Next generation air traffic control automation

by A. S. Debelack
J. D. Dehn
L. L. Muchinsky
D. M. Smith

The automation of air traffic control (ATC) applications has challenged the technologies traditionally used for building and testing large data processing systems. These systems are characterized by complex applications, extensive external and user interfaces, and rapidly changing functional requirements. The Advanced Automation System (AAS) of the Federal Aviation Administration combines high availability, extensibility, and extraordinary functionality into a single distributed system, running at multiple locations. By requirement, the AAS must execute continuously without error, without interruption for upgrades or maintenance, and with the complete trust and confidence of air traffic controllers using it at over twenty centers. Each ATC center uses the same application software coupled with varying quantities of hardware and an extensive base of data to tailor the center to its unique geography and operational procedures. In this paper, we describe the characteristics and architecture of the AAS and focus on key mechanisms of availability and extensibility.

The primary objective of air traffic control (ATC) is to provide separation services for aircraft that are flying in controlled airspace, or where poor visibility prevents pilots from maintaining visual separation. Aircraft are separated from one another and from terrain hazards. Since pilots (in some cases assisted by cockpit computers) fly the aircraft, an important element of air traffic control is the transfer of requests from pilots to controllers and the transfer of clearances from ATC controllers[1] to pilots.

Air traffic control is a closed-loop activity in which pilots state an intent by filing flight plans. Controllers then plan traffic flow based on the total number of flight plans and, when possible, give clearance to pilots to fly according to their plans. When planning conflicts arise, controllers resolve them by clearing pilots to fly alternatives to their plans to avoid the conflicts. If unpredicted atmospheric conditions (e.g., wind speed or direction) or pilot actions cause deviations from conflict-free planned routings, controllers issue clearances for tactical maneuvers that solve any resultant problem, albeit not necessarily in a way that furthers the pilot's goal of reaching the planned destination at a certain time.

Planning conflicts arise when multiple aircraft require the same resource at the same time. Arrival and departure time on runways, and convergence on ground-based navigation aids at the same altitude, frequently cause conflicts. Bad weather can further constrain these resources, aggravating contention. In addition, aircraft operating at different speeds must be accommodated sequentially in crowded areas around airport terminals.

 0018-8670/95/$3.00 © 1995 IBM

Automation has provided controllers with tools to improve the use of these limited resources, while increasing the assurance that safe separation is maintained. However, the introduction and evolution of automation has not been easy. It requires complex infrastructures for communications and surveillance, plus large investments in data processing equipment and workstations.

Even more problematic is the need for aircraft to be equipped with communications and navigation equipment. Private pilots and owners of small aircraft value their independence and by law have the same rights to use airports and navigation aids as commercial and public carriers. Therefore, ATC system modernization must be implemented in such a way that users are not required to upgrade their equipment and are not unduly constrained in their use of exhaustible resources.

The AAS supports all aspects of air traffic control with an integrated suite of applications.

Notwithstanding such considerations, some airspace is necessarily restricted to aircraft with specified equipment or capabilities and requires ATC clearances to be entered. This airspace includes the busiest terminal areas and all airspace above 18 000 feet. Separation services must still be provided by controllers in all areas when necessary due to conditions of poor visibility, so long as the pilot has a compass and a radio and has the appropriate navigation skills.

The Advanced Automation System (AAS) is being implemented by the Air Traffic Control Division of Loral Federal Systems under contract to the Federal Aviation Administration (FAA). Prior to the AAS, ATC automation consisted of two independent systems, one for use in terminal areas where speeds are low and density is high, the other for use in en route flight phases where speeds are high and maneuvers are infrequent. Both environments require radar surveillance, but information about desired flight plan routes can be less useful in terminal areas where maneuvering is frequently tactical and the primary goal is either to land the aircraft or to depart the terminal area. For these reasons, the two systems evolved differently and are not compatible with respect to procedures, equipment, software, or facilities.

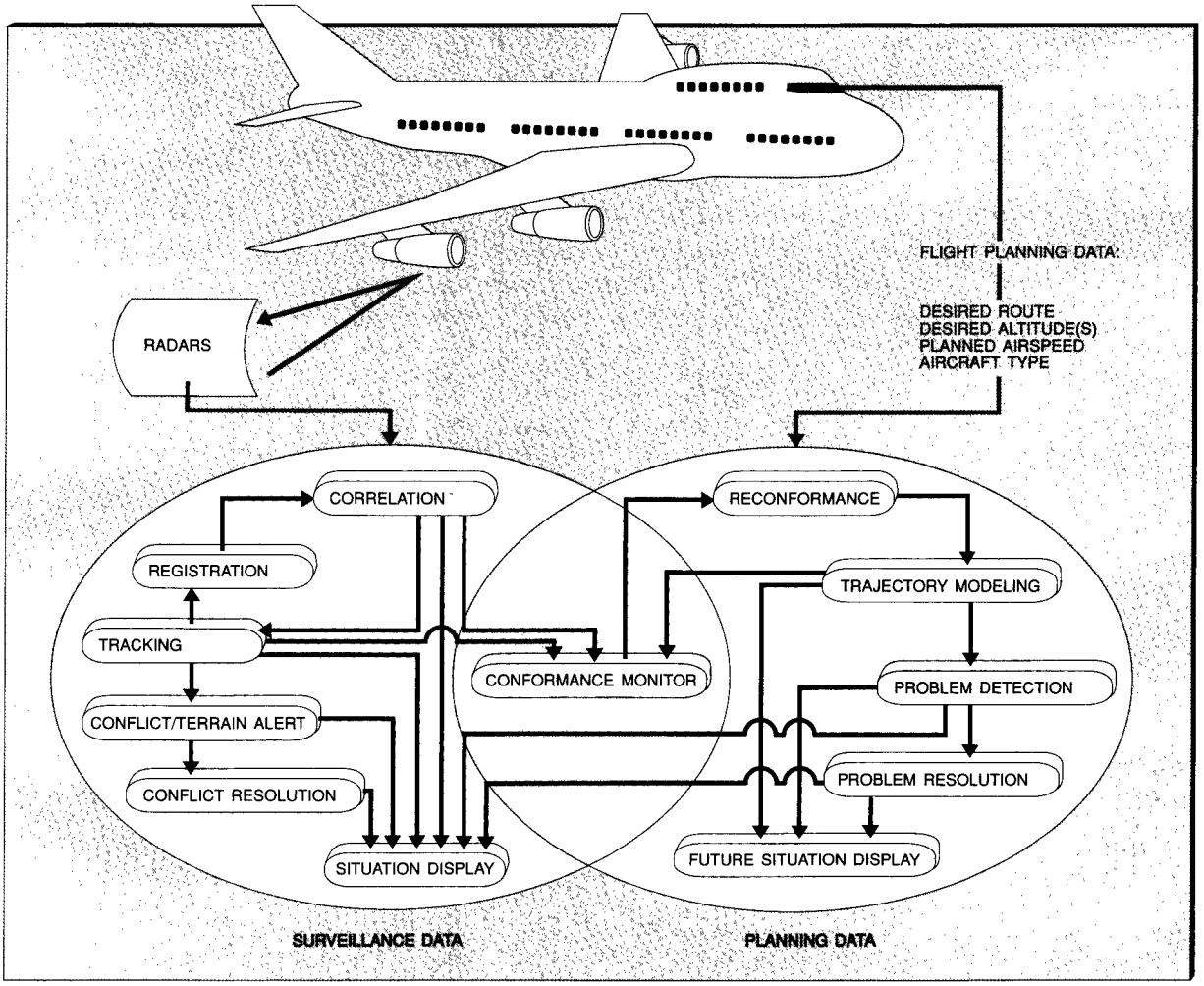
The AAS provides the FAA with an automation infrastructure usable in both environments. To optimize availability, communications, and capacity, the system will first be installed at each of 20 en route centers. Each of these installed systems will provide support for between 25 and 100 ATC sectors. A sector controls a volume of airspace and specified approach or departure functions, or both the airspace and the functions together.

