20 Digital Avionics Modeling and Simulation

Jack Strauss
Zycad, Inc.

Terry Venema
Zycad, Inc.

Grant Stumpf
Zycad, Inc.

John Satta
Zycad, Inc.

20.1 Introduction

In order to realize unprecedented but operationally essential levels of avionics system performance, reliability, supportability, and affordability, commercial industry and the military will draw on advanced technologies, methods, and development techniques in virtually every area of aircraft design, development, and fabrication. Federated avionics architectures, integrated avionics architectures, hybrid systems architectures, and special purpose systems such as flight control systems, engine control systems, navigation systems, reconnaissance collection systems, electronic combat systems, weapons delivery systems, and communications systems all share certain characteristics, which are germane to digital systems modeling and simulation. All of these classes of avionics systems are increasing in complexity of function and design, and are making increased use of digital computer resources. Given this, commercial and military designers of new avionics systems and of upgrades to existing systems must understand, incorporate, and make use of state-of-the-art methods, disciplines, and practices of digital systems design. This chapter presents fundamentals, best practices, and examples of digital avionics systems modeling and simulation.

20.2 Underlying Principles

The fundamental principle of modeling and simulation is stated as follows. The results of most mathematical processes are either correct or incorrect. Modeling and simulation has a third possibility. The process can yield results that are correct but irrelevant [Strauss 1994]. With this startling but true realization of the possible results of modeling and simulation, it is important to understand the different perspectives that give rise to the motivation for modeling and simulation, the trade space for the development effort to include the users and systems requirements, and the technical underpinnings of the practice.

20.2.1 Historic Perspective

The past 30 years of aviation has seen extraordinary innovation in all aspects of design and manufacturing technology. Digital computing resources have been employed in all functional areas of avionics, including communication, navigation, flight controls, propulsion control, and all areas of military weapon systems. As analog, mechanical, and electrical systems have been replaced or enhanced with digital electronics, there has been an increased demand for new digital computing techniques and for higher performance digital computing resources.

Special purpose data, signal, and display processors were commonly implemented in the late 1960s and early 1970s [Swangim et al., 1989]. Special purpose devices gave way to programmable data, signal, and display processors in the early to mid 1980s. These devices were programmed at a low level—assembly language programming was common. The late 1980s and early 1990s have seen commercial and military avionics adopting the use of high-performance general-purpose computing devices programmed in high-order languages. The USAF F-22 fighter, for example, incorporates general-purpose, commercially available, microprocessors programmed in Ada. The F-22 has an operational flight program consisting of nearly one million lines of Ada code and onboard computing power on the order of 20 billion operations per second of signal processing and 500 millon intstructions per second (MIPS) of data processing. Additionally, there is increasing use of commercial off-the-shelf (COTS) products such as processor boards, storage devices, graphics displays, and fiber optic communications networks for many military and commercial avionics applications.

COTS product designers and avionics systems developers are making it a standard engineering practice to model commercial computer products and digital avionics products and systems at all levels of design abstraction. As the complexity of electronics design dramatically has increased, so too has modeling and simulation technology in both functional complexity and implementation. Complex computer-aided design (CAD) software can be several hundred thousand lines of code. These software products require advanced engineering workstation computing resources with sophisticated file and storage structures and data management schemes. Workstations capable of 50 MIPS with several-gigabyte disk drives, connected with high-speed local area networks (LANs) are common. Additionally, special purpose hardware environments, used to enhance and accelerate simulation and modeling, have increased in performance and complexity to supercomputing levels. Hardware accelerators are now capable of evaluating over a million events per second and rapid prototype equipment can reach 10^{11} events per second. At this point in history, the complexity and performance of modeling and simulation technology is equal to the digital avionics products they are used to develop.

20.2.2 Economic Perspective

For commercial system designers, a critical product development factor that has significant impact on the economic viability of any given product release is time-to-market. The first product to market generally recoups all of the nonrecurring engineering and development costs and commonly captures as much as half the total market. This is why technologies aimed at decreasing time-to-market are of increased importance to all commercial developers. Analysis is available showing the amount of development time saved as a result of digital system modeling and simulation [Donnelly, 1992]. At a lower level, cost-sensitive design has two significant issues: learning curve and packaging.

The learning curve itself is best described as an increase in productivity over time. For device manufacturing this can be measured by change in yield, the percentage of manufactured devices that survive testing. Whether it is a chip, a board, or a system, given sufficient volume, designs that have twice the yield will generally have half the cost [Hennessy and Patterson, 1990]. The design reuse inherent in digital modeling and simulation directly shortens the learning curve. Packaging at the device, board, or system level has cost implications related to fundamental design correctness and system partitioning. A case study of performance modeling for system partitioning is presented later in this text.

For military systems designers, many of the same issues affecting commercial systems designers apply, especially, as more commercial technologies are being incorporated as implementation components. Additionally, as will be shown below, mission objectives and operational requirements are the correct point of entry for modern, top-down design paradigms. However, once a development effort has been initiated, the relative cost of error discovery (shown notionally in Figure 20.1) is more expensive at each successive level of system completeness. Thus, a low-price solution which does not fully meet system requirements can turn into a high-cost problem later in the development cycle.

20.2.3 Design Perspective

Bell and Newell divided computer hardware into five levels of abstraction [Bell and Newell, 1973]. The levels are processors-memory switches (system), instruction set (algorithm), register transfer, logic, and circuit. Hardware systems at any level can also be described by their behavior, structure, and physical implementation. Walker and Thomas combined these views and proposed a model of design representation and synthesis [Walker and Thomas, 1985]. The levels are defined as the architecture level (system level), algorithmic level, functional block level (register transfer level), logic level, and circuit level. Each of these levels is defined in terms of their behavioral domain, structural domain, and physical domain. Behavioral design describes the behavior or function of the hardware using such notations as behavioral languages, register transfer languages, and logic equations. Structural design specifies functional blocks and components such as registers, gates, flip-flops, and their interconnections. Physical design describes the implementation of a design idea in a hardware platform, such as the floor plan of a circuit board and layout of transistors, standard cells, and macrocells in an integrated circuit (IC) chip.

