A DESIGN PHILOSOPHY FOR RELIABLE SYSTEMS, INCLUDING CONTROL

by

John R. Gabriel

Mathematics and Computer Science Division

April 1984

This work was supported by the Applied Mathematical Sciences subprogram of the Office of Energy Research, U.S. Department of Energy.
# Table of Contents

Abstract ............................................................................................................................. v

1. Introduction .................................................................................................................. 1

2. Intelligent Systems for Plant Management ................................................................. 1

3. The Design as a Contract .............................................................................................. 2

4. A More Detailed Design Concept ................................................................................ 3

5. Fault Tolerance ............................................................................................................ 4
   5.1 Computer Hardware ............................................................................................. 4
   5.2 Computer Software ............................................................................................. 4


7. A Conceptual Design .................................................................................................... 6
   7.1 The Requirements .................................................................................................. 6
   7.2 Input to the Plant Model ......................................................................................... 6
   7.3 Fault Modes ............................................................................................................ 6
   7.4 Detailed Models of a Fault ..................................................................................... 7
      7.4.1 Mechanical Fault ........................................................................................ 7
      7.4.2 Thermal-Hydraulic Example ..................................................................... 8
   7.5 Formal Theory of Plant Models and Fault Identification ...................................... 8

8. A Suggested System Architecture ............................................................................... 9
   8.1 The Question of Software Failure ......................................................................... 9
   8.2 A Possible Design ................................................................................................. 10

9. A Programming Language ............................................................................................ 11

Acknowledgments ............................................................................................................. 12

References ....................................................................................................................... 13
Abstract

In the past, use of computers and software to manage physical plant has usually involved systems similar to the clockwork automata of the 17th century. The next generation of plant control will include intelligent systems—computer systems having knowledge of the plant and being capable of intelligent behaviour, even though only some control functions will need such expertise. This report develops a framework for a universe of discourse usable by such non-human experts. It is based on the idea that a design has many features of a contract and may be described as a contract between humans and a machine, defining what each must do to attain a goal.

Several points are discussed: the use of techniques in analytical redundancy and their place as analogues in administrative control for conventional techniques in physical control; the use of redundant computer systems to protect against hardware faults; the necessity to prove properties of software used in redundant hardware, because software faults are common modes across redundant hardware; and some issues in choosing a programming language for provable control software. Because proof of correctness is costly, it should be used only where necessary.

This report concludes that the degree of reliability needed by the plant model used in analytic redundancy protection need not be nearly as reliable as the mechanism to detect discrepancy between plant and model.
A Design Philosophy for Reliable Systems, Including Control

John R. Gabriel

1. Introduction

This report deals with the use of intelligent computer systems to manage physical plant. It was originally hoped to be a "handbook" for those concerned with design of such systems. But the discipline is new, and not well developed, so the report actually asks more questions than it answers. The perspective presented here thus bridges the gap between questions that should be asked and the development of answers using existing technology.

Intelligent computer systems are conventionally called "expert systems." Besides differentiation according to methods in computer science, such systems may be classified by the way their knowledge of the world is kept or derived. On the one hand are question-and-answer systems whose knowledge is a collection of empirical fact stored as a set of possible questions and matching answers. At the other extreme are analytical systems whose knowledge is a theoretical model of the universe implemented as a computer program.

The distinction is not unequivocal. Rule-based systems have a little bit of each property, and from one point of view the set of questions and answers is just another simulation model. Nevertheless, there is a real distinction. An analytical model may be able to answer unforeseen questions, but a question-and-answer model can function for a universe where there is no theoretical knowledge.

Most intelligent systems built so far have been more of the question-and-answer variety rather than being driven by an analytical model. Medical diagnostic systems are a typical case (see van Melle, 1981, and Shortliffe, 1976).

The systems we shall discuss, however, are closer to the the type made familiar by the computer game Adventure. These systems are distinguished from both databases and analytical models in that, in addition to having a computable model of their universe of discourse (e.g., a database or simulation), each also has a process to explore the universe of discourse under human direction.

2. Intelligent Systems for Plant Management

In this section we consider intelligent systems for managing artifacts. An artifact (for example, an automated tool) is built by humans to achieve some objectives. Complex artifacts are built from a design specifying in detail what the artifact and the humans must do to meet these objectives. The design is a model of proper behaviour of the artifact and the human necessary to meet the requirements. Because the design is a model, it qualifies as a universe of discourse for an intelligent system to manage use of the artifact.

