CONF-8505137--1

UCRL-92502
PREPRINT

A Distributed Emergency Response System to Model Dispersion and Deposition of Atmospheric Releases

Sandra S. Taylor

This paper was prepared for presentation at the 6th IFAC Workshop on Distributed Computer Control Systems
May 20-22, 1985
Monterey, CA

April 1985

Lawrence Livermore National Laboratory

A DISTRIBUTED EMERGENCY RESPONSE SYSTEM TO MODEL DISPERSION * AND DEPOSITION OF ATMOSPHERIC RELEASES

UCRL--92502

DE85 011847

Sandra S. Taylor

Computations Department, Lawrence Livermore National Laboratory, Livermore, California, U.S.A.

Abstract. Aging hardware and software and increasing commitments by the Departments of Energy and Defense have led us to develop a new, expanded system to replace the existing Atmospheric Release Advisory Capability (ARAC) system. This distributed, computer-based, emergency response system is used by state and federal agencies to assess the environmental health hazards resulting from an accidental release of radioactive material into the atmosphere. Like its predecessor, the expanded system uses local meteorology (e.g., wind speed and wind direction), as well as terrain information, to simulate the transport and dispersion of the airborne material. The system also calculates deposition and dose and displays them graphically over base maps of the local geography for use by on-site authorities.

This paper discusses the limitations of the existing ARAC system. It also discusses the components and functionality of the new system, the technical difficulties encountered and resolved in its design and implementation, and the software methodologies and tools employed in its development.

Keywords

Communications control applications; computer applications; computer control; computer interfaces; control system analysis; data acquisition; emergency preparedness; nuclear plants; system analysis

INTRODUCTION

Background

Federal agencies are charged with operating their nuclear facilities in a manner consistent with the protection of public health and safety. This includes the development of emergency response plans in the event a toxic substance is released accidentally from an operating facility. In 1972, the Department of Energy's predecessor, the Atomic Energy Commission (AEC), realized that its response to nuclear accidents could be improved substantially by developing a capability for real-time estimation of the transport and dispersion of radioactivity released into the atmosphere. They envisioned that such a capability, when integrated with various radiation measurement systems, could help emergency response personnel improve their real-time assessments of the potential consequences of an accident. That vision led to Lawrence Livermore National Laboratory's development of the Atmospheric Release Advisory Capability (ARAC). This capability uses advanced three-dimensional atmospheric transport models to simulate the release of pollutants contained in regional-scale flow systems and to prepare calculations for dissemination to local accident-response officials.

The objective of the ARAC project, as designed in 1973, was to provide real-time predictions of the dose levels and extent of surface contamination from accidental releases of radionuclides from AEC nuclear facilities. Since then, increasing commitments by the Department of Energy (DOE) and the Department of Defense (DOD), combined with aging hardware and software, have provided the impetus for a complete redesign and upgrade of the existing ARAC system. The redesign, which has been underway for the past two years, is being funded by the DOE and the DOD and should meet our current goals by the end of 1986.

Since its inception, the ARAC system has responded to over 125 real-time events and exercises, including the Three Mile Island nuclear power plant accident, the COSMOS satellite reentries, and the TITAN II missile accident. ARAC has also participated in numerous national and international multiagency exercises. Local authorities use the graphically displayed dose and deposition calculations to assess health hazards, formulate evacuation plans, and concentrate measurement and cleanup efforts.

The original computer system, as conceived in 1973, is still in operation, but about 80% of the replacement software system is also operational in a production testing environment. The original system will be replaced in June 1985, at which time the new system will be used by the operations staff. The remainder of the replacement software, as well as additional capabilities, will be completed by 1987. The term *existing system* is used throughout this paper to refer to the system being replaced.

Limitations of the Existing System

The existing ARAC system is distributed on five host processors at the ARAC central facility and can interface with four minicomputer systems located at supported sites across the continental United States. The central facility processors include a Control Data Corporation (CDC) 7600 for executing atmospheric dispersion models and producing graphics products; a Hewlett Packard (HP) 1000 for requesting, receiving, and decoding meteorological data from the Air Force Global Weather Central (AFGWC), which archives meteorological data from around the world; a Digital Equipment Corporation (DEC) 11/34 for preparing input parameters and data for the models; a DEC 11/23 for transmitting inputs to the remote CDC 7600, and a DEC 11/40 for collecting site-tower meterological data, forwarding graphics products to the accident site, handling two-way operator messages with the sites, and archiving meteorological data from both the HP and the sites. Each of the four supported sites has a DEC 11/04 for collecting meteorological data, displaying graphics products produced at the central facility, displaying a simple atmospheric dispersion model using site-tower meteorological data, and handling two-way operator messages with the central facility. When accidents occur at nonsupported sites, a telecopier is used to transmit the graphics products.