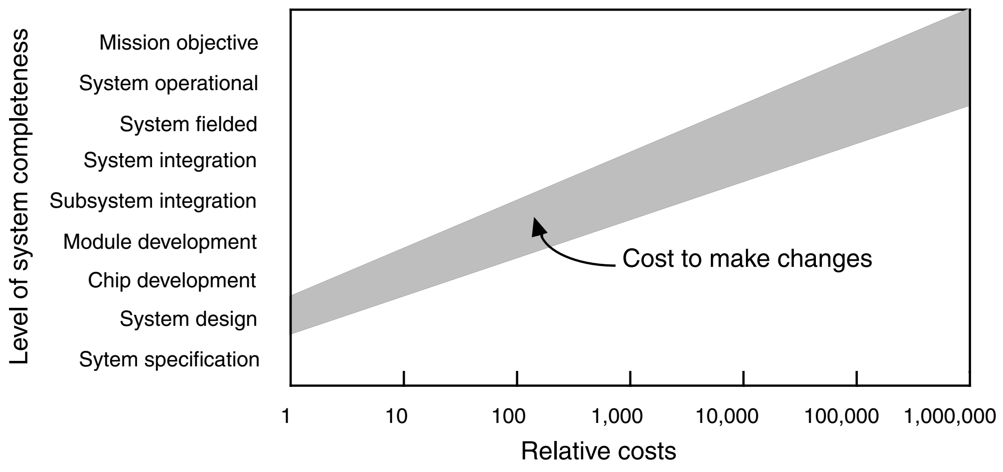

FIGURE 20.1 Relative cost of defect discovery has been shown to increase as a factor of 10 at each successive level of system completeness [Portelli et al., 1989].

TABLE 20.1 Market Factors Comparison for Commercial and Military Market Segments. Each has a different relative priority for the given criterion

Criterion	Commercial	Military
Financial Basis	Market	Budget
Development Focus	Product	Capability
Production Volume	Medium–High	Low
System Complexity	Medium	High
System Design Cycle	Short	Medium–Long
System Life Cycle	Short–Medium	Long–Very Long
Contractual Concerns	Warranty, Liability	Reliability, Mortality

Hierarchical design starts with high-level algorithmic or behavioral design. These high-level designs are converted to circuits in the physical domain. Various CAD tools are available for design entry and conversion. Digital modeling and simulation technologies and tools are directly incorporated into the process to assure correctness and completion of the design at each level and to validate the accuracy of the translation from one level to the next [Liu, 1992].

20.2.4 Market Perspective

It is generally assumed that there are two major market segments for avionics—the commercial avionics market and the military avionics market. As stated earlier, there is great similarity in the technological forces at work on both military and commercial systems. There are, however, several fundamental differences in the product development cycle and business base that are important to consider because they impact the interpretation of the cost performance analysis for modeling and simulation. These differences are summarized in Table 20.1. Consider the impact of commercial versus military production volumes on a capital investment decision for modeling and simulation technology. The relative priority of this criterion is likely to differ for commercial products as compared to military systems.

20.2.5 Requirements in the Trade Space

Technology without application, tactical, or doctrinal context is merely engineering curiosity. The development of an avionics suite, or the implementation of an upgrade to an existing set requires the careful balance of many intricate constraints. More than any other type of development, avionics has the most intricate interrelationships of constraints. The complex set of issues associated with volume, weight, power, cooling, capability, growth, reliability, and cost create the most complex engineering trades of any possible development outside the space program. The risks associated with the introduction of new technologies and the development of enabling technologies create mitigation plans that look like parallel developments. It is little wonder that avionics systems are becoming the most expensive portion of the aircraft development.

To fully exploit the dollars available for avionics development, it is necessary to invest a significant effort in an intimate understanding of the system requirements. Without a knowledge of what the pilot needs to accomplish the mission, and how each portion of the system contributes to the satisfaction of that need, it is impossible to generate appropriate trades and understand the impacts on the engineering process. Indeed, often the mission needs are vague and performance requirements are not specified. This critical feature of the development is further complicated by the fact that pilots often are unaware of detailed technical features of the requirements set, and cannot specifically identify critical system information or presentation requirements.

In the final analysis, the avionics suite is a tool used by the pilot to accomplish a task. The avionics are an extension of his senses and his capabilities. They provide orientation, perception, and function while he is attempting to complete an endlessly variable task. With this in mind, the first and most important step in the design and development of an avionics package is the development of the requirements.

20.2.6 Technical Underpinnings of the Practice

Allen defines the discipline for making predictions of system performance and capacity planning as modeling [Allen, 1994]. He further categorizes techniques in terms of rules of thumb, back-of-the-envelope calculations, statistical forecasting, analytical queuing theory modeling, simulation modeling, and benchmarking. When applied intelligently, all methods have utility. Each has specific cost and schedule impacts. For nontrivial modeling and simulation, the areas of analytical queuing theory modeling, simulation modeling, and benchmarking have the greatest information content, while rules of thumb often hold the most wisdom.

Analytical queuing theory seeks to represent the system algorithmically. The fundamental algorithm set is the M/M/1 queuing system [Klienrock, 1975]. The M/M/1 queuing system is an open system (implying an infinite stream of incoming customers or work) with one server (that which handles customers or does the work) that provides exponentially distributed service. Mathematically, the probability that the provided service will require not more than t time units is given by:

$$P[S \leq t] = 1 - e^{-t/s}$$

where S is the average service time.

For the M/M/1 queuing system, the interarrival time, that is, the time between successive arrivals, also has an exponential distribution. Mathematically, this implies that if τ describes the interarrival time, then:

$$P[\tau \leq t] = 1 - e^{-\lambda t}$$

where λ is the average arrival rate.

Therefore, with the two parameters, the average arrival rate λ and the average service time S, we completely define the M/M/1 model.

Simulation modeling is defined as driving the model of a system with suitable inputs and observing the corresponding outputs [Bratley et al., 1987]. The basic steps include construction of the model, development of the inputs to the model, measurement of the interactions within the model, formation of the statistics of the measured transactions, analysis of the statistics, and validation of the model. Simulation has the advantage of allowing more detailed modeling and analysis than analytical queuing theory, but has the disadvantage of requiring more resources in terms of development effort and computer resources to run.

Benchmarking is the process of running specialized test software on actual hardware. It has the advantage of great fidelity in that the execution time associated with a specific benchmark is not an approximation, but the actual execution time. This process is extremely useful for comparing the results of running the same benchmark on several different machines. The primary disadvantages include requiring actual hardware, which is difficult if the hardware is developmental, and unless your application is the benchmark, it is not likely to represent accurately the workload of your specific system in operation.

20.2.7 Summary Comments

Historically there have been dramatic increases in the complexity and performance of digital avionics systems. This increase in complexity has required many new tools and processes to be developed in support of avionics product design, development, and manufacture.

