An Application of Simulation to Tracking

Mr. David A. Bennett and Dr. Christopher A. Landauer

Pattern Analysis and Recognition Corporation - Rome, NY

Keywords:

distributed processing
network operating system
simulation/emulation
active radar tracking
microprogramming
performance monitoring

ACM category numbers:

3.89
4.32
6.22

ABSTRACT

The AIMER project (Automatic Integration of Multiple Element Radars) is an emulated model of a loosely coupled distributed radar tracking processor. The computational elements of the model are minicomputers similar to the PDP-11. Design goals of the model are to provide a reliable processing system whose computational bandwidth can be dynamically altered in response to changing ground scenario and availability of hardware. A large number of minicomputers connected with multiple packet networks was chosen as the framework for the design.

Building such a network with real computers would not have provided the required reconfiguration flexibility, and so a hybrid simulation/emulation approach was chosen. The instruction set of the minicomputer family is emulated, which allows performance monitoring to be an integral part of the system. Simulation is the key to controlled experiments and comprehensive throughput analysis. The radar and ground environments are simulated with logic residing in one of the emulated minicomputers. The use of this simulation technique has resulted in an extremely flexible test bed for the development of distributed radar tracking system models. The test bed itself can be quickly tailored to other application problems.

This model is implemented on a Nanodata QM-1 emulation support computer. The QM-1 has two levels of microprogram control store, which allow implementor man hours and efficiency to be traded off conveniently.

1. INTRODUCTION

The AIMER system is an emulation of a loosely coupled network of minicomputers whose purpose is to maintain position information about ground objects under surveillance by several independent radars. Primary input to the system is downlink data from airborne MTI radars, and the primary output is a graphic portrayal of the area under surveillance.

The main thrust of the system as it exists today is to investigate the applicability of emulation in designing and evaluating new systems. In order to accomplish this goal, a system must be designed and then emulated. A radar tracking system was chosen because PAR is currently developing a state-of-the-art radar ground processing system for the Department of Defense, and it was felt that the algorithms involved could be fitted very nicely onto a network of emulated minis. The problems of radar surveillance (predictive filtering, track-hit association, shadowing, de-ghosting, etc.) are well known at PAR; this reduced the number of unknown factors in the emulation of the system.

The current approach at PAR is to use multiple minicomputers (PDP-11's) coupled with special purpose floating point processors as satellites. The number of processes per processor is fairly high (between 20 and 50). Except for the satellite floating point processors, the minicomputers are not dedicated to a specific function.

The AIMER approach is slightly different; there are more minicomputers and each one tends to be much more specialized in function. The fact that the system is being emulated in software and not constructed out of hardware allows its designers

CH1437-3/79/0083-0090$00.75
1979 Winter Simulation Conference

to hypothesize virtually any mix of processors and functions. If indicators point to the efficacy of one or two more processors, they can simply be cranked into the model through a "sysgen." Similarly, if a function becomes a computational bottleneck, it is possible in the emulation to "speed up" the processor on which it resides. It is also possible in the AIMER system to represent a non-homogeneous collection of processor types. This capability is not being exploited at present due to time limitations, but it is expected that future work will explore this area.

2. SUPPORT FACILITY

The support facility being used by AIMER is interesting in two respects. First, the control emulation host is a Nanodata QM-1, a machine uniquely suited to the task of emulation, and second, an unusually large mix of machines, operating systems, and languages are involved.

Nanodata QM-1

The Nanodata QM-1 is a SISD (Single Instruction Single Data Stream) computer which has two levels of microprogram store and a very flexible arrangement of internal data paths and registers. In most other respects the machine is unremarkable, but these two properties make the QM-1 a very powerful emulation support host. It is capable of emulating the CPU and I/O controllers of many small to medium scale computers in such a way that the emulation runs as fast as (or faster than) the original machine. Such emulations allow, for example, a computer center to buy only one piece of hardware and have the option to run it as if it were any one (or several) of a set of machines. Existing emulations include an IBM 360, DEC PDP-11, Data General Nova, and Univac 1106. If the machines to be emulated are microprocessors, the QM-1 can provide a uniform software development and control environment for code which runs on each of the microprocessors. In a research laboratory, the QM-1 provides a testbed for "software first" implementations of new architectures.

