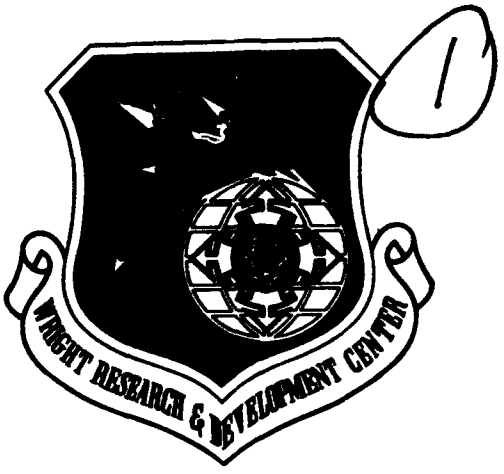
WRDC-TR-90-8007
Volume IV
Part 2

AD-A250 123



INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Volume IV - IISS System
Part 2 - System Design Document

DTIC
ELECTE
MAY 07 1992
S D
D

J. Althoff, S. Barker

Control Data Corporation
Integration Technology Services
2970 Presidential Drive
Fairborn, OH 45324-6209

September 1990

Final Report for Period 1 April 1987 - 31 December 1990

Approved for Public Release; Distribution is Unlimited

MANUFACTURING TECHNOLOGY DIRECTORATE
WRIGHT RESEARCH AND DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433-6533

92 5 04 155

92-12200

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, regardless whether or not the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data. It should not, therefore, be construed or implied by any person, persons, or organization that the Government is licensing or conveying any rights or permission to manufacture, use, or market any patented invention that may in any way be related thereto.

This technical report has been reviewed and is approved for publication.

This report is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations

DAVID L. JUDSON, Project Manager
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

FOR THE COMMANDER:

BRUCE A. RASMUSSEN, Chief
WRDC/MTI
Wright-Patterson AFB, OH 45433-6533

25 July 91
DATE

If your address has changed, if you wish to be removed form our mailing list, or if the addressee is no longer employed by your organization please notify WRDC/MTI, Wright-Patterson Air Force Base, OH 45433-6533 to help us maintain a current mailing list.

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release; Distribution is Unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) SDS620340000	5. MONITORING ORGANIZATION REPORT NUMBER(S) WRDC-TR-90-8007 Vol. IV, Part 2
6a. NAME OF PERFORMING ORGANIZATION Control Data Corporation; Integration Technology Services 6b. OFFICE SYMBOL *(if applicable)*	7a. NAME OF MONITORING ORGANIZATION WRDC/MTI
6c. ADDRESS *(City,State, and ZIP Code)* 2970 Presidential Drive Fairborn, OH 45324-6209	7b. ADDRESS *(City, State, and ZIP Code)* WPAFB, OH 45433-6533
8a. NAME OF FUNDING/SPONSCRING ORGANIZATION Wright Research and Development Center, Air Force Systems Command, USAF 8b. OFFICE SYMBOL *(if applicable)* WRDC/MTI	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUM. F33600-87-C-0464
8c. ADDRESS *(City, State, and ZIP Code)* Wright-Patterson AFB, Ohio 45433-6533	10. SOURCE OF FUNDING NOS.
11. TITLE *(Include Security Classification)* See Block 19	PROGRAM ELEMENT NO.: 78011F; PROJECT NO.: 595600; TASK NO.: F95600; WORK UNIT NO.: 20950607
12. PERSONAL AUTHOR(S) Control Data Corporation: Althoff, J., Barker, S.	
13a. TYPE OF REPORT Final Report	13b. TIME COVERED 4/1/87-12/31/90
14. DATE OF REPORT *(Yr.,Mo.,Day)* 1990 September 30	15. PAGE COUNT
16. SUPPLEMENTARY NOTATION WRDC/MTI Project Priority 6203	

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify block no.)*
FIELD	GROUP	SUB GR.	
1308	0905		

19. ABSTRACT *(Continue on reverse if necessary and identify block number)*

This document describes the IISS design specifications.

Block 11 - INTEGRATED INFORMATION SUPPORT SYSTEM (IISS)
Vol IV - IISS System
Part 2 - System Design Document

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED x SAME AS RPT. DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL David L. Judson	22b. TELEPHONE NO. *(Include Area Code)* (513) 255-7371 22c. OFFICE SYMBOL WRDC/MTI

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE

FOREWORD

This technical report covers work performed under Air Force Contract F33600-87-C-0464, DAPro Project. This contract is sponsored by the Manufacturing Technology Directorate, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. It was administered under the technical direction of Mr. Bruce A. Rasmussen, Branch Chief, Integration Technology Division, Manufacturing Technology Directorate, through Mr. David L. Judson, Project Manager. The Prime Contractor was Integration Technology Services, Software Programs Division, of the Control Data Corporation, Dayton, Ohio, under the direction of Mr. W. A. Osborne. The DAPro Project Manager for Control Data Corporation was Mr. Jimmy P. Maxwell.

The DAPro project was created to continue the development, test, and demonstration of the Integrated Information Support System (IISS). The IISS technology work comprises enhancements to IISS software and the establishment and operation of IISS test bed hardware and communications for developers and users.

The following list names the Control Data Corporation subcontractors and their contributing activities:

SUBCONTRACTOR	ROLE
Control Data Corporation	Responsible for the overall Common Data Model design development and implementation, IISS integration and test, and technology transfer of IISS.
D. Appleton Company	Responsible for providing software information services for the Common Data Model and IDEF1X integration methodology.
ONTEK	Responsible for defining and testing a representative integrated system base in Artificial Intelligence techniques to establish fitness for use.
Simpact Corporation	Responsible for Communication development.

QUALITY INSPECTED 4

Accesion For
NTIS CRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification .....
By .....
Distribution /
Availability Codes
Dist | Avail and / or Special
A-1

Structural Dynamics Research Corporation	Responsible for User Interfaces, Virtual Terminal Interface,and Network Transaction Manager design, development, implementation, and support.
Arizona State University	Responsible for test bed operations and support.

Table of Contents

		Page
SECTION 1.	SCOPE	1-1
1.1	Identification	1-1
1.2	Functional Summary	1-1
1.3	DAPro Project 6203 Enhancements and Additions	1-2
SECTION 2.	REFERENCES	2-1
2.1	Applicable Documents	2-1
2.2	Terms and Abbreviations	2-1
2.3	System Overview	2-5
2.3.1	Background	2-5
2.3.1.1	Relationship of the Test Bed to Other ICAM Projects	2-5
2.3.1.2	Strategy for Evolution	2-6
2.3.2	Summary of Expected Benefits of the Test Bed and IISS	2-8
2.3.3	Test Bed System Overview	2-10
2.3.3.1	Hardware Architecture	2-10
2.3.3.2	Software Architecture	2-12
SECTION 3.	REQUIREMENTS	3-1
3.1	System Definition	3-1
3.1.1	Network Transaction Manager Configuration Item	3-1
3.1.1.1	Network Transaction Manager Mission Statement	3-1
3.1.1.2	Network Transaction Manager Functional Areas	3-1
3.1.1.3	Network Transaction Manager Operational Scenario	3-3
3.1.1.4	Network Transaction Manager Functional Specifications	3-31
3.1.2	Common Data Model Configuration Item	3-46
3.1.2.1	CDM Mission Statement	3-46
3.1.2.2	CDM Functional Areas	3-46

Table of Contents

		Page
3.1.3	The User Interface Configuration Item	3-69
3.1.3.1	User Interface Mission Statement	3-69
3.1.3.2	User Interface Functional Areas	3-69
3.1.3.3	User Interface Operational Scenarios	3-71
3.1.3.4	Functional Specifications	3-80
3.1.4	The Virtual Terminal Interface Configuration Item	3-92
3.1.4.1	Virtual Terminal Mission Statement	3-92
3.1.4.2	Virtual Terminal Functional Areas	3-93
3.1.4.3	Virtual Terminal Operational Scenarios	3-93
3.1.4.4	Virtual Terminal Functional Specifications	3-100
3.1.5	The Communication Subsystem Configuration Item	3-100
3.1.5.1	Communication Subsystem Mission Statement	3-100
3.1.5.2	Communication Subsystem Functional Areas	3-100
3.1.5.3	Communication Subsystem Operational Scenarios	3-100
3.1.5.4	Communication Subsystem Functional Specifications	3-115
3.2	Interfaces	3-118
3.2.1	Information Interfaces	3-119
3.2.2	Services Provided	3-119
3.2.2.1	Integrated Application Programs....	3-120
3.2.2.2	Non-Integrated Application Processes	3-121
3.2.2.3	Common Data Model	3-121
3.2.2.4	Distributed Database Processes	3-121
3.2.2.5	Local Database Management Systems	3-124
3.2.2.6	Network Transaction Manager	3-125

Table of Contents

			Page
	3.2.2.7	User Interface	3-126
	3.2.2.8	Communications	3-127
	3.2.2.9	Test Bed Monitor	3-127
	3.2.2.10	Interprocess Communications	3-127
	3.2.2.11	Virtual Terminal Interface	3-128
	3.2.2.12	Host Operating Systems	3-128
	3.2.3	Protocols and Messages	3-128
	3.2.3.1	AP to AP	3-129
	3.2.3.2	AP to CDM	3-129
	3.2.3.3	AP to NTM	3-129
	3.2.3.4	AP to UI	3-129
	3.2.3.5	CDM to CDM	3-130
	3.2.3.6	NTM to NTM	3-130
SECTION	4.	QUALITY ASSURANCE PROVISIONS	4-1
	4.1	General	4-1
	4.1.1	Responsibility for Test	4-1
	4.2	Special Tests and Examinations	4-1
SECTION	5.0	PREPARATION FOR DELIVERY	5-1
	5.1	Hardware	5-1
	5.2	Software	5-1
	5.3	Documentation	5-1
	5.3.1	NTM Programmer's Manual	5-1
	5.3.2	NTM Operator's Manual	5-1
	5.3.3	Common Data Model Administrator's Manual	5-1
	5.3.4	Precompiler User's Guide	5-2
	5.3.5	NDML User's Manual	5-2
	5.3.6	UIMS User's Manual	5-2
	5.3.7	VTI Programmer's Manual	5-2
	5.3.8	Forms Processor Application Programmer Manual	5-2
	5.3.9	Interim Forms Editor Manual	5-3
	5.3.10	NDDL User's Guide	5-3

List of Illustrations

Figure	Title	Page
2-1	Interconnection of Heterogeneous Systems via Local Area Network	2-10
3-1	NTM Configuration Tree	3-2
3-2	Cluster APs	3-4
3-3	Ease in Implementation	3-5
3-4	Host 1	3-6
3-5	IISS Architecture - Conceptual Model	3-8
3-6	NTM Environment on One AP Cluster	3-9
3-7	NTM Environment on UI WS	3-10
3-8	NTM Environment on COMM WS	3-11
3-9	Operate Network Transaction Manager Functions	3-15
3-10	Manage Messages Functions	3-16
3-11	Manage Processes Functions	3-17
3-12	Maintain Operability Functions	3-18
3-13	Communicate with Application Process	3-19
3-14	NTM Architecture Test Bed Overview	3-20
3-15	Spawning Scheduler	3-23
3-16	Message Authentication	3-26
3-17	NTM IDEF1 Concepts	3-28
3-18	NTM IDEF1 Concepts	3-28
3-19	End-to-End Addressing	3-29
3-20	Guaranteed Delivery	3-32
3-21	CDM - Configuration Tree	3-48
3-22	A-0 Develop Integrated Application Processes	3-48
3-23	A-0 Operate CDM	3-50
3-24	A0 - Operate CDM Functions	3-51
3-25	A1 - Maintain CDM Data Functions	3-52
3-26	Develop Integrated Application Processes	3-56
3-27	Precompile Application Process	3-58
3-28	NDML Precompiler Environment	3-59
3-29	Distributed Query Scheduling & Control	3-63
3-30	User Interface Configuration Tree	3-70

List of Illustrations

Figure	Title	Page
3-31	User Interface Management System	3-71
3-32	Display/Retrieve a Form	3-74
3-33	Filling Out a Form - User Viewpoint	3-74
3-34	Control Diagram - Form Processor	3-76
3-35	Form Processor Start Up	3-77
3-36	User Log On Scenario	3-78
3-37	Application Generation	3-80
3-38	VTI Configuration Tree	3-94
3-39	Interface to a Real Terminal	3-95
3-40	Data Acquisition, Conversion, Transmission	3-96
3-41	Interface to an Existing Application Program	3-98
3-42	Communication Subsystem: Configuration Tree	3-101
3-43	Test Bed Local & Wide Area Network Configuration	3-103
3-44	Genet Permanent Virtual Circuit	3-104
3-45	Writing & Reading into Mail Box	3-107
3-46	Interhost Communication & Overview	3-109
3-47	Message Scheduling (Interhost)	3-111
3-48	Line Protocol Transmission	3-113
3-49	IISS Test Bed System Overview	3-120
3-50	IISS Test Bed Software Information Interfaces	3-122
3-51	Structure of a Typical IISS Application Process	3-123
3-52	Predefined Query Processing	3-124
3-53	Example Protocols Between IISS Test Bed Processes	3-130

SECTION 1

SCOPE

1.1 Identification

This specification establishes the conceptual design of the system identified as the Integrated Information Support System (IISS) otherwise referenced as the ICAM Test Bed. This system is intended to be a computing environment that provides integrated data management facilities and distributed processing for heterogeneous databases resident on heterogeneous computer systems interconnected via a Local Area Network. This computing environment is to be used for demonstrating the integration of the data produced by three distinct manufacturing subsystems: Shop Floor Management (MCMM), Decision Support (IDSS) and Material Requirement Planning (MRP).

This document has been prepared by Project Priority 6201M of the Air Force's ICAM program. The 6201M project was conducted by the General Electric Company. It was subsequently enhanced by Project 6202, conducted by the Boeing Military Airplane Company, and the DAPro Project 6203, conducted by the Control Data Corporation. Other participants included those contractors listed in the Foreword.

These projects were sponsored by the Manufacturing Technology Division of the Air Force Wright Aeronautical Laboratories.

Please refer to the Software Availability Bulletin, Volume III, Part 16, CI# SAB620326000, for current IISS software and documentation availability.

1.2 Functional Summary

The Integrated Information Support System (IISS) is a test computing environment used to investigate and demonstrate and test the concepts of information management and information integration in the contexts of Aerospace Manufacturing. Specifically, IISS addresses the problems of integration of data resident on heterogeneous databases supported by heterogeneous computers, interconnected via a Local Area Network. A Common subject to Project Priority 3101 Computer Based Information System (CBIS) guidelines. (Refer to the System Requirements Document, CI# SRD620340000.)

Data Model is maintained and provides the mechanism required to integrate the data. Initial data integration is targeting the Class II environment; however, IISS is required to be extensible to the Class I data environment. Note: Four data classes have been defined by the CBIS: Class I, data totally managed by CBIS; Class II, data directly accessed by CBIS, but externally managed; Class III, data subject to CBIS standards and

procedures; Class IV, data subject to CBIS guidelines. A fifth class, defined as "private data," is totally outside the control of the CBIS policies and procedures.

1.3 DAPro Project 6203 Enhancements and Additions

The DAPro project 6203 commitment was the adding of the following software enhancements and new software features to the IISS Release 2.3, as well as, continue to support all Test Bed activities:

Common Data Model Subsystem

- CDM reports and interactive application

- DEMO
  - NDDL, NDML, database DDL and data, run-time applications

- CDMR
  - In line code
  - Enhanced aggregation capabilities
  - IISS file input/output primitives
  - Rehost of runtime architecture to IBM
  - Enhanced logging capabilities for redundant data processing

- NDML
  - Logical operators
  - Parenthetical logic
  - Outer join capability
  - Select combination command
  - Complex mappings
  - Redefinition of data fields
  - Repeating data fields
  - Impact analysis (Software Cross Reference)
  - Separate precompilations of modules
  - DB2 inquiry/update capabilities
  - Enhanced redundant data capabilities
  - Enhanced view definitions processing
  - File name/module name generation enhancements
  - FORTRAN generation compatibility with IBM
  - Automatic generation of procedures to compile/transfer of generated code
  - Non NTM version of the NDML precompiler

- NDDL
  - Display/dump CDM contents
  - NDDL session control
  - Enhanced modeling commands:
    - Copy Entity
    - Copy Attribute
    - Copy Model
  - Attributes ownwership altering
  - Enhanced view creation capabilities:
    - NDML WHERE clause compatibility
  - Redundant data definition capabilities

- UIMS version of NDDL
- Non NTM version of the NDDL
- Descriptive text compatibility with UI subsystem
- Enhanced "DROP" processing:
  . enforcement of "no drop" if mappings for CS objects
  . enforcement of "no drop" if software cross references exist for objects

Communication Subsystem

o IBM - VAX COMM upgrade

Electronic Documentation System

o Phase 1
  - SGML Tagger for ICAM documents
  - generic DTD Builder for ICAM documents
  - SGML Parser
  - Layout Editor
  - Document Formatter
  - UI Postscript device driver
  - MacPaint/Postscript device driver
  - EDS User Manual

User Interface Subsystem

o Rapid Application Generator and Report Writer
  - Empty condition
  - Multiple conditions per function key
  - Conditions with Boolean logic
  - Comparison operators in RW conditions
  - Parameter forms
  - Call action
  - Communication calls
o Reverse VTI
o Business Graphs - Phase 1
o New color/graphics terminal support
o Dynamic fields
o Grandchild Support (UI-AP) in NTM
o File I/O primitives
o DB2 Support

Plus

o Add these enhancements and changes for Release 2.2.5 (Project 6202)
  - SYSGEN utility (User Interface)
  - ORACLE to sequential file conversion (User Interface)

All of these new features and enhancements are described in the Software Availability Bulletin, Pub. No. SAB620326000.

Note: These enhancements and new features were added and tested on the current Test Bed environment that consisted of two mainframe computers: a VAX/VMS host and an IBM only. The Honeywell computer used during the 6201 project was not included.

SECTION 2

REFERENCES

2.1 Applicable Documents

1. ICAM Documentation Standards, ICAM Document IDS 150120000A, December 28, 1981.

2. A.D. Little, ICAM Computer Based Information System (CBIS), System Environment Document, ICAM Document SED 310140000, September 1981.

3. A.D. Little, ICAM Computer Based Information System (CBIS), State of the Art Document, ICAM Document SAD 310140000, September 1981.

4. A.D. Little, ICAM Computer Based Information System (CBIS), System Requirement Document, ICAM Document SRD 310140000, September 1981.

5. Control Data Corporation & Dacom, IISS Test Bed CDM System Requirements, March 3, 1982.

6. General Electric Company, Test-Bed System Requirement Document, Revised June 11, 1982.

7. SofTech, ICAM Test-Bed Interim Standards and Procedures, ICAM Document ISP 620150000, February 1982.

8. ICAM Program Office, The Integrated Sheet Metal Center, September 30,1981.

9. N. Tupper, Memorandum for the Record, October 5, 1981.

10. SofTech, IISS Test Bed Network Transaction Manager System Design Specification, Document 1098-7, June 1982.

11. General Electric Company, Test Bed System Specification, revised August 23, 1982.

A listing of all of the documents produced as part of the final report of the 6203 project may be found in the Final Technical Report Volume I (FTR620300001).

2.2 Terms and Abbreviations

Active: Computer-enforced (at compile time or at run time).

Activity Framing: Feature which allows to declare a set of Application Processes as being part of a single operation which makes sense from the user viewpoint. All database changes contained within an activity frame are incorporated or else none are incorporated in the databases.

Application Process: A cohesive unit of software that can be initiated as a unit to perform some function or functions.

Application Process Cluster: The logical grouping of Application Processes and of one Message Processing Unit (MPU) NTM component.

Application Subsystem: An Application Subsystem consists of one or more application processes and performs specific manufacturing management functions. Instances of ICAM Application Subsystems are: MC-MM, IDSS, a commercially available MRP System.

Class II Data: Data for which query activity is under direct control of the IISS and for which update activity is under indirect control of the IISS.

Common Data:

1. Data used by more than one Application Subsystem.
2. Data updated by one Application Subsystem and used by another.
3. Data planned to evolve into a category described by (1) or (2) above.

Common Data Model: Describes common data application process formats, screen definitions, etc. of the IISS and includes conceptual schema, external schemas, internal schemas, and schema transformation operators.

Data Integrity: Improved quality, consistency and recoverability. The Test Bed common data is subject to the following integrity checks:

1. Type checking
2. Existence checking
3. Edit checking (7-digit telephone number)
4. Attribute value checking (shirtsize = small, medium, large)

Deadlock: Two processes are said to be dead locked when each process is waiting on the other to complete before proceeding.

Domain Check: Operation which ensures that the values of a given attribute lie within some prescribed set of values. These values may be continuous, discrete, numeric or non-numeric.

Expert/Novice Mode: The User Interface supports the concept of Expert/Novice mode of user interaction. In the novice mode, the user receives tutorial assistance from the system to guide his

selection of system features and functions. In the expert mode, the time-consuming tutorial assistance is suppressed for maximum efficiency.

Form: Predefined screen format description. The description includes the Textual, cursor positioning, data-checking information required to display or input data into IISS.

Guaranteed Delivery: Test Bed provided service which ensures the delivery of a message to its destination even if the destination process is unavailable at the time the message is issued.

Integrated (Test Bed) Application Process: An Application Process which:

1. Uses the Neutral Data Manipulation Language to retrieve Class II data which may be distributed on several databases resident on the Test Bed.

2. By the end of the contract, it uses the local database manipulation language to update the local database to which it is bound.

3. Performs its terminal input/output operations on the Test Bed terminals.

4. Is controlled from the Test Bed terminals.

Integrated Information Support System: (IISS), a computing environment used to investigate, demonstrate, test the concepts and produce application for information management and information integration in the context of Aerospace Manufacturing. The IISS addresses the problems of integration of data resident on heterogeneous data bases supported by heterogeneous computers interconnected via a Local Area Network.

Non-Integrated (Test Bed) Application Process: An Application Process which:

1. Does not use the Neutral Data Manipulation Language to retrieve Class II data. The Local database Management System data manipulation language is used for local database manipulation (update, retrieval).

2. Performs its terminal Input/Output operations on the Test Bed terminals.

3. Is controlled from the Test Bed terminals.

Non Procedural Query: Value stated query. The query statement focuses on what needs to be retrieved rather than on how to carry out the retrieval operations.