The commercial and military avionics marketplaces differ in many ways. Decisions made quantitatively must be interpreted in accordance with each marketplace. However, both commercial and military markets have economic forces which have driven the need for shorter development cycles. A shorter development cycle generally stresses the capabilities of design technology. Thus, both markets have similar digital system design process challenges. In fact, current policies mandate increased use of COTS products instead of ruggedized versions, and fewer military standards, generally replacing them with "best practices." As this trend continues, there exists potential for both markets to converge on common solutions for these challenges.

The most general design cycle proceeds from a concept through a design phase to a prototype test and integration phase, ending finally in release to production. The end date (be it a commercial product introduction or a military system deployment) generally does not move to the right on the schedule. Designs often take longer than scheduled due to complexity and coordination issues. The time between prototype release to fabrication and release to production, which should be spent in test and debug, gets squeezed. Ideally, lengthening the test and debug phase without compromising either design time or the production release date is desirable. Modeling and simulation does this by allowing the testing of designs before they are fabricated, when they are easier, faster, and cheaper to change.

The design process for any avionics development must begin with the development of the requirements. A full understanding of the requirements set is necessary because of the inherent constraints it places on the development and the clarity it provides as to the intended use of the system.

There are several techniques available for the analysis and prediction of a system's performance. These techniques include rules of thumb, back-of-the-envelope calculations, statistical forecasting, analytical queuing theory modeling, simulation modeling, and benchmarking. Each technique has advantages and disadvantages with respect to fidelity and cost of resources to perform. When taken together, and applied appropriately, they form the rigor base for the best practices in digital avionics systems modeling and simulation.

20.3 Best Practices

As the fundamental principle of modeling and simulation suggests, a correctly executed simulation of any model may yield results that are correct but irrelevant. Therefore, it is imperative to understand the customers, the systems, the allocation of requirements to system components, the performance of the system components, and to be able to trace component performance to system performance to customer requirements. To date, there exists no single development environment that encompasses this complete set of issues. Several discrete development environments and tools do exist. They are maturing at a rapid, although separate, rate, and tend to either complement or overlap each other. As these environments and tools, as well as the design and economic forces, continue to advance, increased use and decreased cost of entry for avionics systems designers will prevail. Presented below are summaries of best practices in requirements engineering and top-down system simulation.

20.3.1 Requirements Engineering

The process of requirement development for an avionics suite begins with the definition of the mission(s) and related requirements. Once the requirements are known, metrics can be assigned for mission utility of the various systems and subsystems involved in the development. This is essential, because the results of the requirements process are not always consistent with the engineering expectations. That is to say, the modeling and simulation process often uncovers operations and relationships that generate unexpected results. The identification and quantification of these metrics is essential to successfully navigate the complex trade space that is inherent in the development of avionics.

The effective generation of user requirements is the first step in the process. Ordinarily the user provides an initial cut at an unconstrained requirements set. The initial set of requirements needs to be validated through a simulation. Ideally, the user has correctly identified the set of unconstrained requirements that will yield the most effective utilization of the systems. If the current set of requirements yields the maximum capability, the task of the simulation process is to measure the decrease in effectiveness associated with the relaxation of each of the requirements parameters. If the current set of requirements is not optimum, then perturbation of system parameters will yield increases in unconstrained performance. In any event, the relationship between the system parameters and operational effectiveness needs to be well established before beginning any system trade activities.

At the completion of the initial assessment of the requirements, further definition of the requirements trade space requires the definition of mission scenarios for evaluation with relaxed requirement sets. The evaluation of mission performance at this level is associated with changes of requirements in association with each other. Changing the system performance parameters at various levels of requirements provides insight into the best starting point for in-depth study of engineering trades. In many instances, this evaluation is done on the basis of total avionics system costs.

Validating the requirements of avionics design by modeling and simulation depends on the validity of the simulations used. In effect, the simulations need to be calibrated to enable a valid trade study. There are several methods of validating the simulations. Many of the approved operational models are accepted as valid requirements generators, but all models need volumes of approved scenarios, threats, and concepts of operation to be valid.

The design of the avionics suite is intimately related to the design of the cockpit and the displays. In earlier designs, the capabilities of the presentations were often limited by the system elements. The integrating element was the pilot, who received most of the data generated by the avionics. Today, presentation has a significant impact on the performance of the avionics suite. In fact, the presentation requirements for the pilot can be viewed as the requirements for generation of information from the avionics. Determination of presentation necessitates simulation of some kind. This is the integration phase of the requirements with the operator.

Man-In-The-Loop (MITL) simulation is the most reliable means of determining presentation requirements. These can be part-task, or full-mission simulations that drive both presentation and capability requirements. MITL simulation is very expensive, but depending on the implementation, can yield a concise evaluation of the requirements set. There are many disciplines involved in the conduct of human factors engineering, and those references should be consulted. The essential point is that MITL simulation at some level of fidelity is required to fully validate the requirements set.

Sometimes the system requirements can be estimated with a Simulated-Man-In-The-Loop (SMITL) simulation. Working models of human performance exist at several levels of fidelity. Many of these models are adept at working with human-directed control applications, and should be evaluated if the design of the system is attempting a control implementation. The existing models are currently less than satisfying on the cognitive modeling of pilots in flight, however, several attempts are being made to develop cognitive models that will allow a much broader application of SMITL simulation.

In summary, the definition of the requirements determines the effectivity of the avionics suite. Avionics designers continue to shed the constraints imposed by technology over the years. The developments of the year 2000 and beyond will be less constrained by the technology, and the effectiveness will be determined by the application of the concepts for employment. A unique and especially potent display is as effective as a new type of circuit that improves receiver sensitivity. Avionics designs of the 21st century will be based on thorough analysis of requirements, not technological innovations.

20.3.2 Top-Down System Simulation

The Top-Down System Simulation (TDSS) is a proven risk reduction methodology that applies top-down design techniques and currently available simulation technology during development to ensure that complex systems perform correctly the first time. The benefits of applying this methodology range from lower overall cost by eliminating avoidable refabrication of hardware, to on-time schedule performance resulting from the increased visibility into hard-to-foresee integration problems. Together, these benefits frequently outweigh the initial cost of instituting a program-wide TDSS process. Its other far-reaching benefits include the resolution of specification ambiguities, validation of the hardware to specification, and implementation of manufacturing, logistics, and reprocurement documentation that is technology independent.

Until recently, the typical design process began by generating specifications to describe the system, subsystem, component, and interface requirements. Then the system design was partitioned into functions and the functions partitioned into hardware and software. During development, the hardware and software components were developed individually, and then brought together when the system was integrated. In most cases, the final integration step, when discrepancies between concept and implementation were discovered, turned out to be the schedule driver. This was true because issues had to be resolved that were unforeseen, and thus not planned for.

