

Modeling and Simulation of Processors and Networks for Smart Navy Aircraft Components and Systems

Alan George, Michael Miars, Robert Todd, and Warren Rosen¹
High-performance Computing and Simulation (HCS) Research Laboratory
FAMU-FSU College of Engineering, Florida State Univ. and Florida A&M Univ.

Researchers in the HCS Research Lab are developing a variety of network and processor models to support the simulation of Smart Navy Aircraft Components/Systems (SNACS) using a coarse-grain, discrete-event, system modeling and simulation tool from NuThen Systems Inc. called Foresight. Foresight is being extended to develop a coarse-grain functional analysis system to portray the behavior of smart systems rapidly and accurately in various environments, and to extract performance and dependability measures from these systems. This paper summarizes on-going research and development efforts to date involving the design, modeling, and simulation of processors, networks, and embedded computer systems for smart aircraft components and systems.

Introduction

The aircraft research and development community is actively involved in the design and use of smart components in advanced vehicle management and avionics. Future military aircraft must perform multiple, complex, tactical operations correctly within reduced time frames and survive. A key aspect of future air vehicle developments, which will allow such flexibility, is the determination of the critical technologies required and the integration of the technologies into the aircraft designs. Advanced air vehicle components and systems that exhibit autonomy and some degree of electrical or mechanical intelligence will offer a solution.

The SNACS project was developed to provide the U.S. Navy with sufficient offensive options while measuring the capability of aircraft to perform effectively in different types of warfighting environments. The need to enhance tactical effectiveness, due to the expanding technological advances from post Cold War threats, has spurred the interest of the Navy to act swiftly in producing the next generation of tactical fighters. The next generation includes components and systems that exhibit a certain degree of electrical or mechanical intelligence in both their operation and in their communication, providing flexibility for future air

vehicle developments. "Smart" components and systems should significantly enhance tactical effectiveness and contribute to the expansion of additional operational attributes such as increased availability, reliability, maintainability, mission time, safety, survivability, and other quantitative and qualitative performance and dependability measures while helping to reduce manpower and other required support. These developments will come from the determination of critical technologies for certain arenas of combat and the integration of these technologies into the platform designs.

A specific technical problem concerning smart components is their integration into future aircraft development and the use of this integration to produce an intelligent and unified Advanced Vehicle Management System (AVMS). This problem involves the physical, logical, and topological interconnects to and from each of these smart components and the processors used to construct them. The main objective of this study is to explore and generate system architecture concepts that will integrate both new and old technology smart components into a cohesive structure to enhance the air vehicle as a system. The incorporation of these smart components will be determined, at first, by developing a simulation testbed to interlock with a graphical user interface and avionics subsystem models. This interface will allow a user to pick and choose as to which components are included in the architecture while monitoring the increase or decrease in factors such as performance and dependability. The performance and dependability factors will be determined dynamically using a simulation testbed of network models and processor models currently being developed at the HCS Research Laboratory. The enhancement of the air vehicle, by treating it as a tightly integrated system, is projected to provide an order of magnitude improvement in effectiveness via dynamic air vehicle system integration. However, better performance is not the only intended result of this project. Several other benefits

¹ Dr. Rosen is with the Naval Air Warfare Center, Aircraft Division, Warminster, PA.

include the commonality among subsystems within the architecture, common fault-tolerant processing, centralized power supplies, and simplified interfaces. These related attributes provide SNACS with an architecture worthy of upgrades to existing military aircraft and, in the long-term, provide a generic base architecture for the design of new aircraft avionics. As with most military research and development, the private sector may also benefit from the SNACS project by the realization of a SNACS architecture for commercial airliners using the same system concepts [1].

Foresight

Foresight is a graphical, general-purpose tool for system modeling and simulation designed to run on UNIX workstations. Designers create models with graphical and textual editors and library elements. The models are analyzed statically for consistency and syntax errors and then executed via the graphical simulator engine. Faster models can be compiled with an optional C-code generator that streamlines models by eliminating unnecessary graphical information.

