Software Design of a Proof-of-Concept Advanced Air Traffic Control Display

Eric M. Shank

Massachusetts Institute of Technology Lincoln Laboratory

1. Introduction

While I was a graduate student at the University of Kansas, most of my interactions with Tom Armstrong were through my involvement with the use, day to day operation, and management of computer systems, first the MODCOMP II, and later the VAX 11/750. It was during the time I spent working with these computer systems, and the faculty and students that used them, that I developed much of the expertise in computer systems that is valuable to me in my current work involving design and implementation of software and algorithms for use in air traffic control systems. In light of this, it seems appropriate to me that my contribution to this tribute to TPA should be a high-level description of a software system that I designed for a proof-of-concept implementation of a proposed air traffic control system.

This software system is an integral part of a proof-of-concept radar display. It was designed as a framework in which implementation, testing, demonstration, and refinement of a number of prototype algorithms could be combined with a data collection function. The high level system design is not closely tied to either the particular application or the details of the prototype algorithms. Therefore, I will describe the design of the framework, and ignore the details of the algorithms and the application.

2. Design Goals for the Display System

Since this system was designed to be used in testing, demonstration and refinement of algorithms, and since the software system, with the prototype algorithms in place and operating, was the final product as well as an analysis tool, the design goals were somewhat different than those often applied to software used in experimental environments. The major design goals were as follows:

1) Real Time Operation - The system was designed for demonstration and testing of algorithms for a time-critical air traffic control application. Therefore, it was essential that the system be efficient enough to provide the necessary functions in real time.

2) Data Recording / Diagnostic Capability - A means of recording data for later analysis was required to support both the data collection function, recording aircraft positions, and the testing and refinement of algorithms, recording additional diagnostic information.

3) Flexibility - Because algorithms were expected to be modified or replaced frequently, it was important that the system design allow for extensive modification of algorithms and the addition or removal of algorithms with minimal impact on the overall system.

4) Manageability - The time schedule for implementation was short - less than a year from design to field implementation - so the design was required to allow the implementation effort to be manageably divided among a small team of programmers.

5) Implementation on an Existing Platform - Because of the short time schedule, it was necessary to implement the system on off-the-shelf commercial hardware, using a commercial multitasking operating system.

3. Display System Top Level Design

The system design that emerged from these design goals is highly modular, very flexible, and implementable on most standard multitasking operating systems. It incorporates a number of independent processing modules, implemented as separate processes, each of which carries out a particular function. This modular design made management of the software development team simple, since each programmer could be assigned one or more self contained processing modules to design, code, and test. In addition, because it allows for a well-defined control and communication interface, it provides a high degree of flexibility. If an algorithm is to be changed, typically only a single processing module must be recoded. If an additional function must be added, it can be incorporated by adding another processing module and its associated communication and control ports.

The high level block diagram in Figure 1 shows the use of individual processing modules for separate functions. In the diagram, rectangular boxes represent processing modules, trapezoidal boxes represent communication ports,

and hexagonal boxes represent control ports. This diagram includes the kernel processing modules that carry out the primary functions of the proof-of-concept display system. Additional modules are configured when additional functions are required.

3.1. Design of a Processing Module

The entire system is event driven, with activity being initiated by the flow of data through the system. Each processing module is based on a common design, having one or more data input ports, and one or more data output ports. Input and output ports may be either communication ports (described below) or external devices such as keyboards, graphics displays, serial communication devices and magnetic tape drives.