The TDSS methodology is designed to incorporate the best features of the typical development cycle and to add additional visibility and risk reduction to the schedule-driving integration phase. This is accomplished by using simulation techniques that allow a "virtual" integration of component models to be performed while the design is still on the drawing board and easy to change. This virtual integration eliminates the costly hardware refabrication that is frequently required when problems are not discovered until the hardware is in the lab. It also drastically reduces the time spent in the lab integrating the hardware and software, because many of the problems have already been discovered and corrected during development. Examples of this methodology include the U.S. Air Force Advanced Tactical Fighter (ATF) Demonstration and Validation (DEM-VAL) development effort. The interoperability of designs of five critical interfaces were tested through simulation. Over the five interfaces involved, the testing revealed over 200 problems, both with the designs and with the specifications on which they were based. These problems would have resulted in many iterations of hardware fabrication during the integration phase, and several of them would probably not have been detected until the system was fielded. The Air Force concluded that the application of a TDSS methodology to the DEM-VAL program alone resulted in a cost avoidance of approximately $200 million, 25 times the cost of the initiative itself.

20.3.3 TDSS Plan

A formal TDSS plan must be implemented at the very start of any development program with the agreement of all design and development participants. This plan must define the goals of TDSS on the program and the means by which these goals will be reached. For example, a specific goal may be to reduce the risk of subsystem integration through the use of modeling and simulation. The plan will define how this may be attained through the use of commercial logic simulators, off-the-shelf third-party behavioral models, and contractor design databases to perform virtual integration before the costs for hardware fabrication are incurred.

The TDSS plan will define the process, milestones, and data interchange mechanisms that will be required to achieve all of the stated TDSS goals. It will define the tasks to be performed, which Integrated Product Team (IPT) is to perform them, and to which IPT the results will be provided. In a hypothetical example, the Computer Development IPT provides design data and simulation analysis results that prove that all required functions are performed correctly and within the maximum allowable time. These results are passed to the System IPT (at higher level than the Computer Development IPT), who will use these results together with the results of other development IPTs to assess whether all components of the system, when integrated, will meet the functional requirements.

Since modeling and simulation are integral to the development process, they must not be considered as unusual or extra. Accordingly, the milestones defined in the plan are the typical development milestones, such as the Preliminary Design Review (PDR) and the Critical Design Review (CDR) for each component, board, subsystem, etc. This will ensure that the data made available by the TDSS process are used in the most effective possible manner. This means that one development phase will not be officially complete without first meeting all TDSS milestones for that phase. The intent is to prevent taking shortcuts around the process in the "rush to fabrication."

For example, the pre-CDR TDSS phase may require that the hardware be simulated at a predetermined level of abstraction prior to fabrication. The hardware cannot be released for fabrication if the TDSS results are not available or the results do not prove that the functional requirements are being met. If the plan has been agreed to up front by all concerned, then there should not be any surprises or additional effort that might increase the schedule. Even if there is additional effort required, the program will save significant refabrication and rework costs as well as integration costs.

Finally, the data formats to be exchanged between IPTs must be defined and agreed upon by representatives of the customer, the prime contractor, and appropriate subcontractors. For example, the plan might define the contents of the design database that will be supplied to the IPTs that are designing related components. So when information passes between the Computer Development IPT and the Signal Conditioning IPT, each group can effectively make use of the data. The intent is to meet the goals of the TDSS process by eliminating ambiguity and/or extra translation steps wherever possible.

It is imperative to prevent inadvertent divergence from the intent of the design. As the design proceeds from the abstract to the concrete, there are definite points at which the level of abstraction incrementally transitions from a higher level to the next lower level. Before that transition is allowed, the design at each stage of abstraction must be verified and validated.

20.3.4 TDSS Process

The digital electronic hardware development program should proceed hierarchically in the same fashion as the system is arranged. First, the overall system must be defined, modeled, and simulated. Once the system-level model is verified, it can be decomposed into subsystems (functions), each of which must be defined, modeled, simulated, and verified. The subsystems are decomposed into circuit boards containing components of various types (custom, common, standard, and COTS). Each element (subsystem, board, component) at each level of hierarchy will have its own development flow, which will resemble the development flow of the overall system. Figure 20.2 shows a typical development process. At the start of the program the system architects draft the system specification and codify the requirements while the design management team develops the overall top-down development program in the TDSS plan. At the System Requirements Review (SRR) milestone, the team agrees on what needs to be designed to meet the needs of the program. After SRR, the team finalizes the system specification and develops a series of models that describe the performance, behavior, and functions of the system. These models will be the unambiguous reference point for the design since they represent the consensus of the system architects as to what the system should do and how it should be partitioned. The models are formally approved at the System Design Review (SDR) so they can also be used to evaluate the technical soundness of architectures proposed by potential subcontractors in response to a Requests for Proposal.

Once the system is partitioned into subsystems, detailed design work begins. Using the SDR approved models, the team develops Machine Executable Specification (MES) versions. These MESs are distributed to the members of the hardware design team, who use them to construct and verify architectural and performance models of the major subsystems. The subsystem models will be integrated to create performance and architectural models of the entire system. The system architects will review these models and verify their correctness with respect to the requirements. The hardware designers will use the system and subsystem models and the appropriate MESs to begin the preliminary design work. They will define the system at further levels of hierarchical detail, including boards/LRUs, interfaces, modules, and components. They will build gradually to fully functional behavioral models of the system. In order to pass PDR successfully, they must verify that the behavioral models produce the same results as the system performance and architectural models. Once this is accomplished, detailed design work can begin.

The process repeats at the structural level, where the hardware designers are building gate-level models of new components and integrating them with behavioral or structural models of existing or COTS equipment. Once again, the exit criteria for CDR is that these detailed models produce the results that match those produced by the PDR models. Once that is verified, hardware can be fabricated with a high degree of confidence. The hardware must be verified against the models. If there is a discrepancy, the hardware must be made to match the model, not the other way around. The hardware is a technology-specific implementation of the design and, as stated previously, the models are the design.