Despite its capabilities, the existing ARAC system has several limitations. It depends on the five computer systems and associated peripheral devices being functional for the duration of the event or accident. It is also labor-intensive and requires operators to move data and products from machine to machine. Thus, its advertised response time of one hour can only be met if everything functions correctly. In addition, it requires that operators be familiar with the hardware, as well as the widely differing styles of man-machine interfaces, on all five machines. Because the man-machine interfaces were not designed for occasional users and emergencies and exercises occur only a few times each quarter, operators can forget how to use the system and the initial set of calculations can often take more than one hour.

When the existing system was being developed, modern programming practices were not employed. There were separate program-

*This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

asw

mers for each machine, and software developers tended to design their part of the application somewhat independently, thereby neglecting necessary communication about interfaces. For example, while one programmer would express wind speed in meters per second, another would express it in miles per hour. When the system was eventually integrated, there were gross incompatibilities in the interfaces and considerable retrofitting was needed to make the system operational. Incorrect assumptions were also made as to who should be accountable for which portions of the software, and some user requirements were not met. As a result, the final product, although operational, was disappointing.

To maintain the existing system, programmers had to be familiar with the operating systems and languages used on all the computers. Since parts of the system were developed in machine language with little documentation and no modern software development tools, it was difficult to find and train replacement personnel to maintain the software. Thus, maintenance suffered and the software deteriorated significantly within about five years.

Requirements for the New System

Because of the complexity, limitations, and deteriorated state of the existing ARAC system, we launched a four-year program to replace the system and to upgrade its capabilities, choosing for ourselves several goals and constraints. Our major goal for the new system was to improve the effectiveness of our response, which led to the following specifications for the new system:

- Results and graphics from a preliminary calculation have to be ready for transmission to a site within 15 minutes after initial notification.
- More sophisticated calculations have to be ready within the first hour and every hour thereafter until the release is over.
- The system must be able to handle up to three emergencies simultaneously.
- The system must support a 24-hour/day staff of highly trained emergency response personnel.
- The system must support up to 100 remote site systems.
- The system must respond rapidly to accidents at arbitrary locations where no site computer system is available.

Remote site support would take the form of a small computer system that would enable sites to communicate with the ARAC central facility, display products generated at the central facility, manage local meteorological data, and continuously and automatically calculate and graphically display a simple atmospheric dispersion model result.

To ensure that the new system could recover and respond despite hardware/software failures, we needed backups in hardware, software, sources of meteorological data, and product delivery systems. In addition, we wanted the new system to be easier for operators and meteorologists at the central facility and novices at the site facility to use and understand. Modern programming practices would be used to develop a more modular system that had clearer interfaces to facilitate integration and that could simplify the addition of enhancements and reduce the cost of maintenance. Since we wanted to have effective project management, this goal was to include a workable implementation strategy, good estimates of the time required to develop software, and automated tools to aid in the scheduling of software development tasks and monitoring progress.

SYSTEM CONFIGURATION AND FUNCTIONALITY

The basic system configuration at the central facility consists of two Digital Equipment Corporation (DEC) VAX 11/782 computers, using three DEC LSI 11/23s as communications front-end processors. DEC PC350 computers and meteorological towers with Handar microprocessors are used as a site system at supported facilities. For more detailed information on the hardware configuration, see Appendix A.)

The site system computers can send accident information to ARAC, communicate with ARAC meteorologists, archive and send meteorological observations to the central facility, make simple model calculations, and display results graphically. In addition, they can receive sophisticated model calculations in the form of contour plots from the central facility and display these calculations graphically over locally stored base maps.

Exercises or emergencies are initiated by filling out an on-line accident questionnaire (Fig. 1). This questionnaire can be filled out by local authorities using the site system or by a meteorologist on the ARAC central system, who may receive the information by telephone. Responses to the questionnaire are sent immediately to the central system. As soon as the central system receives notification of an alert, a microprocessor-based emergency page/alarm system with voice synthesizer alerts ARAC personnel, and automated collection of meteorological data is initiated for stations near the release. This automated data collection will continue, on a scheduled basis, for the duration of the exercise/emergency. When enough meteorological data have been collected, a preliminary dispersion calculation is performed automatically. All of these actions can occur without human intervention before the questionnaire is completed.

Experience has shown that more than half of all requests for ARAC assessments come from nonsupported sites. In such cases, local authorities can phone ARAC for accident information. A Xerox 495l telecopier, attached to one of the central facility's two VAX 11/782 computers, will allow ARAC to send VAX-produced graphics products and textual documents directly from the VAX to the accident site.

Most of the meteorological data for ARAC are requested from the AFGWC at Offutt Air Force Base, Nebraska, and from Carswell Air Force Base, Texas. Within two minutes, we can request, receive, and decode current meteorological observations for anywhere in the world. Data that are not current, but are less than 24 hours old, can be retrieved in 5 to 10 minutes. When the link with AFGWC is down, we can request data automatically from a local organization known as WeatherNet. The system also requests data from remote ARAC meteorological towers that are accessible from either ARAC central or the ARAC site system. Supplementary data from local (sometimes portable) instruments are also used. These supplementary data are entered through either the site or central ARAC system.

