Hybrid Modeling for Scenario-Based Evaluation of Failure Effects in Advanced Hardware-Software Designs

Jane T. Malin
NASA Johnson Space Center
2101 NASA Road 1, ER2
Houston, TX 77058-3696
malin@jsc.nasa.gov

Land D. Fleming
Hernandez Engineering, Inc.
17625 El Camino Real, Suite 200
Houston, TX 77058
land.d.fleming1@jsc.nasa.gov

David R. Throop
The Boeing Company
2100 Space Park Drive
Houston TX 77058
david.r.throop@boeing.com

Abstract

This paper describes an incremental scenario-based simulation approach to evaluation of intelligent software for control and management of hardware systems. A hybrid continuous/discrete event simulation of the hardware dynamically interacts with the intelligent software in operations scenarios. Embedded anomalous conditions and failures in simulated hardware can lead to emergent software behavior and identification of missing or faulty software or hardware requirements. An approach is described for extending simulation-based automated incremental failure modes and effects analysis, to support concurrent evaluation of intelligent software and the hardware controlled by the software.

Introduction

Intelligent software for systems management is typically designed to exhibit the emergent behavior that is needed for flexible and robust operation. Such software uses its resources to make the best of unexpected problems and opportunities. It can autonomously assess a wide range of possible states and contexts and to select appropriate procedure variants that are consistent with goals. In anomalous fault management situations, it may use models of expected system behavior to detect and diagnose degradations and failures and identify appropriate recoveries, consistent with the situation and current goals. When presented with human instructions to perform tests and workarounds, it will use model-based prediction and what-if studies for self-validation of novel procedure variants before taking action. The challenge is to evaluate this emergent behavior, to predict what might happen when the intelligent software executes its reactive plans and tasks in a new situation.

A combination of test and evaluation approaches will be needed for validation of advanced hardware-software designs. The joint behavior of the operating hardware and software must be analyzed, to not only verify the software requirements but also discover missing requirements. Many accidents and major losses have been due to flaws in software requirements (Leveson, 1995). Our experience with evaluation of complex intelligent software has shown that it can be difficult to envision effects of failures and recoveries in complex highly interconnected systems. Consequently, software requirements may not be developed to handle some of these situations (Malin et al., 1998). Therefore, some software problems are the result of missing or faulty software requirements.

We have been taking an incremental scenario-based simulation approach to evaluating intelligent software. Our hybrid models of hardware and operations have been used to evaluate intelligent software for autonomously managing advanced gas processing systems (Malin et al., 2000). The intelligent software (Bonasso et al., 1997) includes a planner, a flexible reactive sequencer, and low-level control. It can also include a model-based mode identification and reconfiguration module (Williams and Nayak, 1996).

One of the challenges of model-based analysis of planned operations is to focus the analysis on relevant events and states. We have been developing a function-based approach to this problem in our current work in the EPOCH (Engineering Product and Operations Cross-cutting Hybrid) Simulation project (Throop et al., 2001), which automates failure modes and effects analysis (FMEA).

Intelligent System Plans and Tasks

Typically, in management of planetary systems such as life support systems, the intelligent software concurrently performs multiple tasks. For example, the Interchamber Monitoring and Control Software for the Product Gas Transfer System (IMC-PGC) in the Lunar Mars Life Support Project Phase III test in 1997 (Malin et al., 1998) performed the following tasks.

- Manage transfer of CO_2 and O_2 between crew and plant chambers.
- Manage CO_2 and O_2 levels in the airlock and plant chamber before, during and after periodic incineration, by reconfiguring gas flows.
- Maintain CO_2 at appropriate levels for activities in the plant chamber.
- Maintain appropriate O_2 levels in plant chamber and pressures in storage tanks. (Independent conventional software managed the O_2 levels in the crew chamber.)
- Detect alert and alarm situations and respond appropriately.

