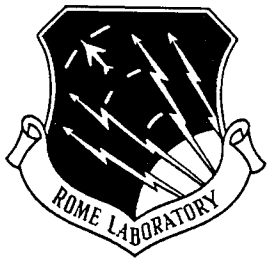
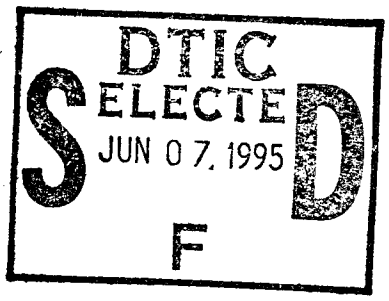
RL-TR-95-33
In-House Report
March 1995



A CLIENT-SERVER APPROACH TO DCE INTER-OPERABILITY THE CRONUS/ISIS PROJECT (CRISIS)

Patrick M. Hurley and Terrance A. Stedman



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

19950605 033

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-95-33 has been reviewed and is approved for publication.

APPROVED:

ANTHONY F. SNYDER, Chief
C2 Systems Division
Command, Control and Communications Directorate

FOR THE COMMANDER:

HENRY J. BUSH
Deputy for Advanced Programs
Command, Control and Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3AB) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1995	In-House

4. TITLE AND SUBTITLE
A CLIENT-SERVER APPROACH TO DCE INTER-OPERABILITY
THE CRONUS/ISIS PROJECT (CRISIS)

5. FUNDING NUMBERS
PE - 62702F
PR - 5581
TA - 28
WU - 25

6. AUTHOR(S)
Patrick M. Hurley and Terrance A. Stedman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3AB)
525 Brooks Road
Griffiss AFB NY 13441-4505

8. PERFORMING ORGANIZATION REPORT NUMBER
RL-TR-95-33

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3AB)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Patrick M. Hurley/C3AB (315) 330-3623

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)
Distributed computing environments (DCEs) support several key attributes that are essential for the development and execution of Command and Control (C2) applications. These attributes include heterogeneity, resource availability, concurrent programming, fault tolerance, and resource management. Many DCEs claim to support all or some of these key attributes. In reality, however, no one DCE can easily and efficiently address all the needs of every application. The needs of large applications that encompass a diverse set of requirements, such as C2 applications, are especially difficult to satisfy with any one DCE. The reason for this is that many DCEs are designed using vastly different design methodologies that make them better suited for different types of problems. The Cronus/ISIS (CRISIS) project is an attempt to combine attributes from multiple DCEs to allow the application designer more freedom when designing complex applications. Specifically, the objective of the CRISIS project is to design and demonstrate the inter-operability of two structurally different DCEs. This was accomplished by building an application that spans two mature yet different DCEs (Cronus and Isis). The rationale for doing this was to utilize the strengths of each distributed computing environment, where a certain DCEs design methodology is better suited for that part of the application.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Distributed Computing Environments, Distributed Systems, ISIS, CRONUS, Distributed Operating System	32
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	U/L

TABLE OF CONTENTS

1.0 Introduction

Distributed computing environments (DCEs) support several key attributes that are essential for the development and execution of Command and Control (C2) applications. These attributes include heterogeneity, resource availability, concurrent programming, fault tolerance, and resource management. Many DCEs claim to support all or some of these key attributes. In reality, however, no one DCE can easily and efficiently address all the needs of every application. The needs of large applications that encompass a diverse set of requirements, such as C2 applications, are especially difficult to satisfy with any one DCE. The reason for this is that many DCEs are designed using vastly different design methodologies that make them better suited for different types of problems.

The Cronus/Isis (CRISIS) project is an attempt to combine attributes from multiple DCEs to allow the application designer more freedom when designing complex applications. Specifically, the objective of the CRISIS project is to design and demonstrate the inter-operability of two structurally different DCEs. This was accomplished by building an application that spans two mature yet different DCEs (Cronus and Isis). The rationale for doing this was to utilize the strengths of each distributed computing environment, where a certain DCE's design methodology is better suited for that part of the application.

This paper is organized as follows: In section 2 a Cronus overview is presented. An overview of the Isis distributed computing environment is presented in section 3. Section 4 compares several mechanisms provided by Cronus and Isis. The CRISIS architecture and two examples are discussed in section 5. Section 6 summarizes the CRISIS work as well as how it benefits current and future technologies.

2.0 Cronus Overview

Cronus is a distributed computing environment that supports heterogeneous computer systems interconnected on a high-speed local area network (LAN) or wide area network (WAN) [Berets93]. Cronus was developed by BBN Systems and Technologies Incorporated to support distributed command and control applications, under the sponsorship of the U.S. Air Force (Rome Laboratory). The present version of Cronus provides support for such diverse systems as Sun workstations running Sun UNIX, DEC machines running VMS or ULTRIX, HP machines running HP-UX, and several parallel machine architectures. Cronus currently supports applications written in the following languages: C, C++, FORTRAN, and Common Lisp.

The Cronus distributed computing environment is based on the *object model.* An object consists of state information maintained in an object database and a collection of rules that govern how this state information may be examined or changed. Each rule represents an operation on the object. Objects and their associated operations are managed by object managers. Operations on objects are invoked by client programs or by other object managers.

Cronus object types define how the objects are to be used and implemented. Types are made up of operation code, operation interfaces, and data structures that specify the representation of the various objects. Types are also placed in a hierarchy structure that allows new types to be created as subtypes of existing ones.

Cronus consists of services, clients, and the Cronus kernel. Services (a service consists of one or more object managers) implement both system and application functions. Current system services provided by Cronus include an authentication service, a symbolic

naming service (global), a network configuration service, a directory service, and an object type definition service. Clients within Cronus are processes that use services. The Cronus kernel itself resides above the native operating system and is primarily responsible for transmitting synchronous or asynchronous operation invocations from clients to services. The Cronus kernel makes the network appear transparent. For example, a client on one host in a given Cronus configuration can invoke an operation on an object type whose manager resides on another host in the network (Figure 1).

