RL-TR-95-117
Final Technical Report
July 1995


# CRONUS ENHANCEMENTS

BBN Systems & Technologies

James Berets, Christopher Barber, J. Hunter Barr, John Bowe,
Natasha Cherniack, Jonathan Cole, Michael Dean,
Chantal Eide, Richard Floyd, Robert Goguen,
Steven Jeffreys, Penelope Karr, Richard Mackey,
Paul Neves, Richard Salz , Kenneth Schroder, Susan Pawlowski Sours,
Stephen Vinter, Edward Walker, Bernard Cosell, Robert Coulter, John Day,
Anne-Marie Lambert, Robert Lebow, Joel Levin, Masoud Marvasti, Martha
Steenstrup, Thomas Tignor, and David Waitzman

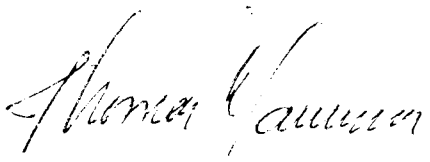


*APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.*

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-95-117 has been reviewed and is approved for publication.

APPROVED: 

THOMAS F. LAWRENCE
Project Engineer

FOR THE COMMANDER: 

JOHN A. GRANIERO
Chief Scientist
Command, Control & 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 |
|---|---|---|
| | July 1995 | Final Dec 88 - Jun 94 |

**4. TITLE AND SUBTITLE**

CRONUS ENHANCEMENTS

**5. FUNDING NUMBERS**

C - F30602-89-C-0029
PE - 63728F
PR - 2530
TA - 01
WU - 50

**6. AUTHOR(S)**

James Berets, Christopher Barber, J. Hunter Barr, John Bowe, Natasha Cherniack, (see reverse)

**7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)**

BBN Systems & Technologies
10 Moulton Street
Cambridge MA 02138

**8. PERFORMING ORGANIZATION REPORT NUMBER**

N/A

**9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)**

Rome Laboratory (C3AB)
525 Brooks Rd
Griffiss AFB NY 13441-4505

**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

RL-TR-95-117

**11. SUPPLEMENTARY NOTES**

Rome Laboratory Project Engineer: Thomas F. Lawrence/C3AB/(315) 330-2925

**12a. DISTRIBUTION/AVAILABILITY STATEMENT**

Approved for public release; distribution unlimited.

**12b. DISTRIBUTION CODE**

**13. ABSTRACT (Maximum 200 words)**

Over the course of this project, four major releases (1.4, 1.5, 2.0, and 3.0) of the Cronus distributed computing environment were produced. These considerable expanded and improved the functionality of the system, increased the system's robustness and usability, improved the system's performance and security, and improved the software engineering process by which the system was developed. In addition, two releases (5.3 and 5.4) of the Advanced Network Management (ANM) system were developed under this effort. This report is organized into 7 sections. Section 1 is the introduction. Section 2 gives overviews of Cronus and ANM. Section 3 gives a chronological description of the progress made during this project, organized by software releases of the systems. Section 4 summarizes the Cronus software development procedures. Section 5 summarizes technical briefings given during the project. Section 6 describes a number of applications in which Cronus was used. Section 7 discusses technology transfer and publication activities.

**14. SUBJECT TERMS**

Distributed computing environment, Distributed operating system, Distributed system, Object oriented system, (see reverse)

**15. NUMBER OF PAGES**

114

**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 | UL |

6. (Cont'd)

Jonathan Cole, Michael Dean, Chantal Eide, Richard Floyd, Robert Goguen, Steven Jeffreys, Penelope Karr, Richard Mackey, Paul Neves, Richard Salz, Kenneth Schroder, Susan Pawlowski Sours, Stephen Vinter, Edward Walker, Bernard Cosell, Robert Coulter, John Day, Anne-Marie Lambert, Robert Lebow, Joel Levin, Masoud Marvasti, Martha Steenstrup, Thomas Tignor, and David Waitzman

14. (Cont'd)

Network management, Distributed processing

# Acknowledgments

## Sponsorship

The Cronus and ANM development described here was sponsored by the U.S. Air Force Rome Laboratory, Griffiss AFB, New York, under contract number F30602-89-C-0029.

## Contributors

The work described here was performed by, and portions of this report were written by the following individuals.

Cronus

James Berets
Christopher Barber
J. Hunter Barr
John Bowe
Natasha Cherniack
Jonathan Cole
Michael Dean
Chantal Eide
Richard Floyd
Robert Goguen
Steven Jeffreys
Penelope Karr
Richard Mackey
Paul Neves
Richard Salz
Kenneth Schroder
Susan Pawlowski Sours
Stephen Vinter
Edward Walker

ANM

Bernard Cosell
Robert Coulter
John Day
Anne-Marie Lambert
Robert Lebow
Joel Levin
Masoud Marvasti
Martha Steenstrup
Thomas Tignor
David Waitzman

## 1. Introduction

This document is the Final Technical Report for the Cronus Enhancements Project. The work for this project was performed over the approximately five year period from December 1988 through June 1994. This project was funded by the U.S. Air Force Rome Laboratory[1] (RL) under contract F30602-89-C-0029.

Over the course of this project, four major releases (1.4, 1.5, 2.0, 3.0) of the Cronus distributed computing environment were produced. These considerably expanded and improved the functionality of the system, increased the system's robustness and usability, improved the system's performance and security, and improved the software engineering process by which the system was developed. In addition, two releases (5.3 and 5.4) of the Advanced Network Management System (ANM) system were developed under this effort. These are summarized as follows.

- BBN completed and delivered release 1.4 of Cronus to the Rome Laboratory on August 31, 1989. The delivery included all software and manuals needed to use, program, and operate the software. A three day project review meeting was held at BBN on September 13-15 at which 12 presentations and 4 demonstrations were provided for RL staff.

- BBN completed and delivered release 1.5 of Cronus on May 5, 1990. This delivery culminated the second project phase. A three day project review meeting was held at BBN on May 23-25 to review the contents of this release; 8 presentations and 3 demonstrations were given to RL staff.

- BBN shipped release 2.0 of Cronus to the Rome Laboratory on May 16, 1991. Release 2.0 was developed over a period of approximately 11 months and completed the third project phase. A two day review was held at BBN on April 30 and May 1 to review the contents of the release. 15 presentations and 2 demonstrations were given to RL staff. Representatives were also present from three other government agencies, in keeping with our goal of trying to facilitate the use of Cronus by the Government.

- BBN delivered release 3.0 of Cronus to the Rome Laboratory on February 16, 1993. Release 3.0 was developed over a period of approximately 12 months, from January to December 1992. The delivery of 3.0 completed the fourth project phase. A one day review was held at Rome Laboratory on February 9 to review the contents of release 3.0. In keeping with our goal of trying to facilitate the use of Cronus by the Government, an additional review was held (under another effort) at the NCCOSC RDT&E Division (NRaD) in San Diego, CA in early April.

- BBN delivered ANM 5.4, with features necessary for managing global systems based on Cronus, to the Rome Laboratory on December 30, 1993. Release 5.4 was developed over a period of approximately 18 months, from August 1992 to December 1993. The delivery of ANM 5.4 completed the fifth project phase. ANM development under this effort included an interim release, ANM Release 5.3, to demonstrate early versions of some of the features being developed in ANM 5.4. A one day review was held at BBN on August 11, 1993 to review the contents of ANM 5.4. The Contract

[1] At contract award the Rome Laboratory was known as the Rome Air Development Center (RADC). For uniformity, we refer to the organization by its current name in this document.

Program Manager was in attendance, as were additional interested parties from RL and the U.S. Army Communications and Electronics Command (CECOM).

Information about Cronus and ANM above and beyond that provided here may be found in their complete documentation sets respectively. The current Cronus documentation set consists of the following:

- *Introduction to Cronus* - BBN Report Number 6986, January 1993
- *Cronus Release Notice* - Release 3.0, December 1, 1992
- *Cronus User's Reference Manual* - Release 3.0, December 1, 1992
- *Cronus Programmer's Reference Manual* - Release 3.0, December 1, 1992
- *Cronus Programmer's Reference Manual (Lisp)* - Release 3.0, December 1, 1992
- *Cronus Operator's Reference Manual* - Release 3.0, December 1, 1992
- *Cronus Tutorial Documents* - Release 3.0, December 1, 1992
- *Cronus Installation Manual* - Release 3.0, December 1, 1992
- *Cronus Database Installation Manual* - Release 3.0, December 1, 1992

ANM documentation includes:

- *ANM Users Guide*
- *ANM Installation Guide*
- *ANM Software Release Notice*

This report is organized as 7 sections. Section 2 gives overviews of Cronus and ANM. Section 3 gives a chronological description of the progress made during this project, organized by software releases of the systems. Section 4 summarizes the Cronus software development procedures. Section 5 summarizes technical briefings given during the project; Section 6 describes a number of applications in which Cronus was used. Section 7 discusses technology transfer and publication activities.

# 2. An Overview of Cronus and ANM

## 2.1 An Overview of Cronus

Cronus is a *distributed computing environment.*, a complete, easy to use toolkit and environment for building and running distributed applications in a heterogeneous environment. Cronus accomplishes the following:

- Facilitates the integration of new and existing application components running on a wide variety of machines and operating systems, and written in a variety of programming languages.
- Provides a powerful, object-oriented framework for decomposing complex distributed applications executing in both wide-area and local-area network environments..
- Decreases distributed application development time.
- Incorporates a mature, high-level inter-process and inter-machine communications toolkit (including naming and location services).
- Includes automatic support for creating and maintaining multiple copies of data on many machines.
- Supports applications executing across one or more administrative domains.
- Simplifies the implementation of applications that use a network of workstations to obtain supercomputer performance, through coarse-grain parallel processing.

Cronus includes:

- A complete object-oriented distributing computing toolkit for developers:
  - specification-driven code generation for clients and servers in a variety of programming languages, including C and Common Lisp,
  - more sophisticated and easier to use than Sun RPC and OSF DCE,
  - with powerful testing and debugging tools,
  - incorporating complete IPC facilities with support for both synchronous and asynchronous invocation;
- Complete support for integrating applications across heterogeneous machines and operating systems;
- Flexible replication facilities to support survivable applications;
- Facilities for authentication and access control;
- Integration of off-the-shelf relational databases (Informix, Oracle, and Sybase); and a
- Complete hierarchical naming system.

Cronus has been designed as a base for the development of large-scale distributed heterogeneous applications. Although internally the system is object-oriented, this aspect of Cronus is largely hidden from application developers. Most of the details of implementing distributed applications are provided by a combination of code automatically generated from an interface specification (including an RPC interface for clients), library routines, and system components. Cronus has been used in a variety of applications, for example the interconnection of Common Lisp-based expert systems with off-the-shelf database systems and FORTRAN simulations. A typical Cronus application is shown in Figure 1.

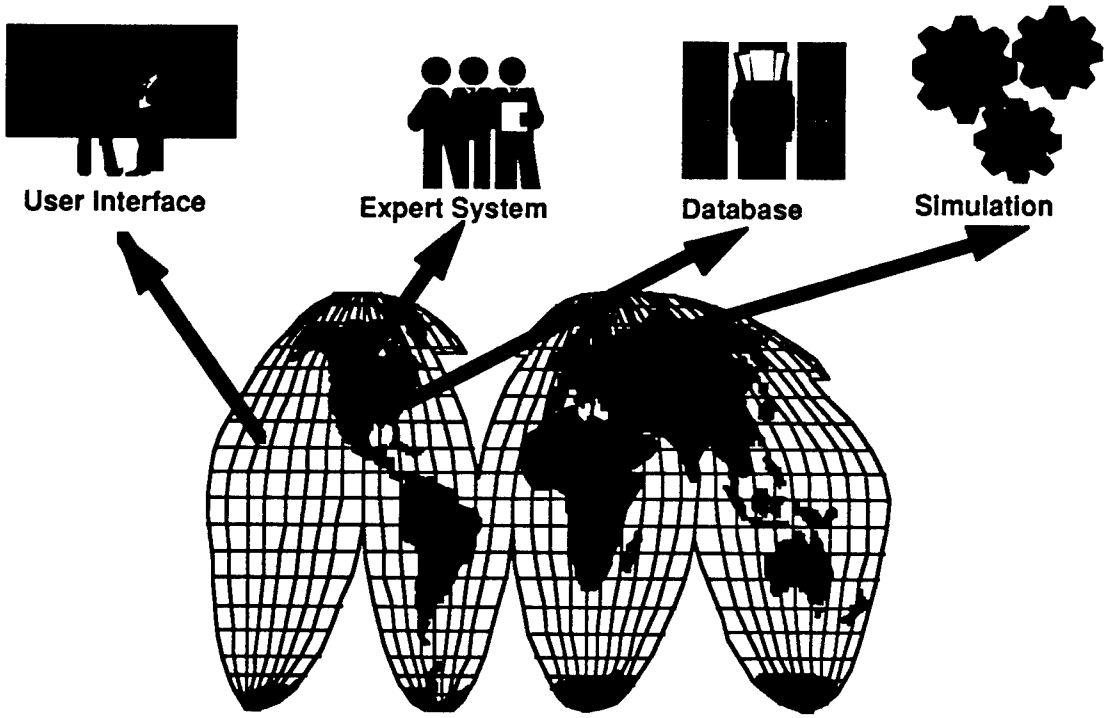



**Figure 1: Global, Highly Heterogeneous Application**

A complete overview of Cronus may be found in the *Introduction to Cronus*. The following sections summarize the key elements of the Cronus architecture.

### 2.1.1 Object Model

Cronus provides a set of services and communication layers on top of native operating systems. Cronus is based on the *object model*: each system resource is an object and is accessed through operations defined by the object's *type*. The object model provides an extensible architecture, in that application developers can cast application-specific resources in terms of new object types. *Type definitions* are organized in a *type hierarchy*, allowing new types to be defined as subtypes of existing types.

Cronus supports heterogeneity by serving as a bypassable layer of abstraction between application programs and native operating systems. Through this approach, application programs gain access to a coherent, uniform (object-oriented) system interface, regardless of computer system base; however, they also retain conventional access to native operating system resources and services.

The following are the basic abstractions of the Cronus object moel:

- **Object:** An object is a resource, such as a file, a directory, a user, a mailbox, an inventory, or a sensor. Objects are generally passive entities that are stored on disk. Objects have also been defined to implement user sessions and in-memory data structures, as well as to encapsulate resources implemented outside Cronus, such as native operating systems, processes, hosts, and relational databases.

- **Type:** A type defines how objects are implemented and used. Applications are implemented by defining new types. A type consists of a set of operation interfaces, code for the operations, and data structures that define the representation of objects. All defined types are organized into a hierarchy, allowing operations implemented by a parent type to be inherited (reused) by a child type. Type inheritance promotes uniformity among types.

- **Operation Invocation**: Objects are accessed by clients through operation invocations. This is the only means of accessing an object, ensuring information hiding and data abstraction.

- **Process**: Processes are the active entities in Cronus. An object manager, composed of one or more processes, is the entity responsible for manipulating all of the objects of one or more types on a host using the operations defined by the types. The Cronus process abstraction corresponds to the process abstraction found in conventional operating systems, and is implemented using the native operating system process.

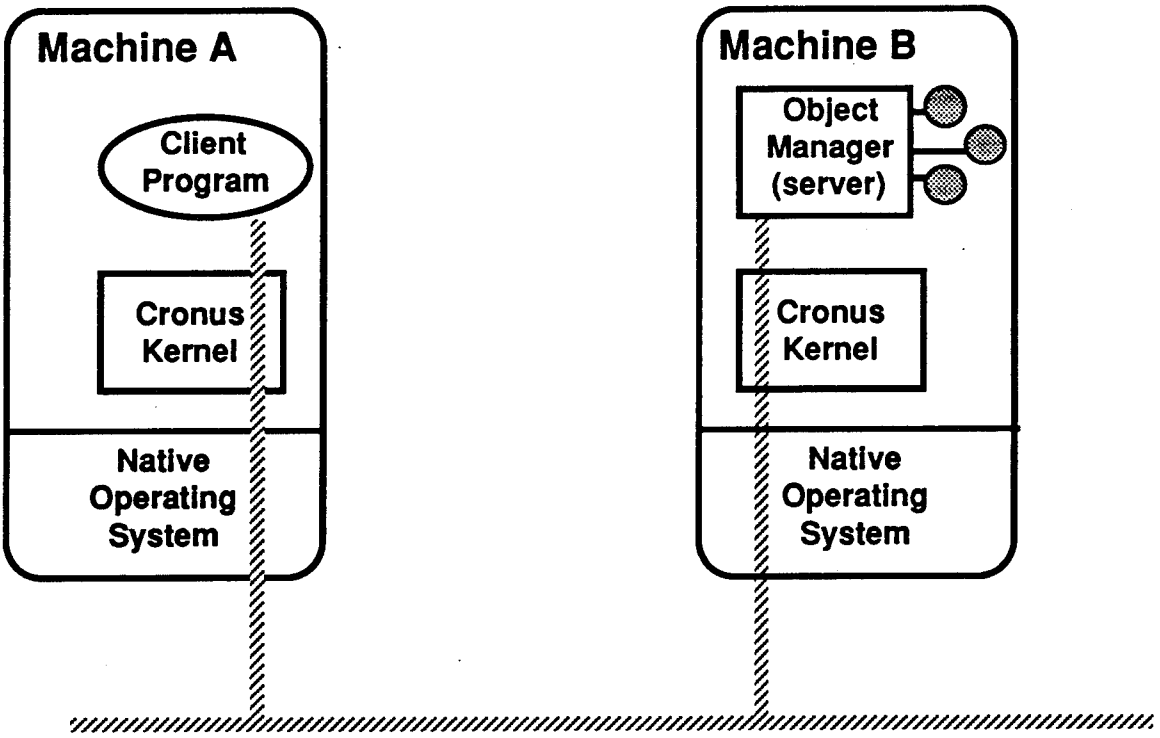



**Figure 2: Cronus Architecture**

Cronus consists of the following components:

- **Services**: A service comprises one or more *manager* processes (i.e., servers) that defines and manages the objects of one or more types. Services implement both system functions and application functions. Current system services include an authentication service, a symbolic naming (directory) service, a network configuration service, a resource monitoring service, and a type definition service. An application is typically composed of several services responsible for several different types of resources.

- **Clients**: Clients are processes that use services. While a service may act as a client to any other service, many clients are not services. Rather, they are processes that interact directly with users, such as user commands, utilities, and application-specific graphical user interfaces.

- **Cronus Kernel**: The primary purpose of the Cronus kernel is to transmit operation invocations from clients to services in a host-independent network transparent fashion. The kernel also implements the basic abstractions of process, host, and operation invocation. The kernel is implemented as a native operating system process that executes in user space, and is run on each host of a Cronus cluster.

### 2.1.2 Interprocess Communication

Cronus interprocess communication (*IPC*) is designed to support operation invocations from clients to services, where the invocations can be synchronous or asynchronous, and can have one or many targets. It is implemented as a series of layers.

Cronus relies on standard communication protocols implemented by the native operating systems at and below the transport layers. Currently, Cronus relies on the *Transmission Control Protocol* (TCP) and *User Datagram Protocol* (UDP), which in turn rely on the *Internet Protocol* (IP).

The lowest layer defined by Cronus is the IPC layer. In a typical scenario, a client process invokes an operation on an object. The invoke is implemented as a message addressed to the object. Objects are referenced by name, not by location. The message is routed by the Cronus kernel to the process serving as the object's manager. The object manager retrieves the message from its message queue to perform the requested operation. The operation is actually performed by a lightweight process (a *thread* or *task*, in Cronus terminology) created by the object manager to execute the code for the invoked operation. Operations are executed concurrently through the use of a nonpreemptible tasking package. Upon completion of the operation, the manager returns a reply to the client. The reply is delivered by the kernel back to the client, who receives the reply. Primitives are also provided to receive messages in parts, and transmission of large messages is automatically optimized through direct client/manager connections.

Above the IPC layer is a layer designated as the message encodement layer. This layer is responsible for encoding and decoding messages using canonical (i.e., system-independent) data representations, allowing transmission of messages between machines with different data representations. Cronus defines canonical data representations for many common data types and structures, and allows developers to define new canonical types from existing ones.

At the highest layer is the RPC layer. This layer presents a remote procedure call (RPC) programming interface to the developer, allowing synchronous and asynchronous operation invocation through high-level function calls. The latter is provided by a

mechanism known as *Futures*. Source code is generated automatically to provide RPC interfaces for invoking operations.

### 2.1.3 Persistent Object Storage

Cronus managers store the state of objects on disk in object databases. This insures that Cronus objects can survive machine crashes, since they are disk-based rather than memory-based. A set of access routines is provided to simplify the storage of variable-length, structured objects.

### 2.1.4 Database Systems

Cronus provides access to off-the-shelf Informix, Oracle, and Sybase relational database systems. This database service allows clients from any host to access the database systems, either with predefined or ad hoc queries. Cronus also provides a forms interface for interactive use.

### 2.1.5 Naming

Cronus has a two-level naming system: high-level symbolic user names and low-level object identifiers. At the user level is a hierarchical symbolic name space, similar to a UNIX directory name space. It differs from the UNIX directory, however, in that symbolic names may be used to reference any type of object (including files, databases, users, and types). An object may be given one or more symbolic names. Symbolic names are maintained by the Directory Service, a distributed service maintaining replicated directories. Users and client programs use the Directory Service to map the symbolic names of objects into their internal system names called *Unique Identifiers* (UIDs). When an object is created, it is assigned a UID that contains the object's type and the host on which it was created. UIDs are implemented in a flat name space, and an object's UID is permanently assigned and guaranteed to be unique over all objects over all time. Generally, symbolic names are used to reference objects in application user interfaces, and UIDs are used internally by system software.

### 2.1.6 Protection

Cronus achieves protection by using *access control lists*. Each object has an access control list associated with it. The access control list for an object specifies the users or groups of users that may access the object and the access rights they possess. Access rights can be defined separately for each type, allowing access controls to be tailored to the application. A distributed Authentication Service authenticates users by subjecting them to a password-based authentication procedure upon login. When a user performs some action that causes an operation invocation, the run-time system compares the authenticated identity of the user with the access control list on the target object to determine if the requested operation can proceed. If not, a permission error is generated.

### 2.1.7 Resource Management

In Cronus, object-level resource management is applied according to the principal of policy / mechanism separation. Cronus provides a set of mechanisms enabling the cooperative enforcement of type-specific policies. The mechanisms includes the following:

- many object managers (implementing a service) can execute on different hosts for each type of object;

- object managers can query the status of their peers, object managers may redirect requests to peers, and

- clients can indicate preferred hosts where operation invocations are routed.

These mechanisms have been used in several services to implement specific management policies, such as a dynamic load balancing policy.

### 2.1.8 Reliability and Fault Tolerance

Cronus supports the migration and replication of objects. Cronus facilities for replication provide the developer with a number of tools that can be used to customize the actual replication mechanisms used for a specific object type.

These facilities are as follows:

- Version Vectors: Every replicated object has a version vector containing a list of host location and version number pairs. This list can be used to determine the locations of all the copies of the object and to detect whether a particular copy is out of date with respect to another. In addition, it can be used to detect whether two copies have been updated in an inconsistent manner.

- Voting Options: Options are provided to specify the number of votes necessary for read and update operations. Since these are independent variables, many different options are available to guarantee availability or consistency, depending on the needs of the particular application.

- Automatic Replication: Automatic replication of objects can be specified as part of the type definition. The number of different hosts that the object should be created on is also specified within the type definition. The actual locations are determined at creation time, allowing for load balancing across multiple managers.

- Incremental or Whole Object Updates: Sometimes it is either convenient or necessary to update remote copies of an object by performing an operation locally and then sending a copy of the new object to the remote sites. However, for large objects such as files, it not efficient to update copies by replacement. In Cronus, developers have the option of specifying whether updating copies of objects should be performed by replacement or by operation.

Cronus also dynamically locates objects when invoking operations, allowing clients always to find an object that has migrated, and always to access a copy of a replicated object provided one is available.

### 2.1.9 Programming Support

The fundamental assumption underlying Cronus programming support is that large-scale applications will be developed in accordance with the object model, just as the Cronus system itself is. Under this assumption, the key to application development is definition of new object types to represent application-specific resources and development of new services to embody the newly defined object types. Cronus programming support focuses on automating the process of developing new services.

Cronus relieves the application developer's coding burden through the use of a nonprocedural program development specification language. A developer provides a

nonprocedural specifications of a new object type, and code is supplied or automatically generated for skeletal object managers, including client remote procedure call stubs, multitasking for concurrent operation processing, data conversion between canonical and system-specific data representations, message parsing and validation, access control checks, operation dispatching, and stable storage management. The application developer completes the object manager by providing routines that implement the operations defined in the new object type.

Cronus programming support also includes:

- Extensive subroutine libraries, including interprocess communication routines, data conversion routines, and RPC interfaces to Cronus objects
- A set of user commands (e.g., list objects in a directory and create an object)
- A set of operator commands (e.g., start a service)
- Operations which are inherited by all objects; they ensure common implementations (e.g., monitoring and control, debugging, and replication and migration support) or common interfaces (access control) among different services
- A sophisticated debugging tool to be used in conjunction with a local debugger to assist in manager debugging
- Source code management software
- A bug tracking facility
- A monitoring service.

### 2.1.10 Platforms

Cronus has been ported to a wide variety of platforms. The following table summarizes the platforms for which Cronus has been compiled, either experimentally or for distribution.

| Vendor | Product | CPU Type | Operating System | First Cronus Version |
|---|---|---|---|---|
| Alliant | FX/80 | Motorola 68020 with proprietary 64-bit CMOS coprocessor | Concentrix | 2.0 |
| Apollo | DN10000 | Apollo PRISM | Domain/OS | 2.0 |
| AT&T | 3B2 | proprietary | UNIX System V | 1.4 |
| AT&T | 6386 | Intel 80386 | UNIX System V/386 | 1.4 |
| BBN | Butterfly GP1000 | Motorola 68020 | Mach | 1.5 |
| BBN | C/70 | proprietary | C/70 UNIX | 1.0 |
| BBN | TC2000 | Motorola 88000 | nX | 2.0 |
| Concurrent | Series 8000 | MIPS | RTU | 2.0 |
| Convex | C-2 and C-3 | proprietary | ConvexOS | 2.0 |
| Cray | Y/MP-EL | proprietary | UNICOS | 2.0 |
| DEC | various | MIPS | Ultrix | 1.5 |
| DEC | various | VAX | 4.2 BSD | 1.0 |
| DEC | various | VAX | Ultrix | 1.0 |
| DEC | various | VAX | VMS | 1.1 |
| Encore | Multimax | National 32x32 | Mach | 1.5 |
| Encore | Multimax | National 32x32 | UMAX 4.2 | 1.5 |

| Vendor | Product | CPU Type | Operating System | First Cronus Version |
|---|---|---|---|---|
| Encore | Multimax | National 32x32 | UMAX System V | 2.0 |
| HP | 9000 Series 300 | Motorola 680x0 | HP-UX | 2.0 |
| HP | 9000 Series 700 | HP PA-RISC | HP-UX | 3.0 |
| IBM | RS/6000 | POWER | AIX | 3.0 |
| Many | PC | Intel 80286 | Xenix | 1.3 |
| Many | PC | Intel 80386 | MS-DOS | 1.5 |
| Masscomp | 5400/5500 | Motorola 680x0 | RTU | 1.2 |
| NeXT | NeXTstation | Motorola 68040 | Mach / NeXTstep | 2.0 |
| Sequent | Symmetry | Intel 80486 | DYNIX | 3.0 |
| Silicon Graphics | 4D60 | MIPS R2000 | IRIX | 1.5 |
| Stardent | Titan | MIPS | UNIX System V | 1.5 |
| Sun | 2/xxx and 3/xxx | Motorola 680x0 | SunOS | 1.0 |
| Sun | Sun 3 | Motorola 680x0 | Mach | 1.4 |
| Sun | 386i | Intel 80386 | SunOS | 1.4 |
| Sun | various | Sun SPARC | SunOS | 1.4 |
| Sun | various | Sun SPARC | Solaris | 3.0 |
| Symbolics | 36xx | proprietary | Genera | 1.3 |
| Symbolics | Ivory | Ivory | Genera | 1.5 |
| TI | MicroExplorer | proprietary | unnamed | 2.0 |

Cronus is highly portable, as the table evidences. Two implementations exist. The primary implementation is written in C; another implementation is written in Common Lisp. Machine-dependent code is confined to a few modules, and porting is relatively inexpensive. In addition to supporting full application development in C and Common Lisp, application components have commonly incorporated FORTRAN code; a C++ support environment is under development, and experimental Ada support was built for Cronus 1.5.