ATC automation evolution

The AAS supports all aspects of air traffic control with an integrated suite of applications that support radar-based and flight-plan-based surveillance, prediction, and resolution. Radar reports are used to generate aircraft *track* estimates of position, altitude, velocity, and acceleration that support surveillance and short-term separation assurance. Flight plans are used to create *trajectories* consisting of predicted position, altitude, velocity, and time for the entire route of flight and are also used to support planning and dynamic rerouting. Figure 1 illustrates this relationship between data sources and several ATC applications.

Correlation associates radar returns with tracks, using either discrete beacon IDs broadcast by on-board transponders or by matching predicted and actual positions. These returns are used to update three Kalman[2] filters, each tuned to address different ATC requirements. The horizontal *display tracker*, a distributed four-state extended Kalman filter, provides accurate position predictions to support correlation and display of aircraft velocity. The two-state *altitude tracker* has been adapted to process highly quantized, noisy data from on-board barometric pressure transponders. This tracker supports correlation, display of climb rate, and conflict alert. The centralized *separation assurance horizontal tracker*, a five-state extended Kalman filter, provides accurate veloc-

Figure 1 ATC applications and related data



ity and acceleration estimates to support registration and track-based conflict alert.

Registration compares the centralized track states with reported positions from multiple radars to estimate range and azimuth bias from each radar. This calculation uses a least squares estimator to minimize the discrepancy between radar reports and the centralized position estimate. The resulting bias corrections may then be applied to incoming radar reports and used as an indication of radar health.

Conflict alert predicts loss of separation between an aircraft and other aircraft, restricted airspaces, or minimum safe altitudes. Because velocity estimates fluctuate, counts of predicted separation violations are kept over several track updates to determine the likelihood of a violation with confidence of up to 99 percent and warning times of 30 to 120 seconds. When a violation is predicted, *conflict resolution advisories* are provided to the controller, suggesting clearances that will increase separation without introducing additional problems. Track, alert, and advisory data are paired with flight plan entries and superimposed over navigation and radar weather maps on a *situation display*.

Flight plans contain the planned route of flight expressed as a sequence of defined fix positions

and coded routes. *Trajectory modeling* uses these positions with filed cruise altitudes and speeds, aircraft performance characteristics, and wind and temperature estimates to build a predicted trajectory for the entire route of flight. The trajectory is recalculated as flight plan amendments are entered to accommodate pilot requests or controller planning. *Conformance monitoring* compares these predictions with radar-based position reports. If the radar position falls outside of the appropriate conformance bounds for the predicted position and maneuver characteristics of the aircraft, *reconformance* adjusts the trajectory to remove the discrepancy and, if possible, revises the modeling assumptions used to build the inaccurate trajectory. The resulting trajectories are used by *problem detection* to predict potential loss of separation between aircraft 8 to 20 minutes before separation violations occur, with acceptable false alarm and missed alert ratios. Problem detection also checks for aircraft with restricted airspace problems and for noncompliance with resource allocation schedules created by the AAS or by external traffic management systems. *Automated problem resolution* (APR) resolves all three problem types.

The maneuvers generated by APR must meet complex constraints and optimality criteria to produce operationally suitable maneuvers that return aircraft to their requested flight profile. These maneuvers must not only resolve known problems without introducing new ones, but must also do so while minimizing the impact on the pilot and the air traffic control system. Several methods are being developed to produce near-optimal maneuvers with acceptable computational overhead.

A heuristic search definition inductively classifies and ranks all possible maneuvers by using known information about the problem and making assumptions about the success and characteristics of maneuvers that have not been generated. As the list of maneuver classes is traversed to generate the required number of maneuvers, the search order and class definitions are modified using information gained from previous maneuver generation attempts.

Maneuvers are constructed by explicitly representing known constraints in an appropriate maneuver state space. These constraints are provided by maneuver type definitions, heuristic search assumptions, and the requirement to maintain separation between aircraft. They form boundaries between allowable and unacceptable maneuvers. Explicit representation within the state space allows constraints to be iteratively applied to account for unanticipated interacting aircraft and refined to account for modeling error in the constraint representation. Appropriate selection of the state space basis reduces many optimality criteria to linear functions of the state variables.

It is important that the AAS provide the framework for both predictable and unpredictable future functionality.

If maneuvers of a single aircraft in a single dimension cannot resolve the problem, maneuvers of multiple aircraft or maneuvers in multiple dimensions are required. These maneuvers have many degrees of freedom and are expensive to optimize in full generality. However, information gained by the failed maneuver types can be used to try to resolve the maneuver by building one maneuver on top of another. Two strategies for selecting the base maneuver are under investigation. The first attempts to remove an aircraft from the closed state space, select a base maneuver from the freed space, and build a second maneuver to resolve the problem created by the base maneuver. The second chooses a base maneuver which, when coupled with the second maneuver type, will result in a later time of violation. The second maneuver then has a greater chance of successfully resolving the first problem.

AAS architecture rationale

Both ATC applications and their underlying computer technology will continue to evolve. Therefore, it is important that the AAS provide the framework for both predictable and unpredictable future functionality. The system has been

designed to be open, to achieve high functional availability, and to be scalable over a range of traffic loads and mixes. High availability is especially important because the failure of a function means that the efficiency, stress reduction, or safety provided by that function becomes unavailable, and the controller must manually compensate.

