Proceedings of the 1983
Winter Simulation Conference
S. Roberts, J. Banks, B. Schmeiser (eds.)

A DEVELOPMENT METHODOLOGY APPLIED TO A RADAR SYSTEM SIMULATION

Jane Harmon, Jay Landreth, Donald Lausch,
Padman Nagenthiram, PhD., and Henry Ramirez

ITT Gilfillan
7821 Orion Avenue
Van Nuys, California 91409

As systems to be simulated grow larger and more complex and the cost of software escalates the need for cost and time efficient development methodologies become critical. This paper describes such a methodology. The methodology is based on standard modern software practices such as top-down design and structured programming. This development methodology has been applied to a radar system simulation which will eventually grow into an air defense simulation employing a surveillance net, and deployed weapons. The objective of the simulation is to provide a computer design tool to aid in the synthesis and parameterization of modern air defense systems.

The simulation embodies computational algorithms as well as the contentions for resources among the various elements of the system.

The radar system simulation is implemented in the SIMSCRIPT simulation program language. It is documented in the Software Design and Documentation Language (SDDL) which is a machine processed and machine reproducible pseudo code.

1. BACKGROUND

Many systems (i.e., air defense systems, computer systems, communication systems, banking systems, etc.), are comprised of a number of asynchronous parallel distributed operations. The total response of such a system is, by nature, complex and probabilistic. Furthermore, the response of each operation varies in relation to the input events. The performance of these systems is usually considered a prime candidate for computer simulation. Conventional simulation approaches do not necessarily provide a means of adequately evaluating the performance of such a complex system. This paper describes the evolution of a discrete simulation for a radar system that quantifies the system performance. The long term project goal is to simulate an air defense system.

A radar system consists of the activities of search, detection, track, and resource management as depicted in Figure 1. The input consists of reflected signals from targets and the environment. Radar outputs are transmitted signals and target data such as position and velocity. Under loaded conditions, all operations are being performed simultaneously, and, in many cases, there is contention for the resources of transmitter power, memory, and bus traffic. In this situation, the operational environment can influence the performance of the system. System performance is based not only upon the errors intrinsic to the computational algorithms, but also upon the resource utilization by the various elements of the system. The total system response is therefore dynamic, that is, varying with the input as the demand for resources changes.

The basic goal of this simulation is to provide a computer design tool to aid radar system synthesis and parameterization. This design tool allows for analysis and prediction of results of radar surveillance (search, detection and tracking) of hostile aircraft employing specific equipment, tactics, and countermeasures in a real world environment where the resources of the radar are controlled within finite constraints.

The radar simulation has been implemented with the SIMSCRIPT simulation program language (Russell 1983). SIMSCRIPT provides a ready-built executive program that allows for the simulation of systems employing concurrent

CH1953-9/83/0000-0409 $01.00 © 1983 IEEE

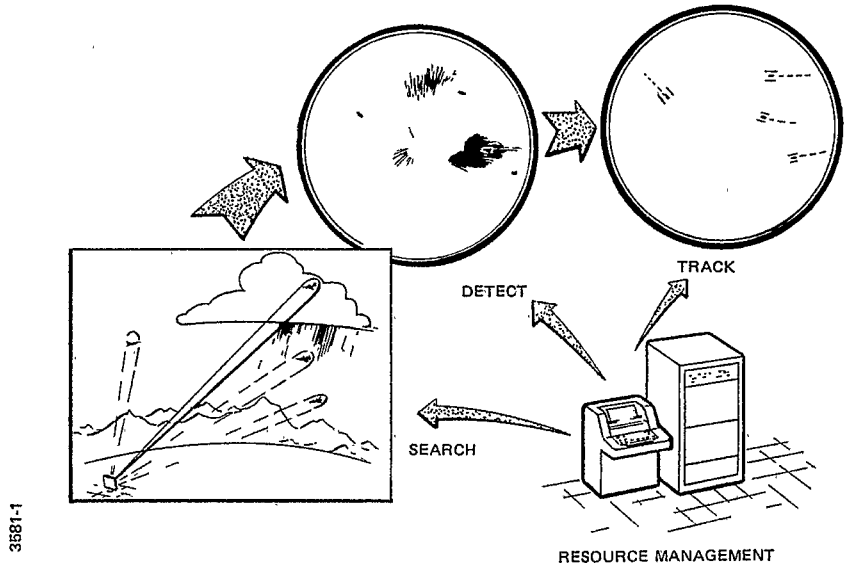

Figure 1: Radar Activities

parallel and serial processing at many levels of processing. The simulation executes on a large general purpose mainframe computer and provides real time emulation by means of a simulated clock and event queues.