## 2.2 An Overview of ANM

The Advanced Network Management system (ANM) is an integrated network management system that includes facilities for both real-time operations and statistical and performance analysis. ANM is a distributed, hierarchical system. This affords ANM a great deal of scalability and reliability. Overloaded components can be enhanced with additional copies running in parallel, sharing the load. Any component can be replaced by one running on a different platform by simply redirecting the streams interconnecting the modules. ANM components interconnect using a protocol running on top of TCP/IP, and so its modules can be relocated anywhere within the network and are not tied to a specific local area network or subnetwork.

ANM is currently used by a variety of military network applications, including the U.S. Air Force Rome Laboratory test networks, networks at U.S. Army CECOM, and ARPA's Defense Simulation Internet. ANM is also in use for NEARnet, the New England regional backbone of the Internet. ANM runs on the Sun Microsystems SPARC platform.

ANM supports object-oriented network management using an object model based on the International Standards Organization (ISO) object model. ANM provides interfaces to a wide variety of devices, including DoD devices which use the Simple Network Management Protocol (SNMP). These interfaces map the devices' particular monitoring capabilities into a suitably structured Management Information Base (MIB) using the ISO Structure of Management Information. A management application connected to ANM is written entirely in terms of ANM's unified MIB, and so is independent of the specifics of the access protocols and monitoring vocabulary of any specific device. Further, since the ANM MIB includes network management information for all of the devices under management, ANM applications can easily be written to combine information from, and manage, a broad spectrum of network devices or distributed user applications.

As shown in Figure 3, the ANM architecture consists of three components: management applications, proxy agent processes, and "kernel"[2] processes that support the distribution of management information requests and responses. ANM employs a distributed, scalable, extensible architecture in order to support the needs of its client base. All connections between architectural components are via network-based inter-process communications (TCP/IP connections), permitting a very flexible configuration of network management components. The top-level management protocol is an alternate encoding of CMIP (Common Management Information Protocol) known as Fraunhofer CMIP, or fCMIP. ANM uses fCMIP to communicate between the major architectural components of the system. Distribution is supported by the use of fCMIP to communicate between all components of the architecture (except for the graphical user interface). The components are described in the following sections.

[2]These kernel processes are distinct from, and should not be confused with, the Cronus kernel or an operating system kernel.

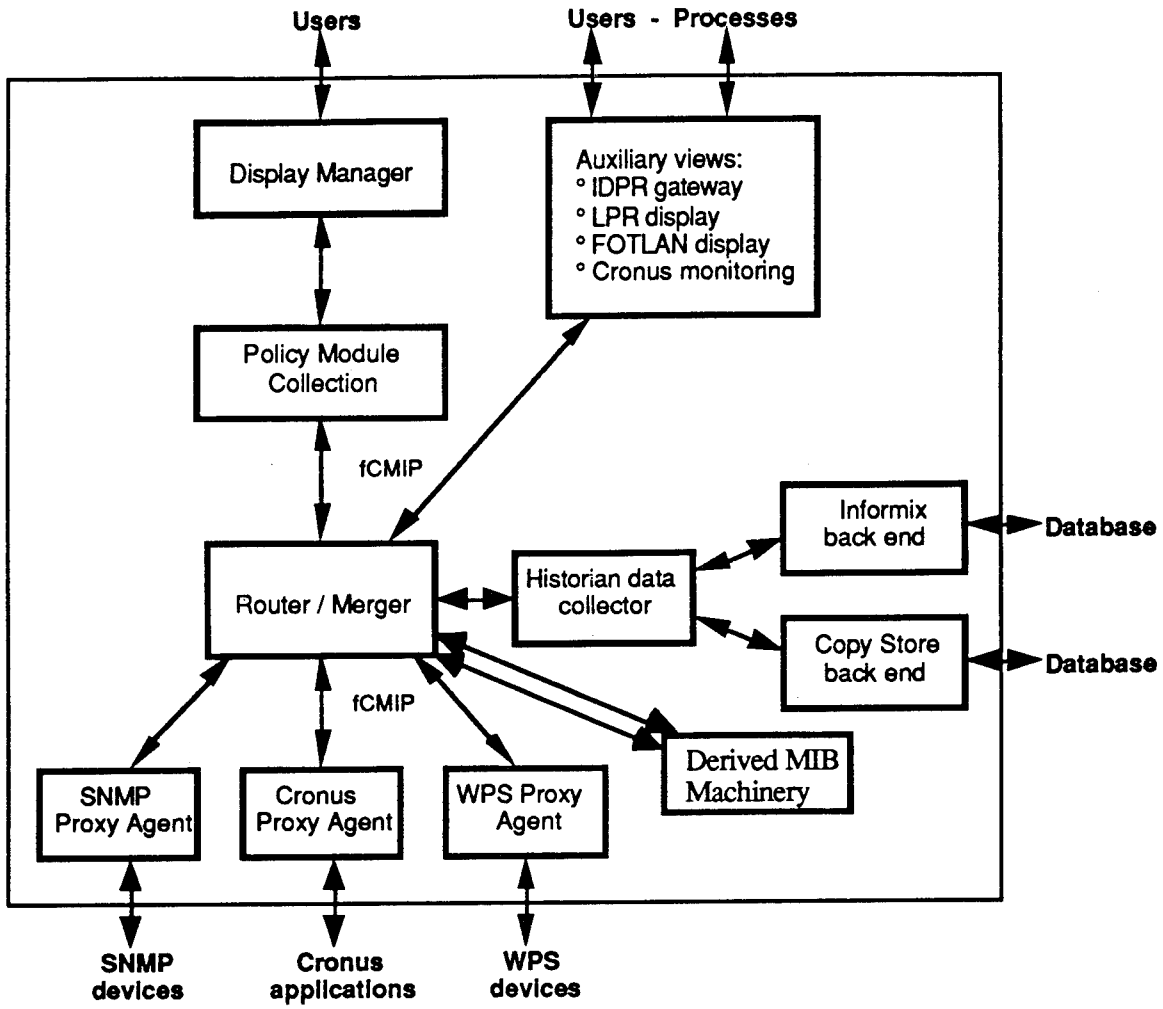



**Figure 3: ANM Architecture**

### 2.2.1 Network Management Components

Graphical user interface: The real-time ANM graphical user interface (GUI) consists of two separate modules. The Display Manager (DM) supports a color map of the network, including hierarchical views, subset views, and user configurable command menus. Its companion software module, the Policy Module Collection (PMC), actually generates the network management requests and translates the responses into directives to the display module. These two modules communicate using a private ASCII protocol.

Other graphical applications: Live tabular displays can run independent of the DM and PMC, generating their own network management requests. They can be started independently or launched by the Display Manager in response to a user command. These have been implemented using Tcl/Tk for rapid prototyping of user interfaces.

Data collection and reporting: The Historian application provides an interface for long term data collection and reporting. The Historian module supports data collections of medium to long term duration (hours or days) and the generation of summary reports from this network data. Back-end programs can store the data into the *Informix* relational database system or into flat text files.

### 2.2.2 Router/Merger

This software module acts as a hub for network management routing decisions. All network management traffic in ANM (between management applications and proxy agents) is routed via a Router/Merger or via a hierarchy of RMs. This allows transparent installation of major system components.

### 2.2.3 Proxy Agents

These software modules translate fCMIP messages into the native protocols supported by the devices being managed. By making these translations available in software modules independent of the devices themselves, ANM is able to quickly provide a unified device interface (based on a unified MIB) to devices that do not support the OSI network management suite. Proxy agents (PAs) implemented are the SNMP PA for all devices managed by the Simple Network Management Protocol (SNMP), the Cronus PA for hosts running Cronus, and the WPS PA for the Defense Simulation Internet Wide-band Packet Switches.

## 3. Project Technical Progress (Chronological)

Over the period of this effort, we developed 4 major Cronus releases as follows:

- Cronus Release 1.4 dated September 1, 1989
- Cronus Release 1.5 dated May 1, 1990
- Cronus Release 2.0 dated March 31, 1991
- Cronus Release 3.0 dated December 1, 1992

and 2 ANM releases:

- ANM Release 5.3 dated September 30, 1993
- ANM Release 5.4.1 dated January 7, 1994.

In this section, we chronologically summarize these software releases.

## 3.1 Cronus Release 1.4 - September 1, 1989

In Release 1.4, we incorporated a number of new mechanisms that facilitated the construction of higher performance Cronus applications, increased the number of supported platforms, and continued our commitment to make Cronus more robust. Some notable new facilities were a mechanism facilitating parallel operation invocation by client programs, and support for direct client-manager connections that bypass the Cronus kernel after setup. We also added the Sun 386i and Sun 4 to our suite of supported host types, and added support for the Mach operating system on the Sun 3 family. A large collection of enhancements and bug-fixes were also incorporated into Cronus Release 1.4, the more significant of which are described below.

### 3.1.1 New Features

Cronus Release 1.4 incorporated two new mechanisms supporting higher-performance applications. These mechanisms, **Futures** and **Direct Connections**, are described here.

#### *3.1.1.1 Futures*

The conventional interface used by Cronus client programs to access remote resources in previous releases was a collection of PSL (Process Support Library) routines, one per operation, that allowed the programs to invoke operations. These remote procedure call (RPC)-style routines, automatically generated by the Cronus manager development tools, combined within each PSL procedure call the code that invokes an operation with the code to get the reply and decode the returned data. The interface was synchronous: it forced the client program to block while waiting for the reply, and made client-initiated concurrency difficult to implement.

In Release 1.4, we introduced **Futures**, a mechanism that can be used to introduce concurrency and coarse-grained parallelism into Cronus programs. Futures allow programs to perform useful work (including additional operation invocations) while waiting for replies from operations in progress. Futures also improve performance in those cases where multiple operations are being performed and communication latency is large compared with operation processing time (for example, when communication is over a satellite communication channel). Using Futures, clients call a non-blocking RPC-style interface to invoke the operation and receive a handle (Future) for that invocation. The Future can later be *claimed* by the client, which is provided with any results that may have been returned. (The client may block or continue with other work if the results are not yet available.) By providing independent calls for performing the invocation and the claim, programs can invoke a number of operations concurrently, maintain a list of the Futures outstanding, and later claim the results as desired.

The generation of Future interface routines was integrated into the Cronus manager development tools. In much the same way that previous releases provided a generated PSL client interface from the type specification, the Release 1.4 manager development tools provided code generation for Futures. Each defined operation had a corresponding generated **FInvoke** routine with the operation's input parameters (for example, *FInvokeobjRemove*) and an **FClaim** routine with its output parameters (for example, *FClaimobjRemove*). In addition to the separate FInvoke and FClaim routines, an **Invoke** routine for each operation (for example, *InvokeobjRemove*) combines the functionality of both the FInvoke and FClaim, in a manner similar to the PSL routines.

One of the limitations of previous Cronus releases was the lack of flexibility available through the PSL routine interface: invoking an inherited generic operation through the PSL interface was not possible, nor was invoking a non-generic operation directed to a designated host. This necessitated modification of the generated routines to accomplish the desired effect. In Release 1.4, the generated FInvoke and Invoke interfaces both contain a new **INVOKECONTROL** structure passed as an input parameter to allow the client to control the behavior of an invocation as desired (for example, to direct an operation to a designated host, or to specify that the client does not wish to receive a reply).

Since the primary function of Futures was to allow coarse-grained parallel processing, in Release 1.4 we provided a comprehensive set of utility functions for managing and manipulating Futures. We implemented an abstraction known as **FutureSets**, which allowed programs to manipulate collections of Futures and automatically claim Future-based invocations in the order that their results become available, removing the burden of managing sets of Futures from the application developer.

In order to control the number of outstanding requests at any one time, we also introduced in Release 1.4 the concept of a **Funnel**. Funnels are used to control the flow of pending invocations by limiting the transmission of invocations from clients using Futures. The size of the neck of the Funnel is specified by the client when the Funnel is created (it can be altered later) and determines the number of outstanding requests permissible. When a Funnel is full, additional operation invocations that are requested using Futures are queued in the client, and will be automatically sent when a reply to an outstanding operation is received.

*Futures and Funnels introduced a powerful mechanism in Cronus for asynchronous operation invocation by clients, which allow Cronus applications to better exploit coarse parallel processing in a distributed heterogeneous environment.*

### *3.1.1.2 Direct Connections*

Experiments and analysis of Cronus releases prior to 1.4 revealed that communication performance could be significantly improved in situations in which a particular client-manager pair was exchanging large quantities of data or communicating regularly. In these situations, the local interprocess communication overhead, context switching, and processing time associated with passing all messages through the Cronus kernels on the client's and manager's hosts limited performance when host failures and object migration occur infrequently.

Pre-1.4 releases of Cronus supported large message transfers using a separate communication channel between a client and a manager for large data transfers. However, this mechanism only bound the client to the manager for the duration of the particular operation invocation, limiting its usefulness. In situations where a client-manager pair was communicating regularly, it was possible to increase overall throughput by transferring data for a collection of operations through a dedicated channel. In Cronus Release 1.4 we introduced a *direct connection* mechanism, similar to large messages, that allowed the kernel to be bypassed for successive communications between a client-manager pair. This resulted in communication performance approaching that of the underlying transport system (TCP/IP) for the communicating process pair.

In Release 1.4, direct connections were explicitly declared. Programs specified the object manager with which they wish to communicate by host and Cronus object type, and the Cronus kernel negotiated connection establishment. Operations invoked on objects (for example, through the FInvoke calls) belonging to the manager for which a connection has

been established were subsequently automatically sent through the direct connection instead of through the Cronus kernel. This behavior continues until the connection is closed, when operation invocations will again be routed through the Cronus kernel.

*The high-level interface provided by direct connections simplifies the task of building higher-performance Cronus applications when large volumes of data or repetitive invocations are present.*

### 3.1.2 New Environments

For Release 1.4, we ported Cronus to several new platforms. This increased the potential Cronus user community and allowed a wider range of systems to support Cronus applications. This section summarizes the features of the additional platforms that were supported.

#### *3.1.2.1 Sun 386i / SunOS 4.0*

The Sun 386i was a deskside workstation family that combined in a single platform the capabilities of the SunOS environment with the ability to simultaneously develop and run MS-DOS-based applications and access PC/AT-bus peripherals.

- CPU — Intel 80386 microprocessor with 80387 floating-point coprocessor
- Estimated MIPS — 3 to 5
- Operating System — SunOS / MS-DOS under SunOS control

Cronus on the Sun 386i leveraged off of two previous supported platforms: the SunOS software environment, and the hardware architecture of the IBM PC/AT 80286-based machines. Cronus on the Sun 386i was a complete implementation, supporting the Cronus kernel, system managers, and the complete set of user commands.

#### *3.1.2.2 Sun 3 / Mach 2.0*

The Sun 3, based on the widely accepted Motorola 68000 family of microprocessors, includes a variety of systems offering a range of price-performance options from low-cost desktop workstations to high-performance servers. The Mach multiprocessor operating system, developed at Carnegie-Mellon University, includes sophisticated facilities for multiprocessing, local interprocess communication, and task synchronization, as well as a user and programming environment similar to Berkeley UNIX 4.3BSD.

- CPU — Motorola 68020 or 68030 microprocessor (with 68881, 68882, or Sun floating point units optional)
- Estimated MIPS — 1.5 to 7
- Operating System — Mach

The implementation of Cronus for the Sun 3 and the Mach operating system in Cronus Release 1.4 was essentially the same as Cronus for the Sun 3 and SunOS. Since the Sun 3 is not a multiprocessor, the initial implementation supported in Cronus Release 1.4 did not attempt to use any of Mach's multiprocessing capabilities. The **arctic** and **examine** programs were not ported, since Mach 2.0 on the Sun 3 did not support the SunView

toolkit. The Bug Report Manager, which contains experimental Cronus software described later, was not ported either.

*3.1.2.3 Sun 4 / SunOS 4.0*

The Sun 4 is a supercomputing workstation family, designed to offer high performance in a wide variety of compute-intensive and graphics applications. It is based on Sun Microsystems's RISC (Reduced-Instruction Set Computer) implementation known as SPARC (Scalable Processor Architecture).

- CPU — Fujitsu SPARC MB86900 or Cypress SPARC CY7C601 (MB86910 and Weitek 1164/1165 or TI8847 floating-point units optional)
- Estimated MIPS — 7 to 16
- Operating System — SunOS

The implementation of Cronus for the Sun 4 was similar to that for the Sun 3. Most notable on the Sun 4 was Cronus's use of the SunOS 4.0 *lightweight process* (LWP) package to implement the Cronus tasking package. Both offer similar functionality, and we chose to maintain Cronus portability across constituent operating systems by maintaining the same upper layer tasking package interface and using the LWP package as a lower level implementation tool. This isolated the Cronus tasking package from the vagaries of the compiler's register assignments and other RISC architecture influences.

### 3.1.3 Notable Enhancements

*3.1.3.1 Kernel*

For Release 1.4, we performed a major internal restructuring of the Cronus kernel to improve its modularity, simplify the addition of new code to the kernel in this and future releases, and make the kernel more portable to new environments.

A number of operations on the object types managed by the Cronus kernel were added or improved. The Cronus kernel object types **CronusHost** and **Process** were extended to support the **ReadACL** and **ManagerStatus** operations. In addition, the **ReportStatus** operation on type **CronusHost** was enhanced to return two additional parameters: a string containing the hostname, and a string containing the current revision level of the Cronus kernel and a brief description of the system hardware. The latter is particularly useful at sites that run a mix of different Cronus releases on different systems simultaneously. To provide increased security in the Cronus kernel, the **Create** operation on object type **Process** was enhanced to require that the invoker be a member of the Cronus access control group *CronusOperators*. Finally, the type definition and documentation for the Cronus kernel were updated to reflect the access control restrictions on operations on the kernel object types.

Circa Cronus 1.4, many large sites started segmenting network traffic using a technique known as subnetting, wherein a single-class IP network is broken into a number of distinct physical networks. For example, without subnetting the Class B IP network 128.11.0.0 would be thought of as a network consisting of $2^{16}$ hosts. Having that many hosts on a single network is unmanageable, so the network might be broken into (for example) $2^8$ networks, numbered 128.11.0.0 to 128.11.255.0, of $2^8$ hosts each. These networks are

known as subnets, and are generally connected using gateways. Previous releases of Cronus treated networks solely by the network class (A, B, C) implicit in the network number, ignoring the possible existence of subnets. Cronus Release 1.4 enhanced the Cronus kernel and the Broadcast Repeater to allow the use of subnets, and to allow the Broadcast Repeater to bridge subnet boundaries for object location.

Finally, we made modifications to the Cronus kernel in Cronus Release 1.4 so that interhost interprocess communication links are managed more effectively in large configurations with many hosts. In conjunction with changes made to the Cronus Host Poller, this improved the overall scalability of the Cronus monitoring system.

*3.1.3.2 Commands*

In Cronus 1.4, we added to and extended the Cronus command set. The new and changed commands are summarized as follows.

A new interactive command, **defineservice**, was added to register a new manager with the Cronus Configuration Manager.

The new command **displayhost** retrieves the information pertaining to a particular Cronus host from the Configuration Manager.

The **displayservice** command was rewritten to provide a more comprehensible output format.

The **examine** command was extended to support examining objects (**examine obj**) and access control lists (**examine acl**).

The **gendoc** command was altered to write its output to the standard output instead of a file (file output is still possible with the new **/output** qualifier), and to allow the generated manual pages to be labeled with an arbitrary manual section name using the **/tag** qualifier.

The **genmgr** command was altered to generate the C code for the manager development tools rather than the Type Definition Manager, with the result that the COS Directory Manager need not run on the host on which genmgr is being run. A new qualifier to genmgr, **/clientonly**, allowed the generation of only the files necessary to build a client program for a particular object type. Another new qualifier, **/operationhierarchy**, allowed the generation of a summary showing the operation inheritance for the object manager.

The new **gethostlist** command obtains a list of hosts and their IP addresses for the local Cronus site from the Cronus Configuration Manager, suitable for use as the Cronus hostfile.

The new **gettypelist** command obtains a list of Cronus object type numbers and their names for the local Cronus site from the Type Definition Manager, suitable for use as the Cronus typefile.

The **showkernel** command as of Cronus 1.4 was modified to support a **/status** qualifier that returns information about the requested host's Cronus kernel version and hardware type, uptime, logging level, and a process and IPC link summary. As a result, the uptime, logging level, and process summary information were removed from **showkernel /connections**.

The **tropic** command was enhanced to support internal variables set to user-provided values through the **setvar** tropic command.

*3.1.3.3 Manager Development Tools*

For Cronus Release 1.4, we made a number of enhancements to our application development tools. These included evolving the version voting replication facilities provided initially in Release 1.3 to improve their functionality, and improving the scalability of Cronus object databases.

In Cronus Release 1.3, the version voting replication mechanism had difficulty handling simultaneous changes to an object, even when a sufficient quorum was collected. This resulted in difficulty in changing newly created objects, discarded or overwritten updates, and inconsistent version vectors. For Release 1.4, the problem was fixed by adding a deadlock-free distributed locking protocol to the replication mechanism, which properly serializes the changes during the vote collection and commit phases. Also for Release 1.4, we improved the update daemon used in the replication software so that it is more efficient, uses broadcast less, and does not interfere with other simultaneous updates. Third, we eliminated all of the important race conditions in the version voting replication software, which should provide for much more predictable operational behavior. Finally, we instrumented the code to allow the replication software to be monitored with the Cronus Monitoring and Control System.

In order to support the storage of more data in Cronus object databases, two changes were made to the Cronus manager development tools. The object database routines for Cronus managers were altered to maintain their index of available objects using a hash table that is sized at manager startup time. Previous releases used a fixed number of buckets in the hash table, causing very inefficient behavior when the number of objects in the database was large. Second, an artificial limitation on Cronus object size to a maximum of $2^{16}$ bytes was lifted, with the result that objects can be as large as $2^{32}$ bytes on most machines.

*3.1.3.4 Libraries*

In Release 1.4, we made a number of additions and changes to the Cronus libraries to improve their functionality. The new library routines are summarized below.

The routine **GetRootDirectory** returns the value of the environment variable CRONUS_ROOT.

The routines **INTERVALtoDOUBLE** and **DOUBLEtoINTERVAL** convert Cronus INTERVAL (time difference) structures to and from C double precision floating-point numbers, allowing values to be easily scaled over a large dynamic range.

The routine **IsGroupEnabled** determines whether a particular group is contained in the current access group set.

The routine **IsLoggedIn** determines for the caller whether the invoking process has an access group set other than NotLoggedIn.

The routine **STRINGtoOBJECT** performs a parameterized conversion to change its input string into a Cronus UID.

The routines **OpenConnection**, **IsConnectionOpen**, and **CloseConnection** support the direct connections facility discussed earlier.

The **ListObjects** routine returns a list of the objects managed by a particular object manager.

We also made improvements to the status checking facilities for in-progress operations that we introduced in Cronus Release 1.3. In the initial release of this software, the status check messages for long duration operation invocations sent using low-effort message delivery were sent reliably. In order to be more consistent, in Cronus Release 1.4 status checks for operation invocations sent reliably are also sent reliably, and status checks for operations sent low-effort are also sent low-effort. The new scheme for status checking low-effort invocations retains the advantages of status checking, and is much more efficient.

*3.1.3.5 System Managers*

In Release 1.4, we modularized the functionality in the Type Definition Manager. Previous versions of the manager supported both data storage and code generation functions, and were non-reentrant due to limitations in the underlying tools used to build the language parser. In order to easily support the handling of multiple target languages (C, Common Lisp, Ada, etc.) as well as multiple simultaneous users, we moved the code generation facilities contained in the manager into a set of code-generating client programs, one per target language (for example, the **genmgr** command). The Type Definition Manager continued to act as a repository for stored type definitions.

In many cases it is desirable for an operation to be associative, manipulating a collection of objects that meet some criteria. Previous releases of Cronus required that operations needing this functionality be hand-crafted. The developer of a manager used library routines to step through the object database by UID, with hand-written code evaluating each object against the particular criteria of interest. Alternatively, the instance variables for the generic object were used as an index for the objects that the manager maintained, with the generic object acting as a lookup table. Both of these approaches were inefficient, required customization for each manager in which they are used, and were hard to maintain.

In Release 1.4, we added experimental **associative access query processing** facilities to the Cronus **Bug Report Manager**. Included was the ability to create indices from and perform searches on the various fields in an object. The indexing functionality significantly improved the ability to rapidly search the manager's database, and greatly increased its usefulness for large collections of bug reports. The experimental facilities in the Bug Report Manager were designed to be easily incorporated into other managers in the next Cronus release. The additional operations supporting associative access build indices (**AddIndex**), destroy indices (**DropIndex**), display the schema (**ShowSchema**), and allow SQL (Structured Query Language)-like searches to be performed (**Query**). In addition to the changes to the Bug Report Manager made to support associative access, the Cronus **bug** and **examine bug** commands were modified as part of this experiment to use the Query operation for searching the object database (instead of the hand-coded Retrieve operation previously provided).

*3.1.3.6 Installation and Operation*

For Cronus 1.4, the Cronus UNIX installation scripts were entirely rewritten from csh to the Bourne shell sh for increased portability to new systems.

The installation scripts for 1.4 correctly recognized a wider selection of host types automatically during installation. In particular, the installation procedures discriminated between various Suns: the Sun 2, 3, 4, and 386i. Previously all Suns were identified as being Sun 2's.

As of 1.4, it was also possible to install Cronus binaries, libraries, and data files in alternate locations. Previous Cronus releases for UNIX systems required that the top of the Cronus hierarchy be called */usr/cronus*. Cronus Release 1.4 allows the location to be selected using the environment variable **CRONUS_ROOT**.

*3.1.3.7 Documentation and Support*

Release 1.4 expanded the documentation set delivered in earlier releases. We added an *Introduction to Futures* to the tutorial documentation, and included a number of additional examples in the Cronus manual set. Documentation was provided for the new commands and library routines introduced, and known mistakes in the documentation were corrected. Portions of the Installation Manual we rewritten.

For supported sites, a Cronus Hotline was introduced to support fast-turnaround responses to use, programming, and operations questions. The hotline, at (617) 873-2111, was an adjunct to the existing electronic mail-based help and bug reporting provided via the addresses cronus-help@bbn.com and cronus-bugs@bbn.com respectively.

## 3.2 Cronus Release 1.5 - May 1, 1990

In Cronus Release 1.5, we added a number of mechanisms that support more sophisticated data access capabilities for Cronus applications, significantly increased the number of supported platforms, and continued our commitment to make Cronus more robust and easier to use.

Two major new capabilities added in Cronus Release 1.5 supported associative access to data used by Cronus applications. First, the addition of *query processing* supplemented the toolkit that is used in the development of Cronus object managers. This capability allowed clients to invoke operations which identify collections of objects by attribute instead of by their Cronus unique identifier. Queries are posed using a subset of the standard Structured Query Language (SQL), that is used by many relational database systems. The second added capability supports associative access to data via the integration of three off-the-shelf relational database products: Informix, Oracle, and Sybase. Starting in Cronus 1.5, client programs can access any of these databases through a set of newly developed object managers.