The QM-1 achieves its power and flexibility as an emulation host in two ways; hierarchical memory and flexible data paths. The memory system consists of up to 256K x 18 bit words of 800 nanosecond core main store, up to 32K x 18 bit words of 80 nanosecond semiconductor control store, and up to 1K x 360 bits of 80 nanosecond semiconductor "nanostore." The function of nanostore is to provide microcode control of the control store code, in the same way that control store provides microcode control of main store code.

Program Development Environment

The program development world of AIMER consists of three machines; the QM1/Nova (the QM-1 emulating a Data General Nova), HIS6180 Multics and a PDP-11.

The Nova is part of the program development cycle because the AIMER PDQ-11 emulator will run as a process under a manufacturer supported micro level operating system. This is an operating system whose kernel is written in microcode, and whose schedulable entities are machine emulators. That is, at this level a process is constrained to reside in control store and typically defines a machine instruction set at the main store level. These machines can themselves be controlled by operating systems which are unseen by the micro operating system and vice versa. The micro operating system is maintained under an operating system resident on Nanodata's Nova emulator. In order to incorporate the AIMER PDQ-11 emulation into the QM-1 a micro sysgen is required.

The AIMER project is using a real PDP-11 to support cross compilation of PDP-11 code. Algorithms at this level are written in C, compiled by an off-the-shelf C compiler, linked with a project specific run-time support system, and carried by tape in image format to the QM-1.

HIS6180 Multics is providing several services to the project. The PDP-11 emulator microcode is being written and cross assembled under Multics. Limited on-line documentation is being produced on Multics. In addition, Multics will soon be the host for a SMITE cross compiler. SMITE is a procedural machine definition language, and provides an abstract framework in which to define a CPU or an I/O channel controller.

3. THE AIMER MACHINE NETWORK

The application scenario consists of using several independent radars to track multiple ground vehicles. The main data management problem is to relate new information (in the form of hit reports) arriving from the radars to the information (in the form of active tracks and isolated hit reports) collected during earlier analyses (an isolated hit report is one not yet associated with an active track).

The present AIMER machine consists of one (or more) steerable MTI radars (MTI stands for moving target indicator, and implies that the radar does not detect an object if its radial velocity relative to the radar is below a small threshold), and a (fixed) network of processors on the scale of PDP-11s. Our current emulation uses a variation

of PDP-11/45 CPU, without the UNIBUS structure, called a PDQ-11.

In order to facilitate probable structural changes in the AIMER machine design, we have chosen to implement each distinct function as a separate process, and initially, to implement each process as a program running on a separate processor. Moreover, in order to support the intended flexibility of the AIMER machine, and to simplify the implementation process itself, we have chosen to use processors with identical instructions sets (though possibly different speeds).

Our choice of instruction set is based on a simplifying assumption. Since the project staff was familiar with the PDP-11, we have chosen a variant of the PDP-11/45 CPU as our basic processor (we call it the PDQ-11). Any other CPU with floating point instructions and a system trap mechanism should work equally well. The I/O mechanism of the PDP-11, called the UNIBUS, was not modelled at all; we are only interested in the processor operations at this stage.

Another advantage of using a PDP-11 based processor is that we can write all of the algorithms in C under the UNIX operating system. Then the only software needed explicitly on the QM-1 is the "PDQ-11" emulator. The importance of a powerful, familiar program development environment cannot be overemphasized.

We have not analyzed the processor connections beyond specifying who sends how much of what to whom. We have not defined specific processor to processor protocols at the hardware level.

The remainder of this section will summarize the various processes in the AIMER machine. Full details may be found elsewhere (see Reference 3).

