1988

NASA/ASEE SUMMER FACULTY FELLOWSHIP PROGRAM

MARSHALL SPACE FLIGHT CENTER
THE UNIVERSITY OF ALABAMA

SIRE - a Simple Interactive Rule Editor for NICBES .

Prepared by:	Alex Bykat
Academic Rank:	Professor
Institution:	University of Tennessee at Chattanooga
Department:	Center of Excellence for Computer Applications
NASA/MSFC:	
Laboratory:	Information and Electronic Systems
Division:	Electrical
Branch:	Electric Power Branch
MSFC Colleague:	David J. Weeks
Date:	August 8, 1988
Contract No.:	NGT 01-002-099 The University of Alabama

SIRE - a Simple Interactive Rule Editor for NICBES .

Alex Bykat

Center of Excellence for Computer Applications

University of Tennessee,

Chattanooga, TN 37403

ABSTRACT

To support evolution of domain expertise, and its representation in an expert system's knowledge base, a user-friendly rule base editor is mandatory. NICBES, a prototype of an expert system for the Hubble Space Telescope power storage management system, does not provide such an editor. In the following, we present a description of SIRE - a Simple Interactive Rule Base Editor for NICBES.

SIRE provides a consistent internal representation of the NICBES knowledge base. It supports knowledge presentation and provides a user-friendly and code language independent medium for rule addition and modification. SIRE is integrated with NICBES via an interface module. This module provides translation of the internal representation to Prolog-type rules (Horn clauses), latter rule assertion, and a simple mechanism for rule selection for it's Prolog 'inference engine'.

Acknowledgements.

I would like to express my appreciation of the hospitality extended during my last two summers spent at Marshall Space Flight Center. I have enjoyed and benefitted from participation in the Summer Fellowship Program. Much of this is due to the efforts of the program coordinators, Mrs. E. Cothran (MSFC) and Dr. M. Freeman (The University of Alabama).

Working in the Electric Power Branch was an enjoyable experience. This was due in particular, to Mr. David J. Weeks' hospitality, warm reception and friendly guidance through the maze of MSFC offices. Thank you Dave! My thanks go also to Dr. G.R. Wallace for both, putting me in touch with Dave and for supporting our cooperation, and to Mr. J.L. Miller, Mr. W.G. Shields and Mr. J.R. Lanier for supporting my stay in their division.

Introduction.

A prototype of NICBES -- NIckel Cadmium Battery Expert Systems -- is currently operational and under testing on the HST test bed. The prototype is written as two separate subsystems: the data handler, and the diagnosis system.

The data handler subsystem is written in Microsoft C language. Its main function is to receive telemetry data from the test bed, and to 'massage' this data into a form suitable for input to the diagnosis system.

The diagnosis subsystem is written in Arity Prolog language. Its main function is to take the data prepared by the data handler, and evaluate this data to discover any exception situations that might be indicated. In particular, the diagnosis tries to evaluate the state of the batteries (as indicated by the data) for possible malfunctions (charge leakages, overheating, etc), and for maintenance operations (recharging, etc).

The diagnosis subsystem displays its findings in terms of graphs indicating various 'trends' of batteries performance, as well as alarm messages (when they are necessary). The graphical operations needed for the trend displays were written in Microsoft C.

The NICBES prototype performs well, but a number of problems have been exhibited during its test runs. In addition, the experience with the prototype has provided insights which allowed to identify some needed additions and improvements of the system. Some of these problems and needed improvements are listed below.

NICBES problems:

1. Battery plots show incorrect data points. This apparently occurs
   a. consistently in the last data point, and
   b. intermittently in other positions.

The first case is probably due to NICBES incorrectly handling the last data point (interrupted orbit?).

The second case may be due to bad data points. If this is the case the origin of the bad data has to be traced. A number of possibilities exist:

A. NICBES data handler garbles the data.
B. NICBES data handler gets confused because of garbled header in the data package.

The data handler receives transmission once every minute. Each transmission lasts precisely six seconds. An orbit telemetry data consists of 96 such transmission bursts, with each burst consisting of a data header followed by 370 telemetry values. A data header consists of a special start-data character -- the character A -- and is followed by nine data items, which identify the history of each data burst.