The methods available in this package have been developed from a succession of related modeling techniques. Combined, these methods provide a versatile means of simulating a wide range of systems at various abstraction levels with minimal design time. History and reference can be found in [2].

Computer-based executability allows designers to verify specifications, functionality, and performance of dynamic systems. Synchronization and resource usage are typical problems that may not be readily apparent from a static model. Executability is achieved by the use of precise definition of functions through primitive elements and specification of the nature of all variables in the data dictionary. Foresight models consist of data flow diagrams (DFDs), state transition diagrams (STDs), mini-specs, library elements, and reusable processes.

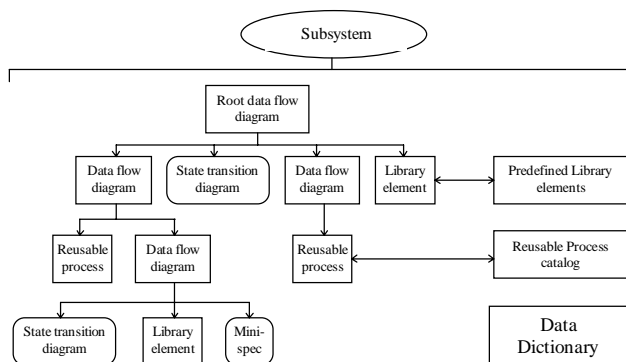


Figure 1. Typical Foresight hierarchy

Effective design of a model in Foresight is accomplished using a hierarchical approach to interconnect the available elements that define the system. At the top of the hierarchy is the root DFD that graphically defines the highest level connections between system modules. The hierarchy branches down to a collection of solely primitive elements. The DFD is not a primitive element and cannot in itself define an executable model. The DFD is required, in general, to represent the interaction of those primitive elements that can provide model executability. The root DFD usually further branches into smaller DFDs as a system increases in complexity.

The first primitive element is the STD which graphically represents the states and transitions of a finite state machine. Actions are not caused by the states themselves but by the transitions connecting them. Transitions paths can be triggered solely by the occurrence of an event (e.g. an input or specified time event) if a single exit transition exists. Logical expressions may supplement these events to choose between multiple transitions exiting a state. The syntax of the operations that occur when performing a transition are quite similar to that used in mini-specs. Mini-specs are primitive elements that are uniquely textual. The Executable Systems Modeling Language used by mini-specs is explained in the users handbook [2] and is quite similar to the C programming language. Examples of an STD and mini-spec implementing the same message generating function are shown in Figures 2 and 3.

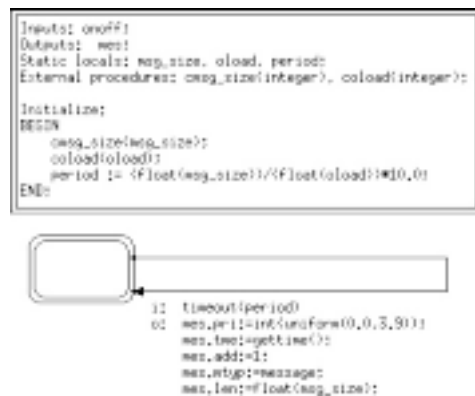


Figure 2. STD message generator example

The key difference between STDs and mini-specs, aside from physical appearance, is in the treatment of more than one data or control flow as an input. Mini-specs use AND firing logic for multiple input flows. All inputs must be available for the element to execute. In STDs, XOR firing logic is used since only one input may be specified as an event for a transition to fire. Before specifying an element, the designer must consider the nature of the firing logic for multiple input flows which

dictates the element that should be used. For elements with single input flows, firing logic is irrelevant and the designer may use the most preferred tool. It is not uncommon to begin a design using one element and later translate it into the other form. The similarity in defining actions is conducive to translation if necessary.

```

Input: none;
Output: msg;
Static locals: msg_size, oload, period;
External procedures: msg_size(integer), coload(integer);

Initialize:
BEGIN
  msg_size(msg_size);
  coload(coload);
END;

Procedures:
BEGIN
  WHILE true LOOP
    period := (float(msg_size))/float(oload);
    msg_ptr := int(ceil(0.5, 9));
    msg_ptr := msg_ptr + 1;
    msg_ptr := msg_ptr % msg_size;
    msg_ptr := msg_ptr + 1;
    msg_ptr := msg_ptr % msg_size;
    writeOutput(msg_ptr, period);
  END LOOP;
END;