These tasks were accomplished by reconfiguring gas flows and flow controllers, by inhibiting or starting gas transfers, and by interacting with safety software and human operators.

Dynamic Hybrid Simulation Approach

Simulation-based evaluation of the IMC-PGT intelligent software application was accomplished by running scenarios where the software interacted dynamically with the simulation model. The CONFIG hybrid simulator, which extends discrete event simulation with capabilities for continuous system modeling (Malin and Fleming, 1999), was used to model the life support hardware, controls and schedules, and plants and crew. For this work, an interface was established between the reactive sequencer and the system model, which included low-level controllers. CONFIG supports fast scenario-based simulation of systems in operation. The ratio of simulated time to real simulator execution time was generally 20:1. Nominal 120-day schedules were simulated, and IMC performance was evaluated for each software task.

Simulated hardware failures were inserted to test the sequencer under each off-nominal condition specified in the requirements. Anomalous conditions and failures in simulated hardware and control can lead to emergent software behavior. When the software appeared to not be operating properly, the problem was investigated and fixed. This approach uncovered problems that had been missed in the more conventional software testing. Some new requirements and design problems emerged in these scenarios.

In more recent work, the interface between the models and the intelligent software has been through the low-level control layer. With this low level interface, it is not necessary to model the low-level control software, and it is possible to test all the interacting modules of the intelligent software.

CONFIG provides capabilities to model failures of configuration, input, capacity, performance, control and operations. The discrete event base provides a framework for causal modeling of states and outcomes, and specifying transition functions that are internal or triggered by external inputs. The discrete event system specification formalism introduced by Zeigler (1976) specifies a system, with a continuous time base but discrete inputs and state transitions, as a structure:

$M = [X, S, Y, d_{int}, d_{ext}, \lambda, ta]$ where
X is the input value set (value-change events arbitrarily separated from each other in time);
S is the sequential state set;
Y is the output value set (value-change events);
d_{int}: $S \rightarrow S$, the internal transition function;
d_{ext}: Q x $X \rightarrow S$, the external transition function, where Q is the total state set = $\{(s, e) \mid s \varepsilon S, 0 \leq e \leq ta(s)\}$;
λ: $S \rightarrow Y$, the output function (value-change produced just before transitioning to the next state);
ta: $S \rightarrow R^{+}_{0, \infty}$, the time advance function (a positive real number for time to the next internal transition, computed upon transition into the state).

The time advance can be a function of d_{int}, d_{ex}, or λ Asynchronous changes in these systems are managed by an event scheduler with a variable time advance.

The continuous time base of discrete event simulation supports both event-stepped time advances and discrete-time-step approaches to continuous simulation. In CONFIG, a discrete-time-step approach, with either linear or exponential approximation, can be used to periodically update continuously changing variables in a component.

Discrete event models of systems are composed of coupled interacting component models. System behavior emerges from the coupled behavior of the components. In CONFIG, the model structure can be "recomposed" during a simulation as the direction and activation of the couplings changes. Parts of the model are activated or deactivated as operating modes of components change or closed off areas of the system are brought into the working system configuration. A global flow and pressure/potential analysis capability tracks dynamic changes in system configurations and model structure during simulation. This global analysis is based on graph searching and efficient computational network approaches. Recently, the coupled model approach has been used to support selection and mixing of simple and complex subsystem models in

simulation experiments. Selecting versions of subsystem models can focus and speed up simulations.

In CONFIG, components have multiple behavior modes. Each mode has state equations that generate behavioral effects, and conditions that govern mode transitions. CONFIG activity models support modeling of controllers, human operator procedures, actions and schedules. Object-oriented model types for devices support modeling of components.

In discrete event simulation, generation of events and time-advances can be random, supporting probabilistic analysis. In use thus far, CONFIG simulations have been deterministic, supporting analysis of specific state configurations and inputs. The intelligent software interacts with and manages the simulated hardware.

