N88-17234

EXPERT SYSTEM VERIFICATION CONCERNS IN AN OPERATIONS ENVIRONMENT

Mary Ann Goodwin and Charles C. Robertson
Rockwell Shuttle Operations Company
600 Gemini
Houston, Texas 77058-2777

ABSTRACT

The Space Shuttle community is currently developing a number of knowledge-based tools, primarily expert systems, to support Space Shuttle operations. This effort is based on the wide-spread realization of the potential benefits of these tools for premission flight planning and real-time flight support. Evolution of these tools into the operations environment is just beginning.

It is proposed that anticipating and responding to the requirements of the operations environment will contribute to a rapid and smooth transition of expert systems from development to operations, and that the requirements for verification are critical to this transition.

This paper identifies the verification requirements of expert systems to be used for flight planning and support and compares them to those of existing procedural software used for flight planning and support. It then explores software engineering concepts and methodology that can be used to satisfy these requirements, to aid transition from development to operations and to support the operations environment during the lifetime of expert systems. Many of these are similar to those used for procedural software.

INTRODUCTION

The range and diversity of specialties and subspecialties required to support Space Shuttle operations develop an enormous amount of the type of skill recently designated "expertise". Expert systems to support flight operations appear to offer significant potential benefits to flight design and dynamics, such as:

- Reducing the manpower and resources required for flight design and dynamics.
- Reducing the dependency on highly skilled people to intervene periodically in fairly standardized tasks, thus freeing them for new development or nonstandard problems.
- Preventing single-point failures or delays due to the unavailability of skilled "experts"; and, similarly, reserving the "corporate knowledge base" should a skilled person become unavailable.
- Improving the quality of certain decisions which require more factors than a human can comfortably consider at once, but which are no problem for a computerized expert system.

A number of expert systems have been built and others are presently being built so that these benefits can be realized. Many more can be expected as the technology becomes an accepted part of the engineer's problem-solving capability and a larger skill base is available for their implementation.

Most of the existing systems are considered prototypes. However, once in the operations environment, they must satisfy the demands of that environment. Because of their potential for affecting flight design decisions that have broad and sometimes critical implications, engineering confidence in the veracity of their results across their lifetime will be of foremost importance to their successful acceptance and integration into flight operations. Therefore, anticipating and preparing to support verification during the lifetime of the expert system should ensure that their potential is realized as well as reduce the time and the human and computer resources required to maintain them.

Following is a discussion of software requirements in the Shuttle operations environment, what can be considered a verified expert system, historical approaches to aid and accomplish verification of conventional programs, and, last, approaches that can be taken during prototype development to aid verification and ease integration of expert systems into the operations environment.

SHUTTLE OPERATIONS SOFTWARE ENVIRONMENT

Flight design and flight dynamics software have an overlapping set of modeling and performance requirements. Premission conceptual flight design may require fairly simple analytic models, whereas operations flight design will require extremely high fidelity models that run much slower than real-time. Flight dynamics, software must support real-time performance and, at the same time, the best possible performance achievable under that constraint is desired. Expert systems for flight design and dynamics must meet these same modeling and performance requirements.

Presently, a great deal of effort is being made to streamline and standardize Shuttle flight design and dynamics. For example, procedures for developing flight products and ensuring their quality have been explicitly defined and documented. Techniques are being implemented to track product development, to ensure that the defined procedures are followed and that approved software are used for product generation. Software approved for product generation is being placed under configuration control, and changes must be formally requested, approved, and verified prior to being made available to the flight designers.

When expert systems are used in the generation of flight products and in support of flight dynamics, they will become subject to the same controls. Often the expert system decision-making capability will be embedded in the application software, so that it will simply account for a portion of a larger system that is under configuration control.

Greater use of database technology is planned to manage flight data and ensure its integrity and commonality among products. Electronic storage and retrieval will pass data among the software programs generating the products. Expert systems will also be required to access and store information in these databases.

This environment will require that expert system verification be one component in the verification process of a complex multilanguage software system that includes conventional languages such as Fortran or C, the embedded expert system shell language, and the embedded database query language.

WHAT IS A VERIFIED EXPERT SYSTEM?

If we are to produce "correct" expert systems, we must produce systems that reflect "correct" knowledge, which for practical purposes may be considered the "best" knowledge, judgement, or decision-making capability the expert possesses or can derive. The verification problem as discussed herein is to provide expert systems that reflect this knowledge and will never reflect any contrary results.