Availability is defined by the FAA as the probability that the system will accurately respond to controller or external system stimuli within an elapsed time that depends on the type of input.[3] Generally, radar and display manipulation commands require about a one-second response time, and most planning inputs require a two-to-three-second response time. Delays beyond these values are allowed so long as they are being experienced at no more than one controller position at any one time. Unavailability of the system occurs when more than one position simultaneously experiences functional response delays beyond the maximum allowed values. Simply stated, the system is available so long as the infrastructure is up and providing full services to at least all but one operational position. Allowable service denial time is about 3 seconds per year for critical functions, 32 seconds per year for conflict detection functions, and about 2.5 minutes per year for planning and display data recording functions. These allowable unavailabilities translate into system availabilities of 0.9999999, 0.999999, and 0.999995, respectively.

Such high-availability requirements are not easily attainable or verifiable. It is easy to verify that a system does not meet them but difficult to verify that it does.[4] Furthermore, they can only be achieved (with contemporary technology) if downtime caused by a defect can be expunged from consideration when the defect is corrected. Loral's contract with the FAA requires that the system achieve the required availability at the time of the operational readiness date of the last site, including the effects of expunging. Additional discussion of the AAS availability may be found in Reference 5.

The requirements for adaptability result from the wide range of operational environments in which the system must execute. This situation has thrust a large amount of table-driven functionality into the design. The requirement for extensibility, as mentioned earlier, has forced the use of industry standards; this approach is drastically different from that used in the predecessor system, where high availability was achieved through the use of special-purpose proprietary interfaces and components.

One final point about the architecture: It was established between 1984 and 1988 as the result of a competitive procurement between the IBM and Hughes Corporations (IBM's Federal Systems Company was acquired by Loral in 1994). Since the system would not be entering the formal test phase until 10 years after establishment of the architecture, the hardware available in the mid-1980s would clearly not serve as the ultimately deployed hardware. This condition meant that the software architecture, baselined in 1988, would have to be flexible enough to be implemented on then-current hardware (a contract requirement) and on future hardware (on which more competitive costs would be based).

AAS architecture derivation

Experience with prior high-availability real-time systems such as the predecessor ATC system and the space shuttle ground and on-board systems has shown that software can compensate for poor hardware availability, but no amount of hardware can compensate for poor software availability. The required availability for the AAS could be met easily if software were not included in the equation. However, with software included, it was clear that recovery times were the most critical factors, given the predicted failure rates. If recovery could be made fast enough so that output from in-process threads could be recovered or reproduced within the required response time, the interruption would not subtract from the unavailability budget of the system. Since the system is distributed and each processor has large amounts of memory, restarts of even single processors could not be allowed in the recovery time line. In fact, no matter how small the individual recovery units of software were made, there was not enough time to perform a software unit restart while remaining within budget. Therefore, a combination of mechanisms was required:

1. A standby copy of each application program must be loaded and ready to run in a processor different from the one for the primary copy.
2. The standby copy must either maintain or be capable of quickly constructing the necessary

internal state so that on being given primary responsibility, it could resume interactions with clients and servers of the failed primary.
3. Individual application programs must be hidden from their clients so that their processor location can be dynamically changed without causing interruption to the clients.
4. There cannot be a centralized recovery manager since such a program would be a single point of failure that could use many years of budget in a single failure.

These mechanisms placed constraints on both the hardware and software architectures.

Software architecture. The software architecture is based on data and constrained by network and processor bandwidth. Functions are collected into modules on the basis of the data they need, but in such a way that the transfer of messages between modules never assumes that the modules will reside in the same processor. Each module is designed to be the primary owner of certain system data and to provide the services or functions that use that data. Some processing threads require several modules to be run sequentially. In some cases slowly changing data that support a large number of functions are replicated among two or more modules. One module is designated as the owner and manages and distributes changes to the data. Every module, anywhere in a thread, must always produce a response to an input even if other modules that serve it do not respond to service requests. At the boundary of the system this response may be a message reject, but internally, away from boundaries, the module must save sufficient state so that the thread may be restarted when the unavailable module is returned to service.

Dividing the system into a few large modules reduces the bandwidth and queuing latencies but requires larger processors. It also means that the standby copy of the module maintains larger amounts of data and therefore requires longer transition times when switched from standby to primary. In contrast, creating many smaller modules increases bandwidth and thread times and leads to replication of more data, which in turn increases complexity. Many trade-offs were made to determine the optimum partitioning of data and function, and in some cases the partitioning is still being modified as system development proceeds.

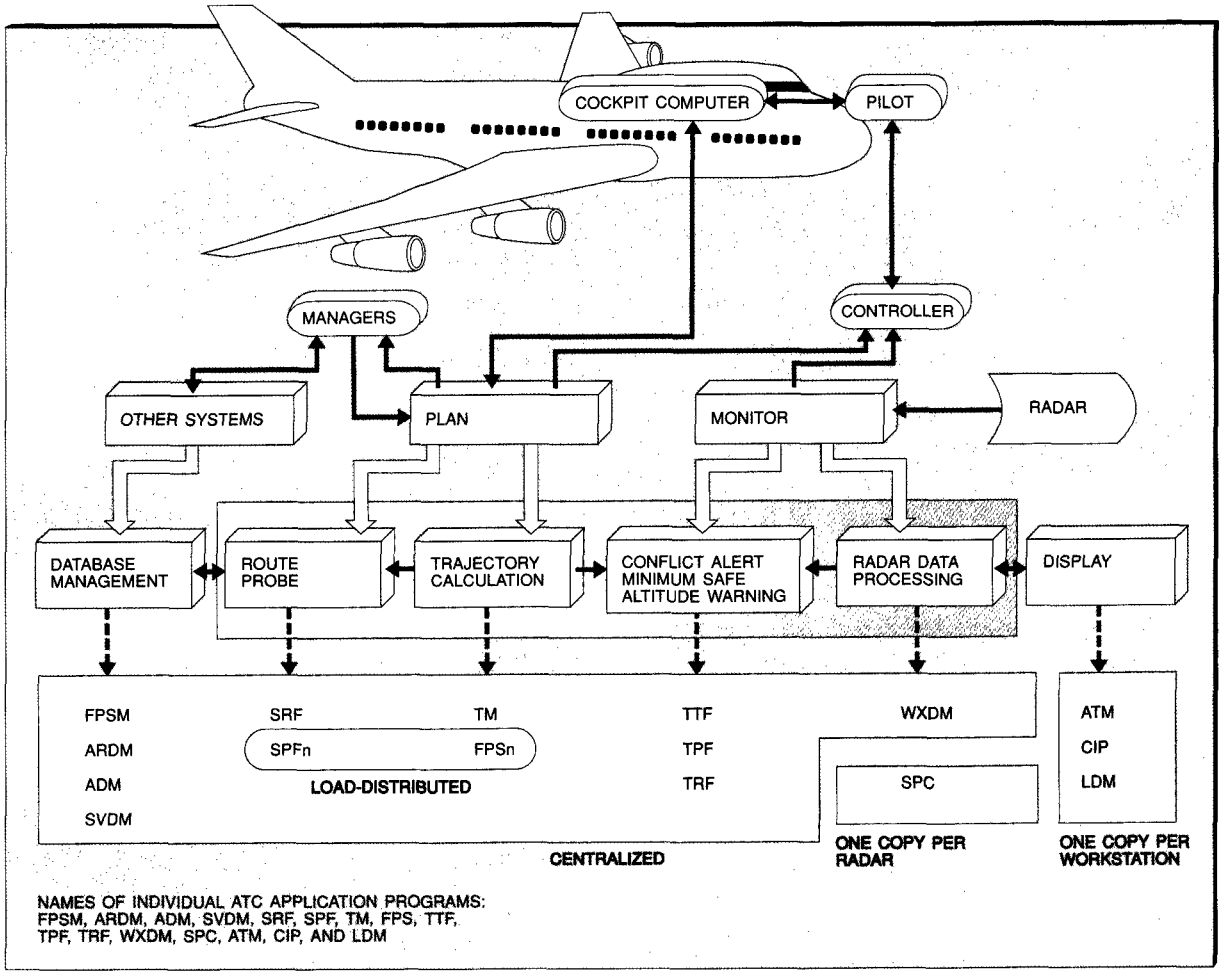
Figure 2 shows the software partitioning currently being implemented and how it was derived. External interfaces and related functions are shown at the top. General categories of application processing are shown in the middle, and the actual application modules are shown at the bottom. For each application module, an indication is given as to whether that module is centralized for the whole system, distributed by external interface or radar, or distributed by workstation. Two centralized modules are subdivided such that they appear as a single server to clients, but such that the workload is spread among them. This distribution is referred to as load distribution and allows small processors to be used but does not improve availability. Note that so long as one radar application and one console application set are running (and assuming that the infrastructure is available), some level of system service is being provided. Controllers may monitor the current airspace situation even if none of the centralized modules is executing.