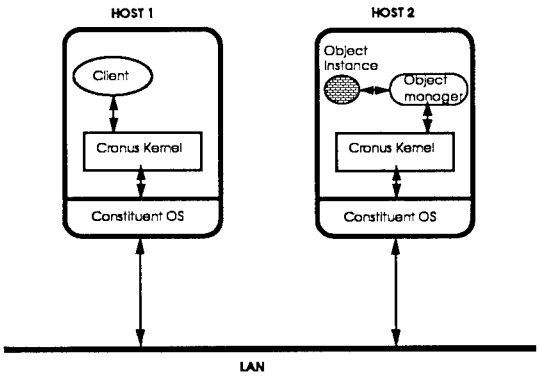


Figure 1. Cronus Communication.

By using an object-oriented approach, Cronus is able to provide a variety of high level abstractions such as: (1) Management of replicated data (on disk); (2) Parallelism, by splitting a computation among several machines; (3) Monitoring and reporting the status of a computation; (4) Dynamic reconfiguration from failures; (5) Support for multi-domained applications; and (6) Support for multi-clustered networks.

3.0 Isis Overview

Isis is a distributed computing environment that provides high level tools which support the development of fast reliable distributed applications. Isis provides mechanisms that support heterogeneous computer systems interconnected on a high-speed local area network (LAN) or wide area network (WAN). Isis was sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and developed at Cornell University, Ithaca, New York. It is currently available commercially through Isis Distributed Systems, Inc.

The Isis distributed computing environment is based on the concept of *process groups* [Birman91][Cooper92]. Process groups are a lightweight programming construct. A single process can belong to any number of groups and there is minimal overhead associated with joining and leaving groups. Groups have a hierarchical namespace, much like a file system namespace, and permit flexible, location transparent addressing. Process groups are also capable of spanning across multiple machines. Isis provides mechanisms for communicating atomically with a group, as one might do to inform its members of some event, or to issue them a request of some sort. Such a communication takes the form of a *multicast*, in which one or all the members of the group receive the message, and zero or more respond (depending on the needs of the particular application) [Birman89].

An important strength of the Isis environment lies in its support for implementing *virtual synchrony* [Birman87]. Virtual synchrony permits the programmer to design a distributed program for execution in a simplified environment, wherein all processes observe events simultaneously and therefore in the same order. Events such as communication with a group or detection of failures are atomic in a virtual synchronous setting: all group members receive a message (or observe failures) if any does [Birman89].

Using process groups and virtual synchrony, Isis is able to provide a variety of high level abstractions such as: (1) Management of replicated data (in memory or on disk); (2) Parallelism by splitting a computation among several machines; (3) Coordination of an external action; (4) Synchronization of concurrent actions (such as when several

processes share a resource that only one can use at a time); (5) Monitoring the status of a computation, process or computer, and triggering user-programmed defined actions should the status change; and (6) Dynamic reconfiguration from failures to include the integration of a recovered machine into the operational system, restarting of the services that should run at that location and the ability to bring the services up-to-date concerning the active state of the system [Birman89].

The Isis distributed computing environment currently supports applications written in C, C++, FORTRAN, ADA and Common Lisp. Isis runs on (and between) SUN, DEC, HP, GOULD, NEXT, and APOLLO equipment, and currently requires the UNIX or MACH operating system. However, ports to other operating systems such as AIX and VMS are being considered.

4.0 Cronus and Isis Mechanisms

Cronus and Isis provide support for several key distributed computing attributes. These attributes include distributed processing, fault tolerance, real time, limited adaptive resource management, heterogeneity (to include languages, operating systems and machine architectures) and support for multi-domained applications. In the paragraphs that follow each of these characteristics will be described further, and the extent to which Cronus or Isis provide inherent support for these capabilities will be compared.

4.1 Distributed Processing

Distributed processing may be defined as concurrent processes cooperating towards common goals where each process is likely to have incomplete and/or inconsistent global state information. Distributed processing provides many desirable attributes to include: *Redundant Processing* - several processes performing the same task perhaps on different data; *Concurrent Processing* - work is divided among several processes, each of which will contribute to the final result; *Resource Sharing* - sharing data, memory, CPUs, disks, and other physical devices for a common goal, or because a host is deficient in the resources it requires; *Reliable Communication* - reliable communication between processes whether they are on the same host or on different hosts connected by LANs and WANs; and *Heterogeneity* - the inter-operability of languages, operating systems and machine architectures.

Cronus and Isis both support the attributes of distributed processing to some degree. However, different characteristics are evident due to the vastly different design methodologies of Cronus and Isis. For example, both systems provide reliable communications between processes but a characteristic of Isis is to also preserve message order. Therefore, if your application needs reliable ordered communications, Isis should be used because Cronus provides no inherent mechanisms to preserve order.

4.2 Fault Tolerance

Fault tolerance may be defined as the ability of a system or component to perform its function, despite the presence of hardware or software faults. Fault tolerant mechanisms must also be able to detect and/or recover from hardware and software faults. Faults fall into three classes hardware, software, and communication. Two examples of hardware faults are: (1) the complete loss of a host/service; and (2) degradation in the performance of a host/service due to overload conditions. Software faults include the following types of faults: (1) algorithmic faults; (2) software component faults (service loss); and (3) timing faults. Finally, two examples of communication faults are: (1) loss of connectivity (local area network or wide area network); and (2) overloaded or congested communications.

Cronus and Isis both provide and/or support a limited number of fault tolerant mechanisms. *Replication* - Replication is defined as multiple copies of a resource maintained on different hosts to improve availability or survivability; *Triple Modular Redundancy (TMR)* - TMR is defined as three processors running the same code on

the same data and voting on the result to mask out an error by any one processor; and *N-Version Programming* - N-Version Programming is defined as N versions of a program (using different algorithms) operating on the same data and voting on the results. The only one of these fault tolerant mechanisms that is directly supported by Cronus and Isis is replication. The other mechanisms, however, can be supported but are more application specific.