Responsibility for the "correctness" of knowledge belongs by definition to the expert. In flight design this is analogous to responsibility for the requirements for conventional flight software belonging to the flight designer. For the expert system, the knowledge must first be correctly acquired from the expert, a responsibility shared by the expert and the "knowledge engineer" or implementer. Then the implementer must reflect exactly what the expert means by creating a rule and fact base in the expert system shell language. This sounds very familiar to those who have generated conventional software systems. The implementation must match the specifications.

Verification of an expert system, then, must verify the adequacy and accuracy of the knowledge base implementation according to specific performance criteria.

LESSON FROM THE PAST (AND PRESENT)

Existing flight design and dynamics software reflect various design and development software engineering methodologies that evolved over the years the software was developed, i.e., the design and development techniques and philosophies vary widely.

Verification is generally considered to be one part of the software engineering process, but the ease with which it can be accomplished has been recognized as being dependent on the techniques and methods used for the preceding development phases. That is, how one does design and implementation affects how one does verification. Approaches that encourage early discovery of errors reduce the time, design impact, and computer and human resources required for corrections.

Verification first occurs prior to the initial delivery of software and it recurs each time the software is modified over its lifetime. For some flight software, the lifetime can be considered essentially unlimited. Modifications of flight design software, whether procedural or expert system, can be expected as

- New flight design requirements occur.
- The knowledge in some area so improves that it is desirable for the flight design software to reflect these improvements.
- Updates are made to the Shuttle hardware or software.
- Significant changes in the state of the art of software and hardware occur that offer desired performance improvements.

It is now recognized that the cost to maintain flight software during its lifetime far exceeds the development cost. Measures taken during development to reduce cost during maintenance are cost effective.

A great deal of research has been done and effort made to develop techniques to support verification of conventional software. These techniques can be placed in two categories: (1) design and implementation techniques that either reduce the likelihood of errors or make them easier to find and correct; and (2) software development support tools that detect and remove errors from the code.

The following techniques are in the first category:

- Project management techniques such as top-down development, design and code reviews, use of program libraries, etc.
- Project design techniques, such as top down design, code modularization, and, more recently,

information hiding, object-oriented design, entity-relationship modeling, data flow diagrams, etc.

- Languages modifications which simplify code and make it easier to understand and debug; e.g., structured code, strong typing.
- Documentation standards, both external and internal to the code.
- Coding standards which not only standardize how code is written but which may also outlaw code considered to be error-prone.

The following techniques are in the second category:

- Development of static code analyzers and dynamic code analyzers. With static code analyzers, the code is parsed, and the parsed information is stored in such a manner that a postprocessor can cross-reference information to detect errors. An example is locating variables used but not set or vice versa. With dynamic code analyzers, the code is "instrumented" with "probes". Special-purpose code is inserted at strategic locations to capture and output data of interest as a routine executes. This output is then postprocessed to provide information to aid verification. For example, it is possible to determine what part of the code is executed and what is not for all test cases in a test case library (See reference 1)
- Improvement of compilers to aid error detection.
- Development of test case libraries that satisfy such criteria considered beneficial to verification as exercising as much of the code as possible and doing "stress testing"to exercise numerically sensitive code.

Development of an automated software development and maintenance support environment for use in all phases of program development, from program design to code generation and program verification, is presently occurring and may greatly impact the update of existing flight software systems and the creation of future systems.

VERIFICATION TECHNIQUES FOR OPERATIONAL EXPERT SYSTEMS

Enter expert systems into the Space Shuttle operations environment. A large flight design simulation could conceivably have "pockets of reasoning" for decision-making at various points in its execution. A decision might be made to determine the type of flight to simulate, the characteristics of the sensors to be simulated, or the environmental models to be invoked. Another decision might be made to output recommendations about the simulation or information about its results that help the flight designer. These types of decisions presently occur at defined points in flight design programs. They simply occur with a man in the loop and sometimes in an off-line mode. Therefore, one might consider them already a part of the software design, and it appears reasonable that the design of an expert system decision-making capability for a flight simulation can be accomplished by extending the conventional system design techniques.

One technique that successfully attempted to do this is documented in references 2 and 3. The well-known hierarchical input process output (HIPO) technique was used to develop requirements, construct the design, and support implementation of an expert system to demonstrate automated rendezvous. Verification was then conducted systematically because of the method of design and implementation.