Hardware architecture. The hardware architecture is distributed in several ways, driven by requirements for availability and extensibility and by processor capacity. An overview of the architecture is shown in Figure 3. One set of central processors is attached to a set of local area networks (LANs) such that each processor can communicate with the others without involving bridges. ATC applications that cannot or should not be distributed are executed on these processors. Two copies (primary and standby) of each application module are loaded in any two of the processors.

Communications between the AAS and external systems utilize gateway processors on another set of LANs. External interfaces are connected through modem splitters such that each interface is terminated at two different gateway processors. For each interface, there is an interface processing module that is preloaded in both of the processors that connect to that interface. One copy of the module is primary, the other is standby. Interfaces to radars are handled in the same way as external systems but have their own set of LANs.

Controller workstations each contain a workstation processor and a display generator. Up to 80 of these processors are attached to a set of LANs. A typical facility will have two or three such sets, and more are possible. Workstations are opera-

Figure 2 Software architecture decomposition



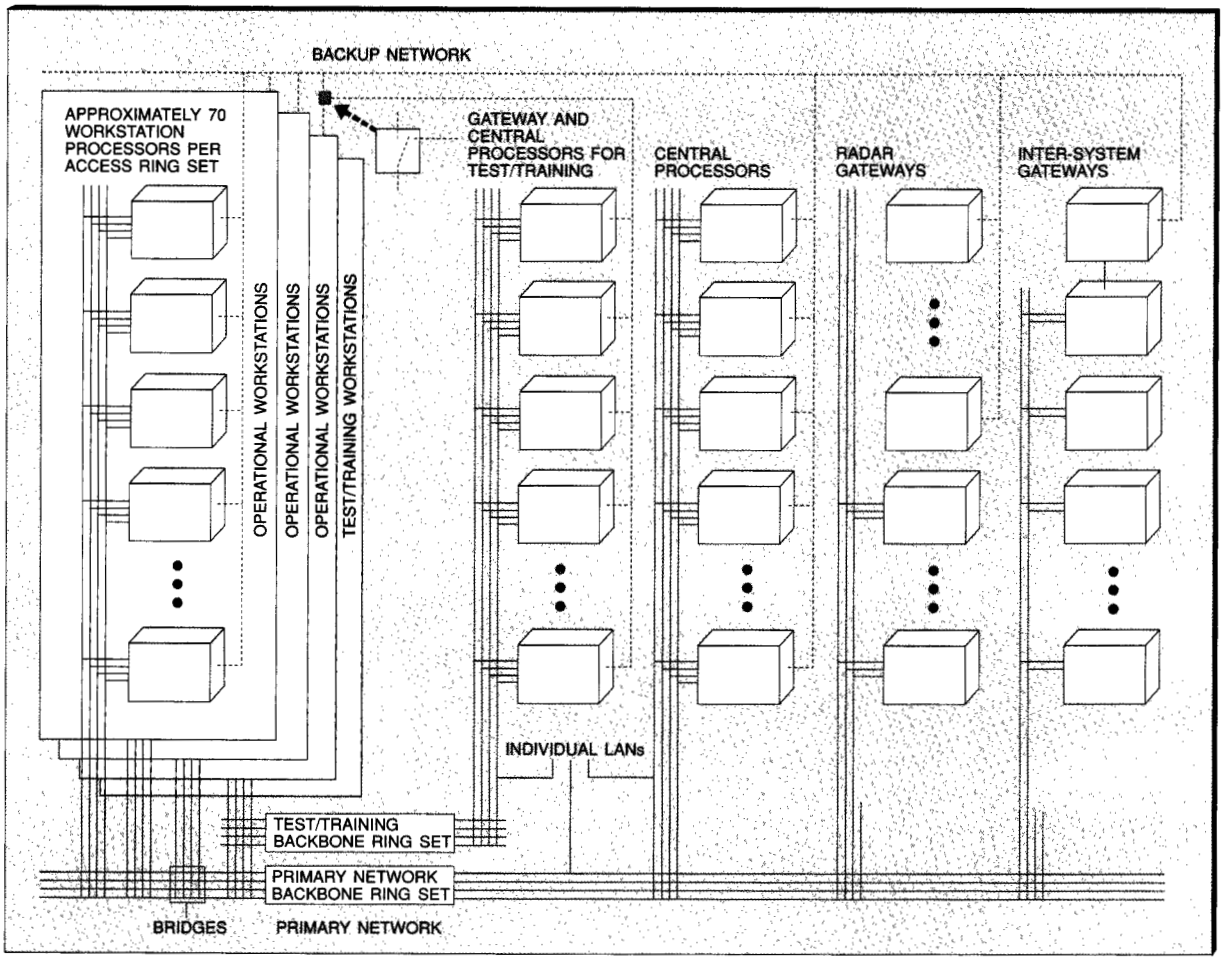
tionally grouped into suites of one to four, operated by a team of controllers. Within each suite, the workstation applications are completely redundant across all workstations in the group. All track control and display data integration applications run in each workstation processor.