In order to separate the characteristics of the radar from the organization of the AIMER machine, the tracking problem can logically be divided into two phases, called the Observer and the Analyzer. The Observer controls all interaction with the radars. The Analyzer performs all computation derived from more than one radar scan, and all interaction with the operators.

The Analyzer sends scan requests to the Observer, indicating particular locations of interest within the target area. The Observer sends hit reports to the Analyzer, indicating approximate locations of moving objects within the target area. The entire AIMER system is driven by a sequence of "surveillance" scan requests, generated internally to the Observer, which are designed to scan through the entire target area periodically. The system is then kept active by returns from the surveillance scan requests.

The three major processing steps to be performed by the Analyzer are association of hit reports to active tracks, updating the active track information using Kalman filter equations, and initiating new active tracks from isolated hit reports.

Track initiation is the process of comparing two hit reports to determine whether or not a new track should be formed from them. The filtering process is designed to (gradually) eliminate measurement and noise errors from the current track information. A Kalman filter is used because (among other things) it was shown to be both effective and efficient for an earlier tracking study (see Reference 1). Track association is the process of comparing a hit report to an active track to determine whether or not to extend the track using the hit report. These computations are by far the most frequent in a dense target environment, since every hit report must be compared to every active track.

Observer

The Observer has two main processor types, called a radar allocator and a radar controller. There is one radar controller for each radar, and one radar allocator. A picture of the simplest version of the Observer may be found in Figure 3-1.

The input from the Analyzer is in the form of scan requests, which arrive at the radar allocator through a queue. The name of the queue is "srra", a mnemonic that stands for "scan requests, waiting for the radar allocator." The radar allocator decides which radar controller should get the scan request.

The radar controller is the interface between the radar itself and the rest of the system. All observation computations are performed by this process, including all coordinate transformations. The results from the radar are translated into hit report lists, and sent to the Analyzer, through a hit report queue. The name of the queue is "hram", which stands for "hit reports, waiting for an association manager."

The most important feature of the Observer is the resolution error problem, which is a direct result of the digital processing performed within the radars. Each radar scan direction is divided (for reporting purposes) in a few hundred range cells that are very thin (about 15 meters deep, but about 100 to 2000 meters in width). The radar response is positive for an object anywhere in the range cell; and the radar does not determine where the vehicle might be in the range cell. The reported position must be the center of the range cell, and the error in this reported position is called the resolution error.

Analyzer

The Analyzer has several kinds of processes,

some of which will be implemented on each of several processors. This replication of hardware is the main source of parallelism in the AIMER system. A picture of the simplest version of the Analyzer may be found in Figure 3-2. The association manager and the track associator may be replicated.

The input to the Analyzer is a sequence of hit reports in a queue from the Observer, and (occasionally) some operator interaction. The main processes are the association manager, the track associator, the track initiator, the track manager, and the operator display. The operator display process works independently of the tracking function, and only effects the tracking function by reading the active track information.

The association manager has the main data management function of the Analyzer. As each hit report is removed from the queue, it is checked against all active tracks for extension feasibility. If an extension is possible, the (track, hit report) pair is sent to the track associator. It is expected that this preliminary pass will reduce the filter computations by a factor near n when there are n tracks.

The track associator performs a Kalman filter computation as part of extending an active track to include a new hit report. This numerical function is the heaviest computational load on the Analyzer.

The track initiator is given isolated hit reports by the association manager, i.e. hit reports that were not found to extend an active track. It checks each isolated hit report against its current isolated hit report list in order to determine possible new tracks that may be formed.

The track manager accepts new active tracks from the track initiator, updated active tracks from the track associator, and responds to control requests from the association manager by returning an active track. It also deletes tracks for which no hit report has been found over a sufficiently long interval.

The operator display is the man-machine interface to the system. It allows the human operator to display active tracks, hit reports, and related items on a display terminal.

4. A DISTRIBUTED OPERATING SYSTEM

This section will describe the operating system that connects the collection of AIMER PDQ-11 machine emulations into a multiprocessor on which tracking algorithms can be tested.