In the pragmatic Shuttle flight operations climate, where expert system design, development and verification is one part of the design, development, and verification of an existing or emerging software system using conventional and database query languages, it appears that it would be helpful to identify where commonality in verification techniques may be applied and where uniqueness is required in the verification of the overall system.

The interfaces between the symbolic reasoning (or expert system) "modules" and conventional modules or database tables or files can identify type conversions required to go from symbolic facts in the expert system module to digital data or other data types in conventional modules and vice versa. This information should be amenable to data flow diagrams, data dictionaries, or other data/information tracking techniques.

Language improvements can be found as revisions to expert system shells are released. Of great significance for flight design and support is the development of expert system shells that work on conventional hardware and allow the shell language to be embedded with conventional languages. While this does not appear to support verification directly, it allows simpler and more natural interfaces with the rest of the software system which will therefore be less error prone.

A set of preliminary experimental documentation standards and complementary coding standards have been defined (reference 4) for the Automatic Reasoning Rool (ART) developed by Inference, and a subset has been adapted to the Clips shell language developed at the Johnson Space Center (reference 5). The standards have been successfully adapted to a number of expert systems being developed to support flight design (references 6 and 7).

The standards were constructed to support the later development of a maintenance tool. Consequently, they were designed using keywords that could cue a parser to the contents of the various types of comments. Two categories of comments were defined: those to support user explanation of the rules and those to support the programmer in implementation and maintenance. Comments in the first category were intended to be extracted automatically to produce documentation for the users. The standards established were designed to support an automated tool that could generate cross-reference information for rules, patterns, and variables.

The standards are divided into three areas. First, a major file is defined which includes the history of the expert system and other pertinent information regarding supporting functions and files. All files should be loaded from this major file, which is the program driver. The second area is the declarative knowledge which consists of ART viewpoint information and definitions of relations, facts, and global variables. The third area deals with procedural knowledge, which consist of rules. The commenting template and explanations for this third area are given in figure 1. It has been found that the procedural template can be adapted to the design phase and used for "knowledge reviews". An additional use found for the procedural knowledge template is training support. As the design expands from functional to various levels of detail, the declarative and major file information can be developed as needed.

The authors propose that a relational database management system could be used to perform useful static analysis for error detection of rule-based expert systems and could be implemented independently or in conjunction with the proposed automation of the documentation and coding standards just mentioned. The relational theory allows semantic relations to be conveniently expressed. Some of the simpler relations that could be expressed as relational tables are defined in figure 2. Table 1 lists various relationships that could then be determined by querying the tables. Some of the most aggravating problems that can occur during debugging have to do with simple typing errors that could often be detected by locating occurrences of a unique, one-time-only pattern. Properly constructed data base queries could isolate unique variable names that are likely in error. Further error detection possibilities exist that space does not permit exploring (e.g., see reference 8).

Several other possibilities exist. It is apparent that with sufficient effort the tables in figure 1 could be utilized to automatically construct the expert system shell code, which would be error free. The specific nuances of the languages in expert system shells will undoubtedly introduce aggravating problems in the implementation of the above, but the goals seem achievable.

Additionally, it is possible to express similar type of relational information about the conventional code, such as that captured by the system in reference 1. The query language could then verify interfaces across the two languages by querying the applicable information in particular tables for each language.

CONCLUSIONS

As expert systems are integrated into the Shuttle flight design and support software packages, an integrated project management and software development and maintenance plan that encompasses conventional procedural languages, the expert system shell language, and database query languages is needed so that verification can be accomplished at a minimum cost and results in the highest confidence in the software system over the life cycle of the flight software.

This goal seem achievable. Design and development methods and coding and documentation standards based on those used for procedural code have been applied to expert system prototypes with good results. Additionally, verification tools for expert systems similar to those for procedural code but relying on database systems to simplify implementation appear conceptually to be beneficial and extendable to include conventional code. The latter method could possibly be extended to produce expert system shell code automatically.

The ultimate validity of the expert system reasoning, however, lies with the expert. No amount of programmer effort can improve the judgement or reasoning communicated to the programmer by the expert.

It is recommended that seriously developing and refining these methods as part of prototype development will contribute greatly to a smooth transition of expert system programs from the development to the Shuttle operations environment.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the community of individuals from the Johnson Space Center Mission Planning and Analysis Division, Computer Science Corporation, LinCom Corporation, McDonnell Douglas Corporation, and Rockwell Shuttle Operations Company, who contributed to the documentation standards of reference 4. Grace Hua and Ann S. Baker of CSC must be singled out for their special contributions.