Measures of effectiveness for a typical radar system are usually denoted as Measures of Merit (MOMs). A tree structure can be used to represent intradependency of the MOMs or conditional probabilities as depicted in Figure 2. This tree is based on measures relevant to the major processes within the system. In this case the first level is the probability of surveillance which represents the combined effectiveness of the search and tracking processes. The probability of track is the conditional probability of track initiation and track maintenance.

Some of the newer radar concepts dictate an adaptive application of the radar's resources (time, energy and processing capability). Under these conditions, the conditional probabilities are time dependent because the load or threat is dynamic. In many cases the load is inundating, hence, it is necessary to conserve the radar resources. At this point, the measures of merit become conditioned upon energy and time utilization.

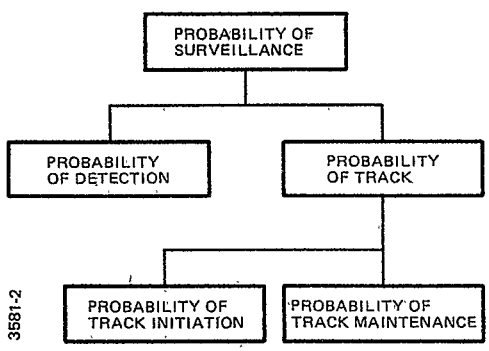

Figure 2: Radar Performance Tree

2. DEVELOPMENT METHODOLOGY

The methodology adopted follows standard modern software practices such as top-down design and structured programming. In addition, some new and unique features have been incorporated. This paper will emphasize these features and demonstrate that they are equally applicable to other similarly large and complex simulation models.

The simulation life cycle for the radar system simulation model is shown in Figure 3. The life cycle begins with a definition phase followed successively by the Software Design, Code and Unit Test, and Integration phase before the simulation finally becomes operational. It is envisaged that subsequent versions of the radar system model, as it grows into an Air Defense Simulation, will also follow this life cycle.

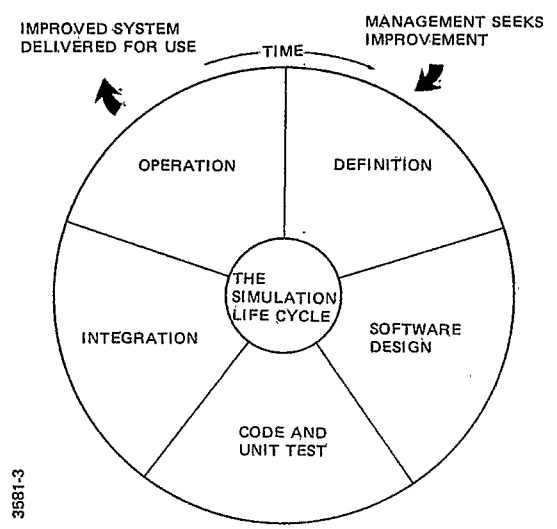

Figure 3: The Simulation Life Cycle

3. DEFINITION

3.1 The Functional Model

The starting point for the simulation is a functional description of the system which states the system requirements. Nominally, the desired inputs, outputs and capabilities are presented to the user group and development group. It is this document that derives the program performance specification.

Detailed specifications are set forth in the program performance specification. These specifications delineate the functions, subfunctions and processes as well as the respective interfaces required to fulfill the functional requirements at the system level. Functional specifications include both logical and mathematical descriptions as well as limitations and constraints.

For the example of a radar system, the functional block diagram and the Scenario driver are depicted in Figure 4. These functions have the following overall requirements.

Scenario: Generate the positional data for targets and environment.

Sensor: Generate signal data for targets and the environment and simulate the radar transmitter, antenna and receiver.

Search: Simulate the activities of initial target detection and position estimation as performed in the radar signal and data processing hardware.

Track: Simulate the activities of target detection and position estimation after the initial contact. In addition, perform the track process in the radar signal and data processing hardware.

Resource Management: Optimize the functions of search and track within the finite constraints of time and energy.

Modification and growth of the model is allowed for by use of the function as a Software Replaceable Unit. It is also possible for these functions to differ in complexity, thus allowing for a combination of levels of simulation detail in the same configuration of the model.

Levels of simulation detail are handled by further subdividing the function into several subfunctional levels. In general, the more detailed the simulation requirements the more subfunctional levels are required. For the radar system model, three functional levels are considered adequate. These are termed function, subfunction and process.