ARAC uses many simulation models, but the two primary models are MATHEW and ADPIC. MATHEW uses surface, tower, and upper-air wind data to develop an initial, three-dimensional, mass-consistent gridded wind field that includes the effects of topography. Using this wind field, the ADPIC code, a three-dimensional particle-in-cell transport and diffusion code, calculates the time-dependent dispersion and deposition of inert or radioactive pollutants. Inputs to these codes consist of questionnaire information, station location, meteorological data, local topography, and site- and problem-dependent parameters. The products of these models are in the form of numerical results, as well as device-independent graphics. The model-prepared graphics for use by local officials consist of plots showing the contours of varying pollutant concentrations overlaying the local geography. However, other graphics can be generated to aid meteorologists at the ARAC central facility.

A digital terrain database from the United States Geographical Survey (USGS) supports the topography portion of the system at the central facility. In three minutes we can extract (for model input) the local topography, at 500-m resolution, for any area in the continental United States. If we have no terrain data, we can either assume flat terrain or manually enter the terrain elevations. USGS digital line-graph data are also available to support the geography portion of the system. These data are divided into separate overlays for water bodies, rivers, streams, roads, railroads, political boundaries, etc. Currently, we are in the process of developing a system to extract local geography for anywhere in the continental United States. In the meantime, we are using a digitizing system to generate base maps for accident sites. Local authorities need this information to orient the contour plots with respect to local landmarks.

TECHNICAL CONSIDERATIONS

Communications Considerations

A good electronic communications system is of primary importance in a distributed application like ARAC, whose hardware components are physically located around the world. Early in the project we determined that the VAX hardware could not handle our remote communications problems. One problem is that the AFGWC transmits requested meteorological observations to us as though we were a teletype with special teletype characters, no protocol, and no XON/XOFF capability to throttle the transmission of data. Since the multi-tasking VAX does not easily handle applications that require

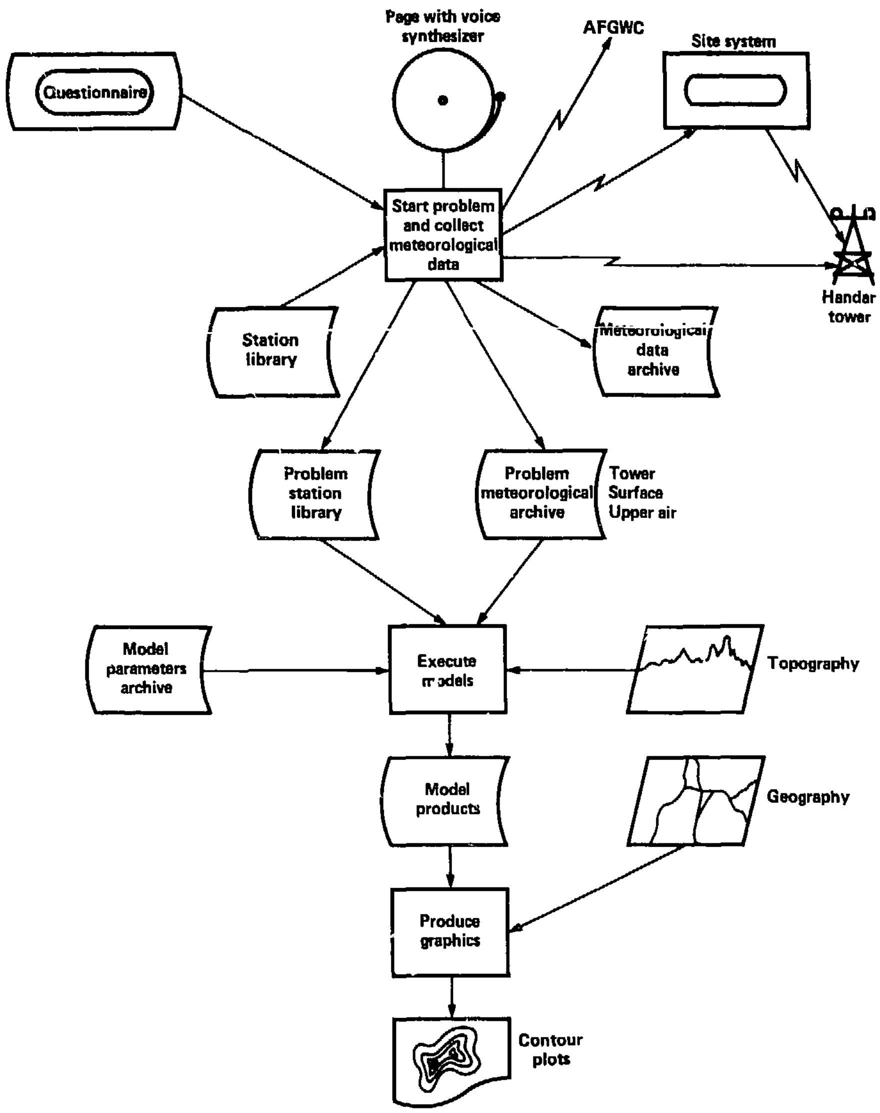

Fig. 1. Diagram of ARAC emergency response system.

interrupt response, we decided to use communications front-end processors to offload communications tasks from the VAX. Note, however, that the overhead in software development required to manage multiple communications front-end processors with multiple lines is considerable and something we would have preferred to avoid had an alternative been available.

