

AD-A261 102



RL-TR-92-302
In-House Report
November 1992



1991 DISE SUMMARY REPORT

**Anthony M. Newton, Vaughn T. Combs, Cheryl L. Blake, 2 Lt,
USAF, Francis A. Dilego, Jr., Scott M. Huse,
Patrick M. Hurley, Terrance Stedman, Jerry L. Dussault,
Robert M. Flo**

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

93-03713



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

98 2 22 045

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-92-302 has been reviewed and is approved for publication.

APPROVED:



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

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 13441-5700. 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

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE November 1992	3. REPORT TYPE AND DATES COVERED In-House
4. TITLE AND SUBTITLE 1991 DISE SUMMARY REPORT		5. FUNDING NUMBERS PE - 62702F PR - 5581 TA - 28 WU - 17
6. AUTHOR(S) Anthony M. Newton, Vaughn T. Combs, Cheryl L. Blake, 2 Lt, USAF, Francis A. Dilego, Jr., Scott M. Huse, Patrick M. Hurley, Terrance Stedman, Jerry L. Dussault, Robert M. Flo		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Rome Laboratory (C3AB) 525 Brooks Road Griffiss AFB NY 13441-4505		8. PERFORMING ORGANIZATION REPORT NUMBER RL-TR-92-302
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (C3AB) 525 Brooks Road Griffiss AFB NY 13441-4505		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES Rome Laboratory Project Engineer: Anthony M. Newton/C3AB (315) 330-3623		
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.		12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 words) The 1991 DISE Summary Report is a progress report on the activity of the Rome Laboratory in-house work in distributed processing systems. In addition to the progress report, there are six condensed articles on the individual research, development, and application projects currently underway within the in-house group. These articles describe two distributed applications, a distributed instrumentation package, two reference papers on distributed database and object management systems, and an evaluation of the Informix Database Management System.		
14. SUBJECT TERMS Distributed Operating System, Distributed System, Distributed Application, Distributed Database		15. NUMBER OF PAGES 32 16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED
20. LIMITATION OF ABSTRACT SAR		

1991 DISE Summary Report

Table of Contents

Foreword	2
Background.....	3
Progress Report.....	3
A Reliable Distributed AF Application.....	5
DAIT: Distributed Application Instrumentation Tool.....	10
Object Translation into Relational Databases.....	13
Information Exchange in Allied Operations.....	16
An Informix Analysis and Evaluation.....	19
DDES: Distributed Database Experiments.....	23

DTIC QUALITY INSPECTED 3

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Code	
* Dist	Avail and/or specified
A-1	

Foreword

This report is organized into three major sections. The Background section gives information on the history of this effort. The Progress Report section details the work performed under the 1991 fiscal year, beginning October 1990 and ending September 1991. The final section exists as a set of associated articles that describe some of the year's efforts. The articles are done in research paper format.

Background

The Command, Control, and Communications (C3) systems that the Air Force (AF) currently uses are a collection of independent and interdependent systems. While sounding like a contradiction, the systems were usually procured for a particular function first, then integrated into the Command and Control (C2) structure later. Each system retains a measure of independence, but the information processed by the collection of systems is critical to our C2 measures and countermeasures. Our interest and thrust within this C2 environment is to provide the capability for all of the individual systems to be integrated into a distributed system (i.e. a system of systems). This entails having a set of heterogeneous hosts that communicate and are jointly managed in a timely fashion with fault tolerance, reliability, survivability, et cetera. This system needs to be capable of adjustment as new demands, capabilities, and technologies are integrated through software or hardware additions. Moreover, all of this must be done at a reasonable cost.

After realizing that the current level of research in Distributed Operating Systems (DOS) had reached a degree of maturity, the Computer Systems Branch of the Rome Laboratory (RL) established an in-house capability called the Distributed Systems Environment (DISE) in fiscal year 1987; wherein, we planned to continue the development of distributed systems technology areas and to provide a demonstration capability for our funding sources. The DISE seeks to provide a setting where: researchers can investigate and demonstrate the issues of concern for Distributed Computing Environment technology, developers can design, create, experiment with, and verify distributed applications, and policy makers and technology managers can see the benefits of distributed technology.

Progress Report

During our fifth year of operation, we continued our role of research and application development. During fiscal year 1991, 5 major projects (2 application development, 1 research, 1 systems development, and 1 training), with 2 minor projects (1 research, and 1 systems development) were undertaken.

The Joint Directors of Laboratories (JDL) Tri-service Experiment is an ongoing application development project originally started in fiscal year 1989. Since its inception and with manpower distributed between Rome Laboratory (RL), Communications and Electronics COMmand (CECOM) and Naval Ocean Systems Center (NOSC), the project has developed and demonstrated the feasibility of integrating separate service applications for joint operations. During the past fiscal year (1991), most of the effort has been put into increasing the reliability and survivability of the federated applications (further information on the AF application can be found in the article: *A Reliable Distributed AF Application*). Work has been performed to integrate data from the object model-based applications into the relational databases most likely to be used by our respective services (Informix, Oracle, and Sybase). This would allow greater access and reliability to the information gathered by the applications for users or programs not conversant with our current environment (further information on the model changes necessary are contained in the article: *Object Translation into Relational Databases*). We will be extending this capability by replicating and maintaining the data among the three relational database systems. Another major effort was in moving the application and system graphics to the X window development package. This allows us to separate the display processing from the display presentation, enhancing the survivability of the presentation graphics.

The distributed application instrumentation project is an ongoing, two-year research program that started in fiscal year 1990 and was slated for completion during fiscal year 1991. The instrumentation package has been completed, with most of the year devoted to implementation, integration, and testing. The display and presentation modules have not been completed, so the project will overrun into fiscal year 1992. Further information on the instrumentation package is contained within the article: *DAIT: A Distributed Application Instrumentation Tool*.

The Australian Experiment is an ongoing application development and research project. Conceived during fiscal year 1989, the project has proceeded slowly as international technology agreements and laboratory manpower agreements were negotiated. The joint development program has both Rome Laboratory (US) and the Electronics Research Laboratory (Australia) cooperating to share distributed operating system and intelligent network technology. During the 1991 fiscal year, key contacts at the ERL have been established to allow us to proceed with the development process. Most of the research work for this experiment will be carried out contractually. Most of the application development will be jointly done by contractual and laboratory engineers. Towards the latter part of the fiscal year, an application began to take shape.

The Alpha Development Environment was a systems development project tasked to design and implement a means of incorporating the Alpha real-time distributed operating system (DOS) into the available DISE structure. The final solution, which minimized networking impact on the principle DISE net and maximized access to the new DOS, involved creating a subnet to absorb the expected data transfer load, populating the environment with multiprocessor workstations and process servers, and establishing a sentinel to limit traffic flow between the subnet and principle DISE net. We were unable to acquire the Alpha system software needed to generate the application and sentinel software.