The control issues included in the design encompass more than conventional control hardware. As Len Pugh of General Electric has pointed out in a recent Liquid Metal Fast Breeder Reactor (LMFBR) program meeting, control includes the following items:
1. Administrative control, i.e., the decisions made by plant managers concerning operation and maintenance.

2. Scientific support for Item 1, i.e., the model contained in the design.

3. Implementation of Item 1 by plant personnel such as operators and maintenance crews.

4. Implementation of Item 1 by plant control hardware.

5. Implementation of Item 1 by plant protection hardware.

An intelligent system to help people perform the actions in Items 1, 3, 4, and 5 should also be based on Item 2; that is, it should help ensure that the plant is operated according to design. An intelligent system to deal with Item 2 is a Computer Aided Design (CAD) system. Because this report is concerned with aids to plant control and operation, questions of automated design are not considered in detail.

3. The Design as a Contract

A plant design is developed from a set of requirements, that is, a set of desired results of joint activity by humans and the plant. The design specifies in detail what the plant shall do and how it shall do it, and what the humans shall do and how they shall do it. It is therefore a contract between humans and the plant specifying actions by both to meet the requirements.

Within this framework the function of the intelligent system in plant management is clear. It monitors breaches (or potential breaches) of contract. It has responsibility either to correct the improper conditions itself or to report these circumstances to a human able to deal with them. If the intelligent system has autonomy over some plant subsystem, then it is a controller; if it does not, then it is an aid to operation.

This framework helps clarify one difficult issue about the function of computer aids to plant operation. A monitor for breaches of contract detects nonconformity to design; it should not be expected to correct for incorrect design. (Perhaps in the future we shall have CAD systems capable of response in real time to design faults; but then those same CAD systems might be able to avoid the design faults in the first place.)

Nevertheless, many unforeseeable consequences of design fault might be detected using simple algorithms and some fail-safe action taken either by humans or a controller. For example, if a small Loss of Coolant Accident (LOCA) had been considered possible in the plant at TMI-2, a fairly simple intelligent system could have been built to verify conservation of mass in the primary loop. The intelligent system would have inferred that either mass was not conserved or that there was a sink for coolant not present in the design (another description for a small LOCA). If the plant Piping and Instrumentation Diagram had been accessible in machine-readable form, an intelligent system could have generated a list of items for a human operator to check; the open block valve would have been among them.

Having agreed that intelligent systems are to detect breaches of contract, we must ask "How?" It is not possible to continuously measure every facet of the plant. Instead, a hierarchical approach is needed. The plant variables customarily measured should allow one to say approximately where an accident is about to happen; and optional measurements should allow a chain of measurements combined with inference, leading quickly to a failing component before the overall plant condition has become serious.
Inference-making, of course, is tricky. It seems to me that an analysis of Licensee Event Reports might well disclose that many such "events" arise not so much from lack of knowledge, but from failure by people to make possible and necessary inferences. Computers can make many more inferences per second than people and generally do not make false inferences. What is needed, then, are intelligent systems for automated inference-making. Ideally, such systems would "consider every possible case." In practice, however, some means is needed to make the computer "consider the most important cases first." Such systems are difficult to build and to use, but those that have been built (for instance, in medical diagnosis) turn out to be diligent and very useful servants.

4. A More Detailed Design Concept

Let us now consider a more detailed design concept. We have a plant comprising a small nuclear reactor used for experiments and controlled by a computer. The computer input is a specification of the desired neutron flux as a function of the time required to run the experiment. Output from the computer determines the positions of reactor control rods, which in turn determine the rate of change of neutron flux in the reactor.

We would like to detect failures in the control computer, the rod drives, and the reactor core. Both the plant design and the safety analysis done to qualify the proposal for the experiment show that, provided the flux follows the requested profile, the plant is not in danger even though some minor failure may have occurred. Clearly the proper strategy for assurance of plant function is to compare the desired flux and the actual flux. If they differ, something may be wrong; if they differ by very much, something is certainly wrong.

