

AD-A124 379

SYSTEM ARCHITECTURAL CONCEPTS: ARMY BATTLEFIELD COMMAND 1/1
AND CONTROL INFORMATION UTILITY (CCIU)(U) JET
PROPULSION LAB PASADENA CA J FERREY ET AL. 25 JUL 82
NAS7-100 F/G 9/2 NL

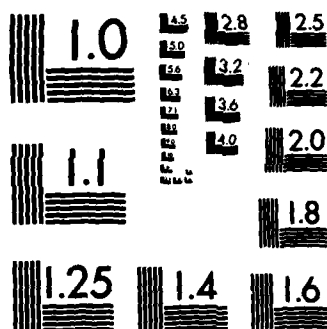
UNCLASSIFIED

END

FILED

1

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

7025-8

2

ADA 124379

Defense Information Systems Program

System Architectural Concepts: Army Battlefield Command and Control Information Utility (CCIU)

J. Fearey
R. King
E. Sadeh
J. Wedel

July 15, 1982

Prepared for
Department of the Army
Communications and Electronics Command (CECOM)
Engineering and Integration

Through an agreement with
National Aeronautics and Space Administration
by

Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California

D-125

This document has been approved
for public release and sale; its
distribution is unlimited.

DTIC
ELECTE
FEB 15 1983
S D E

DTIC FILE COPY

83 02 014 239

Incl 1

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A124379 N/A	3. RECIPIENT'S CATALOG NUMBER N/A
4. TITLE (and Subtitle) System Architectural Concepts: Army Battlefield Command and Control Information Utility (CCIU)		5. TYPE OF REPORT & PERIOD COVERED Interim Report July 1981-June 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) J. Fearey, R. King, E. Sadeh, J. Wedel		8. CONTRACT OR GRANT NUMBER(s) NAS 7-100
9. PERFORMING ORGANIZATION NAME AND ADDRESS Jet Propulsion Laboratory California Institute of Technology Pasadena, CA 91109		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 1L1 62701 AH 92
11. CONTROLLING OFFICE NAME AND ADDRESS US Army Communication-Electronics Command ATTN: DRSEL-SEI-F Fort Monmouth, NJ 07703		12. REPORT DATE 25 July 1982
		13. NUMBER OF PAGES 79
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Distribution unlimited. Release to general public.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Distributed Processing, Command Control Subordinate Systems, Command and Control Systems, Cellular Command Posts		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The U.S. Army Combined Arms Center has identified conceptual requirements for distributed processing in the Army's battlefield automated systems. The basic requirement of the CCIU is to provide a degree of survivability and continuity of operations essential for execution of the air/land battle. To satisfy this requirement, the US Army Communications-Electronics Command Center for Systems Engineering and Integration (CENSEI) and the Jet Propulsion Laboratory (JPL) have embarked on a program to develop an overall architecture for a distributed tactical command and control system for the Army. This		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

17012

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

report describes the Command and Control Information Utility (CCIU), which provides a "public utility" for command and control systems, based on distributed processing. JPL is currently involved in developing the initial phases of the CCIU portion of the CENSEI program. This document, the second in a series, presents concepts and preliminary analysis of the CCIU system architecture. Future revisions of this document will more closely reflect the actual testbed architecture and will emphasize more pragmatic than esoteric concepts. It is intended that this series of documents will converge on a distributed architecture satisfying the guidelines and functional needs of the CCIU. The development of this architecture will require a simplification, condensation and coordination of the material as presented in this document. In addition, concepts reflecting the knowledge gained from currently developed demonstrations will be integrated into the architectural structure. This latter information is important in the progression toward a realizable CCIU architecture capable of operation in a realistic military environment.

7

UNCLASSIFIED

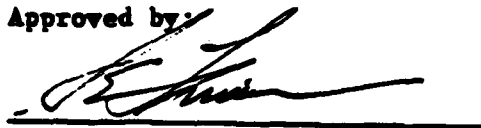
SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

7025-8

Defense Information Systems Program

**System Architectural Concepts:
Army Battlefield Command and
Control Information Utility (CCIU)**

Approved by:



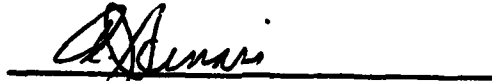
**S.E. Friesema
Manager,
CCIU Task**

Approved by:



**T.H. Thornton, Jr.
Manager,
Information Systems Division**

Approved by:



**A. J. Ferrari
Manager,
Defense Information Systems Program**

Prepared for

**Department of the Army
Communication and Electronics Command (CECOM)
Center for System Engineering and Integration**

**Through an Agreement with
National Aeronautics and Space Administration
by**

**Jet Propulsion Laboratory
California Institute of Technology
Pasadena, California**

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	



CONTENTS

Volume I	<u>Page</u>
<u>Executive Summary</u>	
1.0 Introduction	1
1.1 The CENSEI Development Program for a Distributed Tactical Command and Control Information Utility	1
1.1.1 Operational Need	1
1.2 The Distributed Processing Models	2
1.2.1 The Cellular Command Post DPM	2
1.2.2 The CCS ² DPM	3
1.2.3 The OFFAC-CONOPS DPM	3
1.2.4 The CCIU DPM	3
1.3 Program Details	4
1.3.1 Concept Development	4
1.3.2 Applications Demonstrations	4
1.4 The Role of CCIU in the CENSEI Program	5
1.4.1 The SAC-Doc	5
1.5 Demonstration and Testbed Development	6
2.0 System Architecture	7
2.1 Objectives	7
2.1.1 CCIU Hardware	7
2.1.2 Survivability and System Integrity	7
2.1.3 Security	8
2.1.4 Initialization	9
2.1.5 Performance Evaluation	10
2.2 Overview of the CCIU System Structure	10
2.2.1 System Configuration	10
2.2.2 Distributed Operating System	12
2.2.3 Communication Network	12
2.3 Processor Element Operating System	15
2.3.1 Kernel	17
2.3.2 Status Monitor and Behavioral Template	17
2.3.3 Task Assignment Manager	18
2.3.4 Task Manager	18
2.3.5 Resource Manager	19
2.3.6 Contingency Handler	19
2.4 User Interface	19
2.5 Database Management	20

CONTENTS (CONT'D)

	<u>Page</u>
<u>Figures</u>	
1. CCIU Structure	11
2. Distributed Operating System	13
3. Communication Network	14
4. Processor Element Operating System (PEOS)	16

CONTENTS

Volume II	Page
<u>Functional Description</u>	
1.0 System Architecture Overview	22
1.1 Background	22
1.2 Purpose	22
1.3 Architecture Objectives	22
1.4 System Logical Structure	23
1.5 Function Location	24
1.6 Initialization	25
1.7 System Integrity	25
1.8 Failure Recovery	26
1.8.1 Communication Failure	27
1.8.2 PE Failure	28
1.9 Performance Monitoring	29
1.10 Definitions	30
1.10.1 Procedures and Program Modules	30
1.10.2 Processes	30
1.10.3 Task Force	31
2.0 Data Communications Network	32
2.1 Introduction	32
2.2 General Network Architecture	33
2.2.1 Configuration	34
2.2.2 Backup Scheme	34
2.3 Subsystem Architecture	35
2.3.1 Command Post Network (CPN)	35
2.3.2 Backbone Network	35
2.4 Protocols	36
2.4.1 Data Transmission Protocols	36
2.4.2 Network Control Protocols	37
2.4.2.1 Configuration Protocols	37
2.4.2.1.1 Acceptance Protocol	37
2.4.2.1.2 Exit Protocol	37
2.4.2.2 Status Monitoring Protocols	37
2.4.2.3 Contingency Handling Protocols	38
2.5 CCIV Architecture vs. Data Processing Models	38

CONTENTS (CONT'D)

	<u>Page</u>
3.0 Architecture of the Processing Element Operating System	39
3.1 Introduction	39
3.2 PEOS Kernel	41
3.2.1 Processes	41
3.2.2 Memory Blocks	43
3.2.3 Devices	43
3.2.4 Capabilities	43
3.2.5 Messages	44
3.2.6 Additional Kernel Functions	46
3.3 Task Management	46
3.3.1 TAM Overview	47
3.3.2 TAM Establishment	47
3.3.3 TAM Functions	48
3.3.4 Task Manager	49
3.3.5 TAM Protocols for Communication	49
3.3.5.1 Contract Net Protocol	49
3.3.5.2 Directed Protocol	50
3.4 System Status Monitor	51
3.4.1 Behavioral Template	51
3.4.2 Configuration Tables	51
3.5 Resource Manager	51
3.6 Contingency Handler	52
3.6.1 PE Failure	52
3.6.2 Additional Backup	52
3.6.3 Backup PE Selection	53
4.0 The CCIU Database Management System	54
4.1 Introduction	54
4.2 The Semantic User Interface	55
4.3 Field User Views and Physical Views	57
4.4 Data Control	60
4.5 A Note on Tailoring	61
5.0 Software Security	62

CONTENTS (CONT'D)

	<u>Page</u>
6.0 Human Factors and User/Developer Interface	63
6.1 Access Classes	63
6.1.1 Users	63
6.1.2 Configuration Controllers	63
6.1.3 Developers	63
6.2 User Interface	64
6.3 Software Development Support	65
Glossary	A-1
References	B-1
<u>Figure</u>	
1. PEOS Overview	40

Volume I
Executive Summary

SYSTEM ARCHITECTURAL CONCEPTS:
ARMY BATTLEFIELD COMMAND AND CONTROL INFORMATION UTILITY

1.0 Introduction

The Communications Electronics Command (CECOM) Center for System Engineering and Integration (CENSEI) and the Jet Propulsion Laboratory (JPL) have embarked on a program to develop a distributed tactical command and control system for the Army. The following material describes the CENSEI Development Program for Distributed Processing in general terms and the role of the JPL developed Command and Control Information Utility (CCIU) in the CENSEI program.

JPL is currently involved in developing the initial phases of the CCIU portion of the CENSEI program. This document, the second in a series, presents early concepts and preliminary analysis of the CCIU system architecture. Future revisions of this document will more closely reflect the actual testbed architecture and will emphasize more pragmatic than esoteric concepts. It is intended that this series of documents will converge on a distributed architecture satisfying the guidelines and functional needs of the CCIU. The development of this architecture will require a simplification, condensation and coordination of the material as presented in Volume II of this document. In addition, concepts reflecting the knowledge gained from currently developed demonstrations will be integrated into the architectural structure. This latter information is important in the progression toward a realizable CCIU architecture capable of operation in a realistic military environment.

1.1 The CENSEI Development Program for a Distributed Tactical Command and Control Information Utility

1.1.1 Operational Need

The U. S. Army Combined Arms Center (CACDA) has identified a set of conceptual requirements that indicate the need for distributed processing among the Army's battlefield automated systems. The basic requirement of the CCIU is to provide a degree of survivability and continuity of operations essential for execution of the air/land battle.

The threat of the capability to detect and determine the nature of command posts implies that Command Posts (CPs) must be geographically or physically dispersed. The purpose of this type of

positioning is to deny the enemy the ability to totally destroy essential command and control functions.

The Command and Control of Subordinate Systems (CCS2) concept, as articulated at Battlefield Automation Appraisal Five (BAA V), and approved as the Army objective command and control system, requires the ability to disperse and replicate certain databases among key operational facilities at each echelon of the field Army. Further certain functions must be able to be performed at facilities that are designated as alternate or backup in order to enhance survivability of the entire command and control network. The specific functions will be determined by doctrine and organization.

In the role as Army Command and Control System (ACCS) System Engineer, CENSEI has further characterized this set of conceptual requirements with four generalized distributed processing models (DPMs) described in the following paragraphs. Together with the CACDA conceptual requirements, the models provide resources for the distributed processing solution to Army Command, Control and Communication (Army C3). These combined resources form the basis of the CENSEI exploratory development program in distributed processing; the Command and Control Information Utility (CCIU) is an integral part of this program.

1.2

The Distributed Processing Models

The CENSEI program for development of distributed processing includes both concept development and a vigorous program of applications demonstrations. For the purpose of distributed processing concept development, the previously mentioned Distributed Processing Models provide both a distributed processing architectural basis and representative applications for demonstration.

There are four DPMs: Cellular Command Post, CCS2 Network, OFFAC-CONOPS (Operational Facility--Continuity-of-Operations), and CCIU DPM. They are briefly described in the following paragraphs. For a detailed description, see [DIED 81].

1.2.1

The Cellular Command Post DPM

The Cellular Command Post (CCP) is a collection of Processor Elements that perform the information processing functions of a Command Post. Each Processor Element is a processing cell capable of executing software-driven functions and storing data. The Processor Elements or cells are interconnected by a data distribution system capable of establishing data transmission links between all cells. For physical survivability the cells are geographically separated. There are three forms of Cellular Command Post DPM, representing increasing levels of distributiveness, involving the ability of the system to switch or migrate functional capabilities (both processes and data) among cells.

1.2.2

The CCS²DPM

The CCS² DPM implements the Training and Doctrine Command (TRADOC) concept for partitioning the total Command-Control system into five functional segments (Maneuver Control, Fire Support, Air Defense, INTEL/EW, and Combat Service Support) and employs a designated Control Element in each functional segment. The Control Element functionally coordinates the technical activities of Subordinate Systems included in the functional segment. The Control Element also provides command/staff type information to other Control Elements or accepts such information from them. Processing cells exist at each Control Element and at the subordinate systems of each Control Element. These cells are designed to execute programs and store data. They are interconnected by a data distribution system that can establish links between designated cells. The principal application of distributed processing is to provide CCS² system-level survivability in situations where a Control Element is lost or degraded. There are four forms of CCS² DPMs representing various contingency backup configurations. In these models the technical burden is on the reconfiguration capability of the data distribution system.

1.2.3

The OPFAC-CONOPS DPM

The OPFAC-CONOPS requirement is to provide continuity of operations during movement of a multi-processor, multi-shelter major operational facility. Each shelter that contains a processor may be considered a cell. The system design problem is to develop a distributed processing approach that lets OPFAC perform its functions adequately while some of its cells are moving (displacing). There are three forms of OPFAC-CONOPS DPMs representing distribution levels of the processing system control function.

1.2.4

The CCIU DPM

The fourth DPM represents a system-wide application of distributed processing techniques to Army tactical Command-Control requirements. Architecturally, the CCIU is a collection of computer hardware, software, peripherals, and communications. It is controlled by units in the force structure and bound together in a loosely-coupled distributed resource-sharing network. The workload is shared among the resources and is generally transparent to battlefield users. The purpose of the CCIU is to provide survivable, secure, continuously operable, adaptable, high-performance capabilities for command-control information processing.

1.3

Program Details

The CCIU work program includes study efforts addressing technical alternatives and system design/implementation details, laboratory simulations, and demonstrations to evaluate specific applications

of distributed processing approaches. All the DPMs specified above will be addressed during the laboratory and field demonstrations. For purposes of distributed processing concept development, the CCIU architecture will be the basic data processing architecture; the Cellular Command Post, the CCS2 Network, and the OPFAC-CONOPS model are representative nodal applications of the overall CCIU architecture.

1.3.1 Concept Development

The CENSEI Concept Development concentrates on:

- o The CCIU development at JPL
- o Analytical modeling of CCS2
- o Data compression techniques for Army C2 Information Transfer
- o Architectural concepts for Cellular Command Posts
- o Presentation schemes for CCS2 Commander/Staff information
- o Definition of minimum set of data elements
- o Establishment of data link protocols for distributed processing architecture

These major areas are complementary; together they address numerous technical issues crucial to the development of a distributed Army command and control system.

1.3.2 Applications Demonstrations

At four points during the program the emerging CCIU distributed processing architecture will be evaluated using the four DPMs. The first demonstration, given January 1982, demonstrated the rudiments of the Cellular Command Post in a configuration where two Processor Elements each contained all the functions and either one served as a backup for the other. The complete CCP DPM and the OPFAC-CONOPS DPM will be demonstrated next. That demonstration will represent a major step in the direction of a true distributed CCIU; it will incorporate most of the control structures required for all of the DPMs. The third demonstration will be of the CCS2 model, and the fourth demonstration will be of the CCIU model. Extensive hardware may be required for such a demonstration and it is possible that the test bed may emulate the CCIU.

1.4 The Role of CCIU In the CENSEI Program

The CCIU provides the basic framework for the CENSEI distributed processing program. Within the CCIU there are three task areas:

the System Architectural Concepts Document (SAC-Doc), Demonstration Development and Evaluation, and Testbed Development.

1.4.1

The SAC-Doc

The SAC-Doc is the repository for current thinking on the CCIU system architecture. The document contains ideas directed toward exploring DPM design requirements and concentrates specifically on the system design, and in addition, considers the relationship between the design and the laboratory demonstrations.

The SAC-Doc is intended to go beyond design requirements and laboratory demonstrations because it is not constrained by considerations of currently deployed capability or available technology. It proposes potential problems and potential solutions, although not every problem will have a solution proposed in the document. The SAC-Doc will continue to be updated throughout the year and published annually.

The SAC-Doc is also a source of design concepts for the various CCIU Laboratory Demonstrations. For example, the three-level architecture of CCIU described in the June 30, 1981 issue of SAC-Doc is reflected in the design of CCPl, the first laboratory demonstration. CCIU operating system kernel concepts, currently under development, appear in this document and will be incorporated into DEMO2.

The SAC-Doc also contains some concepts that may never be implemented, perhaps because the technology may not be available in a practical time frame, (for example, a relational associative processor) or perhaps because the concept may be infeasible to implement in the battlefield environment. Fully decentralized control, with its attendant high communication bandwidth requirements, might be an example of the latter. There also may be concepts in the SAC-Doc that have no direct operational requirement expressed or implied by the DPMs; e.g., the use of Artificial Intelligence techniques for using data of varying degrees of credibility.