The RDBMS/Informix Study was designed to provide a greater understanding of the database technology and to evaluate the appropriate use of the Informix database management system in several of our current and future projects. A full report on the effort has been published locally at Rome Laboratory. The article *An Informix Analysis and Evaluation* contains an abbreviated version of that report.

The Distributed Database Experiments was a research project to develop a survivable database management system. What is meant by survivable is that users will have continued access to correct data in spite of various types of failures, such as network partitions, local or remote host failures, and application or system software failures. A full report on this effort has been published locally at Rome Laboratory. The article *DDES: Distributed Database Experiments* contains an abbreviated version of that report.

The Demonstration Configuration was a systems development project tasked to create a new facility for use by the in-house group. It was outstanding in its execution as the upgrading of mechanical and electrical systems, the movement of computer resources, and the acquisition of furniture was completed in minimal time and with minimal disruption to the group.

A Reliable Distributed AF Application

Scott M. Huse
Anthony M. Newton

Computer Systems Branch (C3AB)
Rome Laboratory
Griffiss AFB, NY 13441-5700

1. Overview

The reliable Air Force application discussed within this paper is a subset of broader work produced by the Tri-Service Distributed Computing Technology Experiment[1]. The Tri-Service Experiment is an unclassified activity initiated under the auspices of the Joint Directors of Laboratories (JDL) Technology Panel for Command, Control and Communications (TPC3). The Networks and Distributed Processing (N&DP) sub-panel of the JDL technology panel identified the issue of survivable information processing and distribution networks as one of several research areas. The Tri-Service Experiment is conducted at three sites: Rome Laboratory (RL), Naval Ocean Systems Center (NOSC), and Communications - Electronics Command (CECOM).

2. Experiment Platform

The Tri-Service Experiment established a platform for the development of distributed, survivable applications. This platform consists of a distributed computing environment (Cronus[2]), a synchronization mechanism, a universal storage mechanism, and a set of displays. On top of this platform, each service built a distributed target tracking simulation, based on typical command and control requirements. The platform provided the basis for integrating the three service applications into a unified whole.

2.1 Cronus Distributed Computing Environment

Cronus, developed by BBN Systems and Technologies Corporation, is an environment for the development and

operation of distributed applications. It provides a coherent and integrated systems approach to the development of computer applications which will be spread among several computing resources. Cronus runs on a variety of different heterogeneous hardware bases and operating systems. Cronus operates as a set of user-level processes within the native operating system of the hardware base. By executing at this level, application developers can utilize both the support facilities of their local computing environment and the remote support facilities of Cronus.

By using an object-oriented approach, application components or modules within the Cronus environment are highly transportable. Application components are called *managers*. Each manager is a separate self-contained process that is responsible for manipulating some data (a.k.a. *objects*), and which has an address (a.k.a. a *unique identifier*) through which it can be contacted. Managers accept requests, called *operations*, by other managers or clients. Operations instruct the manager to perform a pre-defined sequence of instructions. Managers can be relocated or replicated at different computers by recompiling the source code. Once compiled and executed, a manager can be seen and reached by every other networked computer system that is running the Cronus software environment.

2.2 Experiment Synchronization

The Tri-Service Experiment offers a remote timing mechanism called the *Timer Manager*. Time, in the Tri-Service Experiment, is measured by *tick* intervals. A tick roughly corresponds to a second interval. When requested through an

operation, the Timer Manager provides the number of ticks which have elapsed since the start of any Tri-Service Experiment test or demonstration. Thus, events in the Tri-Service Experiment can be coordinated.

2.3 Experiment Storage

The Tri-Service Experiment offers a remote storage capability through a software module called the *SimulationData Manager*. This storage capability can be used by experiment components to store information, share information, or both. Application components fill a buffer with the desired information to be stored. Application components also fill a template describing the information to be stored. Both the template and buffer are sent to the *SimulationData Manager*, where they are categorized and stored within relational database systems (Informix, Sybase, Oracle) at remote sites. The *SimulationData Manager* also accepts LookUp requests for information. These requests are sent to the manager in the form of a relational database query, based upon the Standard Query Language (SQL) interface.

3. Prototype Application

The Air Force application is based on a previously developed command and control application called C2 Internet[3]. Software components of that original work were updated and extended to operate within the framework of the Tri-Service Experiment. The scenario for the Air Force application is a high altitude or orbiting sensor platform which is observing the Mediterranean Sea region. The targeting information provided by the sensor is filtered by the Target Filter manager for content and accuracy. It is then fed into the universal storage mechanism provided by the Tri-Service Experiment platform. The Air Force application synchronizes with the rest of the Tri-Service Experiment by accessing the Timer manager.

3.1 Sensor Manager

The Sensor Manager is one of the cornerstone managers of the Tri-Service Experiment. The role of the Sensor

Manager is to get target information from the Target Simulation Manager, process it, and forward it to the Target Filter Manager. Additional information is obtained through the Timer and Weather Managers.

Sensor objects represent an airborne or satellite observation platform. The Sensor Manager uses the epoch time measurement to determine the location it should occupy and the field of vision that it can see. The occupied location and field of vision are considered to be the sensor's mission. Each mission has a start and stop time associated with it. A sensor can run multiple missions during the course of a simulation, as long as they do not involve occupying two different locations at the same time.

The Sensor Manager receives detection data from the Target Simulation Manager. In addition, it also queries the Weather manager for weather conditions that affect its visual region. Based upon the weather report, false detections are generated to simulate the effect of adverse weather conditions. These false detections are merged with the real detections to produce a detection report of the visual region. This detection report is forwarded to the Target Filter Manager for further processing.

3.2 Weather Manager

The Weather Manager maintains information about the state of weather for a particular region. It allows the user to create weather reports which will be used throughout the simulation by the Sensor Manager. The Weather Manager will periodically query the Timer Manager for the epoch time so that weather information can be stored in the Simulation Data Manager for use by the User Interface.

When the Weather Manager receives a *GetWeatherInRegion* request it queries the Timer Manager, replies to the requester, and then notifies the Simulation Data Manager of the new or updated weather report. Typically, only after a time-out interval, will the Weather Manager invoke a Create or Update operation on the Simulation Data Manager to supply information to the User Interface.

3.3 Target Simulation Manager

The Target Simulation Manager works in conjunction with the Sensor and Weather Managers to emulate an Air Force airborne or satellite observation platform where raw observations are generated, processed, and reported. Clock synchronization for these components is provided by the Timer Manager.

The Target Simulation Manager is responsible for providing target detections to the Sensor Manager. The Target Simulation Manager implements the `GetTargetInRegion` operation for the exclusive use of the Sensor Manager. Upon receipt of the operation, the Target Simulation Manager will invoke the `GetEpoch` operation on the Timer Manager. The Target Simulation Manager will then use the results from the Timer Manager to return a list of the targets for a particular region.