Both Cronus and Isis directly support replication. The replication mechanisms within Isis are more flexible and dynamic than those found within Cronus. For example, Isis requires no pre-defined and pre-compiled read or write quorum as does Cronus. However, Cronus replication mechanisms are better able to maintain consistency during and after a network partition. This is because Cronus does have pre-defined read and write quorums. If a distributed application becomes partitioned due to a network failure, Cronus will allow the partition containing a majority of the application copies (i.e., managers) to continue execution pending a recovery from the network failure. Upon recovery of the network, Cronus provides guarantees to insure all copies of the application are brought up to the current (consistent) state. The choice to use Cronus or Isis once again depends on the needs of the particular application.

4.3 Real Time

"Real time may be defined as a system or mode of operation in which computation is performed during the actual time that an external process occurs, so that the computation results can be used to control, monitor or respond in a timely manner to the external process" [IEEE90]. Using this definition, a system with real time response capability would be able to incorporate scheduling and resource decisions based on the current conditions of both system and environment. At present, neither Cronus nor Isis has the ability to comprehensively support such real time computations. However, if only part of an application has a real time requirement then an integration similar to the CRISIS integration, but with real time system support, would address this issue. This shows the strength of the CRISIS environment, i.e. an application programmer is not stuck with any one technology but can exploit different technologies to solve his problems.

4.4 Resource Management

Adaptive resource management may be defined as the ability of a system to change its internal state to be consistent with its external environment. Adaptive resource management should allow for both the static mapping of application components within the system as well as the ability to dynamically react to changes in the runtime requirements that could not possibly be anticipated. Therefore, a base for adaptive resource management should be supported to improve throughput and provide some level of fault avoidance. This base should also support both static and dynamic migration of processes and data.

Adaptive resource management mechanisms can be used to automatically adapt to evolving resource availability. Therefore, in the event of failures, the system may automatically instantiate or migrate application functionality within and among the surviving system resources. Isis provides some support for adaptive resource management that includes "recycling" idle workstations for remote use and managing a pool of "compute servers". It can also manage a reliable replicated service by ensuring that some desired number of copies of the service are always running, despite machine failures. Cronus, on the other hand, provides no such mechanisms [Isis92].

4.5 Multi-domained Support

Support for multi-domained applications is also desirable. This would permit controlled access to resources (e.g., computers and data) in a flexible and efficient manner. Cronus supports multi-domained applications by a mechanism called *clusters*. A cluster is a set of hosts grouped together into a single administrative unit. Each cluster is autonomous, and therefore responsible for its own administration and

control. No host is permitted to be a member of more than one cluster. Clusters allow boundaries to be erected between organizations, but can selectively allow or deny foreign clusters access to services that they support. This feature allows an administrator to selectively choose the services that will be exported from one Cronus cluster to another, thereby limiting the remote computing/communications burden. Isis process groups, by default, provide a much less sophisticated mechanism (compared to Cronus) to support multi-domained applications. Process groups differ from clusters in that a member of one group can join any number of other groups. Process groups also have very little access control for joining a group, and once they become a member they can gain access to all the group services.

5.0 The CRISIS Architecture

The design goals of the CRISIS architecture are very straightforward. CRISIS must demonstrate the inter-operability of two structurally different DCEs (Cronus and Isis) in a form that is easy to use. In doing so, it must not inhibit the functionality of either DCE.

In order to accomplish this goal, CRISIS capitalized on the fact that both DCEs selected support the *client-server* model. In this model, clients make invocations to a population of servers which can be resident on several nodes in the system. The servers are passive entities waiting to service an invocation from a client. Therefore, by using this client-server model, an Isis client can be embedded inside a Cronus application to allow an Isis server to be called from Cronus. The reverse is also possible. A Cronus client can be embedded in an Isis application to allow a Cronus server to be called from Isis.

This is a fairly simple approach to the integration of two complex systems. However, this form of integration preserves the integrity of both DCEs and is fairly simple to use, thus meeting the design goals. A description of the testbed and two examples follow to illustrate the use of such a system.

5.1 The Rome Laboratory Testbed

Rome Laboratory currently has an in-house distributed computer systems research, development and evaluation laboratory called the *Distributed Systems Environment* (DISE). Among the capabilities available within the DISE is the ability to demonstrate key distributed systems attributes utilizing various DCEs e.g. Cronus [Berets93], Isis [Birman91], and OSF/DCE [Johnson91].

The hardware configuration used consisted of three Sun workstations (SPARCstation 20's) connected on an ethernet LAN. The operating system used was Solaris 1 (BSD v4.1.3). The DCEs used were Cronus v3.0 and Isis v3.0.8.

5.2 The Inventory Example

To test out the feasibility of the CRISIS architecture, a simple Cronus manager called the inventory manager was used. The inventory manager is a sample manager distributed with the Cronus software. It simulates an inventory control system which provides operations to create items, add and subtract from the quantity of an item, and delete items.

5.2.1 An Isis Client within a Cronus Manager

The first case is to embed Isis client code inside of a Cronus manager. Below is the canonical representation of the Cronus inventory item type.

```
cantype ITEM
  representation is Item: record
    Description: ASC
      annote" A string that describes the item.";
    Count:      U16I
      annote "The number of items in the inventory.";
end ITEM
```

```
    annote "This datatype stores all the information about
an item in the inventory.";
```

So, an inventory item is a record containing a description string (e.g. hammer) and an unsigned 16-bit integer representing the current quantity of the item in the inventory.

A typical operation on an inventory item is to add to the quantity of that item. This operation is defined in the inventory manager as a generic operation called "AddtoItem". This operation is defined in the inventory manager's typedef file in the manner shown below:

```
generic operation AddToItem(
    Description:    ASC;
    Add:            U16I;
  )
  returns(
    Count:          U16I;
    Description:    ASC;
  )
annote "This operation adds new stock to an inventory
item.";
```

So, operation AddToItem takes a description of an item and the number of these items to add to the database as input and outputs the new number of these items that are in stock and the description of the item as well. For example, if there are currently 5 hammers in the inventory and AddToItem("hammer",5) is invoked, the new count of hammers in stock would be 10.