In the example of the radar system a Resource Management functional decomposition is depicted in Figure 5 where the following subfunctional activities are simulated.

Search Allocations: Generate the desired search radar action queue.

Track Allocations: Generate the desired track action queue.

Scheduling: Merge search and track radar action queues into a final schedule radar action queue.

3.2 The Hardware Model Architecture

After each function has been detailed to the level required for simulation, a hardware model architecture is determined. The hardware model architecture is designed to meet the functional requirements of the radar system given a set of available hardware processors using both direct and bus communication. Figure 6 shows an example of a radar system whose functions, subfunctions and processes have been assigned to specific processors, using direct and bus communication, to form a hardware model architecture.

The available processor types for the hardware model architecture consisted of Special Purpose Hardware (SPH), Radar Programmable Module (RPM), General Purpose Machine (GPM), and High Speed

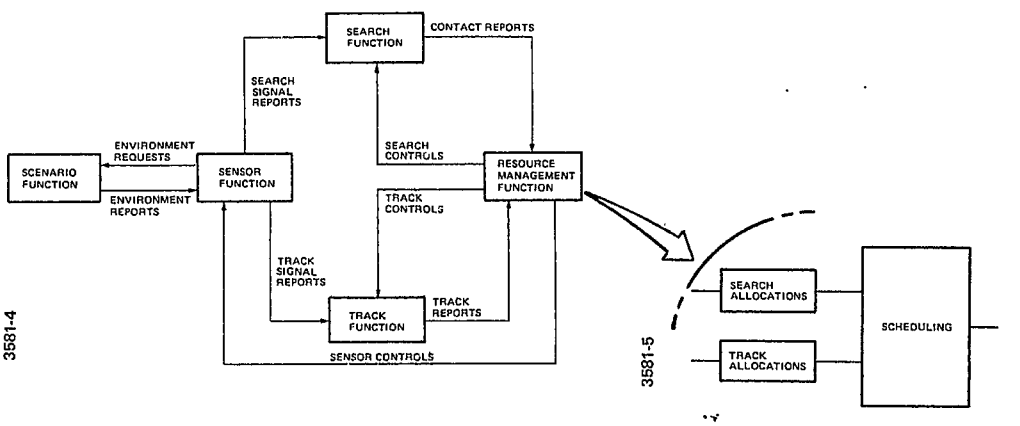

Figure 4: Function Model Of The Radar System And Scenario Driver

Figure 5: Resource Management Decomposition Into Subfunctions

Coprocessor (HSC) units. The Sensor Function was assigned to a single SPH. The Search Function was assigned to a single RPM. The signal processing subfunction of the Track Function was assigned to a single RPM, while the target processing subfunction of the same function was assigned to a GPM. Furthermore, the correlation and association process and the state estimation process of the Track Function were each assigned to HSCs. The resource management search allocation and track allocation subfunctions were assigned to the same GPM while the scheduling subfunction was assigned to another GPM.

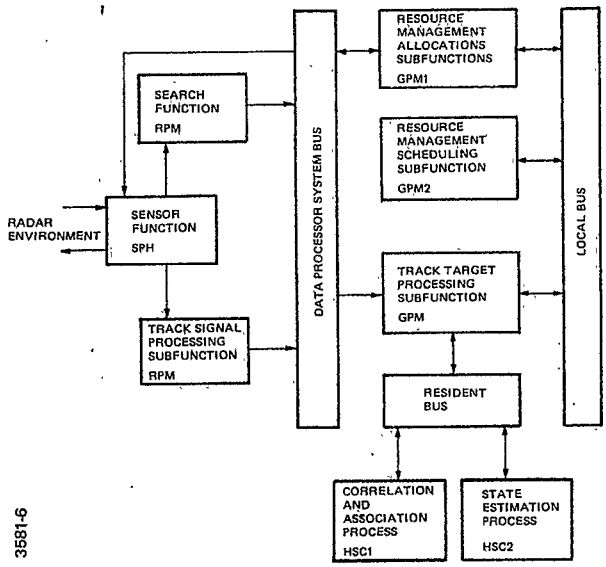

Figure 6: Hardware Model Architecture Of The Radar System

An SPH unit can support multiple data/multiple operation high speed digital and analog requirements. The RPM is a digital bit oriented microprocessor capable of multiple instruction/single data operations per clock cycle. The GPM is a digital word oriented microprocessor capable of single instruction/single data general purpose operations per clock cycle. The HSC is an arithmetic array processing unit capable of high speed computations and is used as a support unit to the GPMs.