To receive the transmission, the data handler looks for this special start-data character. If the character A is not recognized, the data is rejected as noise. When the start data character is recognized, the data handler starts collecting the data and stores it as orbit data files, for a subsequent use by NICBES.

Herein lies a possible source of trouble -- here is a situation where two wrongs can make a right! A misread start-data-character will cause valid data (and in particular the header) to be rejected. A subsequent data item, misread as a start-data-character, may cause acceptance of the remaining part of a transmission as a complete telemetry. Of course, this latter data will be out of phase, and may very well cause bad data points to appear on plots.

To verify this scenario, a software filter should be written. This filter will be applied to (suspect) archived telemetry data files to detect short transmissions. Once the relationship between these faulty files and the bad data points is verified, the filter can be integrated with the data handler, to reject such faulty transmissions.

C. Telemetry data contains bad values. This case should raise an alarm prior to transmission of data to NICBES. Since no alarm is raised there are two 'sub-possibilities':
   C1. alarm does not work (eg. it does not recognize invalid data), or
   C2. alarm is not recognized by NICBES

2. NICBES evaluates batteries performance from data collected over 12 orbits only.

Battery performance trends should be evaluated over a large number of orbits. However, increase in the number of orbits is currently infeasible due to the hardware and software environment of NICBES prototype.

NICBES improvements:

1. Presentation of trends over a long period of time. This requires ability to handle more than the 12 orbits in the current version. The number of orbits to be aimed for is 400 to 500.

   Before we can look at a technical solution to this problem, we have to change NICBES environment. The 12 orbits limitation is due to the NICBES hardware environment which consists of an IBM AT with 640 KB memory and 20 MB hard disk running under DOS.

2. Knowledge Base Rule Editor. It is desired to have a user friendly facility for display and modification of NICBES rules. Currently, to change or even to display a rule, Prolog programming knowledge is required.

   This should be doable over the Summer, provided the NICBES has a well defined knowledge representation. If this latter holds, at least a rudimentary Rule Editor can be build incorporating functions such as

a) display a rule,
b) delete a rule
c) add a rule

3. Graphic representation of battery voltage uses a 'flexible spread' range and computed scale within that range. This makes visual recognition and comparison rather difficult for a human eye.

   The scale have the same range, those the origin point of the scale can be adjusted to represent the (minimal) data value.

4. Multitasking is necessary to continue data collection when NICBES consultation is in progress.

   Desqview was suggested as a quick fix to add the multitasking capability. However, it is my opinion (based on my current and superficial knowledge of DV) that this software will not provide multitasking within one package. I have to investigate that further, but I believe that DV provides 'multiple window multitasking' ie. time sharing of applications, not multitasking within application.

5. Printout of messages without interrupting NICBES data collection.

   This is related to the above requirement.

The above list of needed improvements is arranged in order of priority. However, not all of these improvements can be tackled with currently available hardware configuration. In particular, presentation of trends over a long period of time (improvement 1) requires a large main storage capacity, certainly larger than the 640 Kbyte available on the currently available NICBES computer (IBM PC/AT). Similarly, the multitasking capability (improvement 4) is not available for the IBM PC/AT. Thus, these improvements will be

relegated to implementation of NICBES on an I80386 class of computer.

The requirement for a knowledge-base rule editor can be developed for the current NICBES prototype. It is this new subsystem that is described in the subsequent pages of this report.

Objectives.

The main objective of the following work is to provide the NICBES prototype with a subsystem which supports acquisition and modification of nickel-cadmium battery management rules. This subsystem will be used by the (human) battery management experts to extend and to formalize their experience (with the said batteries) into a knowledge base. This expert knowledge base is captured in the form of production rules, and is used by the (software) expert system -- NICBES -- in diagnosis of battery performance.