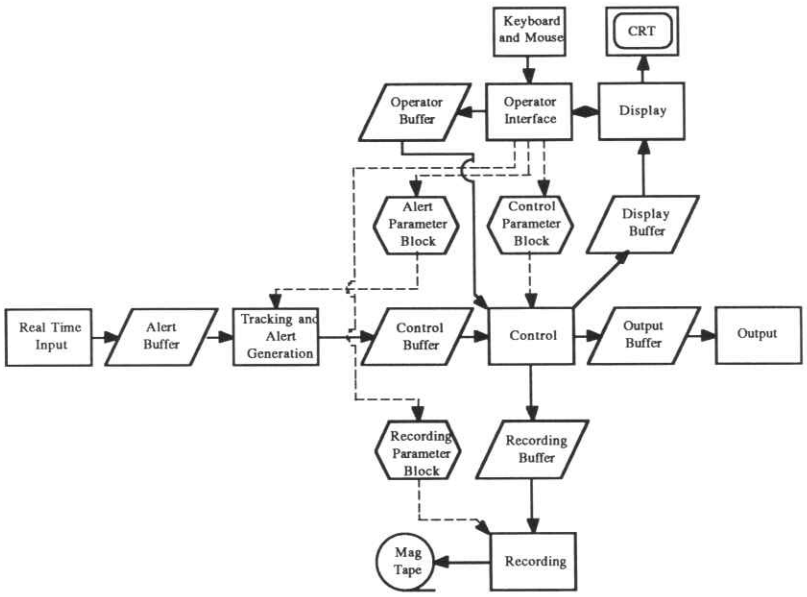

Figure 1. Display system block diagram

The flow of control through each module is similar to that shown in Figure 2. First, the processing module carries out any necessary initialization. When initialization is complete, the processing module waits to be released for normal operation. After it is released, if there is no data available at the input port(s), the processing module goes to sleep. When data becomes available, the processing module wakes up, accepts the data, updates its control parameters if they have been changed, processes the data, and outputs the data. It then checks for additional data at the input port(s). Processing continues until no more data is available, at which point the processing module again goes to sleep. Allocation of processing time between modules is handled by the underlying multitasking operating system. Implementation of this control flow may become fairly complicated, especially when a module has several input and/or output ports. Nevertheless, each processing module is based on this design.

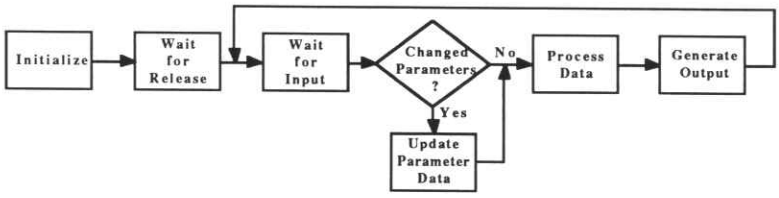

Figure 2. Basic control flow of a processing module

3.2. Communication and Control

Communication between individual processing modules is carried out through communication ports, called buffers, using a message passing protocol. Each buffer contains a linked list message queue. Messages entered into a buffer are queued to the tail of the linked list, and messages are removed from the head of the list. Messages can be entered into a buffer by one or more processing modules, but only one module is allowed to remove messages from a given buffer. Messages are obtained by allocation from a pool of free messages or by removal from a buffer. Once a processing module obtains a message, it owns the message until the message is entered into another buffer or released. Messages entered into a buffer are owned by the buffer. When a message is released, and is not concurrently owned by another processing module or buffer, it is returned to the free pool and made available for future use.

Process control is handled partly through passing control messages between processing modules, and partly through a set of control ports called parameter data blocks. Each parameter data block contains parameters that control the operation of a particular processing module. These parameters can be dynamically modified by the operator through the operator interface.

Access to buffers and parameter data blocks, as well as allocation and release of messages, is carried out by access functions that are common to all processing modules. Any necessary synchronization or mutual exclusion is handled by the access functions. In addition, the buffer access functions generate the events needed to wake a processing module that is asleep while waiting for data to arrive at a buffer.

All messages have a common format that includes a message type field. The type field is used by each processing module to determine whether it needs to process a specific message. Messages that are not processed by a specific module are simply placed unchanged in the output buffer, when the output port is a buffer, and are processed by other modules further downstream. Therefore, when the contents of a specific type of message must be changed, or a new message type added, only those processing modules that process that particular type of message must be modified.

3.3. System Startup and Shutdown

Since the total software system is composed of a set of independent, asynchronously operating, processing modules, each processing module must be started and allowed to complete its initialization before the entire system can begin to function properly. In addition, when the operator interface receives the command to stop the system, it is necessary to see to it that each individual processing module is stopped. These functions are carried out by the executive program.