As shown in Figure 3, each of these LAN sets is interconnected via bridges to a set of backbone rings. Much of the traffic of the system is between primary and standby copies of each application, which are constrained to be on the same LAN sets. Most radar and communication data, however, move between the gateway LANs and the workstation or central processors. Also, display recording data from the workstation processors are transferred from their LANs to the central processor LANs.

The system also contains additional gateway and central processors to use for test and training activities. These processors are attached to a single set of LANs, which are in turn attached to a separate set of backbone rings. A subset of the workstations of the facility are bridged to both sets of backbone rings so that they can be used either for test and training or for live operations.

This hardware architecture was chosen for several reasons:

Figure 3 AAS hardware architecture

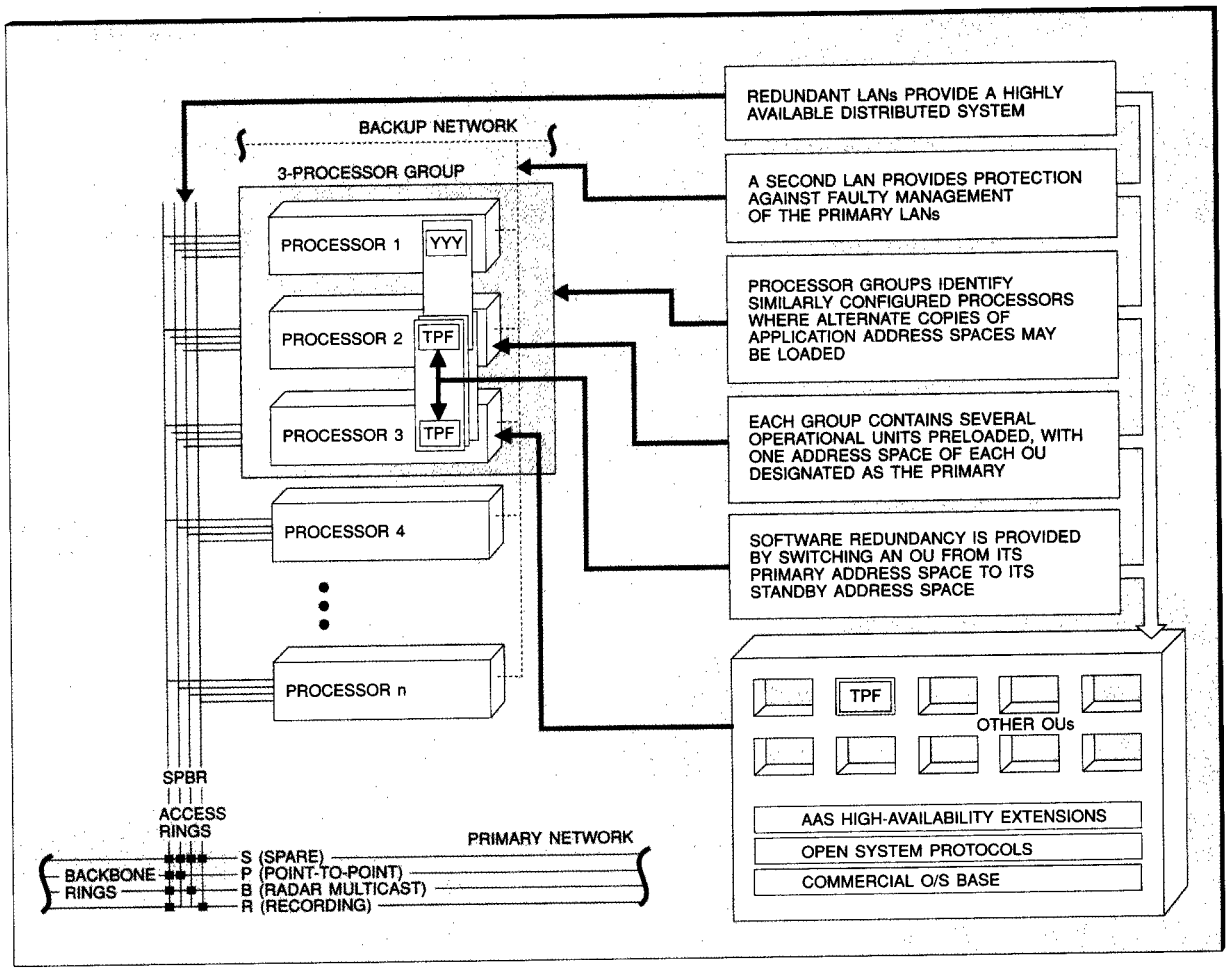


- New subsystems may be added independently of existing subsystems by providing their own LAN sets bridged to the backbones.
- Availability and reliability of the backbone is high because so few stations are connected (bridges only).
- The configuration lends itself to pooling at the FAA's technical center (discussed later), where mimics of up to four sites must be simultaneously configurable.

The collection of LANs described above are referred to as the *primary network*. Each of the four LANs is assigned one of four data types by the network manager of the system. One is always a spare; the other three are for point-to-point, radar multicast, and display recording. The point-to-point LAN is also used for certain nonradar multicasts.

A separate *backup network* carries the data that are needed in modes with the highest availability. This single LAN connects all processors involved in these functions. Bridges and multiple access ring sets are not used here for simplicity and to separate the implementations as much as possible.

Figure 4 Software availability architecture



AAS architecture implementation

The architectural concepts being implemented in the AAS are shown in Figure 4 and are now described. This implementation promotes the high availability and extensibility of the system. It also allows selected components to be upgraded as newer technologies mature. (This is one of the difficult problems facing the builders of highly complex systems that may take many years to complete. State-of-the-art technology at the beginning of system development may be run-of-the-mill at the end. The AAS approach to this problem is similar to other architectures: build a layered system with well-defined interfaces so that the technology supporting a particular layer can be changed without affecting other parts of the system.)

Each LAN in the primary network is a 16-megabit token ring. The backup network consists of a set of Ethernet segments connected by repeaters (forming a single logical LAN). Different network protocols[6] are used on the two networks to avoid a common mode failure of the communication protocols. FDDI (Fiber Distributed Data Interface) and CDDI (Copper Distributed Data Interface) have recently been explored as alternate media technologies.

The target processors have changed over time as technology has changed. Originally, a combina-

tion of IBM RT PCs* and System/390* processors was selected. Currently the IBM RISC System/6000* is being used for all processors in the system. It was chosen for its wide range of memory sizes and processor speeds, for its floating-point performance (many of the complex application algorithms, such as trajectory modeling, make heavy use of floating-point numbers), and for its use of a UNIX**-compatible operating system, AIX*, the Advanced Interactive Executive*. (The use of a commercial operating system allows us to introduce new processor technology with little or no change to the application programs.)

Application redundancy is built upon the processor group concept.