Paired Message: A message which contains either a:

a. Request for a reply, or

b. Reply to a uniquely identified request.

Pre-defined Query Application Processes: Information processing functions implementing predefined data query application processes. The code required for implementation is predefined (manually if necessary) precompiled and linked.

Response Time: Duration of wall clock time between submission of a user request and receipt of the first character of output.

Sychronization Point: Quiet point where the following is true:

1. The Test Bed databases are in a consistent state

2. The state of the queues of pending application process is known and available for future reference

3. The state of the databases is known and available for future reference

4. The state of the queues of pending application process and the state of the databases have been given a common identifier.

System Clean Point: State of the system which satisfies to:

1. The test bed databases are in a consistent state

2. The state of the databases is known and available

3. The state of the queues of pending messages is known and available.

System Quiet Point: Period of time during which the following is true:

1. The dispatch of messages triggering the execution of application processes is suspended.

2. All dispatched application processes are closed (processing is completed).

3. System quiet points are invoked and terminated under control of the Test Bed operator or Test Bed control mechanism.

Terminal Control Words: A neutral representation of terminal features implemented by specific control characters.

Test Bed Utilities: Test Bed functions that either provide Test Bed operability or facilitate the execution and terminal input/output operations of the Application Processes resident on the Test Bed.

Time to Complete: Duration of wall clock time between submission and completion of a user request.

User Interface: Test Bed services which facilitate the man machine dialogs between the Test Bed services, some of the Integrated or Non-Integrated Application Process resident on the Test Bed and the Test Bed user. The User Interface services are available through the Test Bed terminals.

2.3 System Overview

2.3.1 Background

The objective of this project is to establish and operate a Test Bed to validate the concept of Integrated Applications supported by an Integrated Information Support System (IISS). In addition, the project is to establish a set of interim standards and procedures to guide the design of the IISS and to provide guidance to other ICAM projects. Finally, a set of requirements is to be established which will be the basis for enhancements to the IISS.

This project is intended to provide the test and demonstration vehicle for the ICAM Information Support System concepts described in the 30 September 1981 "Integrated Sheet Metal Center" (Threads Document) and the Project Priority "3101 Computer Based Information System (CBIS) Requirements Document." As the strategy for evolution to the "CBIS data Class II" and "CBIS data Class I" (see footnote on Page 1-1 for a definition of the data classes) environments is developed and implemented, the associated costs and benefits can be tracked against the baseline system.

The ICAM products being considered by the Project Priority 2201/2 contractors for implementation in the Integrated Sheet Metal Center (ISMC) can be implemented first on the Test Bed. In this process, the problems of rehosting the software, integrating multiple ICAM products, and demonstrating performance will be identified and solved, thus reducing the risk to the ISMC implementator. Cost and performance evaluations of the products can be done within the Test Bed.

ICAM products not chosen for implementation in the ISMC can also be installed on the Test Bed, providing the same evaluation benefits and reduction of risk to other potential users.

2.3.1.1 Relationship of the Test Bed to Other ICAM Projects

The first Test Bed demonstration is to include the shop floor control system from Project Priority 6103 (Manufacturing Control Material Management - MCMM), integrated with appropriate modules of a commercially available Manufacturing Resource Planning (MRP) system. This integration will be supported by appropriate tools from the Integrated Decision Support System (IDSS), similar in capability to the "mid-configuration" described in the ICAM Program Office's 30 September 1981 ISMC "Threads Document."

The contents of the subsequent demonstrations will depend on several factors and come from several sources. For example, they could come from requirements/recommendations from within this Test Bed project, from the ICAM Program Office, from other ICAM Projects (particularly ISMC Projects 2201/2), and from industry in general. Enhanced capabilities in the manufacturing systems applications area (technical and control threads) will come mainly from projects 5501 (IPS) and the Integrated Decision Support Systems (IDSS) projects 8203 and 8205. System Engineering Methodology (SEM) tools will come from SEM Project 1701.

The Test Bed project has worked and will continue to work closely with other related ICAM projects in determining system requirements, defining standards and procedures for the initial implementation, and identifying deficiencies and voids in the available components. Beyond the functional and application areas, methods to be used in all aspects of the system life cycle are also to be addressed. Aspects to be coordinated with the SEM (System Engineering Methodology - ICAM Project Priority 1701) includes: data definition methods; database design; use of the data dictionary; structured analysis methods; system specification techniques; prototype development; standards for data definition; data manipulation; message definition; and documentation standards and performance analysis techniques.

Needs and priorities for enhancements will be defined by the ICAM Program Office based on recommendations from the Test Bed project coalition and related projects. Requirements for the enhancements will be defined by this Project (6201). Project Priority 1701M (SEM) will have the responsibility for designing and building the required software and defining the required standards, procedures, and guidelines based on the requirements. To facilitate the coordination of this Project 6201 with Project 1701, an informal working arrangement has been established with the Project 1701 Prime Contractor.

2.3.1.2 Strategy for Evolution

It has been estimated that in large U.S. corporations, most of the existing computer applications will be redesigned over the next 10 to 20 years. It is further expected that, due to the rapidly changing computer technology, the construction techniques and operation modes of new applications will bear little resemblence to those of existing systems.

The Project Priority 3101 CBIS coalition provided a set of six principles as guides in formulating a solution for the (relatively) near term which are also extensible for the expected longterm trends. Individually, each of these principles reflects state-of-the-art technology; however, they have not been implemented together to yield an integrated system. These principles are stated as follows:

1. IISS is a key mechanism for the integration of computerized manufacturing. It defines, controls, and executes actions affecting information among various functionally independent subsystems, based on the use of common data.

2. IISS employs a coordinated database approach to support information resource management of various application systems in a closed loop environment within manufacturing.

3. IISS implementation strategy employs several stages of data and application control which allow for increased usage of facilities as management seeks to gain greater benefits from the IISS.

4. IISS operates as a transaction oriented system responding interactively to user commands, rather than to prescheduled batches of computer programs.

5. IISS is accessible from geographically dispersed locations.

These principles and other results of the CBIS project were inputs to establish a starting point, extended and were further articulated by the ICAM Program Office and the Project 6201 coalition. Requirements, specifications, and the overall system design are being developed with a view of both short and long-term implementation plans for the Test Bed. To help focus attention on the long-range needs of the test bed, projections on the future direction of computer systems architecture, and the manufacturing environment to the year 1990 and beyond, and the impact this will have on the Test Bed technology, have been developed. This is published as the "Test Bed Migration Path" in Appendix C of the System Requirements Document (SRD620140000).

The "cost drivers" which the ICAM CBIS Requirements Definition (Project Priority 3101) defined as critically associated with the CBIS environment are summarized in the following nine categories, all of which are being considered as part of the Test Bed IISS design:

1. Data independence - making computer data files independent of the programs which use them.

2. Data nonredundancy - minimizing the number of occurrences of the same data in different files.

3. Data relatability - facilitating the changing of file structure based on specific "views" required by different programs and transactions.

4. Data integrity - improving data quality, consistency, and recoverability.

5. Data accessibility - providing low-cost, user-friendly access to data stored in various files and computers.

6. Data shareability - ensuring that many programs can access the same files simultaneously without degrading performance.

7. Data security - ensuring that data are isolated from users who should not have access to it.

8. Data performance - providing proper controls for changing the CBIS environment over time as changing user needs cause the basic system requirements to change.

9. Data administration - supplying appropriate standards, procedures, and guidelines to ensure consistent evolution of the CBIS environment as demands and technologies change.

Implied above is the need for the IISS to operate in a mixed environment containing old and newly developed applications. It is clearly recognized that the existing applications must be supported, while techniques for technology development and new applications development are concurrently provided.

2.3.2 Summary of Expected Benefits of the Test Bed and IISS

The Test Bed will serve as a step toward realizing the full benefits of a CBIS as represented by the "cost drivers" in the preceding section (Section 2.3.1.2). It will also serve as a facility to assist others to achieve these benefits faster and with less risk. Some of the benefits of the Test Bed may be summarized as follows:

1. Provide testing facility for individual ICAM software products.

2. Demonstrate initial integration of ICAM products.

   - o Data integration via the Common Data Model
   - o Techniques and procedures for more extensive integration of program functions

3. Provide a site for demonstration and evaluation of ICAM products.

   - o Applications
   - o Methodologies
   - o Information support system

4. Reduce risk to subsequent users of ICAM products.

5. Provide standards, guidelines, and procedures.

- o For development of ICAM products
- o For evaluation/adoption by industry

6. Demonstrate strategy for transition from current application processing and development methods to use of the evolving techniques which will subsequently reduce cost and increase system flexibility.

   - o Distributed heterogeneous systems, distributed data, and distributed processing.
   - o Independence of application data from considerations of actual internal storage organization and database Management System access techniques.
   - o Reduced data redundancy.
   - o Automated data validation and constraint checking through the Common Data Model.
   - o Transaction-oriented applications.
   - o Standardized user interface (similar menu construction for all applications, standard user "HELP" procedures, standard error messages, etc.).
   - o Control of execution, in a consistent manner, of processes on different computers using different operating systems.
   - o Facilitation and control of the passing of data and messages between processes on the same or different computers.
   - o Consistent error handling throughout the system.
   - o System-wide control of startup, shutdown, restart, and recovery.
   - o Application programs written using relational database languages referencing nonrelational databases.
   - o Independence of application program from the computer on which the user terminal is located.
   - o Independence of application program from the characteristics of the terminal on which it will be used.
   - o System-supported translation of information formats to host-specific representations.

2.3.3 Test Bed System Overview

The following is an overview of the presently envisioned Hardware and Software Architecture.

2.3.3.1 Hardware Architecture

The hardware architecture of the Test Bed supports the interconnection of the heterogeneous computer systems required to demonstrate the functionality of the Test Bed (Figure 2-1).

The Test Bed hardware architecture supports the interconnection of three computer systems via a Local Area Network (LAN) complemented by Wide Area Communication Services.

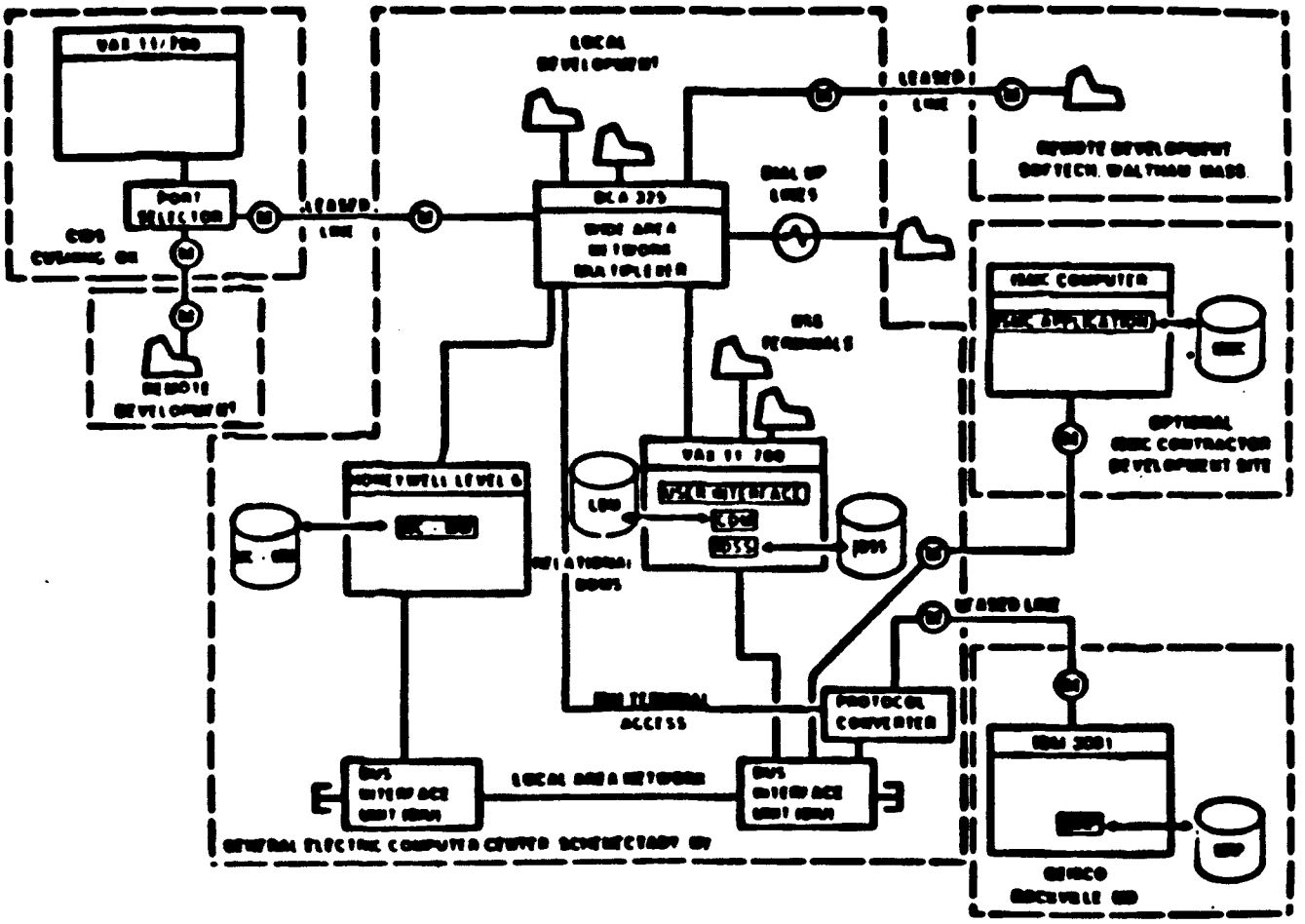


Figure 2-1. Interconnection of Heterogeneous Systems via Local Area Network

The three computers embedded in the Test Bed, at the time of this report, are:

1. A Honeywell Level 6 computer supporting the MCMM database and MCMM application programs.

2. A VAX 11/780 computer or equivalent supporting:

   - o Test Bed User Interface
   - o Test Bed Common Data Model
   - o IDSS 2.0 and its database

3. An IBM 3081 computer supporting an MRP package to be selected in conjunction with the IPS Project Priority 5501 team (see Figure 2-1).

The Test Bed makes use of a Local Area Network to interconnect the Honeywell Level 6 and the VAX 11/780 which are in close geographical proximity. This approach offers high throughput, ease of installation, expansion capabilities, and supports the process-to-process communication capabilities required to integrate the heterogeneous databases present in the Test Bed environment.

The Test Bed makes use of Wide Area Communication lines to extend the functionality and usefulness of the Test Bed to computers which are geographically remote.

1. A synchronous leased line provides medium speed communication capabilities to the General Electric owned IBM 3081 located 3 miles away from the computer center used to develop the Test Bed. (Later moved to Ge facility in Rockville, Md., and then to a Boeing facility in Wichita, Ka.)

2. Asynchronous lines are provided to interconnect the ICAM (CIDS) (Initial Implementation) and ISMC development computers to the Local Area Network hardware, as well as to interconnect ISMC development terminals to the VAX 11/780 of the Test Bed through the User Interface.

The Test Bed hardware architecture allows for expansibility and flexibility.

1. The Test Bed hardware architecture can be expanded to the MAX Configuration described in the Threads Document. Back end data machines and/or additional general purpose computers can be interconnected via the Local Area Network.

2. The Test Bed hardware architecture can be expanded to a full size production system with minimal changes to the software.

2.3.3.2 Software Architecture

The major software subsystems are also indicated in Figure 2-1.

2.3.2.2.1 Distributed Application Data on Heterogeneous Databases

The application data are distributed in heterogeneous database management systems, themselves resident in heterogeneous processors. This approach allows for the integrated query of existing databases by new integrated applications without conversion of the database.

2.3.3.2.2 Class II Data Integration

The Test Bed software and system utilities support Class II (see footnote, page 1-1) data inquiries. These inquiries may be directed toward any database resident in the Test Bed, and are under the direct control of the Test Bed Common Data Model Processor. System utilities perform data integrity checks selectively on the data being retrieved in the system. The early implementation of the Test Bed supports updates on the databases bound to the Application Subsystems. Update activities are under the control of the Application Subsystems and are under indirect control of the CDM to the extent that data entry and messages are checked by the CDM. The data integrity checks performed include edit, domain, and range checking. The data required to support the data integrity checks are contained in the Common Data Model and are under control of the Common Data Model Administrator. Data inquiries use the Test Bed Neutral Data Manipulation language to query Common Data contained in the databases integrated by the Test Bed. The Test Bed Neutral Data Manipulation Language allows the definition of nonprocedural queries which are independent of the structure of the database(s) being accessed. System services allow the retrieval of data contained in more than one database in more than one system.

2.3.3.2.3 User Interface and Data Query via Preplanned Transactions

In the early 1983 implementation, user interface and data query are accomplished by preplanned transactions and messages. (This method has evolved toward ad-hoc inquiries as development of the Test Bed continued after early 1983.) Information queries via preplanned transactions support the manufacturing scenarios which have been identified and constitute a natural first step toward ad-hoc query. Message integrity checking is supported by system functions, and is performed selectively. The information required to support the message integrity checks is contained in the Common Data Model and is under the control of the Common Date Model Administrator. (Message integrity checking: Future)

2.3.3.2.4 Integration and Coordination Through the Common Data Model

The Common Data Model is a resource which is maintained in a centralized fashion to support the following functions:

- Define logical structure of the information common to two or more Application Subsystems. The definition includes entities, their attributes, and the relationships between entities.
- Define the domain and values of the entities.
- Access control or authorization information identifying the operations that can be accessed by a particular user.
- Define the format of the data as stored.
- Catalog of Common database procedures such as schema translation, schema definition, Neutral Data Manipulation statement translation, data translation procedures.
- Locate the specified data in the logical data structure (Test Bed Conceptual Schema).
- Convert query requests to fit the locations of the data and the required processing.
- Aggregate the responses from the various databases.
- Check for the validity and completeness of update requests (Class II environment).
- Support the user interface.

2.3.3.2.5 Integration and Coordination Through the Integrated Network Transaction Manager

The Common Data Model provides a repository for the data describing the data, procedures, and policies shared among the various IISS Application Subsystems.

The Network Transaction Manager provides the operational implementation of the above concepts, the control of the execution of transactions, the control of the flow of messages through the network, the restart and recovery requirements, and the monitoring of performance.

The Network Transaction Manager is invoked to carry out the following functions:

- Dispatch of messages through the IISS Network
- Follow-up on open transactions
- Logging, time stamping of messages

- o Monitoring of system performance (Future)
- o System synchronization
- o Restart of the IISS system
- o Restart and recovery of the databases (Future)
- o Control of Application Subsystems

The Network Transaction Manager controls the execution of application subsystems by processing the Transaction Message queues built on each node. The queues provide the necessary buffering action resulting from the asynchronous nature of the Test Bed Application Subsystem.

2.3.3.2.6 Guaranteed Delivery of Messages

The Network Transaction Manager is also invoked to guarantee the delivery of messages. This service is provided to facilitate the migration of the Test Bed to the Class I environment.

This capability guarantees that messages will be delivered, even if the destination application subsystem is temporarily unavailable. To that effect, the messages are uniquely identified at the level of the IISS by journalizing the message type and Application Subsystem of origin, and by time stamping. Time stamping and journalizing allow for the chronological reconstruction of the transaction input stream. This technique supports the recovery of unavailable nodes or Application Processes.

It should be further noted that the system can guarantee that the message was given to the destination process, and even that the process acknowledged having completed processing. The system cannot, however, guarantee that the receiving process actually did process the message and perform the requested functions, or, for that matter, that the message was even read. The proper processing of messages is totally dependent on the receiving application process and cannot be controlled or guaranteed by the IISS system.

2.3.3.2.7 Standard User Interface

Test Bed User Interface

A Test Bed User Command Language simplifies the task of the user when interacting with the Test Bed. User inputs are through a forms system including menus, formatted data displays, and forms for data entry. The inputs are then formulated into standard messages that are sent to the application processes in the proper host computer.

The User Interface thus provides a unified format to invoke Test Bed System Utilities as well as to support the user dialogues of Application Subsystems specifically designed for the Test Bed.

Virtual Terminal

The proliferation of terminal hardware and the wide disparity in capabilities and features of commercially available terminals create an interfacing problem between IISS and its terminals.

This problem is resolved by defining a specific set of terminal features and protocols which must be supported by the IISS software. This set of features and protocols constitutes the IISS virtual terminal definition.

Specific terminals are then mapped against the IISS virtual terminal software by specific software modules written for each type of real terminal interfaced to IISS. This approach is consistent with the layered software philosophy of IISS since it permits the interfacing of a wide variety of terminals without changes to the IISS application programs.

System-Wide Forms and Protocols

User forms and user protocols are defined at the IISS system level. These forms and protocols define the manner in which the IISS user interfaces with IISS. The forms are data structures with attributes which are enforced by the forms package. Mandatory fields, alphanumeric fields, and numeric only fields are examples of the attributes which are enforced by the forms package.

The forms and protocols are defined by data stored in the CDM, and as such are very flexible and extensible.

SECTION 3

REQUIREMENTS

3.1 System Definition

3.1.1 Network Transaction Manager Configuration Item

3.1.1.1 Network Transaction Manager Mission Statement

The Test Bed is a distributed computer system made up of cooperating processes. As an example of cooperating processes, consider the processing of a distributed query. In this example, processes resident on several hosts are cooperating in the retrieval, selection, unit and format conversion of data resident on several databases.

The Network Transaction Manager performs the coordination, communication and housekeeping functions required to integrate the Application Processes and System Services resident on the various hosts into a cohesive system. The management of mail boxes, message queues, application processes, to name a few, are examples of System Services provided by the Test Bed.

3.1.1.2 Network Transaction Manager Functional Areas

The configuration tree of the NTM is shown in Figure 3-1. Figure 3-1 shows three major functional areas:

- Manage Message
- Manage Processes
- Maintain Operability

The Test Bed is a message-driven system. The Test Bed uses messages to request the execution and termination of Application Processes and system services. Messages are also used to exchange data between Application Processes and System Services.

The Manage Message functional area of the Network Transaction Manager is responsible for the management of these messages.

