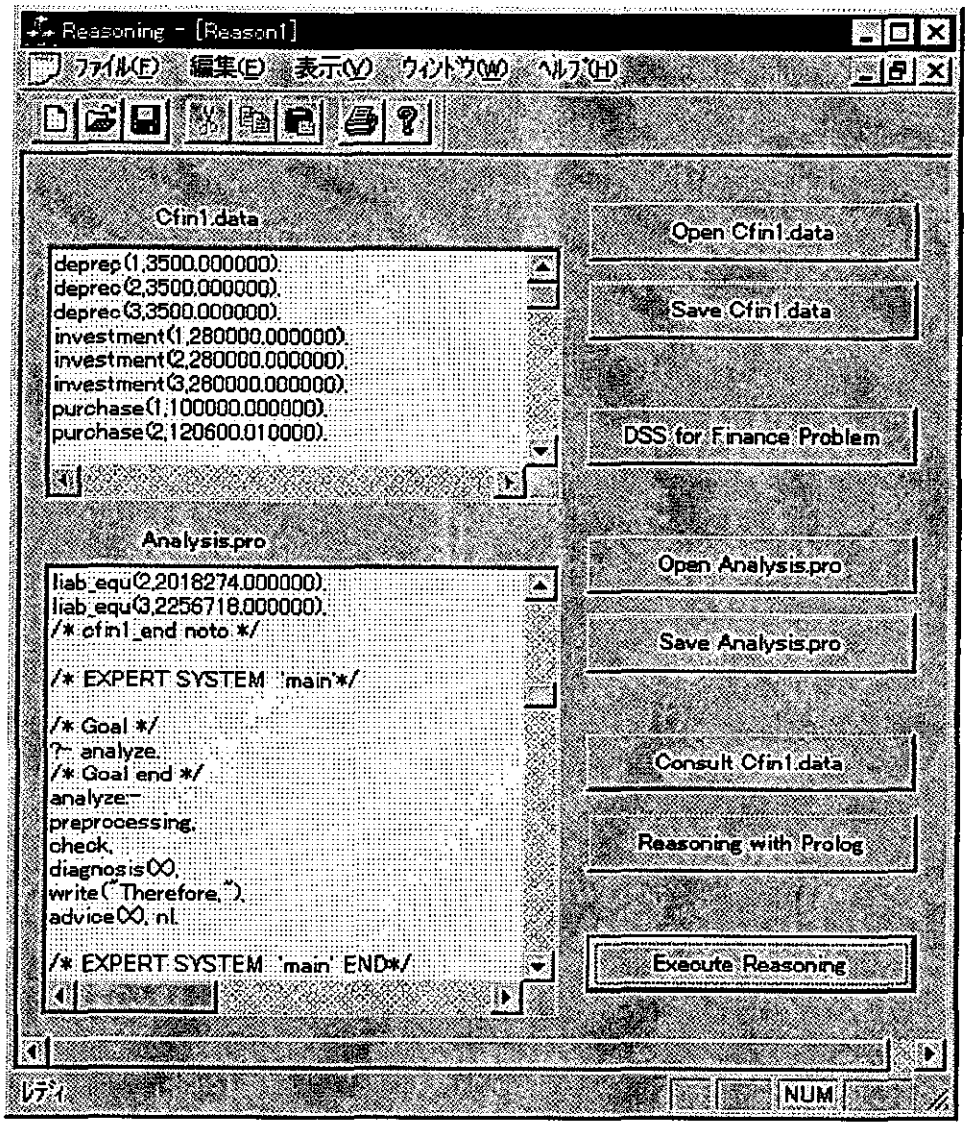

Fig. 4 The result of executing the MSS. Edit box1 displays the facts from the DSS and Edit box2 the PROLOG program after the facts are 'consulted'.

The member function 'Execute' pursues inferences based on the rules and the facts, and the messages (i.e. the results of the reasoning) from the ES are sent back to the PROLOG window which is shown in Fig. 5.

```
Compiling the file:
C:¥Analysis.pro
0 errors, 0 warnings.

sousihon-keizyourieki-ritu ga kakou site imasu.
uriagedaka-keizyourieki-ritu ga zyousyou site imasu.
sousihon-kaiten-ritu ga kakou site imasu.
Therefore,Tokuni, ureyuki, sihon no unyou-kouritu wo kentou si,
kaizensuru hituyou-sei ga arimasu.
Yes.
```

Fig. 5 An example of the messages of the inferences from the ES.

The last message 'Yes' means that all the goal and subgoals are resolved away. Those procedures are handled by the buttons in the interface dialog box as shown in Fig.2.

Chapter 6

SUMMARY AND CONCLUSIONS

We have presented the interface combining the two subsystems, the DSS and the ES without changing the basic structure of the two modules to build the MSS. The interface has enabled us to perform the DSS and to direct the facts from the DSS into the ES for reasoning. The interface merges the facts into the rules in the ES written in PROLOG. Then the ES carries out inference and draws the resolved messages that evaluate the facts which are the output of the DSS simulations based on a specific model, the 'finance' model in the present case.

When the interface program runs, an object is instantiated from a newly defined class, 'CInference' which in turn performs its member functions handling the MSS operations mentioned above. Whereas the new class encapsulates the data and the procedures required by the DSS and the ES, the realized interface has made the proof search procedures by the DSS and the ES clearer and their usabilities higher.

ACKNOWLEDGMENTS

The present interface programs are developed by using Microsoft Visual C++ 5.0 under Windows95 and WindowsNT. The PROLOG program is written in and processed by Strawberry PROLOG ver1.0 developed by D. D. Dobrev, Sofia, Bulgaria. Strawberry PROLOG performs under Windows95 and WindowsNT. The version 1.0 is freeware at the moment.

BIBLIOGRAPHY

1 Peter Jackson. Intorduction to Expert Systems (third edition), Addison-Wesley, 1999.

2 Efraim Turban and Jay E. Aronson. Decision Support Systems and Intelligent Systems (fifth edition), Prentice Hall, 1998.

3 David J. Kruglinski, George Shepherd, and Scot Wingo. Programming Visual C++ (fifth edition), Microsoft Press, 1998.

4 Hayasi, Haruhiko. A New Introduction to Visual C++ Ver. 5.0 (Beginners edition) (in Japanese), SoftBank Books, 1998.

5 Yamasita, Hirosi, Kuroba, Hiroaki, and Kuroiwa, Kentarou. C++ Programming Style (in Japanese), Ohmsha, 1994.

6 Alan R. Feuer. MFC Programming, Addison-Wesley, 1997.

7 John E. Swanke. Visual C++ MFC Progrmming by Examples, RD Books, 1999.

8 Iijima, Jun'iti. Decision Support System and Expert System (in Japanese), Nikka Giren, 1993.

9 Ivan Bratko. PROLOG Programming for Artificial Intelligence, Addison-Wesley, 1986.

*) The availability of the built-in procedure 'consult' depends on the implementation of PROLOG. We are unable to use this function on the Strawberry PROLOG.

[Abstract]

Interface Reflecting a Hybrid of the Decision Support System and the Expert System
—Instantiation of an Object from the Class for Inference—

Hiroshi NOTO

An interface combining the decision support system (DSS) and the expert system (ES) is presented to make a small sized management support system (MSS) without changing the basic structure of the DSS and ES programs. When the interface program runs, an object is instantiated from the defined class for inference of which the member functions carry out the MSS. The realized interface makes the proof search procedure by the DSS and the ES clearer and the usability of the MSS higher.