To achieve this objective, it was necessary to write a software system -- SIRE -- which provides:

1. Internal representation of knowledge in a consistent manner.
2. Capability to display the captured domain knowledge.
3. Capability to add new domain knowledge.
4. Capability to modify captured knowledge in a programming code independent manner.
5. Interface to NICBES to integrate the subsystem and to avail the knowledge base for subsequent diagnosis of battery performance.

SIRE - Simple Interactive Rule Editor.

NICBES was intended to prove the capability of expert system technology in (eventually) autonomous management of the power supply system for the Hubble Space Telescope. Consequently, and as this technology dictates, NICBES has two main components:

1) domain knowledge, and
2) mechanism to manipulate this knowledge.

Building the domain knowledge demands consideration of two aspects:

1) knowledge acquisition and
2) knowledge representation.

Of course, the latter is influenced by the fashion in which the knowledge manipulation mechanism will use it.

Evolution and changes in domain expertise, mandate from an expert system provision for a programming-language independent rule editing. Current version of NICBES, does not support such capability. The rules of NICBES are simply programmed in Prolog, demanding familiarity with Prolog to effect rule changes. Since this essentially requires a new and highly technical skill from the domain experts, it is not a satisfactory situation. Indeed, to change a rule, in addition to knowing Prolog, the domain expert would have to be familiar with NICBES internal code structure. To alleviate this situation we shall implement a 'friendly' rule editor -- SIRE. Figure 1 shows the SIRE environment within NICBES.

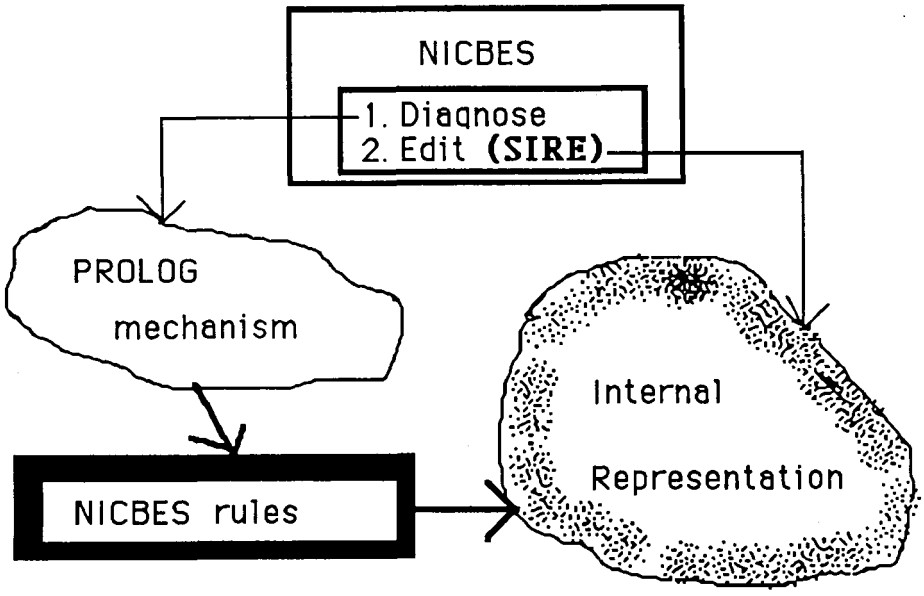

Fig.1. SIRE environment

Rule representation.

As stated previously, current version of NICBES does not represent rules in a consistent and well defined way. The rules are simply programmed in Prolog, with structure depending on each individual rule type. The following is an example of a NICBES programmed rule:

```
advice(Bat,1):-
        write($** ADVICE ON RECONDITIONING BATTERY: **$), NL,
        get_one(Trend1,Trend2,Trend3,Trend4,Trend5,Bat),
        recond(Trend1,Trend2,Trend3,Trend4,Trend5).

get_one(Trend1,Trend2,Trend3,Trend4,Trend5,Bat):-
        ......

recond(T1,T2,T3,T4,T5):-
        write($EOD voltage trend is$), write(T1), nl,
        write($In-charge voltage trend is$), write(T2), nl,
        write($Recharge ratio trend is$), write(T3), nl,
        write($EOD divergence trend is$), write(T4), nl,
```

```
        write($In-charge divergence trend is$), write(T5), nl,
        tell_1(T1,T2,T3,T4,T5).

tell_1(strongly_down,_,_,_,strongly_up):-
        write($Reconditioning advised to correct failing
        capacity.$),nl.
```

In the above, the *advice* predicate invokes the diagnostic evaluation of battery *Bat* performance data. The second argument in advice indicates a choice of the type of diagnostics to be performed. (Currently, there are three choices possible: reconditioning, charging regime, and workload.)