Once a network topology was chosen that provided redundant data paths between processors, the availability mechanisms for the application software were designed.[5] *Processor groups* provide protection against processor failures. A set of applications is contained within a processor group; if one processor in the group fails, the application services that the failed processor was providing are activated in other processors within the group. Group management is provided by membership protocols based on synchronized processor clocks throughout the group. (Clock synchronization is provided among all processors in the facility as a service to applications; the group protocols only depend on clock synchronization within a processor group.) Intragroup communications are kept simple and efficient by constraining all processors in a group to be on the same LAN set. This constraint also ensures that network recovery will always be independent of group recovery. Groups in the AAS architecture typically have three or four processors.

Application redundancy is built upon the processor group concept. A single application (corresponding to the lowest level in Figure 2; for example, the Tactical Predictions Functions, or TPF) is an Ada** program.[7] This program executes in two or more processors within the same processor group. (An instance of an application executing on a single processor is called an *address space*, because that execution entity occupies one virtual address space on the processor.) The collection of address spaces all running the same program within a single processor group is called an *operational unit* (or OU). This OU implements the software "module" introduced earlier. The manner in which an OU provides continuous service in spite of processor or program failure varies from application to application but is based on the following:

1. The group membership protocols rank each address space in the OU (the highest-ranked address space is the oldest).
2. The AAS communication services (one of the high-availability extensions built on top of the commercial operating system base) deliver messages destined specifically for the OU (as opposed to messages broadcast to all address spaces) to the highest-ranked address space.
3. The AAS communication services provide reliable methods to send the same message to all address spaces in an OU.

Given these services, applications use either *tight* or *loose synchronization* to maintain a consistent state in all address spaces. With tight synchronization, all address spaces receive the same input (from a broadcast, for example) and calculate and store any updates to the retained state data. With loose synchronization, only the primary address space receives the input and calculates state changes; these changes are sent to all other address spaces in the OU. When loose synchronization is used, additional protocols are used to recover from the potential loss of the last state data update after a failure of the primary address space. Tight synchronization techniques use less resource at recovery time; loose synchronization techniques use less resource during normal steady state processing.

Additional constraints are placed on inter-OU message traffic to prevent transactions from spanning OU boundaries. (Spanning would reduce OU independence and require cross-OU backup logic.) Every OU in a processing thread must be able to commit its state changes before the thread can proceed to the next OU.

Extensions to the basic program-to-program communication mechanisms (such as TCP/IP sockets or ISO sessions) provide application communication between OUs. These AAS network management techniques allow fast switching of communication traffic when a processor or application address space fails and allow the first order recovery to take place without a centralized name server. Two separate techniques are used in order to keep the failure modes of the two networks as independent as possible:

1. On the primary network, every application address space registers with a centralized name server during initialization. When a connection is desired between the address space of one OU and another OU, the name server returns the list of all address spaces in the destination OU. Connections are then established between the source address space and all destination address spaces. Each address space in the source OU goes through the same initialization logic, thereby establishing connections between all source OU address spaces and all destination OU address spaces.
2. On the backup network, all connections are established through a discovery method. When an address space desires communication with another OU, the requestor broadcasts a message to all processors on the network. Those processors containing address spaces in the destination OU return a message with the processor ID and address space name of the address space of the destination OU to the requestor. TCP/IP sockets are then created with each of the address spaces of the destination OU.

During recovery from an address space failure, the necessary network connections are already in place (which speeds recovery time and eliminates the need for the centralized name server during this crucial recovery period).

These services enable application designers to build fault-tolerant applications. Additional distributed operating system functions and single processor services (such as processor-wide locks, message queuing, data logging, error reporting, and checkpointing) complete the definition of the application address space operating environment. As shown in Figure 4, a single processor configuration may contain many application address spaces. Because applications are isolated from any particular configuration by the high-availability extensions, this configuration can be changed at any time during system execution.

