1. OVERVIEW

Between now and the early part of the next century, significant portions of today's real-time C3 systems will become either functionally inadequate or logistically insupportable. Such systems need to change as new requirements are imposed in any of the many component systems. System changes can result from new external functional requirements or changes to operational or maintenance processes. It is also desirable for the systems to take advantage of the technological developments in hardware and software.

A real-time C3 system should not only carry out the functions of a command and control system such as tracking and surveillance, weapons control, and display, it must also meet the timing constraints imposed. Current real-time C3 systems include Air Force's AWACS, Joint services' JOINT STARS, and Navy's BSY-2.

Due to the continuing budget reductions, new developments of next generation real-time C3 systems may not be possible. Therefore, current real-time C3 systems need to become easier, faster, and less costly to upgrade in capability and easier to support. What is needed is an approach to evolve current real-time C3 systems into the extensible systems required for the future.

MITRE's Evolvable Real-Time C3 project funded under the Air Force Mission Oriented Investigation and Experimentation program attempts to develop an approach that would enable current real-time systems to evolve into the systems of the future. In order to provide a plan for the evolution, an understanding of current systems is essential. Therefore, our initial effort was to study the requirements of existing systems. Based on an investigation of the state-of-the-art in hardware and software technologies, we provided a plan for future systems. Section 2 describes the approach we have taken to evolve the current systems to take advantage of developments in hardware and software. Future directions to developing adaptable real-time object request brokers for next generation C3 systems will be discussed in section 3.
MITRE’s Evolvable Real-Time C3 project has designed and implemented an approach that would enable current real-time systems to evolve into the systems of the future. The candidate evolution approach is to leverage off near-term system upgrade and/or Pre Processing Product Improvement (P3I) activity to put a new architecture framework in place. The emphasis is on transitioning to open architectures which are modular and free from proprietary or unnecessarily complex software designs. The open framework can also accommodate new upgrades more easily. Availability of a suitable software architecture is key for this approach to succeed. The investment plan would continue incremental transition of current systems into more flexible systems. The extensible system architecture would ultimately replace the current hardware and software architecture.

In order to provide an evolution path for real-time C3 systems, one needs to understand the requirements of current real-time C3 systems and the approach taken to design such systems. The project has chosen AWACS as an example to incrementally test out the concepts and architectures to be developed. It has a centralized database which is based on a closed architecture with monolithic custom software. It does not have a state-of-the-art hardware architecture. Processing upgrades to the system is time-consuming and expensive.

The project has chosen to focus on the Surveillance function as the starting point for transitioning AWACS to an open architecture. This is because the target tracking algorithm within the Surveillance function is a prime candidate for system improvement due to recent advances in multi-sensor integration (MSI) trackers. Since the required computational performance has only recently become available in new processors, an MSI tracker would need to execute outside the existing mission computer. Also, many improvements to the surveillance operator interface are possible only with the addition of an MSI tracker. The technical challenge is to identify open software technology applicable to AWACS and other real-time C3 systems that supports this type of function migration out of the existing hardware and software architectures. The successful execution of this project would substantially reduce the risk of transition to an open system of legacy functions.

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.

In the target C3 system, 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. The infrastructure provides the means for the application subsystems to access and share the database as well as to communicate with each other. 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. In our intermediate architecture MSI processing within the surveillance subsystem is hosted on the new infrastructure.

In addition to the infrastructure, we also implemented a data manager for real-time transactions and queries. Object-oriented approach was utilized both for the infrastructure and data manager. We utilized an object request (ORB) to integrate the infrastructure, data manager, and MSI application subsystem. While the ORB used was not a real-time ORB, we expect to replace it by one when such an ORB is available. In the meantime we are giving inputs to the OMG Real-time SIG to incorporate the services provided by the infrastructure including the scheduling service and interprocess communication service into various documents.

3. CURRENT FOCUS

Our phase 1 of the project was carried out over a period of three years. During the first year of the initiative, we have analyzed C3 system requirements with special emphasis on AWACS, prototyped MSI applications component, developed functional models of the system for hypothesized architectures, deduced the infrastructure requirements, and assessed the research and industry trends in real-time systems. During the second year, we acquired selected products and technologies and extended and enhanced them to meet C3 real-time requirements. We also integrated the components using an ORB to provide a real-time embedded infrastructure. The selected functional model was validated using the MSI application module and subsequently the infrastructure architecture was evaluated. During the third year, the MSI application was hosted on the integrated infrastructure and compared architecture performance to system performance requirements. We also established the feasibility of the transition approach.