The right protocols was another consideration. We chose DDCMP, a DEC standard protocol to communicate with the remote site systems. DDCMP currently exists as both a hardware product (in microcode) and a software product for the VAX, but we needed it to communicate between the LSI 11/23 communications front-end processors and the PC350 site system. DEC had led us to hope that they might provide DDCMP support on the PC350, but the capability never materialized and we were forced to write our own DDCMP for both machines. If we had the decision to make again, we would have used a simpler link-level protocol between the LSI 11 and the PC 350. We were able to use DECNET, a standard DEC communications protocol, to communicate between the VAXs and the LSI 11/23s. While we could not convince AFGWC to develop or use a new protocol, the AFGWC was willing to communicate with us using the protocol now being used to communicate with the National Oceanic and Atmospheric Administration (NOAA). Thus, we are currently developing an LSI 11 version of the NOAA protocol to use on our end of the link with the AFGWC.

Our next consideration was whether to write a logical link communications system or a message routing system. Although a logical link system is more versatile, our goal was not versatility, but a single-application ARAC system. In addition, the logical link communications software would have required more development. When we considered that we might have as many as 39 physical links to the communications front-end processor at any one time, each of which could have multiple logical links, we felt that the potential number of concurrent processes (including the models and application programs) was more than the VAX could handle. Consequently, we chose a message routing system.

The user interface to communications was another consideration. We decided that we did not want communications software imbedded in code throughout the system, so we wrote a user interface. A single user routine provides the node, message type, and message. The message type indicates whether the message to be transmitted is a text message or a file specification. This software ensures that the message or file is delivered. It queues all messages until they are acknowledged and automatically resends messages that are not acknowledged. It also routes received messages and files to the appropriate receiver process. If a receiver process does not exist, it creates one.

Another important consideration for the new system was how to establish and maintain a communications link between a site and the central system during an emergency. In a real emergency local phone lines get busy, making it difficult to establish a connection. Once established, the connection should not be dropped, except by operator command, until the problem terminates. At supported sites, we can enhance our ability to establish connections during an emergency by obtaining special phone numbers for the site system. Our communications software also employs the concept of a line *opened permanently* vs a line *opened temporarily*. Whenever a questionnaire for an emergency or exercise is filled out, a line is *opened permanently* in software. When the problem terminates, this is changed to *opened temporarily*. Lines *opened temporarily* are disconnected in communcations software when no traffic has occurred for a default time period. Lines *opened permanently* can only be disconnected by explicit operator command.

Parallel Treatment of Multiple Problems

Because the system has to handle multiple problems simultaneously on the same computer, we had to assign different priorities to each problem so the more important problems would run faster. We did this by associating a level of response with each problem and setting priorities based on the levels of response—normal, data collection, assessment, exercise, alert, and emergency. The priorities for all processes associated with a particular problem are set on the basis of their level of response.

For multiple problems to run simultaneously, multiple copies of a particular program must be run simultaneously. That means that the different copies of a single code must be executed using different data sets. This is accomplished by using logical names for all files in the system. When a problem is started, the logical names needed by the programs are assigned to physical disks, directories, and filenames. The physical names are determined by the names of the sites for which the problems are being run.

Communications with Background Tasks and Interprocess Communications

Although the system runs with minimal user intervention, the meteorologist is free to interact with the system at will. Therefore, the system was designed so that much of it hibernates in the background, detached from any terminal, ready to respond to operator commmands and other discrete asynchronous events. Systemwide, all software for communications and for requesting and decoding meteorological data runs continuously in the background. Background processes are also initiated for managing each problem until it is terminated. For example, filling out a questionnaire for a particular problem causes the following processes to be started automatically:

- The selection of meteorological stations relevant to the problem.
- The issuance of meteorological data requests for those stations and, if the data are not received, automatic requests for the same observations from alternate sources.
- Continued data requests for those stations at default intervals in the future.
- Dialing and connection to the corresponding site.
- The preparation of preliminary calculations of the dispersion and deposition of released material.

Interprocess communications between tasks and with background tasks became a key technical issue for this system. The VAX architecture supports four basic methods which we use for interprocess communications: event flags, mail boxes, global sections, and a lock manager. In the following paragraphs we briefly describe these methods and our use of them.