REFERENCES

1. Software Development and Maintenance Aids Catalog, JSC-22342. Dennis Braley, Mission Planning and Analysis Division, JSC, October, 1986.

2. Software Engineering Techniques Used to Develop an Expert System for Automating Space Vehicle Rendezvous, Daniel C. Bochsler, LinCom Corporation, and Mary Ann Goodwin, Johnson Space Center, Second Annual Workshop on Robotics and Expert Systems, Johnson Space Center, June, 1986.

3. A Software Engineering Approach to Expert System Design and Verification, David C. Bochsler, LinCom Corporation, and Mary Ann Goodwin, Johnson Space Center, Conference on Artificial Intelligence for Space Applications, Huntsville, Alabama, November, 1986.

4. Documentation Standards for Expert Systems, JSC Memorandum FM7/86-187, Robert H. Brown, October 22, 1986.

5. CLIPS Reference Manual, JSC-22552,Christopher J. Culbert, Mission Planning and Analysis Division, Johnson Space Center, March, 1987.

6. Orbital Navigation Expert System (ONEX); McDonnell Douglas Memorandum TM1.2-TM-FM87044-022; A. J. Reich/T5B; December 29, 1986.

7. Guidelines and System Requirements for the Onboard Navigation (ONAV) Console Expert/Trainer System, JSC-22433; Artificial Intelligence Section, Mission Planning and Analysis Division, JSC, December, 1986.

8. Knowledge Base Verification; Tin A. Nguyen, Walton A. Perkins, Thomas J. Laffey, Deanne Pecora; AI Magazine V.8, N.2, Summer, 1987.

```
;-------------------------------------------------------------------------------

;;; GROUP
;;  <group-name>
;;  <narration on purpose, description of control, objective,
    assumptions, etc.
;;; HISTORY AND RESPONSIBILITY: GENERAL
    (contains information common to all rules in the group)
;;  Name of programmer(s):   <name>
;;  Name of expert(s):       <name>
;;  Created on:              <date>
;;; CONTROL FACTS
    (those unique facts inherited from a parent group)
;   <fact>
;;; PARENTS
    (Used for cross-referencing)
;;  <parent group-name>
;-------------------------------------------------------------------------------

(defrule        <rule-name>
;;  If
;;  <english sentence definition of the rule conditions>
;;  Then
;;  <english sentence definition of the rule actions>
;;  End
                .
                .
    <actual rule body>  ;  simple programmer code comment
                .
                .
)
;;; HISTORY AND RESPONSIBILITY: EXCEPTIONS AND UPDATES
;;  Name of programmer(S):   <name>
;;  Name of expert(S):       <name>   (rule-specific exception
                                        to general )
;;  Created on:              <date>
;;  Modified by:             <name>
;;  Modified on:             <date>   (rule-specific update)
;;; RATIONALE
;;  <narrative on heuristics, reasoning, rule-specific assumptions
    and limitations, etc.>
```

Figure 1 - Template for Procedural Knowledge Documentation

A) LEFT HAND SIDE TABLE

RULE NAME	CONDITION SYNTAX	CONDITION CLASSIFICATION
<name>	<condition-1> <condition-2>	< (function, e.g. control, test, pattern, external-function, etc.)>

B) RIGHT HAND SIDE TABLE

RULE NAME	ACTION SYNTAX	ACTION CLASSIFICATION
<name>	<action 1> <action-2>	< (function, e.g. external function, pattern-set, pattern-retract, etc.)>

C) RULE SALIENCE TABLE

RULE NAME	CONDITION CLASSIFICATION
<name>	<number>

Figure 2 - Proposed Expert System Static Analyzer Rule Data Base Format

TABLE I - POSSIBLE USES OF EXPERT SYSTEM STATIC ANALYZER DATA BASE

- Reconstruct rules, including, for example all rules subject to the same condition
- Find patterns set on LHS, but not used on RHS and vice versa
- Determine effect of new rules or rule changes on rule base
  - If new rule sets or retracts a pattern, other rules that use that pattern can be identified
  - Rule dependencies can be identified.
- Find all rules with same salience, or ordered by increasing or decreasing salience
- Find all rules with same control pattern
- Find all external function calls and rules which made call
- Construct English description of rules. (Requires tables not shown in Figure 2.)