It is important to remember that the model and the plant will never match exactly. Therefore the simple three-port controller of conventional theory (which produces as output the difference between the actual plant performance and the desired performance) is unsuitable for plant failure detection. Instead one must test that difference against a statistical distribution; significant departure from the statistical distribution signals nonconformity to design. The test should, of course, sample the differences between predicted and desired behaviour at intervals long enough to ensure that the observations are statistically independent. These intervals are determined by the time constants in the plant equations of motion. The test should have the capability of detecting small drifts out of tolerance—a capability known to require many samples. Additionally, the test must detect radical differences very quickly, before a gradual drift out of tolerance. Thus, what is needed is not a fixed number of samples, but rather a continuous testing that the samples taken to date confirm that the plant is working properly. The necessary theory was developed by Abraham Wald (1945). It will be discussed in more detail later in this report.

Note that the intelligent system observing plant behaviour is driven by the same signal as a control system, namely, the difference between desired and actual behaviour. This exemplifies the similarity between control algorithms and contract monitoring algorithms. The outputs of the two computer programs are, of course, different: the control algorithm moves control rods, while the contract monitor reports "all is well" or "faulted plant module .... " as appropriate. Moreover, the output from the conventional control algorithm goes only to the control rod drives, while the output from the contract monitor is a signature for plant failure.
5. **Fault Tolerance**

An objection to use of computers in plant control is the possibility of fault conditions either in computer hardware or software. Simple control systems are preferable whenever possible because they will have fewer failure modes and these failure modes are likely to be well understood.

5.1. **Computer Hardware**

Hardware must continue to function correctly in spite of single-component failure. Moreover, it must be simple enough to make the probability of multiple-component failure small enough to meet safety requirements such as those necessary to license a nuclear steam supply system for an electric generating plant.

The issue of single-component failure is usually addressed by having several computers run in lock step and by monitoring differences between these computers. A computer system using multiple processors and some form of comparison between their activities must, of course, run the same software in each processor. Thus, the technique of difference detection is limited by “common mode failures”—that is, by the fact that a software failure will be common to all processors and will go unseen by the comparison process.

5.2. **Computer Software**

Obviously, only software of the highest quality should be used in fault detection hardware if the system as a whole is to be reliable. To ensure high quality, several criteria must be met:

1. Software requirements should be complete and drafted carefully.

2. The software should be formally proved to meet the requirements.

3. The requirements themselves should be formally proved to assure desired system function.

4. If possible, all proofs should be done mechanically, that is, by automated theorem proving systems. (Ideally, fault tolerance of the hardware should also be proved mechanically.)

At present it is difficult and sometimes impossible to meet these criteria. It is certainly expensive. Consequently, optimal system architectures should use as little fault-tolerant equipment as possible, and use it to detect and correct faults in the rest of the plant.

In any case, it should be remembered that if the standards are relaxed, even for sound economic reasons in some particular case, they should always be prominent among the stated goals of a project, and a decision not to meet them must be made carefully and recorded clearly.

---

*We note that hardware systems designed to be fault tolerant are in limited use for real time management of plant such as large aircraft.

** The problem of reliable difference detection is sometimes called the “Byzantine Generals Problem,” from a formulation in terms of decisions about strategy by an army commander who knows that some of his generals are traitors but not which ones. For background, see Kolata, 1984.
6. Statistical Method for Fault Detection

The method proposed for general use is Wald's Sequential Probability Ratio Test (Wald, 1957). This in its scalar form assumes that a variable is being measured and that all measurements are independent. It tests two hypotheses:

1. $H_0$, the hypothesis that the variable belongs to a given distribution $D_0$ (the unfaulted case).

2. $H_1$, the hypothesis that the variable belongs to some different distribution $D_1$ (the faulted case).

Generalisations are possible. For example, not all the parameters of $D_0$ need to be known, so that it is possible to use the test to estimate the mean of $D_0$ and detect drift in a measurement. Those methods are not discussed here.

After each measurement, the test procedure may reach one of three conclusions:

1. The variable belongs to $D_0$.

2. The variable belongs to $D_1$.

3. The measurement does not confirm either hypothesis unequivocally.

The test process contains two parameters "a" and "b" besides the description of the two distributions. These are the acceptable probabilities of wrongly concluding that a measurement confirms an hypothesis. The parameter "a" might be the acceptable probability of concluding $H_1$ when $H_0$ was true, i.e., the probability of false alarm if $H_1$ is the faulted case. Then "b" would be the acceptable probability of failing to alarm in case of a fault.