Figure 1 shows a CONFIG graphical interface for the current version of a model of an In-Situ Propellant Production (ISPP) system. This ISPP design includes a freezer for CO_2 acquisition (cc02), a Sabatier reactor for conversion of CO_2 and H_2 to methane and water, an electrolysis-based O_2 generation system (H2O-CELLS) and a zirconia-cell based system (Z-Cell) for generation of O_2 and CO. The gases are liquefied for cryogenic storage.

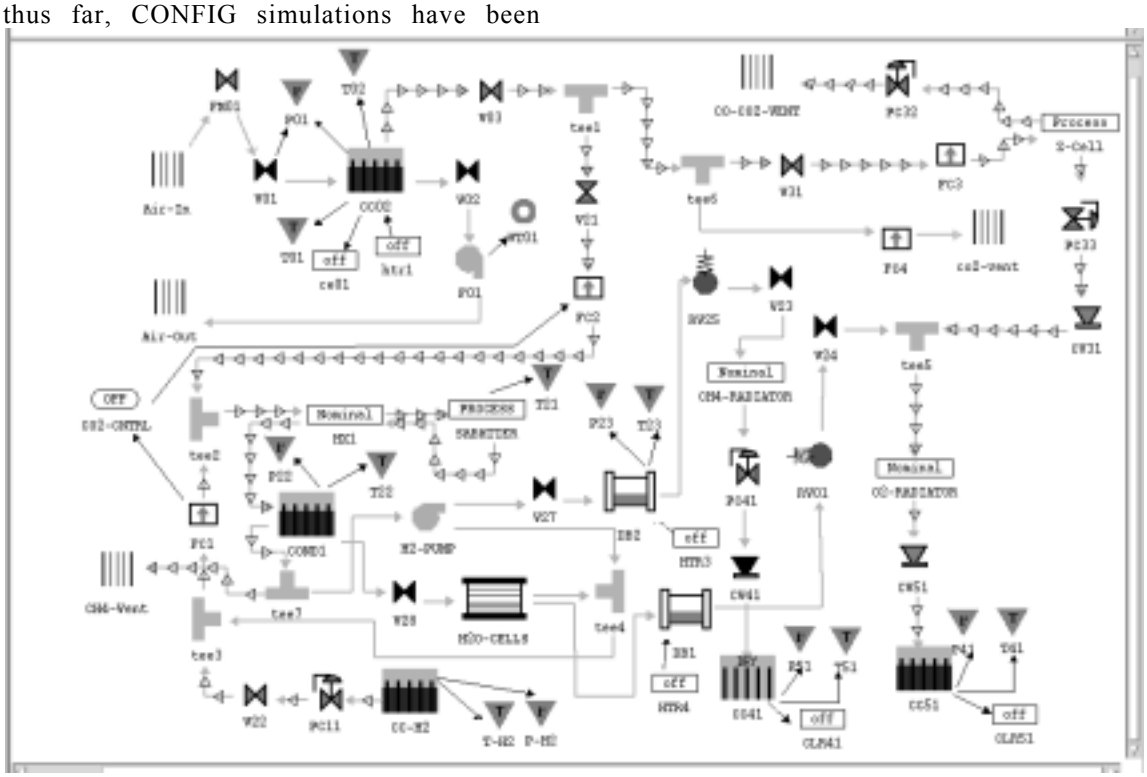

Figure 1. CONFIG Model of In-Situ Propellant Production System

Focusing Analysis with Functions and Scripts

Recently, there has been significant progress in qualitative and functional modeling. The current dominant concept of function is as a role in a design, or as the indicators of that role. The indicators of function are selected effects of a device that are relevant to a purpose or service of the design (Chandrasekaran and Josephson, 1996). These functional effects descriptions can be mapped to states in simulations of device behavior. The concept of function as mapping to simulation effects has led to practical approaches for automated design analysis and FMEA. We are developing an approach to model-based automated incremental FMEA, using a function labeling approach similar to that developed by Price et al. (1995).

