MITRE's Evolvable Real-Time C3 (Command, Control, and Communications) project has developed an approach that would enable current real-time systems to evolve into the systems of the future. The major goals of the initiative include determining the software infrastructure requirements and to identify the migration path for legacy systems. The infrastructure is a collection of all non-application specific software services. This infrastructure provides the software backplane for applications and insulates application software from hardware. Ideally, we want to use COTS products for the infrastructure. The services provided by the infrastructure include operating systems services such as memory management and scheduling, communication services such as interprocess communication, and data management services such as data sharing, querying, updating, transaction management, and enforcing integrity constraints. The infrastructure also provides the mechanisms for interaction between the software components. All of the services must provide an integrated priority scheme and performance predictability.

The project has chosen to focus on the Surveillance function as the starting point for transitioning AWACS to an open architecture. Ideally, all of the application components should be hosted on the infrastructure. The application components for a system such as AWACS will include display, weapons, surveillance and tracking, and communication. Implementing such a system will mean re-architecting the entire AWACS mission computing system. This is not feasible within current budget constraints. Therefore, our approach is to extract certain application subsystems and host them on the infrastructure while the other subsystems remain within the legacy environment. We have hosted part of the surveillance and tracking application on the infrastructure and the rest of the system runs on the legacy environment.
We have designed and developed an infrastructure, data manager, and tracking application. Infrastructure is that component that includes operating system services as well as other services such as interprocess communication and scheduling. Data manager is responsible for managing the data including the tracks. The application we have used to host on the infrastructure is the multi-sensor integration application. Since an evolvable design was a major consideration, we were influenced by object-oriented design and implementation approaches as well as our initial investigation of real-time issues for distributed object management systems. Our infrastructure is essentially a collection of objects interacting with each other. Existing applications as well as new applications can be encapsulated as objects. Our implementation environment is the Lynx operating system environment and our implementation language is C++. We have used a distributed object management system to integrate the various components.

It is critical that the our evolvable real-time command and control system be survivable. That is, the system must handle the timing constraints and degrade gracefully in the presence of faults and intrusions. Therefore, one of the important directions for our project is to investigate issues on survivability. We discuss some of our preliminary ideas toward this direction. In order to develop a survivable system, the real-time requirements of the system must be integrated with the fault tolerance and security requirements. To meet the needs of survivability, we have identified requirements for the Real-time Infrastructure (RTIS) in the following areas: real-time scheduling and constraint enforcement, transaction management, scheduler encapsulation, real-time predictability, admission control, overload management, real-time tasks, real-time threads, interthread synchronization and mutual exclusion, interprocess/interthread communication, fault-tolerance and group communication, portability, and security. We discuss some of the requirements closely related to survivability aspects below.

Real-time Scheduling and Constraint-Enforcement: The RTIS should provide an application with real-time scheduling support services. The services will include the assignment of operating system scheduling parameters on the basis of the application real-time workload (processes and threads with real-time constraints). Current real-time operating systems do not provide services to support the determination of scheduling priorities, on the basis of workload and timing-constraints. That task is left to the implementor. By applying existing scheduling theory, we can provide an RTIS service that will calculate scheduling parameters (priority) on the basis of workload and timing-constraints. The RTIS should also provide time-constraint (real-time and cpu-time) enforcement services. An application will be able to request notification of time-constraint violations (e.g., deadline faults) and then elect to specify fault recovery processing which may include application routines.
Real-Time Predictability: The RTIS should support the implementation of "predictable" real-time applications. The principal design issue is what do we mean by "predictability." Our interpretation here is that we will only use real-time scheduling algorithms in the RTIS for which tractable theories exist: simple schedulability models as in rate-monotonic scheduling, deadline-monotonic scheduling, or earliest-deadline-first; or schedulers that empirically have acceptable behavior such as best-effort scheduling. Standard "feedback-queue" schedulers used in most mainframe time-sharing and Unix systems, emphasizing fairness and deterministic timing behavior, are not acceptable and will not be used. The RTIS will only support a very limited subset of scheduling techniques (rate-monotonic and deadline-monotonic scheduling) since static priority-based preemptive scheduling is the only "predictable" technique that is universally supported in COTS real-time operating systems today.