In Cronus Release 1.5, we added the AT&T 6386, BBN Butterfly GP1000, DEC RISC family (for example, the DECstation 3100), and Encore Multimax to our suite of supported host types. We also made substantial improvements to the performance and reliability of the Cronus implementation for the Mach operating system.

### 3.2.1 New Features

#### *3.2.1.1 Query Processing*

In many cases it is desirable to have an object manager implement an associative operation that manipulates a collection of objects meeting some criteria, for example "Give me a list of all of the blue objects". Previous releases of Cronus required that this functionality be implemented by application developers on a case-by-case basis: the application used library routines to step sequentially through the object database, and application-specific code that found objects with the particular criteria of interest was executed. Alternatively, the generic object for a type was used as an index for the instances of that type that the manager maintained. This approach required hand-crafted code to maintain the index. More generally, neither of these approaches made it particularly attractive to change or extend the indices used, and neither provided software reusability.

In order to provide a general-purpose associative access mechanism, in Release 1.5 we enhanced the Cronus object database facility to include associative access features. Working from existing standards, we provided SQL-like query capabilities whereby client programs can formulate ad-hoc queries and send them to the appropriate object manager for processing. This allowed much more flexible queries than when associative access code is custom-crafted.

The query processing facility added in Release 1.5 was provided through a new object type called **CronusQuery**. This is a subtype of another new Cronus type **SQLDefs**, which in turn is a subtype of type **Object**. All application types that are subtypes of type CronusQuery inherit the ability to support associative access. The CronusQuery type implements the **ShowSchema**, **AddIndex**, **DropIndex**, and **Query** operations that support query processing. (The SQLDefs type will be discussed more in the next section.)

For an application type that is a subtype of CronusQuery, the application developer defines a schema describing the external view of the data maintained by the manager (akin to the instance variables, but a public view rather than an internal one private to the object manager). At run-time, queries may be posed to the object manager, subject to the constraints of the schema. This is done through the **Query** operation, which takes an SQL-like select statement, and returns to the invoker any data matching the query (subject to access control restrictions). Query operations may be executed at a single object manager, or at a collection of object managers of the same type (the former is the default). In the latter case, the manager initially receiving the operation acts as the coordinator for the query operation, distributing the query to other managers for the type and assembling the results for return to the client. (In Release 1.5, the distributed Query operation was only available for non-replicated types.)

The query results returned are represented using Cronus' canonical datatypes. Since each manager has its own schema, and each Query operation may return a different set of datatypes, the data returned (rows of typed values, or **tuples** in database terminology) must be dynamically datatyped. As a result, to complement the query processing facility, we also introduced in Release 1.5 a **self-describing tuple** facility allowing clients to interpret and manipulate these self-encoding data structures.

Several other operations support additional query processing features. The schema defined for an object type may be examined using the **ShowSchema** operation. Queries are normally handled in the manager by stepping through the object database to determine which objects match the request. In order to improve performance, the user can build indices using the **AddIndex operation**, which will decrease the amount of time necessary to scan the collection of objects. The index is automatically managed as objects are modified, and as objects are added to or removed from the manager's object database. A **DropIndex** operation is provided for removing indices.

*3.2.1.2 Database Support*

An underlying goal of the Cronus toolkit philosophy is to support a variety of facilities that satisfy distributed application needs, including data storage and retrieval. For example, one mechanism provided for the storage and retrieval of Cronus objects is the object database. In Cronus Release 1.5, we expanded the number of data storage and retrieval options by providing direct support for accessing relational database management systems through Cronus. We set the following objectives for the DBMS integration:

- provide host-independent access by Cronus clients to several commercially available state-of-the-art relational database products;
- provide Cronus client programs with the full functionality of the database system, including multistatement transaction and schema manipulation operations;
- define a structure that, to the extent possible, provides a standard interface to the different database products, and facilitates the support of future database systems.

To fulfill these goals, in Cronus Release 1.5 we provided a set of new Cronus managers that encapsulated the various commercial relational databases. We added nine new Cronus object types to support this capability.

Most significant were the **SQLDefs**, **DBDefs**, and **DBSession** object types, which define and implement the set of operations that are available across all of the supported relational databases. For each database supported, we also defined an object type that is a

subtype of DBDefs and an object type that is a subtype of DBSession. These subtypes implement a small number of database-specific customizations that accommodate differences between the various database products. In Cronus 1.5, we supported the Informix, Oracle, and Sybase database management systems.

When a client wishes to interact with a database through Cronus, the database of interest is found or created, then opened, using one or more of the operations defined by DBDefs, for example:

| | |
|---|---|
| **ListDatabases** | List the names of all known databases. |
| **LookupDatabase** | Find the database with a particular name. |
| **CreateDatabase** | Create a new database. |
| **OpenDatabase** | Open a database. |

When the database is opened, the database manager hands off the responsibility for all subsequent actions on behalf of the particular client to a **Session** object. For example, if a client opened an Informix database, for further activities the Informix manager would refer the client to an InformixSession located on the same machine. For Session objects, each instance (session) is under the control of an independent constituent operating system process (as shown below). This is in keeping with the model that most DBMS systems apply, and facilitates handling activities such as transactions: each independent user of the database is a separate process. (This is different from the existing Cronus model where all object instances at a single machine are managed by a single object manager.) The front-end processes representing Sessions pass client requests on to the database system.

Having established a session, the client may choose to perform work as part of a transaction, in which case it would use an operation defined by type DBSession:

| | |
|---|---|
| **BeginWork** | Begin a transaction. |

Next, the client might execute a query or some other SQL statement using an operation defined by type SQLDefs:

| | |
|---|---|
| **ExecuteSQL** | Execute an arbitrary SQL statement. |

This operation provides the majority of the remote database access functionality, such as row insertion, select, row update, and so on. (These operations are also available to Cronus object type **CronusQuery**, which supports the query processing capabilities discussed in the previous section.)

After one or more ExecuteSQL operations, if the client was using transactions it may specify what to do with the transaction results:

| | |
|---|---|
| **RollbackWork** | Abort the transaction. |
| **CommitWork** | Commit the transaction. |

These operations are defined by type DBSession. Eventually, the client will be finished and execute another DBSession operation:

| | |
|---|---|
| **Close** | Terminate the session. |

Thus, the collection of operations defined by the SQLDefs, DBDefs, and DBSession object types allow for remote database access using Cronus. Specifically, a series of actions to manipulate an Informix database (without using transactions) might be:

| | |
|---|---|
| **LookupDatabase** | A generic operation on type Informix which returns the object UID of the database desired. |
| **OpenDatabase** | A non-generic operation on the Informix database of interest which opens the database and returns the UID of an assigned InformixSession. |
| **ExecuteSQL** | A non-generic operation on the InformixSession which executes an SQL statement. |
| **Close** | A non-generic operation on the specified InformixSession which closes the database and terminates the session. |

The database software is delivered partly as source code and partly as object code, due to the licensing restrictions of the database interface libraries provided by the DBMS vendors.

*3.2.1.3 Mach*

The Mach multiprocessor operating system was first supported by Cronus in Release 1.4, shipped in September 1989. Between Cronus 1.4 and 1.5, we made several changes which offer substantial benefits over the initial implementation, specifically:

- faster, more robust communication between processes and the local Cronus kernel;
- support for multiprocessing in Cronus object managers; and
- use of Mach-provided mechanisms for communication and tasking.

The Cronus 1.4 implementation of intra-host communication for Mach was built using the User Datagram Protocol (UDP). This mechanism is used on all of our UNIX-derivative constituent operating systems to provide communication between the Cronus kernel and other Cronus processes (both clients and managers) on the same machine. UDP provides Cronus with a flexible, uniform communication interface on many host types and operating systems, which lowers the costs of porting and maintaining Cronus. However, the performance of UDP sometimes compares unfavorably with other local interprocess communication (IPC) mechanisms supported by the constituent operating system. Such is the case with Mach, which provides an IPC mechanism called **ports**. To take advantage of this performance difference, Cronus 1.5 featured a completely new implementation of intra-host communication for Mach that used the port mechanism. Performance studies showed that Mach ports are substantially faster than UDP for intra-host communication.

Using ports instead of UDP for Cronus intra-host communication on Mach increased communication robustness in addition to speed. With UDP, messages can be lost between the Cronus kernel and local processes under high message traffic. Since the UDP-based Cronus intra-host communication implementation provides no flow control, processes may lose messages if their buffers are full due to high traffic conditions. The implementation based on Mach ports is much more resistant to message loss, because it provides flow-control between communicating processes.

Most operating systems incur a high process creation cost. That is, creating and beginning execution of a new process is a high-overhead, high-latency activity. In order to allow Cronus object managers to process multiple operation invocations effectively and efficiently without putting too great a load on the constituent operating system, and to minimize this latency, it is highly desirable to allow object managers to maintain multiple, independent threads of control (also known as tasks) within a single constituent operating system process.

Few operating systems provide primitives to support coroutines or lightweight tasks within a process. The Cronus tasking package was developed to address this problem. On most platforms, we implement the tasking primitives defined by the package using an implementation we have designed to be portable to a variety of machine architectures and operating systems.

However, Mach itself supports a package for lightweight threads of control known as **C/Threads**. Thus, in Cronus 1.5 on Mach, we reimplemented the Cronus tasking package using C/Threads, and each Cronus task within an object manager is actually a C/Thread. This structure preserves the semantics of the Cronus tasking interface and portability of Cronus object managers, while taking advantage of the provided implementation. A similar approach is used with the SunOS LWP package.

Since Mach is designed as a multiprocessor operating system we also augmented the Cronus tasking interface to facilitate limited multiprocessing in Cronus object managers. Application developers have access to the parallel processing capabilities of multiprocessors through these new primitives, called **TaskObtainOwnProcessor** and **TaskAbandonOwnProcessor**. These primitives allow CPU-intensive computations to be performed in parallel with additional operation invocations that the manager might receive concurrently.

*3.2.1.4 Mandelbrot Demonstration*

For Cronus Release 1.5, we added a new demonstration application for which we ship a SunView-based graphical client program and a portable manager. The client displays fractals from the well-known Mandelbrot set (see *The Science of Fractal Images*, Heinz-Otto Peigen and Deitmar Saupe, Springer Verlag, New York, NY, 312pp., 1988), which are computed by one or more Fractal Managers. The calculations of the Mandelbrot set have the property that they can be parceled out among a number of computationally-bound managers operating concurrently. One of the key points illustrated by the demonstration software is that Cronus is capable of improving application performance by providing the infrastructure to support coarse-grained parallel processing among a collection of loosely coupled processors.

The Mandelbrot client and Fractal Manager code serve as an additional example in the use of the Futures facility that was first released in Cronus 1.4. The C language source code for both the graphical client and the Fractal Manager are provided with Cronus Release 1.5. (Common Lisp source code for the Fractal Manager was distributed with the Symbolics release.) The client program makes extensive use of Futures both to parcel out the computations to a collection of Fractal Managers executing concurrently, and also to drive the graphical display concurrently with these computations.

The graphical user interface provides two display windows. One displays the Mandelbrot set as it is computed. The other shows various statistics about the computation itself. Users can control the displays, the part of the Mandelbrot set being explored, the statistics monitored, and many other aspects of the demonstration through the user interface.

### 3.2.2 New Environments

For Release 1.5, we ported Cronus to several new platforms. This increased the potential Cronus user community and allowed a wider range of systems to support Cronus applications. This section summarizes the features of the additional platforms supported.

#### *3.2.2.1 AT&T 6386*

The AT&T 6386 workgroup system is designed to function as a powerful single-user machine or a network server, primarily in office automation applications. It combines UNIX functionality with the ability to run multiple MS-DOS applications simultaneously under UNIX. The AT&T 6386 supports the X Window System and accommodates most existing PC/AT peripheral expansion products.

- CPU — Intel 80386 microprocessor (80387 floating-point coprocessor optional)
- Estimated MIPS — 2 to 2.5
- Operating System — UNIX System V/386

The Cronus implementation on the AT&T 6386 made use of the Wollongong WIN/TCP for 386 STREAMS networking software implementation.

#### *3.2.2.2 BBN Butterfly GP1000*

The BBN Butterfly GP1000 is a modular, expandable, general-purpose multiprocessor incorporating 2 to 256 microprocessors interconnected by a high-bandwidth multistage switch. Each processor has up to 4 megabytes of local memory (to which other processors have access) for a total of up to 1024 megabytes systemwide. With a scalable performance range, the Butterfly GP1000 can address a variety of computationally intensive problems, such as those in the simulation, modeling, and data analysis areas.

- CPU — Motorola 68020 microprocessor with 68881 floating-point coprocessor
- Estimated MIPS — 5 to 600
- Operating System — nX

#### *3.2.2.3 DEC RISC*

The DEC RISC family was a recently introduced collection of high-performance, reduced instruction set computer (RISC) architecture systems ranging from the DECstation 2100 to the DECsystem 5840. With its processing performance, this DEC product family is suited to applications in artificial intelligence, computer-aided design and engineering, simulation and modeling, and data analysis.

- CPU — MIPS Computer Systems R2000 (or R3000) processor with R2010 (or R3010) floating-point unit
- Estimated MIPS — 10 to 65

- Operating System Ultrix

*3.2.2.4 Encore Multimax*

The Encore Multimax is a modular, expandable, general-purpose multiprocessor incorporating 2 to 20 microprocessors interconnected by a high-bandwidth system bus to peripherals and up to 128 megabytes of memory.

- CPU National Semiconductor 32x32 microprocessor with 32081 floating-point coprocessor
- Estimated MIPS 1.5 to 15 (32032)
- Operating System UMAX 4.2 or Mach

### 3.2.3 Notable Enhancements

*3.2.3.1 Manager Development Tools*

For Cronus Release 1.5, we made a number of enhancements to our application development tools.

The **definetype** command no longer relied on the COS Directory Manager to transfer the Cronus type definition to be processed to the Type Definition Manager. When this change was coupled with a similar change to the genmgr command for Cronus 1.4, Cronus managers can be built on systems which do not run the COS Directory Manager.

The manager development tools code generator for the C language, **genmgr**, had a number of improvements:

- As of 1.5, it generated typesafe functions that encapsulate the functionality previously provided by **GetVarData** and **PutVarData**. These functions, which take the form **Get<type>Var**, **GetGen<type>Var**, **Put<type>Var**, and **PutGen<type>Var**, made the manipulation of instance variables in Cronus object managers written in C easier. For example, the routine **GetPrinVar** would be used in the Authentication Manager to get the instance variables associated with a principal object.

- In order to allow application developers and system administrators to identify various versions of Cronus object managers more easily, starting with Cronus 1.5 the message "Generated by Cronus Release @ date" (for example, "Generated by Cronus Release 1.5 @ Fri Mar 30 11:08:47 1990") was inserted both as a comment and into a variable called **GeneratedUnder** in the generated file dispatch.c.

- Improvements were made so that redundant entries are no longer generated in dispatch.c files. The smaller size of this file resulted in a corresponding code size shrinkage for Cronus object managers.

The operation processing routines for type Object, which are inherited by all Cronus object managers, were substantially rewritten. Although this revision had no external effect, it made these routines more consistent with the rest of Cronus, and improved the long-term maintainability of the code. The following were the most significant changes:

- The operation processing routines no longer used the MSL directly; rather, they use the tools-generated output structure for reply handling.

- References to outdated library routines were replaced by references to corresponding newer routines, and references to undocumented library routines were eliminated. Error codes were changed to use the new error code naming convention introduced in Cronus Release 1.4.

- A number of unused variables were eliminated.

The inherited **ManagerStatus** operation had two optional return parameters added to it for monitoring resource usage: **SecondsCPUUsed** and **KBytesMemoryUsed.**

Finally, some languages and tools (for example, parallelizing compilers) require that the main program of an application be compiled using a special tool or with a particular set of options. In previous releases, this sometimes caused a problem for Cronus managers, where the main program came from a library (i.e., mgr.a on UNIX). In Release 1.5, we isolated the main routine for Cronus managers into a single module which calls the routine **cronus_mgr_main**. This allowed the driver program for Cronus managers to be redefined if necessary, but permitted complete compatibility with earlier releases if no special processing was required.

*3.2.3.2 Library*

The routine **CacheInit** and the remainder of the cache management package were improved to make better use of dynamic memory. In Release 1.4, memory was wasted by applying more stringent data structure alignment than was necessary.

The **Nack** routine was improved to automatically include a printable error string as part of the error reply that may be returned from a manager to a client program. This improved the performance of failed operation invocations (when the message is displayed) by eliminating the need for the **GetErrorDefinition** operation in most cases.

*3.2.3.3 Kernel*

In a previous release, we added a command-line qualifier to the Cronus kernel that implemented a software loopback for object location. This was done because not all networking software implementations heard themselves when they transmit the IP broadcasts that are used for Cronus object location requests. In Release 1.5, we automated this software loopback feature, obviating the need for the qualifier (except as an override). When the kernel boots, as of Cronus 1.5 it tests to see if the host hears its own broadcasts, and automatically enables broadcast loopback if necessary.

The **ReportStatus** operation on type **CronusHost** was enhanced so that it returns details about the operating system version that the machine is running, in addition to the host type. (This is displayed by the Cronus command **showkernel /status.**)

*3.2.3.4 System Managers*

The **Bug Report Manager** distributed in earlier releases only ran on the Sun 3 running SunOS. In Cronus Release 1.5, we enhanced the Bug Report Manager so that it will run on all supported UNIX platforms and VMS.

In Cronus Release 1.5, the **Configuration Manager** was upgraded to use the associative access query processing capabilities described earlier. Both the HostData and ServiceData object types implement the operations described earlier by virtue of being redefined as

subtype of type CronusQuery. Schemas were also defined for both of these object types. A Query operation directed to the Configuration Manager to identify all of the machines on a particular network would look like:

*select Name from HOSTDATA where HostAddress matches "128.89.*.*"*

Similarly the **Host Poller Manager** also used the associative access capability as of Cronus 1.5. For example, this Query operation identifies the names of the machines that the Host Poller thinks are down, and the last time that they replied to a status check from the Host Poller:

*select host, lastreply from POLLER where state = "DOWN_HOST"*

Or this query might be used to identify those machines that have a large amount of variability in the time that they take to answer polls, and also have a relatively large average delay in responding:

select host from POLLER where maxdelay > mindelay * 10 and avgdelay > .2

*3.2.3.5 Commands*

In Cronus 1.5, we added to and extended the existing Cronus command set. The improvements simplified Cronus configuration management (**monitorhosts**, **showstatus**), allowed use of the associative access features added (**showschema**, **tropic**), and generally provided improved functionality. In this section, we discuss the various command enhancements that we made.

The **dumpdb** command had an additional **/output** qualifier added, allowing its output to be dumped into a file.

Using a terminal-independent interface, the **monitorhosts** command displayed the status of various machines as determined by the Host Poller Manager.

The **remove** command was updated to use the qualifier package that we added in an earlier release, and was merged into the top-level **cronus** command.

The new **showschema** command displays the columns defined by a particular type that implements the associative access capabilities: column name, whether the column is indexed, what the datatype of the column is, and whether it is an array.

The **showstatus** command was significantly improved. In addition to displaying the service-specific information that it displayed in previous releases, showstatus now also displays the service's configuration (what machines it runs on) and service-independent status information (e.g., how many operations the manager has performed and how much CPU time it has used).

The **tropic** command had a number of new features added:

- Self-describing tuple support was added.
- A new variable, called **ReplyingHost**, indicates which host responded to the last operation invoked. This is useful in directing future invocations to that host, if desired.

- The **newtype** command was enhanced to print out the name of the host which returned the type definition requested.
- Three new built-in commands were added:
  - **showhistory** displays the contents of tropic's ten-command history buffer.
  - **write** allows the value of a stored variable to be put into a specified file.
  - **pipe** allows the value of a stored variable to be input to another command.
- The new **prompt** option can be used to change the prompt string.
- The new **showerrordata** option enables the display of detailed error messages, for those managers that return data as part of their error indication in addition to the standard error codes.

### 3.2.4 Installation and Operation

The installation scripts were improved to support multiple configurations.

The installation scripts were rewritten to use the logical names that are defined for VMS more consistently.

The installation of the Primal File Manager was made an option to the PRISTINE installation in Release 1.5., as it was not a required service.

### 3.2.5 Documentation and Support

Release 1.5 expanded the documentation set delivered in earlier releases.

We added a **Query Processing tutorial** that described building Cronus object managers that use the new associative access facilities. Additional pages in the Cronus Programmer's Reference Manual complemented the tutorial.

We added a tutorial describing the **Mandelbrot Demonstration.**

A new **Database Installation Manual** was added that describes the installation of the database managers.

New commands and features added to existing commands were documented in the Cronus User's Reference Manual.

## 3.3 Cronus Release 2.0 - March 31, 1991

This section describes the features and functions of Cronus Release 2.0, and the benefits of 2.0 over 1.5.

Cronus Release 2.0 contained the following major changes:

- The Cronus kernel and IPC system were substantially rewritten, with performance improvement the primary goal.
- The authentication system was revamped to increase its effectiveness. The redesigned system made use of MIT's Kerberos network authentication system.
- The Directory Manager was rewritten using the Manager Development Tools.
- The Common Lisp implementation of Cronus was updated with respect to emerging Common Lisp standards.
- New installation mechanisms simplified the installation of Cronus and its applications.

In Cronus Release 2.0, we discontinued support for the Primal File Manager. We believed that complementary technologies exist which provide similar functionality in a networked environment (e.g., NFS).

In Cronus Release 2.0, we supported basically the same set of supported host types as in Cronus 1.5, with these changes:

- We added support for the Alliant FX/80 multiprocessor running the Concentrix operating system.
- We discontinued support for the AT&T 6386 running System V/386. This particular Cronus implementation was not in sufficient demand to warrant its support.
- We added support for the Sun 3 running SunOS 4.x, and discontinued support for SunOS 3.x. SunOS 4.0 was initially available in mid-1988; SunOS 4.0.3 was first available in mid-1989; and SunOS 4.1 was available in mid-1990. As of the release of Cronus 2.0, we believed that our customers are no longer using SunOS 3.x.
- Since the performance of SunOS 4.0 was generally acknowledged to be poor on Sun 2 platforms, we continued to support all SunOS releases newer than SunOS 3.4 on that platform.
- We added support for the Symbolics platform running Genera 8.0.1 and subsequent releases, and discontinued support for Genera 7.x. Genera 8.0 was available starting in May 1990; Genera 8.0.1 was available starting in September 1990. As of Cronus 2.0, all Cronus users were no longer using Genera 7.x.
- We discontinued support for the Xenix operating system. As of Cronus 2.0, this particular Cronus implementation was not in sufficient demand to warrant its support.

### 3.3.1 New Features

#### *3.3.1.1 Cronus Kernel and IPC System*

Compatibility between Cronus releases at the level of the Cronus IPC system was provided from 1986 to Cronus 2.0's release in early 1991. While significant new features were added prior to 2.0 (futures, status checking, direct connections, etc.), our desire to leave existing protocols and mechanisms alone restricted our ability to make these mechanisms work efficiently. Measurements made during 1990 showed that significant performance benefits could result from redesigning and reimplementing parts of the kernel and IPC system. As a result, in Cronus Release 2.0, we performed the following kernel and IPC enhancements.

- Most Cronus operation invocations transit very well-known code paths, defined by generated code and by libraries called by generated code. However, the Cronus 1.5 implementation of the Cronus IPC system was excessively layered, containing many functions which could be combined without loss of generality. As a result of this excessive layering, most operation invocations traversed extra software and suffered from degraded performance. In addition, the old Cronus IPC system predated the Manager Development Tools (and code generation) and contained many low-level communication primitives that were rarely used by applications directly, such as **Invoke** and **Receive**. For Cronus Release 2.0, we restructured the Cronus kernel and IPC system to:

  - Eliminate several layers in the runtime IPC library. In addition to improving performance, this made the system easier to understand and port.

  - Implement and make use of expandable buffers. This greatly reduced the cost of layering, by eliminating unnecessary data copying when a new layer's header is added. When a communication buffer is initially allocated, it is oversized. Then, it is packed from the back rather than the front. Each layer adding a header prepends it to the front of the buffer, rather than reallocating a slightly larger buffer and copying an existing one.

- In the past, Cronus intrahost IPC provided communication between the Cronus kernel and other Cronus processes (both clients and managers) on the same machine. The implementation of intrahost IPC on UNIX systems used the User Datagram Protocol (UDP). UDP was used because it was widely available - this lowered the cost of porting and maintaining Cronus. However, since UDP provided no flow control or acknowledgments, messages sometimes were lost between the Cronus kernel and local processes, usually when a machine was heavily loaded. In Cronus Release 2.0, we defined and implemented a *fragmentation protocol*. The new protocol transfers data in acknowledged fragments between the kernel and local processes. This significantly increased the amount of data that can be exchanged by the kernel and Cronus processes (many UDP implementations set a limit of 8192 bytes), and also eliminated data loss for these large transfers.

- Status checking was first introduced in Cronus Release 1.3, as a means for detecting failed operations, and for differentiating between failed and long-lived operations. Status checking was provided by the addition of a Request Manager to the Cronus kernel, and by required client activity to check operations in progress. In Cronus Release 2.0, responsibility for operation invocation status checking was transitioned to the Cronus kernel. Cooperating kernels on a client's and a target manager's machine maintain information about operations in progress. When a failed operation is detected,

client programs are automatically notified. The 2.0 approach allowed a higher degree of accuracy in the detection of lost or dropped operation invocations, and required no client participation.

- Prior to 2.0, Cronus object managers could choose to forward selected operation invocations that they received, for example to implement load balancing. In Cronus Release 2.0, the IPC and kernel mechanisms supporting forwarded operations were redesigned and reimplemented. Because of the changes made to status checking in 2.0, the information kept by the kernel about the status of operations in progress was valuable in implementing forwarding. In particular, Cronus Release 2.0 allowed the forwarding of larger amounts of data than Cronus Release 1.5, and correctly status checks these forwarded operations.

- Cronus Release 2.0 contained significant improvements to the direct connection mechanism first released in Cronus 1.4. Direct connections allow the Cronus kernel to be bypassed for successive communications between a client-manager pair. This can result in communication performance approaching that of the underlying transport system (currently TCP/IP). In Cronus Release 1.4, direct connections were built using a previously existing Cronus mechanism known as large messages. This large message facility had a number of severe limitations, and was dropped in Cronus Release 2.0. In 2.0, direct connections were redesigned and reimplemented independently of large messages, so that connections could be established and accepted by either a manager or a client, and so that direct connections were very easily usable by application programmers.

- Prior to 2.0, Cronus limited the amount of data that could be transferred in a single message between two Cronus kernels to 1400 bytes. Larger messages were sent using the large message facility, which transferred the data over an independent TCP/IP connection. Under circumstances where the amount of data transferred was just slightly over the 1400 byte limit, the cost involved in establishing this new connection vastly outweighed the performance benefit of sending the data over a separate connection. In Cronus Release 2.0, the Cronus kernel was enhanced to exchange arbitrarily sized messages directly. This allowed flexibility in the decision of when to use direct connections, and when to send data via the Cronus kernel.

- Complementing this enhancement, under Cronus Release 2.0 the Cronus kernel itself was enhanced to send and receive larger messages. In particular, in previous releases, the Cronus kernel was unable to fully implement operations that required data transfer to or from the kernel in excess of 1400 bytes. For example, a kernel could only return the first 50 entries in its object location cache in response to a *DumpObjectCache* operation (which is invoked by the **showkernel /cache** command), even though the cache itself was much larger. Since the Cronus Release 2.0 kernel is able to send and receive arbitrarily large messages, previous restrictions on operations invoked on object types managed by the kernel were eliminated. This change improved the function of operations on type **Cronus_Host** and **Primal_Process**.