The Application Processes resident on the Test Bed need to

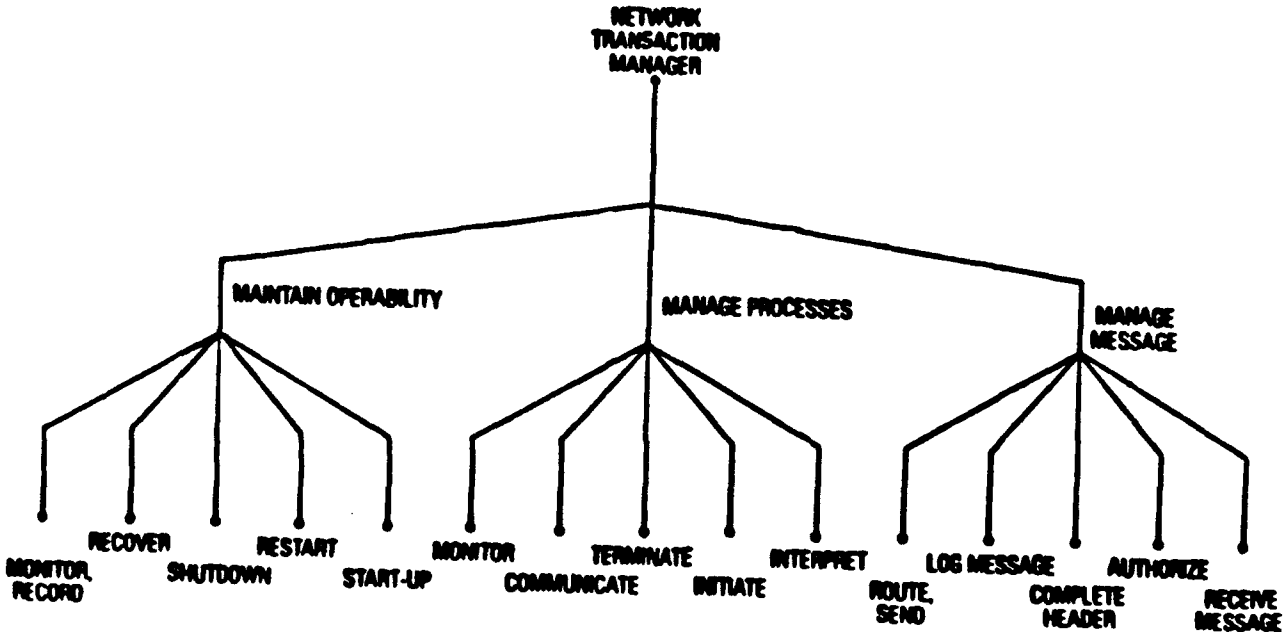


Figure 3-1. NTM Configuration Tree

be managed according to the processing needs of the environment. Managing Applications include functions such as loading, initiation, and terminating processes. These functions are included in the Manage Processes functional area of the Network Transaction Manager.

The Maintain Operability functional area of the NTM contains the functions required to create, modify, maintain the processing environment of the Test Bed. The creation, modification and maintenance of the processing environment calls for processing capabilities to support startup, shutdown, recovery and monitoring of the Test Bed.

3.1.1.3 Network Transaction Manager Operational Scenario

The Network Transaction Manager Operational Scenarios presented here are introduced for the sole purpose of supporting the identification of the functional specifications to be met by the Network Transaction Manager. These scenarios are not meant to imply a specific implementation of these functional specifications. Consequently, the final design of the Network Transaction Manager may implement scenarios which differ from the scenarios shown in this subsection.

3.1.1.3.1 NTM Environment

To describe the environment of the NTM, it is necessary to introduce the concept of an AP Cluster.

On an intuitive basis, an AP Cluster consists of processes related to one application as viewed by the user. Examples of Test Bed AP Clusters are the MCMM, the MRP, and the IDSS AP Clusters. The formal definition of the AP Cluster concept is given in Section 2. In the Test Bed, every Application Process is uniquely addressable.

Communications and Control with each AP Cluster is accomplished via Network Transaction Manager. In the Test Bed environment, the AP Clusters may reside on different hosts, thus the various instances of the NTM may present some differences to reflect the different operating system environments in which they happen to operate.

3.1.1.3.2 NTM Architecture

The AP Cluster concept introduced above offers the following significant advantages:

1. The Application Processes which most frequently access a database are grouped on the AP Cluster containing the database supporting these processes. This grouping minimizes the frequency of off-host data accesses.

2. An instance of the NTM is associated with each AP Cluster to simplify the message traffic. All communications related to one AP Cluster are routed through that AP Cluster NTM. This approach streamlines the type of communications which must be supported. This concept is graphically illustrated on Figure 3-2 and on Figure 3-3. The first figure shows a system which allows the Application Processes to Communicate directly with one another. The second figure shows a system which routes all communications through specific communication programs. This second approach is retained for the Test Bed, and is illustrated on Figure 3-4. This last figure shows the impact of the multi host environment of the Test Bed on the Architecture of the Communication and NTM Subsystems.

More specifically, Figure 3-4 shows that not only each AP Cluster has its own NTM, but that in addition, each host owns an additional AP Cluster, with its own NTM, to communicate with all other hosts. The AP Cluster used to communicate with other hosts is called the COMM AP Cluster.

The above concepts and definitions lead to Figure 3-5 showing a conceptualization of the NTM architecture. The key features of this figure are:

1. The User Interface AP Cluster, and Virtual Terminal Interface
2. The CDM Processor AP Cluster
3. The NTM operator AP Cluster
4. The NTM supports communications between any two AP Clusters shown on the diagram

This figure is, however, a conceptualization. The cross

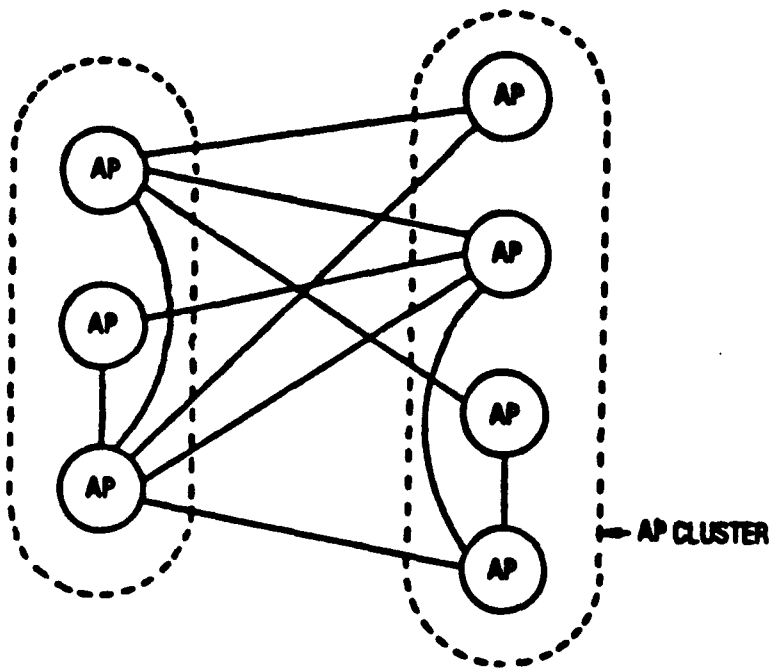


Figure 3-2. Cluster APs

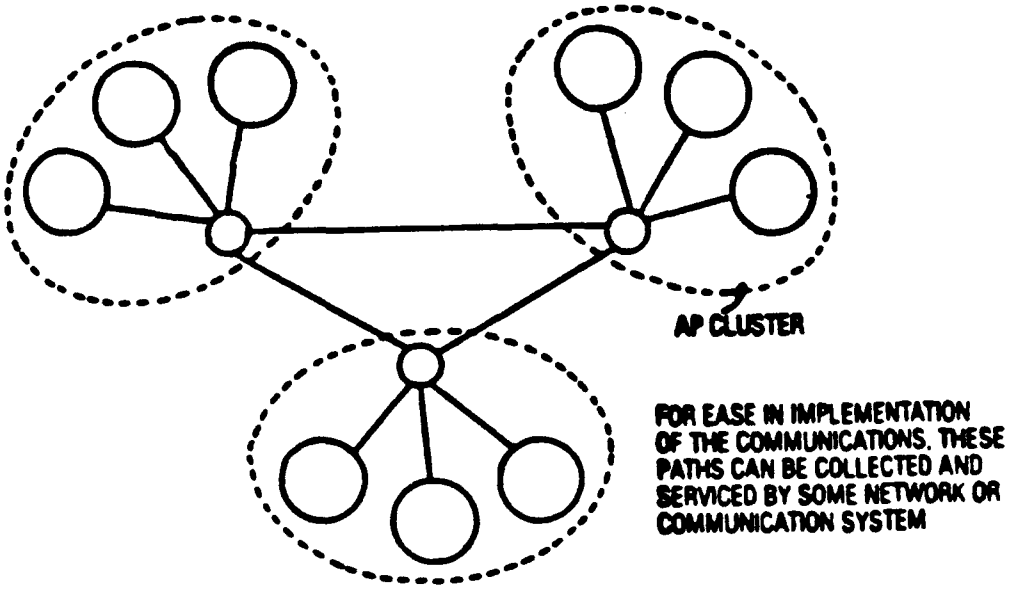


Figure 3-3. Ease in Implementation

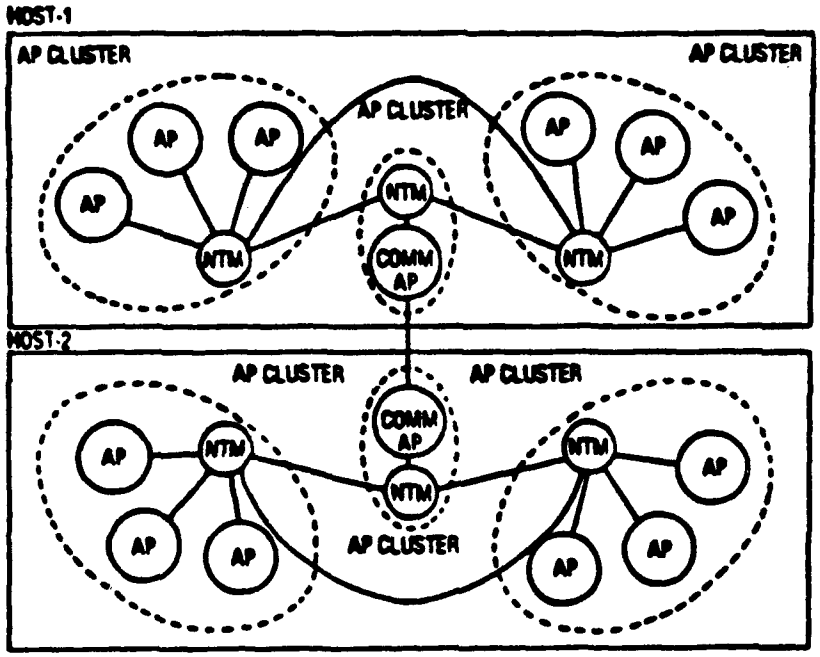


Figure 3-4. Host 1

bar matrix arrangement shown has no implication on the architecture of the Test Bed. The Test Bed NTM architecture is more faithfully represented on Figure 3-4. The Communication Subsystem described in Section 3.1.5 supports the architecture of Figure 3-4.

The interaction of the NTM and its environment is depicted on the IDEFO diagram shown in Figure 3-6. With respect to this diagram, the following observations are made:

1. All messages, whether Inter or Intra AP Cluster messages, are received and handled by the NTM.

2. The NTM places requests to the host operating systems to initiate and/or abort specific Application Processes.

3. The NTM receives status information from the host operating system.

4. The NTM delivers messages to the Application Processes. These messages may be viewed as input data to the Application Process.

5. The NTM receives messages from the Application Processes. These messages may be viewed as output data from the Application Processes.

6. The NTM transmits all messages to their destinations, whether inter or intra off AP Clusters.

Figure 3-7 and 3-8 are variations on the theme shown in Figure 3-9. Figures 3-7 and 3-8 are drawn for the COMM (inter host) Application Process and for the User Interface. From an NTM point of view, Figures 3-6, 3-7, and 3-8 are identical. From an Application point of view, Figures 3-7 and 3-8 show the additional input/output requirements specific to the User Interface and COMM AP Clusters.

3.1.1.3.3 NTM Functional Description

Figure 3-9 is the Top level of an IDEFO diagram portraying the functionality of the NTM. This diagram shows the three major functional areas identified earlier and their interaction.