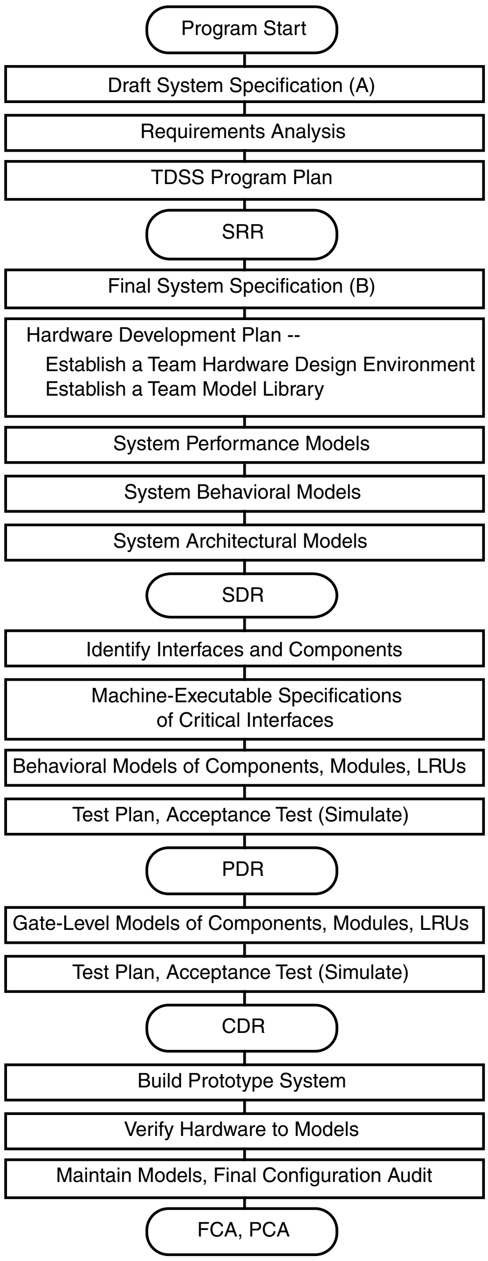
Program Start
Draft System Specification (A)
Requirements Analysis
TDSS Program Plan
SRR
Final System Specification (B)
Hardware Development Plan --
Establish a Team Hardware Design Environment
Establish a Team Model Library
System Performance Models
System Behavioral Models
System Architectural Models
SDR
Identify Interfaces and Components
Machine-Executable Specifications of Critical Interfaces
Behavioral Models of Components, Modules, LRUs
Test Plan, Acceptance Test (Simulate)
PDR
Gate-Level Models of Components, Modules, LRUs
Test Plan, Acceptance Test (Simulate)
CDR
Build Prototype System
Verify Hardware to Models
Maintain Models, Final Configuration Audit
FCA, PCA

FIGURE 20.2 TDSS hardware development flow.

If everything has been done properly, the lowest-level models (and the hardware) will produce results that are traceable back through the design chain all the way to the system specification.

Experience on many programs has shown that the vast majority of system defects can be traced back to misunderstandings due to ambiguity in the specifications. These misunderstandings arise because different hardware designers interpret specifications in completely logical but, unfortunately different, perhaps erroneous, ways. The errors are often not discovered until hardware is actually built and tested. By this time, schedule and budgetary pressures usually preclude leisurely analysis and re-design. So a "workaround" is sought that will overcome the problem, but always at a compromise to system performance.

If a workaround cannot be found, then the program must suffer adverse budget or schedule impacts, often both.

One novel aspect of TDSS eliminates the ambiguous specifications and their associated dangers right from the start. Since a specification is meant to convey the behavior of something, the best way to do that is to provide a readable description that can also be executed on a computer to provide a demonstrable and verifiable example of the behavior being described. A simulation model (written correctly) can be an unambiguous medium in which to embody a specification. A machine-executable specification (MES) is unambiguous because its behavior can be observed under a variety of conditions. The hardware designer does not have to interpret the specification because the behavior can be observed instead. So, if all hardware developers use the same (unambiguous) executable model, they are likely to have the same fundamental understanding of the specification.

20.3.5 Software Modeling in TDSS

A complete system depends on software as well as hardware, and system software design can make or break any system—in performance, schedule, and cost. Good software can bring out the best in mediocre hardware, but poor software can bring excellent hardware to its knees.

Hardware and software are the same at all but the lowest implementation level. Indeed, system specifications and requirements are (theoretically) drawn up with no thought as to hardware/software partitioning. Before software-programmable computers were invented, everything was done in hardware. As software execution speeds increased, more time-critical functions could be handled in software, but from a system standpoint, whether a function is performed by pure hardware or is embedded in code is immaterial as long as the requirements are met and the missions can be achieved. Any complete system design effort must take into account the adequacy or inadequacy of software performance. To verify the adequacy of the "other half" of the system, software performance modeling and analysis should:

1. Account for processing delays on the system or subsystem architecture.
2. Assess the compatibility and portability of software to COTS platforms.
3. Assess the impact of Open Systems Architecture (OSA) requirements on software performance and subsystem interoperability.

Software architecture and performance can be modeled just as hardware can. In both cases, the focus is on input and output rates and amounts, latency (response time), senescence (data age), efficiency, and correct results.

20.4 Performance Modeling for System Partitioning (A Case Study)

The following case study describes a practical application of modeling and simulation used to both articulate and answer system-level questions regarding data latency such that system functional partitioning may be accomplished. Figure 20.3 represents a modern integrated avionics architecture. The system is intended to be fabricated with COTS components and is generalized to include connectivity to existing legacy subsystems.

20.4.1 System Description

This architecture includes the following functional components: Sensor Front Ends representing multiple sensors that receive signals and accomplish analog signal processing; Embedded Computing Assets consisting of computing assets, closely coupled to the sensor front ends, that perform low-latency control processing, digital signal processing, and prepare data for transmission; an ATM Dataflow Network serving as the data transfer medium upon which data are transferred between Embedded Computing Assets, Core Computing Assets, Operator Workstations, and the LAN Gateway; Core Computing Assets