3.4 Target Filter Manager

The Target Filter Manager receives all detections from the Sensor Manager, filters out false detections, and stores real targets within the Simulation Data Manager. In addition, if there are other Target Filter Managers running, it will update them as it stores targets within the Simulation Data Manager. A detection is identified as either *real* or *false* based on the `TargetUID` field. Detections which have a valid UID in the `TargetUID` field are considered to be real targets. Detections that have a NULL `TargetUID` field are considered to be false targets. Real targets are stored in the Simulation Data Manager. False targets are discarded in this version of the Tri-Service Experiment.

In future versions, for a more realistic simulation, false targets might be stored internally and checked against any previous false detections. A false detection could then be upgraded to semi-real target status if, for example, it: (1) was detected twice within a 50 cubic mile region of the original location, (2) had the same target description, and (3) had the same target type. Such semi-real targets would be stored within the Simulation Data Manager as unknown threats.

The Target Filter Manager also maintains various statistics. It keeps track of the number of detections processed, the number of real and false detections arising from the total detections processed, the total number of `FilterDetection` invocations, the number of `SensorData` records, sensor identifiers, and the number of detections sent by each sensor. In addition, the Target Filter Manager provides the capability to remove any history of previous targets that it has seen in the simulation.

4. Reliability Issues

When making an application reliable, one of two approaches must be taken: either ensure that the hardware utilized by the application is fault resistant, or manage the replicated copies of your application software necessary to contend with hardware faults. In studying this application, replication of key components was chosen to increase the reliability measure. A review of the hardware failure effects on the AF application was undertaken.

The Weather Manager is not a critical component in the application. Failure to receive weather information does not impair the ability of the application to provide meaningful data, so the Weather Manager has not received any additional attention.

The Sensor Manager is a vital and critical component within the application. Failure of the Sensor Manager would result in an inability to collect targeting information. Given the design of the sensor components, there is no need to replicate information to multiple copies of the Sensor Manager that may be executing. To imitate sensory devices, each Sensor Manager operates independently. Thus, reliability can best be attained by executing simultaneous copies of the Sensor Manager and assigning some level of overlapping coverage to the surface area that they observe.

The role of the Target Simulation Manager is to provide the targets that will drive the

AF application. The sensor components use the target information to create the tracking information used in other portions of the application. Thus, the Target Simulation Manager is critical to the operation of the application. It is insufficient to replicate the target simulation functionality by having copies of the Target Simulation Manager executing at the same time. The target database must also be replicated. Creation of new targets and updates to existing targets are made on one database, then copies of the new or updated target are spread to all copies of the target database. To enforce consistency (the measure of *sameness* among copies of the database), read-write quorums and version vectors have been established on the target databases.

The TargetFilter Manager receives target track information from the sensor components. This track information is scanned for accuracy and submitted to the experiment storage facility. For each target track submitted to storage, a key is kept that allows updates to be made to the target track. In attempting to make the TargetFilter Manager more reliable, this key must be maintained among any replicated copies of the manager that are executing. To simplify the key management, TargetFilter Managers that fail are not replaced by new managers. Thus, all TargetFilter Managers must be available at the start of the application. As new keys are acquired from the storage facility, the information contained within the key is passed to all TargetFilter Managers that are running. If any target filters fail, the other copies can continue to process information for the failed copy since all copies of the manager have the vital key information.

5. Current Efforts

Currently, the SimulationData manager is capable of storing data in any one of three databases, i.e., Informix, Sybase, or Oracle. This is inadequate from the standpoint of survivability since loss of the SimulationData manager or loss of the database would be fatal to the experiment.

Efforts are presently underway to replicate the SimulationData manager and to enable the SimulationData manager to store data in all three databases.

A startup configuration expert system is presently under development. It is being written in the C Language Integrated Production System (CLIPS). CLIPS is a forward-chaining rule-based language in which data points are stored as facts and the knowledge base is stored as rules. Overall execution is controlled by an inference engine. This expert system will help to automate the experiment setup process by assessing connectivity and resource status.

Cronus 2.0 has recently been released. The Tri-Service Experiment modules are currently being upgraded in order to run under Cronus 2.0.

Documentation of the Tri-Service Experiment is currently being written. A Maintenance manual and an Installation/User manual will be published. In addition, an annual Technical Report will be produced.

6. Future Work

One limitation in the current AF application is the lack of feedback information. Currently, the target track information flows into the experiment storage facility, but, nothing other than the display uses the information. Ideally, decision aid components should utilize the information, perhaps to generate threat assessments or intercept sorties.

Another look at the TargetFilter Manager should also be made. The filtration algorithm is simplistic and should, ideally, be replaced with some type of correlation filter. Also, a better solution should be found to the key management problem. Allowing new Target Filter Managers to start while the application is underway would greatly increase the reliability characteristics of this part of the application. Finally, development of a resource monitoring facility is planned. This

capability would enable the Tri-Service Experiment to perform real-time configuration management for both

survivability and performance-driven adaptability.

End Notes

[1] Gadbois, M., and Anthony M. Newton, *Tri-Service Distributed Technology Experiment*, 1990 Symposium on Command and Control Research, SAIC-90/1508, pp. 150-157, June 1990.

[2] Schantz, Richard E., and Robert H. Thomas, *Cronus Functional Definition and System Concept*, BBN Systems and Technologies Corporation Report No. 5879, 40p., September 1989.

[3] BBN Systems and Technologies Corporation, *C2 System Internet Experiment: System/Subsystem Specification*, Final Technical Report, RADC-TR-88-133, Vol. III, June 1988.

DAIT: Distributed Application Instrumentation Tool

*Cheryl L. Blake
Vaughn T. Combs*

*Computer Systems Branch (C3AB)
Rome Laboratory
Griffiss AFB, NY 13441-5700*

1. Overview

As applications for parallel and distributed computing continue to grow in complexity, the need for tools which provide information on the overall execution of these applications is becoming critical. To meet this need, the Distributed Systems in-house group is building a flexible monitoring environment to aid application designers in designing and understanding the behavior of their distributed applications. The tool consists of a general query based system for the collection of events that occur within an application, and a display system for processing and presenting the events. These events are collected through the use of probes which register events with a distributed database. Procedure calls to these probes are embedded within the application code to be instrumented in order to mark the occurrence of a specific event. The instrumentation tool provides a library of predefined probes, which are based on events that adhere to the object/thread model, and also allows for user-defined probes. The provided display represents these object/thread interactions. The design is flexible so as to permit a wide range of events and displays to be used with the tool.

The DAIT is designed to be a multi-purpose tool. It can be used to show interactions between distributed objects within an object-oriented application and to show how the application reacts to changes in the distributed environment. Although not designed as a debugger, it can aid in debugging an application by providing

information about object states and interactions which can show unexpected behavior. The DAIT is a monitoring device which allows a user to monitor application events. It does not provide control over program flow or state variables. The tool is designed to run at application run-time, but does not provide real-time monitoring. Instead, the tool will collect event information as the application is running and present it when requested. Events can be viewed at run-time; but delays must be expected in gathering and presenting information.