The AddToItem operation was originally written in Cronus. It was implemented in Isis to demonstrate the capability of Cronus to communicate with an Isis server. To do this, the portion of the Cronus AddtoItem operation that actually performed the add was stripped out and replaced with Isis client code. This client code makes a call to an Isis server to perform the add. Isis then returns the description and the new count to Cronus.

In order to make these Isis calls from Cronus, several include files are required in the Cronus operations code, as below:

```
#include "isis.h"
#include "isis_errno.h"
```

These are the standard include files in Isis server code as well.

Below is a pseudocode version of the Cronus AddtoItem operation with the embedded Isis client code. The actual code is provided in Appendix A.1.

```
AddtoItem(...)
begin
  retrieve item from Cronus object database;
  join Isis process group;
  call Isis task invAddToStock(item_name, count, addamt);
  leave Isis process group;
  update item count with result returned from Isis;
  store the updated object in the Cronus object database
end
```

Lastly, here is pseudocode for the Isis AddToStock task as defined in the Isis server code. The actual code appears in Appendix A.2.

```
AddToStock(message)
begin
  get item_name, count, and addamt from message received
    from Cronus;
  add addamt to count;
  return a message to Cronus consisting of the item_name
    and the new count;
end
```

It should be readily apparent that the Isis AddToStock task simply adds an amount to a count and returns the new count and description. While this is a very trivial operation, it nevertheless demonstrates that a Cronus manager can issue a call to an Isis server and receive results back.

This example is executed by first starting the Cronus inventory manager and the Isis server. Using either a Cronus client or

TROPIC (TRansportable Operation Interface for Cronus), the AddToItem operation can be invoked. For example, entering "on {inv} AddToItem hammer 10" through tropic will invoke the Cronus AddToItem operation which in turn will call the Isis AddToStock task.

5.2.2 A Cronus Client within an Isis Server

Next comes the second case: embedding Cronus client code inside of an Isis server. Demonstrating this case was slightly more difficult because the Cronus inventory manager had to first be converted into Isis server code. After that was accomplished, the AddToStock task as defined in the Isis server code was modified to call a Cronus operation to perform the add. This is quite similar to what was done in the first case.

Below is pseudocode for the Isis AddToStock routine with the embedded Cronus client code. The actual code appears in Appendix A.3.

```
AddToStock(message)
begin
  get item from Isis store;
  invoke the Cronus operation AddToStock(item_name,
    count, addamt);
  update the inventory item in the Isis store with the new
    count;
end
```

In order for the Isis server to make Cronus calls, like the AddToStock call above, the following Cronus include files are required in the Isis server code file.

```
/* Cronus includes */
#include <stdio.h>
#include <cronus.h>
#include <getqual.h>
#include <futures.h>
#include "invatmgr.h"
```

Pseudocode for the Cronus AddToStock operation which is called by the Isis task AddToStock follows below. The actual operation code appears in Appendix A.4.

```
AddToStock(...)
begin
  add addamt to count;
  return new count and item_name to Isis;
end
```

Again, this Cronus operation merely adds a numeric quantity to an inventory item and returns the new quantity and description to Isis. Regardless of its simplicity, it demonstrates that not only can Cronus managers communicate with Isis servers, but Isis servers can talk to Cronus managers as well.

This example is executed by starting the Isis server and the Cronus manager. By issuing Isis calls of the form

```
cbcast(group_addr, ADD, "%s%d", "hammer", 25, 1,
                        "%d",answer);
```

the Isis AddToStock task is called which in turn calls the Cronus operation AddToStock to perform the add. The Cronus operation then returns the new quantity and description back to Isis.

Since communication is possible in both directions between Isis and Cronus, it seems likely that these two distinctly different DCEs can be used together to create a single distributed application. This distributed application could take advantage of the features unique to both Cronus and Isis, and thus enhance its capability.

5.3 The JDL Example

The inventory example above proved the inter-operability of Cronus and Isis for a simple example. To test this inter-operability on a more complicated example, the Target Filter (targfil) Manager component of the Joint Directors of Laboratories (JDL) experiment was selected.

The JDL experiment simulates a joint tracking and targeting system. Each service (Army, Navy, and Air Force) provides an application, which is integrated into the overall, distributed environment. Common software elements, user interfaces, data storage devices, and synchronization mechanisms are shared between the three services. The application is constructed of 10 components: synchronization (timing), target simulation, track simulation, track reporting, sensor simulations, target filtering, weather information, situation reporting, data management, and graphical interfaces (see Figure 2).

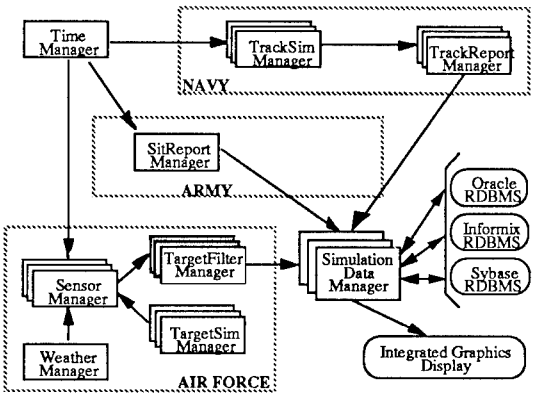


Figure 2. JDL Diagram

The Target Filter manager is a Cronus manager that receives target detections from sensor managers, filters out false detections and sends real detections to a Simulation Data Manager.

Originally, all of the JDL managers were written in Cronus. The goal was to implement the Filter Detections operation as an Isis server and have the JDL simulation run as before. That is, the modification to the simulation would be transparent to the user and both Cronus managers and Isis servers would be used in the simulation.

