

A Logic Generator and Validator

J. Szabo, D. Okrent

University of California, MANE Dept., Los Angeles, California 90024, U.S.A.

D. Cain

Electric Power Research Institute, P.O. Box 10412, Palo Alto, California 94303, U.S.A.

ABSTRACT

The on-line monitoring of a power plant (or any process plant) consists of three primary functions: data acquisition, data analysis, and data presentation. This work relates to the second of the above three functions by presenting advanced methods for generating reliable data analysis computer codes.

The conventional method of analysis code production has the systems' analyst or designer generate rules by which the plant status is being evaluated, while the transcription of those rules to computer code is done separately by a programmer. Subsequently, the analysis code produced by the programmer must be validated against the specifications prepared by the systems' analyst. This work presents a Logic Generator and Logic Validator to streamline these processes.

1. Introduction

The development and implementation of computerized operator support systems for nuclear power plants has accelerated over the past decade with the rapid advances in computer technology. The accident at Three Mile Island Unit 2 gave further impetus to this trend after investigators cited the need for compact displays and computerized information networks to deal more effectively with developing accident situations [1, 2]. In direct response, post-TMI requirements for Safety Parameter Display Systems (SPDS) and Emergency Response Facility (ERF) information systems are presently being addressed by all U.S. utilities and in many foreign countries [3, 4, 5]. The SPDS features a computer-driven display of key plant parameters with trending and simplified alarm conventions. It is meant to assist control room personnel in determining the safety of the plant quickly and accurately. The underlying philosophy is that there are a number of safety functions which have to be fulfilled in a nuclear plant and there is no way a serious accident may happen if these functions are fulfilled. Therefore in theory a limited set of guidelines is enough to cover any emergency which can happen in the plant. That is because no matter what was the initiating event, once the impaired safety function has been identified the resulting actions on the part of the operators are almost independent of the emergency cause. These near-term plant improvements are proceeding in conjunction with longer-term efforts which will provide sophisticated computer algorithms for aiding plant operators in crisis situations.

All of these on-line computerized systems will serve as decision tools for operations personnel. It is clear that these systems must be sufficiently reliable to ensure that the information is correctly given. A computer installation can nominally be divided into its hardware and software constituent parts. During the past decade, very significant advances have been achieved in upgrading computer hardware reliability. Furthermore, large production volume and on-board diagnostics generally guarantee that major problems are identified and corrected. The same, however, is not necessarily the case with software. Whereas some software errors or bugs in computerized operator support systems can be weeded out using structured programming and verification and validation (V&V) procedures, there is no practical way of absolutely assuring software integrity. The software logic which the computer uses to process raw plant data and provide high level information to operators is the "intelligent" part of these systems. It is also the most crucial from the standpoint of overall system integrity. Errors in computer logic software may result from miscommunication between the systems' analyst and programmer that goes on undetected. Rather than causing catastrophic errors in the on-line software which are easily detected, logic errors can allow the system to appear to function properly, but provide erroneous Interpretation of plant data to the operators.

This work addresses the problem of improving logic software reliability by introducing an advanced method for generating reliable logic routines that can be used in computerized operator support systems. Our approach concentrates on the signal processing logic of an SPDS and does not address the other major aspects: data acquisition and display design. We present a Logic Generator which is a computer program that inquires for relevant information through a systematic dialogue with a system analyst, and then automatically translates it into an efficient code which reflects that information. Described also is a Logic Validator which can be used as a tool for showing the user a structural overview of the logic code produced by the Logic Generator by decomposing the software into its input attributes. The Logic Generator and Validator routines apply in general to any process control function and in particular to SPDS installations and a variety of other on-line computer support functions.

This work was performed at the Electric Power Research Institute (EPRI) as part of the Safety Control R&D program.

2. Application Generators

Some major problems encountered in the conventional process of software production served as a catalyst for the introduction of new technologies. The multiple human interfaces in the translation of specifications from one language to another have the potential for causing misunderstandings and mispecifications. The languages involved, including English description, program design language specifications, data design specifications, and the code itself, are inherently inconsistent tools, as they do not utilize common semantics and syntax. The functional specifications for the applications can not be proven to be correct until they are completely built. Errors or conflicts discovered in later phases may require major changes and disrupt the development process. Many of the errors are due to the lack of automation, as all phases of the conventional software production process are

performed manually.

If a large number of fairly similar programs are to be written, it is sometimes possible to segment the synthesis task into two parts, routine portions that are common to all programs in the class and task-dependent portions that must be different for each new program. An "Application Generator" [6] is a program which automatically executes the more routine portions of the program synthesis task and enables the user conveniently to input the task-dependent information so that the desired program can be created.