Event flags. These binary, single-bit flags allow separate processes to synchronize and control processing by setting, resetting, and examining event flags.

Mail boxes. Mail boxes can contain one or more queued messages. Multiple processes can write to a single mail box. Writing to these mailboxes can cause one process to notify another process that there is new mail for it to read. This allows the receiving process to hibernate until it is notified. A single process can be written to service asynchronous notifications from multiple mailboxes and/or other asynchronous events (e.g., completion of input/output, or the expiration of a timer). This capability is used extensively to allow one task to service many applications to facilitate the sharing of code and databases and to synchronize these activities.

Global sections. Global sections can be used as a shared memory access to a file, and multiple applications can map to the same file in memory. After a program calls a single system service to map a global section, the file is treated as an array in memory. Since an application, physically, has only a window into the file, the VAX architecture takes care of paging in referenced data and updating them in the disk file. The VAX architecture also opens the disk file when the first application maps the global section and closes the global section disk file when the last application disconnects from the global section. This feature is highly useful because it allows the user to intervene in automated background tasks. For example, the meteorological stations to be used for a particular problem are selected automatically while the questionnaire is being filled out and are stored in a global section library. The automated background process that requests meteorological data for stations at specified intervals until problem termination has access to this library. At the same time, the meteorologist can interactively add, change, or delete stations in this library, which is shared with the background processes.

With the VAX 11/782, we can use global sections to allow model calculations to run on the attached processor while the graphics display process runs on the primary processor and accesses the results of the calculations in a global section in shared memory. Global section can also be used to affect the processing of a task while it is running. For example, a global section table of communications parameters is maintained in memory. An interactive communications operator process can update these parameters to set a node down for service, to change the time delay for retransmitting messages that were not acknowledged, to cancel a waiting text or file request, etc. These changes will take effect immediately without having to restart communications.

Lock manager. The lock manager allows synchronization of access to one or more resources and is based on the principle of semiphores in granting and denying locks. Once a resource is locked, access is denied, thereby allowing a single process to update or control the updating of multiple resources simultaneously. This feature is extremely useful for input/output where multiple processes need to read and write to the same database concurrently or when a single transaction must update multiple databases simultaneously.

Data Consistency Between Distributed Processors

Data consistency is a requirement in a distributed environment. For example, the location information of a site meteorological tower cannot be contained in only one database. The site system needs this information to run local calculations. The central system also needs this information to associate with an observation, whether the observation is received through the site system or directly from the tower. When key information must be stored on both systems, we have tried to provide a means for verifying that the information is consistent in all places. For example, the location of the meteorological tower is forwarded to the central system with every observation and then validated against the same information in the central system library. When a questionnaire is filled out at the site, site information is forwarded for verification. To handle the slight differences in values that result because of the different hardware being utilized, we established error tolerance windows. Our design goals of minimizing the redundant storage of data and using verification strategies for all information that must be stored redundantly were of the most value in this area.

One data consistency problem we have yet to solve is the fact that the tower configuration is often site-specific. The addition or recalibration of instruments or an increase in collection frequency can result in modifications to the Handar tower microprocessor program, which is then reloaded from the site system to the tower. ARAC central needs to know when such modifications occur so we can have some control. We would also like to maintain the latest Handar program for each site so we can reload the tower from ARAC central if the site system is unable to reload it.

Another data consistency issue is the synchronization of time. We chose Greenwich Mean Time (GMT) as a standard because all standard meteorological data are reported in GMT and it is unambiguous across all time zones. Since site systems may be physically located in any time zone, local authorities are asked whether the time being entered on a questionnaire is GMT, local standard time, or local daylight time. Then, before this information is archived and sent to the central facility, a site-specific time-conversion factor is used to convert it to GMT.

The ARAC central system and all site systems and towers are synchronized to GMT. The VAXs, site systems, and Handar microprocessors have an internal clock with battery backup. To verify and adjust its internal clock, the VAX uses a hardware device to intercept the radio signals that report GMT. Whenever connections are established between the central system VAX and the site system computer, the site system clock will be validated against the VAX clock. Whenever the tower is accessed, the internal clock is checked. If the clock on the tower is in error, the tower program is automatically reloaded with a new clock value. Since 15 minutes of observations are lost every time the tower program is reloaded, we use a window to allow an error tolerance and only reload when the error exceeds the tolerance. One problem we have not resolved on the central system is the fact that the timer used for comparison may not be accurate if the connection to the site system is delayed on the communications front-end processor.

Other data consistency problems involve differences in national and international units of measure. To facilitate the entry of information, we have allowed for most conceivable units. Once received, these units are converted to standard units of measure for archiving and display. By sharing a common data dictionary during development, we ensured that data in both the central and site systems were stored in the same format.