In summary, the SAC-Doc is a current source of specific and exploratory material on system design concepts and architecture. It is a major resource for the specific purpose of building and demonstrating the progressive capabilities/applications of the CCIU.

1.5

Demonstration and Testbed Development

Implementation of laboratory demonstrations is the responsibility of the Demonstration Development task. These demonstrations will be implemented on a testbed system. Testbed hardware is currently off-the-shelf. The testbed system design will be compatible with Military hardware, specifically the TCS at the present time, and the Military Computer Family (MCF) components as they become

available. When they become more widely available, Ada and its software environment may be similarly incorporated.

The laboratory demonstrations will be of the various forms of the DPMs. However, the hardware/software configuration will be that of the testbed, not a fielded or soon to be fielded system. In the future, the necessary tie between laboratory demonstrations and fielded or soon to be fielded systems will be made by Demonstration Evaluation Reports. These reports will evaluate the potential usefulness of various features of the demonstrations on the basis of their applicability to fielded or soon to be fielded systems. CENSEI will determine what further steps should be taken.

2.0 System Architecture

This section summarizes the CCIU system architecture. The details of this material are in Volume 2 of this document.

2.1 Objectives

Four objectives of the CCIU design are (1) eventual use of current military hardware, (2) attainment of a survivable CCIU, (3) security, and (4) provision for elements of the CCIU to enter and leave the network. These objectives result from Ref. [DIED]. An additional objective resulting from good design practice is provision for performance monitoring. These objectives are discussed in this section.

2.1.1 CCIU Hardware

The CCIU will be compatible with existing military hardware. Because of limited availability, this hardware will not be used for the laboratory demonstrations that are a part of the JPL CCIU development program. Commercial hardware will be used for these demonstrations but the design will not include special hardware features that do not exist on military computers. The current military hardware for purposes of the CCIU design is the Tactical Computer System (TCS). Although much of the design can be transferred to the Tactical Computer Terminal TCT, the latter equipment does not support the hardware protection features used by the present CCIU design and considered essential to provide reliability and security.

The performance of a fielded CCIU could be improved if special hardware were used. The SAC-Doc will indicate such situations so that GENSEI can consider and evaluate the use of special hardware. The only example in the current design is considered in Section 2.1.3.

2.1.2 Survivability and System Integrity

The CCIU must continue to provide services to its users after destruction of some of its communication links and loss of some of its processor elements. Survivability is the term used to identify this characteristic which distinguishes the CCIU from most computer networks. Survivability is attained by using redundant communication links and by replicating the user functions at several of the processor elements, termed backup elements. A more complicated operating system requiring new design techniques is required to manage the backup facility. Survivability is also affected by internal malfunctions in individual processor elements; these malfunctions may lead to total loss of an element or to its continued operation in a degraded mode. Malfunctions may arise from hardware failures or from incorrect software.

The development of fault-tolerant computing techniques is currently a very active field of research. Fault-tolerant computer hardware and fault-tolerant software can minimize the effect of failures.

Fault-tolerant hardware provides additional redundant circuit elements in the equipment and the hardware design masks errors caused by circuit failures. The computer will operate correctly until multiple failures have exhausted the capacity of the fault-masking capability to make corrections. Examples of techniques for fault tolerance are triple modular redundancy, and error-correcting codes. Because of the design restriction to existing military hardware (see Section 2.1.1) the development of fault-tolerant hardware is not a part of the CCIU program. Current developments in this field are in the direction of complete processing units that correct errors; self-correcting memories are already available. A fault-tolerant processor could be used in the CCIU if a technically acceptable one were available.

Fault-tolerant software techniques are not as far advanced as hardware techniques. Often fault-tolerant software techniques rely on multiple computation of a partial result, sequentially in time, by subprograms using different algorithms. If comparison of the results indicates an error, the computation is automatically restarted at a previous point where the comparison indicated no error. The CCIU program is not developing methods for writing fault-tolerant software, but results in the field are being studied and will be incorporated as they become available. One interesting development is automatic program verification where the algorithm for a computation is mathematically proved correct. Such techniques are very promising; however, no reasonably sized program has yet been proved correct from the statement of the algorithm down through the production of the computer instruction code.

Hardware or software failure can cause two very different phenomena. First, wrong answers can be produced; this type of failure is quite difficult to detect. A second type of failure can lead to the program producing a continuous stream of obvious nonsense, not producing any results at all, or never terminating. In a multiprogramming system such as the CCIU where many programs reside simultaneously in different parts of memory, a bad program can destroy another program. Well-known techniques, described in the next section, exist to handle these second types of failures.

2.1.3

Security

CCIU security has two interrelated aspects. The first is security in the usual military sense of protecting the information in the CCIU from unauthorized access. The second is protection of the system from computer programs that are attempting an operation outside their proper limits. Such programs may be faulty or they may be deliberately trying to compromise the system.

Encryption can be used to protect information. Techniques for encryption are outside the scope of this program and will not be considered, but they can be superimposed as desired on the communication channels between processor elements and on the data stored in the elements.

Protecting the system from malfunctioning programs is accomplished by the hardware and software mechanisms briefly described below:

- (1) The hardware operates in two modes. One of these modes is only used by the operating system kernel (described in Section 2.3.1) and contains privileged machine instructions that control access to sensitive hardware mechanisms. This special hardware feature is included in the TCS and many commercial processors. Examples of such instructions are access to the memory storage of one program by another, and actual read/write operations on a mass storage device.
- (2) A capabilities mechanism is provided in the software to allow cooperating user functions to share parts of their data. Each shared data object will have a capability associated with it. A capability is a permit to use that object; the creator of the object can transmit the capability to any other user program that should have permission to access the object. It must be impossible to counterfeit capabilities. Software mechanisms enforced by the hardware privileged mode described in (1) above can accomplish this result.

The capabilities mechanism can also be implemented in hardware. Some research computer hardware has been constructed in this way and a commercial implementation will be available in the iAPX432 Microprocessor under development by the Intel Corporation. This type of hardware is superior to software techniques for the protection of capabilities; it should be considered by CENSEI for possible use in the CCIU but will not be used in the present design because of restriction to current military hardware.

The identification and authentication of individual users is related to data protection mechanisms. Passwords which can be encrypted as desired will be used by the CCIU for this purpose.

2.1.4

Initialization

The CCIU design provides well-defined procedures for a processor element to come on-line and join the network; the complementary procedure for leaving the network also exists. Both of these procedures are required for the OFFAC/CONOPS DPM. Security issues are involved in initialization to prevent unauthorized computing systems from joining the CCIU as processor elements. Survivability

issues are involved because leaving the network may be involuntary and due to communication link destruction.

2.1.5 Performance Evaluation

The CCIU will include software to provide statistics on its performance. Such data is essential to adjust system parameters for optimum performance. Gathering performance data incurs overhead in both computing time and storage of the data. In a fielded CCIU most of this activity should be automatically eliminated when the system facilities have been reduced by loss of processor elements. Only the data collection necessary for continuing the reassignment of functions for backup should continue. The current design has not addressed the details of implementing evaluation software.

2.2 Overview of the CCIU System Structure

2.2.1 System Configuration

Figure 1 is a diagram of the physical structure of the CCIU. Computing facilities, called processor elements, are interconnected through a communication network. This diagram does not restrict the form of the communication network; The current network design is described in Section 2.2.3. A cellular command post cell will consist of one or more processor elements.

A typical processor element includes one or more physical processors, a memory storage device, and a mass storage device. Computation is performed by processes; process is defined as a computer program that accomplishes part or all of a user function. A system process is one where the CCIU is the user. Examples of system processes described later are the contingency handler and the resource manager. A processor element will also have one or more terminals for individual users to access the CCIU and possibly connections to data sensors. An example of a data sensor is radar; in this case the entire processor element might be devoted to processing radar data. As a part of the CCIU, the processor element may also perform backup functions for other elements. A distinguishing characteristic of the CCIU design is that different processor elements do not share common memory. The communications network is the only interconnection.

Processor elements may vary substantially in the resources they contain. For example, an element could contain a minimal processing capability and be primarily a terminal multiplexor through which individual users could gain access to the CCIU.

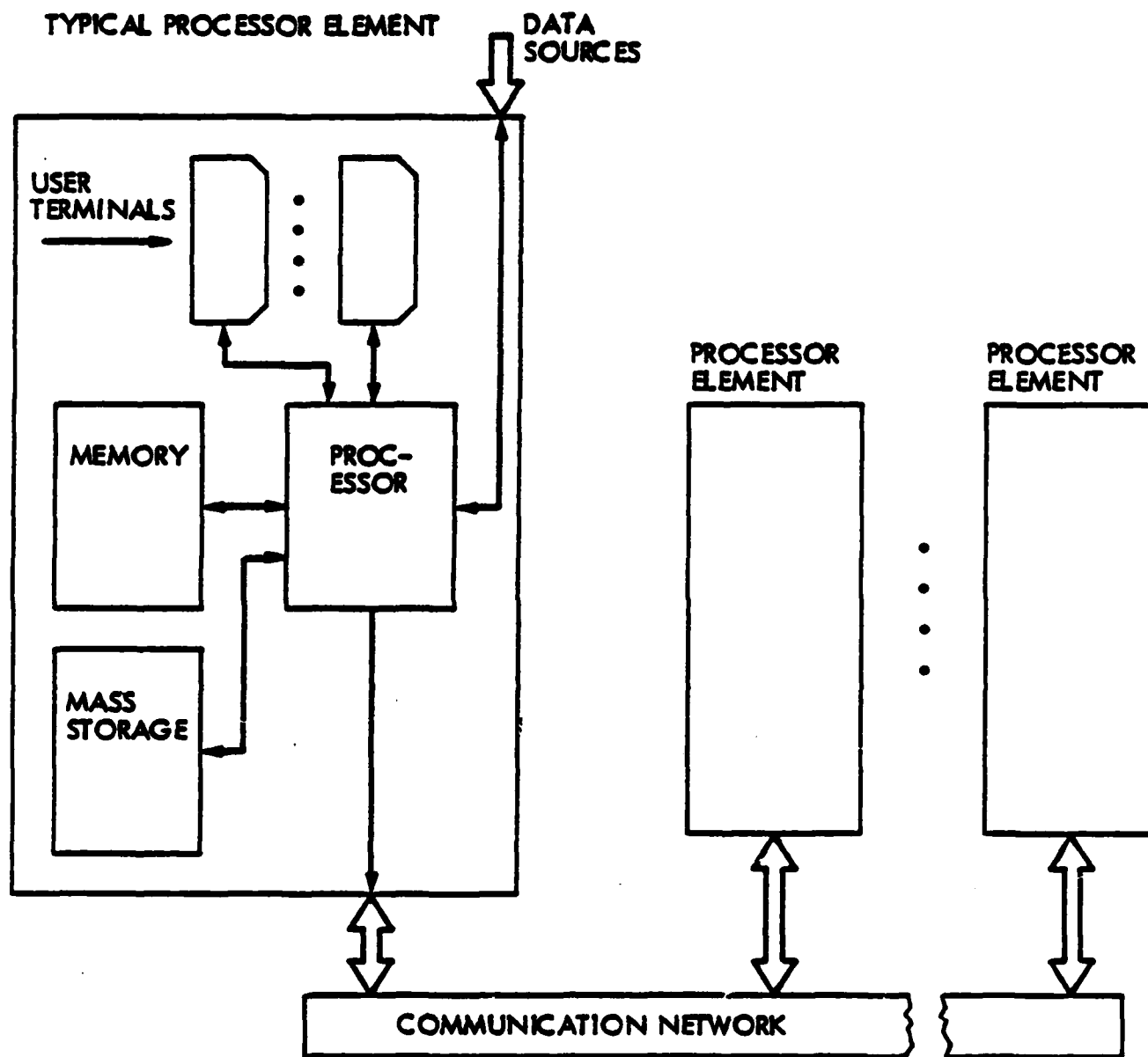


Figure 1. CCIU Structure.

2.2.2

Distributed Operating System

The CCIU supports a distributed operating system which manages the entire network (see Figure 2). The Processor Element Operating System (PEOS) manages the resources of a single processor element. Superimposed on the PEOS are Virtual Information Processors (VIP's) that extend over more than one processor element. These are the Command Slice Functional User (CSFU) VIP and the Control-VIP. The VIP's are implemented as processes on the processor elements.

The CSFU-VIP contains the user processes that perform the user functions of a CCS² command. The designated individual users of that functional command interact with the CSFU-VIP through the User Input Directive Language (UIDL) described in Section 2.4. A CSFU-VIP will normally reside in a command post that can encompass several processor elements.

The CSFU-VIP's use the facilities provided by a Control-VIP. The Control-VIP will include several processor elements within its purview. Its purposes are to know the processor element where specific user functions and system support functions are located and to supervise activation and execution of functions in the proper element. A user function may require the invocation of more than one process and these processes may reside on several processor elements.

The PEOS is described in Section 2.3.

2.2.3

Communication Network

The communication network diagrammed in Figure 3 has been designed to directly support the CCS² DPM. This DPM calls for a more complicated communication network than either the CCP or the OPFAC-CONOPS case. The latter two DPM's are supported by the communication interface as special cases.

Four layers exist in the configuration:

- (1) Command Post Network (CPN) - This local network connects all the subordinate systems of a given command post. In the CCP DPM it is the entire communications network.
- (2) Unit Network - This network links together all the CPN's of a single Army unit (e.g., battalion, division, corps).
- (3) Echelon Network - This network connects the unit networks at the same echelon level (e.g., divisions of the same corps).
- (4) CCIU Network - this network links the echelon networks.

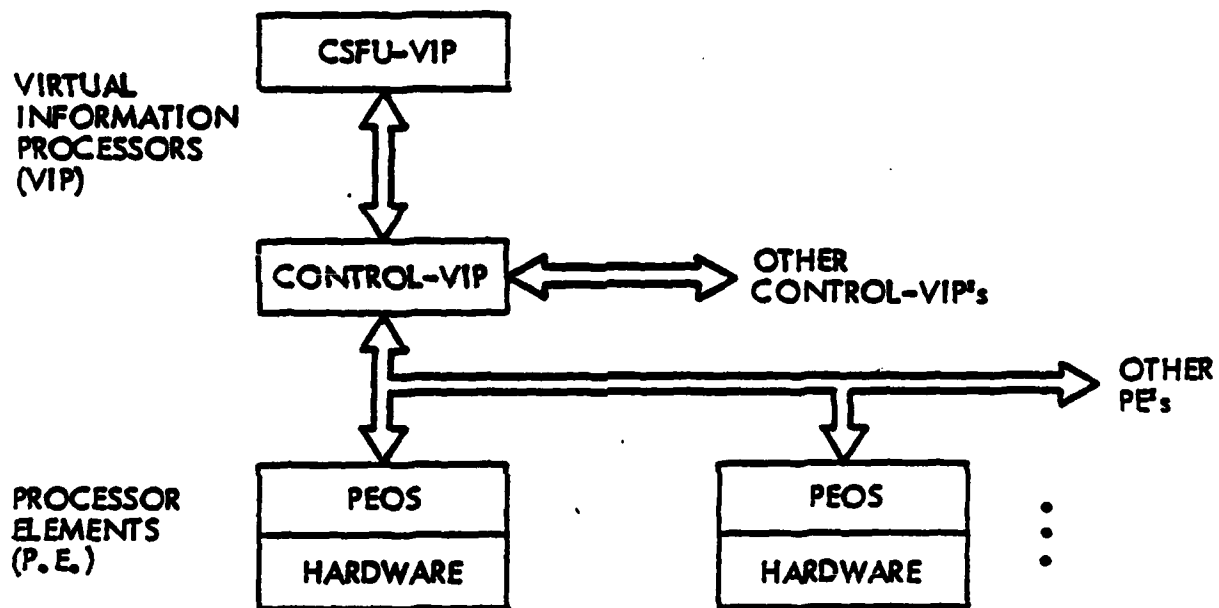
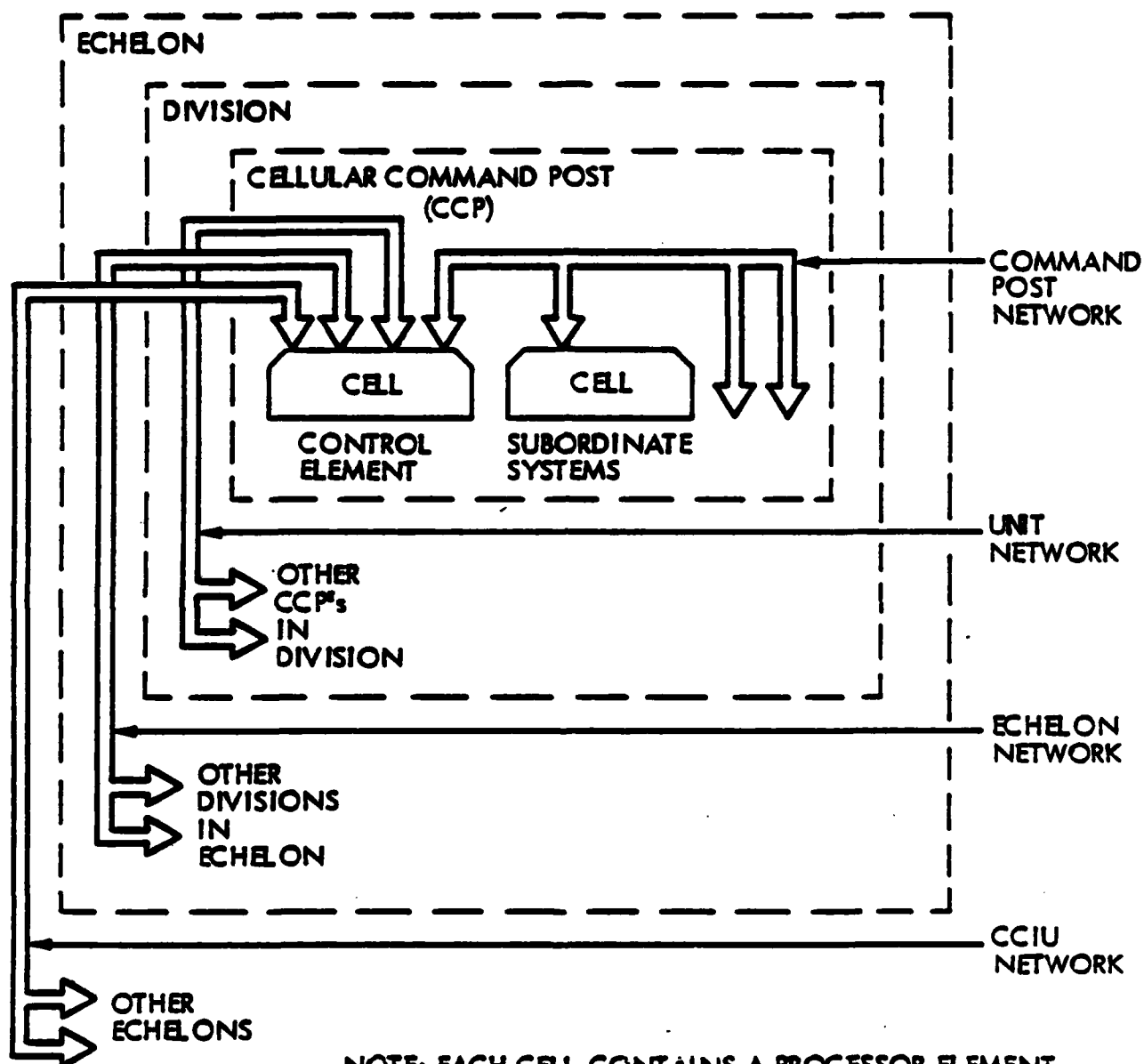


Figure 2. Distributed Operating System.