For Cronus Release 2.0, the Cronus kernel and IPC system was transitioned to its officially assigned TCP and UDP port numbers. The previous Cronus IPC system made use of ports clustered around port 2000 (for default installations). This occasionally resulted in collisions with other services using the same port numbers (for example, Sun's NeWS product also uses port 2000). Cronus Release 2.0 and later releases require only the use of TCP and UDP port 148 for default installations.

The aggregation of all Cronus communication from a set of ports to a single port required the multiplexing of a variety of protocols for various types of Cronus messages. We implemented five for Cronus 2.0:

| | |
|---|---|
| PROTOCOL_LOCAL_OP | messages contain operation requests or replies in transit between a client or manager and its local Cronus kernel. |
| PROTOCOL_FRAG_OP | messages contain operation requests or replies in transit between a client or manager and its local Cronus kernel. PROTOCOL_FRAG_OP is used when a request or reply is too large to transfer using intrahost communication mechanisms via PROTOCOL_LOCAL_OP. |
| PROTOCOL_REMOTE_OP | messages contain operation requests or replies in transit between Cronus kernels. |
| PROTOCOL_PM | messages pass control information between a program and the local Cronus kernel's Process Manager. PROTOCOL_PM messages perform functions such as registering programs as Cronus processes, deregistering these programs when they exit, and telling the Process Manager that a given process manages a particular type. |
| PROTOCOL_OS | messages pass control information between Cronus kernels. PROTOCOL_OS messages perform functions such as interhost communication link management and status checking. |

*The changes made to the kernel and IPC system for Cronus 2.0 resulted in a performance improvement on a typical machine of 25% for operations sent via the Cronus kernel, and 100% for operations sent via direct connections.*

*3.3.1.2 Authentication System*

All activities in Cronus are capable of being access-controlled. When a Cronus user executes the Cronus **login** command, his identity is verified by the Cronus Authentication Manager using a principal (user) name and password combination. Subsequently, when the user executes Cronus (or application) commands, object managers automatically check his identity (by asking the Cronus kernel) and verify that he has permission to perform the requested operation (by checking the target object's access control list).

For Cronus Release 2.0, we redesigned and reimplemented the Cronus Authentication Manager to use concepts developed for the Kerberos network authentication system. Kerberos is a system developed by MIT's Project Athena to provide a robust identification system for a medium-scale distributed computing environment, where neither hosts nor users can be trusted. Kerberos in turn is based on a design by Needham and Schroeder (Using Encryption for Authentication in Large Networks of Computers, Communications of the ACM, December 1978), with timestamps added to prevent playback. The major design goals for Kerberos were:

- No cleartext passwords should be sent over the network.

- No cleartext client passwords should be stored on network servers.
- Client and server keys should only be minimally exposed.
- Compromises should only affect the current session.
- Authentication lifetimes should be limited.
- System should function transparently during normal use.
- Minimal modification should be required of existing network applications.

One of the biggest weaknesses present in the Cronus Release 1.5 authentication system was the Cronus login command, which passed passwords over the network in the clear. This problem was solved by our incorporation, in Cronus Release 2.0, of some of Kerberos's mechanisms into the Cronus Authentication Manager. Other features which have come out of the adoption of some of Kerberos's methods are as follows:

- Processes are unable to masquerade as the Cronus kernel, or as object managers. All clients, managers, and the kernel have clear identities.
- Clients and managers are able to communicate via end-to-end encrypted operations. (This will be implemented in a future release.)

*3.3.1.3 Directory Manager*

The Cronus 2.0 Directory Manager dated from before the Manager Development Tools. As a result, was not implemented using the tools. Although was extremely reliable, the Directory Manager had the following limitations.

- It was not built using the same structure and organization as other Cronus object managers. This made it harder to understand, maintain and improve.
- It was implemented with many of its own private implementations of routines normally found in the Cronus libraries. In particular, the Directory Manager had its own implementation of intrahost IPC and part of the Cronus Message Structure Library (MSL).
- The Directory Manager used two special primitive canonical datatypes (EDIR and ELIST) that required special consideration, and were a nuisance in maintaining interoperability with the COS Directory Manager.
- The Directory Manager did not implement many standard operations such as SetLoggingLevel, ManagerStatus, and DescribeType, and had to implement its own version of others, such as AddToACL.
- The Directory Manager implemented a non-standard replication scheme.

For Cronus 2.0, we reimplemented the Cronus Directory Manager using the Manager Development Tools. In addition to resolving the above limitations, we did the following.

- We added the ability to delegate portions of the directory namespace to other Cronus object managers. For security reasons it is desirable to maintain the mapping between

Cronus principal names and their corresponding UID's in the Authentication Manager. However, it is also desirable that this namespace be accessible through a standardized naming convention. We designed and implemented the new Directory Manager to allow a lookup of, for example, *:prin:jdoe* by using the Directory Manager to resolve the first part of the name (*:prin*) and then having the Directory Manager handoff the remainder of the lookup (*jdoe*) to the Authentication Manager. It is our intent that applications implementing the appropriate operations should be able to assume control for portions of the namespace as well. This facility is also used to delegate portions of the namespace to the COS Directory Manager.

- We refined the Directory Manager's interface specification to alleviate the need for hand-crafted client interface routines, in favor of the more standard *Invokexxx* routines. The semantics of the previous suite of operations defined and implemented by the Directory Manager were not clear. This complicated the use of a generated client interface, and required the use of hand-crafted client interface routines.

### *3.3.1.4 Test Manager*

Beginning in Cronus Release 2.0, we started distributing a new **Test Manager**. This manager integrates various functional tests that have been used internally at BBN for verifying some of Cronus's capabilities, as well as a number of other tests. We included the Test Manager's source code in Cronus Release 2.0. In addition to providing an enhanced testing capability for hard-to-diagnose failures encountered at user sites, the Test Manager gives application developers additional examples in performing a variety of functions in a Cronus manager. The release 2.0 Test Manager contains tests for:

- verifying correct data transfers of various sizes between clients and managers;
- detecting message loss;
- validating the representation of floating point data in Cronus canonical form across heterogeneous hosts
- testing Cronus functions for signaling errors to client programs;
- checking the correct operation of nested operations;
- validating implementation of forwarding;
- exercising Cronus status checking capabilities for long duration operations;
- gathering basic performance data; and
- testing access control.

### *3.3.1.5 Common Lisp Implementation*

For Cronus Release 2.0, we updated the Common Lisp implementation of Cronus to accommodate the new Cronus IPC, authentication, and Directory Manager changes. In addition, the Lisp implementation was enhanced to support two relatively stable elements of the draft ANSI Common Lisp language: the Common Lisp Object System (CLOS), and the Pitman condition system. Both of these mechanisms were then available in the major Common Lisp implementations from Symbolics, Texas Instruments, Lucid (Sun), and Franz.

Our usage was based on the descriptions presented in Guy L. Steele's *Common Lisp: The Language, Second Edition (Digital Press, 1990).* CLOS replaced Flavors as the mechanism for operation invocation and manager implementation. Some minor changes to existing operation implementations (defopr forms) was required, including use of **with-slots** and use of **slot-value** rather than **symbol-value-in-instance**.

In keeping with the shift from the Flavors to CLOS paradigms, the **cronus-send** message passing form of operation invocation was obsoleted by a set of generated generic functions with names of the form **co:<operation-name>** which take both required and optional input fields as keyword arguments (to satisfy CLOS requirements for lambda list congruence). For example, the form

```
(cronus-send {acdb} :authenticate-as "joe" "p")
```

became

```
(co:authenticate-as {acdb} :user-name "joe" :password "p")
```

Existing applications that use cronus-send were converted.

*3.3.1.6 Installer*

For Cronus 2.0, we overhauled the installation procedures used for Cronus. We have implemented and distributed an **installer** program, which supports the initialization of Cronus on a machine, and the installation of both system and application object managers. The installer interprets a documented installation language; installation scripts have been provided for the system object managers. Cronus application developers can implement scripts to install applications. Template scripts are provided.

A benefit of the installer is the ability to use the same installation scripts on a variety of platforms, significantly simplifying maintenance of the installation procedures. The installation procedures for Cronus 1.5 and previous releases were implemented using the command interpreters provided by the native operating systems on which Cronus was run. This meant that installation functionality had to be duplicated, and parallel installation procedures had to be maintained, for each of the command interpreters.

### 3.3.2 New Environments

For Release 2.0, we ported Cronus to one new platform.

*3.3.2.1 Alliant FX/80*

The Alliant FX-series of multiprocessors are bus-topology machines constructed using two types of processor elements called Interactive Processors (IP's) and Advanced Computational Elements (ACE's). IP's are responsible for running the operating system, interacting with users, and providing access to I/O and other devices; ACE's are used for non-interactive CPU-intensive processing. IP's support a base instruction set and registers; ACE's support the base instruction set and registers, plus additional floating point, vector, and concurrency instructions and additional registers. The FX/80 is a VMEbus-based system, supporting up to 12 68020-based IP's, 8 ACE's, and 256 megabytes of physical memory. The Alliant operating system, Concentrix, is a multiprocessor-enhanced implementation of 4.3BSD UNIX.

- CPU — Motorola 68020 microprocessor with proprietary 64-bit CMOS coprocessor
- Estimated MIPS — 20 to 130
- Operating System — Concentrix

### 3.3.3 Notable Enhancements

*3.3.3.1 Manager Development Tools*

In Cronus Release 2.0, **genmgr** was altered to represent enumerated canonical datatypes as C enums rather than as C typedefs and #defines. In other words the type definition fragment:

```
cantype FLAVOR
     representation is Flavor:
     {VANILLA, CHOCOLATE, COFFEE};
```

in Cronus 2.0 generated C code that looks like:

```
typedef enum _Flavor {
     VANILLA = 0,
     CHOCOLATE = 1,
     COFFEE = 2
} Flavor
```

instead of:

```
typedef unsigned int Flavor
#define VANILLA      ((Flavor)0)
#define CHOCOLATE    ((Flavor)1)
#define COFFEE       ((Flavor)2)
```

which was generated in Cronus 1.5. This change was done because virtually all C compilers now support the C enum datatype. Since C enum datatypes can't have duplicate enumerated values, this enhancement provided stricter type-checking for Cronus applications written in C.

In Cronus Release 2.0, **<mgr>enum.c** files were no longer generated by **genmgr**. Instead, their contents are included in <mgr>cts.c files.

In Cronus Release 2.0, we made the **Makefile.skl** and **descrip.skl** files generated by the **genmgr** command use the value of the environment variable CRONUS_ROOT (CRONUS$ROOT for VMS) as the value of the Make (MMS) variable ROOTDIR.

The tasking package, which supports handling multiple threads of control within an object manager, was enhanced to provide more debugging support. A number of routines were added to the library to print out the status of tasks in progress, and more detailed error messages are printed when error conditions are encountered.

*3.3.3.2 Library*

Two routines, **CronusTypeUsedOften**, and **CronusTypeNotUsedOften** were implemented to advise Cronus that operation invocations may or may not be made frequently by a client or manager on a specified object type. These routines are used by the Cronus IPC system in determining when it is useful to establish direct connections.

A new set of functions were provided to allow iteration over the objects in a manager's object database. These routines replaced the existing **DBNext** routine with a more flexible interface allowing iteration in either internal hash table or physical order. The latter provided a significant performance improvement when a manager handles a large number of objects.

A new routine, **MgrSelect**, was provided in the UNIX manager library. MgrSelect mimics the behavior the UNIX **select** system call, blocking the calling task in the manager on activity for one or more specified file descriptors. This enhanced the ability of Cronus object managers to communicate with other UNIX processes.

A new function was added to the manager library, **SelectLocalObjects**, that can be used within an object manager to execute a query on objects in the manager's local object database.

In previous Cronus releases, the Cronus library lacked a way to conveniently grow space allocated with Talloc (in the way that the C run-time library routine realloc allows space allocated with malloc to be resized). In Cronus Release 2.0, we added a routine **Trealloc** that performs this function.

A new set of routines in Cronus Release 2.0 was provided for obtaining information about incoming operation invocations in object managers (for example, the unique identifier of the source process). Previously this capability was provided by references through complex data structures. The new mechanism provides a more easily understandable interface with a higher level of data abstraction.

In Cronus Release 2.0, we provided a package of routines for manipulating an obj[illegible]'s version vectors. This simplified the construction of administrative programs for replication.

*3.3.3.3 System Managers*

We implemented a **TaskStatus** operation on type Object. The TaskStatus operation replaced the ShowTasks operation on type Object, which was specified but never implemented. TaskStatus allows client programs to obtain information about the run-time state of various execution threads in a Cronus object manager.

The MCS Manager ran on VMS as of Cronus Release 2.0. The MCS Manager was updated to garbage collect old event data. In Cronus Release 1.5 and previous releases, the MCS Manager grew until it could no longer allocate memory, then crashed.

The Oracle Database Manager was internally reworked for Cronus 2.0 to use Oracle's Pro*C interface rather than the somewhat older Oracle Call Interface (OCI). This improved performance when querying a database through the Oracle Database Manager and allowed strong typing of numeric datatypes specifying size and scale information.

*3.3.3.4 Commands*

In Cronus Release 2.0, we added two new administrative commands, **addservice** and **removeservice**, to simplify the installation and deinstallation of Cronus managers. The addservice command tells the Cronus Configuration Manager that a particular manager is supposed to run on a particular machine. The removeservice command tells the Configuration Manager that a specified manager is no longer to run on a particular machine.

The **arctic** and **examine** commands were upgraded to work with the X Window System on Sun platforms under Cronus 2.0. That is, both arctic and examine were made usable under either X or SunView. In reimplementing these commands using X, our hope was that as X stabilizes on other platforms, we will be able to support these programs on additional machine types.

The **arctic** command was improved to use the Cronus futures mechanism. The Cronus 1.5 implementation of arctic used some low-level IPC facilities to keep the user interface alive and await messages from the Cronus MCS Manager simultaneously. Migrating to futures (which were developed after the initial implementation of arctic) increased the portability of the client and manager.

The **createprin** command was improved to better handle the specification of user's full name, directory, etc. by the elimination of the need to use the editkeys subcommand.

We integrated a number of commands that existed as standalone programs (**copyfile, list, repeater, unroll**) into the top-level **cronus** command. This saved on disk space in the installed system.

Beginning with 2.0, the **definetype** command issued a warning if a type definition with the same number is used to overload an existing type with a different name (but goes ahead and does it anyway).

The **dumpobject** command was integrated into the **fixobject** command and became **fixobject /dump**.

The **displaygroup** command was enhanced in 2.0 to get a list of all valid Cronus principals through **displaygroup world**.

The **fixobject** command's user interface was improved to make it somewhat easier to understand. In addition, fixobject was enhanced to easily allow the complete dereplication of all objects at a particular host. (This significantly eased reconfiguration.)

The **fixdir** command was obsoleted by the migration of the Cronus Directory Manager to use the manager development tools, The functions previously performed by fixdir were subsumed by fixobject.

A new command **obtainlists** was added. The obtainlists command updates a machine's local copies of configuration information kept in the Cronus hostfile and typefile. This information is obtained from the Configuration and Type Definition Managers respectively.

A new qualifier **/all** was added to the **repeater /status** command. This allows system operators to locate and display the status of any Broadcast Repeaters in operation.

A new **showtypes** command was implemented. This command shows the use of various Cronus object types within the operating environment: which types have been defined in

the Type Definition Manager, which types have been installed (are known by the Cronus Configuration Manager), and which types are currently available.

In Cronus Release 2.0, we added to the **tropic** command a **showtiming** option. Enabling showtiming allows application developers to meter and display the execution time of Cronus operations.

A new Cronus command in 2.0, **verify** allowed the automatic checking of the consistency of the available commands with the command dispatch file.

### 3.3.4 Installation and Operation

The Cronus kernel was modified in Cronus Release 2.0 to update the Cronus hostfile and typefile automatically from the Configuration and Type Definition Managers respectively when Cronus boots on a machine. This simplified the maintenance of these files.

### 3.3.5 Documentation and Support

Release 2.0 expanded the documentation set delivered in earlier releases.

- We added documentation and a tutorial on the new Cronus **installer** program.
- We added a tutorial on the **tropic** command.
- We documented new commands, features that were added to existing commands, and new programming interfaces in the Cronus User's Reference Manual, Operator's Reference Manual, and Programmer's Reference Manual.

## 3.4 Cronus Release 3.0 - December 1, 1992

Cronus Release 3.0 contained the following major changes:

- We significantly improved Cronus's capabilities for building large scale applications in interorganizational internetworked settings.
  - We extended the Cronus kernel and IPC system to support the autonomous but cooperative operation of Cronus clusters.
  - We concomitantly enhanced the configuration, authentication, and directory services.
  - We improved Cronus's resource location mechanisms in wide-area network environments.
- We added support for delegation, a facility supporting the redirection of requests from a manager to one or more local subprocesses. This feature is particularly useful for significantly improving object manager performance on multiprocessor systems.

The set of supported hosts changed from Cronus Release 2.0 to Release 3.0 as follows:

- We added support for the BBN TC2000 multiprocessor running the nX operating system.
- We added support for the NeXT workstation running the Mach operating system.
- We added support for the development of Common Lisp applications (clients and managers) on the Sun 4 using Lucid Common Lisp.
- We discontinued support for Masscomp 680x0-based systems running RTU (i.e., the Masscomp 5400 and 5500). We believed these platforms to be obsolete.
- We discontinued support for the Sun 2 platform. We believed this platform to be obsolete.
- We discontinued support for the Sun 3 running the Mach operating system. This particular Cronus implementation was not in sufficient demand to warrant its continued support. (Our users of this platform had migrated to the NeXT.)
- We discontinued support for building Common Lisp applications on the Symbolics workstation. (Users of this platform had migrated from the Symbolics in favor of the better price/performance offered by running Lisp on Sun workstations.)

Additionally:

- We discontinued support for the COS Directory Manager. (Although it was still shipped - for backward compatibility - it was no longer documented.)

It was our intent that the installation and use of Cronus 3.0 not disrupt existing Cronus users and applications. This was accomplished primarily by making the underlying changes transparent through library routines, the Cronus Manager Development Tools, and other layering techniques. From an application developer's perspective, most of the

changes were made at low levels of abstraction in the Cronus implementation. Thus, most applications ran on Cronus 3.0 after merely recompilation and relinking

### 3.4.1 New Features

#### *3.4.1.1 Cronus Multicluster Enhancements: Kernel and IPC System*

Before Cronus 3.0, independent organizations wishing to share resources with each other faced several difficulties. First, the organizations had to "link up" their separate Cronus installations into a single, inter-organizational environment. Unfortunately it was not possible to do this in a limited way: if an organization shared one of its services, it would allow access to all of its services. Second, Cronus only supported the notion of a single configuration service for a distributed environment. The configuration service plays an important role in Cronus; it serves as the depository and source of information concerning services, hosts, and what hosts run which services. If organizations linked up, they would have to give up their existing separate configuration services, and instead use a single shared configuration service. But by doing so, each organization would give up control over its own assignment of services to hosts, that is give up some autonomy over its own configuration. (A service is the function provided by one or more cooperating object managers that manage the same object types. The configuration service is provided by the set of cooperating Configuration Managers.)

Two organizations linking up would also encounter analogous problems with other services, for example the authentication service. To share authentication information with another organization, an organization had to give up autonomous control over its principals and groups. Similar problems occurred with the directory service.

With Cronus Release 3.0, we introduced the concept of **clusters** as a mechanism to support the inter-organizational sharing of resources. Simply put, *a cluster is a set of hosts grouped together into a single administrative unit.* Each cluster is autonomous, responsible for its own administration and control.

Physical network layout does not have to play a part in determining host cluster membership. Two hosts on the same physical local area network can belong to different clusters. Two hosts widely dispersed and connected by a long-haul network can belong to the same cluster. However, no host is permitted to be a member of more than one cluster.

Clusters support organizations wishing to cooperate and share distributed services. But each cluster can still retain autonomy and control over its own part of the distributed environment. With clusters, Cronus supports the notion of several independent and separate services for configuration, authentication, and naming within a single Cronus environment. Each organization can have its own set of these services, and therefore maintain complete autonomy over its own assignment of services to hosts.

Clusters are intended to allow administrative units to maintain control over their part of a Cronus environment, yet share services across cluster boundaries if so desired.

##### *3.4.1.1.1 Sharing Services via Exports and Imports*

A cluster is not an isolated unit: clusters can cooperate and share services with one another. If a cluster supports a service, it can permit other clusters to access that service. If a cluster does not permit access to a service, then operation requests from foreign clusters on objects managed by the service will be rejected. If the cluster does permit access, the request is

accepted as long as the requester has the necessary access rights as specified on the object's access control list.

Each cluster must explicitly enumerate which foreign clusters have access to its services. A service to which access by a foreign cluster is permitted is called an exported service, or simply an export. This service is said to be exported to the foreign cluster. A cluster must explicitly declare that it wishes to access a service exported by a foreign cluster. An exported service to which a cluster desires access is called an imported service, or simply an import. This service is said to be imported from the foreign cluster.

If a client in cluster B wishes to access a service S in cluster A, then A must export the service to B and B must import the service from A. If the service is imported but not exported, the service will reject any access attempt from B. If the service is exported but not imported, hosts in cluster B will not be able to locate any of the objects.

#### *3.4.1.1.2 Sharing Services via Service Domains*

A cluster can permit objects managed by its services to be replicated in foreign clusters. This is a stronger notion than merely permitting access to a service from foreign clusters. Permitting objects to be replicated across cluster boundaries implies that the service itself spans cluster boundaries, and thus several clusters must cooperate with regards to the administration of the service. In this case, the cluster must explicitly declare the foreign clusters where objects of one of its services may be replicated. The foreign clusters are called the domain of the service.

For a replicated service to run successfully across cluster boundaries, all clusters involved in the replicated service must agree on the service's domain. If cluster A lists B as part of the domain of service S, then B must list A as part of the domain of S. Similarly, if A lists B, and B lists C, then A must also list C. If a foreign cluster is a member of a service's domain, the service is implicitly considered to be exported and imported to the cluster.

#### *3.4.1.1.3 Configuration Manager Changes*

In addition to providing configuration control within a cluster, for Cronus 3.0 the Cronus Configuration Manager was extended to maintain information about a cluster's imports, exports, and service domains.

Since we defined a cluster as an independently managed entity, each cluster must run its own configuration service, and may not have any foreign clusters in the service domain of the configuration service. This restriction ensures that every cluster maintains autonomous control over its service configuration. (The right to modify and create objects managed by the configuration service should be a closely guarded right, restricted to a few trusted and authorized individuals, and never granted to individuals whose "home" is a foreign cluster.)

The Cronus 3.0 Configuration Manager manages three types of objects: Host_Data objects, Service_Data objects, and Cluster_Data objects - Cluster_Data is new.

A cluster's configuration service has a Host_Data object for each host that is a member of the cluster. As in earlier releases, the Host_Data object for a given host contains the host's name, address, and a list of Service_Data object identifiers identifying the services installed on the host.

A Service_Data object records information about a service. As in earlier releases, it contains the service's name, the set of the types the service manages, and the set of Host_Data objects identifying the hosts where the service is installed. Starting with Cronus 3.0, it also contains identifiers for three sets of Cluster_Data objects. The first identifies the clusters the service is imported from, the second identifies the clusters the service is exported to, and the third identifies the clusters in the service's domain.

A Cluster_Data object stores information about a foreign cluster. It contains the cluster's symbolic name and its unique cluster identifier. It also contains identifiers for three sets of Service_Data objects. The first identifies the services imported from the cluster, the second identifies the services exported to the cluster, and the third identifies the services whose domain includes the cluster. The Cluster_Data object also contains a list of the IP addresses of one or more of the hosts belonging to the remote cluster. (This list provides a list of "contacts" in the remote cluster. As will be discussed, the multicluster location algorithm finds an object in a remote cluster by delegating the task to some host in the remote cluster. The host is chosen from the host list maintained in the Cluster_Data object.)

#### *3.4.1.1.4 Object Location*

Cronus supports location-independent invocation; the location of an object does not have to be supplied when invoking an operation on it. The system takes complete responsibility for finding a copy of the object and directing the invocation request to it. The part of the system that locates objects is called the Locator, and is part of the Cronus kernel.

The old (Cronus 2.0 and earlier) Locator was very simple, and as a result suffered from some limitations. To find an object, it would broadcast a Locate invocation on the object. The broadcast would be heard by every host on the network. If a host had a copy of the target object, it responded. If a host did not have the object, it did not respond. The Locator would wait for an affirmative response. If one was not received within about five seconds, the Locator reported a LOCATE_FAILED error.

The old Locator also relied on Broadcast Repeaters in Cronus installations encompassing more than one network. Routers do not forward broadcasts; so broadcasts on one network are not normally heard on another. The Broadcast Repeater and Rebroadcaster (both parts of the Cronus kernel) cooperated to re-broadcast Cronus object location requests to a set of configured hosts on specified networks.

The old Locator suffered from several problems and limitations:

1. It was not robust. The old Locator relied on a single broadcast to find an object. In some environments it is not uncommon for the network to drop broadcast messages.

2. It did not scale well, since it made every host process every locate (except when using cached object locations).

3. It complicated the maintenance and administration of a Cronus environment, by requiring the use of Broadcast Repeaters between networks.

4. The arbitrary five second wait for Locate replies was not appropriate for widely dispersed or heavily loaded systems.

We established the following major requirements for the new multicluster Locator design.

1. The Locator must be robust: it must not fail to locate an available object. If an irrelevant host (not one where the requester or object resides) crashes, the location attempt should not fail.

2. The Locator must be efficient: it must avoid communicating with hosts that do not manage the selected object type.

3. The Locator should find objects in a timely fashion.

The Locator was completely redesigned and reimplemented for Cronus 3.0 and satisfies these requirements. (The old Locator failed to satisfy the first two.)

The new Locator worked as follows, using the unique identifier (UID) of the target object. The Locator first extracts the type of the object from the UID. It then identifies the service that manages the type as follows. The Locator first checks a cache it maintains of service information. If this is unsuccessful, the Locator contacts the local cluster's configuration service. The configuration service keeps track of which services manage what types. The configuration service sends back information about the proper service, which the Locator then stores in its cache.

The information about a service that the configuration service returns to the Locator (and that is cached) includes the following: (1) a list of the hosts in the local cluster that run the service, (2) a list of the clusters the service is imported from, (3) a list of the clusters in the service's domain, and (4) a list of hosts in each cluster mentioned in items (2) or (3). This information comes from data stored in the configuration service's relevant Service_Data, Host_Data, and Cluster_Data objects.

The Locator then makes use of the list of hosts in the (local) cluster that run the service. The locator sends each one of these hosts a Locate request. (The Locate messages are transmitted using UDP, an unreliable protocol. The Locator times out and retransmits them to guard against message loss.)

The Locator then makes use of the rest of the information about the service to search for the object in remote clusters. A host in every cluster the service is imported from is sent a **ProxyLocate** request. Likewise, a host in every cluster in the service's domain is also sent a ProxyLocate request. The ProxyLocate operation is invoked on the generic Cronus_Host object; the UID of the object being located is included as an argument. The Cronus Kernel manages the generic Cronus_Host object; every Cronus Kernel manages such an object. (Every host runs the Cronus Kernel.) A Cronus Kernel that receives a ProxyLocate request looks for the specified object in its own cluster, and then sends back a reply indicating the object was found and its location, or a negative reply. The ProxyLocate request is sent using a reliable protocol (TCP).