The communication architecture chosen for interfacing the radar system processing units, in the example, are direct channel and bus hardware. Direct channels are used between the RPMs and the SPH. The RPMs communicate with the GPMs via a system bus. The GPMs communicate with each other via local bus, and a resident bus is used to interface the support coprocessors to a GPM. Communication architecture between units was determined from the unit interface requirements in which the speed, frequency, and quantity of data pertaining to the transferring of information from one unit to the next was specified. Multiple buses are used in configuring the communication architecture to handle data transfers at different levels of system processing.

A more complex model is not precluded by the specific hardware model shown in Figure 6. For instance, it is possible to add more microprocessors, coprocessors, read-only-memory, and random-access-memory to a microprocessor's resident bus. Just as in the case of the functional model, the modular nature of the processing hardware architecture facilitates the modeling of several different processing hardware configurations.

Configuring the hardware model architecture is an interactive process based upon static throughput analysis. The reconfigurations and corresponding analysis are continued until the hardware architecture is found which is a suitable candidate for simulation analysis. The simulation will be used to dynamically analyze the radar system model not only for overall performance but also for individual processing unit throughput and bus loading. Simulation results may show that additional changes are required to the hardware model ranging from the reassignment of the functional processes to the use of different or additional types of processors and buses.

4. SOFTWARE DESIGN

This phase initiates the construction of a computer model from the conceptual model completed in the definition phase. The design phase is achieved in two stages, top-level design and detailed design. The top-level design stage is mainly a visual definition process in which the high level software structure of the computer model is developed. In the detailed design stage the simulation model is described in pseudo code.

4.1 Top-Level Design

The software design structure is initiated by mapping all the processing units of the hardware model along with their interfaces into a node-message communication architecture. Following this procedure transforms the hardware model shown in Figure 6 to the top-level software structural model shown in Figure 7. Each of the processing units SPH, RPM, GPM, HSC and the Scenario driver is transformed into a master node together with associated processing elements such as subnodes and tasks (not shown in Figure 7). Communication between the processing units is simulated by messages which travel between master nodes.

All master nodes were developed for operation in a similar manner. Basically, a master node must be able to receive messages, determine what action should be taken on that message and initiate that action. If the needed action, on a message, cannot be taken at once, the master node must have the capability to store the message until the necessary action can be taken. A master node handles many different messages. But, all these messages are one of two types; either external, that is from some other master node or internal, that is from one of the master node's subnodes. The action the master node must take in response to receiving a message is to either send it to the appropriate subnode or sent it to another master node. When

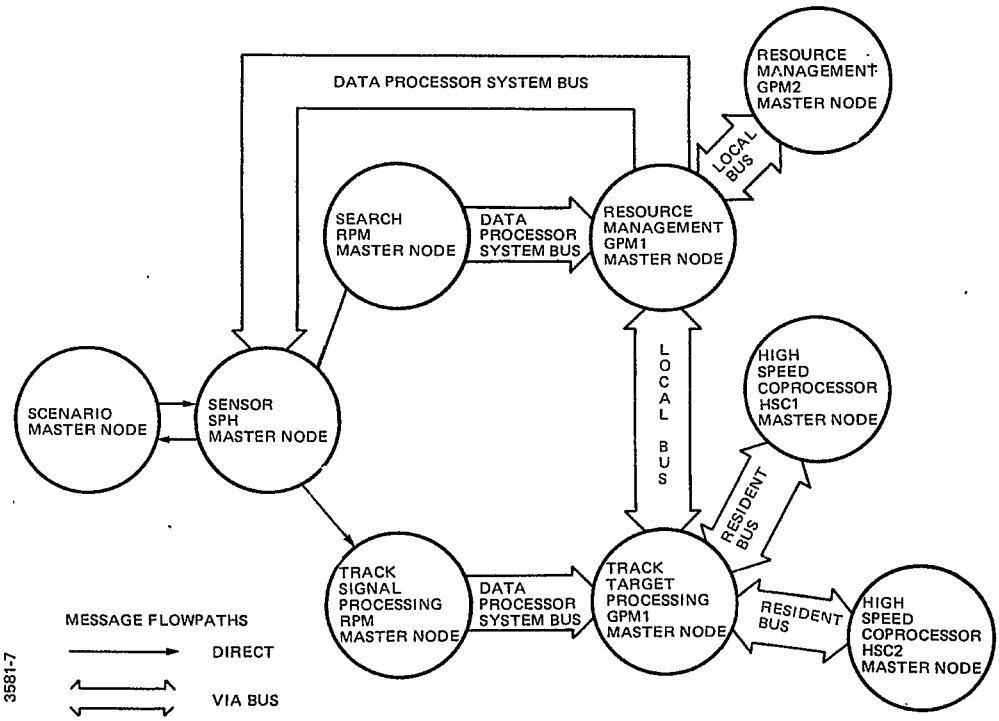

Figure 7: Top-Level Software Structural Model Of The Radar System And Scenario Driver