For example, suppose $H_0$ is the hypothesis "x is normally distributed with mean 0 and variance 0.1." Then $H_1$ is the hypothesis "x is normally distributed with mean 1 and variance 0.3." Now suppose we observe "x = 0.3." The measurement is slightly more likely to belong to the faulted group than to the unfaulted one, but the case is not unequivocally made. If the parameters "a" and "b" were set so as to accept a fairly high probability that a false alarm might be made, the statistical test might return an indication of fault. If "a" and "b" were set to accept only small probability of false alarm, then the test might return a request for another observation.

Note that use of this test makes important assumptions. First, the faulted and unfaulted behaviour of the plant must be known so as to derive the distributions assumed in the two hypotheses. Second, an explicit decision must be made about acceptable probabilities of false alarm or no alarm. By and large, demanding that these must be small forces the algorithm to make more tests before reaching a conclusion; that is, if an unfaulted plant suddenly becomes faulted, setting a requirement that the detection algorithm "be sure" before alarm will cause occurrence of several "equivocal, take another measurement" responses before concluding a fault is present.

For given distributions $D_0$ and $D_1$, and given acceptable probabilities "a" and "b" of false conclusions, the expected number of "equivocal, take another measurement" responses caused by a fault may be calculated. If the inference system returns many more than this number, it is probable that the actual distribution of the measured variable is neither $D_0$ or $D_1$. Within our framework, this is a design error in the plant: it has failed in an unpredicted mode, since the inference system is indicating the plant is neither unfaulted nor faulted. Because of this, the alarm system working from the results of the inference system should have four states:

1. Plant is not faulted.
2. Unable to conclude plant is not faulted.

3. Plant is faulted in known failure mode.

4. Plant is faulted in unknown mode (indicated by a persistent state 2).

The details of the theory for various assumptions about DO and D1 are not presented here; they are to be found in books on statistics. See, for example, Dixon and Massey, 1969, the most "readable" of the standard texts; Hoel, 1962—a standard reference for the reader inexperienced in statistics, but at home in applied mathematics or theoretical physics; Kendall and Stuart, 1968-83, especially Vol. 2, which has a detailed discussion of sequential tests, as well as a number of robust and distribution-free procedures possibly suitable for use in proved reliable software; Wald, 1947—a text concerned solely with this branch of statistics, known as sequential analysis; and Wetherill, 1975—the most useful "handbook" for sequential sampling, with a discussion about the SPRT for testing whether equipment is within tolerance.

7. A Conceptual Design

The example of the small experimental reactor given earlier is in fact close to a system actually used in a reactor now being upgraded. However, an example from thermal hydraulics is perhaps a better instance of use of the concepts.

7.1. The Requirements

The principal requirement is to shut down a reactor in the event of a pipe break or pump failure in the primary coolant loop. For simplicity (to avoid questions of two-phase flow), we assume that the coolant is liquid sodium and the reactor a pool-type LMFBR. On the outlet side of the main primary pump, a small branch pipe is taken off to cool the blanket, carrying a small percent of the total flow. A throttle valve is installed on the branch pipe to control the flow through the blanket. This valve is close to the main reactor vessel, and there is a substantial length of small pipe between the throttle and the takeoff from the main pump outlet.

7.2. Input to the Plant Model

The following measurements can be easily made:

1. The KVA and phase angle of the main pump.

2. The RPM of the main pump.

3. Motor torque (measured by strain gauges installed on the pump motor mounts).

4. The pressure difference across the pump.

5. The pressure difference between the outlet side of the pump and the inlet of the throttle on the small branch. The small branch is taken off very close to the pressure sensor at the main pump outlet.

The measurements assumed above allow simple models of plant to be used for inference about faults. Fewer measurements necessitate more complex inference. The assumptions here allow a straightforward discussion of fault diagnosis.
Additional simplifying assumptions are as follows:

1. Flow in pipes obeys Bernoulli's Law exactly, so that measurements of dynamical similarity to correct the Bernoulli equations are not needed.

2. Similar complications to correct for loss in Tees and other components may be left out.

3. Efficiencies of motors and pumps in unfaulted modes are between 95% and 100%.

4. A reasonable variety of statistical inference mechanisms are available for use.

All of the thermal hydraulic assumptions above are idealisations of the truth. This is not a serious matter, provided that the real plant thermal hydraulics leads to off-normal scenarios having sensor signatures suitable for use by the statistical inference system in fault detection and identification. In an actual conceptual design the early work might assume that Bernoulli's Law holds exactly. This will reveal those parts of the model where special care must be taken. The bottom line is always whether the statistical inference system can detect and alarm a fault within the time needed to allow proper corrective action.