The *get_one* predicate uses orbit telemetry data files to calculate the battery *Bat* trends. These trend values are then "fuzzed" into various intervals, and the latter are given symbolic names such as: *strongly_down, down, up, strongly_up.* Note that *get_one* calculates variables which are then used as initial parameters for the diagnostic rules.

The *recond* predicate takes these symbolic trend names, and displays their values as an overall battery assessment. Thus, *recond* merely reports the trend values, and then passes control to *tell_1* .

Finally, the *tell_1* predicate uses the appropriate symbolic trend values to display NICBES recommendations in English. (Although not shown above, there are altogether 11 different recommendations that can be produced by *tell_1.* of course, different recommendations are produced in response to different symbolic trend values.)

To support SIRE functions, the rules (such as above) have to be structured and represented consistently. The following describes the knowledge representation that shall be adopted for SIRE (and hence NICBES) rule representation.

The internal representation of NICBES rules will consist of the five parts:

1) rule category
2) rule number
3) parameter initialization

4) premise specification
5) conclusion specification.

These five parts will be presented as five arguments of the PROLOG predicate rule/5, eg:

```
rule(Category, Number,
        params(Par_List), premis(Pre_List), conclude(Con_List)).
```

Note that the rule category allows us to structure the rule base into rule groups corresponding to each category. This allows increase in search efficiency during the inference process. Within each category, a rule is identified via its number. Globally, each rule will be uniquely identified by the tupple (rule category, rule number).

1) **Rule category** names are created by the user. Current NICBES rule base will be divided into categories such as reconditioning, charge and load. Other rule categories can be established by the SIRE user.

2) **Rule number** will be used to identify a rule within a rule category. Rule numbers must be unique within a category, though they may be repeated across the categories.

3) **Parameter initialization** list Par_List will consist of elements, each in the form of a valid Prolog term. Typically, these terms will represent assignment of a value to a parameter which is used subsequently in the rule body. The assignments are of the form:

Parameter:= Value

where

Parameter -
specifies the variable to be used in the rule. This is the actual slot that will contain the the parameter's value.

Value -
specifies the initialization procedure for the parameter. The value can be of three types:

ask-
specifies that a value for this parameter should be asked for interactively from the SIRE user.

get_trend (Type,Bat),
get_average (Type,Bat)-
specifies that the function be evaluated and its return value be assigned to the parameter.
<other> -
specifies that *<other>* be assigned as entered to the parameter's value.

4) **Premise specification** list Pre_List will consist of elements, each of which is a Prolog term. The terms specified in the list, combined with the AND operator (Prolog's comma) will constitute the antecedent of the rule.

5) **Conclusion specification** list Con_List will consist of elements of the same form as those in the premise specification list. The terms in Con_List will be executed subject to satisfaction of the rule's premises.

Rule interpretation.

The following is an example of a NICBES rule, stated in its (new) internal representation. This internal representation is followed by it's English interpretation.

```
rule( recondition,
      1,
      params([
            $uses(Bat)$,
            $EOD_VOLTAGE_TREND:=get_trend(eod_voltage,Bat)$,
            $IN_CHARGE_DIVERGENCE_TREND:=
            get_trend(in_charge_divergence,Bat)$]),
      premis( [
            $EOD_VOLTAGE_TREND = down$,
            $IN_CHARGE_DIVERGENCE = strongly_up$]),
      conclude([
            $write('reconditioning is advised')])).
```

English translation:

Parameter initialization:
Get the global value *Bat*
Evaluate *EOD_VOLTAGE_TREND.*
Evaluate *IN_CHARGE_DIVERGENCE_TREND.*