sending a message to another master node the master node must be able to simulate the use of a shared communication bus when appropriate. Thus, a master node functions as a message handler and a subnode controller.

The actual duties of a processing unit in receiving its input data and performing the necessary processing to produce the required output data is reflected by the flow of data in the messages through the master nodes, supporting subnodes and tasks. The message data is comprised of the actual interface data along with the message type, master node origin, master node destination, and priority characteristics.

In the radar system example, many elements operate at the same time, therefore, numerous messages are active in the model simultaneously. The requirements stipulate that the conceptual radar be capable of multiprocessing and multitasking of the various radar activities occurring in parallel. This translates into a software design structure in which various messages can be simultaneously transmitted and processed by different master nodes, supporting subnodes and tasks in the architecture. In addition, the hardware limitations of bus message transmission, when multiple master nodes are competing for bus availability, further complicate the structure. What is required is a time sequence coordinator embodied in a timing executive to ensure that the timing aspects of modeled architecture are compatible with the radar model. The SIMSCRIPT Programming Language, with its data structure, built-in timing executive and other features, is a suitable candidate to translate these requirements to program code.

The SIMSCRIPT timing executive keeps track of the events occurring throughout the simulation. This timing routine allows for the emulation of real time by use of a simulation time clock. Each initiated process in the simulation has a start time and a finish time which is kept relative to the simulation clock. The timing routine references the event set where all the tasks to be performed are filed in a time and priority ordered sequence.

The timing routine will search until it finds the event with the earliest start time, the event is removed from the event set and the simulated time clock is then advanced. If two or more events are scheduled to occur at the same time, they will be executed one after the other, however, the simulated time clock will not be advanced between the first and last event. Hence, in simulated time the events occur simultaneously.

The SIMSCRIPT data structure is illustrated by examining the SIMSCRIPT structure developed for the Resource Management GPM1 as shown in Figure 8. This structure is similar for all master nodes in the simulation. All master nodes are modeled as SIMSCRIPT processes.

The process can be initiated by other elements of the simulation and its the process initiation and the time it takes to run are both controlled by the SIMSCRIPT timing routine. The subnodes under a master node are also modeled as processes so as to simulate the time taken by them when the tasks are performed.

The Search Allocation and Track Allocation subfunctions assigned to the Resource Management GPM1 consist of a total of four processes. Each process is modeled as a subnode with its associated SIMSCRIPT routine (task).