2. Architecture

The DAIT architecture consists of four subsystems: the Extraction subsystem, the Collection subsystem, the Display subsystem, and the Instrumentation Information Service. The extraction subsystem is responsible for the detection of events and for reporting these events to the collection subsystem for storage. The collection subsystem is responsible for accepting information from the extraction subsystem and storing it in a logical fashion. The collection subsystem uses multiple collection objects to allow the extraction subsystem to offload events quickly thus reducing the intrusiveness of the instrumentation tool. The collection subsystem also processes queries received from the display subsystem to retrieve the stored event information. The display subsystem provides data processing routines for organizing and formatting the data received from the collection subsystem. It also provides a user interface for presenting the information to the user.

The application information service is used to provide and maintain information associated with the instrumentation events and instrumented applications that are running in the environment. These four subsystems are integrated to provide the monitoring functions of the DAIT.

2.1 The Extraction Subsystem

The extraction subsystem serves as the interface between the user's application and the collection subsystem. It consists of predefined and/or user-defined probes which have been embedded into the application code. These probes must not adversely affect the execution of the application code. To accomplish this the probes must be designed so that they do not change any variables of the application or alter the flow of execution. They must also be designed to be minimally intrusive. For the provided probes this requires minimizing the amount of internal processing done in forming an instrumentation event to be shipped out to the collection subsystem. This is controlled for the provided probes. Since we have no control over the probes that the user defines himself, it is up to the designer of user-defined probes to adhere to this requirement or to recognize the potential for adverse effects on the execution of his application if they are not adhered to. In order to aid the user-defined probe designer, we have provided a library of procedures for the extraction of event information and have provided a skeleton probe that can be used by the designer to minimize these affects.

Six events are provided by the instrumentation tool which map to an object-oriented view of the application. These events are: State Change, Thread Create, Thread Exit, Thread Entrance, Client Thread Entrance and Client Thread Exit. These allow the user to monitor the critical interactions between objects and the effect of these interactions. The object state information, which is to be collected for the State Change event, is dictated by the user. This allows the user to see only the state information which is of importance to him and reduces the amount

of storage that would be needed to provide the entire object state.

Threads are the impetus for changing state within an object. A Thread Create event is formed whenever a new thread of processing is created. The Thread Exit event occurs when this thread leaves the boundaries of an object as it traverses to another object. A Thread Entrance event occurs when the thread arrives at the boundary of another object.

Clients are a Cronus manifestation which differ from object managers in that they do not have any object instances. Clients are high-level language programs which invoke operations on object managers, but can not have invocations made on them. Thus the Client Thread Entrance and Exit events map to the Thread Entrance and Exit events. These six probes are automatically provided by the instrumentation package and do not need to be inserted into user's application code. They can be enabled within the application's manager initialization code.

2.2 The Collection Subsystem

The collection subsystem includes collection objects and data storage objects (DSO). The collection objects are essentially a store and forward capability. Each instrumentation probe embedded within the application objects sends events to a collection object. The collection object simply acknowledges receipt of the event and buffers the event. When a time or size threshold is reached the collection object ships the events that it has maintained to a DSO for further processing and storage. The DSOs essentially comprise a special purpose distributed database. They categorize, store, and retrieve event data. The display subsystem can then query for particular events or categories of events needed for the display. Some of the categories include: events that have not been seen previously, events associated with a particular object or thread, and events associated with a particular application. Query modifiers such as *and*, *or*, and *not* can be used within the query.

2.3 The Display Subsystem

The display subsystem provides routines for the sorting and processing of data from the collection subsystem as well as a mouse/menu driven interface to the user. This subsystem consists of a session object which provides event processing and the user interface. The DAIT is not tightly bound to the provided display subsystem and has been designed to allow a different display package to be integrated into the collection subsystem. The display subsystem provided is designed to provide an object-oriented view of the interactions within an application based on the provided probes. Limited monitoring is provided for user-defined events (i.e. textual representation).

The user interface provides two presentation options to the user: application play/replay and thread trace. Application play/replay provides a representation of all instrumented events occurring in the user's application. Thread trace provides a representation of all instrumented events which happen along a particular line of processing. Both presentation options provide information on objects state, represented as values of internal variables, and inter-object operation history, listing operations invoked between objects. Strict ordering is maintained at the object level so that all events along a single line of processing will be presented in the order in which they occurred. Objects are represented on the display as circles and operations as arrows between objects. Objects can be grouped and named by the user to assist in recognizing and understanding the application behavior. The user interface provides options to freeze the representation and step through at any time.

The session object formulates queries for the DSOs based on what the user is currently viewing, queries the DSOs, orders the response and formats the response for the display. The session object is not critical to the DAIT and may be left out. If this is done, however, the display must generate queries and the events returned will not be ordered or grouped in any way. The session object primarily serves to

relieve the processing burden of the display since graphics are notoriously CPU intensive.

3. Hardware/Software Requirements

The DAIT runs within the Cronus distributed operating environment and is compatible with the set of heterogeneous hardware and operating systems supported by Cronus (Unix, VMS, Mach, etc./Sun 3.4, MASSCOMP, VAX, etc.). Cronus 1.5 is currently used, but the tool is upgraded to run with Cronus 2.0. A root level make is available to allow easy compilation of the instrumentation package for the installer, and libraries are provided to aid the developer in instrumenting his code. Overall, the package has been designed to flexibly run on a commonly used set of machines and operating systems and allows users to define the information to be monitored which they feel is important or meaningful to their understanding of their application's execution.

Object Translation into Relational Databases

Anthony M. Newton

Computer Systems Branch (C3AB)
Rome Laboratory
Griffiss AFB, NY 13441-5700

1. Overview

The ideas discussed within this paper are a subset of broader work produced by the Tri-Service Distributed Technology Experiment[1]. The Tri-Service Experiment is an unclassified activity initiated under the auspices of the Joint Directors of Laboratories (JDL) Technology Panel for Command, Control and Communications (TPC3). The Networks and Distributed Processing (N&DP) sub-panel of the JDL technology panel identified the issues of survivable information processing and distribution networks as one of several research areas. The Tri-Service Experiment is conducted at three sites: Rome Laboratory (RL), Naval Ocean Systems Center (NOSC), and Communications-Electronics Command (CECOM).

2. Competing Philosophies

Within the experiment, one of the issues to be faced in designing a universal storage system, which utilized off-the-shelf database technology, was the conflict between the view of the distributed computing environment and the relational database interface.

2.1 Cronus Distributed Computing Environment

Cronus [2], developed by BBN Systems and Technologies Corporation, is an environment for the development and operation of distributed applications. It provides a coherent and integrated systems approach to the development of computer applications which will be spread among several computing resources. Cronus runs on a variety of different, heterogeneous hardware bases and operating systems. Cronus operates as a set of user-level

processes within the native operating system of the hardware base. By executing at this level, application developers can utilize both the support facilities of their local computing environment and the remote support facilities of Cronus.