Human Engineering Factors (i.e., Ergonomics)

Since more than half of all emergencies and exercises occur at nonsupported sites, all the capabilities of the site system also had to be resident on the central system so that the ARAC meteorologist could perform the functions normally performed at the site (i.e., enter questionnaire information and supplementary meteorological data). To make it easier for the ARAC meteorologist to help a novice site user fill out forms, we wanted the two systems to look as much alike as possible. However, the software for developing forms (i.e., DEC Forms Management Software) on the two systems was slightly different, and the expertise of the meteorologist and novice user differed radically. This meant that forms that could easily be understood by a novice at the site were too tedious for the ARAC meteorologist. Indicative of the difficulty we had in this area was the fact that we had to iterate three times on the design and implementation of the forms before we arrived at an acceptable tradeoff of uniformity and functionality.

Error Handling for Background and Distributed Tasks

Error handling was also a problem. While interactive processes used brief but descriptive error messages, the maintenance programmer needed more detailed information as to where the error occurred, what the code was trying to do when the error occurred, and a traceback of called routines. A background process also required a way to communicate errors to the operations staff and/or a maintainance programmer. We accomplished this by developing a common error routine with a logical name assigned for output error messages. This allowed the developer to assign the logical to his terminal so he could examine any errors that occurred. In production the logicals were assigned to a shared file designed to hold system- or problem-wide error messages. Key errors that occurred on the remote site system were also formatted into operator messages and sent to the central system, whose maintenance and operations staff is responsible for fixing problems that occur on the site system.

Testing Difficulties

Developing and debugging detached distributed processes, some of which respond to asynchonous events, was more difficult than we had anticipated. The VAX has an interactive debugger that can be used during initial development. When a programmer begins testing the process in a detached mode, however, the debugger can no longer be used. In addition, some processes were highly dependent on run-time events; therefore, it was often difficult to create the events that needed to be tested. Some time-dependent events were also difficult to debug since timers expire artifically when one attempts to run the process with the interactive debugger. Heavy use of our shared error-reporting routine helped determine the scenario that caused an error, but recreating the event could still be difficult.

Once the communications software was in production, it was difficult to test changes and enhancements because the hardware had to be physically attached to either the production software or the software being tested. Therefore, we had to bring the production communications software down in order to bring up a test version of the software. This was also true for other portions of the central system software. While we could simulate the receipt of meteorological data or other information from communications, testing with live information required that we bring down the production software to try new releases.

Backup Systems

In case of hardware or software failure, we have to move problems quickly from one hardware/software component to another. Since our choice of DEC VAX cluster architecture allows us to access the same disk files from either VAX, the backup machine can restart at the point the primary machine failed. Putting our communications front-end processors on Ethernet also allows us to access them from either VAX.

We have both a primary and a backup source for obtaining meteorological observations. If the requested data are not received within the specified time interval, they are requested from the backup source. The primary source for world meteorological data is the AFGWC, and its backup is WeatherNet. The primary source for tower data is the site system, and the backup source is the tower.

Project Management

Effective project management includes a workable implementation strategy, good estimates of the time required to develop software, and automated tools for scheduling software development tasks and monitoring progress. Although we tried several project management tools, we found them to be of limited use for software development. Most are based on the philosophy of predecessor and successor tasks (i.e., laying the foundation before putting up the walls). On a software development system, one could schedule development of every software component to start at the same time, given enough resources. Because of our limited resources and a four-year development plan, we found it useful to schedule intermediate deliverables according to a preferred priority. The criterion used to determine this priority was centered on replacing entire components of the system and on eliminating its dependence on and vulnerability to the existing hardware. We then scheduled preferred, rather than required, predecessor and successor tasks.

Most project management tools also assume that a resource (i.e., a person) can be assigned to any software development task and move from task to task with no loss of performance. Software development tasks require control of what resource is assigned to what task, particulary when different skills are required to develop different software components. Usually, that resource must also complete the assigned task, without interruption, before it is assigned another task.

Halfway through the four-year development cycle we began to use the project management software of Apple's LISA personal computer to produce an overall diagram of how the software components would come together. We then used it to schedule, report, and manage the development and integration of the intermediate components. This proved to be extremely valuable when we had to integrate components of the central system, site system, and communications together at the same time since the people involved in the integration were members of different development teams.

Another consideration for project management was the choice of a common software development methodology. In this case, we chose Yourdon (De Marco, 1978; Page-Jones, 1980), which provides a method for building a paper model of the system. The model is then verified with users and developers of interfacing processes before the code is written. (For more information on Yourdon, see Appendix B.) The use of a common methodology with walkthroughs enhanced communications between developers, improved the quality of the final product, and increased the quantity of reusable code. It also produced a more modular product that simplified the addition of enhancements. Maintenance was easier because the common methodology produced documentation that all developers could readily follow. Common software development tools and libraries made it easier to comprehend and access other people's software. A software librarian was also a valuable asset. Our software librarian ensured that all software and documentation were complete and promptly checked into our library.