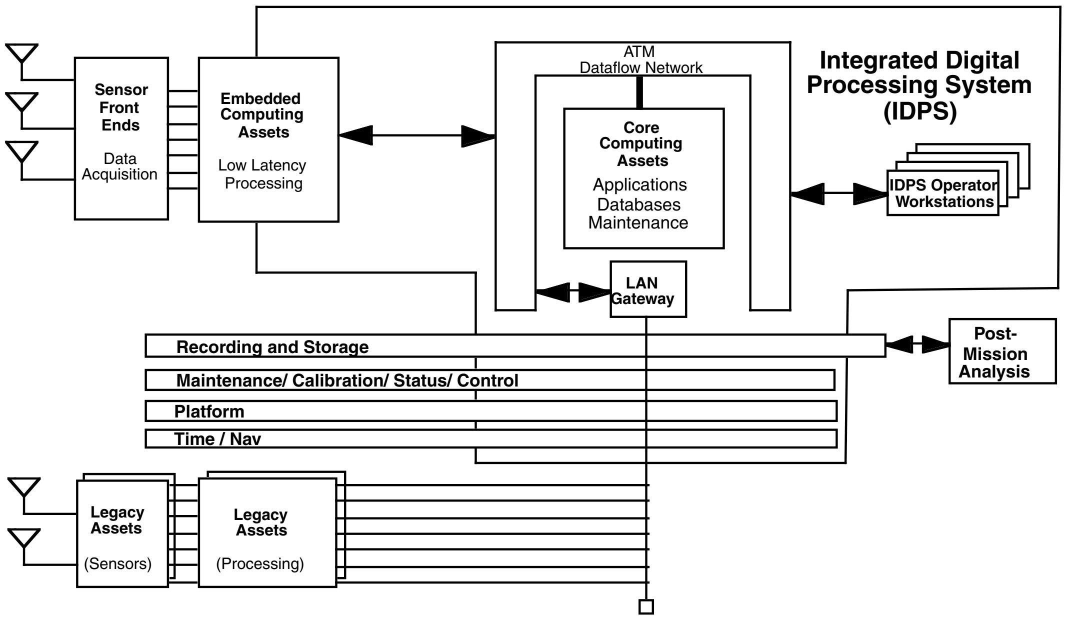

FIGURE 20.3 A generalized integrated avionics architecture.

which include computing and memory devices used to perform application programs and services such as database and maintenance functions; and Operator Workstations servings as the man-machine interface. Additionally, there are recording and storage assets, maintenance, calibration, and control assets, as well as platform-specific and time/navigation assets. Finally, there is the LAN Gateway which serves as the interface to Legacy Systems and networks.

Given the above architecture, the system designer must determine where to allocate software processes that operate on incoming sensor data. The designer must determine which processes are mission critical, which are low-latency processes and which processes are non-low-latency processes that could be allocated to application software. Can the designer allocate all processes to the Core Computing Assets and use the core as a server? What processes, if any, should remain in computing assets local to the incoming sensor data? Is there any incoming sensor data that have to be processed within a time constraint for a successful mission? None of those questions can be answered until a fundamental question is first addressed. Given that a data packet is stored in the RAM of the Embedded Computing Assets, what is the latency associated with that packet's transfer from Embedded Computing Asset RAM, across the ATM Dataflow Network, to Core Computing Asset RAM, and back to Embedded Asset RAM traversing the same path?

The example Embedded and Core Computing Assets each consist of a dual VME 64 backplane configuration, real-time Concurrent Maxion board set (including four 200-MHz R4400 CPU cards [XPU] with 128 MB of RAM per CPU, an I/O card [XIO] conforming to VME 64 standards, and a crosspoint switching architecture capable of 1.2 GB/s peak), and two Fore Systems ATM to VME adapter cards supporting OC-3 155 MB/s SONET fiber (one of these cards is used for redundant purposes only). The Embedded Computing Assets also included MIL-STD-1553, IRIG B, and MXI cards. The Core Assets also include MIL-STD-1553, IRIG B, and SCSI-compatible VME cards. The ATM Dataflow Network (DFN) consists of a Fore Systems ASX-200BX 16-port switch capable of switching up to 1.2 GB/s data across a nonblocking switch fabric. This switch is used to connect the ATM adapter cards using OC-3 155 MB/s SONET fiber. The system is decomposed into the component block diagram shown in Figure 20.4. This is the component block diagram that was used as the basis for a performance model.

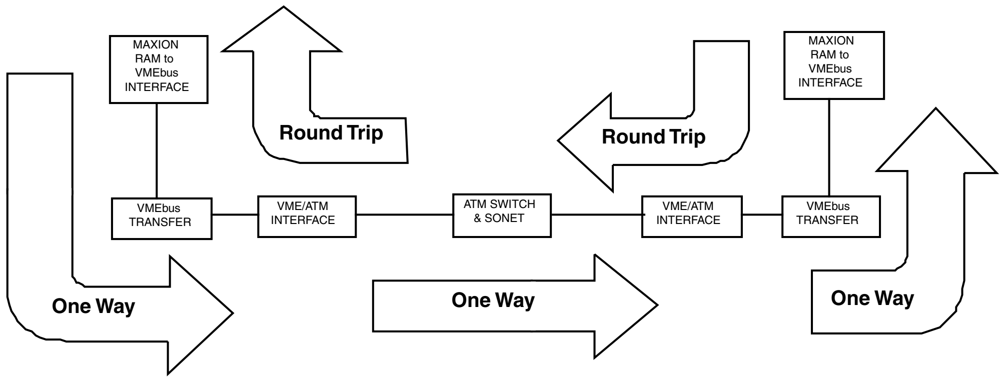

FIGURE 20.4 Component block diagram.

20.4.2 Model Development

Modeling and simulation of the system was accomplished using a commercial computer-aided engineering (CAE) tool (the OPNET modeling tool from Mil-3). This tool is an event-driven simulation tool particularly well suited for computer and network modeling. The tool utilizes hierarchical domains defined as Network, Process, and State domains. A Network domain model for the system was built using the Component Block Diagram shown in Figure 20.4. The Network domain of OPNET allows the specification of nodes and selection and modeling of data transfer pipes (OC-3 SONET, Ethernet, etc.). The Process domain allows the modeler to break down each network node into a configuration of queues, processors, transmitters, receivers, and packet generators. The State domain of the tool allows each processor and queue to be further decomposed into states, with definable transitions between states. Each state of a particular process is defined using C-based programming and simulation kernel calls to simulate the behavior of the process when it arrives at that state. To this end, any protocol behavior can be modeled with the proper combination of states and state behavior definitions. The tool also allows the modeler to probe the simulation at any point in the model to take measurements (throughput, queue capacities, delays, etc.) and analyze the results of these probes while the simulation is running. The system was broken down into modeled components as shown in Figure 20.5 below.

The model for the system was set up to transmit packets (1024 bytes) from RAM located in the Embedded Computing Assets, across the ATM DFN, to the RAM located in the Core Computing Assets, and then back to the RAM of the Embedded Computing Assets via the same path. During the simulation, the computing assets and ATM DFN were loaded at various levels with other data (a known characteristic of the real system was that other data with specific size and frequency rates would also be utilizing the ATM DFN and computer resources at various points throughout the system).

Insertion of these data at various points throughout the model was achieved using the modeling tool's generator modules. This allowed the model to be data loaded at various points in a manner consistent with how the real system would be data loaded during normal operation. For this system, the known system characteristics and protocols explicitly modeled in this example include but are not limited to: I/O read and write delays to/from the Core Computing Assets, CPU wait cycles due to caching operations, crosspoint architecture arbitration, TCP/IP protocol overhead and buffering operations, VMEbus protocol including service interrupt and acknowledgment and delays associated with the various addressing modes of VME operation, ATM segmentation and reassembly delays, ATM call setup delays, ATM fabric switching, and output port buffer delays.