```

Figure 3. Mini-spec message generator example

The final primitive element class consists of the library elements already defined by NuThena. Every element is graphical in nature so that they may only be used in DFDs. The library is divided into 5 categories and contains over 100 elements. The categories include: data manipulation, inputs and outputs, math and logic, signal processing, and timing and validation. More complex elements may require the user to specify parameters such as queue size or periodic rate. These elements provide a general coverage of common elements found in a wide range of system modeling.

The reusable processes are user-defined groups which consist of combinations of the previously described elements. The benefit of reusable processes is that they may be used in different locations of the design without the need for matching the names of the input and output ports of the module to the names of the external flows attached to them. In STDs and mini-specs, the specified input and output ports must be identical in name and number to external flows represented in the parent DFD. Reusable processes incorporate the ideas of encapsulation and inheritance found in object-oriented design [2].

Models can be designed to operate at various levels of abstraction ranging from fine-grain emulation to coarse-grain statistical simulation. Library elements support modeling at both extremes and user-defined processes can be focused at the desired level. There are several library elements for manipulating data flows at the functional level as well as elements that easily support resource modeling and semaphores at the statistical level. As with most simulation tools, tradeoffs are a necessity for a model that balances accuracy with simplicity.

Processing power for simulations is a major restriction for designers; as procedures become more detailed, design time and simulation time increase. Consequently, a well-designed Foresight model may perform some critical procedures at a fine-grain functional level and still interact with less critical procedures in the same model that are simulated using statistical information [2].

Existing C programs and functions defining or simulating a system may be tied into Foresight simulations. External procedure calls are made from either STDs or mini-specs. This feature avoids unnecessary design time costs and adds flexibility to model creation. Competent programmers may find that some functions and procedures are more readily designed in a conventional programming language.

The current version of Foresight does not support global variables. That is, variables used in an element are useful only in that element unless explicitly passed to another element (pointers are not supported either). Explicit individual data flow paths result in excessive overhead for the model, thereby unnecessarily increasing model complexity.

Many of the library elements operate on parameters that may be either injected by another process or set manually, but not both. The manually set parameters can only be changed when the model is in the edit mode, therefore these values cannot be retrieved from a previously recorded input file or from an external function often used for interfacing to a GUI. To be used in these situations, the designer must replicate the library elements into user-defined elements that support parameter injection, a time-consuming task at best.

Library elements may only be used in DFDs. Thus, they cannot be modified or expanded. Future versions of Foresight may provide not only the current library elements, but lower-level STD and mini-spec templates representing common modeling algorithms. Doing so would alleviate the limitation mentioned in the preceding paragraph and lend more flexibility to model design [2].

Processor Models

The models being designed by the HCS laboratory are for use by researchers in evaluating prospective systems. Many of the required features and parameters of possible candidates are being analyzed and evaluated. Processor models are currently defined with respect to parameters required to interact and drive the available network architecture models. After common processing variables needed for interaction with the architecture models are identified, they will be used to form a baseline processor model. The baseline model will then be augmented with specific information on various processors to provide a library of processor models.

A prospective list of parameters that will be needed for accurate processor modeling has been generated and is currently under review. The following list provides a guide for features to be incorporated as the models progress: interarrival rate, message length, source/destination groupings, queue sizes, hardware latency, segment length between modules, stream-oriented vs. bursty traffic, ordered priority of modules and messages, offered load, and schedules for specific scenarios.

HSDB Model

High-Speed Data Bus (HSDB) is the generic name for the buses developed for the F-22 and RAH-66 avionics systems by the U.S. Air Force and Army respectively. There are minor differences between these two buses although both are based on the Linear Token Passing Multiplex Data Bus (LTPMDB) standard from SAE [3]. The Foresight tool provides a model called Linear Token-Passing Network (LTPN) which resembles HSDB.