7.3. Fault Modes

Several fault modes are assumed:

1. Seizure or partial seizure of a motor bearing leads to an efficiency of less than 80% in conversion of electrical energy to mechanical work by the shaft.

2. Breakage of the shaft between pump and motor leads to zero mechanical work being done by the motor shaft.

3. In the absence of pipe break at the pump outlet and concurrent partial seizure of a pump bearing, a pump failure leads to change in mechanical work done by the motor, and an effect on shaft RPM or motor torque or both.

4. Where a pipe break occurs close to the pump removing the hydraulic load, with simultaneous displacement of the pump causing partial seizure, the mechanical load on the motor may remain unchanged.

5. A pipe break in the main loop leads to greatly reduced hydraulic load on the pump.

5. A pipe break in the branch cooling the blanket between the main pump and the throttle valve leads to low pressure at the throttle valve.

6. A pipe break in the branch beyond the throttle causes low pressure in the sensor on the throttle outlet.

7.4. Detailed Models of a Fault

In this section we present two different models of a fault.
7.4.1. Mechanical Fault

Let us first consider the possibility of either a seized bearing or a shorted turn in the motor driving the primary pump.

If the KVA at the motor terminals and its power factor are measured, the electrical input power to the motor is known. If the motor torque and the motor RPM are known, the mechanical output is known.

The efficiency of the unfaulted motor is assumed for our example to be 97.5%. The ratio of electrical input to mechanical output must then be 0.975. We will assume that the statistical distribution of error in measured efficiency is normal, with a standard deviation of 0.5%.*

Having this information, a suitable statistical test must be devised. In addition to the most commonly used tests, there are a variety of less well-known techniques that may apply in special circumstances. In our case, however, the conventional test is good enough. It involves the hypothesis that a variable belongs to a normal distribution with mean in some range. The faulted hypothesis might be that motor efficiency is less than 0.80. However, for faults that occur gradually (i.e.; where the motor efficiency falls slowly from 0.975), the statistical inference will indicate neither clearly faulted conditions nor clearly unfaulted conditions. This is an administrative and plant design question. The answer is determined by the level to which motor efficiency may fall before the reactor should scram. A reasonable design might annunciate a motor efficiency not clearly in the unfaulted distribution, and scram on reaching the clearly faulted level of 80%.

7.4.2. Thermal-Hydraulic Example

Now let us consider a break beyond the throttle valve in the small pipe cooling the blanket. In this case, the primary signature of the fault is assumed to be given by the thermal hydraulic model as a 40% increase in flow down the small pipe to the throttle. Bernoulli’s Law would show a doubling of the pressure difference between the two sensors. Whether or not this can be detected quickly by the statistical inference system depends on the variance of the pressure difference for the unfaulted system. If the faulted case is three or four standard deviations away from the unfaulted mean, it will usually be seen very quickly. If both faulted and unfaulted condition both have the same standard deviation, and the difference in their means is of the order of a standard deviation, then a fault can still be seen but only after a number of measurements.

7.5. Formal Theory of Plant Models and Fault Identification

The fault detection process outlined above is described by a continuous semigroup. This means that the plant has a state space and that if the plant is at some point “P” at time “t,” then if it moves a small amount as the clock moves from “t” to “t+dt1,” and then another small amount as the clock moves from “t+dt1” to “t+dt2,” the net move is the vector sum of the two individual moves.

A plant model implements another semigroup, representative of the plant but simpler and with a smaller state space. This semigroup is said to be homomorphic to the true semigroup of the plant. Motion of the plant in its state space is mirrored by motion of the model in its state space. The connection between the two spaces is that if one takes an infinitesimal motion of the plant, the corresponding infinitesimal motion of the model is a linear transformation of the motion of the plant—usually of much less than full rank, of course, because the model state space has smaller dimension than the plant state space.

*It should be noted that the model does not apply during motor startup.
The preceding argument suggests that if a plant model is defined whose state variables can also be measured on the plant, then the plant and model will evolve together in time. That is, the point representing the model and that representing the plant should remain close together.