SUMMARY

The lessons ARAC learned during the development of its first distributed emergency response system have been incorporated in the de-

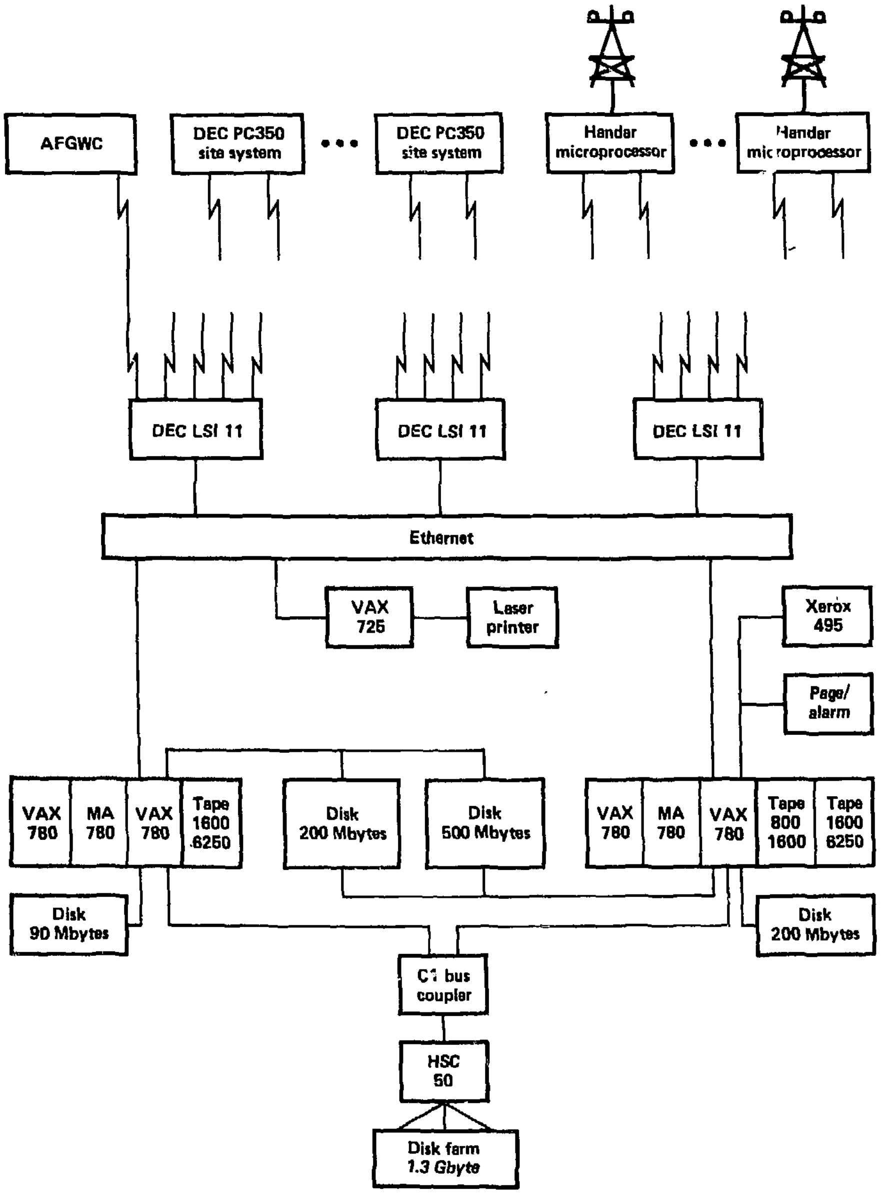

Fig. 2. ARAC hardware configuration.

sign of the new system, which offers greater flexibility and the ability to handle future needs for many years to come. Communications was the hardest problem to solve. Front-end processors solved our hardware problems, but the software needed to control the processors required a large overhead in additional software. Interprocess communications techniques were valuable for communications between processes and for interactions with background processes. Data consistency between the separate processes and processors had to be insured in code rather than rely on humans to enter it consistently. Human engineering factors were important in designing consistent interfaces to all interactive software, but different levels of user expertise required different interfaces. Routing error messages to users when software is detatched or resides on separate processors is also an important design consideration. Testing is difficult when the hardware that must be used for production is needed for testing. Detached processes are also difficult to test. A method for falling back to backup software and hardware systems is also an important design consideration.

We expect to complete development of this new system in 1986. At that time, the VAX architecture will be seven years old and we will have to begin considering new architectures and new capabilities for the next-generation ARAC system.

REFERENCES

Demarco, T. (1978). *Structured Analysis and System Specification.* Yourdon Press, New York, NY.

Page-Jones, M. (1980). *The Practical Guide to Structured System Design.* Yourdon Press, New York, NY.