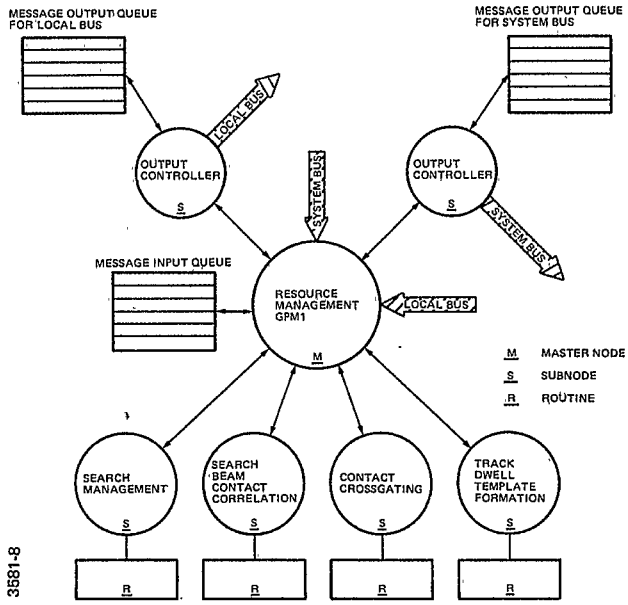

Figure 8: SIMSCRIPT Structure For Resource Management GPM1

Every master node has a subnode for every hardware bus that a master node communicates with. For instance, the Resource Management GPM1 master node has two such subnodes, one to communicate with the local bus and one to communicate with the system bus. These subnodes are termed output controller subnodes and their purpose is to control the output of messages from that master node to other master nodes over the specified bus. Each output controller subnode has a SIMSCRIPT set (queue) to store messages waiting to be output over that bus. Hardware communication buses are modeled by use of the SIMSCRIPT element, resource. In SIMSCRIPT, processes compete for the resources needed to complete their tasks. In the case of a bus, many master nodes may be connected to and use the same bus to send messages to other master nodes. Only one master node can use the bus at a time but more than one master node may need it at that time. When this happens, each master node must wait its turn to use the bus resource and thereby complete its task. The SIMCRIPT timing executive automatically keeps track of which SIMSCRIPT elements are being used and also keeps track of a queue of waiting requests on that resource. As that resource or, in this case, bus becomes free the SIMSCRIPT timing executive begins processing the next waiting request.

Every master node contains an input queue in which messages it must handle are stored. This input queue is modeled using the SIMSCRIPT element, set. In SIMSCRIPT, the set is a linked list. The linkage is done automatically by the system whenever an entity belonging to a set is filed in the set. A member of a set can also be removed from a set, again the linkage modification is performed automatically by the system. When a message is sent to a message node, in SIMSCRIPT terms, it is filed or put into the input set for that master node and that master node is activated. This procedure is the same for external messages (from other master nodes) or internal messages (from subnodes of that master node). When the master node is activated, the master node searches its input set for any, and all, messages upon which action can be taken. Messages for subnodes which are not busy are sent to that subnode and the subnode is flagged as busy.

Once routing of the input message to the proper subnode has been accomplished, the master node directs the subnode to commence processing on the message. The subnode determines which of its tasks are required to respond to the message and allocates the duties among those tasks (in SIMSCRIPT tasks are modelled as routines). In this manner, the tasks perform the computational processing to produce data results for the output message. The subnode subsequently utilizes those results to form a response output message and delegates the responsibility of outputting the message to the master node. The master node determines if the transmission of this output message requires any one of the three bus communications hardware: local, system, or resident bus. If a bus is required, the master node must request this bus, use the bus for the transmission of the message when the resource becomes available and relinquish its use when transmission has been completed. If a bus is not required, the master node outputs the message through a non-bus direct channel to the destination master node. This cyclic procedure of message handling, processing, and outputting is repeated upon the receipt of each input message by any of the master nodes in the radar system and is illustrated in Figure 9.

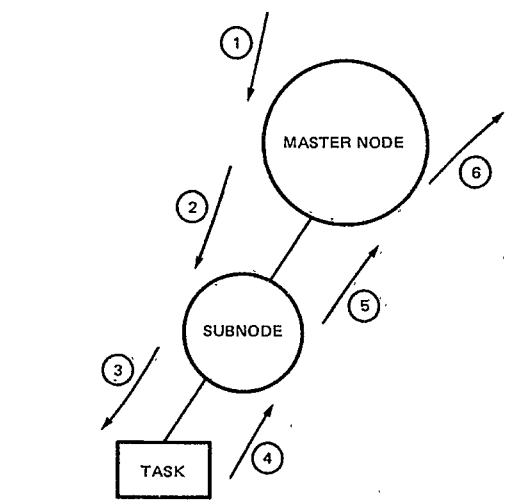

Figure 9: Message Handling And Processing

The capability for expansion and modification of the software structure is enhanced by its modular design. Future expansion can be easily accommodated by the addition or deletion of master nodes, subnodes, tasks, messages and resources. Furthermore, the ability to introduce new algorithms into the radar system could be easily accomplished by replacement of the appropriate master nodes, subnodes, etc., with the corresponding modified master node, subnodes, etc.

This communication architecture structure developed for the radar system model is generally applicable to two categories of simulation. The first category involves a simulation where the functional model is composed of hardware elements which interface through hardware fixtures in a time critical fashion. For example, in a Generalized Computer Communication Network (Russell 1983) the various hardware elements in a typical computer system (i.e., terminals, CPUs, card readers, printers, etc.) were modeled in terms of the master node, subnode, task concept with bus and non-bus communication lines to obtain the throughput of jobs in various system configurations. The second category involves a simulation based upon a non-hardware model or where hardware considerations are of no importance. For example, in the case of the Shipboard Combat System Simulation (SCSS) (Pohoski, Pack, and Mensh 1981) where the various "elements in a combat system" (i.e., weapon systems, C^3, TEWA, etc.) were modeled as purely non-hardware elements in the architecture, the purpose was to gauge the effectiveness of the combat system under various scenario environments.