In practice, however, the plant and model will drift apart, as a result of errors of measurement, unless the motion of the model is in some sense well-conditioned. This is usually achieved by making some measured plant variables inputs to the model. The model outputs in this case must, of course, be well-conditioned functions of the model inputs.

For such a model, the difference between computed state variables and plant state variables will remain small unless the plant fails so as to introduce discrepancies between plant and model. The set of faults detectable by the model are those for which the motion of the model differ from the motion of the faulted plant.

In practice, one finds that various scenarios at some plant operating point lead to motions along different non-orthogonal directions in the space of difference between plant and model. If the dimension of the model space is larger than the number of fault scenarios, then a mapping can be found to make the motions in the difference space orthogonal for the different scenarios. If there are more scenarios than dimensions, then the best we can do is to arrange the mapping to make motions for the scenarios as well separated as possible, and test the hypothesis that the point in difference space is moving along one of the several possible paths. Whether or not such a test can be successfully made depends on whether the mapping is well-conditioned, how well the model represents the plant, and the signal-to-noise ratio in the measurements of plant state.

8. A Suggested System Architecture

In this section we elaborate on the question on software faults and present a possible design for fault detection.

8.1. The Question of Software Failure

It was remarked earlier that software faults may defeat the redundancy protection of multiple-processor systems running in lockstep. The situation is not as bad as it seems, however, for even if an error is present in a software module, if it is never encountered while running the jobstream, it will never affect the computation and is therefore harmless. Thus the quality assurance task for software is not to make certain it is perfect, but rather to make certain it does the job needed to run the plant.

A first approximation to doing this is to prove formally that a program meets its specifications. A second and more difficult task is to prove formally that the specifications are sufficient to ensure the plant is run safely. Because proof by humans is subject to error, one would like to have the proof itself performed by a computer. If this is done, the question still arises whether the proof is correct.

Fortunately the things computers do well and the things humans do well are somewhat complementary. Given a set of axioms and a theorem to prove, if a theorem-proving program supplies a proof, the following may be said about that proof. First, exactly the same process is used to derive complex proofs as is used to derive simple ones, and the kinds of "careless errors of detail" made by people are extremely unlikely in a theorem-proving program that runs the test suite properly. Second, the mechanical prover may be told to print the steps in logic used to derive the proof claimed to be true. These may be inspected by humans. It is our experience that a fault in a result from a mechanical theorem prover is easy to recognize. In practice, this means that mechanical proofs checked by people are most unlikely to be wrong.
Another question arises: "What are the assumptions used in a proof about a software module?" At present, we assume the following:

1. That the FORTRAN compiler (at present we are working with FORTRAN programs) conforms to the ANSI standard. Given plenty of operational experience with the compiler, a high degree of confidence may be developed about this assertion, even though it may not have been proved. In the future, we may be able to prove such assertions about compilers, but that is a long way away.

2. That the machine arithmetic processing the machine representation of real numbers has certain properties making it a reasonable approximation to real arithmetic. These properties are negotiable to some extent (proofs possible for machines whose arithmetic conforms to the IEEE standard may be untrue for older hardware designs), but even old designs are often amenable to weaker but adequate proofs. These facts lead to a preference on our part for computation using integers where possible, but this is by no means essential.

Unfortunately, proofs of properties for programs are difficult and expensive: a representative guess is that it costs a hundred times more to prove useful properties about a program than it does to write the program in the first place. Designs requiring the minimum of "proved" software are therefore the most attractive.

8.2. A Possible Design

A software failure in the detector of difference between plant and model has more serious consequences than a failure of the model leading to differences from an unfaulted plant. The failure of the model will lead to a false alarm, whereas failure of the difference detector can lead to no alarm when an alarm should, in fact, be made. Hence, if a choice must be made for economic reasons between proof of correctness for a model and proof of correctness for a difference detector, the proof for the difference detector should be preferred. In addition, difference detectors seem likely to be generic, and so the cost of proof might be distributed across many plants. (In fact, one of our long-term objectives is to provide a library of proved subroutines for a set of tasks in statistical inference.)

Possible failure of the model is not without consequences. Unnecessary plant shutdowns are likely to be costly. There are also reasons connected with expected availability of other plant safety systems that make it desirable to avoid false alarms. What needs to be done, then, is to make the plant model more reliable. This improvement does not have to go so far as formal proof to be useful, only far enough to make shutdowns from software failures in the plant model unimportant.