Flow of control in the executive program is shown in Figure 3. The executive first initializes all data structures, including parameter data blocks and buffers. It then reads a configuration file that specifies which of the possible processing modules will be active during this activation of the display system. The executive program then enters a loop that starts each configured processing module and waits for it to complete its initialization before proceeding to start the next module. When there are no more modules to be started, the executive releases all of the processing modules and puts itself to sleep pending a shutdown request. When

a shutdown request is received, it executes another loop that stops each module in turn. Finally, when all processing modules have been stopped, the executive program exits.

3.4. Display System Operation

In order to understand the operation of the display system, it is useful to consider the flow of data through the processing modules shown in Figure 1. Radar data enters the system at the real-time input module, where it is formatted and copied into a message. The message is entered into the alert buffer. The tracking and alert generation module removes the message from the alert buffer, processes the data, and adds additional information to the message before entering it into the control buffer. The control module acts as a traffic cop, collecting messages from both the operator buffer and the control buffer. It then determines which of the display, output and recording modules should receive each type of message, and enters the messages into the appropriate buffer(s).

The display module removes messages from the display buffer and uses the information in the messages to update the display graphics. The output module reformats the data from messages removed from the output buffer and transfers it to an external communication device, usually to another display system at a remote site. The recording module copies the messages it receives from the recording buffer to magnetic tape.

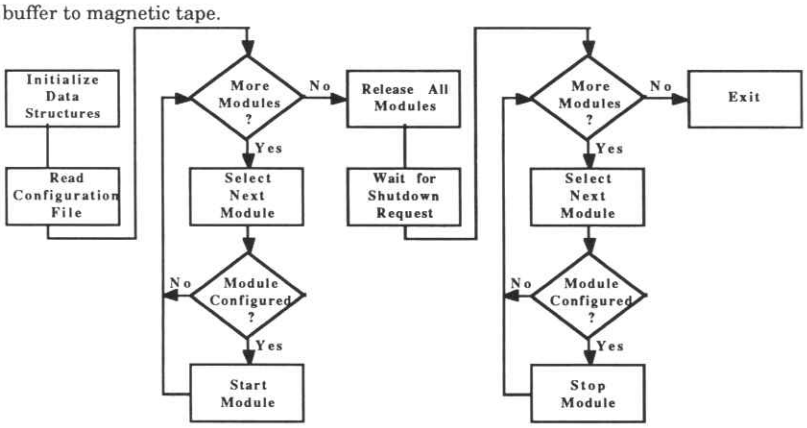

Figure 3. Control flow of the executive program

The operator controls the system from the operator interface. Commands are entered using a mouse or trackball and a keyboard. Since the operator interface module uses the graphic display heavily, it is directly connected to the display module (in the implementation I used, they are coded as part of the same program). Commands are communicated to parameter data blocks or directly to the display module. In addition, each command entered generates an operator message that is entered into the operator buffer.

3.5. Data Recording and Diagnostics

Because of the message passing design of the system, with all messages being passed through the control module, and because all operator inputs generate messages that document the input, the system is capable of recording and regenerating the entire state of the system. This includes any radar data messages received at the input as well as any other messages generated throughout the system. Thus diagnostic and data recording requirements are met.

4. Comments on Realization of Design Goals

Over the past several years, I have been involved in the implementation, testing, operation and maintenance of a display system based on this design. The system was implemented on a VAX workstation platform running the VMS operating system. Implementation from completion of design to first on-site test was carried out in about six months by a team of three programmers. This implementation performs well as a real time display, as a data collection system, and as a platform for analyzing the performance of prototype algorithms.

The flexibility of the design has been demonstrated on several occasions with the addition of functions that were not expected, at design time, to be a part of the system. These additions have required minimal modifications to existing code. In addition, the highly modular nature of the design enhances the overall maintainability of the system.

My experience with this system indicates that the system design has been highly successful. All of the design goals have been realized in the implementation with which I have worked.