APPENDIX A. HARDWARE CONFIGURATION FOR THE NEW SYSTEM

Host computers. Our host computers (see Fig. 2) for the central facility consist of two VAX 11/782s (using VMS). One VAX is used for operations, and the other is used for backup and software development. A single VAX 11/782 consists of two VAX 11/780s with shared memory. Jobs involving primarily input/output (I/O) run in the primary processor to which all the peripheral I/O devices are attached, and jobs whose primary function is not I/O, but calculations, run in the attached processor. This scheduling is performed by the operating system transparent to the programmer. In the ARAC system, a model execution consists of meteorological data inputs, calculations, and graphics outputs. The meteorological data and graphics portions of the process run in the primary processor while the computations run in the attached processor. A VAX 11, 725 with laser printer is used for office automation and for the preparation of technical documents.

The two VAX 11/782s are clustered through a star coupler. Attached to this cluster, through a CI bus, is a 1.3-Gbyte disk farm. The disk farm is used to store the terrain and geography databases, meteorological data archives, the ARAC system software, and the VAX operating system. Files on the disk farm are available for read or write from either VAX. This includes a single copy of the VAX operating system, which is shared by both VAXs. The VAX 11/782s share an additional 700 Mbytes of dual-ported disk space, as well as 280 Mbytes of local disk space that that can be shared, through the cluster, for read or write from either VAX.

Communications front-end processors. Three DEC LSI 11/23s, which are used as communications front-end processors running under RSX11S, are available to either VAX through Ethernet hardware, using Digital Equipment Corporation's DECNET software. Each LSI 11/23 supports 11 dial-up lines and 2 leased lines. The dial-up lines are used to dial to and from the site systems and meteorological towers and are capable of handling 300- or 1200-baud communications. The two leased lines, in reality one multiplexed line that handles one 300- and two 9600-baud ports, are used to request meteorological data (at 300 baud) from the AFGWC and to receive (at 9600 baud) meteorological data and products.

Site systems. The site systems are DEC PC350 professional computers that run under POS, an RSX-based operating system. This system has a color monitor, a dot-matrix printer, a 10-Mbyte internal hard disk, and an internal 300/1200-baud modem. Some site systems also communicate by leased line, at 300 baud, to a Handar 540A microprocessor used for collecting and archiving data from instruments on a meteorological tower.

Meteorological towers. Each meteorological tower consists of one or more levels of meteorological (i.e., weather) instruments with an attached Handar 540A microprocessor to collect and store approximately 10-days worth of observations. Each tower is attached, by a leased line at 300 baud, to a local site system. It also has a dial-up port that the ARAC central communications front-end processors can use as a backup to obtain meteorological data when the link to the site system is down.

Xerox Telecopier. A Xerox 4951 telecopier attached to one VAX and available to the other, through the cluster, gives ARAC the capability to transmit VAX-produced graphics products and textual documents directly from the VAX to a remote telecopier at the accident site.

Page/alarm system. The page/alarm system uses a microprocessor with a voice synthesizer to give voice-synthesized messages to operators in the computer center. On command, it calls on-call personnel by phone or radio-page devices with a voice-synthesized message, informing them th[illegible] hardware or software is down or that they are needed in the ARAC center.

APPENDIX B. YOURDON STRUCTURED ANALYSIS AND DESIGN

The software development staff consisted of about 10 computer scientists, 2 programming technicians, 3 engineers, and 5 meteorologists. Project management viewed a common development methodology as essential for software development. The goals for this methodology were to:

- Present the functionality for the proposed software in a way that would assure the user that requirements were being met.
- Define the interfaces between the various software components.
- Provide a common format for presenting and documenting software.
- Improve the quality of the analyzed system and the individually designed components.
- Simplify maintenance and the addition of new capabilities.

The software development methodology they chose to use was Yourdon (DeMarco, 1978; Page-Jones, 1980), which consists of two basic components: structured systems analysis and structured software design in conjunction with a walkthrough process.

Structured Analysis

Yourdon structured analysis uses data-flow diagrams, a data dictionary, and mini-requirement specifications to design a system and state its requirements. The data-flow diagrams present a layered look at system processes and interfaces. The methodology is based on the theory that modeling data transformations instead of functionality produces a better system design. The data dictionary defines, in one place, all the data elements, records, files, etc. in the system. The mini-requirement specifications state the transformations required to produce outputs from the inputs.

Structured Design

Yourdon structured design uses a structural chart that looks very much like an organizational chart. This chart depicts the boss module and the calling structure for the subordinate routines. Data passed between the routines are shown on the structural chart. Pseudo-code is written for each module it depicts.

One of the premises of the Yourdon methodology is that developers who build a fairly complete paper model of the system before writing code will find it easier to add modifications and corrections to the paper model than to the coded system. People who attend walkthroughs of the paper model are users who determine if the software will meet their requirements, software developers who work on interfacing processes, and peers who review the quality of the paper model. The paper model is updated throughout development and becomes the final documentation, eliminating the need to document software after development.