In the EPOCH project, we are applying functional labels to hybrid quantitative simulation. An engineer can select failure modes and assign function and malfunction labels to selected states of system components. The functional label is mapped to a logical relation on a set of model variables that indicate that the function is occurring. Functional labels interpret the simulated system behavior, and focus analysis on the effects of failures on system functions. The behaviors are reported in terms of achieved functions, missing and delayed functions, malfunctions, and sensor indications.

We include scripts for simulation scenarios in the specification for an automated FMEA. These scripts manage dynamic simulation scenarios that produce the sequence of conditions and configurations that are

necessary for achieving functions in nominal operations. Scripts also dictate when off-nominal conditions and failure modes occur in the failure scenarios. In traditional manual FMEA and fault tree analysis, these complex interactions can be difficult for analysts to understand. This automated FMEA analysis can also be used to validate and further explore manually generated FMEA and fault trees.

Development of incremental FMEA reports follows a process of nested loops. Simulations are performed over a set of scripted operational scenarios, starting with a nominal scenario. Times are logged when function-related outcomes occur. CONFIG logs behaviors as traces of variable values. But this is too detailed for FMEA, which requires an abstraction from the numeric description to a functional description. Times are logged when function-related outcomes occur. Scripts are re-simulated for each failure mode case. It will be possible to randomly select the time of failures or out-of-tolerance inputs, and to randomly or systematically explore the space of effects of continuous degradations. Differences in the function logs (nominal vs. faulty) give the FMEA. The FMEA is regenerated for each design alternative or change, and only failure effects that have changed are reported.

Extending the Approach to Evaluate Intelligent Software

Ordinarily, automated FMEA would be carried out before intelligent software has been implemented. The same models and methods used for automated FMEA can be reused to evaluate intelligent software, even as it changes during a mission to handle unexpected contingencies. Intelligent software can interact with causal models in multiple scenarios where hardware failures and control errors produce emergent software behavior for evaluation. The software requirements indicate what operations to include in the scripts and what hardware failures and control errors to introduce. This type of analysis can also support concurrent engineering of system hardware and software, to take advantage of intelligent software capabilities. We have begun applying this approach to analysis of hardware-software designs for an In Situ Propellant Production System for exploration missions.

Coverage

This approach can combine the benefits of traditional manual software testing analogs of FMEA and fault tree analysis with systematic analysis of functions. What issues resemble the coverage issue for software testing?

EPOCH makes conditions for analysis explicit, as the set of test scripts. Are the scripts (the specified conditions) adequate to project all the effects at issue? A minimal set of scripts should exercise each function of the hardware-software system at least once, when run in the base case (with no failure present except ones in the script). Some scripts will include failures that trigger functions (such as redundancy, warning, safing) that are only exercised under failure. To evaluate intelligent software, the minimal set must include scripts that produce the degradations and malfunctions that the software is required to handle.

Software testing tests code against a specification. Incorrect specification of a script to achieve an intended function may be based on flawed design understanding. This failure to achieve the function can be observed in the scenario-based simulation, and corrected.

The specification can be incomplete because features of the software are not included in it. It can be incorrect because system complexity and interactions are not accounted for in design. It would be possible to omit, from the analysis or the model, "unintended" functions or malfunctions that are permitted by the devices in the system design. The engineer may also fail to map functional labels to the functions that the designer intended.

One yardstick for software testing is the extent to which code is exercised. The analogous measure for model testing is the extent to which the different device modes are exercised. A set of scripts need not exercise every mode for every component. There may be no plausible script that could cause a device to transition to certain modes. But logging unexercised modes should force the engineer to confront the question: "Are you aware that you've never tested this mode?"