The rule:
If
EOD_VOLTAGE_TREND is down and
IN_CHARGE_DIVERGENCE_TREND is strongly_up
Then
write: *reconditioning is advised.*

Note that the elements of the three rule parts are specified as Prolog strings. This is necessary due to Arity Prolog's failure to preserve symbolic names of variables. In Arity Prolog implementation, variable's symbolic name is replaced with a coded name. Although this saves some internal storage, information conveyed by the symbolic name (so vital to program's documentation) is lost at run time. (This is particularly bothersome in debugging stage.)

SIRE capabilities.

SIRE capabilities are accessed via two level menu system. The selections from the two levels are combined providing an (almost) Cartesian product of choices, as indicated in the table below:

Second Level Menu	First Level Menu: show	delete	add	change	statistics	print	quit
<category>	X	X	X	X	X	X	X
rule	X	X	X	X	X	X	X
class	X	X			X	X	X
all	X	X			X	X	X
quit	X	X	X	X	X	X	

SIRE functions

Note that <category> will be actually presented as a list of category names, eg: recondition, charge, load, etc.

SIRE design.

SIRE is written in Arity Prolog Version 4. The editor consists of

1. menu module,
2. rule base module,
3. editor functions module,
4. rule interface module,
5. on-line help module, and
6. SIRE control module.

Menu module.

The menu module consists of SIRE menu definitions. These are statements using the Arity Prolog Screen Toolbox. The menu definitions are kept in file MENUS.ARI which is consulted by the SIRE control module.

As stated, SIRE capabilities are accessed via two level menu system. The first level affords selection of requests for NICBES rule-manipulation functions such as:

show, delete, add, change, save, statistics, help, print, quit.

Selection of any option (other than the quit option) will result in display of a second level menu. The second menu level allows selection of NICBES rules to be manipulated. The choices provided in the second menu level are:

<category>, rule, class, all, quit.

Here, **<category>** represents a list of names of rule categories that actually exist currently in the rule base. The <category> list of names in the menu is updated dynamically, so that it can be looked at as a dynamic test for rule class changes.

Rule base module.

The rule base module contains the internal representation of NICBES diagnostic rules. The syntax of the internal representation is quite simple:

```
RULE        ::= rule(CATEGORY, NUMBER, PARAMETER, PREMISE,
                 CONCLUSION)
CATEGORY    ::= <user defined name>
NUMBER      ::= <integer>
PARAMETER   ::= params( TERM )
PREMISE     ::= premis( TERM )
CONCLUSION  ::= conclude( TERM )
```

where TERM is a string written in syntax as defined by the Arity Prolog term definition. Generally speaking, this means a syntax defined by

```
TERM        ::= <operator> ( ARG_LIST)
TERM        ::= <atom> | <number> | <string> | <variable>
ARG_LIST    ::= TERM
ARG_LIST    ::= ARG_LIST, TERM
```

Note that, as is usual in Prolog, the operators can be written in prefix or infix notation as appropriate.

Rule interface module.

The interface module is provided to integrate the new internal rule representation with the NICBES diagnostic system. The main function of this module is to translate the internal rule representation into a Prolog clause form. This 'executable' form is then supplied to NICBES inference mechanism for evaluation.

The interface module performs as a three stage module. In the first stage the internal form of SIRE rule is retrieved and then translated into the Prolog expression form. It is at this stage that the syntax of the rule is checked -- a rule written in valid syntax is one that conforms to the syntax of a Prolog expression. If the rule is found to be syntactically invalid, a suitable message will be output, and the translation of the rule is abandoned. The user can then use SIRE to correct the rule.

In the second stage a name for the rule, based on the rule's CLASS and NUMBER is generated. The name is then prefixed to the rules body, and the valid clause is then asserted (added) as a rule. Thus a rule now appears as a valid Prolog clause. While asserting such a rule, any existing rule with the same name is retracted (erased). Translation and assertion of the rules into NICBES rule base is done only once -- the first time a consultation for the required class is requested. This minimizes the internal representation overheads due to the translation process.