A typical simulation run of the radar system model encompasses the timing considerations of the first category and the computational algorithms of the second category. However, with easy modification the model could be exercised either as a hardware model with no computational algorithms or as a non-hardware model. When the model is exercised as a non-hardware model, the software architecture becomes a mapping of the functional model shown in Figure 4, rather than the hardware model architecture shown in Figure 6.

4.2 Detailed Design

The use of psuedo code in the detailed design stage serves as a bridge between the simulation analyst, software engineer, and the customer. Additionally, it serves as a design tool for the software engineer. Machine processed and machine reproducible pseudo code (Software Design and Development Language, SDDL) (Kleine 1977) was used because of its desirable aspects of easy updating and modification. In the case of a large and complex simulation with a staff of many software engineers, SDDL can be written by each software engineer for his individual portion and later all the SDDL can be combined to document the entire program.

The SDDL format developed by the software engineers in development of this simulation was of two levels: a functional description portion and a program structure portion. The functional description is primarily used for communication between the software engineers and the non-programming members of the simulation staff. The program structure portion of the SDDL aids the software engineer in development of the program and in the communication between the software engineers about the more detailed aspects of the simulation design.

Each software engineer first completes the functional description portion of the SDDL document for all master nodes, subnodes and tasks under his responsibility. At this point the interfaces between elements of the simulation can be checked and finalized by the software engineering staff members.

The second portion of the SDDL document is the first step in development of the actual program code. The structure of SDDL allows the software engineers to specify and document the structure of the SIMSCRIPT code prior to actual code via the program structure portion of SDDL. This use of SDDL eliminates the need for hand drawn flowcharts which require considerable time to construct and are inconvenient to update and change as a program develops.

When a software engineer is constructing SDDL, certain key words are defined by the SDDL processor to control the processing of this SDDL input file. The SDDL processor will process the input file under the control of these key word statements by providing program structure with various indentation levels. The SDDL processor produces cross-reference listings on items specified. Some items useful for cross-reference in this simulation are global variable names, process names and message names. When all the SDDL for the program has been combined, the cross-reference feature aids in review of the software engineer's design, name, and interface verification.

5. CODE AND UNIT TEST

During the Code and Unit Test Phase, the master nodes, subnodes and routines are coded, compiled and individually tested. With the use of pseudo code there is no clear distinction between the detailed design in SDDL and the beginning of code. The SIMSCRIPT Code naturally evolves as the SDDL gets more detailed. As each function is coded it is tested separately in terms of its subnodes.

At the same time the overall structure of the computer program is built and tested in terms of the master nodes and message flow with program stubs in place of the developing subnodes. Separate software development of the functional algorithms and the system architecture in this manner facilitates development, test and most importantly, integration. Also, the completed overall structural model without the algorithms is useful as a model of the radar system processing hardware.

The Basic Structure of a SIMSCRIPT program consists of three primary elements:

1) A preamble, giving a static predescription of each modeling element (e.g., processes, resources, data structures).

2) A main program where execution begins.

3) A process routine for each process declared in the Preamble.

As each software engineer develops the functional algorithmic code he writes a separate version of the preamble as needed for his sections of code. These separate preambles will later be combined into one final preamble. Each subnode or routine that a software engineer codes is kept separate from the other subnodes and routines so that a change to one will not affect compilation of the others. SIMSCRIPT has the feature that sections of code may be separately compiled so that if a change is made in one section of code it is necessary only to recompile that section using the already compiled preamble. Development time and computer costs are greatly reduced by this feature. When a routine compiles cleanly with the preamble, the software engineer attempts to test it in this isolated subnode condition.

Testing at this phase usually involves writing an input driver to provide appropriate data for the routine under test and writing a print routine to output results. This input driver can be activated to provide data periodically as needed to test the routine. Input drivers are retained for inclusion in the completed simulation. This allows portions of the simulation to be exercised without the need to run the full simulation.

6. INTEGRATION

In this phase the program stubs in the overall structural model are replaced systematically by the subnodes and routines developed separately. The sequence of integration follows the functional flow of activities through the radar system. This methodology of separate development of the overall structural model and the processing modules and subsequent integration in a systematic manner proved to be extremely time efficient.

In addition to the input drivers and print routines developed during the Code and Unit Test phase, testing during integration is enhanced by built-in-test facilities at the subprogram level. These facilities include functional tests which are invoked at different subfunctional levels. Since these functional tests are independent of the algorithms they can be reused during program growth and modification.