Related Work

In our work, we try to span the gulf between development of large continuous simulation models and compact causal models for evaluating hardware-software systems that include intelligent software. We recognize the limits of traditional discrete event representations (Zeigler, 1976). We have tried a qualitative simulation approach using fuzzy-set-based quantization, and found it too cumbersome and limiting. We have not found other hybrid or continuous modeling approaches that smoothly handle dynamic model recomposition during the reconfigurations that are to be expected in practical systems operations and in failure scenarios. Because of the global effects of reconfiguration on systems with feedback loops, an approach using graph analysis of paths seems to be essential.

Beetz and Grosskreutz (1998) have developed compact causal models for temporal projection of robot plan execution. The effect and event rules in their causal models appear similar to discrete event simulation transition rules and event generators. They have developed a compact causal model for a small robot planning

problem. The system can identify possible plan execution failures by developing a small number of randomly selected scenarios that are influenced by exogenous events that can interfere with the concurrent behavior of the robot.

Conclusion

Intelligent systems that control complex systems are difficult to evaluate because the hardware and software components of the system can interact and produce cascading effects that are difficult to anticipate. Our approach addresses software-hardware validation at the system level, by operating the simulated system in a set of scenarios that focus on the effects of operations and perturbations on functional outcomes.

Use of a scenario-based simulation approach has identified system interactions that were unanticipated in the system design and were not identified during conventional software testing. Systematic use of this simulation-based approach will enable us to produce emergent behavior in intelligent systems and evaluate its effects on the functions of hardware-software systems.

References

Beetz, M., and Grosskreutz, H. 1998. Causal Models of Mobile Service Robot Behavior. In Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems. Also in *Linkoping Electronic Articles in Computer and Information Science*. 3 (5).

Bonasso, R. P., Firby, R. J., Gat, E., Kortenkamp, D., Miller, D., and Slack, M. 1997. Experiences with an Architecture for Intelligent, Reactive Agents, *Journal of Experimental Theory of Artificial Intelligence* 9:237-256.

Chandrasekaran, B., and Josephson, J. R. 1996. Representing Function as Effect: Assigning Functions of Objects in Context and Out. In Working Notes of AAAI-96 Workshop: Modeling and Reasoning with Function, 30-37.

Leveson, N. 1995. *Safeware: System Safety and Computers*. Reading, Mass.: Addison-Wesley.

Malin, J. T., and Fleming, L. 1999. Enhancing Discrete Event Simulation by Integrating Continuous Models. In Working Notes of AAAI Spring Symposium on Hybrid Systems and AI.

Malin, J. T., Fleming, L., and Hatfield, T. R. 1998. Interactive Simulation-Based Testing of Product Gas Transfer Integrated Monitoring and Control Software for the Lunar Mars Life Support Phase III Test. In Proceedings of SAE 28th International Conference on Environmental Systems. SAE Paper No. 981769.

Malin, J. T., Nieten, J., Schreckenghost, D., MacMahon, M., Graham, J., Thronesbery, C., Bonasso, P., Kowing, J., and Fleming, L. 2000. Multi-Agent Diagnosis and Control of an Air Revitalization System for Life Support in Space. In IEEE Aerospace Conference Proceedings 2000, (CD). Manhattan Beach CA: IEEE Aerospace Conferences.

Price, C., Pugh, D., and Hunt, J. 1995. Development of a Multiple Model Design Analysis System. In Proceedings of the 3rd International Workshop on Advances in Functional Modeling of Complex Technical Systems.

Throop, D. R., Malin, J. T., and Fleming, L. 2001. Automated Incremental Design FMEA. In IEEE Aerospace Conference Proceedings 2001, (CD). Manhattan Beach CA: IEEE Aerospace Conferences.

Williams, B. C., and Nayak, P. P. 1996. A Model-based Approach to Reactive Self-configuring Systems. In *Proceedings of the 13th National Conference on Artificial Intelligence,* 971-978. Cambridge, MA: AAAI Press.

Zeigler, B. P. 1976. *Theory of Modeling and Simulation*. New York: Wiley.