Appendix B.1 shows the modified Cronus Filter Detections routine. It is important to note that the Cronus Filter Detections routine is not entirely replaced by the Isis server code. The header of the operation is left intact, along with the three standard Cronus operation parameters: a pointer to an operation independent header, a pointer to the operation's input arguments, and an output parameter for returning values. This was intentionally done to facilitate communication with the other Cronus managers in the simulation, most notably the Sensor Managers and SimData manager. If the entire Cronus Filter Detections routine was removed, the Sensor Manager code would have to be modified to be capable of calling the Isis Filter Detections task instead of the Cronus operation. The goal was to keep the modifications isolated to the Target Filter manager to minimize the impact on other JDL managers. Keeping a "stub" of the Cronus Filter Detections operation allows this to be done. The Sensor Managers communicate with the Cronus Filter Detections operation as before. The difference is that the "guts" of the Cronus Filter Detections operation have been removed and replaced with a call to the Isis Filter Detections task (see Figure 3).

```
jdltfilgenFilterDetections(r,input,output)
  OperationParms *r;
  reqjdltfilgenFilterDetections *input;
  repjdltfilgenFilterDetections *output;
{
}
```

Call to Isis Filter Detections Task

Figure 3. Cronus Stub

The first part of this Cronus operation converts the array of DETECTIONDATA into an array that can be handled by Isis. The reason for this is the nested structures within the DETECTIONDATA structure. There is no easy way in Isis to pass an array of structures with nested structures. The workaround was to "flatten" the DETECTIONDATA structure. That is, all of the nested structures are removed and each field is inserted in a new IDD (Isis Detection Data) structure. So, the first part of the code

copies the appropriate DETECTIONDATA information into the "flattened" IDD structure. An array of IDD structures can be passed to Isis with no problem at all. Once the array of IDD structures is passed to Isis, the Isis Filter Detections task converts it back into an array of DETECTIONDATA, filters out false target detections, and stores the results with the Cronus Simulation Data Manager.

The next section of the routine joins the Isis process group and issues the call to the Isis Filter Detections task shown below:

```
nresp=cbcast(group_addr,FILTER,"%X",Idd,input->dimensions.Detections, ALL,"%d",&test);
```

Idd is the flattened array of DETECTIONDATA, and input->dimensions.Detections is the number of detections in the current sensor sweep.

After this call, the rest of the routine simply leaves the Isis process group and frees memory allocations.

In summary, the Cronus Filter Detections routine converts the data into a form that can be used by Isis, joins the Isis process group, makes the call to Isis to perform the target filtering, and then leaves the Isis process group and frees memory. The Isis Filter Detections routine, which does the actual target filtering, is shown in Appendix B.2.

The first part of the Isis FilterDetections routine converts the flattened array of IDD type back into an array of DETECTIONDATA, array "list". After this is done, the array of targets are filtered. New and/or modified targets are updated with the Cronus Simulation Data Manager. The actual update is performed by a call to the Cronus operation UpdateSimObject.

One important point needs to be made about the following line in the Isis server code:

```
i = jdlobj_declare();
```

Without this declaration, the CanTypeToLen call fails, and thus the Isis Filter Detections task fails as well. The reason this call is needed is that a DETECTIONDATA object is a subtype of type JDLOBJ which in turn is a subtype of type OBJECT. In the inventory example, an INVENTORY object was a direct subtype of OBJECT. To quote from the Cronus 3.0 Programmer's reference manual: "All cantypes defined by type object (obj), are already declared so there is no need to call obj_declare(). Whenever using routines that access any standard Cronus cantypes, the user must declare the cantypes first." This is accomplished above by calling the jdlobj_declare routine.

So, in summary, Isis received the array of detections from Cronus, filtered out the false detections, and made the appropriate Cronus calls to store the real target information in a Cronus object database. The entire JDL demonstration runs as if never modified. So the Cronus/Isis integration is a success in this more complicated example as well.

The JDL example above shows the inter-operability of Cronus and Isis. However, it does not show how the unique features of Cronus and Isis were combined to produce a greater capability than either DCE alone. To do this, the Isis Resource Manager is configured to keep a minimum number of copies of the Isis FilterDetections service running at all times. This is superior to the Cronus replication mechanism because Isis will automatically restart the FilterDetections service even after failures occur to maintain the desired number of copies. With Cronus, manual intervention is required to restart a failed service.

6.0 Summary

The CRISIS project focuses on one serious limitation of today's distributed computing technologies. This limitation is that no one DCE can easily and efficiently address all the needs of every application. The reason for this is that many DCEs are based on vastly

different design methodologies that make them better suited for different types of problems. With the technology available today, why should application designers settle for one particular DCE that does not totally fulfill their needs? The CRISIS project addresses this problem by using the client-server model to integrate two different DCEs. This method was fairly simplistic and preserved the integrity of both DCEs.

What is the future of CRISIS? It is useful for legacy software. For example, a Cronus developer can use existing Isis services in the development of applications, thus saving time and money. Similarly, existing Cronus services can be used in the development of Isis applications.

An environment like CRISIS would also be useful as the micro-kernel architecture matures. As DCEs are layered on top of the micro-kernel, CRISIS-like solutions can act as the glue to integrate them and allow them to inter-operate. An application designer could then use several different DCEs to solve a problem and not make the compromises that are associated with choosing just one DCE.

The Object Management Group (OMG) is currently developing a Common Object Request Broker Architecture (CORBA) specification that is trying to address this inter-operability concept. Both Cronus and Isis are undergoing development to become CORBA compliant. Hopefully CORBA compliant DCEs will eventually support heterogeneous distributed computing environments. Until then, efforts like CRISIS demonstrate that this inter-operability is attainable today without undue complexity.

References

[Berets93] Berets, J.C., N. Cherniack, and R. Sands, "Introduction to Cronus", BBN Technical Report, Jan. 1993.

[Birman87] Birman, Kenneth P. and Thomas A. Joseph, "Exploiting Virtual Synchrony in Distributed Systems", Proceedings of the 11th ACM Symposium on Operating Systems Principles, pages 123-138, Austin, Texas, November 1987. ACM SIGOPS.

[Birman89] Birman, Kenneth and Keith Marzullo. "A Brief Overview of the ISIS Distributed Programming Toolkit and the Meta Distributed Operating System", March 1989.

[Birman91] Birman, Kenneth P., "The Process Group Approach to Reliable Distributed Computing", Department of Computer Science, Cornell University, July 1991.

[Cooper92] Cooper, Robert C. B., Bradford B. Glade, Kenneth P. Birman, and Robert van Renesse, "Light-Weight Process Groups", Department of Computer Science, Cornell University, 1992.