By using an object-oriented approach, application components or modules within the Cronus environment are highly transportable. Application components are called *managers*. Each manager is a separate self-contained process, that is responsible for manipulating some data (a.k.a. *objects*), and which has an address (a.k.a. a *unique identifier*) through which it can be contacted. Managers accept requests, called *operations*, by other managers or clients. Operations instruct the manager to perform a pre-defined sequence of instructions. Managers can be relocated or duplicated at different computers by recompiling the source code. Once compiled and executed, a manager can be seen and reached by every other networked computer system that is running the Cronus software environment. Managers are also the only means of providing access to the object. Nothing, other than the manager responsible for an object, can access the object directly. Any client or manager that wishes to access an object must use the operation defined and supported by the object's manager.

2.2 Relational Database Environment

The three databases that were part of the experiment were Informix, Oracle, and Sybase. Each provided access to the data stored by using a Standard Query Language (SQL). For databases, in general, and for relational databases, it is true that the

structure of data is of critical importance to the database environment. Relational databases optimize storage and retrieval of the data based upon the fields having some common link (that is to say, some fields in the database are related to other fields). To facilitate this, the database requires users to define the construction of the data fields that will be stored. Furthermore, references into the database are highly specific and structured, so, anyone who accesses the database must: (1) know the structure of the database and (2) be responsible for the changes made to the database.

3. Model Integration

When contemplating the integration of both the object and relational philosophies, it is best to start at a common point. As discussed above, the object model and the relational database model differ in one fundamental way. The object model is built presuming that the data and its structure is hidden from the user's view. The relational database model, however, presumes that the user is intimately familiar with the format and structure of the data. One common point that can be found is the manner in which objects and relational databases are constructed. Structure is the key element. An object needs structure defined so that the manager of the object can properly manipulate the object. A relational database imposes structure to optimize storage and access characteristics.

3.1 TypeDefinition vs. DatabaseSchema

In Cronus, objects are classified according to object type. The object type defines the structure of the data contained within the object. TypeDefinition is the name given to the process of declaring a new object type. The manager in Cronus which maintains information on all object types is called the TypeDefinition Manager. By accessing the TypeDefinition manager, managers can understand the structure and content of the objects they manage. It is therefore possible to take this information and use it as the basis of a conversion algorithm

which can map the object structure into a schema that the relational database can understand.

One element of contention is the ability of object structures to contain other object structures. This is the ability to create compound/complex datatypes within an object. The relational database, however, does have a similar perspective. While one part of a database cannot "hold" another part, parts are connected (related) to one another. So, it is possible to look at the structure of objects and create a relational map that describes the ways in which certain object datatype structures are connected.

The solution is formulated by breaking the boundary that surrounds an object. Within the object type structure, the *canonical datatype* defines the structure(s) contained within the object. Each of these canonical datatypes would translate to a database table (the corresponding element of structure within relational databases). Each record maintained under that table would correspond to part of the object being stored. If the object contained only one canonical datatype, the record in the database table would contain the entire object data. The unique identifier that is part of every object would be used as the primary key for all records. Thus, by using the unique identifier and applying it in a query to all tables, it is possible to reconstruct all of the object data. This allows other parts of the system which operate within the object model to access the object data using familiar terms. Alternate keys could be formed by performing pattern matches on the canonical datatype fields to detect similarities. While field names might vary wildly in a general purpose system, in command and control systems differing objects tend to present common field names (e.g. longitude, latitude, speed, direction, etc.). Likewise, mechanisms could be provided which would define relations between canonical datatype fields at the time the first datatype is stored. A locking mechanism would provide the serial access to objects that is normally provided by the object manager.

4. Tri-Service Experiment Links

The Tri-Service Experiment established a platform for the development of distributed, survivable applications. In addition to the Cronus distributed computing environment, this platform consists of a synchronization mechanism, a universal storage mechanism, and a set of displays. On top of this platform, each service built a simulated application, based on typical command and control requirements. The platform provided the basis for integrating the three service applications into a unified whole. The scenario for the AF application is a high altitude or orbiting sensor platform which is observing the Mediterranean Sea. The Navy implemented an application to simulate the on-board tracking systems of their surface vessels. The Army scenario simulated the operational reports coming from multiple, mobile, land-based command posts.

So, to effectively integrate the three service applications, the Tri-Service Experiment needed a remote, distributed, and universal storage capability. This storage capability would accept different object structures from applications running in the distributed environment and integrate them into a single, federated database that could be used and shared by all three services as a data source. This software module was called the SimulationData Manager.

In its earliest form, the SimulationData Manager allowed the services to store information without regard to the underlying structure of the data in the object. This was accomplished through the use of the native Cronus object database facility. When upgrading this manager to use the relational databases, a two phased approach would be used.

In the first phase, the translation between object types and relational database schemas were hand-coded and implemented. The SimulationData Manager formed the object model interface with the relational database capability. Application components filled a buffer with the desired information to be stored. Application components also filled a template describing the information to be stored. Both the template and buffer were sent to the SimulationData Manager, where they were categorized and stored within relational database systems at remote sites. The SimulationData Manager also accepted LookUp requests for information. These requests were sent to the manager in the form of a relation database query, based upon the Standard Query Language (SQL) interface.

In the second phase, the SimulationData Manager would be extended to automatically perform the conversion between object types and relational database schemas by the methods discussed above. This phase is not yet scheduled.

End Notes

[1] Gadbois, M., and Anthony M. Newton, *Tri-Service Distributed Technology Experiment*, 1990 Symposium on Command and Control Research, SAIC-90/1508, pp. 150-157, June 1990.

[2] Schantz, Richard E., and Robert H. Thomas, *Cronus Functional Definition and System Concept*, BBN Systems and Technologies Corporation Report No. 5879, 40p., September 1989.

Information Exchange in Allied Operations

Francis A. DiLego
Jerry L. Dussault
Anthony M. Newton

Computer Systems Branch (C3AB)
Rome Laboratory
Griffiss AFB, NY 13441-5700

1. Overview

The application design discussed within this paper is a subset of broader work produced by the Australian Experiment, an unclassified activity initiated under the auspices of the 1988 Memorandum of Understanding (MOU 88/102) "Cooperative Communicating Networks" and partially funded by the US government under the Nunn Amendment. Rome Laboratory (RL), in the United States, and the Electronics Research Laboratory (ERL), in Australia, are the two participating agencies.

The Australian Experiment seeks: (1) to establish new networking technology based on the research topic of policy-based routing, (2) to implement a distributed application across multiple networks in the US and Australia, and (3) to experiment with the issues involved in sharing information and resources in a command, control, and communications environment.

2. Unified Environment

One of the problems with current, joint allied operations is the (clumsy) way in which data is exchanged. Differing data formats and classifications make direct information transfer difficult at best. The environment that we are creating would allow for the transparent transfer of data among multiple applications within a distributed environment.

2.1 Cronus Distributed Environment

Cronus, developed by BBN Systems and Technologies Corporation, is an environment for the development and

operation of distributed applications. It provides a coherent and integrated systems approach to the development of computer applications which will be spread among several computing resources. Cronus runs on a variety of different, heterogeneous hardware bases and operating systems. Cronus operates as a set of user-level processes within the native operating system of the hardware base. By executing at this level, application developers can utilize both the support facilities of their local computing environment and the remote support facilities of Cronus.

