MODELING COMPUTERIZED DEVICES IN MAINTENANCE TRAINERS

Randy Saunders, Principal Software Engineer

HONEYWELL INC.
Training and Control Systems Operations
1200 E. San Bernadino Road
West Covina, California 91790

ABSTRACT

Widespread use of microprocessors in military systems brings new challenges to the development of maintenance trainers. Modeling of analog phenomena requires accurate translation of continuous physical equations into computer form. Modeling the computerized devices requires significantly different skills than would be required to handle purely analog devices. An effective approach to integrating the digital and analog system models is presented here, and the benefits to maintenance training are illustrated.

INTRODUCTION

The impact of computerization, particularly through the use of microprocessors, on training has been primarily seen in four areas:

- New failure modes of the target system
- More complex problem symptoms
- Computerized fault testing and analysis
- Computerized maintenance assistance

These areas will continue to increase the number and variety of training situations that must be presented. The trainer must simulate not only the system as seen through the computer readouts, but also the situations where the computer or its software is the faulty element. Computer failures can be particularly difficult to simulate because the computer itself is usually the diagnostic tool. Computer failures can often be identified only by correlating many problem symptoms.

Traditional modeling techniques rely on characterizations of the system in terms of continuous equations, usually differential equations. The assumption of continuous functions is necessary for the development of a digital computer model of the system. Even stochastic processes can be adequately modeled using probabalistic equations. This approach works well in areas where the underlying physics is well understood. Very complex engine systems and flight dynamics have been handled using these techniques.

These modeling techniques do little, however, to model even the most simplistic of computer phenomenon. A simple keypad with enter, rubout, and ten digit keys presents non-linear behavior. The boundary conditions of too many digits or too many rubouts cannot be handled by a continuous model. However, a simple transition diagram or Petri net will completely define the system. These discrete mathematical tools do nothing to solve the continuous systems described previously.

These different disciplines can be used together to model a complex analog system with embedded digital elements.

Combining these techniques into a complete modeling package requires the development of a consistent approach and data structure. The basic concepts of these different techniques are described below.

CONTINUOUS MODELING

The continuous modeling process addresses four major issues:

- Representation of physical phenomenon in mathematical terms (fidelity)
- Serialization of the mathematical representation so that it can be effectively computed (feasibility)
- Computation time considerations changing the model (practicality)
- Partitioning the computational sequence so system timing constraints are not violated (efficiency)

Each of these issues presents a point of tradeoff between accurate representation of the system and efficient use of available computer resources.

Representation

Developing a mathematical representation of the system is the first step in developing a computer model. The information collected is usually the first source of errors in the model. The heat flow in a modern jet aircraft engine is so complex that closed form representation is impractical. Even if it were possible the detail would overwhelm the largest of computers. The only reasonable approach to modeling this, and most other complex mechanical systems, is to correlate the available physical data into a consistent representation. Correlation causes some information to be discarded or averaged out of equations into "magic constants". These frequently require rework or tweaking to get the right answers out of the model, at considerable cost. A necessary element to an improved modeling system is the control of this cost.

Consider the case of a direct reading oil pressure gauge. This is a needle attached to spring and a small cylinder. Pressing against the cylinder is a column of oil which runs through tubing to a pressure port inside the engine. The pressure of the oil elasticly deforms the cylinder. The spring, through a set of gears, holds a needle mechanism against the cylinder. As the cylinder deforms the mechanism moves, turning the gears and finally the needle. The indication is linearly related to the pressure if the following non-linearities are ignored: compression of the oil, fluid friction on the oil, strain hardening in the cylinder, slack in the gears, the weight of the oil in the column, the weight of the spring mechanism, the weight of the needle, and many others. Taking these factors into account could require a ten page model for a simple oil pressure gauge that is supposed to read two degrees for every pound per square inch!

Serialization

The representation of a system consists of a set of mathematical relationships which are CONTINUOUSLY true. This directly conflicts with the idea of a simulation based on a digital computer. It is not possible to solve all equations simultaneously. This means that the order of solution must be determined. The order may or may not make a difference, and the order may be subject to change as simplifying assumptions are replaced by new relationships. Changes in solution sequence represent another cost that must be managed.

The simple example of the oil pressure gauge seems straightforward to serialize. The initial model is that (degrees) D = 2*P (pressure in PSI). This can be calculated following the equation that defines P. However, it may not be logically related to the other functions in this group. Locating this operation with other logically similar functions such as instrumentation introduces the possibility that the oil pressure reading may not accurately reflect the current pressure. This is impossible in the real system and could seriously impact the fidelity of training. The issue of how much fidelity is required for adequate training is too subjective to address in so small an example. The ability to easily increase the fidelity in problem areas is much more important. Changes are more difficult to make when the serialization is difficult to understand.

Computability

A full serial representation may be possible to construct, but impossible to use within the trainer's hardware constraints. The use of larger computers is precluded by the cost of these machines. Reduction of the complexity of the model by making simplifying assumptions is more economical than buying more hardware. Preserving the fidelity of the simulation is necessary when making these simplifications.