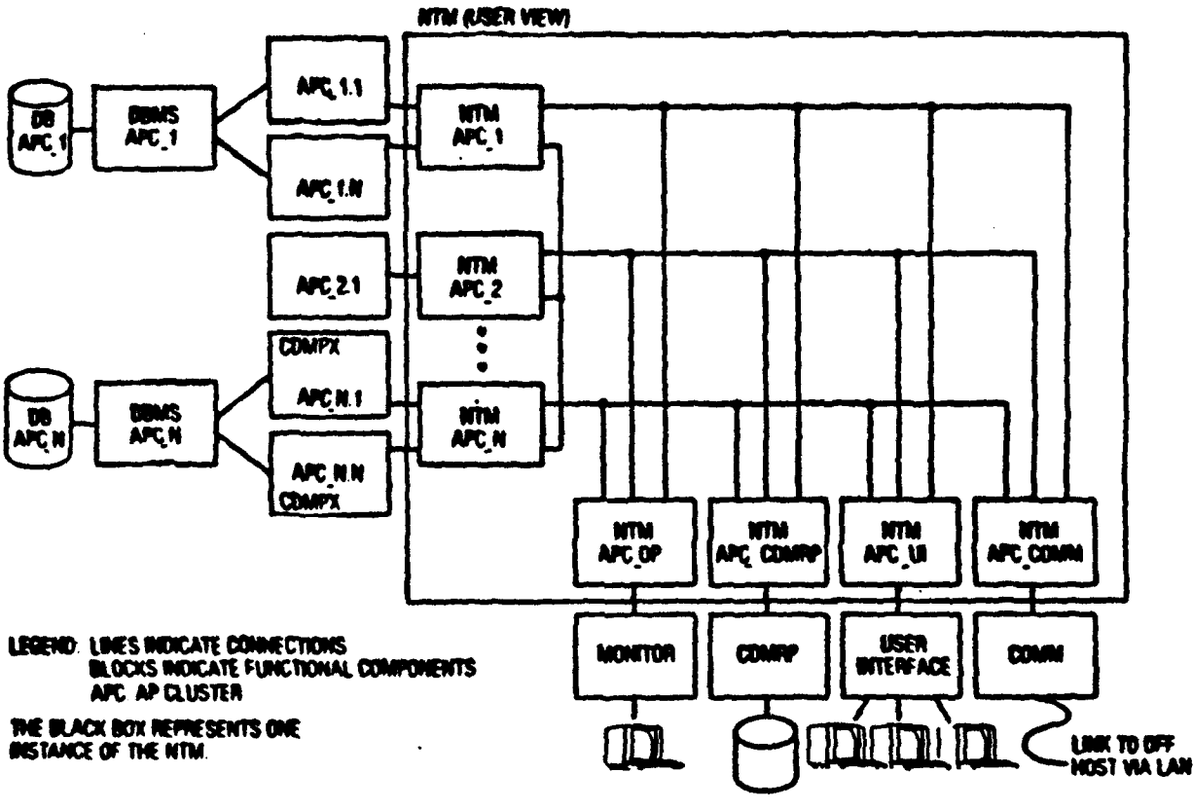


Figure 3-5. IISS Architecture - Conceptual Model

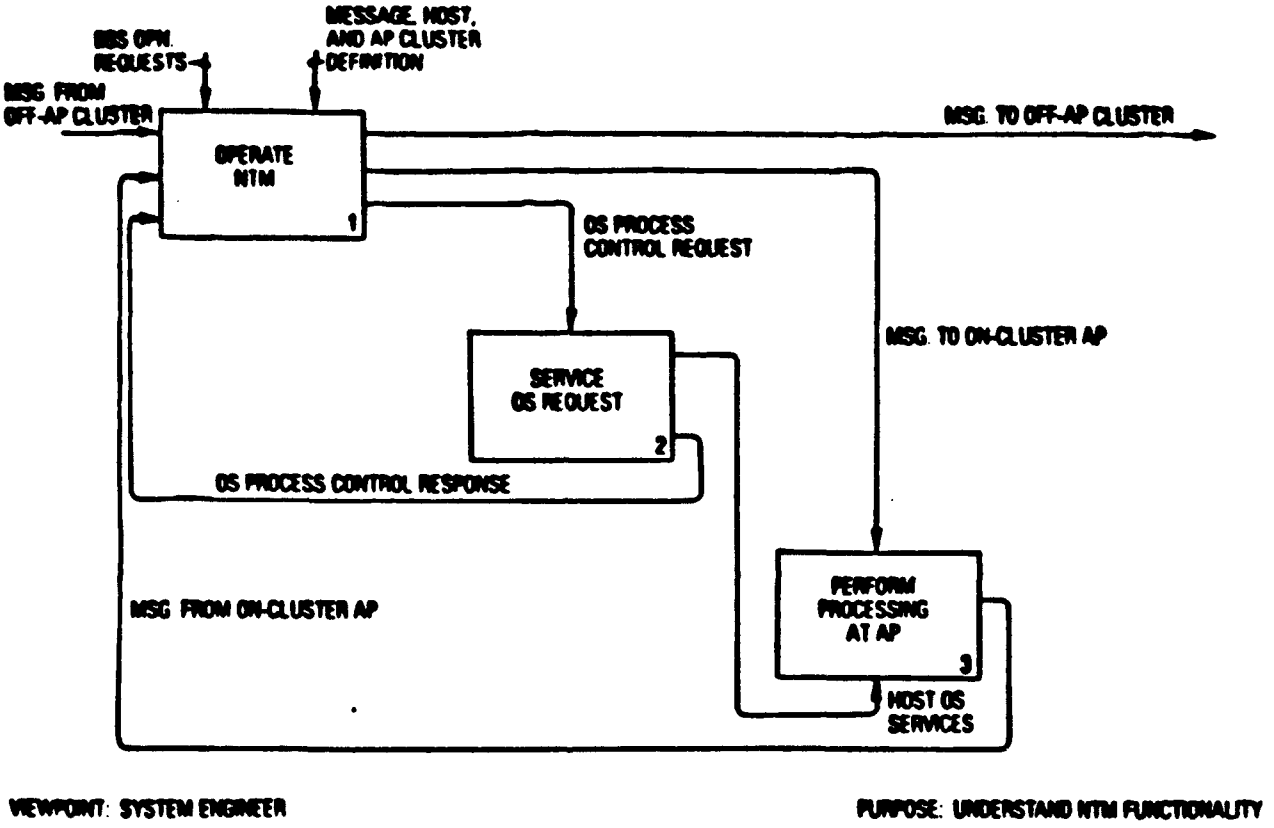


Figure 3-6. NTM Environment on One AP Cluster

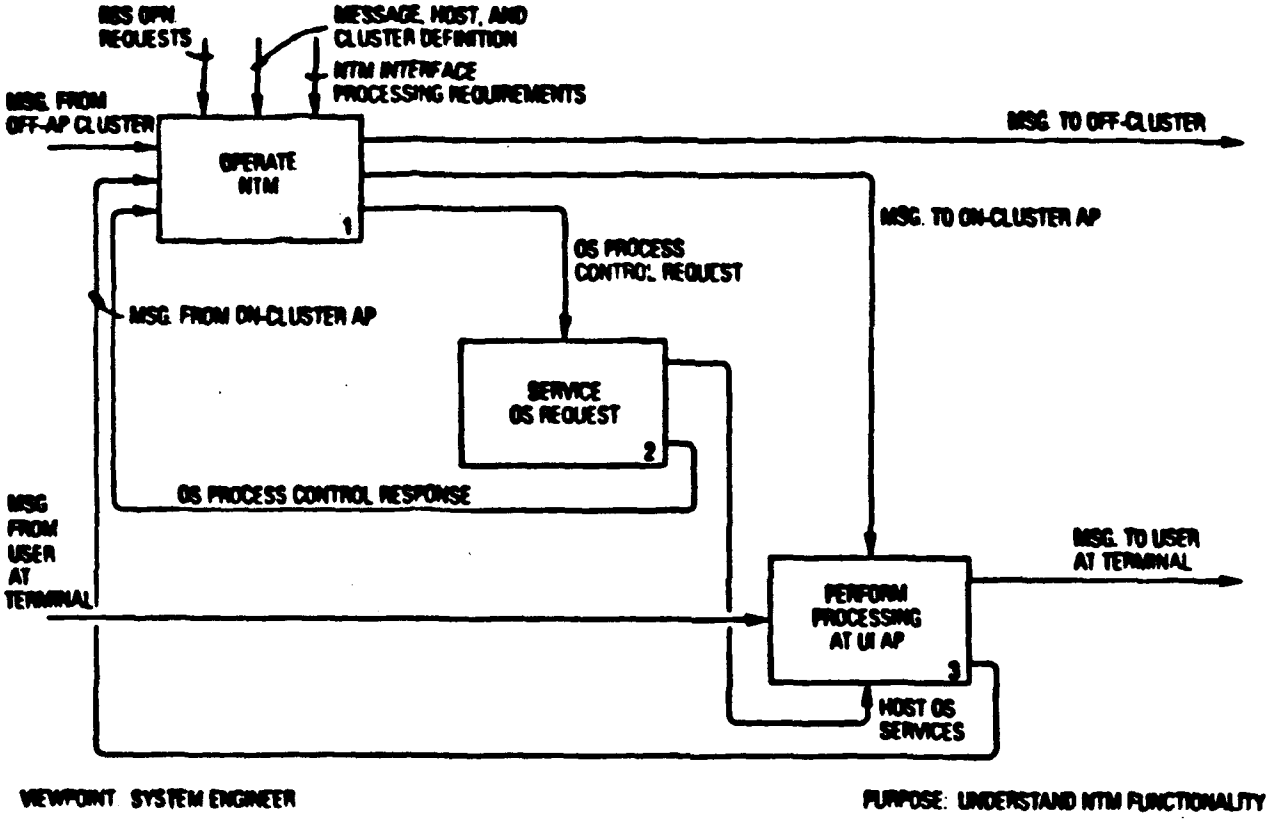


Figure 3-7. NTM Environment on UI WS

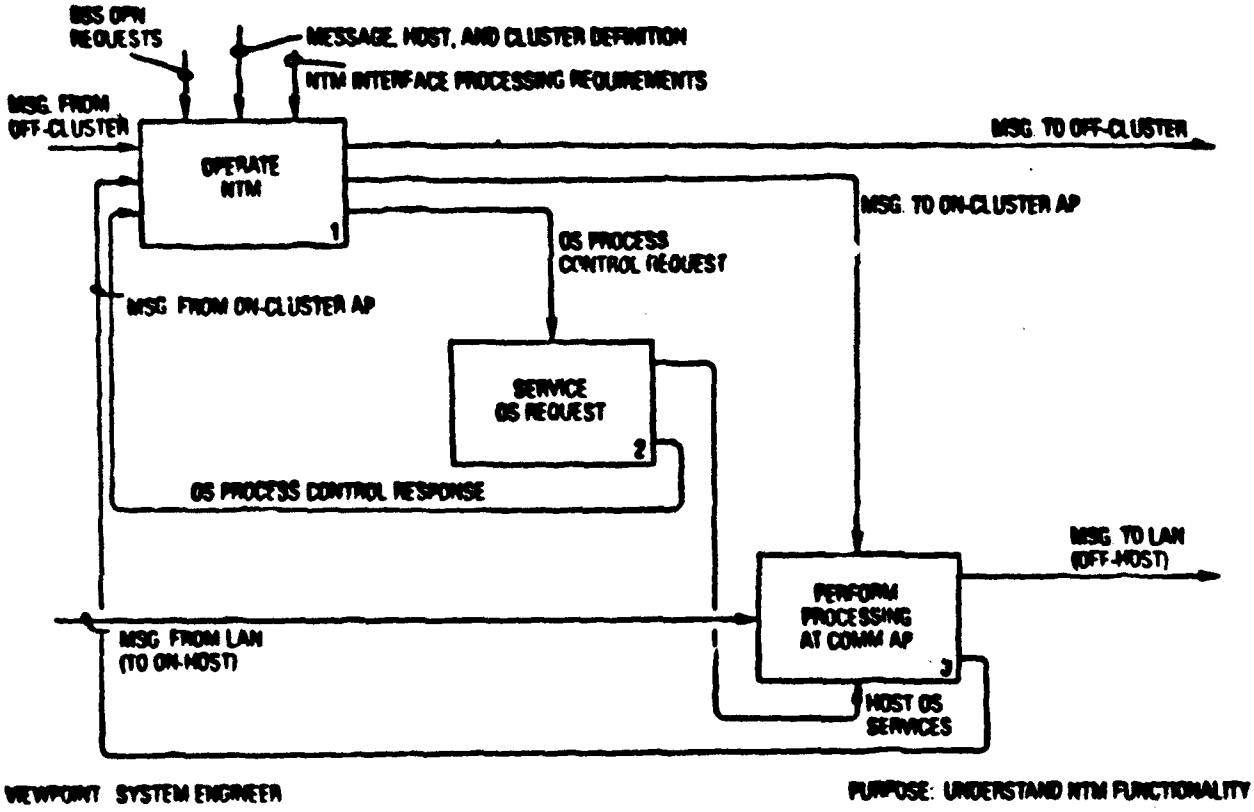


Figure 3-8. NTM Environment on COMM WS

Figure 3-10 details the MANAGE Message functional area. This figure shows that messages, once received by the NTM, are checked for authorization, logged and routed. A message header is appended by the NTM to facilitate the routing and interpretation of the message. At this level of description, the Route and Send Message function includes the routing and sending of a message to an off AP Cluster, on AP Cluster, or maintain operability Application Processes.

Figure 3-11 details the Manage Processes functional area. This figure shows that messages are interpreted and either used to control Application Processes or to communicate with Application Processes.

The initiate Application Process function includes the scheduling of such initiation. The scheduling information is contained in the message initiating the process.

The Abnormally Terminate Application Process functions include the termination of an Application Process and the housekeeping actvities related to such termination. In the Test Bed, termination of an Application Process may be accompanied by the termination of the query processors and data aggregators initiated by that Application Process. Data aggregators, and query processors are described in Section 3.1.2. The information required to keep track of active query processors and data aggregators is maintained by the NTM. For example, a chained list links the various processes (query processors, data aggregators, transformers) to the Application Process which required these services.

Figure 3-12 details the Communicate with Application Process function. Messages are accepted from and delivered to Application Processes. Messages are also paired on an Application Process basis. This supports the detection of unanswered messages and the initiation of corrective action, by the receiver, in the event of a time out. The NTM message pairing capabilities allow the detection of end to end problems (such as the failure of one Application Process to return an expected reply) as well as to detect host and local area network failures. The NTM on the originating AP Cluster detects time out conditions and reports the time out to the process which originated the message.

The detection of malfunctions in the LAN is performed by the COMM subsystems.

Figure 3-13 details the Maintain Operability functional area. This functional area is shown to breakdown into the following key functions; and relates to the IISS System Software:

- o Start up (IISS on host)
- o Restart
- o Shutdown (IISS on host)

- o Test Bed Recovery
- o Monitor and Record Resource Usage

The Maintain Operability functions are also shown to communicate with the IISS operator. The operator acts as the controller of the Maintain Operability of the IISS System Software, and as such is the recipient of the maintain operability status messages issued by the System Software. These messages may also be stored in a file for archive and analysis purposes. Status messages issued by user supplied Application Processes are routed to the user by the IISS software.

The availability of Application Processes and IISS status and error information is dependent upon the services provided by the host operating systems. Consequently the extent and availability of such information varies from operating system to operating system. Status and error information is gathered via a combination of IISS and host operating utilities according to host-dependent procedures.

The Recovery function addresses the recovery of the Test Bed system and its databases. The recovery of the databases themselves is achieved via the roll back and journalization facilities provided by the various database managers. One of the NTM's role is to ensure the synchronization of the recovery operations and to initiate the recovery once the system is at a quiet point.

3.1.1.3.4 NTM/IISS Start Up Scenario

The NTM/IISS start up scenario listed here is given for the 6201M implementation of the Test Bed. The current design calls for the start up of the Test Bed to be initiated from the host consoles. A start up command must be typed on each of the host consoles. Subsequently, the Test Bed may allow for a centralized start up of the system (Figure 3-14).

The start up of the Test Bed Software is thus as follows:

1. On the VAX:

   The IISS operator initiates the "START IISS" procedure file under control of VAX [illegible].

   This procedure starts up the following AP Clusters:

   a. Common Data Model Request Processor

   b. User Interface

   c. Any NTM (for example, IDSS)

   d. The COMM Work Stations with the VAX/Honeywell and VAX/IBM communication services

Each AP Cluster initiates its own request for CDM data to the CDM request processors. On completing its prescribed start up steps, each AP Cluster reports its status to the console used to initiate the start up. The above step then initializes the NTM tables, the VTI configuration tables and the User Interface local form storage. Any information required by the user work station NTM is down loaded from the CDM.

2. On the Honeywell:

The IISS Operator initiates the "START IISS" procedure file under control of GCOS MOD400 from a Honeywell console.

This GCOS MOD400 procedure file starts up the following AP Clusters:

a. The COMM Work Station with the Honeywell/VAX and Honeywell/IBM communication services

b. Any User work station NTM (for example, MCMM)

Each AP Cluster initiates its own request for CDM data to the CDM processor. On completing its prescribed start up step, each AP Cluster reports its status to the console of the Honeywell. The above steps initialize the NTM tables. The AP Clusters are started

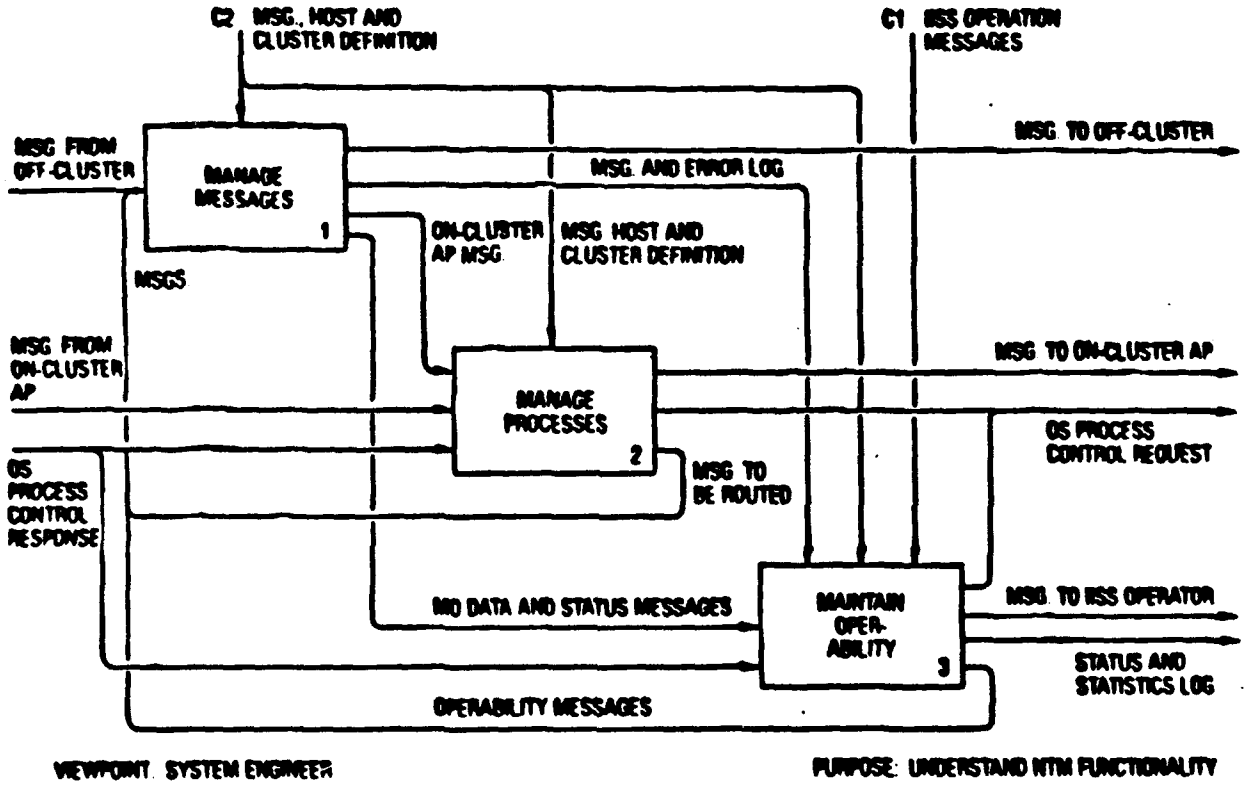


Figure 3-9. Operate Network Transaction Manager Functions

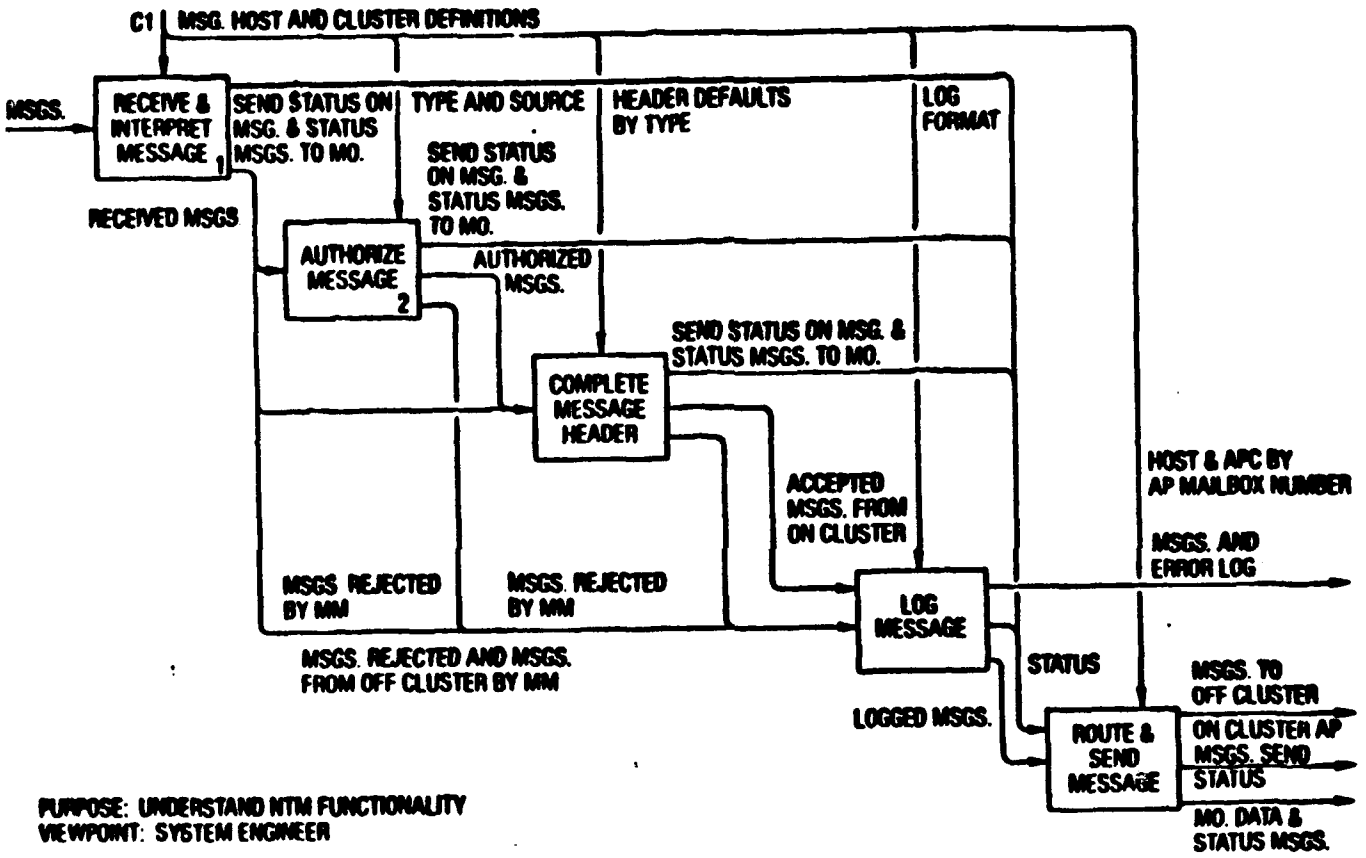


Figure 3-10. Manage Messages Functions

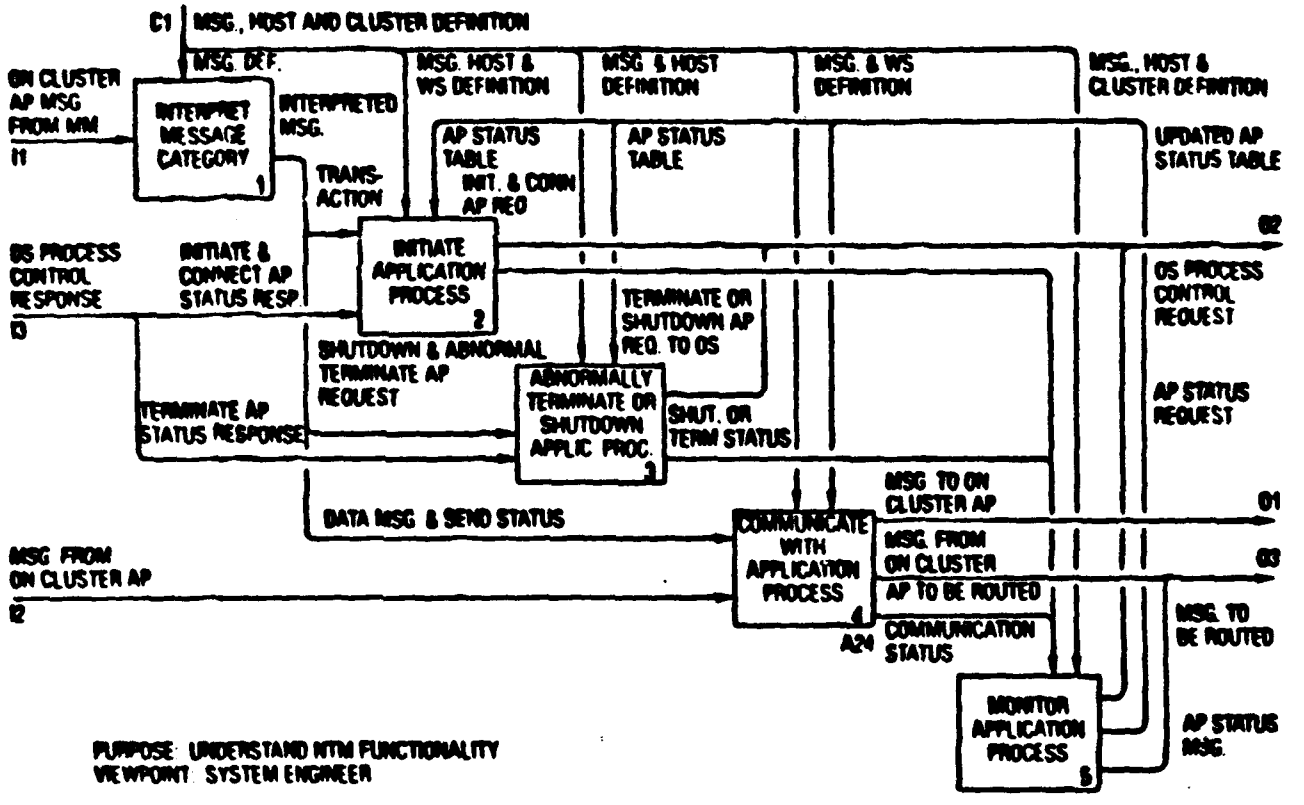


Figure 3-11. Manage Processes Functions

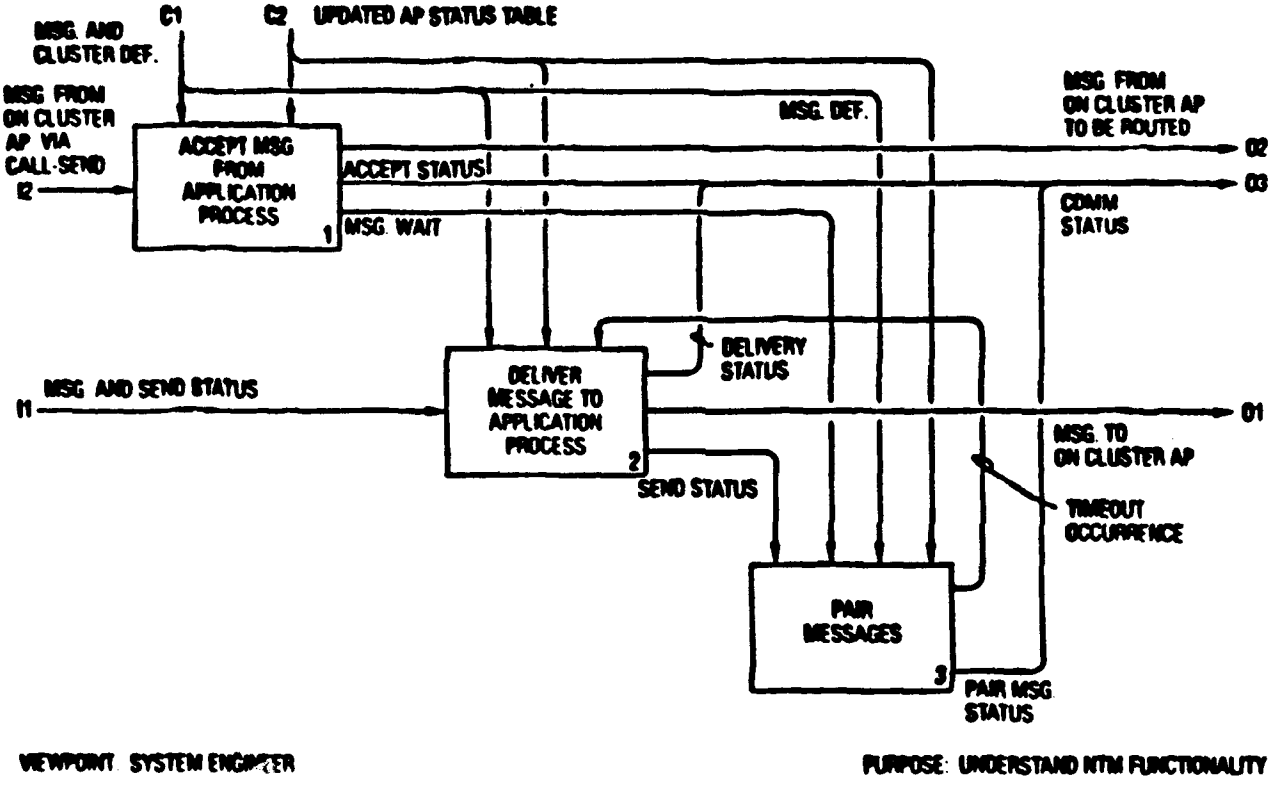


Figure 3-12. Maintain Operability Functions

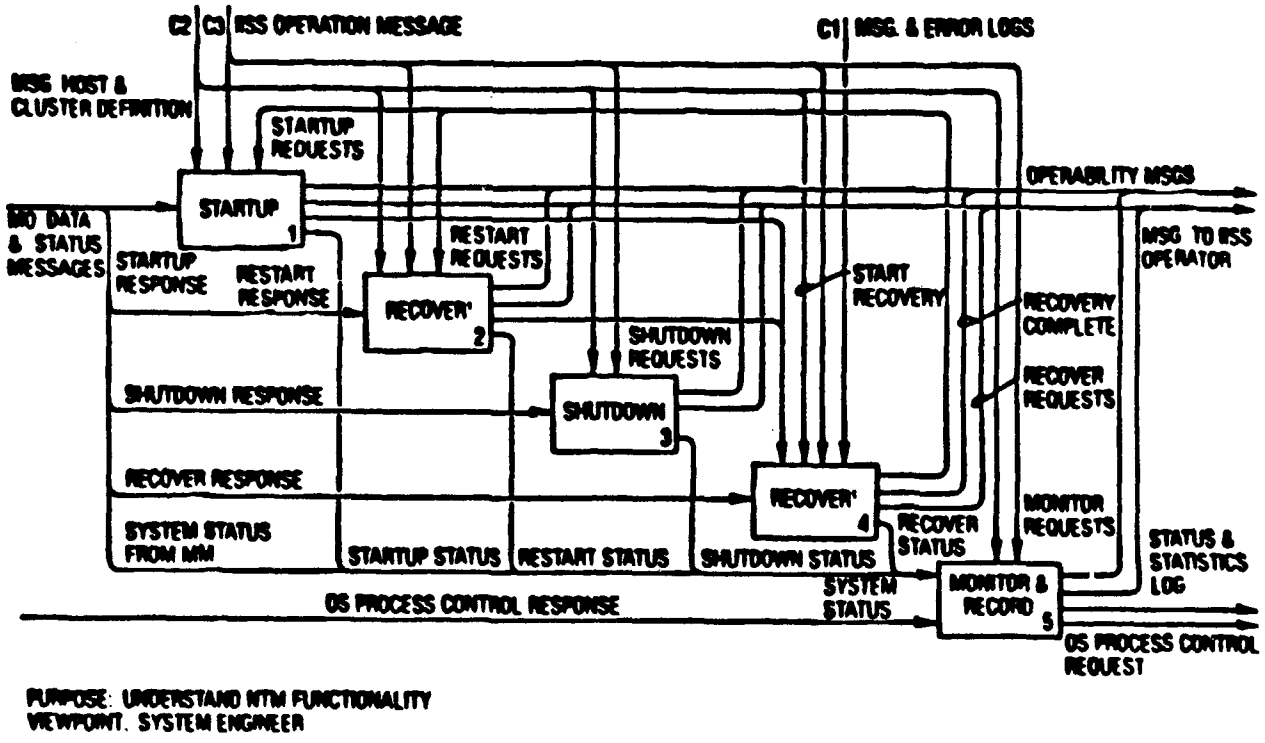


Figure 3-13. Communicate with Application Processes

in the sequence listed above. The Honeywell/VAX COMM AP Cluster must be operational for the Honeywell start up to proceed. As each AP Cluster becomes operational, it notifies VAX IISS console. In the event of difficulties, the error messages generated during start up are displayed on the Honeywell console.