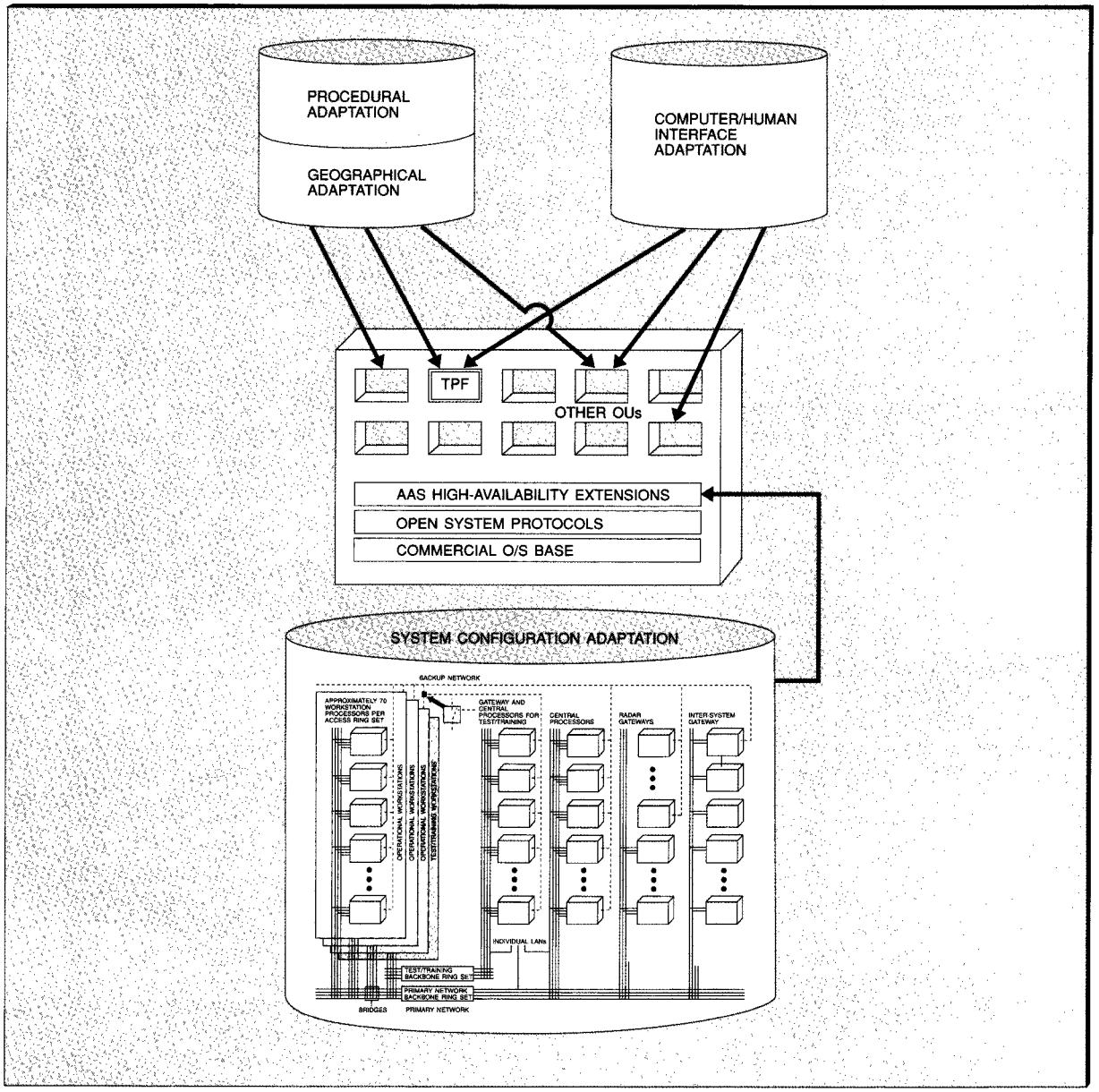
System adaptation and support

Support for the geographically dispersed AAS field installations is the responsibility of the FAA Technical Center (FAATC) in Atlantic City, New Jersey. The FAATC provides capabilities to tailor the system of a field site to its individual operating environment; to perform continual testing of the system (both deployed and new versions); and to maintain all hardware and software components of multiple deployed system versions.

Although all field locations use the same software logic to control air traffic, they are dissimilar in that each has its own description of airspace volumes, real and imaginary geographic boundaries, navigation aids, sector maps, and AAS system configuration. The use of data to cause each center to be different from every other center is termed the *adaptation data tailoring* process. Collection of adaptation data and their entry into FAATC databases are performed by multiple groups of specialists. Data that are not site-unique (like aircraft performance characteristics) are managed by FAATC personnel, whereas local or provincial data unique to a field site are managed by the data specialists of that site.

Incorporation of adaptation data with the ATC software logic is performed at the FAATC using contract-developed support tools. Combining this site-unique adaptation data with the ATC software modules yields a system release for field deployment and on-line execution. Figure 5 illustrates how adaptation data customize a system to a specified geographical and operational domain. For example, center maps (a form of geographical adaptation) define the bounds within which the center controllers exercise jurisdiction. These maps are further subdivided into sector maps for the individual controller teams. Other categories of geographical, procedural, and system configuration adaptation data are used to complete the system definition. These categories include airways and flight plan fixes, special-use airspace, terrestrial features and hazards, navigation aids, aircraft characteristics, computer-human interface adaptation, network topology and processor group configurations, and controller workstation groupings.

Figure 5 Software and system adaptation



SSCC architecture

FAATC testing to verify system enhancements, to generate problem resolutions, and to qualify vendor-supplied software updates is conducted within the System Support Computer Complex (SSCC). Depending on a tester's need for fidelity and the availability of SSCC hardware resources, a variety of test configurations can be created. Configuration fidelity ranges from a simple testbed, to a *logical replication* of an actual fielded system, to a maximum stress setup used for testing the design limits of the system. The ability to faithfully mimic a field site is especially important

for predeployment system testing and problem re-creation efforts.

SSCC hardware must be shared among several classes of development and testing users. This hardware includes controller workstations, simulators that have been developed in lieu of the acquisition of production controller workstations (which are expensive, and unnecessary where most testing is hands-off), and other processors used for gateway functions. The need to share hardware has resulted in the development of a *pools of equipment* concept and a means of switching these pools (refer to Figure 6). Common equipment pools can be switched among mimics to achieve configuration flexibility with fewer components.

The number and composition of the hardware pools are specified to address different test configuration needs and are implemented in a manner that allows them to be switched when and where needed. Pools, in their simplest representation, are sections of a network (actually two networks, because both the primary and the backup portions exist) containing a fixed number of processors. Switching occurs at the network level, whereby a processor pool can be moved from the networks in one particular test configuration to the networks in a different test configuration. To keep the switch costs affordable, the number of switch points for each pool has been limited to three, permitting each pool to have three possible network destinations among the candidate test configurations.

Maintenance of AAS systems deployed from the SSCC to the field sites relies heavily on the centralized resources of the SSCC. In this respect, the AAS follows the lead of the commercial world. Specialized skills, comprehensive software development capabilities, and the extensive use of vendor-supplied software products drive the AAS and the SSCC to a centralized implementation for system maintenance. Specialists at the operational sites interface to the SSCC via telephone communications and also use electronic networking and interactive sessions between the SSCC and the field sites. Data gathering for detected system problems is performed by site-resident personnel, whereas actual problem resolution and fix distribution are SSCC tasks. Electronic browsing of site-collected data files by SSCC experts contributes to the effectiveness of remote problem determination activities. Maintenance of vendor-supplied software products is coordinated through the SSCC. Product version upgrades are received at the SSCC and are then converted into load file images for installation and use by the site specialists.