NOTE: EACH CELL CONTAINS A PROCESSOR ELEMENT

Figure 3. Communication Network.

The design is intended to facilitate local traffic and minimize long-haul traffic. At each level, the processor elements can either operate in a stand-alone mode or serve as gateways to interface with another level. The layered structure conforms with the Army command structure so that each subsystem is under the control of one commander. The layers are uniformly structured.

The layered approach facilitates fast reconfiguration by confining the effects to one layer and minimizing propagation to higher or lower layers. Uniform design in the individual layers makes it easier to reassign functions and reorganize backups. CCIU configuration tables are also kept to a manageable size by confining their scope to one layer.

The price paid for this approach is increased overhead in network operations requiring interlayer traffic. There will be an overall advantage if the distribution of user functions can be made so the majority of the traffic is local.

An important part of the network design is the provision of standard message types with appropriate protocols for common message functions.

2.3

Processor Element Operating System

Figure 4 shows the operating system for an individual processor element. It contains three distinct entities: the kernel, system and user processes, and system utility subroutines. The kernel is not a process; it is the basic part of the operating system which must be initially loaded into the processor element by an operator and will remain loaded at all times. It is described in Section 2.3.1. The system utility subroutines service all the processes and are centralized in one place to save storage space. An example of such a utility is a subroutine that would request the opening of a mass storage file so the requesting process could read it.

All CCIU user and system functions are implemented by means of processes. User processes perform functions for the user and will not be considered in this operating system description. System processes perform functions for the CCIU; a number of them are described in the remainder of Section 2.3. System processes will support distinct services and will call on each other as required to accomplish their work. Processes often need to cooperate with one another and thus must share data and interchange messages. These actions are controlled through the capabilities mechanism implemented in the kernel.

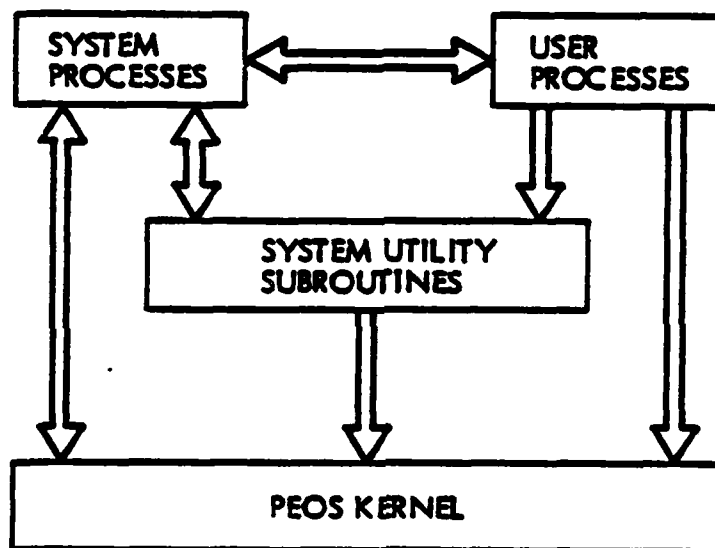


Figure 4. Processor Element Operating System (PEOS).

2.3.1 Kernel

The operating system kernel executes in the hardware privileged mode. It performs the following functions:

- (1) context switching (shifting the processor from one process to another)
- (2) servicing the interrupt-driven Input/Output (I/O) device handlers
- (3) servicing interrupt-driven parts of the communication software
- (4) providing facilities for the capability-based software protection
- (5) assigning physical resources such as memory and mass storage

The kernel is invoked by interrupts and by a set of operating system calls that can be used by processes. Some calls require the calling process to have a capability to make them. These calls could compromise the integrity of the PEOS if used improperly.

2.3.2 Status Monitor and Behavioral Template

The status monitor is a system process that maintains the system configuration tables. It is a separate process in order to achieve modularity of the operating system and is used by all processes needing configuration information. The status monitor is part of a Control-VIP; it provides the VIP with information about all the VIP's processor elements.

Individual CCIU processor elements will have different configurations and resources. The battlefield environment will lead to fragmentation, reconstitution, and reconfiguration of the CCIU. The configuration tables will be changing dynamically. There is an initial configuration table for a Control-VIP that is called the behavioral template and is part of the system data maintained by the status monitor. It encompasses:

- (1) hardware resources and performance characteristics
- (2) communication topology for the processor elements comprising the Control-VIP
- (3) processes available at each processor element
- (4) data and its distribution
- (5) priorities of functions

- (6) lists of functions backed-up at each processor element and backup locations for functions located at the processor element and backed up elsewhere

The Control-VIP will attempt to maintain its configuration as close to the behavioral template as possible.

2.3.3 Task Assignment Manager

The Task Assignment Manager (TAM) locates a process that has been requested by another process and directs the Task Manager (TM) to make this process active and initialize it. The TM is described in Section 2.3.4. The task assignment manager has cognizance of all the tasks in a given Control-VIP. An example of a requesting process is the User Interface Directive Language (UIDL) process, described in Section 2.4, acting as a result of a user command typed into a terminal.

Two ways are currently designated for the TAM to locate a process. The more common is for the TAM to consult the configuration tables (through the status monitor) to locate the process. The TAM then tells the TM (in the appropriate processor element) to run the process. The second technique can be used in case the TAM cannot determine where the process is located. In this situation it broadcasts a message to the processor elements asking if the process is available and if current resources at the location permit the process to be run. The TAM in this way can locate an appropriate processor element. A procedure called the Contract Net Protocol, described in Volume 2, defines the messages and procedures used in the second technique.

The TAM is a critical part of the CCIU and needs backup. If it were replicated completely in every processor element, excessive message traffic would be incurred to keep the TAM database of process locations consistent in all processor elements as the database changes. The current solution to this problem is to have the TAM code available in all processor elements but to have only one TAM active at a given time. A method exists for a new processor element joining the network to locate a TAM, and for a new TAM to be established if the processor element currently providing the TAM should be lost. The Contract Net Protocol can be used in emergencies to establish a new database for the TAM.

2.3.4 Task Manager

The TM is a part of the PEOS in each processor element that initiates the running of a process in its particular processor element. The TAM in a Control-VIP will activate the TM in the appropriate processor element among those within its control. The TM accomplishes its work by calls to the kernel. After the TM has started a process, that process will need to interact with other

processes. The TM also performs the function of setting up channels so messages can pass directly from one process to another. The TM creates a capability for such a message; a message is a shared object between two processes.

2.3.5 Resource Manager

The resource manager allocates the memory, disk space, and other resources of a processor element and contains the scheduler which allocates the processor to active processes time-sharing the processor. Resources are allocated according to their availability and the priority of the requestor. Most resource assignment is local to a processor element. There is a PEOS-resource manager to assign these local resources. A Control-VIP resource manager also exists. Its purpose is to take a global view of resources; for example it may find it expedient to migrate a process from one processor element to another. The resource manager uses other operating system facilities to perform its work. The physical allocation of the resource is accomplished by a resource manager call to the kernel.

2.3.6 Contingency Handler

The contingency handler is a system process that controls the distribution of backup functions among processor elements. It implements the algorithms that decide the next step in order to ensure the survivability of the CCIU after another processor element has been lost. The initial distribution of backups is determined by the behavioral template. A number of circumstances affect the redistribution of backups:

- (1) the number of currently existing backups for functions on the lost processor element
- (2) the present computing load on a proposed new backup processor element
- (3) the accessibility of code to implement the function at the proposed new processor element and the ease with which it can be transferred if it is not available

The contingency handler finds the new backup processor elements. Other system processes effect the decision depending upon the exact circumstances. For example, the status monitor will always need to provide system information; the resource manager may need to transfer code.

2.4 User Interface

The individual user will interface with the CCIU through a User Input Directive Language (UIDL). This language serves two purposes: (1) it isolates the user from direct manipulation of the

CCIU processes, thereby preventing modification of these processes and increasing the system integrity, and (2) it provides an interface with user commands in a vocabulary familiar to the user. A distinct UIDL can exist for each CSFU group in the CCIU. Provision will be made for the user to assemble a sequence of commands and store and recall them under a name selected by the user.

The UIDL will have extensive optional self-documenting features to support minimally trained users as well as a convenient shorthand for supporting expert users.

Field commanders need some degree of control over the CCIU configuration. It must be possible to add or remove special hardware and software required by the current tactical situation. A special group of users called configuration controllers is recognized by the system for this purpose. These users, authenticated by special passwords, will be able to adjust system parameters within prescribed limits and make alterations to the behavioral template.

No provisions are made in the CCIU for the field development of software at the process level. The development of groups of commands for complex procedures as described above is not an exception to this rule. Such procedures invoke a series of processes in sequential order but do not create new processes.

2.5

Database Management

Implementation of a distributed Database Management System (DMS) under the survivability requirements of the CCIU requires substantial work. A design for a DMS that will meet the CCIU DPM requirements has been developed. The initially implemented DMS for the second laboratory demonstration will be reduced in scope from this design; it will implement the survivability rather than distribution of data across processor elements. It will be extended to a distributed DMS for a later demonstration.

CCIU operators do not need to be database specialists to operate the DMS. The DMS provides a set of structures (views) to access the data by name. The invocation of a given view may lead to complex operations on the physical data to obtain the desired data. The user is expected to be primarily interested in displaying various forms of summarized information. Two types of transactions on the database are defined: manipulation and perusal. Perusal transactions will display the data in various ways. Only the manipulation transaction can modify the database. The owner of the data can restrict transactions and views of his data as he desires by granting capabilities (permissions) to users. Users will be able to combine the views they have permission to access into derived views which may further summarize the data according to their own needs. The physical data organization will not be visible

to the user but will conform to the relational model, i.e., in tabular form.

Volume II
Functional Description

1.0 System Architecture Overview

1.1 Background

The CCIU is a general purpose distributed information processing system designed to satisfy the Army tactical command and control requirements. Architecturally, the CCIU consists of a collection of resources such as processors, storage modules, software modules, I/O devices and sensors. These resources are owned by various army units but connected together in a loosely coupled network. The network provides for resource sharing and back up schemes to ensure survivability and easy reconfiguration as well as load balancing.

The CCIU will provide its users with a data processing facility that performs a number of well-defined functions. These functions will be performed by a number of processing elements interconnected into a computer network. The user need not note this method of implementation; it will appear as though he has a dedicated resource to perform his functions.

1.2 Purpose

The purpose of this document is to address the major technical issues involved in the development of the CCIU, suggest implementation approaches, and identify functions and needed resources.

This is a working document in which ideas are recorded. Hence, it should not be perceived as a final and complete design document. It will be updated periodically as new ideas develop. At this point no distinction has been made between ideas that can be implemented using existing technology and ideas that require further research.

1.3 Architecture Objectives

One of the primary objectives of the architecture is survivability of user functions in battlefield conditions. User functions must survive intact as far as possible. In the face of extreme loss of physical equipment, functions should degrade gracefully.

The physical configuration of the system is subject to arbitrary and dynamic change. Communications links may be cut, processing elements and other equipment may be destroyed or removed as a result of battlefield destruction, maneuvering or maintenance (repair). Similarly replacement of defective or destroyed equipment may later add to the network.

There are two extreme ways to achieve survivability. In one, each function and database are replicated at all processors and the system can perform all of its functions as long as one processor element is fully operational (assuming communications links intact). This has unacceptably high costs for the required resources. At the other extreme, a PE contains only its normal functions. If it fails, backup is possible only if similar functions exist at other PE's. Again this is unacceptable. The CCIU will attempt to strike an optimal solution somewhere between the two alternatives partially by replicating functions and partially by migrating functions from one machine to another as required for maximum survivability.

In determining the CCIU architecture the following constraint needs to be considered. The software to perform the functions will not be developed on the operational system and cannot be altered by the user. Strict configuration control will be maintained at all times with well-defined procedures for additions or alterations of the software.

The architecture will realize a system in which each user function can be backed up by replicating part, or all of its programs and data as necessary to secure survivability.

Another important objective of the architecture is security. This subject is considered in more detail in Section 5 and in various places throughout the document as specific design decisions implying effects on security are presented.

1.4

System Logical Structure

The CCIU system is logically structured in three basic layers. The top two layers, called Virtual Information Processors (VIP's), are software layers while the third layer is the physical layer consisting of the processor elements. These virtual processors are:

- (1) The Command Slice Functional User VIP (CSFU-VIP)
- (2) The Control VIP (Control-VIP).

The CSFU-VIP represents the system as it appears to users. It is the virtual machine that contains the programs that accomplish user functions and its program modules may, as the result of the backup scheme, lie in different physical machines. The CSFU-VIP implements the functional elements shown in the Cellular Command Post model, the CCS² model, etc. [DIED 81], but is not aware of the network or the physical structure. The CSFU-VIP interfaces with the Control-VIP. This virtual machine interfaces with the physical layer. It is aware of the network and also knows the distribution of the processes. Although it performs a distinct logical function in the network, it is transparent to the user.

Both the CSFU-VIP and the Control-VIP are part of the Control Element described in the CCS² model [DIED 81]. The CSFU-VIP consists of the application programs and the Control-VIP is realized by software modules of the processor element operating system (PEOS) residing in a processor element. The PEOS manages a network processor element and interfaces with PEOS's of PE's via the Control-VIP services.

A Control VIP normally resides in one PE and other PE's use its services (see Section 3.3). However, the capability exists in all the other PE's to execute this control VIP and methods exist for an orderly transfer of the control VIP function to another PE should it become necessary.

1.5

Function Location

The user functions of the CCIU are fixed (Volume II, Section 6), and cannot be modified by the user. However, their distribution in the system may vary dynamically as the system configuration changes. A dictionary, maintained as part of the configuration table in the control element, indicates the location of application software modules in processing PE's. This dictionary is updated and used by the Control-VIP to locate functions for the CSFU-VIP, and in general is transparent to the user.

This dictionary describes the distribution of functions in terms of the elements of the particular layer of the network in which the control element lies (see Section 2.0). Each of these elements may be a network of a lower layer containing its own control element with a dictionary describing further the distribution of its functions among its processing elements.

The dictionary initially arises from the behavioral template (see Section 1.6). Because of the possible dynamic reconfiguration of the physical network it is not possible to place this table in one PE and always expect it to be available. Therefore, a requirement exists for distributed knowledge within the system. Since it is not known a priori which processor may fail, maximum reliability could be achieved only by a complete, universal replication of the dictionary in every PE. The provision for such a complete replication requires a communication and processing overhead to keep all the copies updated. This situation is analogous to the previously mentioned requirements for function backup. The CCIU will attempt to obtain a cost-effective solution. Currently the design is based on the idea that the dictionary is stored and accessed in one processor only. Provisions are made for automatically transferring the dictionary to another processor so that it will be available if the original processor fails.

1.6

Initialization

A given CCIU network will be initialized with a given set of functions, processors and links -- not all of which may be initially available. This situation gives rise to the concept of a behavioral template. This is a description of the maximum configuration of the network. It includes a description of the placement of the user functions in the appropriate processor elements. It is the system reconfiguration goal to attempt realization of the behavioral template as closely as possible in terms of currently available hardware and software.

A method must exist for initializing the network and for a PE to either join or leave the existing network. There are security issues involved in allowing a PE to join the network; unauthorized processors must not be able to become part of the network.

1.7

System Integrity

System integrity includes characteristics that affect the availability of the system to the user. Two aspects of availability are of concern. The first is the availability of the system PE the user wants to use; the second is the availability of the system services he wants through any PE, especially if the PE he would normally use is unavailable [RENN 81].

The system is normally accessible through terminals connected to a PE. The particular PE may be a minimal one provided only to gain access to the network. In order for the system to be available with sufficiently high probability, a number of features must be included in the hardware and software:

- (1) The basic availability of a given PE must be maximized.
- (2) A backup capability must be provided so that if a PE should fail or become inaccessible due to communication failure the services are still provided to the user through some other PE.