Admission Control: The RTIS should support the optional use of admission control. When admission control is enabled, all requests for the creation of threads with real-time and cpu-time constraints are checked for admissibility and are created only if the new workload (old workload plus new thread) is schedulable. This requirement is imposed because in hard real-time applications, timing-faults lead to system failure. Admission control enables the application to monitor the workload through the RTIS. Additional units of work (threads) will be rejected when they are created instead of when they fail or trigger other failures. This capability is not needed for all real-time applications, only those that have system-critical timing-constraints. The application will enable or disable admission control as part of its initial configuration. We have made this decision in order to simplify the design of the RTIS and see no need to dynamically enable or disable this capability. When enabled, it will permit admission control to restrict the creation of real-time threads to those that constitute "analyzable" architectures for the underlying real-time operating system. This means that an application that uses this capability will not necessarily be portable to other platforms that support other (different) analyzable real-time scheduling algorithms.

Overload Management: The RTIS should support the "graceful" degradation of an application under overload. An RTIS application gracefully degrades when the real-time constraints of an application are missed in order of criticality. When a workload is not schedulable, an overload condition is said to exist. The application will enable or disable overload management when it is initialized. An issue that we have not resolved, satisfactorily, is how to specify the "criticality" of a thread. We will initially simply permit the application to specify a ranking with an integer. Another issue is that overload management is only well understood for a few real-time scheduling techniques: rate-monotonic, deadline-monotonic, and best-effort scheduling.
The RTIS will only support overload management for these scheduling techniques.

Fault-Tolerance: The RTIS should support fault-tolerance by providing group communication services (reliable multicast, atomic multicast, causal multicast, and group membership services). We are borrowing the ISIS model of distributed computation here. Experience with ISIS has shown that this is a powerful approach both for fault-tolerance and distributed computing outside of the normal point-to-point communication model. This supports fault-tolerance by permitting replication of data, real-time processes, and threads. The RTIS will provide support for thread groups: the creation and naming of a thread group, the destruction of a thread group, and the addition and removal of a thread from a thread group. The name of a thread group will be associated with a thread group address. A distributed name service will permit any thread to find the address of a thread group by asking the name service for its address. When a thread group is created, it will be registered with its specified name with the thread group name service. When a thread group is destroyed, it will be deregistered from the thread group name service. The following (ISIS) group communication services will be supported: reliable multicast, causal multicast, atomic multicast. The associated orderings in message delivery will be enforced with respect to changes in group membership.

Security: The RTIS should ensure authorized access as well as authorized modifications to the data. In addition, the system will survive in the present of intrusions and other security violations. This requirement essentially states that the system must be secure and be able to survive in the midst of intrusions. The difficulty with security is that it is often an afterthought. As a result, it will be difficult to meet security requirements with other requirements such as real-time and fault tolerance if security is not considered early on in the design. For example, it has been shown that security and real-time constraints are conflicting goals. However, there are various ways to develop adaptable policies if security and real-time constraints are considered early in the design. Another problem is that a malicious process can manipulate the resources in such a way that the real-time constraints are not met. Appropriate techniques to prevent this from happening are needed.

It is critical that the requirements work well together. For example, often security and real-time processing are conflicting goals. Therefore, one needs to determine adaptable security policies so that the real-time constraints are met and at the same time security is maintained. Therefore, integration of the various requirements discussed here is important. Furthermore, graceful degradation of the system in the presence of faults, malicious processes,
and intrusions is essential. Therefore, research is needed on integrating fault
tolerance, security, and real-time processing so that evolvable, survivable,
adaptable C3 systems can be developed for real-time applications.