[Cronus92] Cronus Programmer's Reference Manual Release 3.0. Computer Software. Cronus, 1992.

[Hurley94] Hurley, Patrick M. and Scott M. Huse, "The Survivable Distributed Computing Environment", Rome Lab Technical Report RL-TR-94-29, June 1994.

[IEEE90] IEEE STD 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.

[Isis92] Isis Distributed Toolkit User Guide and Reference Manual. Computer Software. Isis, 1992.

[JDL92] Joint Directors of Laboratories (JDL) Tri-Service Distributed Technology Experiment Working Group, "Annual Technical Report", 1992.

[Johnson91] Johnson, Brad C. "A Distributed Computing Environment Framework: An OSF Perspective", OSF Technical Paper, Cambridge, Massachusetts, June 1991.

Appendix A

Appendix A.1 (Cronus code with embedded Isis client code)

```
invgenAddToItem(r, input, output)
      OperationParms *r;
      reqinvgenAddToItem *input;
      repinvgenAddToItem *output;
{
Item                          *ip;
register ObjectDescriptor     *objdesc;
int                           i;

/* for find item */
Item        **ItemL;
int         nItemL;
UID         *UIDL;
int         nUIDL;

/* for AddToStock */
char        *desc;
int         count;

/* Isis Variables */
address     *group_addr;
groupview   *group_v;
int         nresp;
short       nmemb;

/* invoke FindItem to get uid */
i = InvokeinvgenFindItem(input->Description,
                         &ItemL, &nItemL, &UIDL, &nUIDL, NULL);

if (i == ERROR)
      PSST(PS_STDERR | PS_X_BAD, PS_LOG_ALWAYS,
      "Can't Find Item:\n\t%s\n",
      ErrorMessage());

/* Get the object from object database. */
if ((objdesc = ReadObjectDescriptor(UIDL[0])) == NULL)
      PSST(PS_LOG|PS_CAN_ABORT|PS_X_BAD,
      PS_LOG_ALWAYS,
      "AddToItem:  Error reading object descriptor:\n\t%s\n",
      ErrorMessage());

if ((ip = GetINVData(objdesc)) == NULL)
      PSST(PS_LOG|PS_CAN_ABORT|PS_X_BAD,
      PS_LOG_ALWAYS,
      "AddToItem:  Error getting object data:\n\t%s\n",
      ErrorMessage());

/* CONNECT TO Isis */
isis_remote_init((char *) 0,1602,1603,Isis_AUTOSTART);
```

```
isis_start_done();

group_addr = pg_lookup(INV_GROUP_NAME);

pg_client(group_addr, "");

/* GET THE CURRENT PROCESS GROUP VIEW */
group_v = pg_getview(group_addr);

/* FIND OUT HOW MANY MEMBERS ARE IN THE PROCESS GROUP */
nmemb = group_v->gv_nmemb;

/* BROADCAST THE COUNT FREQUENCIES REQUEST TO THE SERVERS */
printf("   Broadcasting to %d servers\n", nmemb);

/* CALL the Isis task invAddToStock */
nresp=cbcast(group_addr,ADD,"%s%d%d",
ip->Description,ip->Count,input->Add,1,
"%s%d", desc,&count);
pg_leave(group_addr);

printf("desc = %s, count = %d, \n",desc,count);

ip->Count = count;

/* Provide output parameters for reply. */
output->Count = ip->Count;
output->Description = Talloc (strlen (ip->Description) + 1);
(void) strcpy (output->Description, ip->Description);
output->valid = TRUE;

/* Update the database. */
StoreINVData(objdesc, ip, TRUE);
WriteObjectDescriptor(objdesc);

FreeINVData(ip);
(void)FreeObjectDescriptor(objdesc);
}
```

Appendix A.2 (Isis AddToStock task)

```
invAddToStock(msg_ptr)
     message  *msg_ptr;
{
char  item_name[25];
int   count, addamt;

msg_get(msg_ptr, "%s%d%d", item_name, &count,
        &addamt);
if (addamt < 1)
      {
      reply(msg_ptr, "%d", NOT_POSITIVE);
```

```
        return;
        }
count += addamt;
reply(msg_ptr, "%s%d", item_name, count);
}
```

Appendix A.3 (Isis code with embedded Cronus client code)

```
invAddToStock(msg_ptr)
     message  *msg_ptr;
{
char  item_name[25];
int   addamt, itemindx;
int   i;
int   count;
char  *desc;

msg_get(msg_ptr, "%s%d", item_name, &addamt);
if (addamt < 1)
      {
      reply(msg_ptr, "%d", NOT_POSITIVE);
      return;
      }

itemindx = finditem(item_name);
if(itemindx >= 0)
      {
      /* update Count by invoking Cronus operation */
      i = InvokeatgenAddToStock(item_name,inv_db[itemindx].Count,
                                addamt,&count,&desc,NULL);

      if (i == ERROR)
            PSST(PS_STDERR | PS_X_BAD, PS_LOG_ALWAYS,
            "Error invoking AddToStock:\n\t%s\n",
            ErrorMessage());

      printf("CRONUS returns -> Description is %s, New Count is
             %d\n",desc,count);
      inv_db[itemindx].Count = count;
      reply(msg_ptr, "%d", inv_db[itemindx].Count);
      }
else  /* Return error */
      reply(msg_ptr, "%d", NO_ITEM);
}
```

Appendix A.4 (Cronus AddToStock Operation)

```
atgenAddToStock(r, input, output)
    OperationParms *r;
    reqatgenAddToStock *input;
    repatgenAddToStock *output;
{
```

```
printf("Cronus operation AddToStock received Isis
       invocation\n");

/* Check to make sure positive number of items are being
   added. */
if (input->AddIt < 1)
      {
      Nack(r, E_MUST_ADD_POSITIVE);
      return;
      }

/* Update the object. */
input->OrigCount += input->AddIt;

/* Provide output parameters for reply. */
output->ReturnCount = input->OrigCount;
output->ReturnDesc = Talloc (strlen(input->OrigDesc) + 1);
(void) strcpy (output->ReturnDesc, input->OrigDesc);
output->valid = TRUE;
printf("Cronus operation AddToStock returning results to Isis\n");
}
```