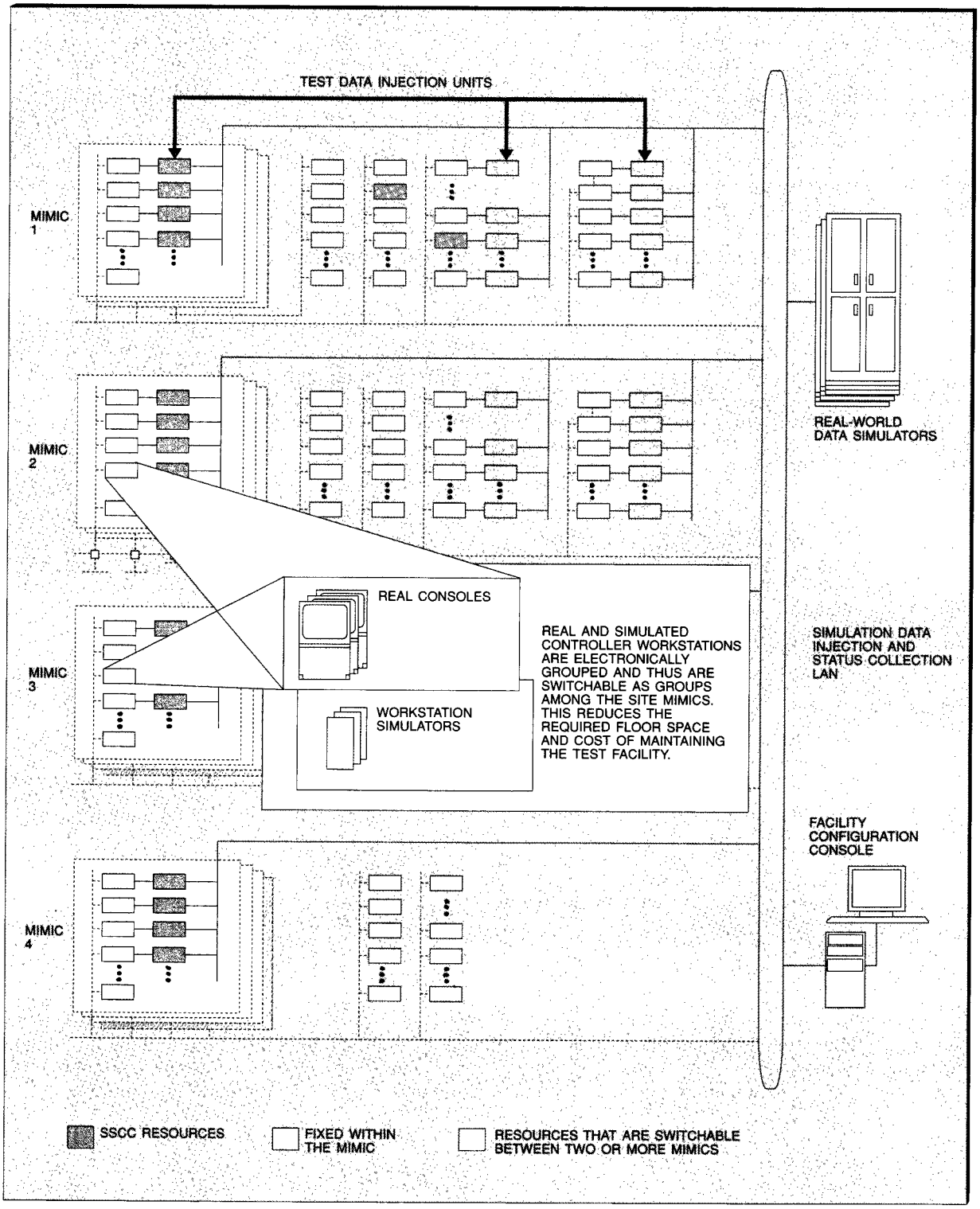
AAS software is targeted to execute in many field locations, at various system release levels, and with the appropriate adaptation data products of the site. This widespread variation is compounded by the quantity of both AAS-developed and vendor-supplied software (with common, unique, and function-specific classes of software). The need for an effective and thorough set of configuration management (CM) controls is mandatory. The SSCC uses the Software Configuration and Library Management (SCLM) product for mainframe-targeted software, and the Configuration Management Version Control (CMVC) product for RISC System/6000-targeted software. Together with specialized tools and procedures, these products provide the essential ingredients for the CM solution. All logic and data modules created during software development, used for system testing, and extracted for system release generation reside in SSCC databases under formal CM authority.

Summary

The AAS has demonstrated its ability to meet the stringent performance and reliability requirements in configurations with as many as 230 nodes under a variety of failure scenarios. The same architecture handles systems with as few as six processors. Throughout the protracted development phase, during which thousands of individual requirements have changed, the basic architecture has remained constant.

In this paper, we have discussed the system architectural considerations involved in building and testing a particular system (an air traffic control system) that must be responsive, highly available, and extensible. Commercial technologies were used wherever possible to allow for technology insertion over the life of the system (which is one form of extensibility). A distributed architecture allowed for horizontal growth and redundancy. Several unique structures were developed, making use of that distributed architecture, to accommodate the commercial components

Figure 6 SSCC architecture



into a system that exhibits state-of-the-art continuous availability and is extensible both in current size and future function.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of X/Open Co. Ltd. or U.S. Department of Defense.

Cited references and notes

1. In this paper, "controllers" will be used to refer to air traffic controllers.
2. The Kalman filter is the minimum mean squared linear estimator.
3. A. Avizienis and D. Ball, "On the Achievement of a Highly Dependable and Fault-Tolerant Air Traffic Control System," *Computer* **20,** No. 2, 84–89 (February 1987).
4. R. W. Butler and G. B. Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," *IEEE Transactions on Software Engineering* **19,** No. 1, 3–12 (January 1993).
5. F. Cristian, R. D. Dancey, and J. D. Dehn, "Fault Tolerance in the Advanced Automation System," *20th International Conference on Fault-Tolerant Computing* (June 1990), pp. 6–17.
6. The ISO (International Organization for Standardization) seven-layer stack is used for the primary network. TCP/IP (Transmission Control Protocol/Internet Protocol) is used for the backup network.
7. The choice of Ada was not strictly dictated by the FAA. However, when design began (in the mid-1980s), Ada was the only language that satisfied their language requirements. Many features of Ada (separate package specifications, standard language definitions across all platforms, exception handling semantics) make it appropriate for a large, fault-tolerant system. Even with the current emphasis on object-oriented languages such as C++, these Ada features may still make it the proper language for such highly available systems.

Accepted for publication August 5, 1994.

Drew Debelack *Loral Federal Systems, 9221 Corporate Boulevard, Rockville, Maryland 20850 (electronic mail: debelack@lfs.loral.com).* Mr. Debelack is a staff engineer, scientist in the Advanced Automation System Algorithm Design Department. He received his B.S. degree in physics and B.A. degree in mathematics at the University of Texas at Austin in 1987. He joined the IBM Federal Systems Company in 1987 and transferred to Loral with its acquisition of Federal Systems in 1994.

Jonathan D. Dehn *Loral Federal Systems, 9221 Corporate Boulevard, Rockville, Maryland 20850 (electronic mail: dehn@lfs.loral.com).* Mr. Dehn is a senior technical staff member and Certified Senior I/T Architect for Loral's FAA Air Traffic Control Systems. He received his ScB and ScM degrees in computer science from Brown University in 1975 and 1977, respectively. He joined the IBM Federal Systems Company in 1977 and transferred to Loral in 1994 when it acquired Federal Systems. For the last 10 years, he has concentrated on fault tolerance and performance aspects of the AAS. Mr. Dehn is a member of the ACM and the IEEE.

Lawrence L. Muchinsky *Loral Federal Systems, 9221 Corporate Boulevard, Rockville, Maryland 20850.* Mr. Muchinsky is a senior systems engineer on the Advanced Automation System project. He is the chief engineer for the System Support Computer Complex. He joined IBM in 1965 as an associate programmer, and has been involved in design and implementation activities for several large real-time systems, including the Goddard Real Time System, the National Airspace System 9020 project, and the Ground Based System for Shuttle. He transferred to Loral with its acquisition of Federal Systems in 1994. Mr. Muchinsky received a B.S. degree in mathematics from St. Francis College of Loretto, Pennsylvania in 1962.

Donald M. Smith *Loral Federal Systems, 9221 Corporate Boulevard, Rockville, Maryland 20850 (electronic mail: donald@lfs.loral.com).* Mr. Smith is a senior technical staff member, and Certified Senior I/T Architect for Loral's FAA and Worldwide Air Traffic Control Systems. He received his B.S. degree in electrical engineering from the University of Washington in 1965. He joined the IBM Federal Systems Company in 1965 and transferred to Loral with the acquisition of Federal Systems in 1994. Mr. Smith has a background in software architectures for high-availability man-rated systems.

Reprint Order No. G321-5558.