The fault detection software has three possible outputs: not faulted, unquestionably faulted, and possibly faulted (take another measurement). These are the values of a three-value system. Some software doing pure logic must take these outputs of the fault detector and reach a binary decision—whether or not to shut the plant down, for example. This software must also be proved to meet its specifications. However, since the specifications are in terms of pure logic, the proof should be straightforward.

The conclusion from this is that one possible design might separate the plant models from the inference mechanisms for fault detection and the three-valued logic following them. One would then perhaps have a set of proved inference mechanisms and three-valued logic programs running in ultrareliable hardware. The rest of the software monitoring compliance to specification would run in a plant data-acquisition system, possibly replicated but not running in lockstep. The outputs from this possibly triple data-acquisition system concerned with compliance would then be sent to the ultrareliable statistical inference and action system.
9. A Programming Language

If one goes to the expense of building fault tolerant hardware accommodating statistical inference tests and the three-valued logic following them, then it seems reasonable to ask that the software used in the system should be proved to conform to specification.

One's first thought is that FORTRAN is a suitable programming language. The following arguments suggest this may not be true:

1. A FORTRAN program usually needs a manufacturer's operating system to support it. Therefore, proof of properties includes proof not only of the FORTRAN compiler, but also of the run time library and the operating system environment. This is an impracticable task, and forces us to rely on operational experience for confidence in all but the application.

2. The FORTRAN application is likely to use floating point arithmetic. The properties of a manufacturer's floating point hardware vary a good deal from machine to machine. Because the life of a computer is between ten and twenty percent of the life of a plant, control algorithms may need to be ported across machines. This process may invalidate a proof.

3. The algorithms commonly used for statistical inference are simple enough so that they may often be coded in assembler. Implementations in assembler do nothing to improve portability; and, because of the large number of instructions, tend to make proof impractically expensive.

These arguments suggest that a "portable assembler" of some kind might be a useful tool for implementing provable inference and control algorithms. Such a language has been in use for real-time control since the 1960's. It is called FORTH, and it has properties making it an interesting possibility as a provable language for small application programs such as the statistical inference mechanism and simple laws of physics such as conservation of mass. The essential properties of FORTH, from our point of view, are the following:

1. FORTH has its own operating system environment: it does not need a manufacturer's operating system for its support.

2. The language is an extensible instruction set for a stack machine, with about a dozen primitive operations coded in assembler. These perform operations like adding the top two integers on the stack and putting the result back on the stack, or fetching the contents of a named storage location to the stack. In this sense FORTH is a software implementation of a "reduced instruction set" concept.

3. A production implementation of FORTH will have a hundred or so elementary operations, complex ones being defined as sequences of simpler operations, the chain of definition ending ultimately in the dozen or so primitives coded in assembler.

4. Applications are written by defining new operations in terms of existing ones. The hundred standard FORTH operations are rich enough to allow writing quite complex programs in moderately few lines of code. For example, the standard text editor provided in a FORTH implementation is about 300 lines of code.

Admittedly, FORTH code is not as easily readable by people as is FORTRAN or BASIC. It is, however, very easily readable by computers. A FORTH opcode is defined in terms of simpler ones by a statement like
which says "to perform OP, first perform S1, then S2, then S3, and last S4."

These semantics map well into the language of logic programming and first order predicate calculus: the goal OP is the ordered satisfaction of the subgoals S1 S2 S3 and S4. Thus it should be more straightforward than for many other programming languages to determine the "meaning" of a program and show it is equivalent to a requirement. Recent work by Barrow [1983] on verifying that a hardware design meets requirements seems to apply to the verification problem in this case.

Because FORTH traditionally has been used on small computers having no floating point arithmetic hardware, the FORTH user community is accustomed to solving real time control problems in fixed point arithmetic, and FORTH implementations have special features in their instruction sets for this purpose.

With the advent of the IEEE standard for floating point arithmetic, and its implementation in VLSI chips, it is not so obvious this is necessary or even important. However, most control systems have actual digital signals with limited dynamic range, and real time control has been done in FORTH using rational fraction arithmetic for almost two decades. A decision to use rational fractions in computation, and not floating point arithmetic makes the computation exact in a rational fraction representation of the physical world.