Since the AIMER machine consists of multiple processors, it is necessary to establish distributed operating system controls over the programs on the machine. For the purpose of system control, one of the PDQ-11 processors has been set aside and given special control instructions not available to the other machines. This distinguished processor is called the kernel machine. In addition, each processor (including the kernel) contains a Resident Monitor which serves as the operating system's local enforcement arm. The Monitor is a scaled down operating system in its own right, keeping the state of each processor in line with the policies of the kernel. Thus, there is a split of functionality between the kernel machine, the Resident Monitors, and the other processors. The collective name for this tripartite distributed asynchronous multiprocessor operating system is AMOS. Figure 4-1 illustrates AMOS.

Feature Classes

AMOS provides several classes of features:

- o interprocess communication
- o process scheduling
- o file services
- o debug support
- o accounting
- o protection

Interprocess communication is of fundamental importance to the successful operations of the system. Because of the high volume of data flow throughout AIMER, it must be both reliable and efficient. A process sends a data packet to another process by issuing the "enqueue" instruction. The parameters are checked for consistency by the code which implements the enqueue function, and control is given to the Resident Monitor. At this point the Monitor decides which (if any) of the queue handling processors is required, and sends the data packet to an appropriate destination. It is possible for the queue process associated with the destination to reside on the same processor as the sender, or for there to be no explicit queue associated with the destination process. These considerations are all resolved by the Resident Monitors, with minimal microcode assistance and no kernel interaction.

Process scheduling is of two types: inter- and intra-processor. Intra-processor scheduling is fairly simple, and handled completely by the local Resident Monitor. The process set in a processor is static, and changed only when the whole machine is reconfigured (this is an unimportant restriction, made mainly because the QM-1 lacks file system support). The Monitor allocates CPU bursts (time slices) to each processor. The

system does not attempt currently to adjust burst time in response to dynamically changing loads; this will be considered for future versions. Information from such an algorithm would give a good indication of when a new processor should be added to the configuration or the process/processor mix changed.

Interprocess scheduling is currently also fairly simple. It can be described as event driven demand scheduling of processor time based on data packet flow. Each packet typically represents a demand for computational work to be performed upon the information therein. The notion of packet queueing as a system "primitive" thus obviates the need for explicit intraprocessor scheduling. When a processor becomes idle, it waits for more work requests to appear in its queue, from whatever source.

There are some problems to this approach, however. For instance, it is difficult to predict a maximum size for each queue which is both cost effective under average load and adequate under peak load. Also process priority can be adjusted in two ways, neither of which is very flexible. The first is to reconfigure the system so that more (or fewer) copies of a process are available on independent processors. The second is to upgrade the hardware a processor is running. Both alternatives imply substantial and non-trivial hardware upgrades in an operational system. (The current AIMER system, being a software model and operational system, effects these changes through software parameterization.) The ability for the system to dynamically balance workload with respect to "available" hardware is called for, although not presently implemented.

File services are provided to each process by the kernel machine. Each process has available to it a fixed number of logical files, which are mapped into physical disk, tape, printer, or display files by the kernel. At present, the only operations allowed on files are read logical block and write logical block. Actual reading and writing and device support are handled by the QM-1 micro level operating system. Access to the micro level I/O primitives is only possible from the kernel machine. Formatting, character string conversion, and record blocking are all supported by the Resident Monitors.

Debug support hooks at the emulation level are difficult to discuss because their nature and implementation will change drastically when the present emulated system is moved to an operational configuration. The emulation system was designed with debug visibility in mind, and provides the operator complete control over all aspects of the machine. This includes the ability to stop the clock, take complete snapshots, reproduce transient behavior, profile code execution at the micro level, and set break points which can test arbitrary conditions. Since these capabilities are central to the purpose of the project, the AMOS has been designed to make them both possible and easy. Break point and timing controls are built into the microcode, and the Resident Monitor and kernel have provision for journalling extensive trace and state information on tape. Many of the present debug aids would require special hardware in a non-emulation environment.