By using an object-oriented approach, application components or modules within the Cronus environment are highly transportable. Application components are called *managers*. Each manager is a separate self-contained process, that is responsible for manipulating some data (a.k.a. *objects*), and which has an address (a.k.a. a *unique identifier*) through which it can be contacted. Managers accept requests, called *operations*, by other managers or clients. Operations instruct the manager to perform a pre-defined sequence of instructions. Managers can be relocated or replicated at different computers by recompiling the source code. Once compiled and executed, a manager can be seen and reached by every other networked computer system that is running the Cronus software environment. Managers are also the only means of providing access to the object. Nothing, other than the manager responsible for an object, can access the object directly. Any client or manager that wishes to access an object must use the operation defined and supported by the object's manager.

2.2 Policy-Based Information Transfer

With the development of the policy-based gateway, information can be routed through computer networks based on an increased number of factors. Traditionally, information is routed through computer networks by minimum distance calculations. Networks which are connected with gateways that implement policy-based routing algorithms are capable of moving information through networks based on the contents of the information. Thus, factors such as security, priority, classification, sensitivity, etc., may be combined to form a set of constraints (i.e. a policy) defining the conditions for transmittal and delivery of information.

3. Demonstration Application

The application chosen for demonstration will simulate a distributed regional surveillance system. To represent this environment, each country will have a distributed set of software modules that will simulate a particular surveillance platform (e.g., an Over-The-Horizon radar or AWACS). The configuration flexibility afforded by Cronus will be used to perform experiments in resource management and fault-tolerance.

3.1 Application Scenario

The scenario depicts a joint exercise between the US and Australian forces. Hostile incursions into the island chain off Australia's northern coast have resulted in the need for increased surveillance of the region which is being provided by numerous surveillance systems including Australia's OTH radar and a US AWACS. The transparent sharing of surveillance information between US and Australian forces will be demonstrated using the distributed system and Cooperative Communications Network. As the system's performance degrades due to network overloading or component failures, the system will be configured to maintain adequate performance and insure continued access to surveillance data. The need for more dynamic interaction between

system operators and network managers will be investigated.

The scenario will demonstrate: (1) distributed processing, (2) information access control, (3) transparent access to distributed data, and (4) information exchange using policy-based networks (i.e., routing enforced by policy-based gateways) which are subject to dynamic change. Both the American (US) and Australian force roles can be reversed.

3.2 Application Components

Each site will have a set of networks interconnected by policy-based gateways to simulate an internet environment. The two internet environments will be connected with a set of policy-based gateways to allow communication between the internets. The Cronus distributed computing environment will be installed on machines in both internets and will be used to create an application across both sites. Traffic injectors will probably be used to simulate varying system and network loads during the experiments.

The software structure for the OTH radar application will consist of a set of sensory modules, a set of tracking filters, and a database repository. Multiple, independent sensory modules will report to a set of cooperating tracking filters. The tracking filter modules will formulate and identify hostile and friendly tracks, then store the information within the database repository. For survivability, the database repository should be replicated.

The software structure for the AWACS radar application will consist of modules which emulate the AWACS plane and a database repository. The AWACS modules will combine sensory and filtration functions so that each AWACS module will execute independently. The AWACS modules will report tracks to a replicated database repository which is separate from the OTH database repository.

Each database repository will be constructed with facilities for exchanging information between them. These

database repositories will be accessed and manipulated by a Mission Data Manager . There will be multiple mission data managers each managing a country's associated database. When data is exchanged between countries, the Mission Data Manager will perform an access

control check to determine whether the information can be exchanged. Separately, the policy based gateways will arbitrate to see if the information can actually pass between sites given the current network topology and policy agreement.

The designs and ideas expressed in this paper are in the initial stages of development and are subject to change.

An Informix Analysis and Evaluation

Francis A. DiLego
Robert Flo

Computer Systems Branch (C3AB)
Rome Laboratory
Griffiss AFB, NY 13441-5700

1. INTRODUCTION

The main objective of this in-house task was to conduct an analysis and evaluation of INFORMIX-SQL. This evaluation was done in order to determine the system's strengths, weaknesses, ease of use, and potential in future in-house R&D efforts.

INFORMIX is an off-the-shelf relational database management system (RDBMS). It runs in either stand-alone or networked environments on a number of UNIX-based computer systems. User-designed applications can be developed using INFORMIX-SQL application tools, which is based on the ANSI-standard Structured Query Language (SQL) to provide access to the INFORMIX-SQL databases. INFORMIX-SQL supports commercial database applications, such as on-line transaction processing, reporting, and decision support. Applications may reside on the same UNIX system as INFORMIX-SQL. INFORMIX-SQL may also be accessed transparently from other UNIX servers or UNIX and PC workstations across a network.

2. FEATURES OF THE INFORMIX SYSTEM

The following is a list of the major features of INFORMIX-SQL. According to the INFORMIX-On-LineTech Sheet, supplied by Informix Software, Inc., INFORMIX-SQL has the following capabilities:

- 1) Optimizes the processing of high volumes of data from very large databases shared by many concurrent users.
- 2) Maximizes the performance benefits of parallel processing on multiprocessor UNIX operating systems.
- 3) Employs advanced query optimization to perform complex queries used in reporting and decision support.
- 4) Supports applications that require database availability 24 hours a day, seven days a week.
- 5) Features an advanced logging, checkpoint, and recovery scheme that will automatically recover databases after system interruption.
- 6) Utilizes the UNIX "shared memory" facilities.
- 7) Supports its own disk mirroring capability which protects data from magnetic disk failures.
- 8) Supports the "client-server" model.
- 9) Supports multimedia databases.

3. INVESTIGATION METHODOLOGY

The following is a brief overview of the INFORMIX-SQL evaluation process. Each step of the evaluation process is discussed in detail, with the results of each test used as criteria to evaluate the INFORMIX-SQL system in its aforementioned capacity.

- 1) **Documentation.** The initial step in the INFORMIX-SQL evaluation was to review all available and relevant INFORMIX-SQL documentation, the INFORMIX-SQL Reference Manual and the INFORMIX-SQL User's Guide. Each of these documents was read and evaluated completely.
- 2) **INFORMIX-SQL Installation.** After we reviewed all pertinent INFORMIX-SQL documentation, we then proceeded to install the INFORMIX-SQL software on the Sun Microsystems 3/260 and 4/100 workstations running on the UNIX operating system here at Rome Laboratory.