3. On the IBM 3033:
The start up procedure described above is repeated on the IBM 3033. The procedure is initiated under control of MVS from an IBM console.

The AP Clusters brought up on the IBM 3033 are:

a. The COMM Work Station with the Honeywell/VAX and Honeywell/IBM communication services

b. Any User Work Station NTM (for example, MRP)

3.1.1.3.5 NTM/IISS Shut Down Scenario

The shut down of the IISS system can be initiated from any IISS terminals by an IISS operator with the proper authorization. The shut down is graceful and complete. To that effect, the following capabilities are provided:

1. Warning messages are sent to all IISS terminals. These messages are repeated at reasonably spaced time intervals.

2. Further IISS logins are disabled when a system shut down is in process.

3. Processes are allowed to run toward completion for a reasonable grace period (for example, 15 minutes). This includes the query processors and data aggregators initiated by the Application Processes.

4. Status of queues are saved on each processor. (Not implemented, nor possibly desireable in near future)

5. Processes, data aggregators, query processors still running at the end of the grace period are killed under IISS operator control.

6. The shut down process proceeds first with the termination of User Application Processes, and second with the termination of Test Bed Services.

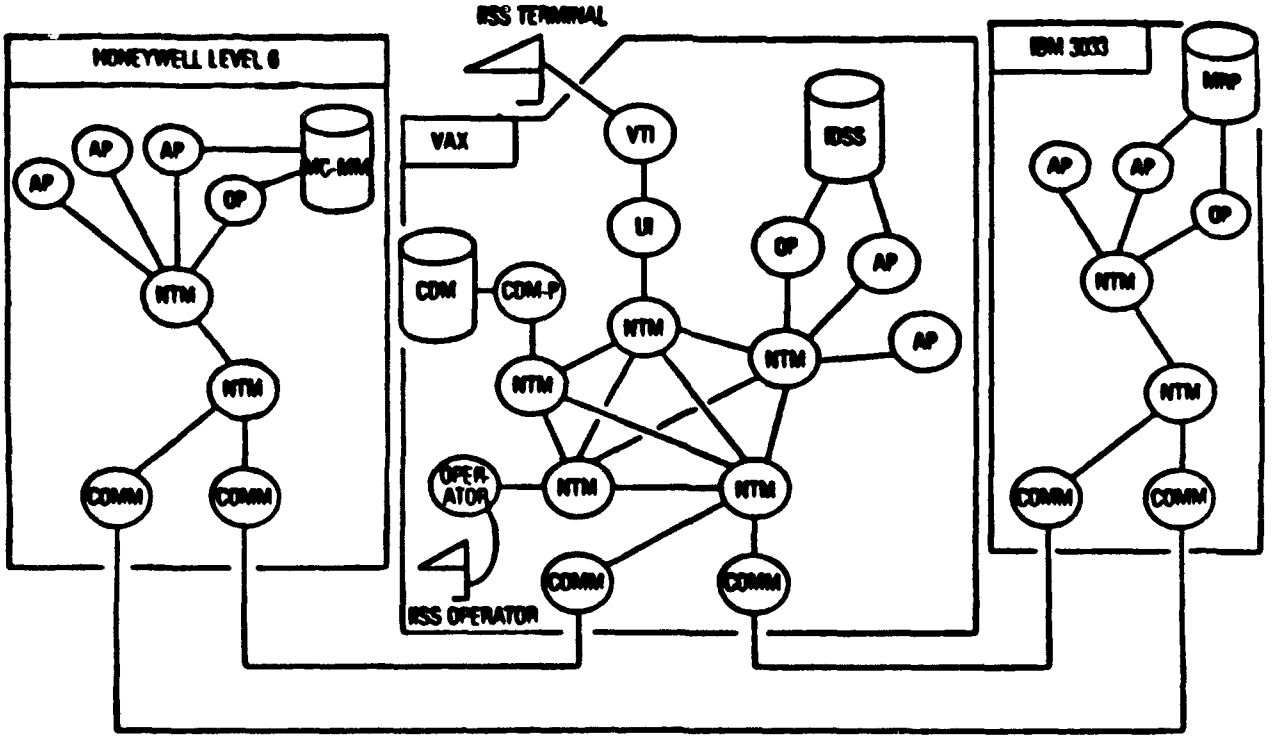


Figure 3-14. NTM Architecture Test Bed Overview

7. A message is sent to the host console upon completion of the shut down.

8. The shut down of the IISS system can be initiated on a host by host basis from the host consoles. This capability serves as a back-up in the event of LAN and communication malfunctions.

3.1.1.3.6 NTM/IISS Host Shutdown Scenario

The shut down of a selected host of the Test Bed proceeds with the same logic that followed for the shutdown of the entire Test Bed. In fact, the shutdown of the entire Test Bed is viewed as the shutdown of each host, per the above scenario. The shutdown procedures are described in the operator's manual.

The following steps are carried out during the shutdown of the Test Bed software:

1. The status of the message queues are checkpointed (Not implemented)

2. The application processes still running at the expiration of the shutdown count down period are killed under supervision of the IISS operator.

3. The Test Bed System Services (NTM, OP AP Cluster, COMM AP Cluster, etc.) are terminated.

The above steps are repeated on each host.

3.1.1.3.7 NTM/IISS Host Start Up Scenario

By the same reasoning, the start up of a selected host is performed as explained in the section entitled "NTM/IISS Start Up Scenario". The scenario is, however, limited to one host. The start up of any host other than the CDM does not progress past the request for CDM data if the CDM host is not already up.

3.1.1.3.8 IISS Recovery Scenario

The IISS Recovery Scenario is not addressed in the early release of the Test Bed. Definition of the functionality and implementation of the recovery mechanism is an enhancement to the Test Bed.

3.1.1.3.9 Application Process Scheduling

The NTM initiates Application Processes at the request of the Test Bed user or at the request of Application Processes already running. The initiation of the data aggregators and query processors are examples of this second eventuality. The initiation of the Application Process is only performed for authorized requests. The initiation is prioritized and proceeds on a FIFO basis at equal priority. The execution of some Application Processes may be linked to wall clock time, time delay, or may be conditional to some event.

The information required to control the initiation of Application Processes (priority, schedule, condition) is carried by the Application Process request message.

Consider Figure 3-15. The scheduling of the Application Process initiation is controlled by the scheduler. The scheduler keeps an on-going watch of the Application Process queues containing the requests for start up time and may be deactivated when the system is executing a Quiet Point Command or when the system is about to be shut down. The scheduler may be reactivated upon Command to resume processing of Test Bed Application Processes.

The NTM supports multiple instances of a given Application Process. Multiple instances of Application Processes may be created in response to requests from multiple users.

The NTM automatically initiates additional instances of application processes which have been granted the privilege to have multiple instances. This privilege is declared in the CDM. The data describing the Application Processes defines the maximum number of instances which can be running simultaneously. The CDM Administrator authorizes the duplication of selected Application Processes.

3.1.1.3.10 Maintain Directory of Active Application Processes

The NTM receives status information from the operating system of the local hosts. This information is used to create and to maintain a list of active application processes on the AP Cluster. This list is used to clean up an AP Cluster whenever an Application Process is aborted or terminates.

3.1.1.3.11 Maintain Directory of Offspring Application Processes

The Test Bed Application Processes generate offspring application processes whenever they perform a distributed query or update. In the distributed query environment, these offsprings include data aggregators and query processors. The NTM maintains a list of the data aggregators and query processors which have been requested by the query scheduler. This list is used to abort the data aggregators and query processors in the event that the parent Application Process terminates or is aborted. The list identifies the offspring application process, the target AP Clusters and the parent Application Process.

3.1.1.3.12 Application Process Termination

For integrated Application Processes, the normal and abnormal termination of an Application Process is known to the NTM. The NTM receives status information from the local host operating system. The NTM provides the following services upon the termination of an Application Process:

1. Normal Termination
   The name of the Application Process that terminates is removed from the AP Cluster active application process list. Usage statistics for the Application Process are recorded.

2. Abnormal Termination
   In the event of abnormal termination of an Application cancelling the active offsprings of that Application Process which may still be active or queued up for execution. The NTM performs this task by taking advantage of the offspring application process list to notify the AP Clusters which may be processing or about to process the aborted Application Process offspring application processes. The NTM's on these AP Process, the NTM assumes the responsibility for Clusters make use of the active application process directory to abort active offsprings (data aggregators, query processors) or to remove these offsprings from the spawning request queues.

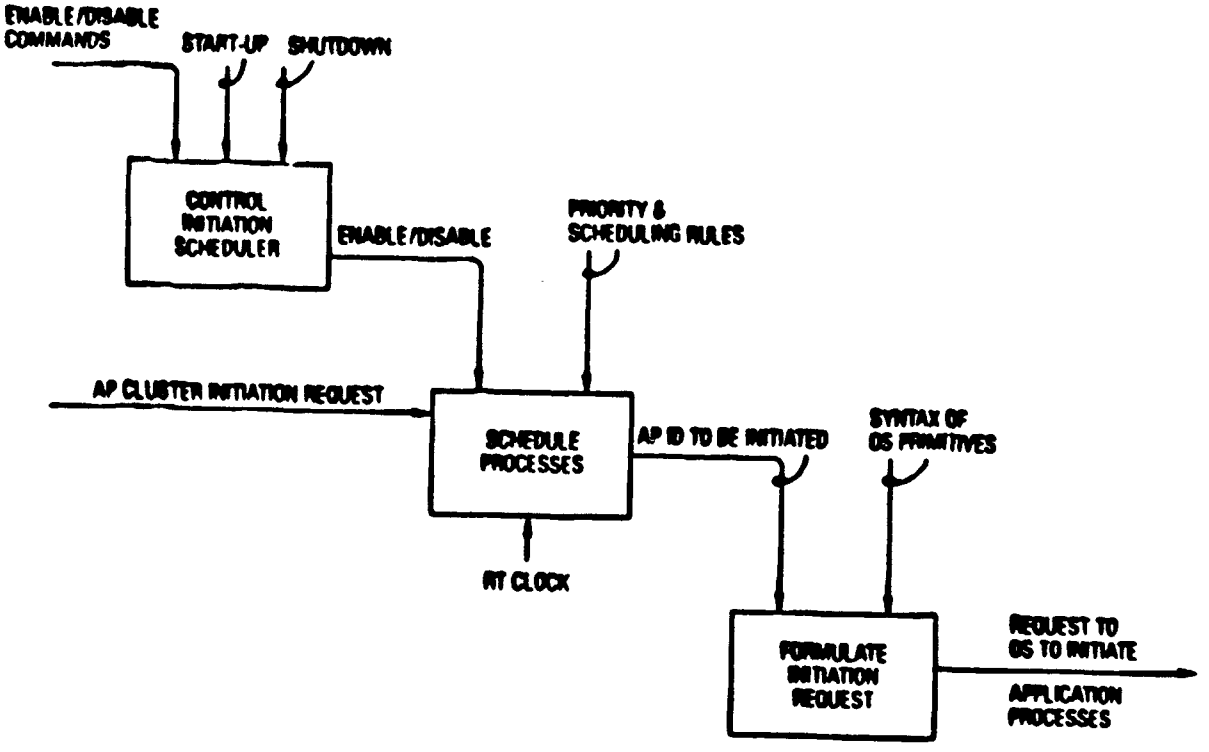


Figure 3-15. Spawning Scheduler

The NTM's of the offspring nodes report the completion of the clean up operation to the NTM of the cancelled Application Process AP Cluster. The abnormal termination process continues with the steps described under the normal termination scenario.

3. Housekeeping
The IISS operator may invoke the abnormal termination process described above to free the system from Application Processes and Offspring Processes which have not been cancelled following the normal termination of an Application Process. This eventuality may occur with improperly written Application Processes or in the event of hardware/software failures.

3.1.1.3.13 Exception Handling

The NTM is actively involved in the handling of exception calls made by an AP Cluster Application Process to the local host operating system. For the Test Bed to act as a cohesive and robust system, the following functions need to be performed:

1. The occurrence of an exception call in an Application Process must be reported to the initiator (person or software) of that Application Process. The report must include sufficient information for the initiator to decide on a course of action.

2. The report must be in a standard format so that the interpretation of the error condition is not dependent upon the host on which it occurred.

3. Exception calls must be logged for further analysis and trouble shooting.

The above functions are implemented on the Test Bed by allowing the NTM to intercept the exception calls to the operating system. Thus the NTM can notify the initiator of the Application Process of the type of error which occurred. The local error codes are first converted to the Test Bed (Neutral) error codes. The Test Bed error codes and the host error code mapping information are contained in the CDM. Exception calls originating in the Test Bed system service software are reported to the IISS operator. Such calls could indicate serious problems with the IISS system software.

3.1.1.3.14 Communication with Application Processes

The NTM communicates with Application Processes via mail boxes. These mailboxes are named, created and operated via the primitives described in Section 3.1.5 (Communication Subsystem). The primitives described in Section 3.1.5 support multiple instances of the same Application Process.

3.1.1.3.15 Message Authorization

The NTM is the funnel through which all Test Bed messages flow. This is the natural place for message authorization enforcement. In the Test Bed data, access privileges are granted to Application Process via the schema granted to it, and security is enforced by controlling the access to the Application Processes themselves.

Consider Figure 3-16. This figure shows the various data access mechanisms used in the Test Bed.

Application Process Number One (APP1) has been granted access to databases 1 and 2, via the external schema granted to it by the CDM Administrator. The precompiler has generated the query processors QP11 and QP12. APP1 can thus invoke these two query processors. If User 1 has been granted access rights to APP1, it thus can query databases 1 and 2.

Assume, for the sake of discussion, that User 2 has not been granted access rights to databases 1 and 2, and hence does not have the schema information required to precompile the necessary query processors QP21 and QP22 required to access databases 1 and 2 from an Application Process to which he has access privileges.

User 2 can thus only gain access to database 1 and 2 by invoking directly or indirectly an Application Process such as APP1. The direct accessing of QP11 and QP12 is not deemed feasible since these two precompiled query processors are tied (message destination, control) to APP1. In the Test Bed, each Application Process and each user is granted through its legal user role authority to send and authority to receive messages from and to other users. This scheme is used to prevent the two access paths shown in dotted lines on Figure 3-16. The NTM is charged with enforcing the transmit and receive rights of each application, by comparing the source and destination information contained in the message with the authority to send and authority to receive granted to each user. This information is defined in the CDM by the CDM Administrator. All communications between Application Processes are routed via the NTM.

The above scheme can, however, be easily defeated. If APP1 creates a private copy of the data, no control on the access to this private copy can be exercised through the Test Bed system. Administrative procedures or other automated procedures (prepass of the code) can be used to disallow this possibility.

3.1.1.3.16 Message Header

The NTM appends a header on all messages it receives from the Application Processes. This approach allows for the layering of protocols.

For example, the header contains the message category, source, destination information and various flags (logging, statistics, etc.) controlling the kind of message handling services to be invoked.

The NTM strips the header before handing the message over to its destination.

3.1.1.3.17 Message Logging

The NTM serializes and logs messages. The logs are kept on the host where the message originates. The message serial number is the concatenation of the AP Cluster identification number and of the serial number sequentially assigned to the message in the AP Cluster where it originates. Messages are time stamped as necessary. The time stamps are obtained from the host system clock and specify date, time of day.

3.1.1.3.18 Message Routing

The NTM routes the messages to their final destination. To that effect, the NTM appends a physical address to the message. The message physical address is derived from the logical address supplied by the user and from the logical to physical maps provided by the CDM. Undefined logical addresses are flagged to the user.

The user supplies a logical address which uniquely identifies the Application Process at the system level.

The system, however, through the mapping tables defined in the CDM will expand this logical address to a physical address.

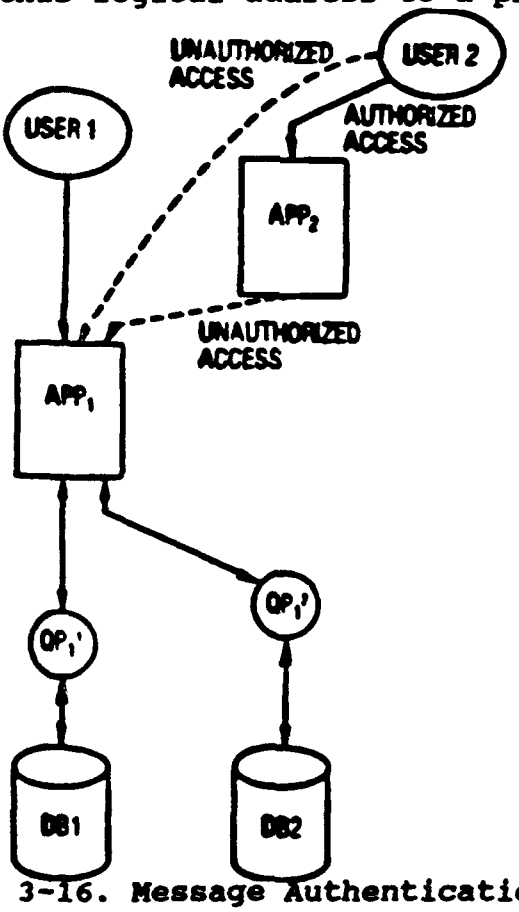


Figure 3-16. Message Authentication

The NTM, through the mail box naming rules, establishes the proper end to end correspondence which exists at any point in time between the various instances of cooperating Application Processes.

The above concepts are reflected in Figures 3-17 and 3-18.

3.1.1.3.19 Message Pairing

The NTM pairs messages. That is, it keeps track of the request and response message pairs flowing through the system. A time out allows the detection of unanswered messages. The Application Processes which initiated the unanswered request message of a message pair is notified of the time out. It initiates any recovery or contingency action it may support.

3.1.1.3.20 Error Logging

The NTM logs errors in an error file. A file is maintained on each host and can be accessed centrally by the IISS operator. The file can be cleared under control of a procedure activated by the operator. The files are maintained locally to increase the likelihood of capturing error messages. (See Section 3.1.1.3.3)

3.1.1.3.21 Message Integrity Checking

Message integrity checking enhances the reliability and security of the Test Bed system. Message header integrity checks are performed by the NTM. Message data integrity checks are performed by the destination Application Subsystem. These checks are supported by Common Data or by Private Data known to the subsystem. The NTM checks the data targeted to its own use.

1. Integrity Checking of the Header

   This type of integrity check is performed by the NTM of the AP Cluster originating the message, and is performed in inter host as well as intra host communications.