The merits of the different Application Generators are:

- a. Can be used by non-programmers to create programs.
- b. Improve the productivity of trained programmers.
- c. Program maintenance is improved because of consistent program structure from application to application.
- d. Methodology can be based upon a rigorous, formal mathematical structure.
- e. The specifications can be proven to be logically consistent and correct.

Differences may exist among the generators in regard to what they do and how they do it, but a common method of performance exists. All seem to rely on a preset framework which automatically leads the user through a series of menus, tables or English language questions. As the user responds, the generator translates the input into the programming code which reflects this input. The input data from the user may come as answers to specific questions, as a less specific filling in of blanks, or in a graphical mode where the user "prints" a screen to define data format.

3. Logic Generator

The Logic Generator is an Application Generator which is directly useful in automatically generating a real-time computer code to support SPDS displays [7]. In the SPDS computer logic operates on the key plant parameters determining the status of a given safety function, integrates the results, and provides alarm indications to operators whenever a critical safety function is jeopardized or violated. These alarm boxes may be linked to the plant data base through fairly elaborate logic functions. This logic could be "hand-coded" and subjected to rigorous verification and validation procedures [8]. Alternatively, the logic can be generated semi-automatically using a specialized logic generator tool, and validated with a complementary tool.

The key to the latter approach is the realization that:

- (1) There is a fairly small set of basic logic operations which cover the realm of functionality needed for control room applications.
- (2) There is a generic logic model or "structure" which can be invoked.
- (3) The logic model can be linked as modules to generate computer logic of arbitrary complexity.

The basic unit of the logic production, the "logic primitive", is a three-stage module. In the first stage a small set of basic logic operations is used to combine a relatively large number of inputs into a few higher level components. Thus a new set of parameters is passed from the first stage into the second stage. This new set may contain aggregates only, original input parameters only, or a mixture of both kinds. A member of this new set is defined as an output component.

In the second stage the different component states are defined and, in case a component is a continuous variable, the definition includes ranges of values for each state. This follows the assumption that for control purposes the exact value of a continuous input variable is of secondary importance, as long as it is within a predefined range. Thus at the end of the second stage all possible values of the different inputs have been mapped into a limited number of discrete states.

The value assigned to the output of each logic module depends upon the state of each output component. In the third stage a mapping function is defined through which an attribute is assigned to the module output based upon the state configuration of the output components. Thus, if output component A (which can be reactor power for example) is in state 1 and output component B (control rod position for example) is in state 2, a particular color is assigned to the module output. A different color attribute will follow component A in state 2 and component B in state 1, etc.

The inner workings of a typical module is illustrated in Fig. 1 where the inputs are a combination of analog and discrete signals. The output of the logic is the assignment of a color to a display element. However, the logic output could have been assigned numerical or character value.

The three stages of the logic modules define aggregates, components, and configuration attributes, as shown in the figure. In this example the aggregation process reduces the input data streams down to two "aggregate" signals by selecting the highest value from one, and the number of "true" signals in the other. In the second stage the components are produced by discretizing the aggregate values. The aggregate analog signal in this example is discretized into three states, whereas the digital signal is tested to determine if its value is two. These component values are subsequently passed to the third stage, where a mapping between input state combinations and output states is achieved. The output state in this instance is set to "green" and could drive a CRT display element to show a green color.

The internal structure for logic modules is completely general and allows diverse inputs and value setpoints to be mapped into outputs representing specified logic functions. Furthermore, the output of any one module can also serve as input to other modules. The result can be cascaded logic structure of fairly arbitrary complexity.

The Logic Generator is a FORTRAN code which interacts with the user in a question-answer format to automatically generate other FORTRAN routines which represent these logic modules. The code thus produced is not optimal from a computer performance vantage point. What it does achieve, by imposing a standard framework for logic development and systematic input, is an optimization of the computer software for reliability and maintainability.

4. Logic Validator

The Logic Validator is complementary to the Logic Generator. Whereas the Logic Generator is a Fortran routine that translates user input into applications code, the Logic Validator "reads" the applications code and decomposes it into parameters which can be compared to the input specifications. This readily enables the user to determine that the logic performs the intended operations without requiring the user to inspect the actual source routine.

One of the options of the Logic Validator is to produce a map showing how output

components are obtained from input variables. In addition the Logic Validator has a variety of options for analyzing the logic software which has been generated. This ensemble of analysis functions "reads" the as-built logic software and develops a structural decomposition that can be compared with the original software functional design.