Partitioning

Once a computable serialized representation has been developed, it is unlikely that all the stages of the algorithms will require the same sampling rate. Sampling is the process that maps the effects of time onto the model. In general the model will be run periodically to update values. The amount of CPU resource available in this period is likely to be insufficient to run the full model for a complex system. However, not all parts of the algorithm require the same sampling period to achieve acceptable output accuracy. The partitioning

process isolates submodels by sampling rate.

The oil pressure gauge can be functionally related to a fuel quantity gauge. While oil pressure may change suddenly indicating some action needs to be taken fuel quantity is a much slower changing phenomenon. These two model parts do not require the same sampling period to present equally accurate information.

Maintenance Training

These four areas impact maintenance training additionally in the need to accurately simulate the system failure modes. Information on faulted operation is usually much less detailed than that for normal operation. Failed systems are generally less stable than correctly functioning ones, resulting in aliasing problems in failure symptoms caused by sampling periods that do not accommodate the failure instabilities.

DISCRETE MODELING

Discrete mathematics is the theoretical basis for most computing algorithms. However, simulating a military avionics computer at the gate level makes as little sense as simulating an engine at the molecular level. The important use of discrete mathematics in military systems is for control algorithms. It is this use that warrants simulation algorithms based on the same principles. Among the many examples of this are:

- Finite state machines used to determine system modes and states
- Queueing theory used to regulate system service rates in varying conditions
- Petri nets used to synchronize and coordinate multiple processes in the operational system

By understanding, and making use of, the operational program structure a simulation can gain from the design and test work that went into the operational system. As programming practices become better standardized via the use of Ada and the software development standardization the opportunity to actually use the same control structures will be recognized.

State Machines

Two major benefits are attained by modeling the state or mode transitions of a computer system by a finite state automaton (FSA). The first is completeness of the simulation. The regular nature of the FSA control tables rapidly points out gaps in the model's coverage of the system. Even when these gaps deliberate (they represent functions not appropriate to the trainer), the decision is clearly stated. This clarity points to the second benefit, localization. Listing the modes and the events that cause a mode transition can effectively localizes these transitions into a few modules. This simplifies changes and enhancements of the model to broaden its use or accommodate system changes.

Queueing Theory

State or mode changes must be integrated with the system control activity. This control activity is frequently handled, both in operational systems and trainers, by a distributed computer system. In a distributed processing system, the allocation of information responsibility to computers is a major design decision. Although this bears no necessary correspondence to the processing distribution in a modeled system it gives an excellent definition of the objects upon which the model is to be based.

Modeling a software system by the pieces of data (objects) it uses and the manipulations (operations) it performs on them is referred to as the "object-oriented" approach. This approach leads directly to the use of queueing theory to analyze the way messages containing objects flow from operation to operation. This analysis provides much of the timing information needed to model a computer system.

The same analysis can be applied to the simulation computer program. Using the object-oriented approach in the design phase will improve the quality of the simulation. Isolation of model segments by the objects operated on allows testing to be carried out earlier in the development phase. Malfunction symptoms can be located directly with the model elements that define the normal system. Most importantly, response time can be improved by partitioning model components into several micro computers. This improves the time characteristics of the trainer and improves fidelity.

Network Analysis

Data flow network analysis can show possible reductions of the computation required during each time cycle. By extracting the data flow characteristics of a multi tasking software system it is possible to determine the frequency and priorities used in updating the system objects. This information can guide the modeler in optimal organization of the model's evaluation of the corresponding objects.

INTEGRATED APPROACH

The goal of recent Honeywell modeling analysis has been to develop a common environment where both discrete and continuous modeling can be used effectively. The major obstacles that were overcome are:

- Procedural structure of software restricts the use of multiple modeling techniques
- Different disciplines are required which are possesed by few modelers

- Some implementation specific knowledge would be required to accurately specify models

The key to resolving these difficulties was the introduction of automation to the modeling workplace.

Software Structure

The structure of most training programs is procedurally hierarchical. This means that even though the system is broken up into a tree of interfaces, control passes from one level in the tree to the next. This is fine for simulating monolithic software systems, but makes it very difficult to simulate distributed computing systems or analog systems. Forcing the model into a single task causes most of the fidelity problems discussed above in the serialization section. Operational software developers were forced to distributed architectures to improve response time. The same approaches must be taken by modelers working to simulate these systems.

The modern distributed system consists of a number of independent processes that operate, according to well defined rules, on a group of objects. The control of these objects and the run-time scheduling of the processes is handled by some supervisor process. The object-oriented approach is used in designing the simulation programs. This approach isolates impact each process can have on the rest of the system. This isolation mechanism makes it practical to combine programs operating under several different paradigms. This provides the first step in allowing the two modeling methodologies to be used together.

Common Operators

One of the first benefits of the object-oriented structure is clarity in the partitioning function. This function divides the work to be done into logically related tasks based on which objects are to be operated on. The time and control division is handled transparently in the supervisor process. Within each task the description can take any form appropriate to the function being modeled. Continuous methods can be used for areas where non-linear effects from microprocessors are not a problem. Discrete methods can be used to simulate the computer elements. Careful selection of the objects that are shared between each function makes this selection invisible outside the specific submodel.