For example, sa a service is imported from cluster RL. The service information includes a list of some of the hosts in cluster RL, which includes the host "x.rl." The Locator chooses "x.rl" from the list and directs a ProxyLocate request to it. (The Locator's choice of a host from the list is arbitrary.) The Cronus Kernel on x.rl then has its Locator look for the object in just its own cluster. x.rl's Locator follows the same steps discussed above to accomplish location. Its Locator sends out Locate requests to each host in its cluster running the relevant service and collects the responses. If all are negative, it sends a negative reply back to cluster BBN's Locator. If one is successful, a positive reply to the ProxyLocate is transmitted immediately. If the attempt to send the ProxyLocate to x.rl fails

because it is down, cluster BBN's Locator selects another host from its list of hosts in cluster RL, and tries again.

BBN's Locator waits for responses to its Locate and ProxyLocate requests. The Locator stops with a successful outcome when the first positive reply is received. If all the replies are negative, the Locator becomes suspicious that its cached service information is out of date. If the service was installed on an additional host since the information was cached, or if the service was imported from an additional cluster, and if the object resides on the added host or in the added cluster, the Locator will obviously fail to find it. The Locator checks if its cached service information is out of date by calling upon its local cluster's configuration service. The configuration service maintains an ascending modification count for each service. The Locator stores the modification count for each service in its cache. Checking if the cached information is out of date is a simple matter of comparing the actual versus the cached modification count for the relevant service. If the cache is out of date, it is updated, and the Locator tries to locate the object anew. If the cache proves to be up to date, the Locator reports back a LOCATE_FAILED error. A host processing a ProxyLocate also similarly checks if its cache is out of date if it fails to find the requested object.

#### *3.4.1.1.5 Invocation Request Delivery Options*

Cronus supports location independent invocation. A client does not have to supply any location information when invoking an operation on an object. The system itself finds the object and routes the invocation request to it. The Cronus manager development tools generate RPC stubs that send operation requests to managers. The INVOKECONTROL argument in these stubs allows control over the instances eligible to receive a particular invocation. For Cronus 3.0, we expanded the use of the INVOKECONTROL structure. When an INVOKECONTROL structure is provided, three fields specify the instances eligible to receive the invocation: *HostUse*, *Host*, and *Cluster*. The interpretations of the Host and Cluster fields depends upon the value of the HostUse field. The semantics of the choices for the HostUse field are given below.

| | |
|---|---|
| **HOST_ANY** | Any object instance can receive the invocation; the values of the *Host* and *Cluster* fields are ignored. This is the default behavior. |
| **HOST_HINT** | Like HOST_ANY, but *Host* specifies a host where the sender believes the object is. |
| **HOST_DIRECT** | Only the host specified by the *Host* field is eligible to receive the invocation. |
| **HOST_ANY_IN_CLUSTER** | Any object instance residing in the cluster indicated by the *Cluster* field can receive the invocation; object instances in other clusters are ineligible. |
| **HOST_HINT_IN_CLUSTER** | Like HOST_ANY_IN_CLUSTER, but *Host* specifies a host where the sender believes the object is. |
| **HOST_ANY_IN_DOMAIN** | Any object instance residing in the service domain of the object's type is eligible to receive the invocation. Any object instance not residing |

| | |
|---|---|
| | in the local cluster's service domain for the type is ineligible to receive the invocation. |
| **HOST_HINT_IN_DOMAIN** | Like HOST_ANY_IN_DOMAIN, except *Host* specifies a host where the sender believes the object is. |

In addition to directing an invocation to a single object instance, one might want to broadcast a message to several instances of an object. The following HostUse field values specify that an invocation request is to be distributed to several object instances and gives control over the set of instances eligible to receive the request:

| | |
|---|---|
| **HOST_ALL** | A copy of the invocation is sent to all instances of the object. |
| **HOST_ALL_IN_CLUSTER** | A copy of the invocation request is sent to all instances of the object in the cluster specified by the value of the *Cluster* field. |
| **HOST_ALL_IN_DOMAIN** | A copy of the invocation request is sent to all instances residing in the local cluster's service domain for the object type. |

Reliable delivery is not provided when a request is broadcast to several instances. The request might be received by some eligible instances, but not by others.

*3.4.1.2 Authentication System*

Cronus 3.0's multicluster enhancements were designed to provide authentication (the assurance that when two parties interact with one another, each is certain of the other's identity) and privacy (the protection of data against unauthorized release) above and beyond that provided in earlier releases. There are three types of security for requests and their associated replies in Cronus 3.0, listed in order of increasing safety. The *Security* field of the INVOKECONTROL structure is used to specify a request's security. The field may contain any of the following values:

| | |
|---|---|
| **SEC_NONE** | Self-explanatory. (This is the default for messages sent unreliably.) |
| **SEC_AUTHENTICATED** | Messages contain an authenticator. (This is the default for messages sent reliably.) |
| **SEC_PRIVATE** | Messages contain an authenticator and the data portion will be encrypted. |
| **SEC_DEFAULT** | The default for the type of message being sent. |

Note: Although it was included in Release 3.0, this feature was considered experimental. Because we believe additional work is required to provide the desired functionality, we neither established nor documented an installation procedure that enabled this feature. It is presently unavailable in the Lisp implementation.

*3.4.1.3 Directory Manager*

The directory service supports a hierarchical symbolic name space for Cronus objects. The leaves of the name space are entries containing arbitrary object UIDs. Pathnames are formed by separating the component names with ":" characters. For example, the path name *:cronus:config:hosts:pats* might resolve to the identifier of a particular Host_Data object, presumably the one that contains information about the host named *pats*.

In Cronus 3.0, to ensure that a cluster has autonomous control over its own directory name space, the service domain of the directory service is limited to a single cluster. The same directory cannot be stored in two different clusters. However, one can mount a remote cluster's directory name space. This makes the remote cluster's entire name space appear to be hung off an arbitrary path name in the local cluster's name space. *Mountpoints* were initially introduced in Cronus 2.0, but have been refined in Cronus 3.0.

Mountpoints simplify accessing remote cluster directory name spaces. Without mountpoints, users would have to explicitly specify the cluster whose directory service should resolve a given path name. When mountpoints are used, path names can always be submitted to the local cluster's directory service. Mountpoints also permit symbolic links from the local cluster's name space into the remote cluster's name space to be readily supported.

Briefly, mountpoints were implemented as follows. Mountpoint entries in the directory name space contain the UID of the root directory in the remote cluster, and an indication that it is a mountpoint, rather than a normal directory. When the directory service encounters a mountpoint when resolving a pathname, it invokes an operation on the foreign root directory to resolve the rest of the pathname.

*3.4.1.4 Delegation*

Included in Cronus 3.0 was a new **delegation facility**. Using delegation, a Cronus manager process may spawn multiple instances of itself. Incoming operations are then automatically distributed among the child processes for execution. On machines with more than one CPU, the child processes can execute operations simultaneously, achieving greater operation processing throughput. (We assume that the native operating system transparently schedules multiple active processes onto idle processors when possible.)

The delegation package was implemented via a new Cronus type **Delegation_Object**. Managers using this facility must implement their types as subtypes of Delegation_Object. Little additional code is needed: some code is needed to initialize the delegation package when the manager starts up, and linking with one additional library (delegation.a) is required. Delegation_Object implements these operations of note:

| | |
|---|---|
| **Shutdown** | Stop child processes |
| **generic StartMoreChildren** | Spawn more processes in addition to those automatically created at manager start-up. |
| **generic DelegationStatus** | Return status information on the object manager and all of its child processes. |

Some operations should not be delegated to child processes. Examples of these operations include ManagerStatus and Locate. The delegation package uses the DelegatableOperation routine to determine if an operation can be delegated to a child process. Except for some

inherited Cronus operations, the default version of this routine allows all operations to be delegated. If a manager wishes to require that some operations by implemented by the parent process, it can provide its own version of this routine.

The number of child processes a manager spawns can depend on a variety of factors. In addition to static information such as the number of processors or amount of memory in a machine, it might also depend on characteristics such as the current load average on the machine or the time of day. The DCM, or **Delegation Configuration Manager**, provides a convenient, flexible way to maintain this information. Configurations are specified by formulas (each stored as an object) written in a simple algebraic language. For example, the formula:

```
default $0 = 0930; /* assume working day */
default $cpus = 1;
if ($0 < 0830) then return $cpus ; end if;
if ($0 > 1730) then return $cpus ; end if;
return 1;
```

specifies that the number of subprocesses spawned should be 1 during the work day, and the number of CPUs on the machine during non-work hours.

In large environments, it might be inconvenient to create a formula for every host that might ever run a manager. In order to address this problem, the DCM provides three levels of matching to determine which formula should be evaluated:

| | |
|---|---|
| host | The formula is for a specific host. |
| hosttype | The formula is for a specific host type, from the BINARYTYPE cantype in config.typedef. |
| default | If no other match is found. |

When finding a formula to evaluate (through either the generic Evaluate or generic Lookup operations), the DCM tries to first find a "host" match. If none is found, it looks for a "hosttype" match. If that fails, it evaluates the "default" formula. The DCM implements these operations:

| | |
|---|---|
| **generic Create** | Create a new formula. |
| **Modify** | Modify an existing formula. |
| **Display** | Return the information stored in a formula. |
| **generic Evaluate** | Find the appropriate formula and evaluate it. |
| **Evaluate** | Evaluate the specified formula. |
| **generic Lookup** | Return the UID of the formula that would be evaluated for this host. |
| **List** | Summarize the formulas stored. |
| **CheckSyntax** | Verify the syntax of a formula provided as a character string. |

At present, the delegation facility is only available on UNIX systems. Also, since it is intended primarily for compute-bound activities, it does not support general access to the Cronus object database.

*3.4.1.5 Commands*

A number of new commands were added in Cronus 3.0

| | |
|---|---|
| **clustedit** | Create a new cluster, display and modify cluster configuration. This command is similar to examine cluster (below), except that it is forms-based rather than being a graphical user interface. |
| **createmount** | Mount the root directory of a remote cluster into the directory namespace of the local cluster. For a remote directory to be mounted successfully, the mounting cluster must import the directory service from the remote cluster, and the remote cluster must export it to the mounting cluster. |
| **createservicemount** | Delegates a portion of the local directory namespace to another service. For example, delegate all lookups in the directory :prin to the Authentication Manager. |
| **dcmedit** | Create, examine, and modify DCM formulas. |
| **displaycluster** | Display summary information about a cluster. |
| **examine cluster** | Browse or alter the services exported and imported by various known Cronus clusters. Since multicluster operation also introduces the possibility of configuration inconsistencies between clusters, the examine cluster command also helps identify these inconsistencies. For example, if cluster A lists B as part of the domain of service S, but B does not list A, an inconsistency exists. (Inconsistencies in service domains can cause replicated services to malfunction.) Alternatively, a cluster could import a service from a foreign cluster that does not export it. (If the service is imported, the administrator probably believes the service is accessible, and might like to be warned if this is not the case.) |
| **setpassword** | This command allows a privileged user to change the password for a selected Cronus principal. It was implemented in response to user comments that it was not possible to use an account if its password was forgotten. |
| **setpeerprin** | Register the principal of a remote peer with the local cluster. (Used for authentication of services in the same service domain but in different clusters.) |

Many other Cronus commands were upgraded to function properly in a multicluster environment, primarily to make sure requests were processed in the proper cluster. In addition, a **/cluster** qualifier was also added to several commands. The qualifier permits

the user to force the command to display information about any desired cluster (the local cluster is the default). For example, the command **displayservice /cluster=bbn** lists all the services defined by cluster **bbn**'s configuration service.

### 3.4.2 New Environments

For Release 3.0, we ported Cronus to two new platforms and one new programming environment.

#### *3.4.2.1 BBN TC2000*

The BBN TC2000 series of multiprocessors are highly scalable, high-performance parallel processing computers. The TC2000 architecture consists of function cards, each of which includes a Motorola 88000 microprocessor, memory, and optional I/O capabilities. The function cards are interconnected by BBN's multistage Butterfly switch technology. The TC2000 supports from 8 to 504 processors, and up to 16,128 megabytes of memory. The TC2000 supports two concurrent multiprocessor operating systems, nX and pSOS$^{+m}$. nX is a UNIX 4.3BSD-compliant operating system developed by BBN and based on the Mach kernel from CMU. pSOS$^{+m}$ is an advanced real-time executive, tuned for time-critical applications.

- CPU — Motorola 88000 microprocessor
- Estimated MIPS — 152 to 9,576
- Operating System — nX

#### *3.4.2.2 NeXT*

The NeXTstation is a high-performance workstation built by NeXT Computer Inc. The NeXT includes substantial hardware support for multimedia: an entry-level system includes a speaker, microphone, and integral audio digital signal coprocessor; other models include additional graphics and video coprocessors. The NeXT's system software includes the NeXTstep object-oriented application development environment, and the Mach operating system (customized for the NeXT).

- CPU — Motorola 68040 microprocessor with Motorola 56001 digital signal processor (Intel i860 graphics coprocessor and JPEG image compression coprocessor included on some models)
- Estimated MIPS — 18 to 25
- Operating System — Mach / NeXTstep

#### *3.4.2.3 Sun / Lucid Common Lisp on Sun 4 / SunOS*

Sun / Lucid Common Lisp is an integrated programming environment built around a complete implementation of the Common Lisp standard. Sun / Lucid Common Lisp includes the Common Lisp Object System (CLOS), the standard for object-oriented programming in Common Lisp.

(Previous distributions of the Lucid Common Lisp implementation for the Sun were made to selected customers. With this release, we added full support for that implementation, including support for the construction of Cronus object managers.)

### 3.4.3 Notable Enhancements and Bug Fixes

*3.4.3.1 Manager Development Tools*

With Cronus 3.0, we added support for unsigned and signed 64 bit integers as primitive datatypes. The names for the canonical datatypes are U64I and S64I respectively. Since 64 bit integers are not supported by most C compilers (nor by most hardware architectures) we defined the internal representation as follows.

```
#ifdef HAVE_LONGLONG_DATATYPE
      typedef unsigned long long                              ui64;
      typedef long long                                       i64;
#else
      typedef struct _ui64 { unsigned long high, low; }       ui64;
      typedef struct _si64 { long high, low; }                i64;
#endif
```

(long long is the emerging standard for the C language's large integer.) This allows the use of 64 bit datatypes on machines that lack native support.

To prevent incompatible versions of clients from inadvertently invoking operations on newer (or older) object managers, in Cronus 3.0 each invocation is accompanied by a checksum which is verified by the manager on receipt of an operation. Invocations on incompatible managers are automatically rejected at run-time. This feature may be disabled by compiling the generated code with the symbolic NO_XSUM defined.

Starting with Cronus 3.0, the format of object database names conforms with the scheme used by most operating systems: **filename.extension**; for object databases, extension is "odb" and filename is the type of object the database contains. Thus, the two object databases managed by the Authentication Manager are now **prin.odb** and **group.odb**, instead of objectdb.20 and objectdb.21 as in Cronus 2.0. Existing object databases will be automatically renamed when a Cronus 3.0 object manager accesses a Cronus 2.0 database.

The manager start-up routines were enhanced to log the amount of free space in the object database, so that a system administrator can determine when it is time to compact the object database. The effect of this and the preceding change is that the startup for the Authentication Manager (for example) now looks appears as follows.

```
*** prin.odb ***
52 entries, 0 free regions
0 bytes out of 26624 free (%0.00)
*** group.odb ***
5 entries, 1 free regions
512 bytes out of 3584 free (%14.29)
free region sizes: smallest 512, largest 512, average 512
```

The implementation of the inherited **ListObjects** operation was changed slightly, in response to user comments. In previous releases of Cronus, ListObjects returned the UIDs for all of the Cronus objects of the specified type known by the target object manager. This frequently caused confusion because the operation returned the UIDs of those objects that were logically deleted (i.e., marked for deletion but not yet fully removed). Starting with

3.0, ListObjects returns only the UIDs of those objects that are currently available. The functionality previously available through ListObjects is now available through a new **ListAllObjects** operation.

Starting with Cronus 3.0, we added code to the **genmgr** command to generate ANSI C function prototypes in the appropriate manager include files.

*3.4.3.2 Library*

Starting with Cronus 3.0, all Cronus library routines conform to ANSI C conventions. Function prototypes for all library routines are included in the Cronus include directory.

A variety of new routines were added to the Cronus library supporting the manipulation of cluster identifiers: **GetMyCLUSTERID, IsMyCLUSTERID, IsSameCLUSTERID, ISAnyCLUSTERID, SetAnyCLUSTERID, CLUSTERIDtoSTRING, CLUSTERIDtoSTRNUM, STRINGtoCLUSTERID, STRNUMtoCLUSTERID,** and **FutureGetReplyingCluster**. Also, **PSST** was enhanced to parse the *%{CLUSTERID}* format string.

The library routines supporting directory management were changed somewhat. **DirParse, DirParseDirectory, DirCreateMountpoint,** and **DirReadMount** were eliminated. The routines **DirGetRoot, DirCreateRemoteDirMountpoint, DirCreateServiceMountpoint, DirEntryTypeToString, DirEntryName, DirEntryVersion, DirEntryObject** were added.

In Cronus 3.0, we added functionality to support the automatic execution of application-specific code when an object is changed by a manager. The routines **DBSPYEstablishMonitor** and **DBSPYCancelMonitor** respectively allow for defining and cancelling handling functions that will be called within an object manager whenever objects are created, updated, or deleted. One example of use of this function is to maintain the in-core object index used by the Cronus query processing code. The new functions were implemented as part of the Cronus object database package.

The Lisp implementation of Cronus 3.0 supports future sets, per a number of customer requests. The functions added to the Lisp implementation were **ready-p, claim, make-future-set, add-future, remove-future, extract-ready-future,** and **future-set-count**; they behave similarly to their C language counterparts. The addition of the claim function also makes it easier to insure that the correct code is executed when a reply is available. The Cronus 2.0-style call:

```
(let ((future (co:operation-name uid inputs :invoke-only t)))
 (multiple-value-bind (outputs)
  (co:operation-name uid :receive-only future)))
```

should be changed in Cronus 3.0 to:

```
(let ((future (co:operation-name uid inputs :invoke-only t)))
 (multiple-value-bind (outputs)
  (claim future)))
```

Old style calls will continue to work.

For Cronus 3.0, we included some additional routines for debugging tasking-related problems. The routine **TaskPrint** prints out a list of the tasks active in the manager and

their state. **TaskStack** switches to the environment of the requested task. **TaskDump** lets one see the task descriptor for the indicated task. The routine **TaskBreak** is a dummy routine in which to stop for debugging tasking-related problems.

*3.4.3.3 Cronus Kernel and IPC System*

When a UNIX process (such as the Cronus kernel) is inundated with UDP messages from another local processes, it may drop some. In Cronus 2.0 and previous releases, our UNIX implementation made use of UDP for communication between the Cronus kernel and local processes. For the most part this worked, but occasionally messages were lost. (This behavior was observed in previous releases only rarely, the most common case being when an application used futures to invoke many operations simultaneously.) For Cronus 3.0, *UNIX domain streams* were adopted for local communication on operating systems that support them. This should completely fix the problem (locally reliable communication mechanisms are already used on VAX/VMS and Mach).

In previous releases, when Cronus was installed on a large system it was often difficult to determine where a given instance of Cronus or a particular manager was running. For Cronus 3.0, we added a new operation **GetEnvironment**, implemented by type Object (and hence inherited by all managers) and by the Cronus kernel. The operation returns the value of CRONUS_ROOT, the working directory of the kernel or manager, the process id, and the user name that the process is running under.

In earlier releases, the Cronus kernel could not open more than a relatively small number of IPC connections to other machines (as few as 15) at a time. This restriction was caused by a limit (set by the operating system) on the total number of open file descriptors (i.e., connections, log file, etc.) a process could have open at a time. For 3.0, the Cronus kernel juggles an unlimited number of "virtual" connections on top of a limited number of actual IPC connections. In addition, on UNIX systems the Cronus kernel now attempts to increase the number of file descriptors available to it. Some platforms will allow this limit to be raised as high as 256 file descriptors (SunOS); others are still higher.

*3.4.3.4 System Managers*

In previous releases of Cronus, the Cronus libraries were coded so as to allow principal and group names to be respectively accessed via the simulated directories *:prin* and *:group*. With this release, we used the mount point mechanism introduced in Cronus 2.0 to redirect lookups in these directories to the Authentication Manager. Thus :prin and :group can be manipulated the same as any other Cronus directory.

In previous releases, the directories *:cronus:config:hosts* and *:cronus:config:services* were used to contain symbolic names for **HostData** and **ServiceData** objects respectively. However, these names were created by the installation procedures, separately from the creation of the objects themselves. This occasionally resulted in inconsistencies between the items named in the directory and the objects managed by the Configuration Manager (particularly when an installation failed for some reason). With Cronus 3.0, we used the mount point facility to redirect lookups to these directories to the Configuration Manager. Inconsistencies are no longer possible.

The Pro*C-based **Oracle Database Manager** (introduced in Cronus 2.0) used an incrementally allocated array to hold tuple pointers. For queries involving large numbers of tuples, this algorithm could consume prodigious amounts of memory, since space was not reallocated along the way. In Cronus 3.0, the Database Manager was modified to use a

linked list structure, as the Cronus 1.5 manager did. This decreased the amount of space consumed by a query from O(n2) to O(n) for n tuples.

*3.4.3.5 Commands*

The **bug retrieve** command had three new qualifiers added: **/columns**, **/delimiter**, and **/header**. These allow selected information to be extracted from the Bug Manager for use by other programs. The **/columns** qualifier identifies which fields are to be selected; they will be separated by **/delimiter** (if specified; tab is the default) and preceded optionally by a header (if **/header** is specified).

The **createprin**, **modifyprin**, and **installer** commands were modified to prompt for a new or changed password twice, and to verify that it is entered the same both times. This insures that the user knows what the password is before it is created or changed, and decreases the likelihood that a typographical error creating or changing a password will result in a user not being able to log in.

The crsql command had a number of enhancements:

- The command was extended so that it can now be used with Oracle databases. (Informix continues to be supported.)
- Temporary files are created by crsql in an appropriate place rather than in the current working directory. This eliminated problems with creating and using forms when the user doesn't have permission to write into the current working directory.

The **gendoc** command had a new qualifier added, **/invoke**, which causes the command to generate nroff source files for documenting the client-side interface (Invokexxx, FInvokexxx, FClaimxxx) to a manager. This can be used to generate manual pages like the XXPSL pages in section 3 of the Cronus Programmer's Reference Manual.

In Cronus 3.0, we enhanced **genmgr** with a new **/clientonly** qualifier. Running the command with this qualifier will generate client code only for the selected manager.

The Cronus **installer** had a number of new extensions added to it:

- A new command line option **/xdebug** permits the interactive debugging of installer scripts.
- The built-in variables **clusterid** and **hostid** identify the local cluster and host respectively.
- The new commands **badcommand**, **create cluster**, **create servicemount**, **stopservice** and **update locatemap** were added.
- New options for setting variables were added.

The Cronus **showkernel** command was enhanced to add a new process owner (**/powner**) qualifier. The new version of the command behaves very similarly to **showkernel /processes**, but displays the owner of the process instead of its process UID.

The Cronus **showkernel /cachestat** command was enhanced to display statistics on cache performance, and on performance locating objects on the network. It also maintains

more accurate statistics. The information displayed by earlier versions of the command (cache hits / cache misses) was misleading, because cache misses included lookups of objects that were unavailable (and might not have even existed). The new implementation of statistics gathering in the kernel, and the numbers displayed by the command, are now more representative.

### 3.4.4 Installation and Operation

When objects in a Cronus object database frequently are added, removed, or change size through modification, the database may become fragmented and disk space can be wasted. For Cronus 3.0, we added a new **compactdb** command that will compact an object database by eliminating the free space in the file. This results in substantial savings in disk space.

Occasionally, developers may wish to make changes to the internal representation used for storing the instance variables of objects in an object database. Unless the developer wishes to repopulate existing databases from scratch, these databases will need to be converted from their old format to their new format. For Cronus 3.0, we created a new Cronus command, **genconv**, to assist in this process. The command works by taking as input a configuration file, and old and new versions of the relevant type definition. It then generates code for a conversion program, which can be customized by the developer. The program automatically handles a number of trivial cases without intervention.

Starting in Cronus 3.0, each object database contained an internal version string specifiable by the application. These can be used by application or system developers to insure manager executable / database version compatibility.

### 3.4.5 Documentation and Support

#### *3.4.5.1 Documentation*

Beginning with Cronus 3.0, we distributed the Cronus reference manuals as part of the release in on-line form.

Release 3.0 expanded the documentation set delivered in earlier releases.

- We documented many new commands supporting Cronus multicluster operation.
- We included documentation and a tutorial on the new **Cronus Delegation Configuration Manager**.
- We included documentation on the **crsql** command.
- We updated the *Introduction to Cronus*.
- We documented additional new commands, features that were added to existing commands, and new programming interfaces in the Cronus User's Reference Manual, Operator's Reference Manual, and Programmer's Reference Manual.
- We updated the Mandelbrot tutorial.

The example Mandelbrot client program was redesigned and reimplemented using the XView package. (The previous version used SunView.) It can therefore be built on any platform where the XView libraries are available. We also fixed a number of bugs in the

Mandelbrot client, and added a new feature allowing the user to specify that a particular host have an arbitrary number of invocations outstanding at any given time.

#### *3.4.5.2 Support*

Shortly after the release of Cronus 2.0, we set up a mechanism to ease the distribution of bug fixes to Cronus users. Patches and bug fixes by FTP can be obtained over the Internet from the machine **cronus.bbn.com** (or **pineapple.bbn.com**). Users may log in with user name "anonymous," and should send their electronic mail address as the password to this account (this enables us to track its usage).

In the FTP home directory, the file **cronus-overview** contains a short summary description of Cronus (including references) and the file **CronusFAQ** answers some frequently asked questions about Cronus. Bug fixes and updates are stored in subdirectories named **cronusxx** (where **xx** is the release number) under which are subdirectories for each machine type. The directory **cronusxx/ALL** contains items that aren't specific to a particular platform.

## 3.5 ANM Release 5.3 - September 30, 1993

This section describes those features which were new in ANM release 5.3. Most notably ANM 5.3 added an **Auxiliary View** feature, a **Derived MIB** facility and a **Cronus Congestion Indicator** display. Additional enhancements included several precoded views and some changes made to facilitate features to be delivered in ANM 5.4.

### 3.5.1 New Features

#### *3.5.1.1 Congestion Indicator.*

The **Congestion Indicator** program is an X-window based application which collects and displays various types of Quality of Service (QOS) data for a group of source and destination host pairs. Data is collected continuously at time intervals determined by the user. Once collected, data is represented as a box, which is sized based on the Quality of Service information collected, and then displayed on a grid where the source hosts are listed on the horizontal axis and the destination hosts are listed on the vertical axis. The supplied display supports only Cronus traffic.

The congestion indicator program accesses a collection of script files (suffixed with *.tcl*) at run time. The file "initcongind.tcl" contains initial values for polling interval, minimum threshold and maximum threshold. The program will look for this file first in the user's home directory and, then, in the current working directory. If the file does not exist, the program will use its own initial values for these parameters.

#### *3.5.1.2 Derived MIB (DMIB)*

The machinery to generate information derived from MIB variables was implemented in ANM 5.3. The DMIB functionality is used by the congestion indicator to gather the information necessary to report Cronus congestion.

#### *3.5.1.3 Auxiliary Views*

The ANM user was given the ability to develop useful tabular views through a new **Auxiliary View** facility provided in AMM 5.3. Using an interpretive language, **Tcl**, which eliminates the need for compilation, new views are easily added to the users library. Auxiliary Views includes a library of functions that permit the user to interface with the ANM acquisition machinery and through that interface obtain data directly from the network.

Operational displays that have been developed using the Auxiliary Views libraries were included in this release. They are used as examples of how to write displays using the Auxiliary Views capability; they can also be used with minor modifications to create the users own views.