The only quantitative difference found between LTPN and LTPMDB was the number of overhead bits which has since been set in accordance with the LTPMDB standard. Topology and performance analysis changes have been the most significant changes in the crossover from the original LTPN model to the current HSDB model. Processor and network interface modules are now separated so that the functions of each may be clearly defined. Data collection modules for throughput and latency have been altered from those in LTPN so that data is written to formatted output files for spreadsheet analysis. Queue usage tracking is handled by resource modeling which simplifies data collection and analysis. A two-node root DFD is shown in Figure 4.

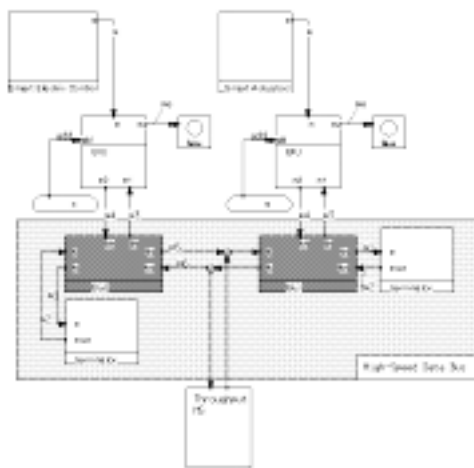


Figure 4. 2-node HSDB model

Messages are constructed in the processor model on a rather low level. A calculation using desired message

lengths and offered load to the full network determines the interarrival rate of messages from the processors. Messages are represented by a dummy string of bits with no informational content. Overhead categories including the destination address, message type, priority, and time-stamp are appended here but are not assigned a specific bit length. The simulated data messages are then sent from the processor module to the Bus Interface Unit (BIU). In the BIU, resource modeling is used to regulate medium access and transmission. Access to the bus is controlled by a processing resource model named "bus". This process resource is assigned a nominal processing ability of 50-Mbps (i.e. standard specification). The procedures in the BIU triggered when messages are to be sent make requests based on the specified message length and pre-defined overhead lengths for a portion of the "bus". The "bus" will grant the use of the resource on a first-come, first-served basis for a time indicated by the number of bits that are to be transmitted. After this time, the resource automatically becomes available for future requests. Abstracting bus activity to this high-level is designed to save time in simulation. Latency and throughput measurement are made at the functional level where individual message transmission can be viewed for verification.

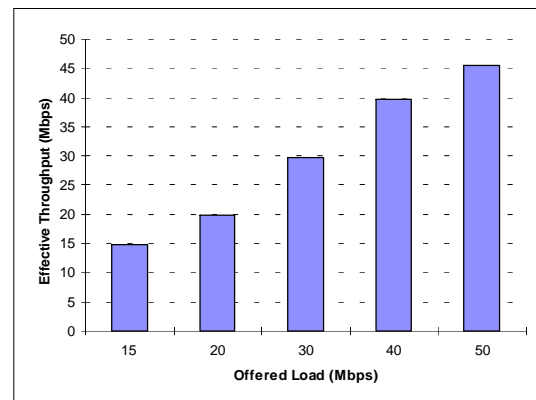


Figure 5. Effective throughput vs. Offered load

A series of experiments are currently being conducted with basic processor models and the HSDB model. Initial results have helped to verify and validate the basic structure of the model. By setting values of base data rate, queue sizes, offered load per station, number of stations, etc., we can estimate the effective throughput and one-way latency of transfers on the network. For example, using the base data rate of 50-Mbps associated with HSDB, a series of two-second simulation runs were executed for a scenario with the following parameters: ten stations; packet size of 32,840 bits based on 32768 bits of data and 72 bits of overhead; 24-bit tokens; four uniformly-distributed priority levels and a 25-slot queue

per level per node; interarrival rate is varied in order to generate the desired offered load; and values for THT, TRT1, TRT2, and TRT3 of 0.5, 0.7, 0.7, and 0.95 milliseconds respectively (see [3] for further information on these token holding and rotation parameters).