Diagnostic facilities at higher levels are written into the C programs for the Resident Monitors, the kernel machines functions, and all AIMER tracking programs. It is possible, therefore, to get traces and dumps of AMOS level information (i.e., kernel service requests, current processor states, and network status), AIMER machine level information (i.e., contents of queues, lists, etc.), and tracking program level information (i.e., locations of tracks and isolated hits).

Accounting is provided by AMOS explicitly for resources of two levels: CPU usage (at the emulation level) and network usage. Microcode accounts for all CPU usage through the mechanism of micro instrumentation. All instructions and memory references account for their own use of these resources in terms of elapsed time. Each processor has an internal elapsed time clock. Instruction self timing is useful in the emulation because each CPU can be parameterized as to its memory cycle time, its computational power, and its floating point instruction speed. Network usage is tallied by the Resident Monitors, and accounting entries summarize the load, source, and destination attributes of each packet. Statistics are also accumulated for the dynamic attributes of all common data structures such as queues.

Protection is provided primarily by physical separation of the processors. This system is not intended to be a general purpose one requiring password and file access protection. The emulation provides memory protection which models physical separation, but allows unrestricted file sharing. Interprocessor memory separation is felt to be a minimal allowable level of protection support. More than this is beyond the scope of the current effort; less would not be adequate during debugging.

Multiple Levels of Abstraction

The project can be viewed at several perspectives, from microcode design up to man/machine interface. Figure 4-2 is a summary of these levels. Higher level numbers represent more abstraction; level n is supported by (i.e., implemented in) level n-1.

Many systems contain multiple levels; AIMER is interesting in that the lowest level extends into microcode and that one of the levels spreads across a network of concurrent processors.

5. PERFORMANCE MEASUREMENT

One of the design objectives of AIMER is to produce a system whose performance can be ascertained quickly and unambiguously. It is important to be able to perturb a component of the machine and verify the exact performance changes which follow from such a perturbation. Measurements must be detailed enough that analytical models can be compared with the empirical behavior of AIMER. The following section discusses considerations important to the fulfillment of this goal.

Since the QM-1 is an SISD machine, processes running on the various emulated PDQ-11's cannot actually be run concurrently. This problem is handled on most mainframes by interleaving the computations for the different processes, using one of several kinds of scheduling algorithms. For the AIMER machine, we need to compute performance statistics based on the assumption that the various processors are actually running at the same time.

For this reason, we have chosen to simulate the concurrency of the different PDQ-11 processors that comprise the AIMER machine. The scheduling of these processors is a different kind of problem from the synchronization and scheduling performed by AMOS. In fact, the description of AMOS can be taken, with slight modifications, to be the programs running on a network of hardware PDQ-11's; the simulation techniques allow us to model such a network on the QM-1.

Origins in GPSS

The simulation technique employed in AIMER derives from GPSS (General Purpose Simulation System). GPSS consists of, among other things, transactions and blocks. A block is a completely analytical model of a computation. A mathematical function is evaluated to determine how long the computation lasts, and the value of the function may be made dependent on data flowing into the block (a transaction). Transactions are started initially in "generators" and can be made to spew out at regular intervals, once, twice, according to any of several time distributions, and so forth. Thus, transactions are spontaneously created by generators, move from block to block as conditions allow, and finally cease to exist when they enter a terminate block. GPSS keeps detailed records about the state of the entire system across its life, including such things as the sizes of queues, the activation times of blocks, and the paths of transactions.

The AIMER machine similarly has processors (blocks), tracks, scan requests, and hit reports (transactions). The data packets are generated and then move from processor to processor, eventually being consumed. Like GPSS, AIMER will maintain a future events chain for the purpose of scheduling computations in processors. Both systems automatically capture data about model behavior in a way which is convenient to reduce and analyze. The primary advance of AIMER is that computation is elaborated explicitly where necessary to provide accurate timing of block execution, and data is passed explicitly from block to block. Because of this, it is possible to construct systems in which components may be either analytical models or empirical realizations or any combination of the two.