The CCIU will be disconnected from all the users whose terminals are physically connected to an unavailable PE. The reestablishment of service is a logistics problem of gaining access to another terminal connected to an available PE. The problem from the CCIU design standpoint is to ensure that the service is available on some other PE and that a terminal, possibly connected to still a third PE can access it.

Backup can be achieved by redundant facilities, (1) in the form of additional PE's placed in the system only to provide backup or, (2) in the form of replicated data and the processes for accessing it deployed in existing PE's of the system. Replication of data requires procedures for ensuring the consistency of the various

copies should the data be needed without warning, e.g., due to failure in the primary processor.

Availability is a term borrowed from the field of fault tolerant computing. The hardware is designed to continue operation even though faults have occurred because fault masking is employed, or the hardware detects faults and institutes recovery procedures. The computer will continue to operate and produce correct results, possibly with degraded speed or "throughput" until hardware that has failed can be replaced. To accomplish correct operation, the computer's initial hardware must be augmented by additional redundant hardware. The hardware must be designed so it indicates failures have occurred. Maintenance personnel can replace the faulty hardware before the capability for correction has been exhausted.

The CCIU program is not specifically developing fault tolerant redundant processors. As previously mentioned, this specialized field is presently receiving great emphasis elsewhere. The CCIU program is concentrating on the special circumstances that apply to its implementation as a network. The requirement for the CCIU as a whole is to continue to operate in spite of the possible permanent loss of some of its processing facilities as well as the temporary or permanent loss of some of its communication links.

The development of fault tolerant software is another research field of current interest. In general, correct software is written by cycles of development and testing eventually followed by certification.

A user program should be certified before use on the CCIU. Nevertheless, it is probable that some will occasionally fail. Recovery will probably consist of first retrying the process. If it continues to fail a backup should be implemented. If the failure is due to an uncommon operation that caused a bug to slip through the certification procedure, the backup will probably fail, too. The CCIU will generate an error message but will not stop the use of the program.

1.8

Failure Recovery

System integrity will be considered in two major categories: (1) communication network aspects, and (2) PE aspects. These two categories are quite different; in both cases it is software in a PE that undertakes the appropriate action until hardware can be restored or replaced.

A contingency handler process in the PEOS provides a degree of failure recovery by reconfiguring the network, activating backup schemes, and redistributing functions.

Recovery from loss of a PE will consist of reassigning the computing load to maximize the performance of the highest priority tasks in the remaining facilities. Recovery from loss of a communication link will consist of finding another path between the affected PE's, possibly an indirect path. In some circumstances the network may become fragmented into two or more networks. Reestablishment of a single network when new links are available is a complicated problem. This fact results primarily because even in a network with totally decentralized control there is information which must be global to the network. The handling of this information, when the network becomes disconnected and reconnected, requires merging the split databases which may have become inconsistent.

The basic fault-masking techniques and reconfiguration procedures should still be applied to individual PE's to increase their availability. It is assumed the system will be designed so it can contain fault-tolerant processors and other hardware. In the following discussion, some aspects of fault tolerant recovery procedures will be considered. We reiterate that correction of faults can be carried only so far with a reasonable amount of redundant hardware. There will always be a limit beyond which the hardware will not work. Provision must be made for the orderly shutdown of a machine to avoid damage to databases and erroneous computation if fault recovery has been exhausted.

1.8.1

Communication Failure

Communication failures can take place when a path is, or is not in use. If the path is in use, active recovery techniques are necessary to ensure that processes currently communicating do not simply stop without indication because they are waiting for a message from the other process. The normal way of detecting such failures is through a timeout mechanism in the process using the channel. This is a polling technique that not only detects a communication failure but also any problem in the respondent that has caused it to cease performance. If a problem is detected, the OS may be notified and steps taken for recovery. The steps may include an attempt to set up an alternate channel to the remote PE, or an attempt to get the data required from a completely different source. The contingency handler, as well as the task assignment manager, will be involved in this work.

On the other hand the failure of a communication channel not in use is not apparent until a message needs to be sent. Then all the mechanisms for alternate routing or service will have to be invoked. Communication failure is, however, basically asymmetrical with respect to receiving/sending messages. In order to send a message, the OS is active; it can therefore determine something is wrong with the channel and send an error message to the process sending the message. Using polling techniques to determine channels are usable when they are not in use should be considered.

This increases the overhead but provides better system availability for the user.

1.8.2 PE Failure

PE failure may require a series of complex actions. Before considering it we will classify types of failures that may occur by the method of detection: (1) detection in the PE itself, or (2) detection by another PE. Some types of failure may be found in both categories depending on the specific mechanism.

A failure that is not detectable by the PE itself may result in two situations. First the network may not be able to communicate with the PE. This situation may result from a communications link failure. Network protocols invoking routing algorithms check alternative paths to the PE automatically. Thus, a communication failure means all possible communications links to the PE failed. From the network point of view this is the same as a PE failure. The backup PE's should attempt to implement the high-priority services provided by the missing PE.

A situation potentially more damaging to the network could occur if the failed PE attempts to flood the communication channels with spurious messages or garbage. In this case the channel connection should be shut off. If the failed PE is large enough to have a separate communication front-end processor, part of the monitoring procedures in the PE should be cross-checking so problems outside the front-end processor can be signaled.

If a PE can detect internally that a failure has occurred, it can decide to recover based on the redundant hardware available, or shut itself down after notifying the network.

Specific examples of PE failures that can be locally detected and do not incapacitate the PE completely, include failure of a user program (or a non-critical PEOS module), and failure of a peripheral. A peripheral can be a disk with important data that must be obtained from other sources. In some circumstances it may be desirable to transfer the entire process using this data to the machine with the backup data to lessen the communications overhead. For example, a request for summary information from a database may require many queries but result in little output.

The previous discussion considered fault detection from the network point of view. However each PE will have one or more users accessing its facilities. The user will presumably be able to determine when his PE is unavailable. Well-defined procedures for the user must be set up to handle problem detection. In many situations the user will be able to effect repairs and have the PE rejoin the network.

Performance Monitoring

Two aspects of performance monitoring are important to the CCIU. The first aspect is field monitoring carried out during actual field operation of a CCIU. The use of the data collected in this monitoring is considered in Section 6. The second is performance monitoring during the development of the CCIU. Implementation of one or more test systems to experimentally test and verify design concepts is necessary. Monitoring performance of these test systems is essential and will be more extensive than in the final field system. In a test system the monitoring facilities can provide physical means of simulating external environment effects. This use of the monitoring facility would not exist in a field system. Both uses are considered appropriate for this document and are discussed in this section.

Monitoring in the field and the test CCIU have two important functions. First, monitoring enables the CCIU maintainers to decide how well the CCIU performs its intended function. Secondly, the monitor can be used to tune the system by varying parameters and observing the system's performance. Examples are the adjustment of priorities, the distribution of common functions among PE's, etc. Additionally, deficiencies in the design or bottlenecks in the operation can be identified so the system can evolve into a better one. An example of this is the identification of significant communication delays. These delays might be mitigated by additional links between PE's; this constitutes a change in the hardware configuration. The delays may be prevented by reassignment of function or backup; this is an example of the adjusting mentioned earlier. A third possibility is that delays might indicate a defect in the system logic that needs correction.

Monitoring also has an important use in keeping an audit trail. System use by individual users can be recorded to the extent desired for later analysis. This use is important from the standpoint of security.

Monitoring does require additional hardware and software in the CCIU and requires some expenditure of resources and processing time. In a field environment, the use of these resources for monitoring may be inadmissible after some of the PE's have failed and the CCIU is operating in a degraded fashion; monitoring is a function whose extent must be controlled by the CCIU contingency handling mechanisms.

Monitoring is a relatively low-priority function in a field environment and should be decreased or eliminated entirely when the available facilities are decreased. This type of monitoring control is uncommon and needs study. In the testbed, monitoring is a relatively high-priority function; it will tell where problems occur when equipment and communication links become unavailable.

1.10 Definitions

Definitions of Procedures, Program Modules and Processes only are provided in this subsection to ensure that precise meanings for these terms are specified. All other definitions appear with the Glossary and Reference sections.

1.10.1 Procedures and Program Modules

A body of code which is already linked and ready for execution will be called a procedure. Note this is not a standard meaning for the name procedure. It is roughly equivalent to the one in [WULF 81].

Such a procedure will normally reside on mass storage and will be replicated as required for survivability. When a procedure is to be run, the Task Manager (TM) will create a process from the procedure and run it. In order to create the process the task manager may need assistance from the Task Assignment Manager (TAM) to locate a processor where the procedure may reside. The TAM and TM are described in Section 3.3.

1.10.2 Processes

A process is defined as an instance of a procedure executing on a machine.

All processing done in the CGIU will be accomplished by processes. This includes not only the user processes but the operating system processes with the sole exception of the kernel of the operating system. The kernel, which must be in core at all times is not realized as a process. In order to preserve transparency of the location of processes we will require the processes only communicate among themselves by messages. This method is required if the processes happen to be in different machines and will be used if they are in the same machine. This method specifically excludes shared memory as a communication technique; some situations are later noted where relaxation of this requirement should be considered.

The operating system creates a process by entering data into tables in the kernel named a "process control block". The number of concurrent processes depends on storage availability. The TM will create and run a process by calls to the kernel. A possible kernel implementation is presented in Section 3.2. A process may be in one of the several states noted.

- (1) Active. This is the running process. Only one process can be active at any given time on a single processor.
- (2) Waiting. A process loaded into memory and waiting to be run when the scheduler calls for it. To run it requires a context switch.

- (3) Blocked. A process is blocked when it is waiting for an I/O operation to be completed. A blocked process is swapped out to backup storage and the scheduler must request the kernel to load it.
- (4) Submitted. This is a process that has been created but never loaded and run.

1.10.3

Task Force

Several processes may cooperate to perform a single task. This group of processes is called a task force. This concept is particularly useful for a distributed task that consists of several processes on different machines. A task force may have a central "executive" which coordinates its component processes, stores common data, etc. but the individual processes will communicate among themselves and with the executive by messages exactly as all independent processes do. The concept of a task force is due to Jones at Carnegie-Mellon Univ. [JONE 79].

Cooperating processes require some synchronization mechanism to permit one process to invoke another as well as communicate with one that is already running.

2.0 Data Communications Network

2.1 Introduction

This section discusses network architectural concepts and identifies subsystems of the data communications network for the CCIU system. The network architecture addresses some of the technical issues presented in the CENSEI document [DIED 81] and attempts to satisfy the general requirements specified in the four distributed processing models (CCP, CCS², OFFAC-CONOPS, and CCIU).

The communications network provides the physical and virtual links needed for transferring control, data and voice message among the various subsystems and users of the CCIU. The network facilitates distribution of data and processing functions. It also makes a backup scheme possible.

Some of the basic architectural concepts presented here are:

1. Layered architecture that conforms with the Army command structure.
2. Uniformity of control structure at the various layers.
3. Subsystems designed to operate in a stand-alone mode.
4. Localization of control and data traffic.

Four basic layers have been identified. From top to bottom they are:

1. CCIU network
2. Echelon network
3. Unit network
4. Command Post Network (CPN)

Several network control protocols have been identified. They include:

1. Configuration protocols - Acceptance protocol, Exit protocol
2. Status Monitoring protocols
3. Contingency Handling protocols - Backup Activation/Deactivation protocols and reassignment protocols.

The first part of the section deals with architectural concepts; network protocols are discussed in the second part.

General Network Architecture

The CCIU network architecture is layered to enhance survivability and mobility. The basic subsystems of the CCIU are the subordinate systems [DIED 81] connected by the command post network. They constitute the bottom layer. The subsystems of the higher layers are configured in the following manner: subsystems of a given layer are linked together by a network to form a subsystem of the next higher layer. Each subsystem contains a control element that may or may not be distributed. The control element provides management of the subsystem network and an interface to a higher level network.

The design philosophy is to build subsystems to support local activities and to minimize long haul control and data traffic. Subsystems at each level are designed to operate either in a stand-alone mode or using the gateway capability, as part of the higher level network.

The layered structure is designed to conform with the Army command structure; each subsystem is under the control of one commander. Backup schemes are designed at each subsystem and between subsystems at the same layer.

The layered approach facilitates fast reconfiguration by confining reconfiguration effects to the reconfigured subsystem and minimizes their propagation to higher or lower layers. The uniform design of the various layers makes it easier to reassign functions and reorganize backup schemes. It keeps configuration tables at a manageable size and facilitates mobility of subsystems by making it easier to attach or detach a subsystem from a higher layer network. It alleviates data flow problems and shortens delays by providing communications paths within the subsystem for local data and control traffic.

The price paid for this is in terms of increased overhead for the interlayer traffic. However, since the majority of the traffic is local, the advantages outweigh the disadvantages.

The CCIU network is comprised of four configuration layers as mentioned above. The first and lowest level is the Subordinate System. It is a cell of the cellular command post and consists of one or more collocated processor elements (e.g. in a single trailer). The CPN is a local area network (maximum distance between elements 1-10 km.) linking together a set of subordinate systems and a control element. The control element is also a cell of the CCP formed by processor elements. It controls the CPN and provides the gateway capabilities to allow attachment to the backbone network. Each CPN may implement a different functional segment of the CCS² model ([DIED 81]).

The Unit Network links together the CPN's belonging to the same Army unit (battalion, division, corps). The Echelon Network links together Unit Networks at the same echelon level (e.g. divisions of the same corps). Finally the CCIU network links the echelon networks to a single network.

2.2.1

Configuration

The network is designed so that each subsystem maintains its own local configuration table. The configuration table records the updated status of each element in the subsystem, the distribution and status of the various functions within the subsystem, routing information, and backup schemes, among others. It is used for routing messages, resource allocation schemes, network initial configuration, reconfiguration and more.

Starting with the Subordinate Systems, the subsystem in the next higher level builds its local configuration tables based on the configuration tables of the lower level subsystems it links together.

The CCIU configuration tables maintain a dictionary with an entry for each resource of the Subsystem in which the updated status of the resource is kept (see Section 1.5). The table initially is the same as the Behavioral Template (see Section 3.4.1). For example, a configuration table of a Subordinate System has entries for each terminal, I/O device, CPU, auxiliary storage, sensors and other resources as well as a list of the available functions. A CPN configuration table records the status of each Subordinate System in the CPN and also a list of functions and their locations.

When a device is added to or taken from the Subordinate System, or whenever a device changes its status (e.g. performance is degraded or upgraded), the Subordinate System updates its configuration table to reflect the changes. This change is propagated to the control element at the CPN level only if it affects the functionality of the Subordinate System. Similarly a CPN updates its configuration table whenever a Subordinate System is added, lost or recovered or whenever its status has changed.

2.2.2

Backup Scheme

Various backup schemes could be devised for the CCIU [DIED 81]. They will not be discussed in detail in this section. However, it is assumed that the backup schemes will conform to the layered structure.

Elements within a subsystem back up each other as well as the control element functions. For example, at the CPN level, subordinate systems within a CPN may back up each other (in processing capability, storage capacity etc.) and also have the capability of managing the CPN. Similarly, at the Unit network level, CPN's may

back up each other (e.g. by migrating processes, redundant distribution of functions, etc.). At the Echelon level, Unit networks back up each other (e.g. one Unit network making a function available to a Unit network that lost a similar function). Finally, at the CCIU level one Echelon network may back up another Echelon network at the lower echelon level.

Special network control protocols to facilitate backup schemes such as Backup Activation/Deactivation protocols will be developed. They are discussed in Section 2.4.2.3.

2.3 Subsystem Architecture

The CCIU network contains two basic types of networks. The CPN is configured around a local area network, and as such, uses local network techniques such as bus (or ring) structure, carrier sense multiple access or token passing protocols [CLAR 78]. The higher level layers are configured as short, medium and long-haul backbone networks. The physical structure of the networks will be consistent with existing military communications. Bandwidths will therefore be limited to the TAC range.

2.3.1 Command Post Network (CPN)

The Command Post Network (CPN) is designed as a stand-alone system that can be attached/detached to/from a Unit Network in a way that is transparent to the users. It can function independently from the Unit Network but when attached to the Unit Network, it serves as an element of this network under control of its control element, while retaining autonomy over local functions.

The proximity of the elements of a command post makes it possible to configure the Command Post Network as a local area network. This local network serves several purposes. It serves as a concentrator for the Unit Network, and provides for fast local communications among the users in the command post. It links together several subordinate systems and various terminals, sensors and possibly mass storage devices shared by them, and a network control element that provides local network control and a gateway to the Unit Network.

The logical structure of CPN is a bus structure, (broadcasting network). This allows for easy reconfiguration and facilitates the backup scheme.

2.3.2 Backbone Network

The backbone network comprises the three higher layers of the CCIU network: the Unit Network linking together the CPN's of a single Army unit (e.g. battalion, division, corps), the Echelon network tying together the the Unit Networks of all the units in the same echelon (e.g. all the divisions of the same corps), and the CCIU

network that connects the echelon networks into a single network.

Networks at each layer should be able to function independently of the other networks at the same or higher levels. To facilitate the stand-alone capability and the interface to the higher layer, each network has a control element that may be either centralized or distributed.

To enhance survivability, the control elements at each level should be upward compatible so they can back up control elements at the same, lower, or higher levels.

Communication medium used by the backbone network is likely to be a combination of radio, satellite communications, and terrestrial networks (The TRI-TAC army network, public network, telephone lines). The selection of a particular medium depends on the mobility of the units, physical environment, and other considerations.

2.4

Protocols

Protocols are needed to create the virtual communications medium with certain desirable characteristics that are more useful than the "naked" physical links. They provide services such as message sequencing, priorities, error and flow control. They also establish standard data elements and communications conventions [MCQU 78, TANE 81].

The protocols of the CCIU serve two basic purposes. Data transmission protocols serve the CSFU-VIP and the Control-VIP by facilitating the transmission of data and control messages among the subsystems of the CCIU. The network control protocols serve the Control-VIP in resource allocation, status monitoring, configuration and reconfiguration of the network.