Also included in the model were known data sources that were used to simulate other traffic on the ATM Dataflow Network. A file of System Loading Parameters was also developed made up of a collection of all the modeling parameters available. These include, but were not strictly limited to VMEbus addressing mode, TCP socket buffer thresholds, ATM switching delay parameters, SONET data transport overhead,

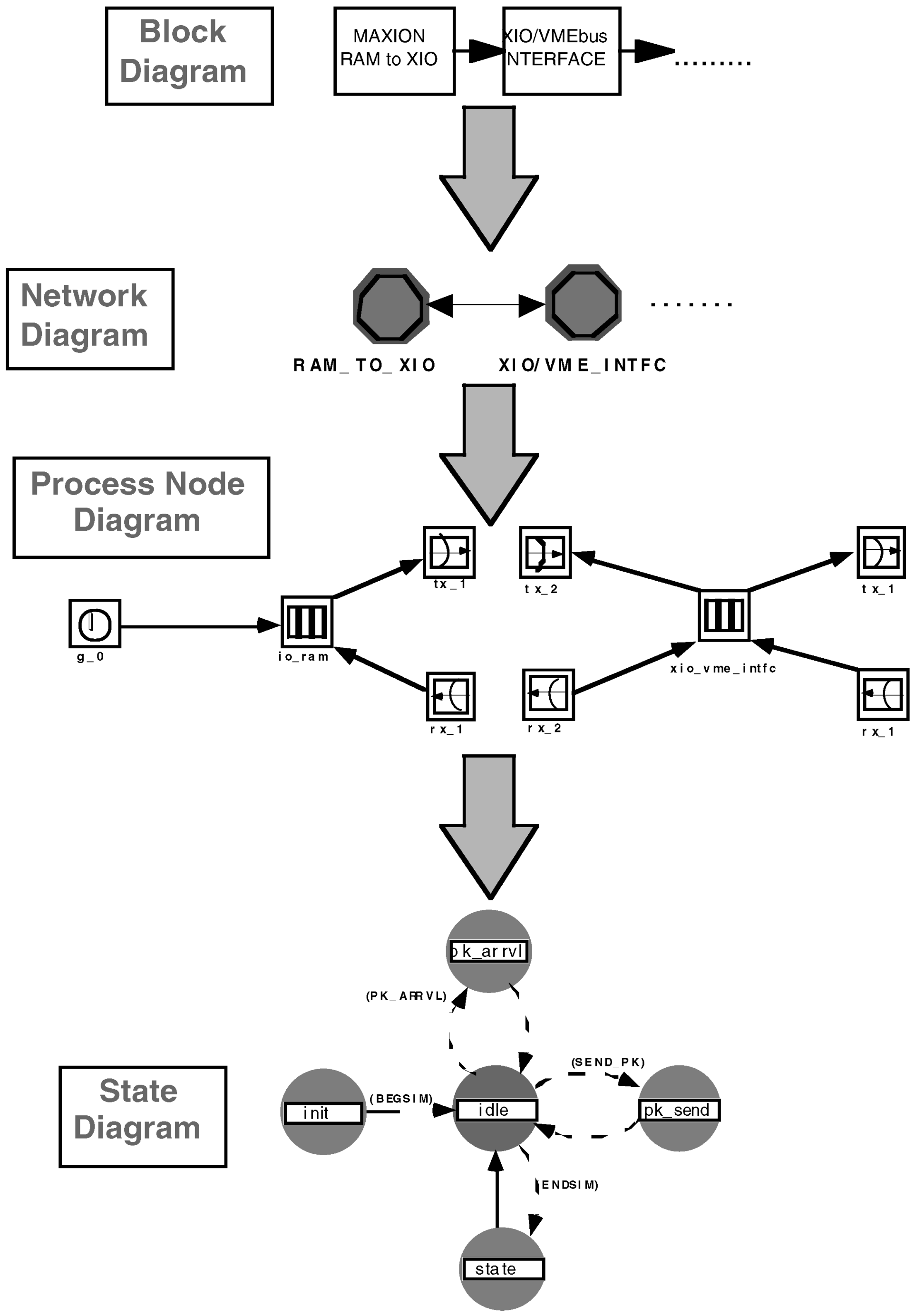

FIGURE 20.5 Modeling decomposition process includes component to network to process to state diagram deveopment.

proprietary communications layer overhead, and many others. The simulation was run 10 times with the System Loading Parameters held constant. The average latency over the 10 simulation runs was plotted as a function of those parameters. The parameters were changed, and another 10 simulations were run at that particular level of system loading. System Loading Parameters were varied from 0.5 to 1.0 in increments of 0.5.

System Load Parameters	RAM to RAM Latency
0.05	0.001444
0.10	0.010342
0.15	0.015953
0.20	0.027908
0.25	0.034309
0.30	0.039111
0.35	0.041234
0.40	0.043421
0.45	0.053901
0.50	0.059921
0.55	0.062344
0.60	0.063495
0.65	0.067401
0.70	0.075105
0.75	0.091120
0.80	0.099101
0.85	0.115230
0.90	0.136987
0.95	0.142335
1.00	0.150428

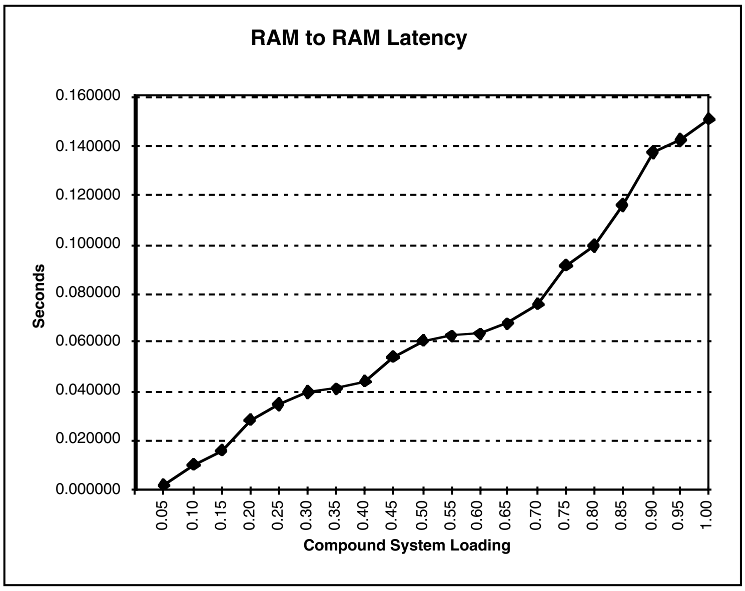

FIGURE 20.6 Case study simulation results.