The future events chain in AIMER is driven by either an analytical prediction of future completion time or the exact completion time of an algorithm on an emulated processor.

Timing Prediction

Instruction timings are obtained by embedding elapsed time counting in the microcode which implements the processor instruction sets. Thus, there is no necessity for sampling program behavior statistically with an external monitor, or reliance on application level measurement checkpoints. Algorithm timing is done at a level completely transparent to the algorithms themselves, so there is no dependence on the sampling program behavior. Each instruction execution and memory reference knows how long it will take on the processor it is modelling and automatically signals the data collection mechanism when invoked.

The minicomputer processor being modelled is almost identical in instruction set to the DEC PDP-11. Resolution of instruction timing is maintained to 10 nanoseconds, a limit which is felt to be adequate for a family of minicomputers. Processing rates are computed separately for CPU execution, memory reference time, and floating point instruction time. The speed of each of the processors in the model is individually specifiable.

Data Collection

Data is collected and written out to mass storage by the kernel processor. The kernel machine is one of the emulated minicomputers, but it has been given control and data collection primitives above and beyond the other emulated machines. Control is passed to the kernel machine at regular intervals of simulated time for the purpose of data reduction and disposal.

Primitive units of data accumulated by the kernel machine are:

- o Packet transitions through the network including queue and list contents
- o Usage of each processor (CPU time/elapsed time)
- o Dynamic data structure behavior
- o Miscellaneous reports generated by application logic
- o Ground truth of simulated scenario

The ability of the kernel machine to freeze the state of the entire system is invaluable in data collection, debugging, and reproducing complex transient behavior.

Data is collected and reported symbolically by the kernel without requiring any explicit data collection statements in the tracking algorithms. This guarantees that observing the system does not alter its behavior.

Data Reduction

Statistics and summary information about the modelled system are presented to the user in two ways: tabularly and graphically. The extent of tabular output can be selected dynamically, with options ranging from a summary profile at given intervals to a detailed system snapshot at every state change.

Tabular data are of two types: raw and summary. System summaries (profiles) consist of cumulative processor utilization reports, system structure maxima, current structure contents and scheduled future events. Raw data contains additional profile data, reports of detailed behavior of specific processors, and application specific information about algorithm behavior and states.

Graphical data reduction is optionally performed on profile data only. The behavior of most of the profile indicators may be plotted with respect to time. A window of interest may be chosen interactively, and data will be plotted which falls within that window. Any or all of the indicators may be drawn superimposed on the same plot for comparison purposes.

Graphical output is invaluable in quickly spotting trends of activity across the whole model or in identifying throughput bottlenecks.

6. CONCLUSIONS

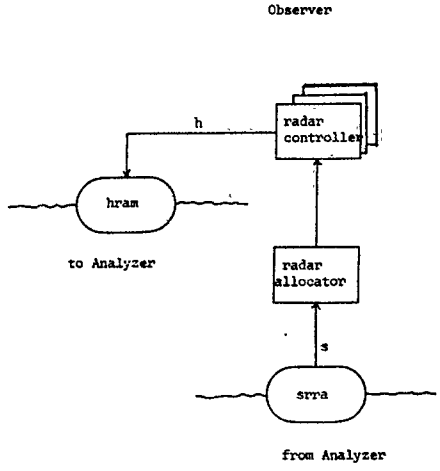
This paper describes the current status of the AIMER machine. The original design of the system has undergone many revisions, based on the identification of bottlenecks in the data flow. The instrumentation facilities described in Section 5 made these identifications very simple.

The current version of AIMER is being used to generate detailed performance characteristics for several configurations of the system. The numerical results will become available in the near future. A particular question of interest relates to the advantages of different kinds of parallelism. It is our feeling that far higher throughput rates can be obtained by using conventional processors organized in a network than by using a vector machine to overlap the filter equation computations.