In the third stage the Prolog inference mechanism is invoked with the asserted rules supplied as the rule base. This process consists, essentially, of evaluation of rules in the order of their assertion

sequence. If a rule succeeds, the result is exhibited to the user; if the rule fails, another rule is extracted from the rule base and tested.

While a rule is evaluated, this module retrieves the global values indicated in the parameter initialization part and makes them available to the rule. In addition, the 'basic NICBES operations' are evaluated (as indicated) to provide the initial values to the rule variables. (These basic operations, eg. *get_trend, ask, :=,* etc., are defined in this module.)

Since many rules may require evaluation of the same operation for the same battery using the same data, there is a considerable opportunity for inefficiency. To avoid this, each operation which obtains new information, generates a 'lemma' which is then used by the repeated calls without a need to repeat the 'proof'. Thus, for example, a new information obtained by *get_trend(charge,1)* will be asserted as a new fact; next time the *get_trend(charge,1)* is called, this fact is then simply retrieved rather then recomputed.

Editor functions module.

This function provides the capabilities for modification of the internal knowledge representation. The editor functions currently provided include:

show, delete, add, change, save, statistics, help, print, quit.

On-line help module.

SIRE has an extensive on-line help available to it's user. The on-line help describe all of the major functions of SIRE, and is accessible via the *help* selection in the first level menu. The capability of the on-line help is further indicated in the following figure.

	First Level Menu					
	show	delete	add	change	statistics	help
help	X	X	X	X	X	X

SIRE online help

Control module.

SIRE control module provides access to the functions of the editor. This module calls the menu modules to display the menus, and subsequently combines the user responses to determine which of the editor functions are to be activated by the editor function module. On completion of user requests, the control module performs maintenance operations on behalf of the rule module.

Conclusions.

The objectives of this project have been successfully met. The SIRE subsystem has been successfully integrated with the NICBES prototype, and will provide it's users with a system for display of the knowledge base status, addition of new knowledge in the format of SIRE/NICBES rules, and modification of these rules as the dynamically evolving expertise may require.

In addition to the above, a rudimentary diagnosis explanation facility has been implemented. It provides not only display of conclusions, but also identification of rules which were instrumental to reaching the conclusion. Although this explanation facility should be further developed, it will already be of benefit by providing a means to pinpoint the rules and to focus on the knowledge which may need revision.

All of these functions are now 'user-friendly'. Thus, the user of NICBES no longer requires knowledge of programming in Prolog, nor does he require knowledge of the internal structure of NICBES code.

As a by-product of this work, it has been ascertained that the NICBES prototype code must be rewritten in order to function as a useful product. This conclusion is inevitable when the sluggishness of NICBES is compared with its current 'size parameters'. Thus, although NICBES has only some 40 rules in its knowledge base, and refers to only 12 orbits, it is already heavily i/o bound. The main reasons for the frantic disk activities exhibited by NICBES, are due to the DOS limitation of 640 KB for memory, the size of the Prolog system, and the storage cost of recursive calls in Arity Prolog.

The slow-down due to the excessive disk activities is further compounded by the battery performance trends and averages calculations. For example, although only 12 orbits are taken into account in calculation of a performance trend, such calculation takes about 20 seconds. With five trends to be calculated, the user has to wait for almost two minutes. Now for the crunch: if number of orbits is increased to 1000, the calculations (as programmed now) of five trends will take 3 hours!!

References.

Martin Marietta Corp. "Final report for NICBES", MCR-85-641, 1986

Martin Marietta Corp. "User manual for NICBES", MCR-86-673, 1986

Martin Marietta Corp. "Requirements specification for NICBES", MCR-86-674, 1986

Martin Marietta Corp. "Program maintenance manual for NICBES", MCR-86-675, 1986

The Arity/Prolog Programming Language, (Version 4), 1986

The Design Arity Screen Toolkit, 1986

Bykat, A. "User manual for SIRE", 1988

Bykat, A. "Expansion of NICBES capabilities", MSFC, NAG8-105, 1988