#### *3.5.1.4 Cronus Configuration*

Two programs that can build Cronus configuration files were included with this release of ANM.

| | |
|---|---|
| *build_cronus_config.sh* | builds a Cronus Proxy Agent configuration file configuring all the hosts in the Cronus cluster. |

| | |
|---|---|
| *build_cronus_nmdb.sh* | builds a Display Manager *nmdb* configuration file configuring all the hosts in the cronus cluster. |

#### *3.5.1.5 Policy Gateway*

Policy Gateway support was added in this release, in a limited fashion. The Interdomain Policy Routing (IDPR) MIB extensions were added to ANM. IDPR MIB objects are writable using the CMU SNMP tools included in this release, but Auxiliary Views that write (send SNMP_SET commands) were still under development. Auxiliary Views to permit viewing IDPR data were incorporated. The views provided used GET functionality only.

#### *3.5.1.6 IP Traffic Tagging*

IP traffic tagging was supported in ANM starting with ANM 5.3. ANM programs will tag all TCP and UDP traffic sent directly by the program. Note that traffic sent indirectly, like Cronus traffic from the Cronus PA, or NFS traffic (perhaps from accessing NFS-mounted files) is not tagged. This feature required a TCP kernel patch which is available from SRI. IP traffic tagging is an optional feature, and may not be applicable to all installattions.

#### *3.5.1.7 FDDI Support*

The FDDI MIB extensions were added to ANM, and an FDDI view was provided with this release. The view obtains those FDDI MIB variables that Cisco, version 9.1, routers implement.

### 3.5.2 Enhancements

As of ANM 5.3, alert message types are output indicating device and interface state changes (such as a device going down). These messages appear in the DisplayManager's alert window. The DisplayManager now writes the alertlog in its startup directory. Also, the alertlog can be rolled, which means that the alertlog is closed, renamed, and a new alertlog created.

ANM can now set the number of ticks per xmit to an SNMP device that is down. The basic PDU timing period can also be set.

SNMP Traps can now be displayed in the alert window, as received by the Carnegie-Mellon University SNMP trap receiver program (snmptrapd). This is enabled through the new "alertcom" nm.init function.

SNMP physical interfaces are now supported by the Display Manager.

An unzoom feature has been added to the Display Manager.

The Display Manager now supports drag and drop editing between views.

### 3.5.3 Changed Features

New alerts have been added to the DM to inform operator when the Proxy Agent or the Router/Merger stops responding.

The Historian now has the ability to timestamp in either GMT or Local Time.

## 3.6 ANM Release 5.4.1 - January 7, 1994

This section describes those features which are new or changed from ANM 5.3 to ANM Release 5.4.1.

### 3.6.1 New Features

#### *3.6.1.1 LPR Support*

Support for a network of **Low Cost Packet Radios** (LPR) was added in this release.

The DisplayManager and the Policy Module Collection were modified to display the point of presence of the LPR Network. The LPR network is represented by a single icon for the network, positioned on the displayed map where the PC Network Interface Unit (NIU) is located. Detailed information on the LPRs in the network is available through supporting Auxiliary Views.

The PC (NIU) was modified to support the ANM interface through use of the SNMP protocol.

#### *3.6.1.2 Host Agent and Quality of Service*

The Quality of Service proof of concept, begun in the last release, was extended to include live data from the monitored network and hosts. To enable the collection of the additional data, a **Host Agent** was developed to reside at hosts in the network and communicate with ANM.

The enhanced QoS process works in the following manner. The user enters a request using a set of metric Indicators. ANM receives the request and formats it into the SNMP requests needed to acquire the information. The requests are sent to the host named in the request. The Host Agent collects the information, either by calls to the host operating system or by spawning processes to obtain the information by direct measurement. The collected information is returned to ANM and then to the QoS process which formats the information for display and presents it to the user in two formats, the original matrix format and a raw data format in an Auxiliary view. For this release, the data collected from the network consists of the last round trip delay between selected hosts, and three CPU load averages.

#### *3.6.1.3 Policy Gateway*

Policy Gateway support was enhanced with this release to support setting policies. Phase 1 Acquisition Machinery was added to the SNMP PA.

Auxiliary Views to permit viewing IDPR data were incorporated. The views provided in this release use GET and SET functionality.

The IP Traffic Type may now be set on a per-SNMP device basis.

#### *3.6.1.4 SNMP Configuration Checking Tool*

There is a prototype textual auxview that compares the configured paths on selected SNMP devices with the live addresses and reports problems. This is very useful to find obsolete configurations where a router has been reconfigured to have some different IP addresses.

The auxview's name is "TAV_compare_paths_allsys.tcl". It takes one optional argument, a simple regular expression naming the SNMP devices to check. It will, by default, check all SNMP devices available from the SNMPPA's to which it is attached. The results of the check will be displayed as text. We recommend you save the output in a file.

### 3.6.2 Enhancements

Selected Display Manager icons have a white crosshatched stipple over the icon. Selected lines have a white dashed line over the line. You can still see the underlying icon or line status color.

The SNMPPA, DM and the PMC were enhanced to support FDDI devices. Two Auxiliary Views are provided to look at the supported variables, the MAC and SMT tables. This feature has only been tested with Cisco routers running version 9.1 of the Cisco routing software.

The script "anm_start_av" was enhanced to remove lock files from the tmp directory when the auxiliary view exits. If the auxview exits abnormally then the lock files may not be removed.

The PMC will signal the DisplayManager if it is working. If the DisplayManager does not receive any data from the PMC after a configurable interval then it will ring the bell and issue a message in its message line at the lower left hand corner of its display.

The DisplayManager supports both fixed-size subnetting and variable-size subnetting. Fixed-size subnetting means that a network is divided into a set of subnetworks of equal size, driven by a single subnetwork mask. Variable-size subnetting means that a network is divided into subnets of different sizes starting at any subaddress.

The ANM User's Guide was heavily revised

Several new Policies were added to ANM.

- Policy *snmp-sys-allif* (SNMP System All Interfaces) is used to poll for the status of an SNMP system and the status of all of its network interfaces. The system's icon is colored yellow if the device is only reachable using a secondary path, if the device has not responded to a few recent polls, or if any of the interfaces is down or in a testing state operationally or administratively. An alert is issued in the DisplayManager's alertlog with a description of the problem.

- Policy *lightstream-sys* (Lightstream System) is used to poll for the status of an Lightstream system, its chassis, and the status of all of its cards. The system's icon is colored yellow if the device is only reachable using a secondary path, if the device has not responded to a few recent polls, if the chassis status bits are set, or if any of the cards has a problem. An alert will be issued in the DisplayManager's alertlog with a description of the problem.

- Policy *lpr-net* (Low Cost Packet Radio Network) is used to poll for the status of an LPR network as seen from the PC attached to it. The icon is colored based upon the lprstatusMonitorState variable in the lprstatus table. An alert is issued in the DisplayManager's alertlog with a description of the problem.

- Policy *fddi-sys* (Fiber-Distributed-Data-Interface System) is used to poll for the status of an SNMP system that also implements the FDDI MIB. The icon is colored yellow if there is a problem with the general system or a problem on the FDDI ring. Problems on the FDDI ring are currently just determined by examining the fddiSMTCFState variable in the fddiSMTTable. An alert will be issued in the DisplayManager's alertlog with a description of the problem.

- Policy *fddi-ip-if* (Fiber-Distributed-Data-Interface IP addressable Interface) is used to poll for the status of an FDDI interface on an SNMP device. The line is colored yellow if there is a problem in the MIB-2 interface operational status and adminstrative status variables or in the fddi MAC table RMT state variable. An alert is issued in the DisplayManager's alertlog with a description of the problem.

A menu item, "Generate Historian Config File", was added to the Auxiliary Views to make the generation of the Historian Configuration File easier for the ANM user.

The SNMPPA now supports source routed paths. This allows you to configure the route that the SNMPPA will take to an SNMP device. This is useful in situations where routing in the network has failed.

### 3.6.3 Changed Features

The format of the alerts printed by the DM was changed.

A new alert was added to the DM to inform the operator when the Router/Merger stops responding.

The Historian now has the ability to timestamp in either GMT or Local Time.

The **inf_storer** (Informix Storer) now has a feature to allow stuffing data from flat text (ASCII) files into a Informix database and then exiting. This feature allows data to be collected and initially stored via the copy_storer into a large file and then easily stuffed into an SQL database.

In a Historian configuration file, there is now the ability to turn off writing attribute names in Historian output files. This feature was requested by some ANM User's because the flat text files so produced will be smaller and use less disk space.

# 4 Software Development Procedures

This section summarizes some of the software development procedures used under the Cronus-related portion of the activities described here.

Quality control and configuration management procedures are integral parts of the Cronus software development process. Our approach was incrementally developed over several years, and was additionally refined during the production of each of the releases described earlier.

Quality control is achieved through a set of procedures that ensure the orderly development, integration, and testing of software. Early in the development cycle, developers must evaluate one or more approaches for solving a problem. Later in the cycle, they must be willing to reevaluate these approaches based on lessons learned. As the code is developed, it is important to keep in mind the potential needs of future releases and users as well. The results of these thought processes should be made available to future developers, either through formal design documents, or through informal design notes.

As code is developed, it should be under some kind of source code control system, and subject to additional controls when approaching release time. Critical code should be evaluated by more than just its developer. Prior to delivery, each Cronus release is subject to a wide variety of tests to verify its integrity: Finally, mechanisms need to be in place to track and fix bugs after the system has been delivered.

## 4.1 Evaluating and Reevaluating Several Approachs

Our redesign of the Cronus interprocess communication (IPC) system is a good example of how to evaluate different implementation approaches before proceeding. In evaluating various approaches for release 2.0, we spent considerable time making an effort to understand the limitations of the existing Cronus 1.5 IPC system. The IPC redesign required careful consideration and evaluation of various needs, including the satisfaction of performance improvement goals, the realization of desired functionality and usability, and effective integration with underlying operating systems. For example, the status checking mechanism for long duration operations could have been implemented in either the client run-time library or in the Cronus kernel. We decided in release 2.0 to implement it as part of the Cronus kernel - this significantly decreased the IPC code in client programs, at the expense of a slight increase in the amount of code in the Cronus kernel.

Many examples of reevaluation took place during the development of Cronus 3.0. First, we took advantage of the fact that an interim release between Cronus 2.0 and 3.0 incorporated prototype versions of some components. Based on experience gained with these prototypes, we made improvements for the versions included in 3.0. For example, our initial implementation of mount points was redesigned after we discovered some limitations of the approach in the multicluster version. Second, we improved some mechanisms to take advantage of additional mechanisms provided by the native operating systems on which Cronus runs. In particular, we migrated our intrahost IPC mechanisms for UNIX platforms from UDP to UNIX domain sockets, to improve performance and reliability in this and future releases. Third, we reacted to the changing state-of-the-art and user requirements by migrating the Cronus software from outmoded platforms to newer ones. We migrated the Cronus Lisp implementation off of the special-purpose Symbolics machine onto lower-cost / higher-performance commodity hardware. And we migrated the Cronus Mach implementation off of the Sun 3 onto the NeXT workstation. (The former

activity leveraged off of previous work done in Cronus 2.0 to adapt the Cronus Common Lisp implementation to emerging standards, such as the Common Lisp Object Systems (CLOS).)

In summary, we view continuous improvement through evaluation and reevaluation as being critical for the development of successful systems.

## 4.2 Internal Software Design Notes

In addition to formal deliverables, during the project a wide variety of technical notes were produced. The more important of these are available as part of the BBN DOS Note series, which dates back to our early involvement in the development of distributed operating systems. The notes produced during this effort were as follows.

"Cronus Integration Issues," DOS Note 120, Paul C. Neves and James C. Berets, March 1989.

"Supporting Large Object Databases in Cronus," DOS Note 121, Rick Floyd, March 1989.

"Networking Software on VAX/VMS Systems," DOS Note 122, Steve Jeffreys, April 1989.

"MCS Events and Instrumentation," DOS Note 123, Steve Vinter, September 1989.

"Cronus 1.4 for the AT&T 3B2 and 6386 - Release Notes," DOS Note 124, Edward F. Walker, December 1989.

"Cronus on Mach 2.0 - Special Release 1.4.MACH.1," DOS Note 125, Edward F. Walker, January 1990.

"Cronus Simulation Experiment," DOS Note 126, Ken Schroder, January 1990.

"Future Directions for Replication in Cronus," DOS Note 127, Richard Floyd and Stephen Vinter, April 1990.

"Integrating Kerberos into Cronus," DOS Note 128, Rich Salz, June 1990.

"Coda, the Code Distribution Aide," DOS Note 129, Rich Salz, August 1990.

"Cronus Installation Language," DOS Note 130, Rich Salz, August 1990.

"Porting Cronus to the Sun SPARC," DOS Note 131, Paul Neves, June 1989.

"Porting Cronus to the Interactive 386/ix," DOS Note 132, Richard Salz and Herb Lison, June 1990.

"Integrating Cronus and Banyan VINES," DOS Note 133, Ward Walker and Steve Vinter, August 1990.

"Supporting Cronus C++ Applications," DOS Note 134, Rich Salz, December 1990.

"The Delegation Facility," DOS Note 135, Rich Salz, October 1991.

"Issues in Porting Cronus Applications to ANSI C Platforms," Christopher Barber, June 1992.

"Cronus Ada Tutorial," Mike Dean.

"Cronus 2.0 IPC Design Notes," Robert Goguen.

"Cronus C++ Integration," Mike Dean et. al., work in progress.

## 4.3 Walkthroughs

Starting with release 2.0, we used code walkthroughs as an additional software engineering methodology to apply to the development of Cronus software. Walkthroughs are generally acknowledged to be one of the most effective means of improving software quality. In addition to being a very low cost means of finding bugs, walkthroughs provide a forum for software developers to exchange information about the detailed operation of particular software components. Our walkthroughs were very successful: a variety of suggestions for significant code improvements were made, and the project staff was very enthusiastic about applying this approach to other Cronus components in the near future. During the course of this effort we conducted three code walkthroughs of the Cronus IPC system, and one walkthrough each of the Cronus 3.0 database conversion tools, status checking code, and tropic program. These resulted in significant quality improvement of the Cronus software.

## 4.4 Baselevels

Configuration management and base-line generation are our primary means for coordinating multiple developers. All Cronus software is under configuration management control to prevent simultaneous access to changing code by multiple developers and to track these changes. Following release 1.5, we overhauled our configuration management system to add some missing functionality. Our new system added better control over the integration of new software, provides better audit trail maintenance on source code files, and increases the ability of developers to add new components.

Using our configuration management system, we develop baselines, or incremental versions of the system, during the release cycle, to ensure orderly integration of changes. For Cronus release 2.0, we developed three separate baseline snapshots of the system, one prior to the IPC changes, one prior to the Kerberos changes, and one as a pre-release version of 2.0 that was subject to extensive testing. Between Cronus 2.0 and Cronus 3.0, we developed eight separate baseline snapshots of the system for internal use.

Our quality control also included significant double-checking of corrections to bugs discovered during final system testing. For the last several weeks prior to final system delivery, all source code changes were reviewed by at least one developer in addition to the developer making the change.

## 4.5 Testing

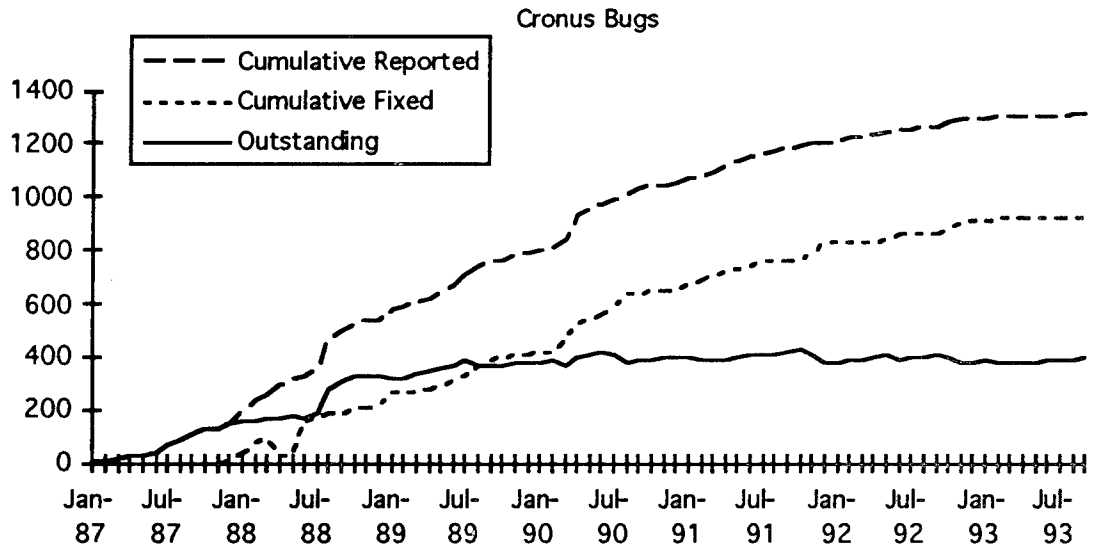
Once a snapshot has been extracted from the source code control system and built, we perform numerous test installations of various types (new installations, additions and updates to existing installations) on various types of machines prior to the release of the software.

- We use a collection of different test suites to validate system behavior.
- We bring up applications on top of Cronus that exercise various critical system features.
- We extensively review documentation to insure consistency with the delivered software.

For Cronus release 2.0, we integrated several existing Cronus tests, and added a variety of new tests into a new Test Manager. In addition to using the Test Manager for testing Cronus at BBN, we have included its source code in Cronus release 2.0. This provides an enhanced testing capability for hard-to-diagnose failures found at user sites, as well as giving end users some examples of how to perform a variety of functions in a Cronus object manager.

## 4.6 Bug Tracking

Beginning in 1987, we tracked the number of Cronus bug reports opened and closed over time. The graph summarizes these results through July 1993.



# 5. Project Technical Briefings

This section summarizes Cronus technical briefings given during the period of performance.

## 5.1 Major Project Reviews

**Cronus Release 1.4**
*September 13-14, 1989*

Overview
Asynchronous Communication: Futures
Update: Cronus Support for Multiprocessors
Release Generation and General Enhancements
Security Enhancement Plans
Testing Cronus
Cronus Applications and Support Activities
Cronus Performance Analysis
Update: Cronus Support for Ada
Query Procesing in Cronus
Monitoring and Control

**Cronus Release 1.5**
*May 23-25, 1990*

Overview
Associative Access for Cronus Managers
Remote Access to Relational Database Management Systems
Update: Cronus Support for Ada
Cronus Applications and Support Activities
Multiple Inheritance and Cronus
Cronus Performance Measurement and Analysis
Cronus IPC Resign
Access Control and Authentication in Cronus,
JDL Demo
New Replication Ideas
Cronus, Mach, and Multiprocesors

**Cronus Release 2.0**
*April 30- May 1, 1991*

Overview
Access Control and Authentication
Cronus Directory Manager
Monitoring and Control System
Update: Common Lisp Implementaiton
Cronus IPC: A Comparison between Cronus 1.5 and 2.0
Cronus IPC Rewrite
Cronus Source Control and Release Generation
Cronus Installation Language
Cronus 2.0 Miscellaneous Enhancements
Cronus Support and Application Activities
Cronus Data Analysis Applications
Cronus Application: DART
Update: Ada Implementation of Cronus
Cronus Port to HP9000 Series 300

Cronus on MS/DOS

**Cronus Release 3.0**
*February 9, 1993*
Major Project Milestones
Cronus Overview
Cronus Multicluster Enhancements
Directory Manager Enhancements
Delegation Facility
Object Database Converter
New Ports
Notable Enhancements
Support
Recent Applications
Summary

## 5.2 Rome Laboratory Technology Exchange Meetings

During the Cronus Enhancements project, we participated in six Rome Laboratory Computer Systems Branch Technology Exchange Meetings. These meetings enabled us to inform other contractors of our progress, learn about other contractor's activities, and help form mutually beneficial technical relationships. A summary of the topics we covered follows.

**The Evolution of Cronus**
*January 10, 1989*
*Stephen T. Vinter*
Goals
Distributed System Problems
What is Cronus?
The Past, Present, and Future

**Cronus Parallelism and Heterogeneity**
*November 8, 1989*
*Stephen T. Vinter*
Increasing Parallelism in Cronus Applications
Experience with Parallel Processors
Work with Ada
Current status
Future Directions

**Cronus Distributed Computing Environment**
*November 28, 1990*
*James C. Berets*
Cronus Overview
Cronus Release 1.5 Summary
Cronus Use and Applications
Cronus 2.0 and Multicluster Plans

**Cronus Evolution**
*November 19, 1991*
*James C. Berets*
Cronus Overview
Cronus 2.0 Summary
Cronus Multicluster Architecture

**Cronus Enhancements**
*January 14, 1993*
*James C. Berets*
Cronus Overview
Cronus 3.0 Preview: New Features
Recent Ports: Convex, NeXT, and Lucid Common Lisp
Current Activities

**An Update on Cronus**
*October 15, 1993*
*James C. Berets*
Cronus in Brief
Recent Technology Development
Cronus Release 3.0 Summary
Cronus / ANM Integration
Cronus / Mach Integration
Use of Cronus in DoD Applications
Increased System Availability

## 5.3 Other Technical Briefings

**Briefing for RL COES**
*Advanced AI Technology Testbed - Integration Issues*
*January 16, 1989*
*Stephen T. Vinter*

**Distributed Computing in the COES Testbed**
*March 1, 1989*
*Stephen T. Vinter*

**Cronus Enhancements Kickoff Meeting**
*March 2, 1989*
*Various Presenters*

**Air Force Geophysics Lab Briefing**
*May 31, 1989*
*Richard E. Schantz, Stephen T. Vinter*

**Presentation to NASA's Life Science Division**
*May 1989*
*Richard E. Schantz*

**Presentation at Princeton Plasma Physics Laboratory**
*Cronus Overview*
*May 31, 1989*
*James C. Berets*

**Special Software Engineering Research Center Meeting**
*Ada and the Cronus Distributed Computing Environment*
*November 1, 1989*
*S. Vinter and M. Dean*

**Open Software Foundation Member Meeting**
*Cronus: Modular Support for Building Distributed Applications*
*November 6, 1989*
*James C. Berets*

**NGCR OSSWG Review, Mobile, AL**
*Cronus: Modular Support for Building Distributed Applications*
*January 26, 1990*
*James C. Berets*

**Naval Ocean Systems Center**
*Cronus Status Briefing*
*March 27, 1990*
*James C. Berets*

**Boeing Computer Services**
*Cronus: Modular Support for Building Distributed Applications*
*June 4, 1990*
*James C. Berets*

**GE ATL**
*Cronus: Modular Support for Building Distributed Applications*
*July 12, 1990*
*Stephen T. Vinter*

**Compaq Computer**
*Porting Cronus to the Interactive 386/ix System*
*Integrating Cronus and Banyan VINES*
*Integrating PCs with UNIX Systems using Cronus*
*August 15, 1990*
*Stephen T. Vinter*

**NUWES**
*The Cronus Distributed Computing Environment*
*August 28, 1990*
*James C. Berets*

**Computer Sciences Corporation**
*The Cronus Distributed Computing Environment*
*September 12, 1990*
*James C. Berets*

**GTE Laboratories**
*The Cronus Distributed Computing Environment*
*October 31, 1990*
*James C. Berets*

**COLSA**
*The Cronus Distributed Computing Environment*
*February 27, 1991*
*James C. Berets*

**CECOM**
*The Cronus Distributed Computing Environment*
*March 6, 1991*
*James C. Berets*

**Siemens-Nixdorf**
*The Cronus Distributed Computing Environment*
*June 13, 1991*
*James C. Berets*

**SDIO NTB**
*The Cronus Distributed Computing Environment*
*September 6, 1991*
*James C. Berets*

**IRS Financial Management Service**
*The Cronus Distributed Computing Environment*
*September 26, 1991*
*James C. Berets*

**Rome Laboratory**
*Cronus Overview*
*January 14, 1992*
*James C. Berets*

**NOSC DC$^2$ Project Review**
*An Update on Cronus*
*January 28, 1992*
*James C. Berets*

**SDIO NTB**
Integrated Heterogeneous Computing Environments: Cronus
March 19, 1992
James C. Berets

**NATO Workshop**
*Cronus Applications*
*May 14, 1992*
*James C. Berets*

**NRaD**
*Cronus 3.0 Review*
*March 30, 1993*
*James C. Berets*

**Digital Equipment Corporation**
*Cronus Overview*
*September 14, 1993*
*James C. Berets*

**Object Managment Group Technical Committee Meeting**
*Object-Oriented Distributed Computing with Cronus*
*February 3, 1994*
*James C. Berets*

**Theater Battle Management Evolvable Systems Techology Day**
*Building Evolvable DoD Software Systems Using Object-Oriented Approaches*
*August 3, 1994*
*James C. Berets*

# 6. Uses of Cronus

Cronus has been used to construct a variety of applications. Some of them are described here.

## 6.1 APS (1994 - present)

The Contingency Theater Automated Planning System (CTAPS) integrates various Air Operation Center (AOC) applications including Rapid Application of Air Power (RAAP), Advanced Planning System (APS), and the Force Level Execution (FLEX) in order to generate a single Air Tasking Order (ATO). The ATO generation process consists of crisis assessment, course of action development, campaign planning, and battle planning and force coordination.

The purpose of the Joint War Interoperability Demonstration (JWID) 1994 exercises was to show the ability to support single threaded, integrated planning of the air portion of joint force efforts. Each CTAPS application involved in the exercise demonstrated planners addressing their areas of emphasis in collaboration with other planners located both locally over LANs and remotely over WANs.

The creation of an ATO relies on separate applications to contribute pertinent information. Each applications feeds required data to the next level of planing tools. For example, the RAAP application produces target nomination lists and weaponeering options for APS; APS will use this data to generate an Air Battle Plan (ABP). Once the ABP has been created, the FLEX application may be required to replan (parts of) the mission due to unexpected events. Because there is no underlying infrastructure which manage the separate applications, monitoring and maintaining the system can be a very difficult task.

As complex systems (such as CTAPS) evolve, even the simplest tasks can become increasingly difficult. Advances in hardware capabilities encourages the use of specialized resources and as a result the main obstacle now lies in linking these resources together. Issues such as data representation and data transfer across hardware platforms, resource location, and maintainability should not be a concern to the application developers. Abstractions powerful enough to hide the complexity of the system are needed so that the application developers can concentrate on what the application does, not how it will accomplish it. Using a distributed working environment, such as Cronus, will provide these abstractions.

The work done in integrating Cronus with the CTAPS applications will demonstrate the interoperability possible between the multiple applications as well as simplifying the overall system. The currently fielded version of APS does not support a demonstration of the sort planned for JWID. The APS application is an integrated, force level air battle planning system which is used to develop ABPs. In order to overcome certain limitations of APS, portions of the system were rewritten to use the Cronus distributed computing environment. In addition, Cronus was used to help integrate RAAP and APS as well as the FLEX and Marquee applications. Three areas in APS were improved for the JWID 1994 exercises: the data transfer mechanism used between RAAP and APS, the database access mechanism used between APS and Oracle, and the IPC mechanism which APS used.

RAAP, APS, and FLEX support different phases of the ATO production process. The first step of this process is performed by RAAP; it's outputs are the weaponeering options and target nomination lists. This information is created using data stored in RAAP's

Sybase database. Once created, these lists are written to the local disk as pipe delimited ASCII files, and then loaded into the CIDB Oracle database. APS then queries this database to retrieve the target/weaponeering information and loads it into the APS Oracle database using Oracle's SQLNET. Once this information has been loaded, APS users can begin work to generate an air battle plan. The main disadvantage to this data transfer approach was the expense in writing and reading these files to/from disk.