- 3) **INFORMIX-SQL Familiarization.** After successfully installing the software here at Rome Laboratory, we then proceeded to familiarize ourselves with INFORMIX-SQL by implementing, executing, and testing the various INFORMIX-SQL capabilities that are outlined in the INFORMIX-SQL User's Guide and experimenting with the sample demonstration examples that were provided with the INFORMIX-SQL software.
- 4) **Install Testing Databases.** This task involved a survey of local databases that showed potential for use in the testing routines used to evaluate INFORMIX-SQL capabilities. After surveying the local community for possible testing databases we chose several based upon, among other factors, size, data, and format.
- 5) **Design Testing Routines.** Throughout this task when necessary, we designed and coded testing routines to evaluate INFORMIX-SQL's capabilities, functionality, and performance. The testing routines were grouped into ten (10) areas the results of which are documented fully in the Evaluation section of the technical memo(RL-TM-92-1 "An RDBMS Evaluation"), each under the heading of the testing area name (e.g., Test 4 - Environment).
- 6) **Perform Tests.** This task involved performing the tests that were designed in the previous task, and documenting the results.
- 7) **Test Evaluation.** In this task, we reviewed the results of the tests performed and the tests themselves. We also checked for errors in both the testing procedures and INFORMIX-SQL's responses to the testing procedures. Retesting was done when deemed necessary.
- 8) **INFORMIX-SQL Evaluation.** In this task we analyzed, interpreted and commented on the results of the INFORMIX-SQL testing.

4. EVALUATION

In our analysis and evaluation of INFORMIX-SQL we determine its ease of use and potential for use in future in-house R&D efforts. The strengths/weaknesses of

INFORMIX-SQL as a relational database management system are also noted. Each of the following ten (10) sections represents a different, yet specific, testing area. We thoroughly discuss the testing procedure (i.e., what we did and how we did it), our rationale for testing this area of INFORMIX-SQL, and report the results of the testing and evaluative procedures (i.e., what happened, what should have happened) in the technical memo.

In keeping with our investigation methodology listed above, when reviewing the documentation we found it to be adequate with room for improvement. This is with reference to the documentation of errors. This area of the documentation was somewhat confusing and cumbersome to work with. This fact, however, did not become apparent until the testing phase of the evaluation. This aside, the documentation of features and examples of use were most acceptable. Overall, we found each manual contained useful information that could be used as reference material throughout our testing procedures.

The installation task for INFORMIX-SQL is trivial and only a limited knowledge of the constituent operating system is required. The installation procedure was automated and following the installation guide provided was not difficult. The installation went smoothly and no problems were encountered.

At this point we proceeded to familiarize ourselves with some hands-on training. We followed the beginners guide and completed the sample demonstration examples provided. This went very well. We also attempted to do things (e.g., Table updates, deletes, joins, creations other SQL commands) that were not part of demonstration workout but were along the same lines as the examples provided. We found everything to execute as expected.

The test database chosen contained many data formats of various length, many tables, and it varied in the amount of data contained in each table. The database we choose originated from an Oracle RDBMS.

An ASCII dump of the tables was made and one of the utilities supplied with INFORMIX-SQL, namely the dbload utility, was used to convert the data into the INFORMIX-SQL format. Once this was done we could then move on to the testing area of the evaluation.

The following is a list of the ten tests which we designed for the evaluation.

- 1) Data Type Boundary Values
- 2) Case Sensitivity
- 3) Operational Semantics
- 4) Environment
- 5) Error Detection, and Correction Mechanisms
- 6) Capacity
- 7) System Degradation
- 8) Security Features
- 9) Recovery Capabilities
- 10) SQL Grammar

When we implemented these tests, we documented the results for later analysis. Not only did we cite our results in the designated test areas but we also commented on what would have been nice to have, and our overall impression of what resulted. Again, the results of this evaluation were documented more fully and published in a technical memo (RL-TM-92-1 "An RDBMS Evaluation").

To summarize, the testing and evaluation phases of this effort went smoothly. INFORMIX-SQL on the whole performed in a manner consistent to its intended design specification. This is not to say, however, that INFORMIX-SQL did not have its flaws or draw backs. For instance, INFORMIX-SQL does not support nested transactions. This is a feature which is nice to have since it adds versatility for the user and can save time when searching for flaws in a transaction log relative to rollforwards and rollbacks. In the cases where we ran into problems while testing it was usually a problem with the supplied user interface or error detection and correction mechanisms. The user interface supplied with the software can be cumbersome to work with at times. The controlling key strokes changed from window to window and from one thing to another within one

window of execution. In the case of error detection and correction mechanisms, the problems came from undocumented errors and anomalies. For example, where errors are concerned, we managed to generate errors that were not listed in the reference manual. These errors would show up on the screen but would not have an associated listing in the reference manual so that a solution to correct the problem could be sought. An example of what we refer to as an anomalies was when we compiled a form specification file. The file used as a form specification file for an INFORMIX-SQL form was really an executable program. We did this deliberately to see what kind of errors would occur. This was done simply by putting a .frm extension on the "bogus" file at the operating system level; Then entering back into the INFORMIX-SQL environment and telling it to compile this file. The thing to note here is that the software tries to help the user by entering encountered errors into the form specification file for easy debugging. This is a very useful tool. However, this compiler, unlike many compilers, continued to generate an error file in "/tmp" even when a very large number of errors was encountered. This became a problem because the file grew beyond the remaining file systems capacity of 35 megabytes. The compilation should have stopped at some predetermined number of errors and notified the user to fix the errors already detected and then recompile to get the rest. So, as you can see there are a few bugs but nothing that we found was insurmountable. Therefore, because of its ease of use, relatively low cost, and performance gained in various areas that INFORMIX-SQL definitely has potential for use in C3 applications. We also feel that conclusions concerning INFORMIX-SQL as a RDBMS drawn from its performance in each of the testing areas, should be viewed from the point of view relative to the user's needs.

For the readers convenience, we have included , at the end of the published tech. memo (RL-TM-92-1 "An RDBMS Evaluation"), a "Glossary" and two "appendices" (Appendix A and Appendix B). Both the "Glossary" and the "Appendices" can assist the reader in further

understanding the scope of the in-house task. The "Glossary" includes a collection of important definitions relating to database technology. "Appendix A" included a discussion of the general characteristics and components of a relational (database) system. "Appendix B" includes a collection of figures (e.g., program segments, charts, diagrams) that are referred to throughout the report. The information included in these three sections will allow the user to become more familiar with databases, database concepts, and relational (database) systems.

DDES: Distributed Database Experiments

*Patrick M. Hurley
Terrance Stedman*

*Computer Systems Branch (C3AB)
Rome Laboratory
Griffiss AFB, NY 13441-5700*

1. Introduction

A survivable database management system (DBMS) would be desirable for many applications including military command and control, airline reservation, banking, and other systems where access to correct data is crucial. What is meant by survivable is that users will have continued access to correct data in spite of various types of failures, such as network partitions, local or remote host failures, and application or system software failures. The Distributed Database Experiments (DDES) project was undertaken in an attempt to produce a survivable DBMS using off-the-shelf hardware and software components.

2. Background

Availability and consistency are two fundamental issues involved with replicated database systems. Those which stress availability are interested in providing a high degree of access to the data. This high degree of access is attained by making a number of copies of the database reachable by its users. Issues involved with keeping all of the copies consistent are emphasized less because of the desire to provide fast access.