Appendix B

Appendix B.1 (Cronus Filter Detections Operation with Embedded Isis client code)

```
jdltfilgenFilterDetections(r, input, output)
    OperationParms *r;
    reqjdltfilgenFilterDetections *input;
    repjdltfilgenFilterDetections *output;
{
/* Isis Variables */
message     *msg_p;
address     *group_addr;
groupview   *group_v;
int         nresp;
short       nmemb;
IDDTYPE     *Idd, *Idd_test;
int         I, test;
char        *temp;

/* Notify the StatusDisplay Manager */
DisplayStartOp( r->msgdes.source );

/* "Flatten" the array of DETECTIONDATA */

Idd = (IDDTYPE *) malloc(sizeof(IDDTYPE) *
                         input->dimensions.Detections);
for (i = 0; i < input->dimensions.Detections; i++)
      {
      temp = UIDtoSTRING(input->Detections[i]->OriginUID,NULL);
      strcpy(Idd[i].OriginUID, temp);
      free(temp);
      strcpy(Idd[i].OriginName,input->Detections[i]->OriginName);
      temp = INTERVALtoSTRING(input->Detections[i]->Time,NULL);
      strcpy(Idd[i].Time, temp);
      free(temp);
      Idd[i].snr = input->Detections[i]->snr;
      temp = UIDtoSTRING(input->Detections[i]->Target->TargetUID,NULL);
      strcpy(Idd[i].TargetUID, temp);
      free(temp);
      strcpy(Idd[i].TargetDescription,input->Detections[i]->Target-
            >TargetDescription);
      strcpy(Idd[i].TargetType,input->Detections[i]->Target-
            >TargetType);
      Idd[i].TargetNum = input->Detections[i]->Target->TargetNum;
      strcpy(Idd[i].ThreatValue,input->Detections[i]->Target-
            >ThreatValue);
      Idd[i].Latitude=input->Detections[i]->Target-
            >TargetLocation.Latitude;
      Idd[i].Longitude=input->Detections[i]->Target-
            >TargetLocation.Longitude;
      Idd[i].Altitude= input->Detections[i]->Target-
            >TargetLocation.Altitude;
      }

printf("Connecting to Isis\n");
/* need to define a new format type in order to pass an array of
   records */
```

```
isis_define_type('x', sizeof(struct ISISDetectionData), idd_conv);

/* CONNECT TO Isis */
isis_remote_init((char *) 0,1602,1603,Isis_AUTOSTART);

isis_start_done();

group_addr = pg_lookup(TARGFIL_GROUP_NAME);

pg_client(group_addr, "");

/* GET THE CURRENT PROCESS GROUP VIEW */
group_v = pg_getview(group_addr);

/* FIND OUT HOW MANY MEMBERS ARE IN THE PROCESS GROUP */
nmemb = group_v->gv_nmemb;

/* BROADCAST THE COUNT FREQUENCIES REQUEST TO THE SERVERS */
printf("   Broadcasting to %d servers\n", nmemb);

nresp=cbcast(group_addr,FILTER,"%X",Idd,input->dimensions.Detections,
             ALL,"%d",&test);

printf("Isis error number = %d \n",isis_errno);
printf("test = %d \n",test);
printf("NUMBER RESPONDING nresp = %d \n",nresp);

for (i = 0; i < input->dimensions.Detections; i++)
      {
      free(Idd[i].OriginName);
      free(Idd[i].TargetDescription);
      free(Idd[i].TargetType);
      free(Idd[i].ThreatValue);
      }

free((char*) Idd);

pg_leave(group_addr);
printf("Leaving Isis\n");

/* Notify the StatusDisplay Manager */
DisplayFinishedOp();
}

/* This function is called by isis_define_type inorder to pass an array
of structures of type inv_element in a message */

idd_conv(Idd)
      struct ISISDetectionData *Idd;
{
      msg_convertlong(&Idd->snr);
      msg_convertlong(&Idd->TargetNum);
      msg_convertlong(&Idd->Latitude);
      msg_convertlong(&Idd->Longitude);
      msg_convertlong(&Idd->Altitude);
}
```