It is important to note that the Logic Validator is not by itself a guarantor that logic software is valid in all respects. It does not detect specification errors by the system analyst, but it does provide a facility for easily comparing the software end-product with the input design specifications to assure that there is consistency. Also, since the Logic Validator operates on the applications software and is independent of the Logic Generator, it can detect errors which might be introduced owing to flaws in the Logic Generator routine itself.

5. Conclusions

The Logic Generator and Validator is written entirely in FORTRAN 77, and is a transportable tool. It allows a utility to produce its own SPDS code or other applications software tailored to specific needs.

By linking logic modules together, more elaborate logic functions can be easily built and validated using these semi-automatic tools. The modular Fortran code that is produced has been shown to be reasonably compact and efficient. The interactive session solves the problem of a designer who is not a programmer interfacing with a programmer who is not a designer in the conventional code production process.

The Logic Generator produces a standardized code which can be easily inspected by the user. The comments added during the production stage allow the Logic Validator to "read" the standard code and decompose it into its parameters which are then displayed to the user, either in words or in graphical manner through a map.

A feature allowing partial validation of input signals has been added to the code. If through other independent means it is found that the reliability of one of the input signals is in doubt, this information will propagate to the higher level signals and the output will be presented to users with a caution flag.

A standard worksheet is provided to organize input data and information. Since it contains all the information needed to recreate the logic module, this worksheet doubles as software documentation.

There are some areas for this work to be extended. For instance, the Logic Generator could be integrated more directly with a colorgraphics display builder, so that graphic displays can be developed, logic configured, and links to the data base can be established in a step-wise sequence. The Logic Generator might also be extended to support data input processing in addition to output. Signal validation and quality checks on input data could be developed using a variation of the Logic Generator tool. The modular Fortran routines thus produced can interface the input data stream prior to archiving into the computer data base.

References

- / 1 / KEMENY, J.G., "Final Report of the President's Commission on the Accident at Three Mile Island", Washington D.C. (1979).
- / 2 / UNITED STATES NUCLEAR REGULATORY COMMISSION, "Requirements for Emergency Response Capability (Generic letter #82-33)", NUREG-0737, supplement 1, (1982).
- / 3 / LEVY, S., "Fundamental Safety Parameter Set for Boiling Water Reactors", NSAC-21, (1980).
- / 4 / EPRI, "Disturbance Analysis and Surveillance System Scoping and Feasibility Study", EPRI NP-2240, (1982).
- / 5 / PETRICK, W., "Workshop on Emergency Response Facilities", NSAC-57, (1983).
- / 6 / HOROWITZ, E., KEMPER, A., NARISIMBAN, B., "Application Generators: Ideas for Programming Language Extensions", Proc. of 1984 Annual Conference, Association for Computing Machinery, (Oct. 1984).
- / 7 / MULLEE, G.R., ABUROMIA, M.M., "Simulator Evaluation of the Boiling Water Reactor Owner's Group (BWROG) Graphics Display System (GDS), Sandia National Laboratories, AL0-1019, (1983).
- / 8 / STROKER, E., "Verification and Validation of the SPDS", NSAC-39, (1981).

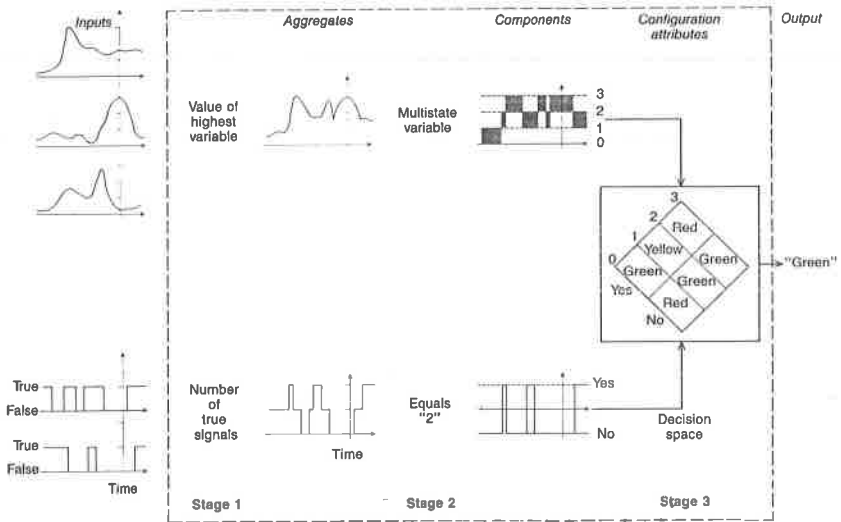


Fig. 1 Internal Structure of Logic Module