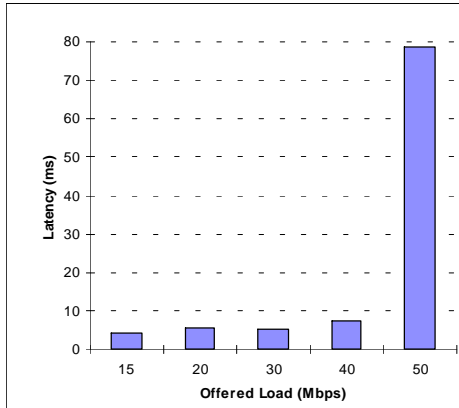


Figure 6. Latency vs. Message size

As can be seen in Figure 5, the effective throughput of the model asymptotically approaches the base data rate but does not reach it due to the bandwidth associated with packet overhead bits, token bits, and token processing time. Since token-passing protocols inherently provide for stable throughput even when offered load meets or exceeds the network maximum, the effective throughput reaches its maximum of approximately 45-Mbps (i.e. 50-Mbps less overhead) when an offered load of 50-Mbps or more is presented. This behavior is expected to continue for simulation runs longer than two seconds. Similarly, the average one-way latency of the model will theoretically approach infinity as the network becomes congested. Figure 6 illustrates this trend for a simulation time of two seconds. An offered load of 50-Mbps will eventually lead to network saturation and exponentially increasing latencies once all of the queues in the nodes have filled.

Dependability Modeling

The state transition diagram in Foresight is naturally suited to handle Markov models, the analytical foundation of many dependability models [5]. Unfortunately, models with more than a dozen states cause the transition conditions to become cumbersome and difficult to follow. Examples of "small" models that STDs can handle well are TMR with a spare, reconfigurable duplication, single standby sparing, etc.

For this research task, dependability modeling will also be attempted at the same level of functionality as the performance model. While techniques and injection locations are being explored in Foresight, investigation of other dependability modeling software is ongoing.

Conclusions and future research

This paper summarizes the on-going research and development efforts to date involving the design, modeling, and simulation of processors, networks, and embedded computer system architectures for smart aircraft components and systems. Based on a coarse-grain, discrete-event, systems modeling and simulation tool, models are currently being developed and tool extensions are being constructed in the HCS lab for the performance analysis of processors and networks in a SNACS system. In addition to HSDB, other interconnects to be modeled with coarse-to-medium granularity will include the MIL-STD-1553 and MIL-STD-1773 serial buses and the ANSI/IEEE Standard 1596-1992 Scalable Coherent Interface. All of these models will be designed to interface with smart aircraft subsystem models under development by the Navy, including models for smart fuel systems, smart stores management systems, smart power systems, smart structures and skins, smart actuators, etc. In addition to performance modeling, methods are also being investigated to support the dependability modeling of these AVMS architectures in order to determine subsystem and system reliability, mission time, survivability, safety, etc. Using these models as building blocks, future research efforts will include the development of baseline and candidate architectures, comparative analysis of candidate architectures, and eventually the study of fault-tolerant, distributed computers for SNACS systems.

Acknowledgements

We gratefully acknowledge the support of Mr. Sig Rafalik at the Naval Air Warfare Center, Aircraft Division and the Office of Naval Research.

References

- [1] Gault, Kenneth, Carlos Bedoya, Chris Miller, and John Mohr, *Advanced Vehicle Management System (AVMS) Architecture Studies*, Flight Dynamics Directorate, Wright-Patterson AFB, OH, 1992.
- [2] Gaiser, Brad, *Foresight User's Guide*, Release 3.00, NuThena Systems, McLean, VA, 1994.
- [3] SAE Committee AS-2, Interconnect Networks, *Linear Token Passing Multiplex Data Bus*, SAE Aerospace Standard AS4074, Warrendale, PA, 1993.
- [4] Hammond, Joseph and Peter O'Reilly, *Performance Analysis of Local Computer Networks*, Addison-Wesley, Reading, Mass., 1986.
- [5] Johnson, Barry, *Design and Analysis of Fault-Tolerant Digital Systems*, Addison-Wesley, Reading, Mass., 1989.