Appendix B.2 (Isis Filter Detections Routine)

```
IsisFilterDetections(msg_ptr)
      message *msg_ptr;
{

int                  j, i, top, MapNum, ndx, new, nlist, MapFlag, ThisList,
                     Updated;
int                  NumReal = 0, NumSemi = 0, NumFake = 0;
int                  *flag;
UID                  thisuid, *simuid, null_uid;
BOOL                 FOUND;
LOC3D                pos;  /* Lat, Long, Alt */
char                 *desc, *dtype, *threat;
DETECTIONdata        **list;
octet                dd_canvalue;
long                 dd_canlen;
SENSORdata           **SensorStatsTemp;
int                  count;
IDDTYPE              *Idd;

printf("*** Entered the Isis filter routine. ***\n");

msg_get(msg_ptr, "%+X", &Idd, &count);
reply(msg_ptr, "%d",5);

list = (DETECTIONdata **) malloc(sizeof(DETECTIONdata *) * count);
for (i = 0; i < count; i++)
      {
      list[i] = (DETECTIONdata *) malloc(sizeof(DETECTIONdata));
      STRINGtoUID(Idd[i].OriginUID,&list[i]->OriginUID);
      list[i]->OriginName = (char *) malloc(strlen(Idd[i].OriginName) +
            1);
      strcpy(list[i]->OriginName,Idd[i].OriginName);
      STRINGtoINTERVAL(Idd[i].Time,&list[i]->Time);
      list[i]->snr = Idd[i].snr;
      list[i]->Target = (TARGETdata *) malloc(sizeof(TARGETdata));
      STRINGtoUID(Idd[i].TargetUID,&list[i]->Target->TargetUID);
      list[i]->Target->TargetDescription = (char *)
      malloc(strlen(Idd[i].TargetDescription) + 1);
      strcpy(list[i]->Target
             ->TargetDescription,Idd[i].TargetDescription);
      list[i]->Target->TargetType = (char *)
            malloc(strlen(Idd[i].TargetType) + 1);
      strcpy(list[i]->Target->TargetType,Idd[i].TargetType);
      list[i]->Target->TargetNum = Idd[i].TargetNum;
      list[i]->Target->ThreatValue = (char *)
            malloc(strlen(Idd[i].ThreatValue) + 1);
      strcpy(list[i]->Target->ThreatValue,Idd[i].ThreatValue);
      list[i]->Target->TargetLocation.Latitude = Idd[i].Latitude;
      list[i]->Target->TargetLocation.Longitude = Idd[i].Longitude;
      list[i]->Target->TargetLocation.Altitude = Idd[i].Altitude;
      }

msg_delete(msg_ptr);

nlist = count;
Report.TotalDetections += nlist;
Report.OpsProcessed++;
```

```
GetNullUID(&null_uid);

i = jdlobj_declare();

/* Check if detections are Real/Semi/Fake */
for (ndx = 0;ndx < nlist; ndx++)
      {
      /* copy vital information used as the key parameters when calling
      the SimulationData manager. */
      CopyUID( &(list[ndx]->Target->TargetUID),&thisuid );
      CopyLOC3D( &(list[ndx]->Target->TargetLocation),&pos );
      threat = malloc( strlen(list[ndx]->Target->ThreatValue)+1 );
      strcpy( threat,list[ndx]->Target->ThreatValue );
      desc = malloc( strlen(list[ndx]->Target->TargetDescription)+1 );
      strcpy( desc,list[ndx]->Target->TargetDescription );
      dtype = malloc( strlen(list[ndx]->Target->TargetType)+1 );
      strcpy( dtype,list[ndx]->Target->TargetType );
      i = CanTypeToLen( DETECTIONDATA,list[ndx] );
      if( i == -1)
            PSST(PS_LOG, PS_LOG_ALWAYS,
            "CanTypeToLen Error:\n\t%s\n",
            ErrorMessage());
      printf("Return value of CanTypeToLen call: %d\n",i);
      printf("threat = %s, decs = %s, dtype = %s \n",threat,desc,dtype);

      /* check if real detection */
      if (NOT IsNullUID( &thisuid ))
            {
            printf( "PROCESSING Detection[%d]\n",ndx );
            NumReal++;

            /* change coords to minutes */
            pos.Latitude = ((pos.Latitude / 10) / 60);
            pos.Longitude = ((pos.Longitude /10) / 60);

            printf( "  Detection is REAL @
            (%ld,%ld,%ld)\n",pos.Latitude,pos.Longitude,pos.Altitude );

            /* check to see if detection has been seen before.  if so,
               update it.  if not, create/store new detection with
               simdata manager. */
            simuid = CheckStorageList( thisuid );
            if (simuid == NULL)
                  {
                  printf( "  Detection is being STORED\n" );

                  /* Send detection to simdata manager */
                  simuid = &null_uid;
                  FOUND = StoreSimObject( simuid,&pos,AirForce,
                                    TargetData,threat,dtype,
                                    DETECTIONDATA,i,list[ndx] );

                   /* if operation sucessfully completes, add the
                      simdata object UID to the storage list and report
                      to other targfil managers. */
                  if (FOUND == OK)
                        {
                        printf( "  STORING Detection\n" );
                        AddToStore( thisuid,*simuid );
```

```
                        printf( "  Notifying other TargetFilters of
                                    Detection\n" );
                        IdentifyTarget( thisuid,*simuid );
                        }
                  else  /* log reason why operation failed */
                        {
                        printf( "  Unable to complete operation:
                                    SimulationData Manager\n" );
                        printf( "    %s\n", ErrorMessage() );
                        }
                  }
            else  /* this detection needs updating */
                  {
                  printf( "  UPDATING Detection\n" );
                  UpdateSimObject(
                        simuid,&pos,NULL,NULL,NULL,NULL,
                        DETECTIONDATA,i,list[ndx] );
                  }
            }
      else  /* process the fake detection */
            {
            NumFake++;
            }
      }  /* iterate to next detection */
printf( "Finished DetectionSet: %d Detections\n\n",nlist );

Report.RealDetections     += NumReal;
Report.SemiRealDetections += NumSemi;
Report.FakeDetections     += NumFake;

printf("*** Exited the Isis filter routine. ***\n");

return( OK );

}


/* init */
void
pg_init()
{
printf("in pg_init\n");
}

/* This function is called by isis_define_type inorder to pass an array
of structures of type inv_element in a message */
idd_conv(Idd)
   struct ISISDetectionData *Idd;
{
msg_convertlong(&Idd->snr);
msg_convertlong(&Idd->TargetNum);
msg_convertlong(&Idd->Latitude);
msg_convertlong(&Idd->Longitude);
msg_convertlong(&Idd->Altitude);
}
```

Rome Laboratory

Customer Satisfaction Survey

RL-TR-_____

Please complete this survey, and mail to RL/IMPS, 26 Electronic Pky, Griffiss AFB NY 13441-4514. Your assessment and feedback regarding this technical report will allow Rome Laboratory to have a vehicle to continuously improve our methods of research, publication, and customer satisfaction. Your assistance is greatly appreciated.
Thank You

_____

_____

Organization Name:_____(Optional)

Organization POC: _____(Optional)

Address:_____

1. On a scale of 1 to 5 how would you rate the technology developed under this research?

5-Extremely Useful 1-Not Useful/Wasteful

Rating_____

Please use the space below to comment on your rating. Please suggest improvements. Use the back of this sheet if necessary.

2. Do any specific areas of the report stand out as exceptional?

Yes___ No_____

If yes, please identify the area(s), and comment on what aspects make them "stand out."

3. Do any specific areas of the report stand out as inferior?

Yes___ No___

If yes, please identify the area(s), and comment on what aspects make them "stand out."

4. Please utilize the space below to comment on any other aspects of the report. Comments on both technical content and reporting format are desired.

*U.S. GOVERNMENT PRINTING OFFICE: 1995-610-126-50171

MISSION

OF

ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.