During the second phase of the project we are extending ORBs to be adaptable for real-time systems with a special emphasis on quality of service. The goal of this project was to identify the desired features of an open ORB, create a design and to implement a prototype in coordination with the AWACS effort. This implementation is tailorable to demanding real-time and fault-tolerance requirements through the use of metaobject protocols and reflective techniques.

The components of the system are the ORB, services such as scheduling and data management and the application frameworks. Each component consists of various subcomponents and protocols. The idea is to switch the subcomponents and protocols without disrupting the operation of the system. For example, one could switch the scheduling protocols or the transport protocols or even the application components. New versions of a component can be derived such that new functionality, such as performance and system monitoring, can be added. The fact that these modifications can be made relatively easily represents a tremendous economic value.

The effort that we have described has been influenced by the work on metaobject protocols carried out at Xerox Palo Alto Research Center. More recently, there is also work on adaptable ORBs at Washington University in St. Louis. We have combined the work described in these efforts together with our previous work on evolvable real-time command and control systems.
Some of the work carried out at Lockheed Martin on integrating ORBs with ODBMSs have also impacted our work. We have an initial demonstration system intended to convey technical methods used and at least hint at future applications for those methods, but does not require previous familiarity with those methods to understand the demonstration. The demonstration was run on networked 386 machines. The operating system used was LynxOS 2.4, a real-time version of UNIX. The programs were written in Python, a dynamic rapid-prototyping interactive language, with support for scripting, pattern matching, object-oriented programming, and many other "specialty" areas. The GUI was implemented using Tkinter, a Python adaptation of Tk, a widget set initially developed to run with Tcl. We used SYLU, a Python implementation of Xerox's ILU developed by Scott Hassan at Stanford University, for our ORB. SYLU is the only CORBA ORB on the market, written in a dynamic language, such as Python, with complete source code available. We first ported Python to Lynx OS. Then SYLU was built on the Python virtual machine on top of Lynx OS. While Python is an object oriented language, SYLU did not use all of its capabilities. SYLU's object structure was predominantly flat, using almost no inheritance. The only exception is the Transport/Protocol layer which contains a superclass for both the transport and protocol, making it possible to derive different protocols and transports for a particular implementation.

The project demonstrates dynamic server adjustment to match a particular client's needs. As an example, a server running in RPC mode would get a signal to change its mode to HTTP so it may service a different, (higher priority) client. To the client, this change would be seamless. The project demonstrates this concept with the exception that the server did not dynamically change its mode. If a new client required a different mode, the server was shut down once it became idle. The server was then restarted in the appropriate mode. Although shutting the server down and restarting did not demonstrate the dynamic aspect of the concept, the change was transparent to the client. In our design of an adaptable real-time ORB we chose to use the Meta Object Protocol approach. The preliminary design consists of the ORB baseobject and metaobject specification, their interaction and the interfaces they expose to the users and applications.

Our model consists of objects in the ORB, distributed objects (CORBA objects) and CORBA Services. We classify these objects into three categories: the distributed objects are the baseobjects (see Meta Object Protocol terminology). All the objects in the implementation of the ORB, services and facilities are metaobjects of metalevel 1. Objects in the ORB and distributed objects that control the execution of baselevel objects and metalevel 1 objects (through a meta interface) are metaobjects of metalevel 2. Consider, for example, our earlier example with Protocol substitution in a CORBA ORB. If Protocol is an object inside the ORB, it is a metalevel 1 object according to the definition above. A metalevel 2 object for the Protocol could be an object that controls which instance of Protocol is currently being active - e.g., Sun RPC or HTTP. Another example could be the following: an object inside the Concurrency Control service is a metalevel 1 object. An object that can control which CC algorithm is used in a particular situation (e.g., priority ceiling, two-phase locking, etc.) would be a metalevel 2 object.

Our future plans are to examine various features like quality of service, fault tolerance, data management, transaction processing for real-time applications within the framework we have created for adaptable ORBs.