Once all of the target and weaponeering information has been loaded into the APS Oracle database, APS users can begin work to generate an ABP. Each APS user is represented by an APS instance which is comprised of four modules (processes): the User Interface (UI), the Assessment Tools module (AST), the Map module (MAP), and the Query Transaction Process (QTP). Each APS instance communicates with an Air Battle Planning (ABP) component to generate an air battle plan (ABP). The UI is the means of communication between the user and the system. The AST process is responsible for evaluating the impact and feasibility of elements for a potential plan. The MAP process provides a graphical view of the last known locations, restricted airspace, and distances from a particular base to target within a certain region. The QTP process is the APS interface to the Oracle database which holds both static and dynamic data. The ABP process is a set of algorithms (written in Ada) which perform knowledge base management, constraint checking/management and auto planning.

Accessing the APS Oracle database is accomplished by the QTP module which uses Oracle's Pro-C programmer's interface. Once the data has been retrieved, it is sent back to the requester in a pipe delimited string using APS IPC library. This requires the QTP to parse the returned data, create the pipe delimited string, send the string to the requester using APS IPC, and then the requester parsing the received string and re-formatting it. While this was a functional approach, it was not a desirable one. Since both RAAP and APS accessed off-the-shelf databases, the Cronus Database managers were used to supply a uniform database interface.

The other area which needed improvement was the IPC mechanism used by APS. Since one of the objectives for the 1994 JWID exercises was to demonstrate distributed collaborative planning capabilities, Rome Laboratory wanted to demonstrate multiple APS users working together to create a single Air Battle Plan over a wide area. While the currently fielded version of APS could achieve this over a local area network, it was not possible to do so over a wide area network. The major problem was the way the APS IPC registration mechanism was designed. The original implementation depended on broadcasts to work properly; a mechanism which is not supported over WANs. The overall design of APS's IPC layer was also lacking in the following areas: it suffered from a single point of failure (the ipm_mang process), each module performed a lot of polling (checking for new messages and/or address bindings), and the way the IPC library was implemented would make it difficult to integrate APS with other applications (RAAP for example). The IPC work was divided up into four phases, where each phase incrementally improved upon the preceding phase and addressed one of the previously mentioned problems. Phase 1 addressed the problem with the registration process, phase 2 addressed the single point of failure problem, phase 3 addressed the excessive polling problem, and phase 4 addressed the interoperabilty issues.

Cronus was also used to help integrate the FLEX and Marquee applications. The FLEX system monitors air battle plan execution, detecting deviations, determining the impact of those deviations on the plan, and suggests courses of action to repair the plan. The Marquee provides a graphical timeline representation of the missions planned in the Air Tasking Order (ATO). Previously, the functionality of these two programs were bundled together as one (FLEX/Marquee), so when Rome Laboratory decided to unbundle them, it

became necessary for the two programs to have access to global status reporting data. FLEX and Marquee needed a reliable mechanism which would push out new status information to both applications.

For JWID '94, Cronus was successfully integrated into APS to help the IPC mechanism used between APS modules and other CTAPS applications. These achievements allowed for the successful demonstration of APS at the JWID '94 exercises over a WAN environment as well as simplifying the data transmission between APS and RAAP. Our improvements to the APS IPC library increased performance dramatically while at the same time simplified many of the IPC algorithms, including those for process registration and addressing. Previously required processes (ipm_mang and imph) were functionally replaced (with the Crimph Manager) which resulted in a stabler, more reliable environment. The use of Cronus to transfer data from one relational database to another (RAAP to CIDB to APS) as well as from APS to Oracle, encourages the idea of distributed computing. Many of the limitations dictated by the database vendors become null issues when using Cronus to access them. Restrictions such as applications having to run on the same machine as the database, or programming support for the C language only, no longer exist when using Cronus.

## 6.2 JWID 94: Network Management Integration (1994 - present)

At JWID 94, two network management systems, ANM (the Advanced Management System) and IMS (the Integrated Management System), ran at different sites. Each system maintained a network map marking the up/down status of hosts. Cronus was used to pass status information between the two systems, keeping the two displays synchronized.

ANM was installed on a SPARC machine running SunOS4 at ACOM, Norfolk, Virginia and IMS on one of two SPARC machines running Solaris 2.3 machines at Ft. Gordon, Georgia. Cronus was installed at both sites on three hosts. A Cronus application *MIST*, (Managed Incremental State Tracking) maintained a representation of known host state information in a MIST object. The MIST application kept the ANM and IMS system synchronized without polling. Also, it used less than 2kb/s over a two hop satellite network and automatically recovered from link failures.

Two clients, *mistify* and *vigil* were used to transfer the state from ANM and pass it to IMS. On the ACOM host, ANM wrote host status information to the mistify client as it detected changes in a host's up/down state. Mistify sent the status changes to a mist object which tracked the state of ANM.

At Ft. Gordon, the vigil client continuously received host status from the mist manager. The mist manager kept track of which host status updates had been received by the client and blocked if the client had received all updates. Multi-tasking in the mist manager allowed it to handle multiple vigil sessions. As it received a host state update, the vigil client wrote it to the IMS system. IMS then used this information to update its network map display.

An application management tool was developed to facilitate debugging as well as to provide a view of the overall configuration's status. This tool displayed an iconic representation of the MIST programs involved and the flow of data between them. The up/down status of the programs, hosts, and network was indicated by changes in icon color. Additionally, Cronus kernels and managers had menus containing Cronus commands to start and stop and display helpful information.

## 6.3 Common Prototyping Environment (1993 - present)

The Common Prototyping Environment (CPE) is one of a number of activities of the ARPA-Rome Lab Planning Initiative (ARPI). The purpose of the CPE is to support technology integration experiments within the planning community, and to allow comparison and evaluation of planning tools by establishing standard communication mechanisms between tools. In addition, the CPE seeks to promote on-line access to data and scenarios for the transportation planning domain, to support the transition of research prototypes to operational systems, and to further the state-of-the-art of distributed concurrent, cooperative planning in a heterogeneous environment.

CPE components interconnected using Cronus include those contributed by: the University of Massachusetts, Carnegie Mellon University, GE CRD, ISX Corporation, BBN Systems and Technologies, and SRI International.

## 6.4 ARGUS (1993 - present)

ARGUS is a versatile software system for the storage, retrieval, and analysis of radar, infrared, acoustic and other types of target signature data. Such signature data are often referred to as MASINT, or Measurement And Signatures INTelligence data. The collection, analysis, and management of signature data on military vehicles, ships and aircraft is vital to the development and evaluation of advanced sensor and weapons systems that are increasingly depended upon to not only detect and track potential targets, but distinguish between friendly and hostile systems.

ARGUS is designed to assist a wide range of users within the DOD, involved in the acquisition, analysis and management of large quantities of target signature data. At one end of the user spectrum are analysts who need to locate, retrieve and operate on specific sets of signature data to conduct quantitative or qualitative analyses. At the other end of the spectrum are managers who don't necessarily need to access the data directly but do need to be aware of what data are available to effectively coordinate signature acquisition efforts, analysis tasks, and production schedules.

In support of this diverse user base, ARGUS is designed to handle a wide range of signature and other related data, such as target geometries or models, and provide extensive on-line documentation of these data. In addition to the core database, ARGUS also provides access to a variety of software tools for the display, analysis and management of signature data.

ARGUS is intended to operate on standard computer workstations running the UNIX operating system and X windows. The initial implementation of the ARGUS prototype is on SUN SPARC workstations. ARGUS is a combination of custom and standard, off-the-shelf software and is written in ANSI standard SQL, C, and Common Lisp programming languages. Cronus is being used in a system integration role for ARGUS: communicating ARGUS components, both LAN and WAN based, cooperate via Cronus operation invocations.

ARGUS was originally aimed at supporting the U.S. Army Foreign Science and Technology Center (FSTC) in the day-to-day management and analysis of their rapidly expanding volume of MASINT data. However, the architecture upon which ARGUS is based is quite versatile, and the system can be readily adapted to support the data storage, retrieval and analysis needs of a variety of other intelligence, Test & Evaluation (T&E), and system development applications.

Recently, ARGUS, in combination with rapidly developing DOD secure network technology, is evolving to become the basis for the implementation of a distributed National Target Signature Data System (NTSDS). As such, ARGUS will not only be a resource that can help individual organizations within the DOD signature community improve the local management of their own data, but will also provide a common vehicle through which the community can interchange data, coordinate activities, and reduce fragmentation.

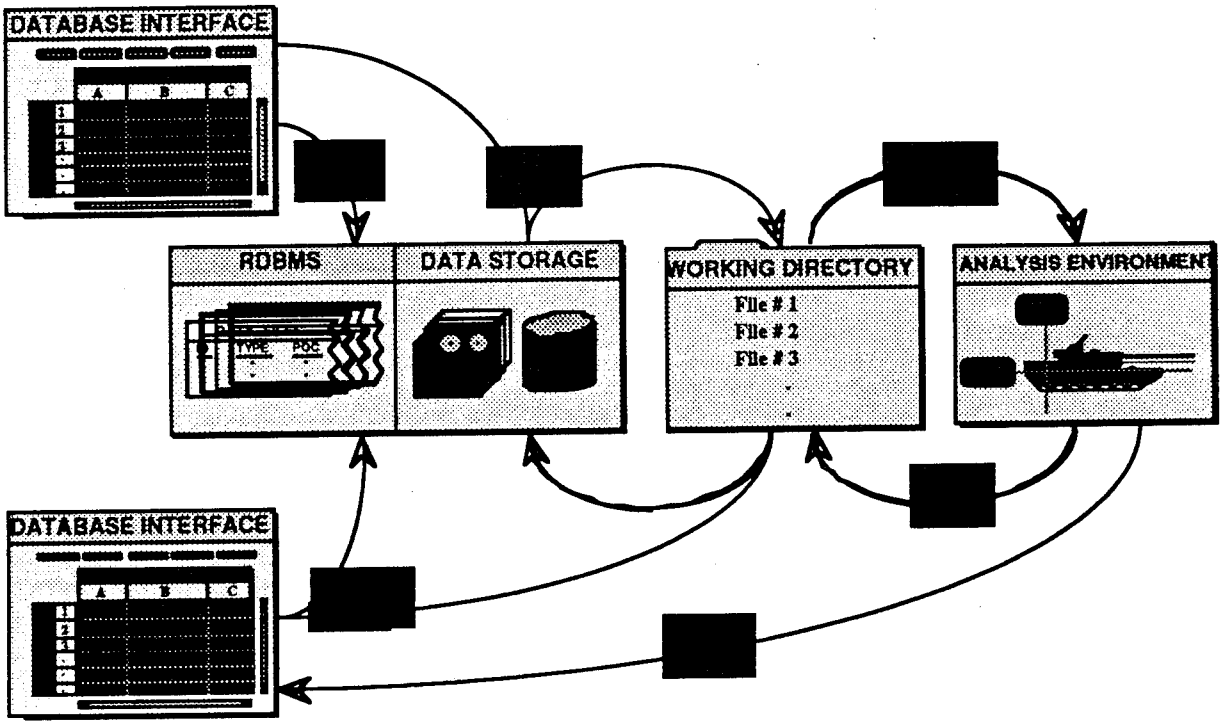



**Figure 4: ARGUS Data Flow**

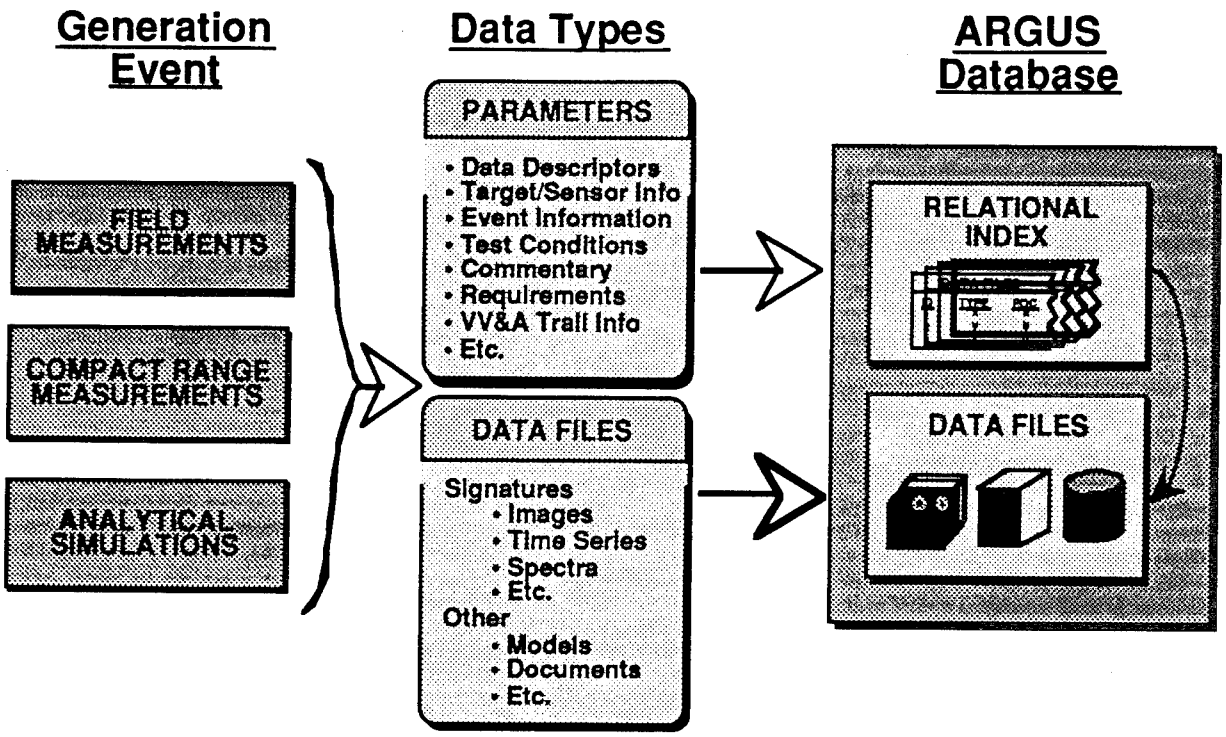



Figure 5: ARGUS Architecture

## 6.5 DART (1991 - present)

The Dynamic Analysis and Replanning Tool (DART) is another application where Cronus has been used to connect dissimilar components. DART started as an effort to inject knowledge-based planning and scheduling technology into the upperreaches of the U.S. Transportation Command, which controls all large military cargo aircraft and ships. Using DART, transportation analysts assess whether sufficient air- and sea-lift capacity exists to move units and equipment to their destination according to schedules determined by warfighting commanders. Much of the analysis done by DART is based on Time-Phased Force Deployment Data (TPFDD) stored in the Worldwide Military Command and Control System (WWMCCS).

When examining a scenario, DART retrieves appropriate information from WWMCCS and then applies intelligent transportation planning tools to analyze and refine the data. (Capabilities also exist for returning a refined TPFDD to WWMCCS.) The bulk of the DART analysis software is written in Common Lisp. Most of the data is stored in an Oracle relational database.

Oracle provides programming tools for executing database queries from the C, FORTRAN, COBOL, and Ada programming languages, but does not provide an interface from Common Lisp. Most Common Lisp implementations under UNIX support some sort of "foreign function call" mechanism that allow calls to the C runtime, but this is cumbersome and error-prone, particularly when dealing in a preemptive multi-tasking environment where garbage collection could occur at any time. The turnkey Cronus Database Interface Managers provide a uniform high-level interface to several different relational DBMSs. They interact particularly well with the dynamic typing of objects in Common Lisp,

allowing a single function call to return converted data in a directly usable form that Common Lisp programmers would naturally expect. During a typical run of an analysis model, DART uses Cronus to retrieve several megabytes of data from several hundred thousand Oracle records.

The original DART system was developed during an intense 10-week effort during Operation Desert Storm, and was used operationally in its latter stages to model and track the flow of personnel and equipment to the Persian Gulf region. The high-level expressiveness of Common Lisp and the availability of building blocks such as Cronus made this rapid development possible.

Cronus is playing a bigger role in follow-on efforts, including the interconnection of standalone DART systems and on-line access to external data sources.

## 6.6 AAITT (1989 - present)

Martin Marietta Laboratories, in conjunction with other contractors, has developed an Advanced AI Technology Testbed (AAITT) for the U.S. Air Force Rome Laboratory. Key in the AAITT architecture is a plug-and-play approach to developing decision aids: within AAITT, individual components from within the Tactical Command Control, Communications, and Intelligence domains can be integrated flexibly. AAITT's design includes three elements: a Distributed Processing Substrate (DPS), a Modeling, Control and Monitoring Workstation (MCM), and the core simulation and database modules. Because of Cronus's extensive support for heterogeneity (languages, operating systems, machines), and its easy-to-use development tools, Cronus was selected as the Distributed Processing Substrate of choice.

AAITT application components have included:

- The Air Force Mission Planning System (AMPS), developed by the MITRE Corporation for planning and replanning air tasking orders
- LACE, a land-air combat simulator for tactical engagements in the European theater
- TAC-DB, a tactical database of NATO capabilities, developed by Knowledge Systems Corporation (KSC).

## 6.7 BBN Office Automation Applications (1988 - present)

### 6.7.1 Phone (1988 - present)[3]

Corporate telephone directories, by their nature, suffer from problems with accuracy and completeness: new employees join a company, existing employees move from one office or department to another, corporate reorganizations take place, and so on. As a result, printed directories are already out-ot-date when they are distributed. For example, BBN's telephone directory is published in hard-copy form twice a year, and information contained in it must be "frozen" approximately one month prior to distribution.

In order to provide a more accurate and complete version of the BBN telephone directory, a Cronus-based on-line version of the directory was developed for UNIX and VMS systems.

[3]Some text provided by an internal memorandum from J. Stephen Groff of BBN.

The Cronus-based system is a client / server system. A copy of the BBN telephone directory database is loaded into an Informix database and resides on a Sun workstation. A Cronus-based telephone directory Phonebook Manager running on this machine allows clients to access this database. Clients with the appropriate BBN Telephone Directory applications software and Cronus kernel software can access the server. CLient software was developed for VAX ULTRIX, VAX VMS with Wollongong TCP/IP, Sun 3 and 4 Symbolics running Genera, Masscomp running RTU, and the Butterfly GP1000 running Mach. There are approximately 100 BBN machines running Cronus software which currently access the BBN Telephone Directory on-line, some of which are located in remote offices (e.g. Bellevue WA., Canoga Park CA., San Diego CA., etc.).

Two user interfaces exist to access the BBN Telephone Directory server. One is a (terminal independent) menu-driven interface. The second interface is a simple command line interface which allows searches by last name. Both of these interfaces could be accessed through a guest account for people who do have accounts on machines where Cronus is installed.

### 6.7.2 Calendar (1988 - present)

Since 1988, BBNers in several departments have used an on-line shared calendar to let each other know about upcoming travel and vacation plans, visitors to BBN, and special events like meetings and demonstrations. In addition to acting as a reminder system and assisting in meeting planning, the Calendar System makes it easy to determine whether or not someone is at BBN on a particular day.

The Calendar System uses Cronus to provide remote access to a shared Informix relational database of schedule information. In essence, Cronus encapsulates the Informix database and makes it available over the network to BBNers running Cronus on their machine. Thus the database can be remotely accessed without the bother of using TELNET to connect to the actual machine on which it resides, and with the appearance that it is actually available locally.

The Cronus Calendar System supports a variety of user interactions. First, a forms-oriented user interface that runs on UNIX (Ultrix, SunOS, etc.) and VAX/VMS systems can be used to enter, modify, display, or browse stored information. Queries to the database may optionally be passed through filters, so that you can focus on a particular groups of people in which you are interested. In addition to the interactive forms interface, one can subscribe to daily calendar distribution by electronic mail. This will cause the Calendar System each morning to send the subscriber electronic mail containing information for that day. Finally, the system can produce hardcopy calendars for printing and distribution, including a three-week-at-a-glance calendar in BBN Slate format.

The Calendar System's user interface can be learned in about ten to fifteen minutes through experimentation.

## 6.8 JDL (1988 - present)

The lead command and control laboratories of the U.S. Air Force, Navy, and Army have, sincd 1988, been evolving a tri-service demonstration of distributed system technology. The development of this demonstration is designed to illustrate the benefits of inter-service cooperation in a command and control context and to serve as a testbed for wide-area, multi-organization distributed applications.

The desire to build applications that span organizational boundaries exposes some unique problems in application development. There is a need to maintain interoperability even when different organizations are responsible for building and maintaining different parts of an integrated application. The system architecture must support evolution, to allow the integration of new or improved components over time. Also, the contributing organizations may wish to remain autonomous, even though they are collaborating in the development and execution of a distributed application. For example, an organization might want to be responsible for configuration management and for access control over the resources it owns. Finally, the operational environment should support application components executing across a high-latency, (possibly) low-bandwidth wide-area network. This poses a set of problems not faced in low-latency, high-bandwidth local area network environments. Object-oriented techniques facilitate the desired interoperability and evolution. As extensions to Cronus, BBN has incorporated new system mechanisms to support *clustering* of resources. (A cluster is essentially a group of host computers managed by a single administrative entity.) The most recent Cronus release included this capability.

## 6.9 Reporting and Tracking System (1987 - present)

The Cronus (Bug) Reporting and Tracking System is a set of tools to help groups of users and maintainers generate, track and close (bug) reports. It can also be used to track hotline calls, enhancement requests, etc. Reports are conveniently accessible in a distributed environment: they are stored in the (Bug) Report Manager and are accessed transparently over the network by one of two user interfaces. Underlying Cronus facilities provide query processing support in the Bug Report Manager to provide SQL-like retrieval capabilities.

Two user interfaces are provided. The first is an X Window-based graphical user interface. The second is an interactive forms-based / command line interface (for non-X users and quick use).

## 6.10 CASES (1988 - present)

The Capabilities Assessment, Evaluation, and Simulation System (CASES) is a campaign-level decision support tool used by military planners to graphically create, edit, and analyze operations plans in both deliberate and crisis planning. CASES provides a distributed computing environment operating over both wide-area and local-area networks to make the advantages of high performance computing capabilities available to planner/analysts.

CASES is a planning aid component of the Navy's Operations Support System (OSS) command center upgrade program. CASES supports the analysis of maritime operations including strike warfare, antisubmarine warfare (ASW), and battle force defense. An ongoing major upgrade will significantly expand the warfare analysis capabilities of CASES including anti-surface warfare, strikes against critical mobile targets, shallow water ASW, theater missile defense, mine warfare, land warfare, surveillance, command and control, and logistics.

CASES is a potent tool supporting the two-tiered command concept. Using the Theater Analysis and Replanning Graphical Execution Toolkit (TARGET) system, CINC level planners can develop skeletal courses of action (COAs) which can then be passed to CASES for more detailed planning and evaluation of alternative COAs.

Using CASES, military planners can create, store and recall operations plans from existing libraries throughout wide-area networks. Plans can be developed collaboratively with planners from diverse commands editing a plan simultaneously. With the support of workstation video tele-conferencing, interactive collaborative planning becomes a uniquely powerful tool for developing well-coordinated operational plans with a high probability of success in the complex environment of joint operations.

CASES provides a shell for the integration of a variety of warfare simulation and performance prediction models. CASES provides planners with a complete set of tools for choreographing the campaign analysis. The human computer interface allows planners to define and modify a wide variety of model inputs which drive the simulation.

All warfare models in the CASES model suite are monte-carlo based simulations which allows the planner/analyst the freedom to explore a wide variety of options and examine a range of possible outcomes. CASES provides both an interactive and a batch evaluation capability. The interactive mode provides the planner with a capability to "sanity-check" the plan and ensure that all operations are occurring as intended and that plan objectives are being achieved. Batch evaluation provides a full set of monte-carlo based results which gives the planner a set of measures of effectiveness with which to compare alternative courses of action.

By integrating tools for the creation, storing, and editing of plans with advanced technology and wide-area networks, CASES provides a true distributed, collaborative decision making environment which provides the military planner with the capability to:

- review and compare multiple alternative COAs quickly
- evaluate a broad range of options
- make timely decisions based on the current situation
- respond quickly to changing scenarios

CASES is used by a variety of commands including JCS, CINCPACFLT, and COMSEVENTHFLT. CASES is compatible with both the DTC-II/Sun 4 and the TAC-III/HP architectures. The CASES model suite runs on a wide variety of high performance computing platforms including Encore, Cray, Sequent, etc.

Cronus is particularly designed for large-scale distributed applications involving many components operating in diverse computing environments like CASES. CASES is a complex Cronus application which includes a large number of application-specific object managers, and a variety of client programs.

## 6.11 THETA (1985 - present)

Currently in its third phase, the THETA (Trusted HETerogeneous Architecture) project is developing a prototype secure distributed computing environment heavily based on Cronus. THETA is aimed at satisfying NCSC B3 requirements for security. THETA assumes a layered architecture and is designed accordingly: like Cronus, it assumes that it will be hosted on top of a native operating system. In THETA's case, the assumption is that the native operating system will also be a secure operating system, and that secure networking capabilities will be available. In Phase I of the project ("Secure Distributed Operating System"), BBN, in conjunction with Odyssey Research Associates, developed system goals and requirements, and an early model for the system. In Phase II ("Experimental Secure Distributed Operating System"), Odyssey Research Associates, in conjunction with BBN, refined the model and developed an early prototype on AT&T 3B2 and 6386 systems running System V/MLS. Currently, in Phase III ("Evaluation /

Enhancements for THETA"), Odyssey Research Associates in conjunction with Trusted Information Systems is expanding the functionality and completeness of the system, as well as porting the system to a number of new platforms (for example the Sun Compartmented Mode Workstation).

## 6.12 TVE (1987 - 1989)

The Cronus Technology Validation Experiment (TVE) was begun in February 1987 at the request of the Air Force Electronics Systems Division (ESD) and the Rome Laboratory (RL) as part of a program to develop and evaluate support for large, distributed Strategic Defense Initiative (SDI) simulations. On this project, BBN focused on the Cronus distributed system and its support for distributed application development and operation.

As part of their program to study SDI BM/C$^3$ architectural issues, the Air Force Electronics Systems Division and MITRE Corporation have been developing a set of simulation programs which model SDI system components and the environment in which they operate. To begin to explore how these simulations could be developed and operated in a distributed environment, the Cronus TVE was devised. We were primarily interested in how the simulation model could be incorporated into an object oriented distributed system; in whether the resulting distributed simulation could be operated efficiently across a large geographical area incorporating a variety of computer systems; and in what ways an operator can effectively configure, monitor and control the simulation as it runs.

For this Cronus experiment, we chose a simulation previously developed by MITRE to explore SDI data processing and battle management algorithms. The simulation models threat trajectory and orbital propagation based on previously generated threat scenarios, limited weapon system operation, sensor system operation, sensor correlation, target track prediction using a Kalman filter, determination of weapon coverage of the threat using a two-stage filter, and assignment of individual weapons to targets. Our experiment methodology was to adapt this suite of existing, non-distributed SDI simulation programs to operate in a distributed environment using Cronus and its distributed application development tools.

The overall MITRE SDI simulation system consisted of three major functions: threat generation; simulation; and display. Threat generation produced simulated booster trajectories, described as an evolving series of position and velocity *state vectors*. The simulation models threat trajectory and orbital propagation based on previously generated threat scenarios; limited weapon system operation; sensor system operation; sensor correlation; prediction using a Kalman filter and known reference trajectories; determination of weapon coverage of the threat using a two-stage filter; and assignment of individual weapons to targets. Display allows the monitoring of the simulation progress.

The non-distributed simulation was comprised of seven primary simulation programs (including two sensors) and a few auxiliary programs. All components operated on a single VAX/VMS system. Most of the software was written in VMS Fortran, with limited sections written in Pascal and VAX assembly language. The operation of the simulation was basically manual and sequential: programs generate inputs to other programs and some programs must be run to completion before other programs are started. The principal mode of communication among the programs was the use of files, although local host interprocess communication (specifically, VMS mailboxes) was used among a small, concurrently operating subset of components. We were supplied with threat files for two scenarios: for a single booster; and for 105 boosters. The threat scenario was selected by choosing one of two suites of input and simulation object parameter files and by various

execution parameters entered interactively when the modules are started. Both scenarios are run with two sensor platforms and 1300 weapon platforms.