   For example, the header integrity check includes the following checks:

   a. Edit of each field

   b. Reasonableness of the user-defined fields

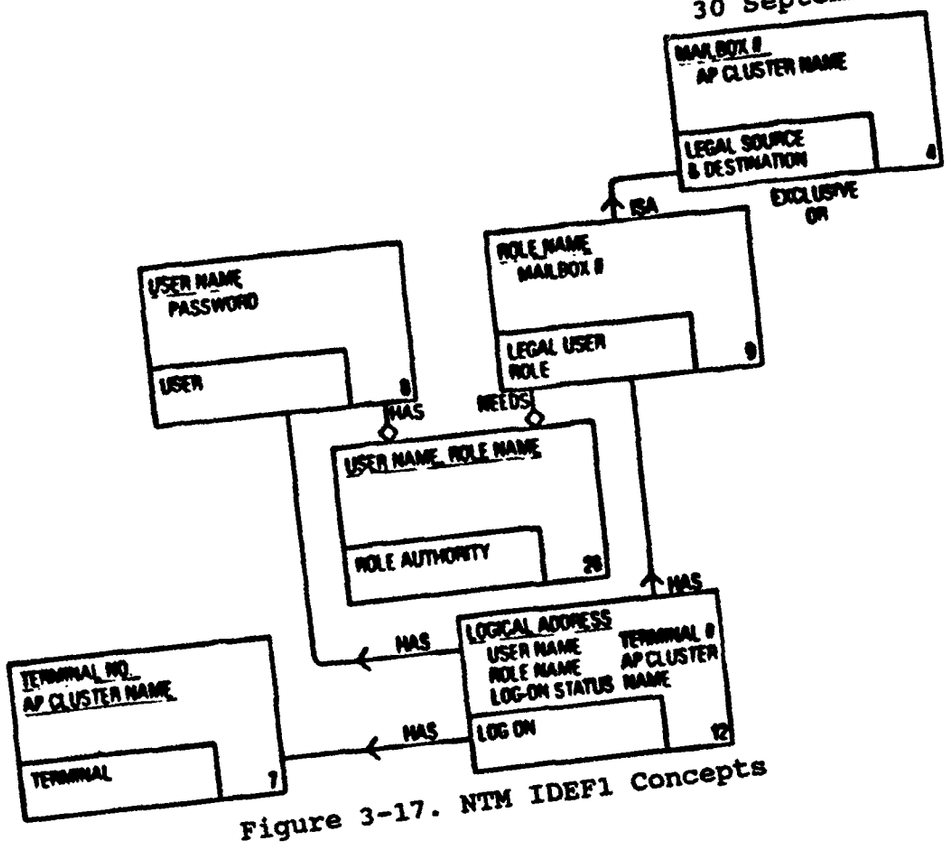


Figure 3-17. NTM IDEF1 Concepts

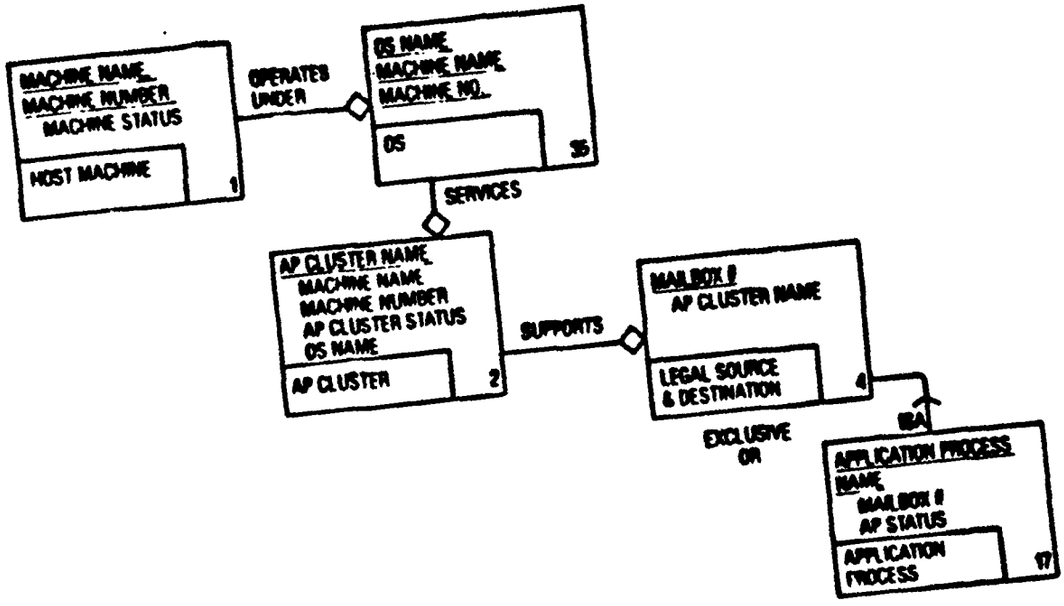


Figure 3-18. NTM IDEF1 Concepts

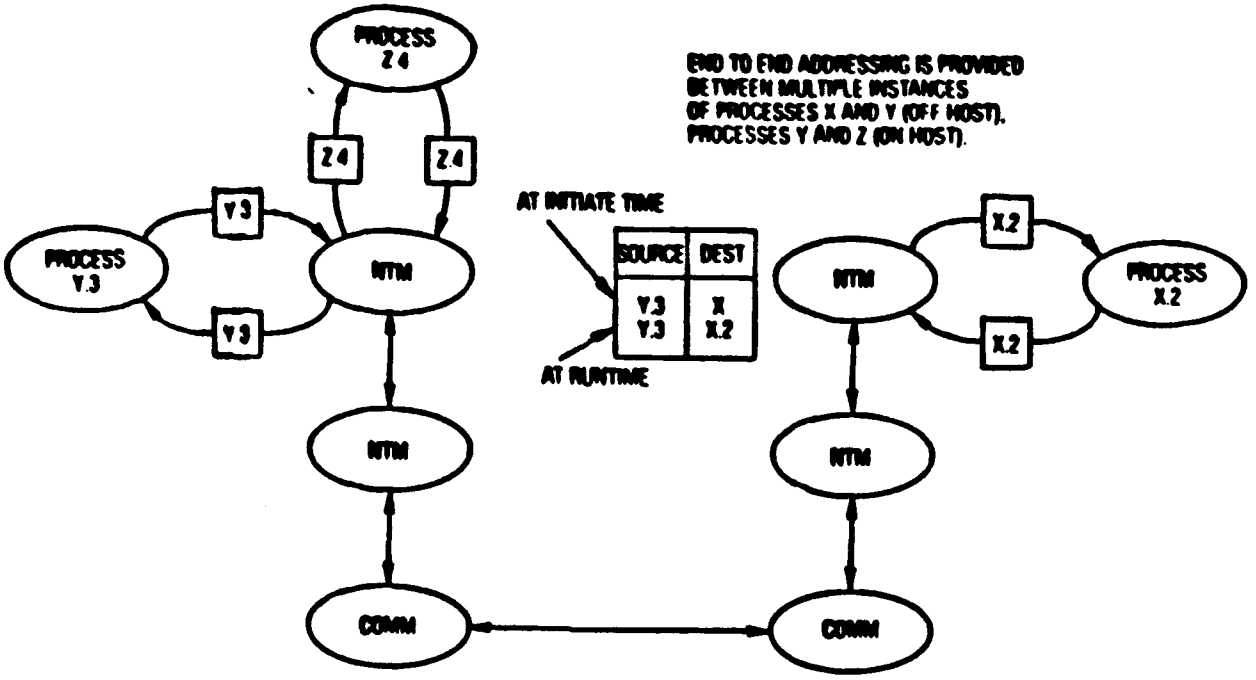


Figure 3-19. End-to-End Addressing

In the event of errors, the sender is notified. The correction of the errors is left to the sender. The error message is logged in the Test Bed error file.

2. Integrity Checking of the Data Section of NTM Bound Messages (Future)

   This type of integrity checking is performed by the receiver of the message. The data supporting this integrity check is supplied to the NTM by the CDM, on an AP Cluster basis and may be kept locally for performance reasons. The types of data integrity checks include:

   a. Number of data elements expected in the message

   b. Edit of each data element (alphanumeric, numeric, strings). Each data element editing information describes the number of symbols expected.

   c. Range check of each data element (for numeric data types)

   d. Domain check

In the event of detected errors, the sender of the message is informed of the presence of errors. The correction of the errors is left to the transmitting Application Program. The error message is logged in the Test Bed error file.

3.1.1.3.22 Message Guaranteed Delivery

When requested, the NTM guarantees the delivery of messages given to it by an Application Program. Messages to be handled in a guaranteed delivery mode are of the appropriate message type and use special interface services.

Guaranteed delivery is a service provided by the NTM for interhost and intrahost communications between Application Process Clusters or within the same Application Process Clusters. Figure 3-20 shows how guaranteed delivery messages are transmitted and acknowledged. The Application Process which originates the message (source) receives acknowledgement from NTM 1 after the message has been journalized on non-volatile memory (File 1). The NTM on the AP Cluster of the destination Application Process (NTM 2) acknowledges receipt of the message after the message has been journalized in non-volatile memory (File 2). The destination process acknowledges receipt of the message explicitly (via an acknowledge message) to the AP Cluster NTM 2.

Attempts to deliver the message are initiated when either one or the following conditions occurs:

- Message is first received by NTM 1 or by NTM 2
- Destination AP Cluster is restarted
- Destination Application Process is initiated

The guaranteed delivery messages are time stamped and logged. A utility allows the scanning of the log and extracting the messages whose ages exceed a given threshold. Such messages are displayed on the IISS operator console for disposition by the IISS operator. The display includes the following information:

- Time stamp
- Source, destination
- Full message contents

3.1.1.3.23 System Statistics Gathering

The NTM gathers the following system usage statistics:

- Message type, serial number, time stamp
- Application Process start and stop time

The NTM statistics gathering capabilities can be controlled on a message-by-message basis or on a system wide basis through NTM configuration data defined by the System Administrator and downloaded from the CDM at start up time to the NTM.

The statistics are kept in files on each of the Test Bed hosts. This approach is simple. The statistics files can be accessed and cleared by the IISS operator with proper authority. The reduction of the raw data provided by the system statistics gathering capabilities include the following computation:

- o Message type (total number)
- o Message to a specific destination (total number)
- o Message from a specific source (mean, std deviation)
- o Application Process run time (mean, std, deviation)

Vendor supplied accounting packages resident on the various hosts supplement the statistics provided by the Test Bed Software.

3.1.1.3.24 Data Downloading

Each NTM is responsible for requesting its configuration/operational support data from the CDM when restarted. Thus each NTM sends to the CDM request processor a downloading message request which identifies the AP Cluster and required information which is returned by the CDM.

3.1.1.4 Network Transaction Manager Functional Specifications

The functional specifications implied by the scenarios presented in Section 3.1.1.3 are identified and are presented in this Section.

3.1.1.4.1 Update NTM Table (Future)

Mission: To initiate and to implement the downloading, as required, of CDM data needed to support the operation of the NTM.

Functional Specifications:

1. Formulate request for NTM data by transmitting a message to the CDM Request Processor. The message uniquely identifies the originating NTM.
2. Receive the NTM data transmitted by the CDM.
3. Check the NTM data transmitted by the CDM.
4. Initialize the NTM tables and store NTM data received from the CDM.

5. Detect non-response from CDM request processor.

6. Log error in Test Bed error file.

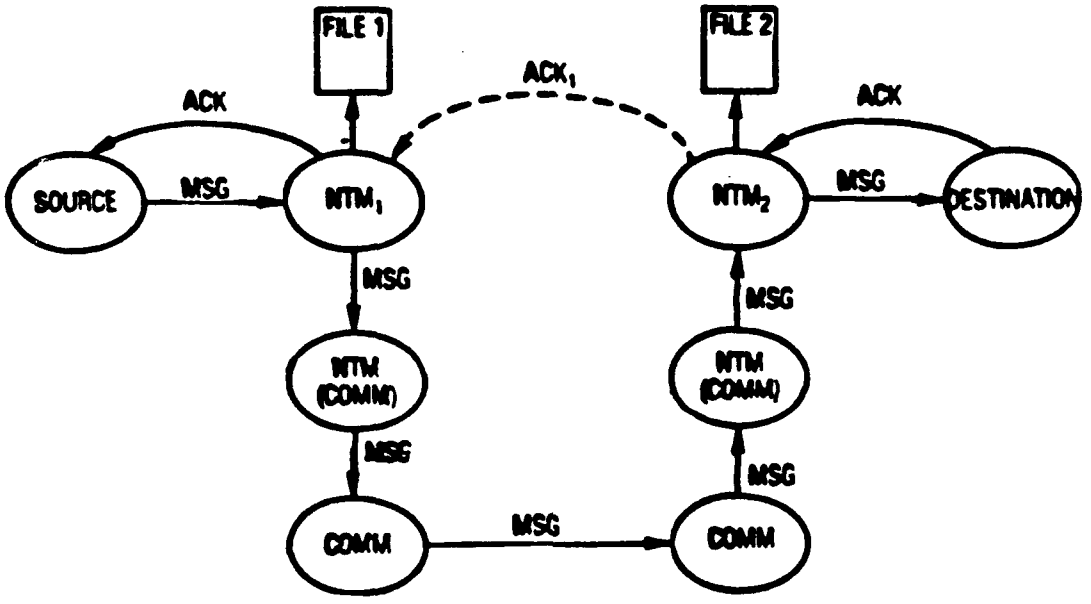


Figure 3-20. Guaranteed Delivery

7. Report completion of download operation and status to host IISS operator station.

3.1.1.4.2 Receive and Interpret Message

Mission: To receive and to interpret messages sent to the NTM by the Application Processes or by the COMM Subsystem.

Functional Specifications:

1. To name uniquely the mail boxes to be used in communicating with one specific instance of an Application Process

2. To detect when a mail box has been written into

3. To read the message contained in the mail box

4. To provide any acknowledgement, if required, to the transmitting process

5. To remove the header of the message, if required

6. To identify the nature of the message

7. To route to its proper destination

8. To detect Application Process message initiation requirement

9. To detect Application Process completion message

10. Above functions are implemented by making use of the communication primitives described in Section 3.1.5

3.1.1.4.3 Authorize Message (Future)

Mission: To enforce the legal source/legal destination constraints declared for the NTM messages, and to report exceptions.

Functional Specifications:

1. To enforce the legal source constraints associated with a given AP Cluster

2. To enforce the legal destination constraints associated with an AP Cluster

3. Legal source and legal destination constraints are defined in the CDM by the System Administrator. The legal source and legal destination constraints are defined on an Application Process basis.

4. To log violation of the legal destination and source constraint on the error log

5. To notify the transmitter Application Process of legal destination constraint violations

6. To obtain legal source and legal destination constraints

7. To allow the definition of AP Cluster authorization privileges, as well as authorization by specific Application Processes.

   Example: MCMM, * : All MCMM
   Example: MCMM, 26: Only MCMM 26

3.1.1.4.4 Complete Message Header

Mission: To formulate and to append the proper NTM header to a message transmitted to the NTM for processing.

Functional Specifications:

1. To obtain all the information from the source Application Process. For example:

   o Message type

- o Destination (logical)
- o Delivery trigger
- o Message length
- o Data type (ASCII/EBCDIC or (BINARY)

2. To obtain all the predefined information supplied for the message. For example:
   - o Logging flag
   - o Statistics flag
   - o Test flag
   - o Integrity check flag
   - o Guaranteed delivery
   - o Message priority
3. To create the following information:
   - o Message serial number (on an AP Cluster basis)
   - o Message time stamp (on a host basis)-only if logged
4. To obtain from the NTM configuration:
   - o Logical source name
5. To obtain the length of the message header containing the above information
6. To format the above information in a header per the format specifications described in Section 3.3
7. To report any errors to the source Application Process
8. To log any error in the Test Bed error file

3.1.1.4.5 Log Message

Mission: To create a permanent, user-readable record of the messages transiting in the Test Bed environment. This record is a complete description (header, data section) of the message.

Functional Specifications:

1. To obtain the message logging information from:
   a. The message header
   b. The NTM operational option data defined in the CDM.

2. To log messages whenever the message header flag or the CDM-defined log message system flag is set. (Future - initially all messages are logged)

3. When appropriate, to log the information (header, data) contained in the message

4. The record is kept in the message log file kept on each host

5. To allow for the centralized access of the message log file by the IISS operator

6. To allow for the selective clearing of the file by the IISS operator according to a procedure

3.1.1.4.6 Send Message

Mission: To route and transmit message to its appropriate destination.

Functional Specifications:

1. To determine the physical destination address of the message from its logical destination address

2. To decide whether the message is to be routed on AP Cluster, off AP Cluster, or off host

3. To transmit the message to the COMM AP Cluster for off host transmission. The message is to be transmitted to the appropriate mail box, according to the priority of the message

4. To obtain acknowledgement of receipt, if required

5. To detect and report, if appropriate, the following error conditions to the message initiator:

   a. Destination address is unknown (mailbox not found)

   b. Destination address is the initiator itself

   c. Destination address is an inactive process

6. To log the above error conditions in the Test Bed error file

7. Above functions are implemented by making use of the communication primitives described in Section 3.1.5

3.1.1.4.7 Interpret Message Type

Mission: To examine and to differentiate between messages of different types for the purpose of providing proper NTM actions and responses.

Functional Specifications:

1. To obtain the message type information
2. To recognize all valid message types
3. To invoke the proper NTM processing of the incoming message
4. To detect and report undefined message types

3.1.1.4.8 Initiate Application Process

Mission: To request the local host operating system to load and to execute a given Application Process.

Functional Specifications:

1. To apply the following scheduling rules to select the Application Processes to be initiated:
   - o Selection based on absolute time (Future)
   - o Priority based selection rules (Future)
     - Higher message priority first, lower priority last
     - Messages of the same priority are scheduled on a first in first out basis
     - Lower priority messages are aged whenever a higher priority message is selected (future)
2. To formulate a request to the local host operating system to load and execute the Application Process described in the message extracted from the message queues
3. To create, to name, new mailboxes
4. To maintain the higher and lower message priority queues. Maintenance functions include:
   - o Removal without omission or duplication of messages that have been dispatched
   - o Detection of overflow
   - o Reporting of errors in queue management
5. To obtain status information from the host operating system
6. To maintain the list of active application processes

7. To report the failure of initiating an Application Process to the requestor, if appropriate, and to log the failure, with appropriate error description in the Test Bed error file

8. To record the Application Process identification and time of day in the Application Process Activity Log (start time) (Future - Activity Log).

3.1.1.4.9 Terminate Application Process

Mission: (1) to perform the housekeeping operations associated with the normal termination of an Application Process, and (2) to perform the error notification and housekeeping operations associated with the abnormal termination of an Application Process.

Functional Specifications:

1. To detect the termination of an Application Process

2. To recognize normal and abnormal termination conditions

3. In the event of normal termination:

   a. To record the Application completion time in the Application Process Activity Log (Future)

   b. To clear any mail boxes used to communicate with the Application Process just terminated

   c. To notify the requestor of the termination (if required)

   d. To maintain the list of active application processes

4. In the event of abnormal termination:

   a. To record the Application termination time in the Application Process Activity Log (Future)

   b. To obtain the termination code or status from the local operating system

   c. To generate the appropriate Test Bed error code equivalent to the local operating system abnormal termination code

   d. To notify the requestor of the abnormal termination condition, if required.

   e. To clear any mail boxes used to communicate with the Application Process just terminated

   f. To initiate the termination of any offspring Application Processes of the Application Process (Future)

g. To update the directory of active Application Processes by deleting the Application Process just terminated

h. To update the directory of offspring Application Process by deleting the offspring Application Processes identified in Step f

3.1.1.4.10 Exception Handling

Mission: (1) to gain knowledge of exception calls placed by any on AP Cluster, Application Subsystems and Test Bed System Services; and (2) to log and notify requestors of such calls.

Functional Specifications:

1. To keep informed of all exception calls to the local operating system generated from within the AP Cluster

2. To obtain the error status code associated with the exception call and the identification of the offending Application Process

3. To map the local operating system error codes into equivalent Test Bed error codes

4. To record occurrence of errors and identification of offending Application Processes into Test Bed error file

5. To notify the requestor/initiator of the Application Process of the error status as appropriate.

3.1.1.4.11 Communication with Application Process

Mission: To accept messages and to deliver messages to the Application Process.

Functional Specifications:

The above mission is accomplished by making use of the Send message and Receive message functional capabilities described in Sections 3.1.1.4.6 and 3.1.1.4.2.

3.1.1.4.12 Message Pairing

Mission: To match message pairs (question/answer) and to report open pairs at the end of time out period.

Functional Specifications:

1. To recognize messages requesting pairing services

2. To extract the identity of the expected messages in reply to the message requesting pairing processing

3. To initiate a watch dog timer upon receipt of the pairing processing request message. The source defining the time out period (CDM or message, or system default) and the duration of the time out. (Release 2.0 implementation is "system default")

4. To close the pair when the expected message is received and to disable the watch dog timer

5. In the event of a time out, to notify the Application Process who initiated the message requesting pairing processing

6. To record the occurrence of the time out in the Test Bed error file. The record identifies:

   - o The Requesting Application Process
   - o The message type and/or serial number requesting pairing processing
   - o The time of day
   - o The destination Application Process

3.1.1.4.13 Error Logging

Mission: (1) to log the occurrence of an error with sufficient information to identify:

- o The nature of the error
- o The offending process
- o The end user of the process
- o The time of day of the error

and (2) to gain access to the log from a central location (IISS operator) with an authorized procedure

Functional Specifications:

1. To maintain an error logging file on each Test Bed host
2. To allow access to this file by an authorized user
3. To support the chronological ordering of the error file obtained by concatenating the local error files maintained on each host (Future)
4. To allow the clearing of the error file under control of an authorized user with an authorized procedure
5. To log the following information for each error:

- o Nature of the error in Test Bed error codes
- o Unique identification of the offending process
- o End user of the offending process
- o Time of day of the error

3.1.1.4.14 Message Integrity Checking

Mission: (1) to detect messages which have: (a) improperly formulated headers, (b) improperly formulated data sections (Future); and (2) to report and log such conditions.

Functional Specifications:

1. To allow header integrity checking control via NTM operating options defined in the CDM. (Future: Rel 2.0 - options defined in NTM)
2. To allow data integrity checking control via NTM operating options defined in the CDM (Future). In the early implementation, Application Processes are responsible for message data integrity checking.
3. To support header and data integrity checking control at the NTM and system level.
4. To support data integrity checking control on a message basis. (Future) In the follow-on implementation, this service is provided if either:
   a. Data integrity checking is requested in the message header
   b. Or if data integrity checking is requested at the system level via information provided by the NTM.

   Data integrity checking may, in addition, be specified to be performed by the NTM or by the Application Process.
5. To perform, when requested, the following message header integrity checks:
   - o Editing of each field
   - o Self consistency of the following user defined fields:
     - Message type
     - Destination

- Delivery trigger
- Data type

6. To perform, when requested via CDM flag, the following message data section integrity checks: (Future)

   o Number of data fields

   o Editing of each field

   o Range checking of each data element

   The data used to support the above checks is defined in the CDM and downloaded to the NTM tables on start up.

   The CDM definitions recognize:

   o Integer notation

   o Floating point notation

   o String notation (n characters)

   o Undefined range

   o Upper, lower range limits

   o Domains

7. To report the occurrence of errors to the message initiator via predefined Test Bed error codes

8. To log the occurrence of the error in the Test Bed error file. The record includes:

   o Message type

   o Message serial number

   o Message initiator

   o Error code

   o Time of day

9. To prevent the propagation of an erroneous message through the Test Bed

3.1.1.4.15 Resource Usage Statistics

Mission: To gather the following Resource Usage Statistics:

   o Usage frequency

   o Processing time

- Message frequency
- Process response time

Functional Specification:

1. For each message, to gather the following information at the sender and receiver AP Cluster:
   - Message type
   - Message serial number
   - Message time stamp
2. For each Application Process, to gather the following information on the Application Process AP Cluster: (Future - Application Process logging files are not in the Release 2.0 implementation)
   - Application Process name
   - Start time
   - Completion time
   - Above information is gathered if it is available
3. To log the above information in files maintained locally (message logging file and Application Process logging file (Future))
4. To support the querying and concatenation of the message logging files and Application Process logging files from a central location (Future).
5. To clear the message logging and the Application Process logging files by an authorized user
6. To compute the following statistics from the concatenated message and Application Process logging files: (Future - The raw data is collected but not processed to obtain this information in Release 2.0)
   - Message frequency by message type
   - Application process usage frequency by application process type
   - Application process processing time (mean, std dev) by application process name and for all application processes
   - Application process response time (mean, std dev) by application process name and for all application processes

3.1.1.4.16 Guaranteed Delivery

Mission: To guarantee selectively the delivery of messages to Application Processes.