Protocols provide for standard information exchange messages between processes, and are a part of the communications architecture. The Control-VIP protocols discussed here are closely related to the system processes used by the PEOS and described in Section 3. References are given to the appropriate section in the subsequent descriptions.

2.4.1

Data Transmission Protocols

In general, it is advisable that the CCIU data transmission protocols will be compatible with the ISO-OSI reference model [ZIMM 80], which is becoming the standard for the civilian networks. This will enable CCIU to take advantage of existing public networks when possible.

It is quite conceivable that requirements on the CCIU will necessitate modifications from the ISO-OSI reference model in order to enhance security measures and reduce overhead.

2.4.2 Network Control Protocols

The network control protocols are subdivided into three categories: configuration protocols, status monitoring protocols, and contingency handling protocols.

2.4.2.1 Configuration Protocols

These protocols help in configuring the network. They facilitate an orderly attachment/detachment of CCIU subsystems (e.g. a CPN) to/from the network. Two protocols have been identified: the Acceptance Protocol used during an attachment and the Exit Protocol used when the subsystem is detached.

2.4.2.1.1 Acceptance Protocol

When a subsystem is attached to a higher level network (e.g. a CPN to the Unit Network), the Acceptance Protocol is carried out between the network's operating system and its counterpart in the subsystem being attached. Messages are exchanged to make the subsystem "known" to the network and vice versa. The content of the messages is used to update the appropriate configuration tables. They convey information about functional distribution, processing and storage capabilities and more. This protocol also allows for security checks and authentication procedures.

2.4.2.1.2 Exit Protocol

When a subsystem is detached from the higher level network, the Exit Protocol is utilized. This is done in order to update the configuration tables and activate backup procedures to replace the lost resources (e.g. migrate processes, reassign functions, etc.).

Exit protocol is also used when a subsystem sustains such a considerable loss that it cannot totally recover by itself, or when it is going down but has sufficient time to go through the protocol.

2.4.2.2 Status Monitoring Protocols

Status protocols monitor the status of the various subsystems and communication links. This monitoring activity is necessary so the network keeps track of loss or recovery of functions, resources, and communication links. Examples of algorithms that can be used for this purpose are the echo algorithms [CHAN 79]. Status Monitoring is considered in Section 3.4.

Two basic approaches to monitoring are mentioned briefly. One approach is status reporting where subsystems send their status update report periodically to the control element. The other approach is status inquiry, where status update inquiries are issued periodically to each of the subsystems by the control element. Yet another approach is the use of the combination status reporting and inquiry.

Whenever a major loss is detected as a result of status monitoring a contingency handler routine is invoked.

2.4.2.3 Contingency Handling Protocols

These protocols are used whenever reconfiguration is needed as a result of orderly detachment of units or as result of a temporary or permanent loss of resources or communication links. To reconfigure the system the contingency handler (see Section 3.6) needs to activate backup resources. For this purpose the Backup Activating Protocol is used between the control element and the backup subsystems. If the loss is temporary (e.g. for repair), when the resource is up again the backup resource needs to be deactivated. This calls for another protocol, the Backup Deactivating Protocol.

The CCIU, at all levels, may need to reassign functions as well as reorganize backup schemes resulting from heavy losses. This task conceivably requires quite complex communications among the various operating systems. A set of protocols will be needed for this purpose. They are referred to collectively as the Reassignment Protocols. The contract net protocols are examples of such protocols [SMIT 79].

2.5 CCIU Architecture vs. Data Processing Models

The CCIU network architecture discussed here supports the Data Processing Models (DPM's) described in the CENSEI Exploratory Development Program [DIED 81].

The Cellular Command Post (CCP) can be realized by a single CPN.

Each of the functional segments of the CCS² model can be realized by a specialized CPN. The various layers of the backbone network correspond to the echelon and interechelon communications.

The OPFAC-CONOPS model is supported by the stand-alone and gateway capabilities of the subsystems and the conformity of the architecture to the command structure.

The layered structure supports the general objectives of the CCIU.

3.0

Architecture Of The Processing Element Operating System

3.1

Introduction

In Section (1) an overall view of the network operating system was given. The network was described as a hierarchical structure of virtual processors: the Command Slice Functional User (CSFU-VIP) and the Control-VIP built on Processor Elements (PE's). This section describes in detail the operating system that exists in each of these PEs.

Figure 1 shows an overview of the PEOS. It is comprised of three layers: user, supervisor, and kernel. The PEOS includes software modules that control the local operations of the processing element and also contains modules of the distributed network operating system that provide access to remote resources.

The kernel is at the lowest level of the PEOS. It contains a set of operations that are called by a well-defined set of primitive commands. These calls are used by the kernel in order to perform the primary functions of managing the processor, the primary memory and the controllers for peripheral devices. Section (3.2) outlines a possible implementation of the kernel.

With the exception of the kernel, all processing on the machine is done by processes defined in Section (1.10.2). In order to accomplish their work processes call on the primitive operations of the kernel and communicate with other processes to exchange information. Processes on a given PE may share reentrant common utility subroutines. The CCIU accomplishes all communication between processes, either local or remote, by means of messages (as opposed to shared memory). Using messages for local interprocess communication causes additional overhead. However, a unified communication method for all processes makes interprocess communications independent of the location of the communicating processes and the migration of processes from one PE to another (to accomplish backup) is facilitated. To communicate with a remote process the PEOS uses its network interface module.

The two outer layers of the PEOS, the supervisor and the user layer, contain nothing but processes. The kernel operates in the hardware privileged mode.

Processes run concurrently sharing memory, I/O devices, CPU's, and other resources. It is necessary, therefore, to protect them from injury by a malfunctioning process. It also necessary to schedule resources. All this is accomplished by the kernel (running in the privileged mode) by restricting process access to resources such as the memory, processor time, and by controlling communication to other processes. In the CCIU, a capability mechanism has been chosen for controlling process access to anything it does not own.

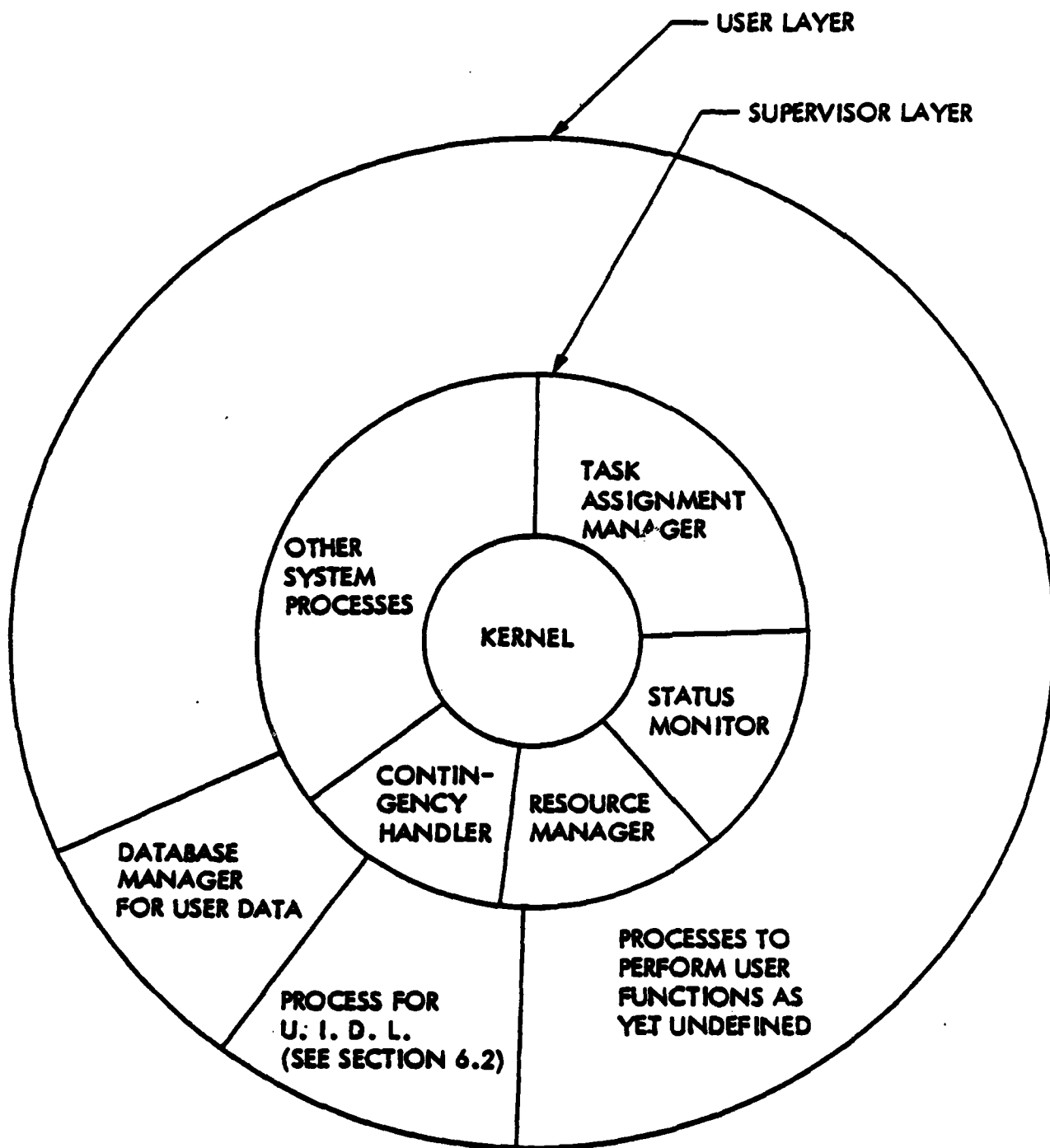


Figure 1. Processor Element Operating System (PEOS)

In the supervisor layer there are a number of processes identified as being peculiar to the CCIU system. These are: (1) the task assignment manager, and task manager that controls locating, location and activation of tasks, (2) the resource manager that controls assignment of PE resources and may attempt to off-load processes onto another PE if that is desirable, (3) the contingency handler that attempts to redistribute backup functions within the system as necessary to handle problems arising from failed PE's and communication links, and (4) the system status monitor that contains information regarding system configuration status and CCIU performance. Note all of these processes require some knowledge of other system PE's and lie within a Control-VIP.

The user layer contains processes that are accessed by the user or are common sub-processes used by the primary user processes. Due to the well-defined nature of CCIU functions, these user processes will be command interpreters that accept user commands and execute appropriate functions.

3.2

PEOS Kernel

The PEOS kernel will be the only part of the PEOS that operates in the hardware-privileged mode and has access to all privileged instructions. It contains the capability mechanisms. All other software will operate in the unprivileged mode. Not all of the "trusted code" is in the kernel. The trusted code outside of the kernel is able to perform its function by accessing restricted objects through its capabilities; access is mediated by the kernel.

The kernel operates the machine hardware, i.e., the processor, the memory and the peripheral devices. The kernel provides entry points for user and system software to access functions it provides. These entry points permit the kernel to manage the software objects it provides to other software modules. Basic types of objects supported by the kernel are:

1. Processes
2. Memory blocks
3. Devices
4. Capabilities
5. Messages

These software objects are described next.

3.2.1

Processes

A process was defined in Section 1.10.2. A function of the kernel is to manage the allocation of the processor and other resources to

the processes competing for their use. As a message-handling-facility the CCIU requires that large numbers of processes be created, used and destroyed. The process management facilities must be able to do this without incurring a large overhead.

Primitives required for process management are: (1) create process, (2) destroy process, (3) initialize process, (4) run process, and (5) return from process. These primitives are described below.

- (1) Create process - constructs a new process out of a program module (more than one concurrent process may be created from the same module). Invoking this primitive causes a Process-Control-Block (PCB) to be created with suitable entries that identify the process to the system and add this PCB to the appropriate queue.
- (2) Destroy process - removes all knowledge of the process from the system tables.
- (3) Initialize process - obtains initial memory space for the process and provides initial information for it. Initial information may consists of initial values for some variables and messages from the calling process. The initial memory for a process consists of space for its variables and stack space. See, e.g. the Local Name Space (LNS) in [WULF 81] and see [MYER 80].
- (4) Run process - passes control of the processor to the process. It is normally invoked by a scheduler.
- (5) Return - blocks a process from further using the processor. A process can use this primitive to relinquish control of the processor back to the PEOS. Ordinarily, the PEOS will regain control of the processor as a result of other circumstances. Either an interrupt will occur or the running process will make a service call to the PEOS that will result in the OS using the processor to process the service call. The OS can make a scheduling decision regarding the next process to run.

The two primitives "run" and "return" are intimately concerned with scheduling the processor resource. Normally the PEOS will assign a priority to a process from information provided by the process initiator. A time allotment based on the process priority will be assigned and used for sharing the processor among processes. Time given the process each time it runs depends on both time allotment and machine load. Processes will continue to run until their time allotment is used or they stop for a reason outlined above. The kernel decides which process to run next. Note: waiting processes of higher priority than the process currently running can obtain the processor at any time interrupt point. It is planned that process priority assignment be handled outside the kernel by some scheduling process. Management of the ready process queue is so

closely tied with kernel functions such as processor assignment, memory swapping, interrupt handling, etc., it should be in the kernel itself.

The kernel provides access to information about processes on request to a process with proper capabilities. As an example, the priority scheduler needs to know how much accumulated processor time a given process has used to determine changes in the priority of the process so that an absolute time limit to accomplish the work is met.

3.2.2 Memory Blocks

The kernel must have the ability to assign memory space to processes both for their exclusive use and for controlled-sharing with other processes in the same processing element. Memory is organized hierarchically in two levels: main storage fast memory and secondary storage on disk or other media. The PEOS includes a virtual memory manager that implements the paging system.

3.2.3 Devices

The kernel controls I/O operations and physical I/O devices. I/O device control primitives are (1) start I/O, (2) I/O completion. Invoking the I/O interrupt handler is not a kernel primitive because it is initiated by the hardware itself.

3.2.4 Capabilities

The general consensus is that the provision of security in an OS must be part of the basic design and not something added to an existing system. The latter technique has been used many times and has always led to systems that were easily penetrated. The security technique currently considered the most promising is the capabilities technique. For more on the subject of operating system security in general see [HSIA 78]. Capabilities-addressing of system objects is used in several current research OS's: [POPE 79; WARD 80; MCCR 80; BHAT 80; SINC 80]. The inverse of capabilities, the access list, is used in the MULTICS system at M.I.T. Note: capabilities-addressing has other advantages in an OS [FABR 74]. Capabilities can be implemented by means of software or by direct hardware decoding of the capability. The latter is a more secure technique but requires a special processor memory unit. Such a hardware technique has been used in the IBM SWARD machine [MYER 80], in the IBM System 38 [BERS 80], and in the INTEL 432 system. We will assume that the software-based technique using ordinary hardware is being implemented in the CCIU.

Our model provides that every process is given some private memory accessible only to it; the memory is protected by the memory management system address access limits. Any sharable object and any message object is created and given a name by the operating

system on request. Sharability of memory blocks should be considered; this can significantly decrease the process communication overhead. The block would be created by agreement between the processes obtained through message passing.

There are three primitives required in the kernel for capability management: create, grant, and revoke. In addition to these primitives the kernel must also use its knowledge of process capabilities to control access to objects when other kernel primitives are used by the various processes. In the UCLA secure UNIX kernel [POPE 79] the capability information storage is inside the kernel. Other systems do not do this; it tends to make the kernel large thus consuming a large amount of fast memory storage. A middle ground technique would be to provide secondary storage accessible only to the kernel. This can be done if the kernel implements a simple, paged file system with its own capability control, and the user-accessible file system is a process using the basic pages. This is the file system proposed in previous paragraphs. Such a file system (but without capabilities in secondary storage) has been used in [POPE 79] and in the XEROX WFS system on the experimental Ethernet.

A capability code should not be reused for a new object after the original object has been deleted. This leads to very long capability strings in the mechanization (e.g., 82 bits in the SWARD system, [MYER 80]); it may be desirable to investigate a technique for relaxation of this requirement.

It is necessary distinguish among various types of access rights in the capability. Different access controls must exist for reading from, writing into, using, and deleting objects. In addition, there should be a set of rights granting permission for altering capabilities as they are passed to other processes. Such alteration may consist of restricting the rights. In [WULF 81] the necessity for amplifying rights in some situations is considered.

3.2.5

Messages

The message handler will be a trusted process outside the kernel. The kernel will be involved in message handling because (1) a capability will be involved for sending the message and (2) the actual link level of communication will involve interrupt-driven hardware. Messages may also serve as a synchronization means between processes. [RAO 80] discusses message handling procedures.

Messages are a different category of object from other objects. Other objects handled by the kernel exist inside the PE and the kernel would immediately know what resources are available when a request is made. Messages go to another process that may be in another PE; there is no guarantee the other process is going to read the message within any given time. Therefore, the message

resource is not under kernel control and the strategy for handling all cases is difficult to determine.

Messages can be used to set up a direct channel between two processes for the exchange of data. See [WARD 80] where such channels are called streams and treated as a generalization of the UNIX pipe facility. This facility will be implemented outside the kernel. The kernel will be involved in creating and checking the capabilities required.

The strategy for message handling should attempt to avoid deadlocks. Deadlock avoidance has been shown to be an NP-complete problem; one solution that has been implemented is to detect deadlocks and resolve them. There are two common message strategies: (1) the sending process blocks after sending a message until it receives a reply, or (2) it continues immediately and tests later to see if the message was received by the receiving process. Both of these strategies can easily lead to deadlocks. The no-wait strategy will fail if the buffer storage resource is used up and no more messages can be queued. This can happen if a process does not read its incoming messages. The wait strategy will fail if two processes happen to send messages to each other simultaneously and neither can proceed until the other has responded.