It has been far easier to begin with a familiar set of basic tools (the PDP-11 minicomputer) and build a network from them than to design the tool expressly for the problem at hand. This result of the AIMER project has its analogy in the way circuit cards are built from some (more or less) standard chips. We needed a moderately powerful processor, with some sort of escape mechanism and floating point computation ability. On the other hand, we didn't want to dedicate a large mainframe to the problem, since they seem to be better at other things anyway. These consider ations led us directly to minicomputers in general, and familiarity led us to the PDP-11 in particular.

REFERENCES

1. "The Multilateration Radar Surveillance Strike System/Atlas I System," Dr. J. Sammon, D. Miedaner, July 8, 1977, (rev. August 1977), PAR Report 77-19.
2. "QM-1 Hardware Level User's Manual," Nanodata Corporation, Rev. 4, March 1976.
3. "AIMER Interim Report," D. Bennet, Dr. C. Landauer, November 24, 1978, PAR Report 78-49.
4. "An Environment for Research in Microprogramming and Emulation," Roser, Robert F., Frieder, Gideon, and Eckhouse, Richard H. Jr., CACM Vol. 15, No. 8, August 1972, p. 748-760.
5. "The Total System Design Methodology," N. Bruce Clark, Summer Computer Simulation Conference, July 1978.



h hit report
s scan request

Figure 3-1

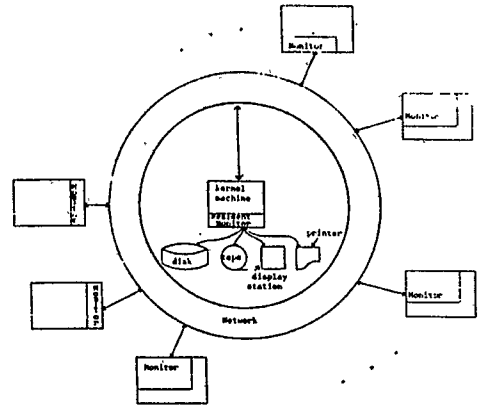
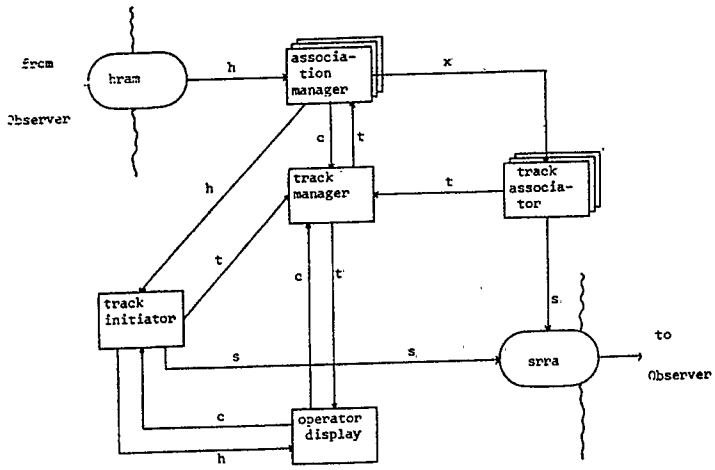


Figure 4-1



c control, commands
h hit report
s scan request
t track
x track extension

Figure 3-2

1. Emulation level — a set of instrumented processors
   - PDQ-11 instructions
   - artificial clock
   - instruction timings
2. Simulation level — a set of μ-programmable machines
   - simulation of concurrency
   - processor switching
   - kernel I/O functions
3. AMOS level — an instrumented network of processes
   - Resident Monitors
   - Network Monitors
   - kernel machine functions
4. AIMER machine level — an information system
   - AIMER tracking programs
   - user machine functions
   - contents of queues, lists, etc.
5. Tracking program level — a tracking system
   - system displays - tracks, hit reports, etc.

Levels of Abstraction Figure 4-2