Common Objects

The degree to which the conventions and assumptions of one task can be hidden from the others is controlled by the choice of objects. Objects must represent the actual form of the information, not just a data item that is easy to compute within a submodel. The impact of this is minor since these objects form the most readable interface.

The interface from submodel to submodel is handled by object service routines. These routines form a queue of objects to be handled by each model. The implementors need only be concerned with operations on these queues. The interconnection is handled by the queue server. This provides additional isolation between submodels. This isolation is particularly useful in the maintenance phase of the project when submodels can be replaced, or reused, without concern for their interconnection. Such isolation is not possible in an environment where interconnection is handled procedurally by calls from one routine to another.

MAINTENANCE TRAINING

Many maintenance training problems can be solved in this structure. One such problem is the simulation of faulty equipment. Isolating the objects and working on them in logically related groups simplifies the introduction of malfunctions, due to the close correspondence between the operational subsystems and the submodels. After studying how a malfunction affects a particular system the effects can be related to the objects of the model, and the tasks which generate them can be changed. Even if a malfunction requires completely different actions or modeling techniques than the normal operation, the worst case solution is to devise two submodels and handle the routing of messages depending on the presence or absence of the malfunction at run time.

Another problem in maintenance training is the increasingly widespread interest in computerized maintenance aids. Future maintenance trainers will not only teach repair of the actual system, but also operation of the maintenance aids. This new capability can be easily translated into an interface with an existing process. The objects used by the maintenance aid can be very simply defined since its operation should be quite simple for it to be effective. These objects, although they have no submodels which directly use them, can still be seen and controlled through the same mechanisms used for the rest of the model. This has particularly interesting applications in the area of retrofitting the use of aids to trainers designed in using the object-oriented approach. Much of the a procedurally interconnected trainer's value would be lost if maintenance aids were introduced for use in conjunction with it because it would be necessary to communicate with the maintenance aid simulation from every segment of the existing model. This would involve modification of every routine. An object-oriented system can reroute the objects used by the maintenance aid model

without changing any of the existing models.

Implementation Detail

A major problem with both discrete and continuous modeling is the need to understand the implementation methods in order to know what simplifications will allow the model to run in the machine provided. The object-oriented approach does not eliminate this problem. It does provide an easy and consistent method limiting the scope of the change to only a small set of submodels. The implementation details are not needed in advance, but when restructuring is needed only the models themselves are changed; the interconnection follows automatically. The same argument applies to model changes required to increase the detail or fidelity. A technique that can effectively deal with the time constraints on a model is yet to be developed, although experience shows that the object-oriented approach does not use significantly more time than procedure call based approaches.

Hardware Interface

One of the stickiest areas of the integration between the model and the final trainer is the hardware interface. The object-oriented approach completely removes this concern as a byproduct of the isolation mechanism. The programs that interface with hardware are just a different group of tasks in this scheme. The model always works with a set of objects defined by the modeler. The interface tasks map the object to electrical signals required to operate the hardware and vice-versa. The number of integration errors can be reduced by clear definition of the objects, but more significant reductions in problem repair time result from having a single, well defined location to make interface changes.

Automation

This integrated approach divides the modeling process into a large number of individually simpler tasks. However, the amount of detailed information resulting from the modeling effort increases. Correct transfer of this information from the modeler into the application program is a considerable risk. The modeler needs computer automation to aid in this detailed tracking. Assigning the maintenance of these minor details to a computerized system significantly improves the consistency of the documents and models over those generated manually. Names are more consistent and the I/O interfaces are much more specifically defined. Avoiding these potential ambiguities allows many problems that would normally be found in integration to be resolved in the much less costly specification and design phases of development.

Work is currently underway to expand the scope of the automation to include the automatic generation of modeling programs. This would combine with test case emulation capabilities to remove much of the software validation work from the actual trainer. The cost savings both in labor and investment in support hardware appear to be significant.

CONCLUSIONS

The object-oriented modeling approach forms a significant tool for incorporating many modeling techniques into a single trainer. It also leads to some significant implementation savings. Effective use of automation can hold the modeling costs constant while improving the consistency and modularity of the model. Interfaces to hardware and other trainer elements are isolated from the model, so fewer and less extensive changes are needed to support trainer enhancements or system changes. Malfunction support and fidelity related areas can be changed with minimal side effects. Most significantly, individual submodels are isolated so that changes in one, for any reason, do not impact the operation of others. This can lead to improvements in reuse as well as reduced integration costs. Honeywell has used this approach on recent projects. The capabilities available are being actively expanded through the development of more sophisticated automation in this area.

ABOUT THE AUTHOR

MR. RANDY SAUNDERS is a Principal Software Engineer with Training Systems. He is currently responsible for internal research and development in the area of Modeling Automation and Tooling. He holds a Master's degree in Engineering from Harvey Mudd College in Claremont California. He has been lead software engineer on the E-3A (AWACS) Radar Maintenance Training System.