Another consideration for synchronization is the action to be taken by a receiving process when it wants to read a message. It may continue immediately with other work and check occasionally, or it may block until the message is received. A receiver process that is listening for messages from anyone and attempting to do some other work simultaneously needs the former strategy. A process of this type is exemplified by the 'listening' socket on the Arpanet that sets up the secondary channels used when two hosts communicate.

The basic primitives for message passing are send message and receive message. Blocking or non-blocking options are essential to provide for fault tolerance and synchronization. They can be part of the arguments to the call. The basic kernel primitives will normally be issued only by the message handling process and not by any user process directly.

Sending a message requires an address. The address can be the name of a process or it can be the name of an entity established for messages that communicate with the process. The latter technique is commonly known as a "port". If there are large numbers of temporary processes in existence, it could be difficult for a sending process to know the name of the process it wishes to communicate with. This difficulty is overcome by the port that is a standard receive point for the classes of messages it redistributes. If this technique is used in the CCIU the port is usurping one of the prerogatives of the TAM which knows the location of all processes; therefore, the technique has not been adopted.

3.2.6

Additional Kernel Functions

The kernel has some functions that do not fit into the category of system object handling. These functions follow:

The kernel will be responsible for the initial actions of exception handling. This will be true because an errant program will be detected when it accesses memory outside its limits. The program could also be in a loop and doing nothing useful. Illegal accesses will cause a direct trap to the kernel. A loop will cause no harm to the system except for reduction in useful resources; the scheduler will only allow part-time access to the processor for the looping process. It may cause serious harm to the user because the process is not performing its function; methods for detecting such processes will be investigated. If the process is capable, it should handle final actions or exceptions because exceptions are synchronous events that occur every time the program runs. However, the program may not be capable of recovery and the OS may abort the process. In this case the task manager is invoked and the contingency handler notified; survivability now requires action. Permitting a process to handle its exceptions is common practice, usually limited by requiring that exceptions occurring in the exception handler cause an immediate abort.

Synchronization of remote cooperating processes must be handled by the network interface. Blocking options on messages is simplest. Techniques such as this may have too much overhead for processes in the same PE that need to synchronize. We assume again that these local synchronizations are common and we need to consider special actions for them; e.g., a set of kernel primitives implementing event flags. This type of synchronization is used in contemporary OS (v. DEC RSX11M). User programs that employ basic synchronization primitives like semaphores or event flags can be difficult to understand and maintain [BRYA 79]. Synchronization is a prime example of a side-effect. Provision of synchronization by constructs in the language employed by the user in writing the program leads to programs that work sooner and are more easily maintained. This puts the work of providing the synchronization on a system process implemented in the run-time support of the language.

3.3

Task Management

Task Management is a distributed function; recall that a basic property of the design is that no process needs to know where another process is located in order to call that process. It has been divided into two sub-functions implemented by the Task Manager (TM) and the Task Assignment Manager (TAM), respectively, although these processes call on other processes for support. Task management requires global knowledge and is a Control-VIP function.

The TM is replicated in every PE and oversees task management for the particular PE. The TAM is a process that maintains global knowledge about the location of processes in the CCIU. It oversees locating processes as well as possible migration of processes; the TAM assigns responsibility to a particular PE for process execution. TAM procedures exist in every PE to ensure survivability. However, as described below, the TAM in only one PE is fully functional at any one time.

3.3.1

TAM Overview

The TAM is logically in a Control-VIP distributed over several PE's. The TAM database in the configuration tables includes a dictionary to the location of processes and their backups. Section 2 of this document has described a hierarchical network structure based on layers of sub-networks each of which is expected to contain internally most communication messages among the PE's within its own layer. The sphere of control of a Control-VIP that contains a given TAM will extend over one of these subsidiary networks; the configuration table will normally contain information regarding processes in that layer. In the event that a large amount of traffic exists between TAMS in different Control-VIP's, special entries in the configuration table can be set up for individual cases. The limitation on configuration tables to one network layer necessitates that the TAM use the contract net protocol (described later) to locate processes outside of its own Control-VIP. In the ensuing description of the TAM, the discussion is primarily about the action of the TAM in controlling task assignment in a single Control-VIP; this is expected to be the normal case.

Methods are provided for regenerating the TAM database in case it is destroyed by PE failure. Regeneration must be a last resort recovery procedure because of the time and communication resources it uses up. On the other hand, the database for the TAM must be up to date at all times. To minimize the problem of update and concurrency control for this database, the TAM primarily executes in a single PE. The TAM processes are backed up in several or possibly all PE's to ensure system integrity. All PE's will be running a small part of their TAM code to communicate with the primary TAM process and to monitor for the continued existence of that primary process.

3.3.2

TAM Establishment

If the CCIU is operational, a TAM exists. If the PE that contains the current TAM fails, it is necessary to reestablish a TAM. A procedure will exist for identifying a new TAM that is triggered when some process attempts to access the TAM and finds that none exists. This process of reestablishing the TAM may take time because it is possible the database in the new TAM is out of date. A way of handling the situation and preventing it from arising is

for the residual TAM in each PE to notify the primary TAM of changes in processes it contains, and in case of a new TAM to identify all processes. The procedure for backup process migration is implemented through the TAM (it is initiated by the contingency handler) and therefore some of the communication traffic in exchanging process location data is alleviated. The alternative arrangement of having completely replicated databases that are updated every time a configuration change occurs is presently considered to have too high a communication overhead.

The procedure for reestablishing a TAM after a failure is not simple. A possible complication is that two PE's recognize there is no TAM simultaneously and both start recovery procedures. Care must be taken that two TAM's are not inadvertently established. Some algorithms applicable to the problem are described in [LUNN 81] and [GARC 82].

If a single PE is running, it uses its own TAM module with the standard configuration table subject to the knowledge only one PE exists. In this situation the TM does all the assignment work. When a PE is started (cold start), the operator may or may not elect to join the network. If the PE joins the network, its first action is to let the network know that it is on-line, and then locate a TAM. If no TAM exists (because it has just failed) the situation previously mentioned, where more than one PE discovers there is no TAM, can arise. A TAM can effectively "disappear" if a problem in the communication subnetwork causes its PE to become disconnected.

If a failure occurs in such a way that the network becomes fragmented into two or more parts, another serious recovery problem arises. In this case the recovery procedures will result in TAM's existing in each network fragment. Assuming the network again becomes reconnected, synchronization of the two TAM databases and "retirement" of one of the TAM's must occur. The exact recovery procedures for all of these abnormal conditions have not been worked out in detail.

3.3.3

TAM Functions

The exact way the TAM performs its location and assignment functions are dependent upon the degree of decentralization of control. Two different approaches are used by the CCIU. In the first approach, the TAM locates, by a series of query messages, a PE that is willing to execute a desired process. If several PE's are willing, the TAM makes a decision based on any information known to it or supplied in the acceptance messages from the several PE's. After this decision is made the TAM notifies the selected PE. This protocol for locating a PE is called the Contract Net Protocol (CNP) and its details are expanded in a later section.

In the second approach, the TAM consults the configuration table and arbitrarily selects a PE known to have the resources to accomplish the task. This PE is directed to run the process. It may be incapable of doing so in which case it notifies the TAM and another PE must be selected. This technique is known as the directed protocol.

It is clear that the contract net protocol has considerably more message overhead than the directed protocol. It is a much more "decentralized" approach because the TAM does not need to know initially the location of a suitable PE or even if one exists.

3.3.4

Task Manager

The TM is closely linked to the TAM but it exists to control task assignment in the particular PE, and perhaps to manage tasks via the CNP. The TM knows which processes are resident in the PE and may not notify the TAM at all if it needs to run one of them. Such an action is expedient to decrease communication overhead; it does not provide a global optimization to run the process in the PE that is, (through some criterion), most suitable at the given time. In case of PE overload, the TM may still decide to notify the TAM so that a system-wide decision can be made on what action to take. The continued use of the TAM/TM processes during the time that a process is running, and communicating with the process that started it, is an unnecessary load on the system. Therefore, one function of the TM is to set up addresses for messages to pass from one process to the other for communicating control messages and data. Such an address is a capability for one process to communicate with another.

3.3.5

TAM Protocols for Communication

This section describes the two protocols by which the TAM locates a PE capable of accomplishing a task; the TAM then assigns it responsibility for running the appropriate process.

3.3.5.1

Contract Net Protocol [SMIT 79]

This is a protocol where the initiator (a TAM), issues a task announcement either to all PE's or to PE's on a bidders list previously established by the initiator. PE's receiving the announcement may respond with a bid. After a certain time determined by the initiator (that can be stated in the announcement), the initiator selects a bidder using an internal criterion for establishing the "best" bid. Then the selected PE does the task.

This protocol permits nearly-complete autonomy of the participating processors, automatically exploits parallelism in the bidding process, and is structurally rich enough to support security and privilege levels of task execution. It is compatible with priority levels, nonhomogeneous processor resources, distributed execution

of tasks, and other peculiarities of the distributed computing environment.

A typical task announcement would include an eligibility-specification reflecting resource requirements for the task. The task announcement would also include an abstraction of the task adequate for the bidder to determine its eligibility, a specification for the information required in the bid, and a time by when the bid must be made if the processor so intends.

The establishment of a bidder's list (a TAM-peculiar function), requires a special type of task announcement giving eligibility specifications and task abstractions, but not requiring the bidder to commit itself to accomplishing the task at present .

A bid would provide the requested information and its issuance would commit the processor to do the task. The issuance of the bid requires estimation of its future processing-load by the issuing processor. The initiator should issue a task award message so the several PE's that have issued bids do not continue to commit resources for a job awarded elsewhere.

3.3.5.2 Directed Protocol

This is a protocol where the TAM directs a PE to accomplish a task. The PE selected by the TAM is determined from the CCIU configuration tables.

A PE must have the capability to refuse the task; it may be incapable of doing it. The TAM must be capable of coping with the conditions resulting from such a refusal. A PE may find it undesirable or impossible to do a task for several reasons:

- (1) It may not have the facilities, either data or process code, to accomplish the task.
- (2) It may be overloaded with current processes of equal or higher priority.
- (3) It may anticipate that overload will exist in the near future.
- (4) It may anticipate that some resource, say, disk storage space, will be exhausted by the task making it impossible to complete.

Item (4) above will require an absolute refusal to be issued. In the case of item (1), it is possible that the missing resource can be sent from some other processor if that other processor is unable to do the job itself. In all of (1) to (3), however, a system decision dependent on loading and priorities of many PE's is needed. Consequently, it is part of the TAM function to make this

decision. The PE elements do not possess the information or evaluative techniques required for the decision.

3.4 System Status Monitor

The system status monitor module knows the status of the system. It has a local aspect, keeping track of the status of resources in the PE, and a distributed aspect involving knowledge of the resources of other PE's in the Control-VIP.

The system status monitor is the module that maintains the configuration tables. These tables include the process location table containing information of the locating of processes and their backups. It also contains the behavioral template described below. Note the system status monitor is the only module actually accessing the configuration tables. All requests for information go through it simplifying implementation of any necessary changes in details of how information is stored.

3.4.1 Behavioral Template

The behavioral template describes the initial configuration of the CCIU PE as determined by the CCIU Configuration Controllers (see Section 6) from knowledge of particular PE hardware and software resources. The template also contains global information regarding the location of CCIU processes and functions. Thus, the template is part of a control-VIP and is a distributed database.

The CCIU's configuration will change during its existence because of PE's that have been taken out of service, PE's that are inaccessible for some reason, etc. The actual configuration may also change because system loading on a given PE may make it expedient to migrate some process from one PE to another. Therefore, the current configuration may differ from the template.

It is a system goal to approach the template as closely as possible during its operation. For example, if a PE is not available survivability may require the backup of its functions to be migrated from one PE to another. The code could be obtained from the PE where the function is currently executing or from another backup PE if multiple backups exist.

3.4.2 Configuration Tables

As the CCIU evolves during its operation, the actual configuration will be stored in the current configuration tables maintained by the system status monitor.

3.5 Resource Manager

The resource manager is the module that manages allocation of memory, disk space, processor usage and other PE resources. Other

modules can request these resources; they will be allocated depending on availability and priority of the requester. The resource manager therefore contains the scheduler for the processes running on the system; the actual context switch to run another process is handled by the kernel. The resource manager is also queried by the TAM to determine if resources for a new process whose activation is requested by another PE are available. Although the resource manager is primarily a local process, it has a distributed aspect residing in a control-VIP. In order to control its local resources the resource manager must be cognizant of other resources. It may, for example, find it desirable to migrate a process to another location to reduce the local loading.

3.6 Contingency Handler

The contingency handler activates backup modules and controls the allocation of backups in the event of PE failures.

A primitive form of contingency handler existed in the Demo 1 CCP; the contingency handler consisted of a 1-element table that indicated PE state.

3.6.1 PE Failure

If a PE failure is detected the contingency handler is notified and consults the configuration table to determine if the particular PE is a backup for any function in the failed PE. In this case, the task manager is notified to schedule the particular function for this PE.

This type of activation places the burden of determining the correct startup procedure on the function itself. Startup may differ radically depending on the function. In particular, some functions may only require a cold start while others may need information from the user input up to the present time. In the latter case the function's database needs to be updated in the backup PE's as the function runs. All of these situations are specific to the particular function.

3.6.2 Additional Backup

The contingency handler needs further action to determine whether or not an additional backup should be instituted. In some cases, a function may already have an additional backup. The desirability of additional backup is examined and if necessary the contingency handler selects an additional PE and oversees the installation of backup capability in that PE. This action can necessitate transferring code from one PE to another or notifying an operator to take an action to retrieve code from off-line backup storage.

3.6.3

Backup PE Selection

Techniques for selection of a new backup PE need further research. The abilities of a PE to perform the work, the load on the remaining PE's, and the amount of work needed to secure a valid backup are all involved. In addition, the contingency handler must not impose too much overhead on the particular PE where it is running [MAJU 80]. It is expected that heuristic search methods may be useful.

4.0

The CCIU Database Management System

4.1

Introduction

The CCIU Database Management System (DMS) is a distributed system that manages data at all levels of the Army command. Examples are the description of the CCIU configuration, the field locations information, manpower and other logistic information, map data, electronic mail, summary data, and all database transactions used to manipulate field data.

The goals of the DMS are: to provide fast access to data, to maintain security, integrity, and availability of data, and to provide data manipulation capabilities such as data reduction and data integration. DMS is unique among database systems in that it is required to be highly mobile and survivable.

The data managed by DMS is subdivided into two classes: user data and system data. User data consists of field data and knowledge-base data. The field data includes data acquired during operations such as sensor data (e.g., radar information), reports, messages, and saved data products (e.g., CRT images, tables). The knowledge-base contains invariant data that are loaded at the initialization of the CCIU and serve as reference data (e.g. computation tables). System data is the data necessary to control the CCIU. This data is not accessible by the user. It includes among others the behavioral template (see Section 3.0) and configuration tables.

DMS has two central functions: the first is to provide descriptions of the available data. This is the data dictionary function. It is used as a roadmap to the design, use and evolution of field user views. The second function of DMS is to manage data storage, manipulation, integrity, and security, as well as database concurrency. This is called the physical view of DMS; there is only one physical view representing the underlying structure of the union of all field user views.

There has been a significant amount of research addressing various issues in distributed databases. Results of this research have been reported in numerous papers. For the sake of brevity, these results are only referenced in this report. It is assumed that whenever appropriate commercial database software and conventional software engineering techniques will be used. The references given are not intended to be complete; they only indicate the large body of computer science literature relating to DMS. For a general discussion of distributed database management, see [FARB 75] and [GRAY 79]. Transaction processing in distributed systems is discussed in [GRAY 81]. Locking mechanisms are discussed in [CHU 74], [MENA 78], and [MENA 79]. Query distribution is the topic of [CHU

79]. Finally, various recovery mechanisms are discussed in [HAMM 78].

4.2

The Semantic User Interface

DMS must be usable by non-database specialists. A number of recent database research efforts have concentrated on developing semantics database models. These models attempt to embed the semantics of an application environment within the database schema by providing high-level data structuring and manipulation primitives. The goal is to make the database readily usable and evolvable by non-expert users. DMS utilizes a semantic database interface to field user views. Such an interface makes the database specification easily accessible during view formation and use.

Rather than basing their modeling constructs on the logical record (as the relational, hierarchical, and network models do), semantic models typically represent the application environment as a collection of objects. Objects have attributes and are classified according to types and subtypes. Semantic models provide static and dynamic capabilities. The designer provides high-level, application-specific database transactions within the database schema that the user can directly initiate. DMS provides a unified set of semantic static and dynamic structures to model control and field user views. A summary of essential techniques and recent research in semantic database models is provided in [KING 82].

Control views are the views of the database used by the system. Within the DMS they are modeled using special network constructs. The designer of a CCIU configuration specifies such entities as the network PE names, PE locations, and communication links between PE's. The constructs available to specify the static aspects of field data are essentially a unification and simplification of those found in various semantic database models. The dynamic aspects are expressed in a simple programming language containing primitives that directly manipulate semantic static constructs. Similar constructs are used to model user views. These views consist of objects and transactions. There are two types of transactions, manipulation transactions that update data, and perusal transactions that do not.

DMS includes an interactive facility for browsing the semantic dictionary to define field user views. A field user view consists of a number of disjoint logical schema, each formed using the semantic user interface. The definer of an original view will only allow access to perusal transactions and may not include manipulation transactions in the field user view available in the control view. Any user can define new, arbitrary perusals at any time, but may not define new manipulations, except on his own original views. An interactive facility also exists for guiding the user through the definition of new perusals. A generalized report writer exists for formatting data for output. A number of summary functions are

available for deriving summary field data. In summary, the DMS user interface supports the easy manipulation of field data and "meta" data concerning schema structure.