20.4.3 Modeling Results

The modeling simulation results are shown in Figure 20.6. Each point on the plot represents 10 simulation runs. During each simulation run, the average round trip latency for the target data packets was measured. In a lightly loaded system, according to the simulation results, the target data could be expected to traverse the round trip path presented earlier in an average time of time of 0.001444 s. In a heavily loaded system, again using the simulation results, the data packets experienced a latency of up to 0.150140 s. This was the maximum result just prior to 1.0 loading.

These results were provided to the system designer as performance-based partitioning data and were used to allocate system functions between the Embedded Computing Assets and the Core Computing Assets.

20.4.4 Summary

The actual modeling and simulation effort that formed the basis for this case study was performed as part of a functional allocation process for an airborne system. The results allowed for system partitioning around a 150-ms boundary. By determining the latency early in the design phase, the design could be optimized to make best use of the available processing assets and avoid costly reallocations later in the system development.

This example further demonstrated several advantages to modeling and simulation of a system vs. "vendor data sheet" calculations. First, vendors often times report "best case" or "burst mode" performance characteristics. Modeling and simulation allows parameters of the system to be varied such that "heavily loaded" or "degraded performance" modes of the system may be investigated. Another distinct advantage is that the model is able to generate synchronous data, asynchronous data, and data based on probability distributions. Data latencies in the model are based on modeled event-triggered protocol disciplines rather than deriving answers on a calculator or spreadsheet. Not every aspect of a real system can be modeled completely in every tool, but the known characteristics of the system that relate to performance form a quality model for performance estimation in a tool as was used here.

20.5 Research Issues and Summary

As mentioned above, not every aspect of a real system can be modeled completely in every tool, but the known characteristics of the system that relate to performance form a quality model for performance estimation. Clearly, market forces will continue to drive the quantity, quality, completeness, and rate of change of system engineering environments. As a practical matter, better integration and traceability between requirements automation systems and system/software engineering environments are called for. A powerful result of this integration will be the improved traceability between proposed system concepts, their driving requirements, the resultant technical configuration(s), and their cost and schedule impacts.

The requirement to integrate the trade space across technical, program, financial, and market boundaries is likely to continue and remain incomplete. To paraphrase Brooks, there is more common ground than the technical community will admit, and there are differences that most managers do not understand [Brooks, 1995]. Automation alone will not reduce the complexities of this effort until there is a common, multidisciplined, quantitative definition of cost vs. price in system performance trades. As to the automation issue, the size, content, and format issues of current and legacy technical, financial, programmatic data bases will continue to grow and diverge until the stewards of college curricula produce graduates that solve more problems than they create [Parnas, 1989]. These new practitioners will develop future information systems and engineering environments which encompass the disciplines, language(s), and methods critical to the practice.

Defining Terms

ATM:	Asynchronous Transfer Mode
CAD:	Computer Automated Design
CAE:	Computer-Aided Engineering
CDR:	Critical Design Review
COTS:	Commercial-off-the-shelf (products)
FCA:	Functional Configuration Audit
MES:	Machine Executable Specifications
MIPS:	Millions of instructions per second, or, misleading indicator of processor speed
MITL:	Man-in-the-loop
OC“n”:	Optical Carrier Level n (i.e., OC-3, a SONET specification)
OSA:	Open Systems Architecture
PCA:	Physical Configuration Audit
PDR:	Preliminary Design Review
RAM:	Random Access Memory
SDR:	System Design Review
SMITL:	Simulated-man-in-the-loop
SONET:	Synchronous Optical Network
SRR:	System Requirements Review
TDSS:	Top Down System Simulation

References

Allen, Arnold O., 1994. *Computer Performance Analysis with Mathematica.* Academic Press, New York.

Bell, C. G. and Newell, A., 1973. *Computer Structures: Readings and Examples.* McGraw-Hill, New York.

Bratley, P., Fox, B., and Schrage, L., 1987. *A Guide to Simulation,* 2nd ed., Springer-Verlag, New York.

Brooks J.R. and Fredrick P., 1995. *The Mythical Man Month.* Addison-Wesley, Reading, MA.

Donnelly, C. F., 1992. Evaluating the IOBIDS Specification Using Gate-Level System Simulation, in *Proc. IEEE Natl. Aerosp. Electron. Conf.,* p. 748.

Hennessy, J. L. and Patterson, D. A., 1990. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, San Francisco, CA.

Kleinrock, L., 1975. Theory, *Queueing Systems,* Vol. 1, John Wiley & Sons, New York.

Liu, Hsi-Ho, 1992. Software Issues in Hardware Development, in *Computer Engineering Handbook.* McGraw-Hill, New York.

Parnas, D. L., 1989. Education for Computing Profressionals, in Tech. Rep. 89-247 ISSN-0836-0227.

Portelli, W., Oseth, T., and Strauss, J. L., 1989. Demonstration of Avionics Module Exchangeability via Simulation (DAMES) Program Overview, in *Proc. IEEE Natl. Aerosp. Electron. Conf.,* pp. 660.

Strauss, J. L., 1994. The Third Possibility, in *Modeling and Simulation of Embedded Systems,* Proc. Embedded Computing Inst., pp. 160.

Swangim, J., Strauss, J. L., et al., 1989. Challenges of Tomorrow—The Future of Secure Avioncs, in *Proc. IEEE Natl. Aerosp. Electron. Conf.,* pp. 580.

Walker, R. A. and Thomas, D. E., 1985. A Model of Design Representation and Synthesis, in *Proc. 22nd Design Automation Conf.,* pp. 453–459.

Further Information

Arnold Allen's 1994 text, *Computer Performance Analysis with Mathematica,* Academic Press, New York, is an excellent introduction to computing systems performance modeling.

For Performance Modeling and Capacity Planning the following organizations provide information through periodicals and specialized publications:

- The Computer Measurement Group (CMG); 414 Plaza Drive, Suite 209, Westmont, IL 60559, (708) 655-1812
- Institute for Capacity Management; P.O. Box 82847, Phoenix, AZ 85071, (602) 997-7374
- ACM Sigmetrics; 11 West 42nd Street, New York, NY 10036, (212) 869-7440

For Computer-Aided Engineering the following list of World Wide Web sites provide information on vendors of major tools and environments:

- www.zycad.com
- www.mil-3.com
- www.mentorgraphics.com
- www.synopsis.com
- www.rational.com