The debug features of SIMSCRIPT facilitate the construction of these built-in-test features. For instance, the SNARP.R routine was utilized as a means for outputting message flow information indicating the functional flow of activities in the simulation. This user supplied debug routine is automatically called by the SIMSCRIPT executive whenever a run time execution error is detected allowing the user to examine message flow status (i.e., message data, message origin, message destination, etc.) at error time.

Additional use of SIMSCRIPT debug features included monitored variables and BETWEEN.V language constructs. The monitored variables provision allows the user to specify important radar parameter variables whose values can be automatically output (without the need for numerous print statements) whenever they are used in a computational equation during program execution. As a further debug enhancement, the BETWEEN.V represents a user specified routine which is automatically called at each radar system timed activity. This operation produces a time history of significant radar events.

7. OPERATION

Figure 10 shows the operational configuration for running the simulation program. The user communicates with this program via a CRT terminal through User Friendly Interface (UFI) software. Outputs from a simulation run may be directed to the user's CRT terminal or output to a printer and/or plotter. Graphical output data is currently being transported via tape and is

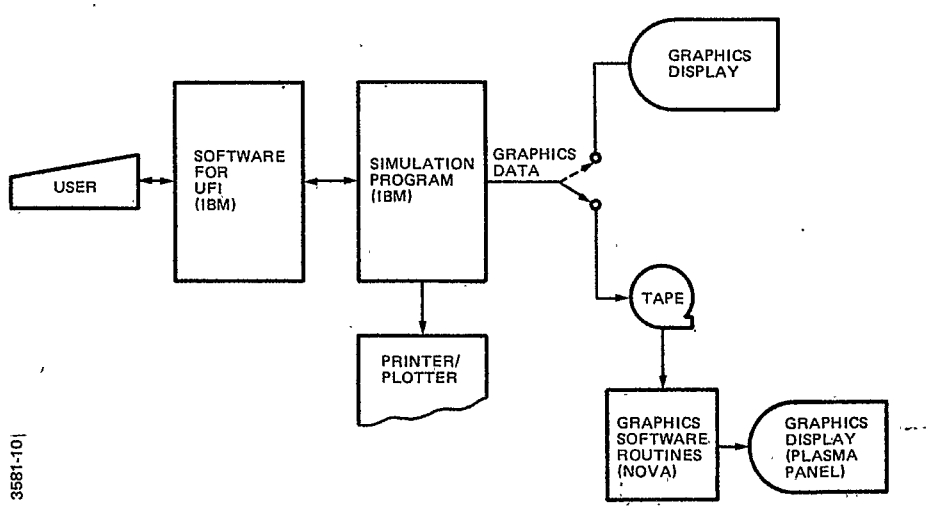

Figure 10: Operational Configuration Of The Simulation Program

displayed on a flat panel plasma display using graphics software routines on a Nova minicomputer. However, as indicated in Figure 10, plans are underway to directly link a graphics display to the IBM mainframe.

The UFI software enables a user who has very little computer knowledge to interact with the simulation program with the minimum of "learning overhead". The UFI employed is a set of tree-structured menus. Via this UFI the user is able to enter/modify input data, select data for output, run the simulation and perform analysis on the output data.

Analysis outputs from the simulation as it is currently implemented will be presented at the conference. There are either statistical summaries which have an alphanumeric format or plots displayed on the graphic display.

8. FUTURE PLANS

Only a limited version of the radar system simulation is currently operational. More detailed versions to aid in the system design of particular radar systems are being currently implemented. The methodology described in this paper allows the model to be very rapidly configured to simulate most modern radar systems.

The radar system model has been developed as the surveillance portion of an Air Defense System. The Air Defense System model envisaged includes the entire area of seabased or landbased theater of operations employing a surveillance net and deployed weapons.

REFERENCES

Kleine H (1977), Software design and documentation language, Technical Report No. 77-24, Jet Propulsion Laboratories, NASA, Pasadena, California

Pohoski MW, Pack DK, Mensh DR (1981), The ship combat system simulation (SCSS), simulation design description and simulation user information, phase 4, Naval Oceans Systems Center, San Diego, California; Naval Weapons Center, China Lake, California; and Naval Surface Weapons Center, White Oak Laboratory, Silver Springs, Maryland.

Russell EC (1983), Building Simulation Models with SIMSCRIPT II.5, CACI, Incorporated-Federal, Los Angeles, California

Russell EC (1983), CACI Network II.5 TM, a Computer and Communications Network Simulator, CACI, Incorporated-Federal, Los Angeles, California