Functional Specification:

1. To provide the following guaranteed delivery services to messages requesting this service

2. To acknowledge the receipt of a guaranteed delivery message to the initiating Application Process or to the NTM forwarding such a message.

3. To initiate the delivery of such a message to its next delivery point until an acknowledgement is received from the delivery point (NTM or ApplicationProcess)

   Attempts to deliver the message are initiated when either one of the following occurs:

   - o Message is first received
   - o Target AP Cluster is restarted
   - o Target Application Process is initiated

4. To notify the IISS operator in the event of failure to deliver the guaranteed delivery message. Failure to deliver such a message is recognized when the age of the message exceeds the age limit set for the AP Cluster. In the event of delivery failure, the following information is provided to the IISS operator: (Future - Release 2.0 is a manual process to review message files)

   - o Time stamp of messages (date, time of day)
   - o Source, destination
   - o Contents of message (header, data section)

5. To record guaranteed delivery failures in a special log on the host where the failure was detected

3.1.1.4.17 System Status Broadcast

Mission: To broadcast system status information to all IISS users. The IISS operator initiates the broadcast and supplies the text of the message to be broadcasted.

Functional Specification:

1. System service which allows the broadcast to all terminals

2. Service can be invoked by:

   a. IISS operator

   b. Privileged programs

3. Operator or privileged program supplies text of broadcast

3.1.1.4.18 IISS Start Up

Mission: To start up IISS Software Services on a Test Bed host. Start up operations are initiated by the IISS operator.

Functional Specifications:

1. To accept IISS operator start commands and data inputs from the IISS operator console, or from a command file

2. To start up the IISS Software Services required to support the functionality of the Test Bed:

   a. To initiate, in the proper sequence, the IISS software modules required to support the Application Cluster Configuration assigned to the host.

   b. To carry out the start up scenario for the specific host application cluster configuration (loading of operational option data, dialogs with other hosts, etc.).

3. To report the completion (or any error message) of the IISS Software Services start up to the IISS operator.

3.1.1.4.19 IISS Shutdown

Mission: To shutdown the IISS Software Services running or dormant on a Test Bed host. Shutdown operation is initiated by the IISS operator.

Functional Specifications:

1. To accept IISS operator shutdown commands either from:

   a. The host IISS Operator Console, or

   b. A shutdown message transmitted by the NTM

2. To broadcast shutdown messages to all IISS terminals at a frequency prescribed by the IISS operator

3. To monitor processes still active at the end of the shutdown countdown period and to report the identities of these processes to the IISS operator

4. To terminate, under the direct control of the IISS operator, those processes found active at the end of the shutdown period

5. To save all pertinent information required for the restart of the Test Bed without omission or duplication of application processes and without the loss of concurrency between the databases, journals, and application process request queues. (Future)

6. To terminate all IISS Software Services in the proper sequence

7. To report the completion of the shutdown operations to the IISS operator

3.1.1.4.20 IISS Restart

Mission: 1) To restart the IISS Software Services on a given Test Bed host; 2) To achieve synchronization with other Test Bed Application Processes and to maintain database concurrency. Restart is initiated by the IISS Operator.

Functional Specifications:

1. To accept IISS operator restart commands and data inputs from the IISS Operator Console or from a Command file

2. To startup the IISS Software Services required to support the functionality of the Test Bed

   a. To initiate, in the proper sequence, the IISS Software modules required to support the host Application Cluster Configuration.

   b. To carry out the startup scenario for the specific host Application Cluster configuration

   c. To achieve synchronization of all Application Process request queues in the IISS system, without omission or duplication of application processes (Future - request queues are not saved during shutdown)

   d. To maintain concurrency of the on host databases with the other databases integrated in the Test Bed

3. To report the completion (or any error message) of the restart of the IISS Software Services to the IISS operator

3.1.1.4.21 IISS Recovery (Future)

Mission: To perform the recovery of the IISS databases, journals, and application process request queues, without omission or duplication. The IISS recovery is initiated by the IISS operator.

Functional Requirements:

1. To ensure that Recovery processing is equivalent to normal processing
2. To accept Recovery initiation commands and other recovery input data from the IISS operator
3. To be able to roll back, roll forward to and from an IISS operator designated checkpoint
4. To re-apply the stream of application processes submitted during normal processing without omission or duplication
5. To allow the exclusion, under IISS operator control, of one or several application process types from the stream of application processes submitted to recover the databases
6. To maintain concurrency between application process queues, database journals, and the databases integrated in the Test Bed
7. To suspend normal processing while performing the recovery operations
8. To notify the IISS operator of the completion (or any error messages) of the recovery operation

3.1.2 Common Data Model Configuration Item

3.1.2.1 CDM Mission Statement

The Common Data Model serves two purposes in the implementation of the Test Bed.

First, The CDM is a repository of system information. This information is used at compile time or at run time. The information contained in the CDM is used to support all operational phases of the Test Bed life cycle: maintenance, application development and operation.

Second, the CDM allows application processes to query Common Data distributed among the Test Bed databases without regard to its location and format.

3.1.2.2 CDM Functional Areas

The configuration tree of the CDM is shown on Figure 3-21, which shows three major functional areas:

- o CDM Maintenance
- o Application Development
- o CDM Request Processor

3.1.2.2.1 CDM Maintenance Functions

The CDM Operational Scenarios presented here are introduced for the sole purpose of supporting the identification of the functional specifications to be met by the CDM processor. These scenarios are not meant to imply a specific implementation of these functional specifications. Consequently, the final design of the CDM processor may implement scenarios which differ from the scenarios presented here.

3.1.2.2.1.1 Operational Scenarios

The description of the Common Data, that is the Common Data Model and the system data, must be entered, edited and retrieved.

The CDM Maintenance Operations referenced above are performed by the CDM Administrator. These operations are represented on the attached IDEF0 diagram entitled "Operate Common Data Model", node A1 (Figures 3-22, 3-23, 3-24, and 3-25 for the IDEF0 diagrams).

The following narrative supports the discussion of the "Maintain CDM Data" node. With respect to this node, one can draw the following remarks:

1. A schema of the CDM must first be built in order to retrieve, store and edit any CDM information.
2. To ensure a self consistent CDM, the relationship of any update to the CDM entity or attribute definition must be checked. A minimum level of checking includes uniqueness checking. The sheer amount and the diversity of the data items to be stored in the CDM militate for computer assisted checking.
3. Any update to the CDM takes place once the checks listed above have been performed and did not reveal any errors.
4. The data dictionary relationships supported by the CDM are automatically updated as part of the update function.
5. The version number of the CDM is likewise automatically maintained. To be useful, the CDM version numbering system must be of sufficient granularity to avoid recompilation of application processes unaffected by a change in the CDM data. (Future)

3.1.2.2.1.2 CDM Maintenance Functional Specifications

The CDM Maintenance functions are performed by the CDM Maintenance utilities to create, revise and expand the CDM data.

The CDM Maintenance utilities perform the following functions:

1. Access Control: Access control to all CDM Maintenance functions are performed under strict access control. Access control is enforced via password and user name.

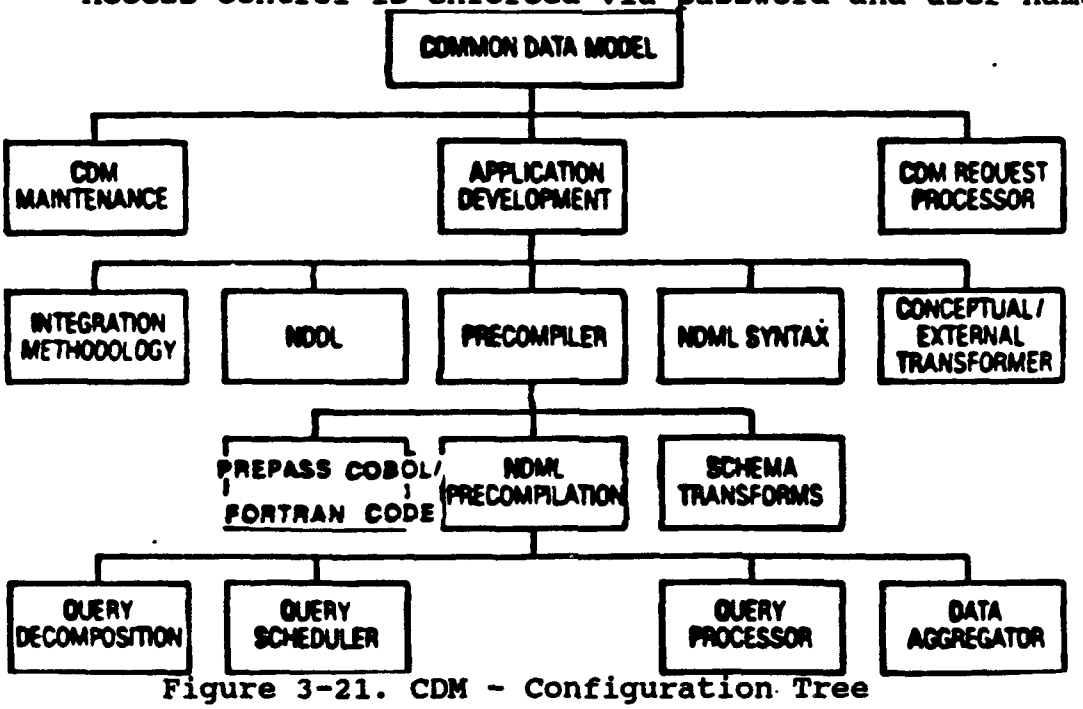


Figure 3-21. CDM - Configuration Tree

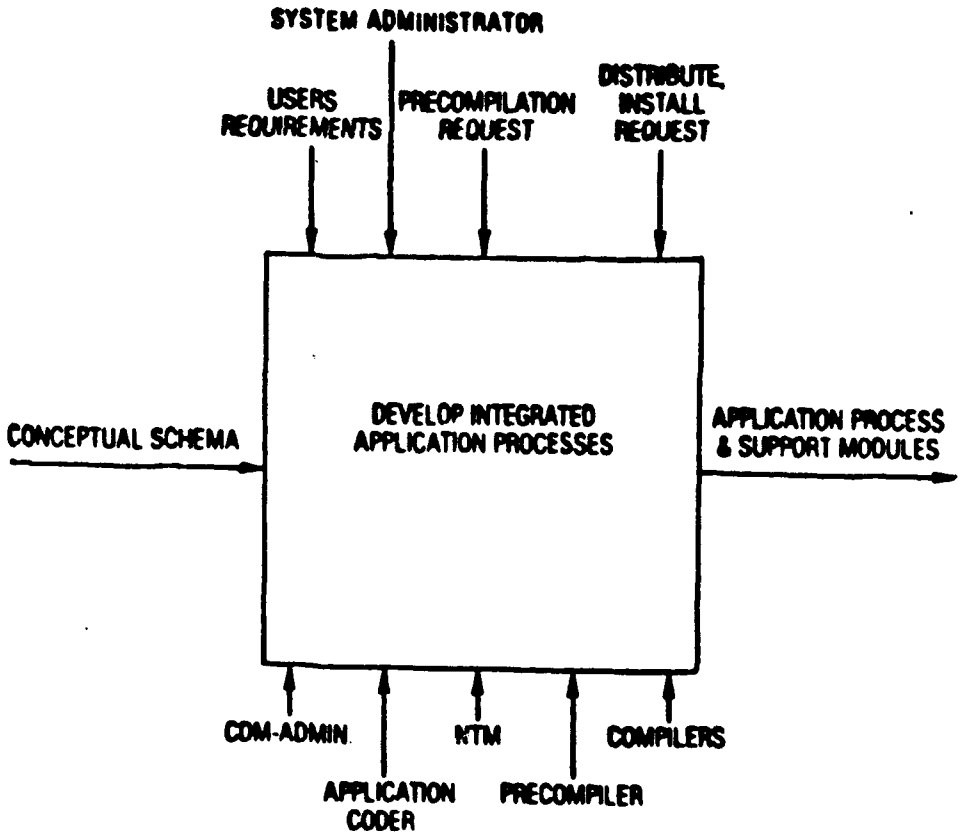
2. CDM Data Access: The following data manipulation functions are provided:

   - o CDM Data Entry Functions
   - o CDM Data Update Functions
   - o CDM Data Edit Functions
   - o CDM Data Selective Retrieval Functions
   - o CDM Data Reporting Function
   - o CDM Data Delete Function

   The above functions are applicable on all CDM data.

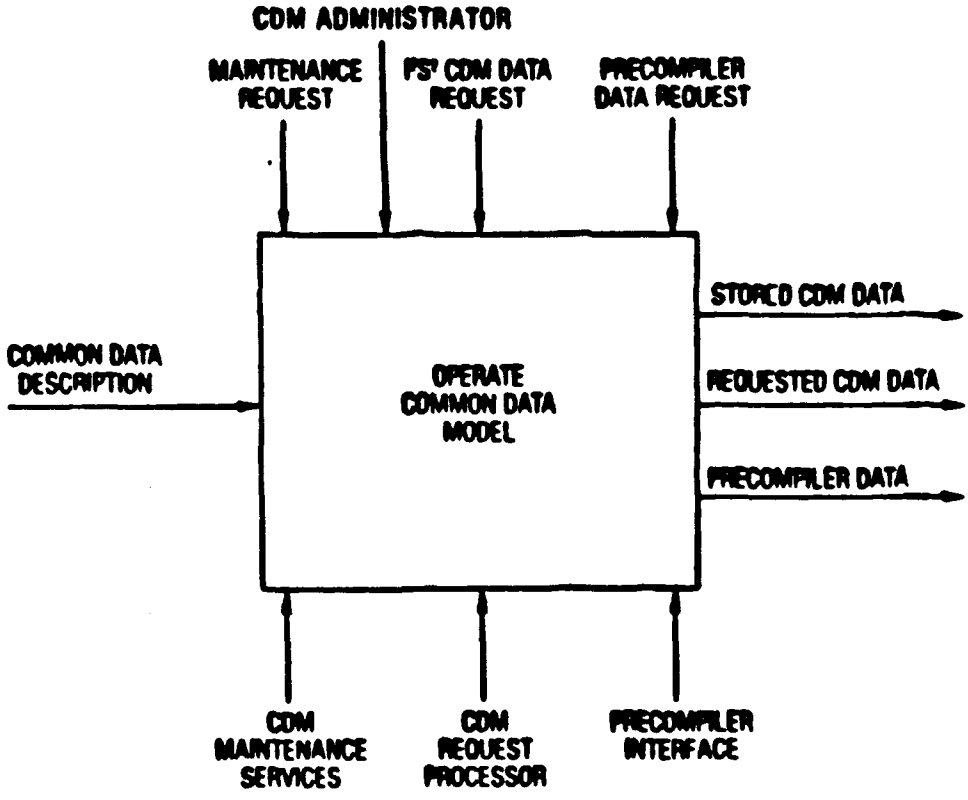
3. CDM Data Definition: The above data entry functions are used to define all data known to the CDM. The data known to the CDM includes:

- CDM Schema
- Application External Schemas
- Test Bed Conceptual Schema
- Integrated databases Internal Schemas
- Mappings Existing Between the Above Schemas
- Domain Information
- Range Information
- Network Resources (location of AP Clusters, databases, configuration) (Future)
- Test Bed Security Information (Future)
- User Interface Support Information (Future)
- Virtual Terminal/Real Terminal Mappings (Future)
- Error Messages (Future)



PURPOSE TO IDENTIFY DEVELOPMENT STEPS OF INTEGRATED APPLICATION PROCESSES
VIEWPOINT SYSTEM ENGINEER (ISS)

Figure 3-22. A-0 Develop Integrated Application Processes



PURPOSE: TO MODEL CDM FUNCTIONS
VIEWPOINT: ISS SYSTEM ENGINEER

Figure 3-23. A-0 Operate CDM

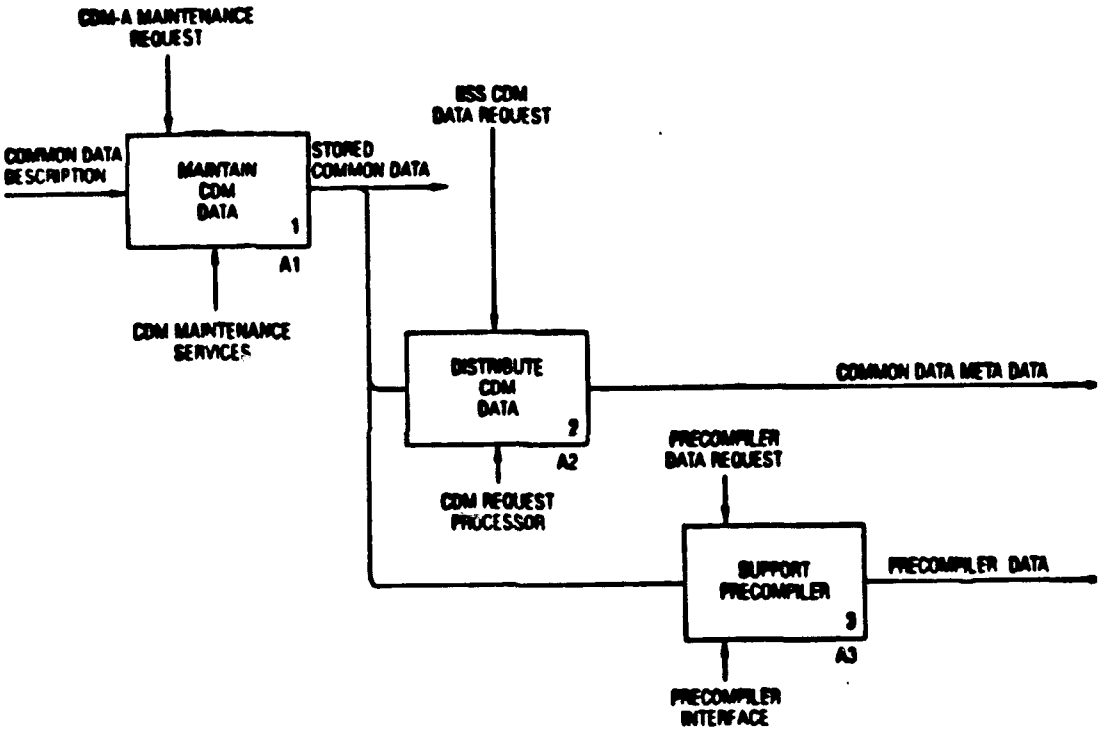


Figure 3-24. A0 - Operate CDM Functions

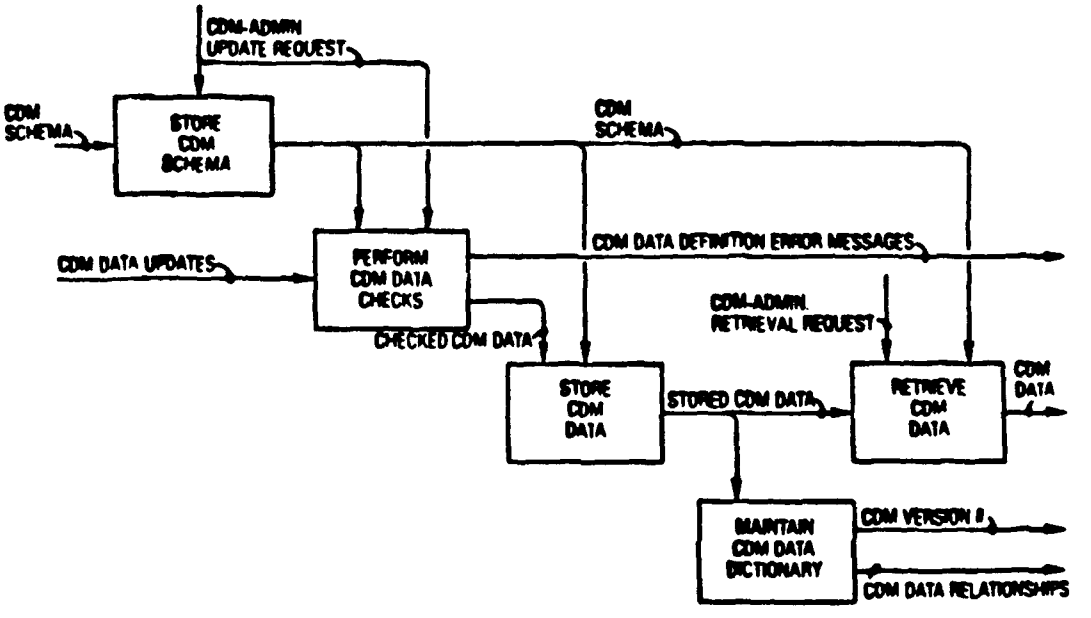


Figure 3-25. A1 - Maintain CDM Data Functions

- o Message Format Definition (Future)
- o Application Process description (single instance, queue driven,...) (Future)

4. CDM Data Checking: The following CDM data checking functions are provided:
   - o Uniqueness of relation names at the conceptual level
   - o Uniqueness of entity names at the conceptual level
   - o Uniqueness of attribute use class names at the conceptual level
   - o Uniqueness of key class names at the conceptual level
5. CDM Data Dictionary: The following data dictionary functions are provided:

- o Index of Conceptual Entities
- o Index of External Entities
- o Index of Internal Entities
- o Cross Reference of External Entities and Application Processes
- o Glossary defining the various entities, attributes, relationship

6. CDM Version Number: A CDM data version numbering scheme is provided. This scheme maintains automatically the version number of the information used by an Application Process and the version number of the CDM data used to compile the Application Process. Version number identifies CDM data changes that require recompilation of Test Bed Application Processes and downloading of CDM data. (Future)

7. CDM Integration Methodology Utilities: The declaration and checking of Extended IDEF1 constraints are supported.

8. CDM database Recovery and Concurrency Control: The recovery of the CDM database is supported via the recovery mechanisms provided by the CDM database manager (ORACLE).