Consistent database systems, on the other hand, provide a high degree of confidence in the sameness of all copies of the database. In consistent systems, users will receive the same data from every copy of the database. This high degree of consistency is achieved by coordinating updates for all replicated database copies and involves considerable overhead. Issues involved with availability are stressed less because of the desire to maintain consistency.

As shown above, there is always a trade-off between availability and consistency because of the overhead required to maintain consistency. Consistent systems are less available because of this overhead and similarly available systems are not highly consistent because they delay this overhead in order to achieve fast access.

3. DDES Design Overview

A survivable DBMS will continue to function even in the event of hardware/software failures. This survivability can be attained by maintaining mutually consistent, replicated copies of the DBMS in a manner which is transparent to its users. These replicated copies must be kept on separate hosts in the network in order to achieve the desired level of survivability. The management involved with maintaining these replicated copies includes the underlying communication between them and some form of DBMS replication strategy. The use of a distributed operating system would handle the communication issues and quite possibly aid in the implementation of the replication strategy itself.

To design replication and consistency control mechanisms, the object model was considered. Briefly stated, the object model says that an object is comprised of both state information and rules to govern how the state information may be examined or changed. Our design strategy therefore is to encapsulate the DBMS into the object model, i.e., by considering the DBMS as the state information and defining rules to examine or change the data contained within the DBMS.

The replication strategy chosen uses full replication of the data because the DDES system stresses survivability. Partial replication was considered but not used because it is inherently less survivable than full replication. Another consideration in the design of the replication strategy is the trade-off between availability and consistency. Remember the definition of survivability defined in this paper is to ensure access to correct (most recent) data in spite of various types of failures. By stressing consistency, the DDES system will ensure that a user will never receive old, possibly incorrect data from database queries.

One way to maintain such a highly consistent system is with the use of a locking mechanism and transaction logging. The locking mechanism locks the data, allowing asynchronous updates if possible, and then releases the locks. The transaction logging is necessary to bring previously inaccessible databases back to consistency before they are restored to service.

4. DDES Implementation

While keeping the design issues in mind, a search was conducted to see what technology existed to support the implementation of such a system. The search resulted in the selection of the Cronus distributed operating environment and Informix, an off-the-shelf relational DBMS (RDBMS). Cronus was selected because it provides many desirable characteristics of a distributed computing environment. Cronus is object based, supports object replication, and has an object locate mechanism. Cronus currently provides support for the Oracle, Sybase, and Informix database systems with a user interface and development environment. The Informix RDBMS was chosen because it was readily available, however either Sybase or Oracle could have been used with minimal modifications to the system. All that is really required from the DBMS is an interface to Cronus and support for SQL (Structured Query Language). SQL was developed by IBM and has become an industry-standard database query language.

Cronus does, however, have its shortcomings. Cronus does not support replication of the Informix RDBMS which is a must from a survivability standpoint. This is because Cronus only provides an interface to the Informix RDBMS which means that Informix is not a Cronus object and therefore cannot be replicated. This is where DDES comes into play by providing the mechanism to maintain consistent replicated copies of the Informix RDBMS.

5. DDES Overview

The DDES system consists of a User Interface (Sunview) , Clients (Cronus), Transaction Managers (Cronus), RDM Replicated Database Managers (Cronus replicated managers used to maintain consistency of the replicated DBMS's), and N copies (two copies in our experiment) of the Informix RDBMS. Figure 1 shows the software layout of DDES and what type of hardware it runs on in our configuration. Each of the software modules are briefly described below.

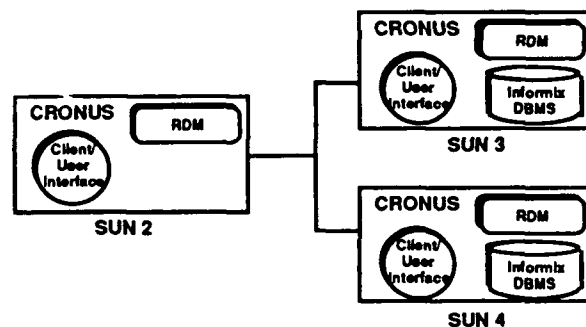


Figure 1: DDES Design Overview

The user interface and the embedded clients act as one functional system. The user interface is a mouse driven menu system that allows the user to select what transaction(s) to perform at the click of a button. Once the transaction has been selected, the user interface calls the appropriate embedded client to interact with the user. The client prompts the user for information required to perform the requested transaction. Once the client has all the required information, it invokes the appropriate Cronus Transaction manager to perform the transaction. The client also

displays the results of the requested transaction through the user interface.

The Cronus Transaction manager's job is to perform the transaction(s) requested by the client while maintaining database consistency. It accomplishes this job with the use of the RDM (Replicated Database Manager) which allows the Cronus Transaction manager to lock the table(s) necessary on each of the replicated databases. If the lock is successful on all accessible databases, the Transaction manager performs the transaction(s) in parallel on each copy of the replicated database. Once this transaction is completed on each database, the Transaction manager uses the RDM to release all locks obtained by the transaction. The results are then returned to the calling client.

The RDM is really the heart of the DDES system. It is mainly responsible for keeping persistent knowledge of the state of the DDES system. This state information is highly survivable because it uses Cronus replication. It keeps state information such as DBMS status which includes DBMS availability and table lock information. The RDM uses all this information to give locks to Transaction managers. The RDM also maintains transaction logs in order to automatically bring recovered DBMS's back to consistency.

6. Conclusion

The goal of this project was to produce a survivable DBMS. By using a simple design approach, existing technology, and a newly designed locking mechanism, we were able to obtain the desired level of survivability and therefore meet our goal.

In addition to survivability, we wanted the DDES system to be highly consistent in order to ensure that users would have continued access to correct data. We were able to maintain consistent, replicated copies of the database at all times. At no time were any accessible databases inconsistent with one another, thus guaranteeing that only correct data would be accessed. Inaccessible databases were brought back to consistency before

becoming accessible by applying updates from transaction logs.

Several experiments were performed on the DDES system to confirm that the desired level of survivability and consistency was achieved. These experiments consisted of subjecting the DDES system to various types of failures, such as network partitions, local host failures and several different software failures. A network partition failure was accomplished by manually removing a host from the network, a host failure was accomplished by abruptly bringing down a host, and software failures were accomplished by killing the process of the software being tested. The software failures tested included the user interface, the transaction manager, the RDM, the Informix RDBMS, and Cronus. Within limits, these failures were tested in combination with each other. As noted previously, the DDES system was able to maintain a desired level of survivability and consistency in spite of these various types of failures.

Although no formal bench marking was performed on the DDES system, good overall system performance was experienced due to updating the databases asynchronously. In the future, we need to benchmark the system to determine its overall performance. This bench marking will also aid in determining where future performance enhancements can be made. Thought has also been given to incorporating the DDES software into Cronus to make it easier to use and to possibly increase the speed of operation.