The simulation components were as follows.

- *Trajectory / Orbital Propagation (Boosters and Sensors)* numerically integrates state vectors to simulate the movement of the sensors, weapon platforms and boosters described in the threat data provided as input to the simulation. The threat data input file is a series of sensor and weapon platform state vectors at time zero, combined with booster state vectors sampled at times ranging from time zero to the end of the scenario. The output of this module is a series of state vectors describing sensors, weapons and boosters for each epoch from time zero to the end of the scenario. Output is passed to the sensors via VMS mailboxes to allow parallel operation.

- *Orbital Propagation (Weapon Platforms)* numerically integrates state vectors for weapon platforms. This module is similar to the Trajectory / Orbital Propagation module, but passes its output through a file. This eliminates the restriction on the number of active objects which can be handled, but also disallows parallel operation with other modules. Since changes of weapon platforms status do not occur in the current simulation, the file approach is useful here and allows significantly greater numbers of platforms to be simulated. Since, booster status may change as a result of weapon actions in the current simulation, the booster simulation must be simulated in parallel with other components. A follow-up filtering phase, the *Geographic Filter*, calculates *Kinetic Kill Vehicle (KKV)* threat coverage based on straight line KKV trajectories from weapon platforms to booster launch sites. It accepts the series of weapon platform state vectors and filters out data representing infeasible weapon-target pairings.

- *Sensors* generate two-dimensional sensor observations. These modules, of which there may be one or two, accept state vectors sent via mailboxes and emulate the behavior of two dimensional orbiting sensors. This *phase-plane* data represents what the emulated sensor records based upon the environment situation described in the input state vector data. The output of each sensor is a series of two dimensional sensor observations from time zero to the end of the scenario. The output is recorded in a file for later processing.

- *Correlation* performs multi-sensor correlation and two-dimension to three-dimension conversion. This association and fusion processing takes data from multiple sensors, associates target detections from the data and uses the associations and the sensor positions to estimate the three-dimensional position of the targets. The module also makes use of the original threat data to simulatefeedback from the Kalman filter. This simulated feedback data, which will actually be supported in a later version, is used to resolve ambiguities between targets. The output is a series of correlated three-dimensional sensor observations from time zero to the end of the scenario.

- *Kalman Filter* uses Kalman filtering algorithms to predict future target positions. It reads the three dimensional sensor observations and produces a series of target predictions. This module also uses a set of matrices, provided by the pre-filter, which describe a variety of known booster reference trajectories. It also uses a file of known booster launch sites. An auxiliary *Kalman pre-filter* generates matrices used by the Kalman filter which represent a set of known booster reference trajectories.

- *Dynamic Assignment Filter* calculates KKV threat coverage, calculates probabilities of kill for feasible engagements and performs weapon assignments. In the current

simulation these assignments are not fed back to the booster simulation; however future versions will include KKV trajectory simulation and simulated interception with the boosters.

The results of our efforts was a distributed version of the original simulation model. It operated in a testbed that included both Sun UNIX and VAX/VMS systems. The testbed was distributed across three sites: BBN, Cambridge, MA; MITRE, Bedford, MA; and RL, Rome, NY. This software was demonstrated in February, 1988. After this initial demonstration, we integrated improved versions of the simulation algorithms and used the distributed simulation for performance evaluation studies.

## 6.13 INDEXER (1986 - 1989)

The Intelligent Document Indexing and Retrieval System (INDEXER) system was developed by BBN for the U.S. Navy. The intent of this project was to provide a large scale, multi-user distributed document storage and retrieval system which incorporates intelligent components, based on a highly extensible and scalable architecture.

INDEXER provided basic card-catalog-style and keyword indexing and retrieval. Documents and document indices were stored in a distributed fashion among several machines. Users interacted with the system through a graphical interface which supports document viewing, cataloging of new documents, and library searches.

Intelligent, concept-based indexing of documents was also supported by INDEXER. This form of document indexing and retrieval was supported through use of a knowledge representation system. Domain models were developed for the various documents in the system, and were used as the basis for document indexing and retrieval. This concept-based system supported retrieval queries whose target documents did not necessarily directly contain the key words or phases specified in the original query.

The INDEXER system was implemented using Cronus. Cronus provided an object-oriented architecture for developing indexing, storage, and other system modules independent servers. In INDEXER, Cronus supported the integration of modules implemented in both C and Lisp, as well as integration of commercial and custom software.

INDEXER ran on Sun 3 and Sun 4 machines, and consisted of Cronus-based indexing and retrieval software, the KREME knowledge base system, an Informix relational database, and the BBN Slate multimedia document editor.

# 7. Technology Transfer / Advancement of State-of-the-Art

## 7.1 Installations

In addition to being a highly productive R&D effort, the Cronus Enhancements effort has been successful at delivering highly stable, usable software to customer sites. The following sites are representative of those that have used Cronus either directly or as part of an application installation.

Advanced Decision Systems
Center for Naval Analysis
Cimflex Teknowledge
Defense Intelligence Agency
DISA Joint Demonstration and Evaluation Facility
GE Advanced Technology Laboratory
GTE Laboratories
Harvard / Smithsonian Center for Astrophysics
ISX Corporation
McDonnell Douglas
MITRE Corporation
Johns Hopkins University Applied Physics Laboratory
NASA Goddard Space Flight Center
National Computer Security Center
National Parallel Architecture Center (NPAC) at Syracuse University
Odyssey Research Associates
SRI
U.S. Air Force Rome Laboratory (RL)
U.S. Army Communications and Electronics Command (CECOM)
U.S. Army Foreign Science and Technology Center (FSTC)
U.S. Navy NCCOSC RDT&E Division (NRaD)
U.S. Navy Pacific Missile Test Center
U.S. Pacific Command
U.S. Transportation Command
University of California, Los Angeles

## 7.2 Training

During the period of performance, we improved and modified our Cronus Workshop. This five day workshop is designed to familiarize attendees with the methodology and tools for designing and building Cronus distributed applications, and with the capabilities of Cronus and its relevance to the customer's problems. We have found the workshop to be instrumental in increasing user's understanding of Cronus concepts and mechanisms, and also in the general concepts of object-oriented distributed systems.

The workshop is intensive and is oriented toward C programmers fluent in the language. No previous Cronus experience is required. The workshop consists of approximately 50% lecture sessions and 50% hands-on Cronus programming exercises. The lecture sessions in turn consist of about 50% conceptual and 50% practical material typically including the following major areas:

Motivation
Cronus Object Model

Cronus System Managers
Cronus Command Set
Cronus Database Managers
Protocol Layering
Asynchronous Mechanisms
Manager Development Tools
Cronus Kernel
Other Environments: Common Lisp and Ada
Building Distributed Applications
Some Example Cronus Applications
Future Development
Cronus and CORBA
Question and Answer

During the five year contract period, the course was taught 18 times, 9 times at BBN and 9 times at customer sites. (12 of these courses were funded independently of this activity.) The dates of these workshops and the number of attendees were as follows:

Course #7; January 1989; 10 attendees
Course #8; May 1989; 16 attendees
Course #9; June 1989; 14 attendees
Course #10; December 1989; 11 attendees
Course #11; March 1990; 16 attendees
Course #12; July 1990; 9 attendees
Course #13; November 1990; 14 attendees
Course #14; April 1991; 12 attendees
Course #15; June 1991; 12 attendees
Course #16; July 1991; 7 attendees
Course #17; November 1991; 6 attendees
Course #18; March 1992; 23 attendees
Course #19; bruary 1991; 11 attendees
Course #20; March 1993; 15 attendees
Course #21; April 1993; 11 attendees
Course #22; December 1993; 7 attendees
Course #23; April 1994; 15 attendees
Course #24; September 1994; 6 attendees

Contractors learning about Cronus at the workshops included:

Advanced Decision Systems
Analytics
BBN
Cimflex Teknowledge
COLSA
Computer Sciences Corporation
Draper Laboratories
GE Advanced Technology Laboratory
IBM FSC
ISX Corporation
Logicon
MITRE
McDonnell Douglas
SAIC
Syscon

TASC
Texas Instruments

Government agencies learning about Cronus at the workshops included:

U.S. Army CECOM
NRaD
National Security Agency
Pacific Missile Test Center
Rome Laboratory

Academic institutions learning about Cronus at the workshops included:

Johns Hopkins University Applied Physics Laboratory
San Diego State University
University of California, Los Angeles
University of California, San Diego

Construed more broadly, the Cronus workshops have served to promote the concepts of object-oriented distributed computing to a wide audience. In a sense, the wider acceptance of these concepts is important to the longer-term adoption of these concepts into commercial products.

## 7.3 Dissemination of Technical Information

For many years, BBN has provided paper copies of Cronus documents and reports on request to interested parties in the Government, academia, and industry. Since early 1992, BBN has also provided some Cronus technical information electronically via a repository accessible via anonymous file transfer over the Internet. We believe that this mechanism, which is now widely being used in the research community, has significantly broadened the number of people who can leverage off of our pioneering work in distributed systems. A sampling of organizations that have obtained our overview document "Introduction to Cronus" electronically over the Internet from May 1993 through September 1994 includes:

ACE Associated Computer Experts, Netherlands
Advanced Decision Systems
Amdahl Corporation
ARCO Oil and Gas
Art+Com, Germany
AT&T Bell Laboratories
Australian National University, Australia
Auto-trol Technology Corporation
Bell Northern Research, Canada
Brookhaven National Laboratory
Brown University
Bull SA, France
Cal Poly State University
Carnegie Mellon University
Catholic University of Nijmegan, The Netherlands
CERN
Chorus Systems
Chungang University, Korea
Colorado State University
Columbia University
Conservatoire National Des Arts et Metiers, France
Control Data Corporation
Defence Science and Technology Organization, Australia
Defense Research Establishment, Canada
DePaul University
Duke University
East Stroudsburg University
Erasmus Universiteit Rotterdam, Netherlands
Ericsson Telecom AB, Sweden
ESRI
Facultes Universitaires de Namur, Belgium
Fermi National Accelerator Laboratory
George Washington University
Georgia Institute of Technology
Goldstar Corporation, Korea
GTE Government Systems
GTE Laboratories
Halliburton Company
Harvard University
Hewlett-Packard
IBM
Imperial College, London
Indiana University
INMOS Corporation
Institute for Information Industry, Taiwan
Intel
JP Morgan
Katholieke Universiteit Leuven, Belgium
Korea Advanced Institute of Science and Technology, Korea
La Trobe University, Australia
Langley Air Force Base
Lawrence Livermore National Laboratory
Lehman Brothers
Matrix Corporation
Michigan State University
Middle East Technical University, Turkey
Mississippi State University
MITRE
Mitsubishi Electric Corporation, Japan
Motorola
NASA
NASA Langley Research Center
NASA Marshall Space Flight Center
National University of Singapore, Singapore
Naval Research Laboratory
New Mexico Tech
New York University
Nippon Electric Company
Northeast Parallel Architecture Center, Syracuse University
Northwestern University
Novell, Inc.
Open Software Foundation
Oracle Corporation
Pennsylvania State University
Princeton University
Royal Institute of Technology, Sweden
Saint Cloud State University
Seoul National University, Korea
Siemens AG, Germany
Stanford University
Stephen F. Austin State University, TX
Stratus Computer
Sun Microsystems
SUNY Buffalo
SUNY College of Technology
Syracuse University
Technical University of Ilmenau, Germany

Technische Universitaet Muehchen, Germany
Teknekron Communications Systems
Texas A&M University
TFL, Denmark
Tour Total, France
Trinity College, University of Dublin, Ireland
U.S. West
Union Bank, Switzerland
Universita degli Studi di Milano, Italy
Universitaet Frankfurt, Germany
Universitaet Hannover, Germany
Universitaet Hildesheim, Germany
Universitaet Kiel, Germany
Universitaet Koeln, Germany
Universitaet Paderborn, Germany
Universite Catholique de Louvain, Belgium
University College Cork, Ireland
University of Alberta, Canada
University of Arizona
University of Bath, UK
University of British Columbia, Canada
University of California, Berkeley
University of California, Los Angeles
University of Cincinnati
University of Colorado
University of Illinois at Urbana-Champaign
University of London, England
University of Manitoba, Canada
University of Maryland Baltimore County
University of Massachusetts
University of Massachusetts - Lowell
University of Michigan
University of Pennsylvania
University of Pittsburgh
University of Sydney, Australia
University of Texas at Arlington
University of Toronto, Canada
University of Tromsoe, Norway
University of Twente, Netherlands
University of Waterloo, Canada
Unix Pros
Washington University, St. Louis
Western Michigan University
WilTel
Worcester Polytechnic Institute
Yuan-Ze Institute of Technology, Taiwan

## 7.4 Citations in the Literature

One measure of the success of an R&D activity, and its effectiveness at technology transfer is the quantity and breadth of citations in the literature. Cronus has been cited numerous times over the last five years. The following citations (some annotated) illustrate some of the places where Cronus has been cited. Papers or reports with one or more BBN authors are noted with a preceding asterisk (*).

"Design Strategies for Object-Oriented Simulation Testbeds that Support Software Integration," Michael L. Hilton and Craig S. Anken, to be published in a Monograph by IEEE Press.

* "The ARPA / Rome Laboratory Planning Initiative Common Prototyping Environment: A Framework for Software Technology Integration, Evaluation, and Transition," Mark H. Burstein, Richard Schantz, Marie A. Bienkowski, Marie E. desJardins and Steven Smith, *IEEE Expert*, accepted for publication.

"Distributed Vertical Model Integration," Robert R. Lutz, *John Hopkins University Applied Physics Laboratory Technical Digest*, February 1995, accepted for publication.

"Using THETA to Implement Access Controls for Separation of Duties," Rita Pascale and Joe McEnerney, *Proceedings of the 17th National Computer Security Conference*, October 11-14, 1994.

*Heterogeneous Software Configuration Management Apparatus*, David C. Lubkin et. al., U.S. Patent No. 5,339,435, August 16, 1994.

* "Building Object-Oriented Distributed Applications Using Cronus," James C. Berets and Michael A. Dean, *Proceedings of the 4th Annual IEEE Dual-Use Technologies and Applications Conference*, Volume II, pp. 236-245, May 23-26, 1994.

* "Distributed Computing with Photon," Edward F. Walker and Carl D. Howe, *Proceedings of the 4th Dual-Use Technologies and Applications Conference*, Volume II, pp. 246-254, May 23-26, 1994.

> This paper describes Photon, a distributed computing environment. Photon's design facilitates the construction of high-performance distributed applications, particularly those in which network latency is the dominant factor in determining application performance. Photon is loosely modelled on Cronus, and the two are briefly compared in the paper.

"A Common Prototyping Environment for Planning and Scheduling Technology," Karen M. Alguire and Louis J. Hoebel, *Proceedings of the 4th Annual IEEE Dual-Use Technologies and Applications Conference*, Volume I, pp. 139-144, May 23-26, 1994.

"The Adaptive Fault-Resistant System," P. Thambidurai, B. Gupta, D. Sutton, A. Kitchen, and T.F. Lawrence, *Proceedings of the 4th Dual-Use Technologies and Applications Conference*, Volume II, pp. 271-276, May 23-26, 1994.

> This paper mentions Cronus as one of the baseline systems used to implement an experimental fault-tolerant distributed system that dynamically and adaptively

allocates resources in the presence of conflicting demans from different applications.

*Adaptive Fault Tolerance*, GE Aerospace Advanced Technology Laboratories, RL-TR-94-65, Rome Laboratory, May 1994.

*An Introduction to the THETA: A Secure Distributed Operating System*, ORA TM-94-0012, Odyssey Research Associates, February 11, 1994.

This report provides an introduction to THETA (Trusted HETerogeneous Architecture), a secure distributed operating system based heavily on Cronus.

* *Gigabit Networking*, Craig Partridge, Addison Wesley Publishers, Reading, MA, p. 322, 1994.

*Distributed Computation Tools Experiment*, Thomas J. Brando and Myra Jean Prelle, MTR 93B0000151, MITRE Corporation, November 1993.

"Object Orientation in Heterogeneous Distributed Computing Systems," John R. Nicol, C. Thomas Wilkes, and Frank A. Manola, *Computer*, pp. 57-67, June 1993

This paper discusses the role of object-orientation in the construction of distributed computing systems. A number of example systems are described, including Cronus.

"Distributed Computation Tools Experiment," Myra Jean Prelle, *Proceedings of the 3rd IEEE Dual-Use Technologies and Applications Conference*, pp. 24-30, May 1993.

*Integration Of Data Between Typed Objects By Mutual, Direct Invocation Between Object Managers Corresponding To Object Types*, Dana Khoyi et. al., U.S. Patent No. 5,206,951, April 27, 1993.

* *Introduction to Cronus*, James C. Berets, Natasha Cherniack, and Richard M. Sands, BBN Systems and Technologies Technical Report 6986, January 1993.

This report gives a detailed overview of Cronus.

* "Uniform Access to Signal Data in a Distributed Heterogeneous Computing Environment," Steven Jeffreys, *International Telemetering Conference Proceedings XXVIII*, Instrument Society of America, pp. 707-714, October 1992.

This paper describes a software interface to telemetry data in a heterogeneous distributed environment, implemented using Cronus. It also describes the integration of this data access subsystem with a data analysis expert system.

*Distributed Office Automation System With Specific Task Assignment Among Workstations*, Charles Lapourtre and Gerard H. Rolf, U.S. Patent No. 5,136,708, August 4, 1992.

"Resource Management: Support for Survivable and Adaptable C3 Applications," Gary L. Craig and Vaughn T. Combs, *Proceedings of the Command, Control, Communications and Intelligence Technology and Applications Conference*, IEEE, pp. 295-299, June 1992.

"Advanced Artificial Intelligence Technology Testbed," John S. Zapriala and Russell R. Irving, *Proceedings of the Command, Control, Communications and Intelligence Technology and Applications Conference*, IEEE, pp. 44-47, June 1992.

*Experimental Secure Distributed Operating System Development - THETA Phase II*, ORA * Corporation, RL-TR-92-110, Rome Laboratory, June 1992.

* "Building Distributed Applications: Why Use Objects?," James C. Berets, *Workshop on Object-Oriented Modelling in Distributed Systems*, NATO AC/243 Panel 11 RSG.1, Quebec, Canada, May 1992.

This paper describes the benefits of using an object-oriented approach to building distributed applications.

"Adaptive Fault Tolerance in Complex Real-Time Distributed Computer System Applications, K.H. Kim and Thomas F. Lawrence, *Computer Communications*, v. 15, no. 4, pp. 243-251, May 1992.

"A Survey of Asynchronous Remote Procedure Calls," A. L. Ananda, B. H. Tay, and E. K. Koh, *Operating Systems Review*, v. 26, no. 2, pp 92-109, April 1992.

This paper describes various designs for RPC systems that support asynchronous operation. Seven different systems are contrasted, including Cronus's futures mechanism.

"Experiences with Accommodating Heterogeneity in a Large Scale Telecommunications Infrastructure," John R. Nicol, C. Thomas Wilkes, Richard D. Edmiston, and Joseph C. Fitzgerald, *Proceedings of the 3rd Symposium on Experiences with Distributed and Multiprocessor Systems*, USENIX Association, March 1992.

This paper describes two applications of Cronus. The first uses Cronus's capabilities to remotely access off-the-shelf relational databases; the second implements a video server using Cronus.

*JDL Tri-Service Distributed Technology Experiment*, L. R. Dunham, M. J. Gadbois, and M. F. Barrett, Naval Command, Control and Ocean Surveillance Center RDT&E Division Technical Document 2332, March 1992.

This report describes an experimental distributed application built using Cronus by the Naval Command, Control and Ocean Surveillance Center RDT&E Division (NRaD), the Rome Laboratory (RL), and the Communications and Electronics Command (CECOM).

"Nesting Actions through Asynchronous Message Passing: the ACS Protocol," Rachid Guerraoui, Riccardo Capobianchi, Anges Lanusse, and Pierre Roux, *Lecture Notes in Computer Science*, v. 615, pp. 170-184, 1992.

*An Overview of Projects on Distributed Systems*, Alfred J. Lupper, University of Ulm, 1992.

This survey describes 86 different projects in distributed computing, including Cronus.

"The Clouds Distributed Operating System," Partha Dasgupta, Richard J. LeBlanc, Jr., Mustaque Ahamad, and Umakishore Ramachandran, *Computer*, v. 24, no. 11, p. 34-44, November 1991.

Cites Cronus as a distributed, object-based system.

"A Distributed Environment for Testing Cooperating Expert Systems," Jeffrey D. Grimshaw and Craig S. Anken, *AGARD Conference Proceedings 499: Machine Intelligence for Aerospace Electronic Systems*, NATO Advisory Group for Aerospace Research and Development (AGARD), September 1991.

This paper describes a general framework for integrating decision support systems using Cronus.

* *Cronus Port to the HP 9000 Series 300: Final Report*, Michael A. Dean, BBN Systems and Technologies Technical Report 7616, June 1991.

This report describes the methodology for porting Cronus to a new machine and discusses the process within the context of a port to the HP 9000 series 300 running HP-UX.

"ASTRA - An Asynchronous Remote Procedure Call Facility," Ananda, A.L., Tay, B.H., Koh, E.K., *Proceedings of the 11th International Conference on Distributed Computing Systems*, IEEE Computer Society, pp. 172-179, May 20-24, 1991.

Compares ASTRA mechanisms with other related ones, including Cronus futures.

"Autonomous Heterogeneous Computing - Some Open Problems," Hermann Schmutz, *Lecture Notes in Computer Science*, v. 563, pp. 63-71, 1991.

"Current Trends in Distributed Systems," Gunter Muller, *Lecture Notes in Computer Science*, v. 555, pp. 204-224, 1991.

This paper references Cronus as an example of a system facilitating distributed computing in a heterogeneous environment.

"Transforming LANs into Virtual Supercomputers," James Kobielus, *Network World*, v. 7, no. 49, p. 1, December 3, 1990.

This article describes some mechanisms for doing network-based parallel computing, and includes a description of Cronus's futures mechanism.

"Vanguard: A Protocol Suite and OS Kernel for Distributed Object-Oriented Environments," Finlayson, Ross S., Hennecke, Mark D., and Goldberg, Steven L., *Proceedings of the Second IEEE Workshop on Experimental Distributed Systems*, p. 42-44, October 11-12, 1990.

"Automated Extensibility in THETA," Joseph R. McEnerney, Randall Browne, Rammohan Varadarajan, and D. G. Weber, *Proceedings of the 13th National Computer Security Conference*, October 1990.

"Adaptive Fault Tolerance: Issues and Approaches," K.H. Kim and Thomas F. Lawrence, *Proceedings of the 2nd IEEE Workshop on Future Trends of Distributed Computing Systems*, pp. 38-46, September 1990.

*Distributed System Evaluation*, Vaughn T. Combs, Patrick M. Hurley, Charles B. Schultz, and Anthony M. Newton, RADC-TR-90-185, Rome Air Development Center, July 1990.

This report describes a variety of benchmarks that were performed on Cronus. Using Cronus Release 1.5, the authors measured and compared Cronus and SunRPC application-layer performance, and measured the performance of Cronus IPC and replication mechanisms.

* "CASES: A System for Assessing Naval Warfighting Capability," Bruce M. Anderson and John P. Flynn, *Proceedings of the 1990 Symposium on Command and Control Research* (available as Science Applications International Corporation Report SAIC-90/1508), June 1990.

This paper describes a knowledge-based decision-support application written in Common Lisp that is using Cronus to access a variety of multiprocessor-based FORTRAN simulation models and a relational database.

"Tri-Service Distributed Technology Experiment," Manchi J. Gadbois and Anthony M. Newton, *Proceedings of the 1990 Symposium on Command and Control Research* (available as Science Applications International Corporation Report SAIC-90/1508), June 1990.

This paper describes an application that uses Cronus in a multi-site internetworked environment to implement a distributed command and control simulation.

* "Asynchronous Remote Operation Execution in Distributed Systems," Edward F. Walker, Richard Floyd, and Paul Neves, *Proceedings of the 10th Int'l Conference on Distributed Computing Systems*, pp. 253-259, May 1990.

This paper describes the design and implementation of a Cronus mechanism known as "futures". Futures support asynchronous operation invocation at a level of abstraction similar to remote procedure calls (RPC), and can be used to build applications that perform coarse-grain parallel processing using networks of computers.

* *Cronus and Open Systems Interconnection: A Functional Comparison*, James C. Berets BBN Systems and Technologies Technical Report 7297, April 1990.

This report discusses Cronus in the context of the ongoing standardization work in the ISO Open Systems Interconnection (OSI) efforts, and includes a brief survey of a number of OSI implementations and some ideas for future work on Cronus in an OSI environment.

"Retrospective on DACNOS," Kurt Geihs and Ulf Hollberg, *Communications of the ACM*, v. 33, no. 4, pp. 439-448, April 1990.

This paper describes a research project conducted by the University of Karlsruhe and IBM Germany. A comparison with Cronus is included.

"Networking the New Workstations," Daniel P. Dern, *CommunicationsWeek*, p. 29, April 9, 1990.

*The Matrix: Computer Networks and Conferencing Systems Worldwide*, John S. Quarterman, Digital Equipment Corporation, p. 91, 1990.

This book mentions Cronus in a survey section on distributed operating systems.

"The Security Policy of the Secure Distributed Operating System Prototype," Norman Proctor and Raymond Wong, *Proceedings of the 5th Aerospace Computer Security Applications Conference*, December 4-8, 1989.

*Distributed Systems: A Comprehensive Survey*, Uwe M. Borghoff and Kristof Nast-Kolb, Technical Report No. TUM-I8909, Techn. Univ. Munchen, November 1989.

"The SDOS System: A Secure Distributed Operating System Prototype," Raymond Wong, Matthew Chacko, Eugene Ding, Brian Kahn, Norman Proctor, John Sebes, and Ram Varadarajan, *Proceedings of the 12th National Computer Security Conference*, October 10-13, 1989.

* *Cronus: A Distributed Computing Environment*, Proposal to the Open Software Foundation's Request for Technology on Distributed Computing Environments, October 6, 1989, 95pp.

"The Secure Distributed Operating System - An Overview," Rammohan Varadarajan, Joseph R. McEnerney, and D. G. Weber, *Proceedings of the 1989 Workshop on Operating Systems for Mission Critical Computing*, September 19-21, 1989.

* "Distributed Query Processing in Cronus," Stephen T. Vinter, Nilkanth Phadnis, and Richard Floyd, *Proceedings of the 9th Int'l Conference in Distributed Computing Systems*, pp. 414-422, June 1989.

This paper describes extensions to the Cronus object storage facilities that support associative access to objects and distributed query processing.

* "Integrated Distributed Computing Using Heterogeneous Systems," Stephen T. Vinter, *SIGNAL*, v. 43, no. 10, pp. 157-162, June 1989.

This article provides an overview of Cronus and the system development issues in heterogeneous distributed systems.

"The Software Bus - A Vision for Scientific Software-Development," D.E. Hall, W.H. Greiman, W.F. Johnston, A. X. Merola, S.C. Loken, and D.W. Robertson, *Computer Physics Communications*, v. 57, no. 1-3, p. 211-216, 1989.

* *Cronus Multiprocessor Investigation*, Michael A. Dean, BBN Systems and Technologies Technical Report 6930, December 1988.

This report classifies multiprocessor systems, provides a brief survey of some of the commercially available multiprocessor systems, and describes various approaches for porting Cronus to multiprocessor systems.

"Upper Layer Interoperability," Franco Vitaliano, *ConneXions: The Interoperability Report*, v. 2, no. 10, pp. 6-10, October 1988.

Contains a one paragraph description of Cronus as an example distributed operating system.

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: 1996-710-126-20102

# *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.