In a field user view, there are two varieties of objects: descriptor objects are atomic strings of characters or numbers, and generally serve as symbolic identifiers in a database; abstract objects are non-atomic entities, defined in terms of their relationships with other objects through attributes. An attribute is a mapping from one type of object to another, and is generically associated with a type; at any given time, each object of the modified type (attribute domain) has a specific attribute value that is a subset of the modifying type (attribute range). If an object uniquely identifies an instance of an object type, it is a primary attribute; otherwise, it is a dependent attribute. Attributes have a number of options that enforce integrity constraints; for example, attributes may be defined as being single-valued or multi-valued, non-null, etc. Subtypes of descriptor and abstract object types can be defined by specifying predicates on the values of attributes, or by performing summary operations on the instances of existing types. A subtype inherits all of the attributes of its parent type, and may have new attributes as well.

Field data consists of raw and derived data (subtypes). An example of raw data object is printing-devices. Printing devices would perhaps have attributes describing the type of output they produce (device-type), the computers they may interface with (required-host), and the identification number of the devices (device-number). Line-printers is an example subtype, derived with a predicate on device-type. Another example abstract object type is radar-images, with the attribute image-type serving as a unique identifier, and classification serving as a dependent attribute that gives a numerical indication of the threat indicated by each radar image. An example subtype is summary-radar-data consisting of one radar image per image type; that image represents the numerical average (in terms of classification) of the threat.

A transaction is a high-level manipulation of an event database. Each transaction definition specifies a particular generic type of transaction by defining the following: a collection of parameters, that are of descriptor or abstract object types; a collection of working subtypes that define the object subtypes the particular transaction manipulates and a collection of actions. An action is either an invocation of another transaction or an invocation of a primitive data operation (e.g., add instance to type, remove instance from type, update attribute); the actions of a transaction are structured using sequencing, selection/conditional, and iteration control structures. Transactions are organized in a type/subtype hierarchy, in a manner similar to the object type/subtype hierarchy; one transaction is a subtype of another if the value types of the parameters of the former are subtypes of the value types of the parameters of the latter, and if the actions of

the former are intuitively a special case of the actions of the latter.

A restricted form of transaction is a perusal, which accommodates unplanned queries of event databases and may not update data. A perusal is the specification of an object subtype at database access time; it typically consists of a subtype of descriptors. An example perusal might be the subtype of integers, consisting of all device-numbers; the number describes a printing device.

An example application event type is distribute-map-data, with parameter location (the identifier needed to isolate a network PE). The working subtypes of distribute-map-data may be subtypes of the object type map-data, which the various subtypes isolated by examining an attribute called distribution-indicator. The actions of distribute-map-data would be to invoke the network to ship the required map data to each PE. Presumably, this event would be initiated by a central map data authority. Another possibility: subtypes of the event would be initiated by the commanders at the various PE's. That is, a commander would request current map data on his location whenever it is desired.

4.3

Field User Views and Physical Views

DMS data sharing is defined along two dimensions: the first measures the degree of sharing, and the second, the level of command where the data sharing takes place. With respect to the first, field user views may consist of either primitive data schema or unification of one or more preexisting views. The field user views have various security levels assigned to them. One security level means that no field user other than the original owner of the view may use it. (The control views and physical view always have this security level). Other security levels indicate what clearance is necessary to use the view. A field commander can protect certain data from sharing by forming a separate view describing it, and giving that view a high security level. Some forms of data may be so confidential that even its description cannot be made "public"; this type of data would not be made available for sharing by anyone. A field user view definition also indicates the location of any other views used to create the given view.

If a user view can be used in other views (if it has any security level other than the most restrictive), then a description of that view appears in the data dictionary. The data dictionary is available for reading to all users, it indicates the structures of the various views, their location, and their security levels. To build a new view a user can either define new data types and transactions, and/or consult the control view and request the network to provide access to existing user views. A new view is the sum of any new or borrowed views that form it; no specific operations exist for merging views. If a user view designer wishes

to compress or summarize data, it must be done through the DMS user interface.

As part of its operation, the CCIU would collect live data (radar, map, order of battle, etc.) and produce summary data. A typical field view would consist of two separate views, one representing the data collection and summary data production events and corresponding objects. The other view would have a lower security level, and would provide summary data perusing events for other users to initiate. Each field user provides other users with the capability to read his data. A given user may also integrate other existing field user views into his own. These "borrowed" views would be used to direct the updating of data the given user is collecting; for example, the producer of map data may need to read summary radar data.

The other spectrum along which data sharing is measured is the level of command at which the data is being used. Very possibly, a field commander which allows sharing only on application events that did not update his data. Any other events would be available for use by other commanders, at any level of command. Also, commanders above him could almost certainly have complete access to all his user views. Most likely, higher command levels would augment their views with events that condense the data in lower views; this would allow commanders to focus on major issues. Thus, views are likely to be grouped into a tree, with each commander having access to the views of all individuals below him. However, in order to avoid duplication of data, all users (especially those absorbing views from lower levels of command) would most likely derive all their views from original views. In other words, if two users have partially overlapping views, and if they have a common commander, that commander would have two logical "copies" of part of his data if he were to blindly place both views in his own.

The physical view is not visible to field users. It serves as a logical representation of the physical state of the database and assists the physical designer in the process of implementing semantic data constructs. It describes the location of the data represented by each view, including information concerning duplication of data, and database control. The physical view is therefore made up of a number of physical schema. The separation of data views from the data (called data independence) provides for mobility: as the CCIU configuration evolves, data can be shipped between processors on the network without the user's awareness. In other words, the user knows what processor maintains the interface to his view (the schema), but need not be aware of the location of the actual data. Duplication of data provides for survivability and reliability any number of duplicate physical schema can be kept and DMS will coordinate their correctness; DMS will automatically swap in a duplicate if a data set is lost or damaged. Like data independence, duplication is invisible to the field user.

Physical schema describe the raw field data existing on the CCIU network. Physical schema describe the physical level of field data, and the physical view is a control view of the physical level. Logical schema are semantic dictionaries of the field data, as viewed at the user's level. Logical schema provide a description of the structure of available data, and a description of predefined transactions for manipulating data. (Note: a distinction must be made between a database schema and the data; the form of actual data is described immediately below.)

The physical data level is a machine-independent organization of data, described in terms of the relational model. It serves as the starting grounds for the physical implementer of a CCIU system. It also provides a framework for representing the high-level semantic constructs in a form closer to typical data structures and access mechanisms. (It is logical to use a commercial relational database system as the underlying database; in this way, the translation from the physical view to an actual database would be fairly simple.) All logical views of capability and field data can be decomposed into physical views. (Although logical views of field data may be described in terms of other logical views, and not directly in terms of physical views.) All types and subtypes are mapped into relations; attributes are mapped into columns, and attribute domains are mapped into columns domains. Options are associated with columns, and are enforced procedurally at data manipulation time. Each unique object in the database is represented as a unique internal identifier; anywhere a given identifier occurs, it represents the same object. Transactions are represented as source code in a host language (e.g., Ada); that is, all transactions are compiled into a standard programming language.

The physical level also addresses a number of implementation considerations. As redundancy is an important mechanism in DMS, each physical schema is given a name and a version number. All versions with the same name are considered identical at all times.

Each physical schema is also given a location name, in order to facilitate network transmissions of data updates. (Logical schema do not require location names, as the logical level of data is available to all users and updates are performed through the physical level.)

In a battle situation, DMS must provide reliable data manipulation and querying facilities. The key to DMS's reliability lies in its semantic interface and strict data independence. Field users may easily construct and use data views, and may specify arbitrary levels of partial sharing. Disappearance of a PE is invisible to the user if a copy of all required field and meta data still exists. Further, separation of physical implementation (in terms of some underlying database management system) and logical implementation decisions (in terms of a physical view) provides for complete flexibility in evolving a physical implementation without

changing the field user's view of the data. In this way, DMS can maintain a reliable view of the database, regardless of whether changes in field conditions have resulted in reorganizing field data or physical devices.

4.4

Data Control

DMS must support description and manipulation of a wide variety of data. Semantic constructs available for data definition and manipulation provide simple, uniform techniques for structuring and evolving all forms of data, from net description information to map data. Similarly, uniform mechanisms exist for controlling update and duplication of all forms of data. DMS provides reliable database mechanisms and wraps them in a user-friendly interface. This section briefly discusses special data control aspects of DMS: decentralization, security enforcement, integrity control, database locking, and duplication. DMS is a decentralized database system. This means that updates to independent field user views, the physical view, or the control view do not have to be addressed through a central interface. There is no processing "bottleneck": if a user wishes to update his data he does not have to address the network, he merely performs the update locally. Cross-network updates will occur only when one user reads data from another user's view, or when a duplicate data set is being updated.

A security code is associated with each logical view of field data. As a logical view is built recursively, it inherits the strictest security code of its components. Thus, a field commander may protect his data against use by other field commanders. Physical views also have security levels that identify individuals authorized to update the underlying implementation of the corresponding user views. Access to data is mediated through the capability mechanism of the operating system to enforce security. All data is assumed to be encoded; the user interface automatically decodes data, assuming the user has proper clearance (that is, some sort of log-on procedure).

Most integrity mechanisms are automatically controlled through enforcement of attribute options. For example, restricting an attribute to being single-valued or non-null implies an important integrity constraint. DMS could easily be extended to include a number of other integrity constraints, including ones that control relationships between attributes. This would mean extending the predicate mechanism used to derive subtypes to include derivation of attribute values. For example, it could be defined that the value of one attribute of one type must be equal to the numerical sum of two attributes of another type.

Small levels of granularity on locks may be derived for pre-defined transactions. The semantic interface enables very efficient concurrency control by examining working subtypes of the various transactions to derive the exact subtypes and attributes an appli-

cation event manipulates. Data contention will not be a problem in a given DMS implementation; the owner of an original user view will probably retain data update privileges exclusively for himself and his superiors. In other words, the vast majority of cross-network data operations will consist of perusals of summary data that is updated in a periodic manner. (Raw radar transmissions will probably be of little use to anyone). Therefore, a large amount of data reading will not require locking, as summary data will be updated in batch.

The definer of a physical view may provide for any number of versions of the view. Because of its importance, the control view's physical view is a likely candidate for multiple versions. CCIU will automatically enforce version control, and will automatically locate a duplicate when a user attempts to access a "missing" data set. Also, data may be archived off-line for extra protection against loss.

The physical implementer of a DMS configuration would normally define a number of duplicate physical views, in order to protect data against destruction. If a network PE (computer) "disappeared" during battle, DMS would automatically locate any needed duplicate data sets. Due to the extreme level of data independence, physical implementation or network configuration may be changed without effecting the user views at all. If a network failure or PE disappearance results in the control view or physical view being updated, users may continue processing their data uninterrupted (unless all duplicates of a data set or all paths to a required data set have been destroyed).

4.5

A Note on Tailoring

DMS must be supported on a network of small machines. Due to the necessity of utilizing existing Army equipment, the various machines on the network may be a variety of brands. Further, commercially available database management systems suitable for small computers typically do not support distribution of data. The semantic modeling constructs available in the data definition facilities of DMS are simple enough to be supported in a minicomputer environment, and very little software would be needed to support the schema merging and data manipulation operations. The data control and schema definition facilities of the underlying commercial database management system will have to be tailored to suit the needs of DMS.

Software Security

The mechanism for security has been treated in the description of the PEOS kernel. As described it consists of a capability for system objects unique to each object and assigned by the kernel when the object is created. The creator possesses the initial capability for the object and can instruct the kernel to grant that capability to other processes as desired. In the previous description reference was made to other possible systems for protection.

This protective technique is primarily intended to provide a secure operating system where an erroneous or malicious process cannot injure another process, cannot crash the system, and cannot gain access to data if it has not been given a capability for that access. We have not, however, addressed certain other security issues of equal importance to the user. These issues are concerned with the granting of capabilities to others by processes that legitimately possess them. A process must be capable of granting a capability that it has for an object to another process if the object is to be shared. This situation can result in a capability being passed along to several processes in order that they may operate on the object involved. If security level compartments are required it is important that a highly-classified process not be capable of writing highly classified data into a lower level area of classification. We believe that our capability method provides a framework for the development of such secure systems. Finally, we have not considered encryption in any way since it is not within the scope of the project. Possibly encryption could be used on system data or on communication channels between processors.

A protection problem also exists with initial assignment of capabilities. Capabilities must be assigned by the configuration controllers (see Section 6) for the CCIU system. From the time of their initial assignment until they are entered into the kernel database they must be protected by means outside the system.

Protection of the user interface from unauthorized use will be handled by the usual methods. Perimeter protection can be used for this purpose as well as user authentication passwords.

6.0 Human Factors and User/Developer Interface

6.1 Access Classes

Access to the CCIU will be provided for three classes of personnel. These are (1) users, the normal user of the CCIU, (2) configuration controllers, and (3) software developers.

6.1.1 Users

Users of the CCIU will be those military personnel who make use of the facilities and information which the CCIU can provide. These users will interact with the CCIU through a specialized directive command language that will in turn provide access to sublanguages appropriate to the particular function they are using. This interaction will be considered in more detail in Section (6.2). Access to the CCIU will be controlled for individuals by encrypted keys or passwords.

6.1.2 Configuration Controllers

It is essential that certain authorized personnel have more detailed access to the inner mechanisms of the CCIU to exercise control over its internal functions. These are called configuration controllers. They will be able to make changes in the behavioral template to provide some flexibility in the configuration of the CCIU. They can, for example, change priorities of functions and set up the initial configuration of backups. A configuration controller will also be able to cause a given PE to join or leave the CCIU. In a full-scale CCIU there will also be various global state variables which the CCIU attempts to adjust for optimal performance. An example is the location of backup PE's for a function that may need to change as PE's are disabled. The selection of these sites should be an automatic CCIU operation, based on existing PE's and their traffic loads. The basic algorithm the CCIU uses for selection may contain parameters which are selected by the configuration controller. This job is particularly important in the initial testing stages as a particular CCIU complex is implemented and experience is gained in its performance.

It is emphasized that a configuration controller has special privileges recognized by the CCIU when he 'logs on'. A user cannot make changes in the CCIU internal state.

6.1.3 Developers

A third access class is developers. This is the group that develops software to be used on the CCIU and is able to create new functions or modify existing ones. Field commanders will not develop CCIU Software. All developers will be in Post Deployment

Support Commands and have access to special CCIU's or components as needed.

It is a consequence of the above situation that field CCIU installations will not need to possess such software as Ada compilers, linkers, and the other environmental support software necessary for program development.

6.2

User Interface

The user interface to the CCIU will be through the User Interface Directive Language (UIDL), which is a problem-oriented executive language. It will delineate specific functions that can be performed by the system. For example, a possible command could be: "DISPLAY HISTOGRAM OF DEADLINED VEHICLES BY DEADLINE-CAUSE". This type of executive language is in contrast to a language which is procedural and would require the user to make separate requests to query the database to find the appropriate vehicles, construct a histogram, and then activate a display of the histogram. A series of commands used sequentially to accomplish a higher level function, not directly implemented, can be stored and then recalled by name.

The user language must be designed so it can be used easily by inexperienced users. This can be accomplished by including several features described below. (1) On-line commands will be provided which inform the user of the available commands in the system. (2) While executing a given command the CCIU will provide, if requested, additional information as to what alternatives are available when user input is needed. (3) Automatic recognition of commands when they are distinguishable from other commands, although the complete command has not been input, will be provided. (4) The user can set a level of expertise into the system and the amount of information he receives for prompting will be adjusted accordingly. (It is very frustrating to look up additional information off-line and equally frustrating to watch a great deal of useless prompting when the user is an expert). This type of user interaction was pioneered in the TENEX operating system developed by Bolt, Beranek, and Newman.

The user interface also provides a great deal of control over access to the system. When the user "logs on" to the CCIU his identity is known. The logging process can authenticate the user, and connect him directly to a command processor specifically designed for his level of access and with capabilities for the appropriate data and command objects. This is possible because the user is completely isolated from direct interaction at the machine level and has only the capabilities granted by the log in process. A similar access control was pioneered by MIT and is used in their MULTICS system.

6.3

Software Development Support

Development of CCIU software is only possible by individuals carrying the special privileges of the developer and is not done in the field environment.

GLOSSARY

ACCS	Army Command and Control System
Ada	Defense Department computer language.
Applications User	A non-privileged user who has access only to appropriate user applications.
Army C ³	Army Command, Control and Communication
Associative Mass Storage System	A mass storage system in which data can be accessed by content rather than address, e.g., LIST NAMES OF ALL RED-HAIRED MEN, rather than LIST LOCATIONS 165FA, 165FB, etc. (locations pointing to names of red-haired men).
BAA V	Battlefield Automation Appraisal Five
Backbone Network	Comprises the three higher layers of the CCIU network.
Backup	Replication of the user functions at several CCIU nodes and use of redundant communication links to guarantee survivability.
Behavioral Template	The hardware, software, communications configuration that the CCIU attempts to maintain [SAC-Doc] (See Current Configuration).
BER	See Bit Error Rate.
Bit Error Rate	A ratio describing the transmission quality of a channel.
CACDA	The U.S. Army Combined Arms Center.
CAL	See CCIU Architectural Level.
Capabilities	Capabilities are system-wide names for objects (e.g., I/O devices, memory pages, data types, messages). A process has access to an object only if it possesses a capability for that object.
Capabilities Mechanism	Allows cooperating user functions to share data objects, each with a capability associated with it.
Capability	Each shared object has a capability associated with it; a capability is permission to use that object.