Issues such as overflow and underflow must still be addressed, but they seem possibly more manageable. The whole question of rounding error then becomes one of the extent to which the rational fraction arithmetic properly represents the real arithmetic of the physical universe. That is, it becomes a question for the plant physicist and engineer in the context of a particular problem rather than a question about the representation of real numbers by finite strings of bits.

Such a change of perspective seems to have consequences for plans to prove correctness. If a program is to be verified by somebody reading it and reasoning about what it does, there seems no doubt FORTRAN is a superior language, and PASCAL or PROLOG better still. However, if a program is to be read by another computer program for this second program to reason about the first, it is at least possible that FORTH is a superior language.

The usual approximations to elementary functions are rational function approximations. Although these are in floating point arithmetic, it is possible that computation as rational fractions in integer arithmetic could be used in control tasks. There are enough potential difficulties in this to require investigation to confirm the conjecture, and it is likely that the method should be used only for those applications where properties must be proved. The engineering tradeoff is that the cost of software development would be increased, but the cost of proof might be substantially reduced.

Acknowledgments

A short account of the ideas in this report was presented at the Workshop on Large Plant Control Systems, Oak Ridge National Laboratory, February 14, 1984. Encouragement from R. Kisner, A. D. Alley, and other participants to expand a more detailed account is gratefully acknowledged.

Discussions with friends have helped develop the ideas presented here:

Insight about analytical redundancy and its variants comes from John Deyst, John Hopps, and others at C.S. Draper Laboratory. Brian Smith at Argonne has spent numerous hours discussing proof of correctness with me. Janet Anderson at Argonne has been most helpful in finding references to SPRT technology and in giving expert advice about statistical questions in general. Friends in the nuclear reactor
community who have helped form my opinions, sometimes by opposing them, are Dave Burrows (ANL-W, TREAT), Jack Mott (GE, LMFBR), Joe Cook (ANL-E, TREAT/CSD), Walt Lipinski (ANL-E, TREAT/SAREF), Tony Valente (ANL-E, TREAT/SAREF), Greg Chisholm (ANL-W, EBR-II), and Leo Beltracchi (NRC). Jack Mott and Dave Burrows merit special mention for having suggested use of FORTH. In the context where the suggestion was made, I was and still am disinclined to listen; but I am sure the seed was planted for my present view of FORTH through their ideas.

Gail Pieper, our technical editor in MCSD, has helped greatly in preparing this report. A document such as this, attempting to develop new ideas and communicate them to a diverse group, needs special care to make sure that what is understood by the reader is what was intended by the author, and that the author properly understands the material being dealt with. Gail's practical help in making this document usable by a diverse audience is hereby acknowledged; without it, I would have failed in my purpose of technology transfer.

It also goes without saying that responsibility for remaining faults either of fact or of expression is mine alone.

References


Wald, A. 1943. Sequential Analysis of Statistical Data: Theory. A report submitted by the Statistical Research Group, Columbia University, to the Applied Mathematics Panel, National Defense Research Committee, Sept. (The technology was declassified in May 1945 and a report published by the Columbia Group as a whole. The method was discovered by Wald, but he remarks that the problem was posed to him by Milton Friedman and W. A. Wallis, saying that their clear formulation of the problem gave him a good start on finding the solution.)


Distribution for ANL-84-29

Internal:
J. R. Gabriel (50)
A. B. Krisciunas
P. C. Messina
D. M. Pahis
T. M. Woods (2)
G. W. Pieper
G. H. Chisholm (2)
R. W. Lindsay
L. R. Monson
J. I. Sackett

ANL Contract File
ANL Patent Department
ANL Libraries
TIS Files (6)

External:
DOE-TIC, for distribution per UC-32 (178)
Manager, Chicago Operations Office, DOE
MCSD Review Committee:
J. Bona, U. of Chicago
J. C. Browne, U. of Texas, Austin
G. Golub, Stanford
S. Gerhart*, Wang Institute, Tynsboro, Massachusetts
L. P. Kadanoff, U. of Chicago
W. C. Lynch, Xerox Corporation, Palo Alto
M. F. Wheeler, Rice U.
D. Austin, BES-DOE
C. Oliver, BES-DOE

L. Beltracchi, NRC
D. Cain, E.P.R.I., Palo Alto
J. Deyst, C. S. Draper Laboratories