GLOSSARY

CCIU	Command and Control Information Utility
CCIU Architectural Levels	The architecture of the CCIU is structured in three levels called the CCIU Architectural Levels. They are the Command Slice Functional User Level, the Control Level, and the Processor Element level.
CCP	Cellular Command Post - a DPM. A collection of processing elements that together perform the information processing functions of a command post.
CCS ²	Command and Control of Subordinate Systems - a DPM.
CECOM	U. S. Army Communications and Electronic Command
CENSEI	Center for System Engineering and Integration-- a component of CECOM.
Channel	A path along which signals carrying data can be sent. The term implies one-way communication, while "circuit" implies two-way communications [DAVI 79].
CNP	See Contract Net Protocol
Command File	A file containing a sequence of directives, either in the User Interface Directive language or a procedural language that causes execution of a sequence of processes.
Command Slice	A level in the chain of command, e.g., Corps Level → Corps Slice.
Command Slice Functional User	The key member of the basic user population for which CCIU is designed. Generally corresponds to a functional cluster at a particular command slice.
Command Post Network	The level of the CCIU communication network that interconnects the processor elements of a CCP.
Communications Protocol	A strict procedure required to initiate and maintain communications. Protocols may exist at many levels in one network such as link-by-link, end-to-end, and subscriber-to-switch.

GLOSSARY

Concurrency Control	Management of parallel processing to maximize parallelism while maintaining process synchronization.
Configuration Controller	A specially-privileged user who has access to User Interface Directives that modify CCIU configuration and control.
Configuration Tables	CCIU actual configuration stored in the current configuration tables; maintained by the system monitor.
CONOPS	Continuity of Operations
Consistency	A database may be viewed as a finite set of objects that are assigned values. The set of values taken by all objects is called a state of the database. Associated with the database is a set of assertions about database objects and relations between database objects called consistency constraints. The state of a database is consistent if the values taken by all objects constituting the database satisfy the consistency constraints. [LELA].
Context Switching	Term applied to operating system activity of allocating the Central Processor Unit (CPU) of a computer to one process that is ready to execute and at the same time moving the process that formerly had the CPU to some form of waiting or killed state.
Contingency Handler	An OS routine which oversees actions taken to ensure or increase survivability after a failure.
Contract Net Protocol	A protocol for migrating the responsibility for executing a process that preserves a high degree of autonomy among processors [SAC-Doc], [SMIT 79].
Control Element	Coordinates technical activities of Subordinate Systems included in the functional element; provides command/staff type information to other control elements or accepts such information from them.

GLOSSARY

Control Level	A CCIU Architectural Level.
Control-VIP	Virtual Information Processor that implements the Control Level of the CCIU Architecture [SAC-DOC].
Correctness	A program may be represented as a set of logical assertions or a theorem. If the theorem (or set of assertions) can be proved by formal logic, the program is said to be correct.
Correctness Prover	An automated capability to prove program correctness. See Correctness.
Current Configuration	The hardware, software, communications configuration that exists in the CCIU at any time. May be different from the Behavioral Template as a result of environmental causes.
CPN	Command Post Network.
CSFU Level	Command Slice Functional User Level of CCIU.
CSFU-VIP	Virtual Information Processor that supports a Command Slice Functional user.
Database	A collection of interrelated data stored together to serve one or more applications [MART 77].
Database Management System	The collection of software required for using a database [MART 77].
Database Manager	See Database Management System.
Database Model	A generalized logical framework for describing the relationships among data objects. There are currently three widely accepted data models: The Network Model, the Hierarchical Model, and the Relational Model. Adapted from [CARD 79]. These three models employ low-level record constructs. <u>Semantic Database Models</u> are database models that present to the user a view of his database environment that is expressed in terms of high-level constructs. These constructs are more semantically expressive of the user's environment.

GLOSSARY

Deadlock	A deadlock is said to occur when a process can never be granted all the resources it needs to reach completion (adapted from [LELA]).
Directed Protocol	A protocol for migrating the responsibility for executing a process. The processor to which the process is directed has only limited choice to refuse the assignment.
Distributed Processing Model	DPM: One of a set of models which characterize the back up and communications requirements for the CENSEI distributed processing development program.
Distributed Processing System	A computing system in which processing is distributed among two or more processors. "True" distributed processing is usually associated in the literature with "control decentralization" (see e.g., [LELA 79]).
DMS	See Database Management System.
DPM	See Distributed Processing Model.
Echelon Network	Links together unit networks at the same echelon level.
Echo Algorithm	One of a class of decentralized algorithms for determining arbitrary characteristics of a network in a manner that avoids exponential growth of associated message overhead.
Ethernet	A type of local network protocol first developed by the Xerox Corporation.
Functional Cluster	Refers to clusters of command and control functions that are characterized by a high level of interaction within the cluster but a lower level of interaction between clusters. There are five functional clusters: Maneuver, Fire Support, Intelligence and Electronic Warfare, Air Defense Artillery, Combat Services Support.
I/O Device Driver	Processor software dedicated to servicing specific I/O devices.
ISO-OSI Open Systems Interconnection Model	The Open Systems Interconnection Model. An International Standard Model for the structure of the communications interface in a network PE [ZIMM 80].

GLOSSARY

Kernel	A layer of the PEOS; implements the basic system primitives.
LNS	Local Name Space
Locking	A mechanism that limits access to a data object to only one transaction as long as necessary to preserve consistency [LELA].
Logical Message	Message in the format recognized by a process (see Physical Message).
Loosely Coupled System	The rate of interaction BETWEEN processors making up the system is small compared to the rate of interaction WITHIN processors [SIMO 80].
Mass Storage	Very large (megabyte) secondary storage, usually consisting of a hierarchy of storage media in which data is staged from slow-access, low-cost media to rapid-access, higher-cost media.
Migration	As used here, refers to the capability to designate any appropriate processor to execute a process. It may require moving code to the designated processor.
Non-Procedural Language	Language that specifies WHAT rather than HOW. (See Procedural Language.)
Operating System	Those program modules within a computer system that govern the control of equipment resources such as processors, main storage, I/O devices, and files. These modules resolve conflicts, attempt to optimize performance, and simplify the effective use of the system [MADN 74].
Operating System Kernel	The operating system nucleus. The kernel provides a small number of primitive operations callable from user processes. It implements a number of abstract objects and valid operations on each object. It is the only module in the system empowered to execute hardware privileged instructions [POPE 79].
OFFAC/CONOPS	Operational Facility; Continuity of Operations - a DPM

GLOSSARY

OS	See Operating System.
Ownership	In the SAC-Doc, the authority of a field commander to move items that are CCIU resources and to elect whether or not to put them at the disposal of the CCIU.
Packet Protocol	A block of data handled by a network in a well-defined format including a header and having a maximum size of data field. Consequently, a message may have to be carried as several packets [DAVI 79].
PCB	Process-Control-Block
PDSS Command	Military commands that have the responsibility for Post Development Software Support.
PE	See Processor Element.
PEOS	Processor Element Operating System
Physical Message	Message in the format recognized by the carrier. (See Logical Message)
Post Deployment Software Support	See PDSS command.
Predicate	That which is asserted about a subject.
Predication	A logical assertion.
Presentation Control Process	A process that initiates and maintains a presentation. Presentation is the ISO-ANSI term for the data translation activity that takes place between heterogeneous objects, e.g., that necessary to maintain a display.
Privileged Functions	Functions to which access is limited to authorized users.
Privileged Machine Instructions	Instructions which can only be executed when the machine is in a special mode utilized only by the OS.
Privileged User	One capable of modifying CCIU software in ways over which the CCIU has no control.
Procedure	A body of code, linked and ready for execution

GLOSSARY

Process	An instance of a procedure that is active on a PE. Processes accomplish the computing on a system.
Process Control Block	An information storage block, created for every process at the time the process is created, that contains information needed for the process to execute, e.g., process priority, capabilities, data, etc.
Process State	The status of a process relative to its eligibility to gain access to the CPU [SAC-DOC]. There are five states: Active, Waiting, Blocked, Submitted, Inactive or Killed.
Process Synchronization	Multi-process systems involve atomic operations. An operation is a set of actions performed by one or more processes at the instigation of another process. "Atomic" means either all actions are completed and a consistent system state is achieved, or no actions are completed and the current state is unchanged. Synchronization is the process of enforcing atomicity despite conflicts among concurrent operations, unreliable communications, variable communications delays, variable numbers of processes and processors [JENS 80].
Processor Element	A collection of one or several processing units under control of a SINGLE operating system. It is the fundamental computer element of the CCIU.
Processor Element Level	A CCIU Architectural Level.
Protocols	Provide services such as message sequencing, priorities, error and flow control. Also establish standard data elements and communications conventions [MCQU 78, TANE 81].
Query	A type of database transaction in which data is retrieved from the database.
Relation	A flat file or table. A relation is the basic data object of the relational data model. Rows in the table are called "tuples." Columns are called "domains."

GLOSSARY

The contents of a row-column intersection is called an attribute value. The term "flat file" refers to the requirement that an attribute be atomic, i.e., not a repeating group; (adapted from [CARD 79]).

Relational Model	A set-oriented data model. See Database Model
Reliable Network	Communications system that delivers messages in sequence and accounts for lost and duplicated messages.
Resource Manager	Module that manages allocation of memory, disk space, processor usage and other PE resources.
Roll-Back	The process during reexecution of a failed transaction, of restoring the database to its (consistent) state before the transaction started (see Roll-Forward).
Roll-Forward	The process of reexecuting a failed transaction.
SAC-Doc	System Architectural Concepts Document—short title for Preliminary System Architectural Concepts: Army Battlefield Command and Control Information Utility (CCIU).
Schema	A map of the overall logical structure of a database [MART 77].
Semantic Database Model	See Database Model
Semantic Dictionary	A description of the logical structure of a database including all logical views and available data transactions.
Slice	See Command Slice
SS	Subordinate System. A processor element of a CCP. (Compare with Control Element.)
Subschema	A map of the logical structure of that subset of the schema used by a single application or group of applications [MART 77].
Survivability	Probability that a system can continue to perform its intended function at an acceptable level in a hostile environment. Specifically, the CCIU's ability to continue operation even with loss of some of its communication links and processor elements.

GLOSSARY

System Integrity	Includes all internal aspects of the CCIU concerned with proper operation.
TAM	See Task Assignment Manager
Task Assignment Manager	A Task Force that tries to maintain global knowledge of the CCIU and oversees the locating of processes and migration of processes.
Task Force	A collection of cooperating processes, perhaps coordinated by an executive process.
Task Manager	Controls task assignment in an individual PE.
TCS	Military Tactical Computer System
TCT	Military Tactical Computer Terminal
TM	See Task Manager
TRADOC	Training and Doctrine Command
Transaction	Any activity that involves retrieving, adding to, changing, or deleting database objects.
Transparent	See Transparent/Virtual
Transparent/Virtual	In the sense used in the SAC-Doc, transparent and virtual are related. A virtual object is one that appears to the user to be present, but is not. A transparent object appears to the user NOT to be present, but is. A CONTROL-VIP is a virtual object. A disk I/O processor is a transparent object.
UIDL	See User Input Directive Language
Unit Network	Links together the CPN's belonging to the same Army unit.
User	A non-privileged user of the CCIU.
User Input Directive Language	High-level language used by users to interface with CSFU-VIP.
VIP	See Virtual Information Processor
Virtual	See Transparent/Virtual

GLOSSARY

Virtual Circuit

A system in which messages are delivered to a specified destination over many different routes that may be transparent to the user. The link appears to the user as a direct connection between the sender and receiver.

Virtual Information Processor

A virtual machine (i.e., a software processor) optimized to meet an interrelated set of requirements.

REFERENCES

- [BERS 80] Berstis, V., "Security and Protection of Data in the IBM System/38", Proc. 7th Annual Symposium on Computer Architecture, May 6-8, 1980, P. 245.
- [BHAT 80] Bhatt, D., P. Sadayappan, R. Kieburztz, and D. Smith, "An Operating System Kernel for a Hierarchical Multicomputer", IEEE Publication, CH-1598, 1980, P.665.
- [BRYA 79] Bryant, R., and J. B. Dennis, "Concurrent Programming", in "Research Directions in Software Technology", P. Wagner ed., The MIT Press, 1979.
- [CHAN 79] Chang, E., "An Introduction to Echo Algorithms", Proceedings, 1st International Conference on Distributed Computing Systems, Oct. 1979.
- [CHU 79] Chu, W., and P. Hurley, "A Model for Optimal Query Processing for Distributed Databases", Tutorial: Centralized and Distributed Database Systems, eds. W. W. Chu and P.P. Chen, IEEE, 1979.
- [CHU 74] Chu, W., and G. Ohlomacher, "Avoiding Deadlock in Distributed Databases", Proceedings ACM National Symposium, Volume 1, November, 1974.
- [CLAR 78] Clark, D., and K. Pogran, and D. Reed, "An Introduction to Local Area Networks", Proceedings of the IEEE, November 1978.
- [DIED 81] Diedrichsen, L., "CENSEI Exploratory Development Program Plan for Distributed Processing", March 15, 1981.
- [FARB 75] Farber, D., "Distributed Databases-An Exploration", IEEE Computer Networks:Trends and Applications, June 1975.
- [FABR 74] Fabry, R., "Capability-Based Addressing", Comm. of the ACM, July 1974, Vol 17, No. 7, P. 403.
- [GARC 82] Garcia-Molina, H., "Elections in a Distributed System", IEEE Trans on Computers, Vol C-31, No. 1, Jan. 1982, P. 48.
- [GRAY 79] Gray, J., "A Discussion of Distributed Systems", IBM Research Report, RJ2699(34594), IBM Research Laboratory, San Jose, 1979.

- [GRAY 81] Gray, J., "The Transaction Concept: Virtues and Limitations", Proceedings of the Seventh International Conference on Very Large Databases, Cannes, France, September 9-11, 1981.
- [HABE] Habermann, A., "Introduction to Operating Systems", Science Research Associates, 1976.
- [HAMM 78] Hammer, M., and D. Shipman, "An Overview of Reliability Mechanisms for a Distributed Database System", Proceedings, COMPCON, Fall, 1978.
- [HEST 81] Hester, S., D. Parnas, and D. Utter, "Using Documentation as a Software Design Medium", Bell System Technical Journal, Oct. 1981, P. 1941.
- [HSIA 78] Hsiao, D., D. Kerr, S. Madnick, "Operating System Security, A Tutorial of Current Research" COMPSAC Proceedings, 1978, P. 220.
- [JONE 79] JONES, A. et al, "StarOS a Multiprocessor Operating System for the Support of Task Forces", Proc. 7th Symposium on Operating System Principles, Dec. 1979.
- [KING 82] King, R. and D. McLeod, "Semantic Database Models", in Database Design, ed. S. B. Yao, 1982 (to appear).
- [LELA] LeLann, G., "Distributed Data Management", Unpublished manuscript circa 1979.
- [LELA 79] LeLann, G., "An Analysis of Different Approaches to Distributed Computing", Proc. 1st Int. Conf on Distributed Computing Systems IEEE, Oct. 1979.
- [LUNN 81] Lunn, K., and K. Bennett, "An Algorithm for Resource Location in a Loosely Linked Distributed Computer System", Operating Systems Review, Vol. 15, No. 2, April 81, P. 16.
- [MAJU 80] Majumdar, S. and M.Green, "A Distributed Real Time Resource Manager", Proceedings Distributed Data Acquisition, Computing and Control Symposium, IEEE, 1980, p. 185.
- [MCCR 80] McCreory, T., "The X-Tree Operating System: Bottom Layer", COMPCON, Spring 1980, P. 340.
- [MCQU 78] McQuillan, J., and V. Cerf, "Tutorial: A Practical View of Computer Communication Protocols", IEEE Computer Society, IEEE Catalog No. EHO 137-0.

- [MENA 78] Menasce, D., and G.J. Popek, and R. Muntz, "Centralized and Hierarchical Locking in Distributed Databases", Tutorial:Distributed Database Management, IEEE, 1978.
- [MENA 79] Menasce, D., and R. Muntz, "Locking and Deadlock Prevention in Distributed Databases", Tutorial:Centralized and Distributed Database Systems, eds., W. Chu and P. Chen, IEEE, 1979.
- [MYER 80] Myers, G., and B. Buckingham, "A Hardware Implementation of Capability-Based Addressing", Operating Systems Review 14, 4, Oct. 1980, P. 13.
- [POPE 79] Popek, B., M. Kampe, C. Kline, A. Stoughton, M. Urban, and E. Walton, UCLA Secure Unix. NCC Proceedings, Vol. 48, 1979, P. 355.
- [RAO 80] Rao, R., "Design and Evaluation of Distributed Communication Primitives", ACM Conf. Distributed Computing West, 1980.
- [RENN 81] Rennels, D., A. Avizienis, and M. Ercegovac, "Fault Tolerant Computer Study", Final Report, JPL Report 80-73, Jet Propulsion Laboratory, Cal Tech, Feb 1981.
- [SINC 80] Sincoskie, W. and D. Farber, "The Series/1 Distributed Operating System", Description and Comments, Proc. COMPCON, Fall 1980, P. 579.
- [SMIT 79] Smith, R., "The Contract Net Protocol:High-Level Communication and Control In A Distributed Problem Solver", Proceedings, 1st International Conference on Distributed Computing Systems, Oct. 1979.
- [TANE 81] Tanenbaum, A., "Network Protocols", Computing Surveys, Vol. 13, No. 4, December 1981.
- [TAUS 78] Tausworthe, R., Standardized Development of Computer Software, Jet Propulsion Laboratory, Caltech, 1978, 2 Vols.
- [WARD 80] Ward, S., "TRIX:A Network-Oriented Operating System", COMPCON, Spring 1980, P. 344.
- [WULF 81] Wulf, W., R. Levin, and S. Harbison, HYDRA/C.MMP: An Experimental Computer System, McGraw-Hill Book Co., 1981.
- [ZIMM 80] Zimmermann, H., "OSI Reference Model-The ISO Model of Architecture For Open Systems Interconnection", IEEE Transaction on Communications, COM-28 (April 1980).