3.1.2.2.1.3 Implementation Steps

To reduce development cost and to meet the development schedule, the CDM Maintenance functions are implemented in a stepwise fashion.

Step 1 is implemented by making use of the maintenance utilities provided by the database Manager supporting the CDM. The ORACLE Relational database Manager has been selected for the CDM. As part of its standard utilities, ORACLE provides:

- o Access Control
- o Access from a VAX Native Terminal
- o Data Entry

Step 2 would include all other functions not included in Step 1. The following list applies to the ORACLE implementation:

- o Access to Maintenance Utilities from an IISS Terminal
- o Access through the NDML to the CDM data
- o Forms assisted maintenance of the CDM

3.1.2.2.2 Application Process Development

3.1.2.2.2.1 Operational Scenario

The development steps of an integrated Application Process are portrayed in the IDEF0 diagram shown on Figures 3-26 and 3-27.

With reference to this diagram, the major steps of the development of an integrated Application Processes are:

1. Establish Data Requirements of Application Process

   Based on the data requirements of the user, the CDM establishes an External Schema suitable to support the application process contemplated. If all entities and attributes required by the application process already exist in the Conceptual Schema, this operation reduces to formulating an External Schema which is a subset of the Conceptual Schema. Otherwise, the new entities required by the new Application Process under development must first be added and integrated in the Conceptual Schema, and reflected in appropriate Internal Schemas.

   From an operational viewpoint, the definition of the External Schema and any addition to the existing Conceptual Schema are performed by the CDM Administrator (for data security reasons). The CDM Administrator uses the Test Bed Neutral Data Definition Language to define the required schemas.

   Once the External Schema has been developed by the CDM Administrator, a report of the External Schema is then obtained and communicated to the Application Developer (application programmer).

   For data security reasons, however, the External Schema is referenced by name in the Application Process and is directly obtained by the Precompiler from the CDM. This approach guarantees tight data security, since the Application programmer cannot modify the schema and hence his data access privileges, without explicit action from the CDM Administrator. The printout of the External Schema conveys the information required by the Application Developer to write the integrated Application Process.

2. Generate Code of Application Process

2a. Integrated Applications (Class II Environment)

   Based on the user requirements and on the External Schema developed; the Integrated Application Developer designs and codes the required Application Process. In the Test Bed, the Application Process is either a COBOL

or Fortran program which queries data distributed across the Test Bed databases as if the data was contained in a single database bound to the Application. All data queries are formulated via the Test Bed Neutral Data Manipulation Language. The I/O operations are performed via Test Bed supported I/O services. User interaction is facilitated by the Test Bed provided User Interface. This interface supports forms management, data checking and menu handling capabilities. The Application programmer references the user interface services via procedure calls.

2b. Existing Applications

Existing Applications need to be brought under the control of the NTM to be under the control of the IISS user. This is done by making existing applications come under the control of the Test Bed known to the CDM. This definition information is used to update the relevant NTM tables, and permits the IISS user to cause the execution or the cancellation of existing applications from the IISS user terminal. Existing Applications, in general, perform Input/Output operations to the user terminal and to the database and files supporting them. Database and file I/O operations of existing applications are, in general, satisfactory to the IISS user and do not need tc be disturbed when the application is brought under the control of the the Test Bed.

On the other hand, the terminal I/O operations need to be redirected via the NTM to take place the IISS user terminal. The redirection of terminal I/O's can be accomplished either by:

1. Modifying the I/O statements contained in the existing Application Process Code so as to perform the nondatabase I/O via the NTM and UI services provided by the Test Bed (Send Message, Receive Message, Output Screens, etc.), or

2. Modifying some to the system I/O routines that are linked with existing applications so as to effectively redirect the I/O operations to the NTM. This approach, which works well on some machines (e.g., IBM CICS) and does not require customized modifications of each application, is not, however, possible on all machines.

3. Precompile Code of Application Process

3a. Isolate and Replace NDML Statements

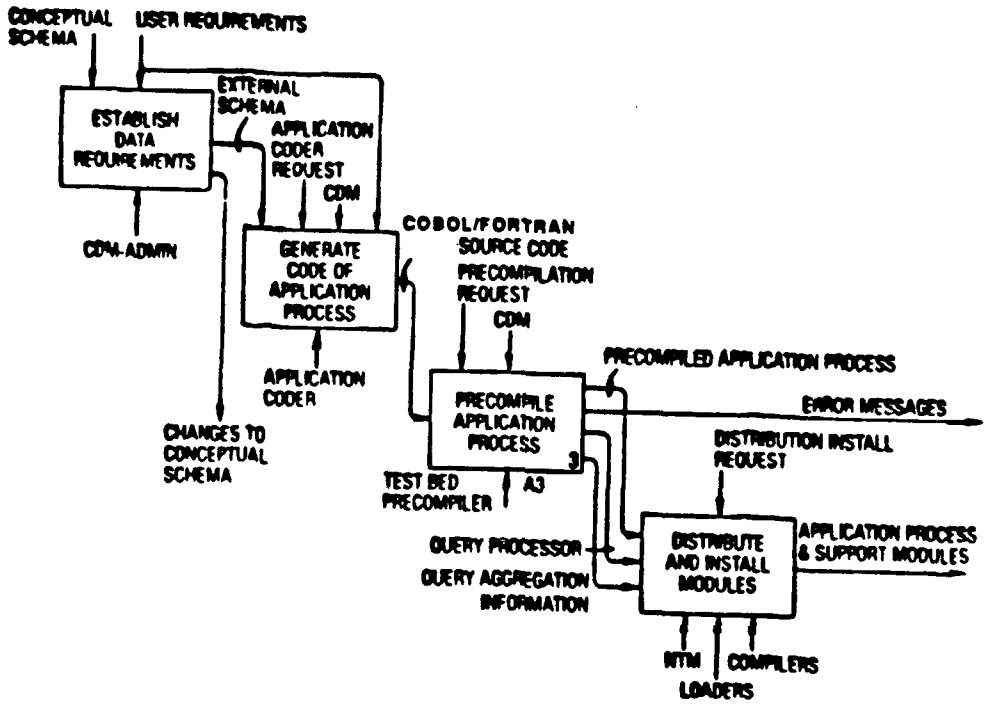


Figure 3-26. Develop Integrated Application Processes

The above development steps result in a COBOL or Fortran program which implements the logic of the intended Application Process. The program thus written is not however compilable by the host provided COBOL or Fortran compiler. The reasons for this are that the program contains:

- o A reference to an external schema specified in the Test Bed NDDL not recognized by the host.
- o NDML statements which are not recognized either by the COBOL or Fortran compiler or by any of the host supported database managers.

Thus, the integrated application programs must be precompiled to remedy the above problems. The precompiler will, in addition, transform the NDML statements into procedures which can be executed by the database managers integrated by the Test Bed.

The problems associated with the aggregation and data transforms pertaining to the distributed environment are also resolved by the precompiler.

In the 6201M implementation of the Test Bed, the precompiler is resident on the VAX and is bound to the CDM database manager. In the same environment, all queries are precompiled by the VAX precompiler and distributed to the host for final compilation. The IISS system concepts and design allow these functions to be performed on the Honeywell or on the IBM computers.

Per the above scenario, the Test Bed precompiler generates the following classes of output:

1. Precompiled application process code
2. Single node query processors (COBOL code)
3. Data aggregators (tables)
4. Query stager/scheduler (tables)
5. Conceptual to external transformers (COBOL code)
6. Error message

The precompilation process is shown in Figure 3-27, and the interaction of the precompiler with its environment is shown in Figure 3-28. The precompiled application process code is now a COBOL or Fortran Program in compilable format, where the NDML statements have been suppressed and replaced by procedures which initiate messages to the corresponding single node query processors and data aggregators.

Once successfully precompiled, the precompiled application process source code files are sent to their respective hosts for compilation. These transmissions take place via the Network Transaction Manager.

3b. Formulate Single Node NDML Query

The single node query processors are COBOL procedures which query data from a single database. As such, these procedures contain data manipulation statements which are specific to the target database managers. Also, the query processors are tailored to the target data structure.

The steps required to generate the single node query processors are these:

- The multi node conceptual NDML query is decomposed into a set of single database NDML query processors.

- The resulting multi node conceptual NDML query is analyzed and an optimizing decomposition/aggregation strategy is formulated. The strategy reflects the storage characteristics of the distributed databases as expressed by the Internal Schemas or rather their conceptual equivalent.

- The multi node conceptual NDML query is decomposed into a set of single database NDML query processors.

3c. Generate Query Processor

- For each single database NDML query processor, an internal schema access path is determined. The access path reflects the data structuring characteristics of the database holding the data. This information is derived from the internal schema of the database.

- A COBOL procedure based on the access path

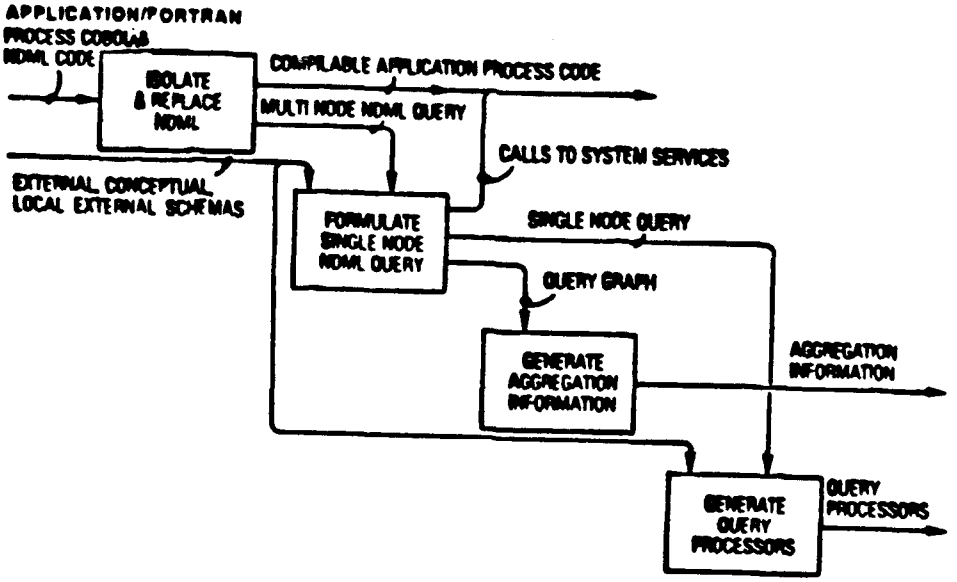


Figure 3-27. Precompile Application Process

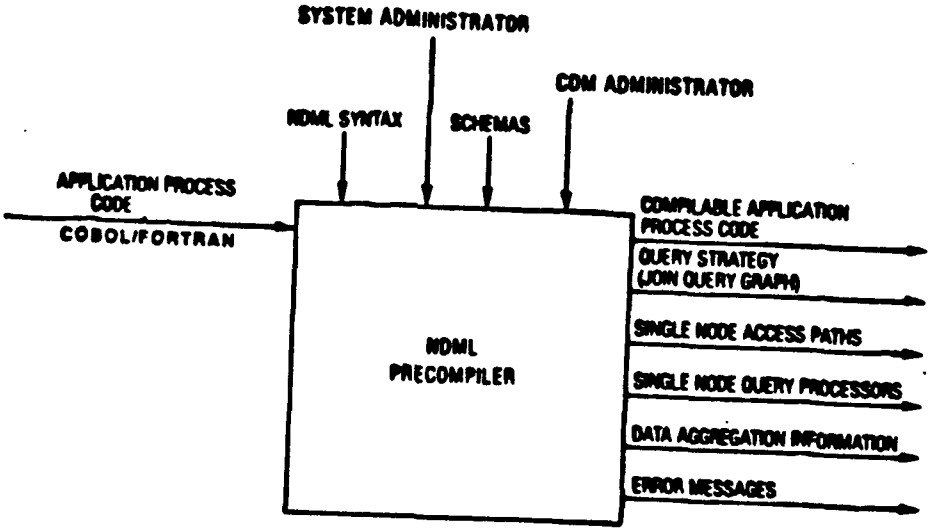


Figure 3-28. NDML Precompiler Environment

determined above is then automatically generated. This procedure is non-database manager specific(although it reflects the data structure of a specific database) and as such contains generic CODASYL data manipulation statements.

- In addition to the logic required to carry out the data retrieval operations, the query processors contain the logic required to transform the data retrieved from the internal format and units to the format and units prescribed by the Conceptual Schema. The transformation to the conceptual format and units allow performance of data integrity checks prior to aggregation as well as simplifying the declaration of the data integrity checking support information.

- Once the generic query processor has been generated, the generic CODASYL data manipulation languages are replaced by the specific data manipulation statements recognized by the target database manager.

3d. Specify Data Aggregation Step

The query aggregators perform the Join and Project Operations required to assemble the data specified by the distributed query. These join and project operations are specified at precompile time by the precompiler. The join operation allows assembly of data contained in two relations sharing a common attribute on a tuple by tuple basis, whereas the projection operator allows discarding of attributes not relevant to the query at hand.

The query graph which reflects the strategy required to fulfill the distributed query assigns the nature, sequence and location of the operations performed by the various aggregators.

The data aggregators aggregate data presented to them in conceptual format and units. Likewise, aggregators present data to other aggregators in the same format. However, the last aggregator routes the aggregated data to the conceptual to external transformer. This module performs the format and unit transformations required to return data in external conceptual formats and units to the querying Application Process.
The data aggregators aggregate data supplied by two different data sources. These sources may either be another data aggregator or a query processor.

3e. Specify Query Scheduling

The query scheduler controls the processing of a distributed query through scheduling and staging instructions specified by the precompiler. The control functions performed by the query scheduler include:

- Monitoring the status of the various query processors, data aggregators, and conceptual to external transformer involved in the distributed query processing.

- Scheduling of the data aggregation operations to minimize the transmission time (main controllable component of cost of a distributed query).

The scheduling of the data aggregators is performed dynamically, at run time, by the query scheduler.

The query scheduler uses actual data byte counts supplied by the query processors and data aggregators to decide on the location of the next level of the data aggregation operations. Figure 3-29 illustrates the control and data flow originated and received by the query scheduler.

The query scheduler has the following responsibilities:

- o Request initiation of the query processors
- o Request initiation of the data aggregators
- o Request initiation of the conceptual/external transformer
- o Dynamic selection of the data aggregation sequence
- o Monitoring of status messages sent by the data aggregators and query processors
- o Signalling to the application process (status, data)

The implementation of the query processors assumes that the query processors do not receive any intermediate results from data aggregators or other query processors. This simplification allows the parallel initiation of all query processors.

4. Distribute and Install Modules

Once the query processors have been generated, they must be distributed to their respective hosts, compiled, assembled and linked.

The compilation, assembly and linking steps are performed by the compiler, assembler and linker of the respective host where the module (query processor or data aggregator) is to run.

The source code for the modules are transmitted from the VAX to the target host via the Network Transaction Manager. The operational data controlling the data aggregators is distributed, at run time, via the NTM.

3.1.2.2.2.2 Functional Specification

The development of integrated Application Processes via the above scenario implies:

1. A schema integration methodology to setup the Common Data Model
2. A Neutral Data Definition Language to declare Common Data Resources in an Integrated Application Process
3. A Neutral Data Manipulation Language to retrieve data contained in heterogeneous databases
4. A precompiler to substitute the Neutral Data Manipulation Language statements with a set of cooperating procedures performing the data retrieval operations specified

The functional specifications of each of the above building blocks required to develop integrated Application Processes are as follows:

1. Schema Integration Methodology

   A methodology is required to define external and conceptual schemas in a flexible and unambiguous fashion.

   The methodology to be developed focuses on:

   - Degree of normalization required to build a conceptual schema that can be expanded/contracted with minimum impact on existing schemas (external, conceptual, and internal).
   - Guidelines for the efficient integration of existing databases.
   - Mappings between external, conceptual, and internal schemas.

2. Neutral Data Definition Language

   A Neutral Data Definition Language (NDDL) is provided by the Test Bed. Neutral refers to the non-database specific aspect of the data definition language provided by the Test Bed. The Test Bed NDDL is used to describe the various schemas of the integrated Test Bed databases.

   The Test Bed NDDL thus must support the definition of:

   - External schemas
   - Conceptual schemas
   - Internal schemas
   - The mappings existing between the schemas listed above
   - Relational schemas
   - Network schemas
   - Entities or relations
   - Attributes in a relation or in a record
   - Domain definitions compatible with COBOL

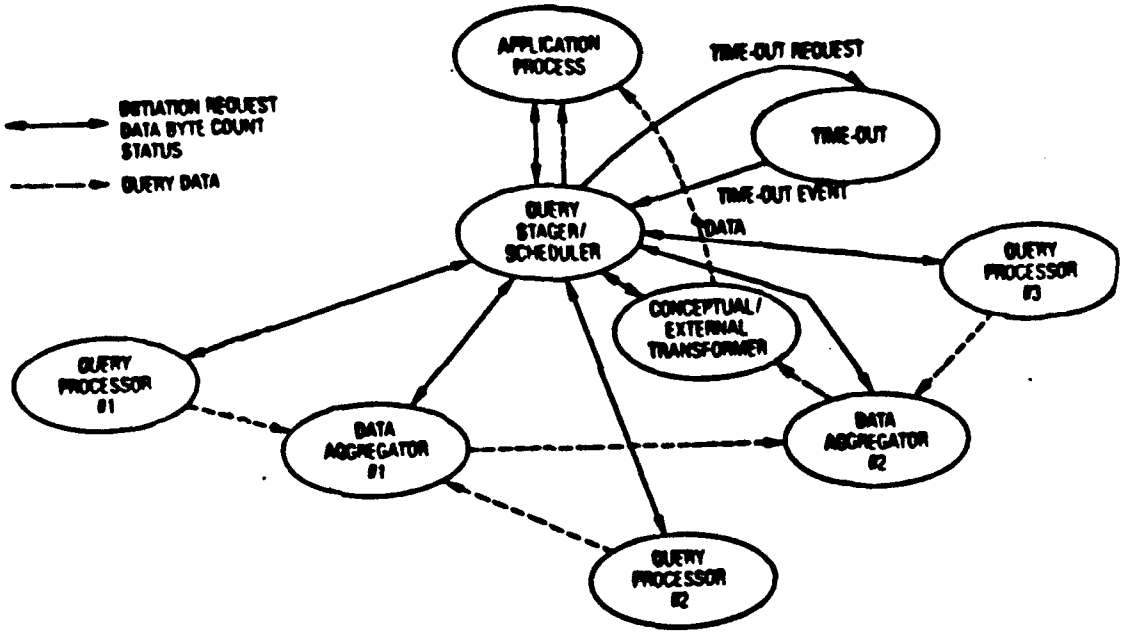


Figure 3-29. Distributed Query Scheduling & Control

- o Attribute ranges (numeric, Boolean)
- o External schema name
- o Hierarchical schema (or future enhancements)
- o Attribute unit and format representation
- o Derived attributes

External schemas are referenced by application processes. The external schema is not physically distributed, it is kept in the CDM and is referenced by the application process. This reference is used by the Test Bed precompiler at compile time.

3. Neutral Data Manipulation Language

A Neutral Data Manipulation Language (NDML) is provided by the Test Bed. The NDML is used to query the data contained in the databases integrated by the Test Bed.

The Test Bed NDML functional specifications are:

- o To be a relational oriented language
- o To be compatible with COBOL
- o To be non procedural
- o Each statement is delimited by easily recognizable characters
- o To support the retrieval of a set of tuples
- o To use variables definition compatible with COBOL
- o To return a status code (completion, error) upon execution
- o To support queries embedded in a COBOL program
- o To support retrieval queries qualified by the following predicates:
  - Equal
  - Not equal
  - Greater than
  - Greater or equal
  - Less than or equal
  - Less than
  - Boolean operator AND
  - Boolean operation NOT (FUTURE)
- o To support retrieval from one or more relations
- o To support at least retrieval queries constrained by at least 4 level of qualifiers
- o To support restriction (selection) operations
- o To support natural and equal join operations
- o To support projection operations

- o Not to cause a locking of the database when querying data
- o To use variable definitions compatible with FORTRAN
- o To support requests embedded in a FORTRAN program
- o To support update requests embedded in a COBOL program
- o To support insertion of a single tuple
- o To support modification of a set of tuples
- o To support the deletion of a set of tuples

The initial implementation of the 6201 NDML is not intended to support distributed data updates. However, the NDML syntax and implementation shall include update commands.

4. Application Process Precompiler

The Test Bed Precompiler performs the following functions:

4a. Prepass of COBOL Application Process Code

- o Prepass COBOL 74 Source Code
- o Recognize invoked external schema by names.
- o Recognize all NDML statements and replace these statements by equivalent comment statements
- o For each NDML statement:
  - Assign a unique number to the query scheduler
  - Insert a procedure call to the uniquely identified query stager/scheduler controlling the execution
  - Establish work area to receive the data in a COBOL compatible fashion

4b. Formulate Single Database Query

- o Obtain external schema from CDM
- o Transform external NDML query into conceptual NDML query
- o Formulate query decomposition based on data location
- o Decompose NDML conceptual query into NDML conceptual single database query

- o For each subquery:
  - Assign unique number to the subquery
  - Generate query processor

4c. Generate Query Processor

To generate the single database query processors implied by one NDML query, the precompiler needs to perform the following functions:

- o Formulate access path
- o Generate code for query processor or generate equivalent tables to perform required data retrievals
- o Insert internal schema - if required
- o Insert data format operations
- o Insert data to perform data integrity checks
- o Insert code to generate byte counts for retrieved information
- o Insert code to transmit retrieved information byte count and retrieval status to the query scheduler
- o Insert code to store retrieved data into a designated file
- o Specify the format of the data handed to the query processor
- o Use standard error messages and status information
- o Generate status code (error, completion) upon execution

In the 6201M implementation, the query processors are restricted to target the following database managers:

- o IDS II on Honeywell Level 6, operating under GCOS MOD400 V03 (Limited Tests)
- o IDMS on IBM 3081 under MVS
- o IDBMS on VAX for IDSS AP Cluster (IDSS 2.0) (Not Tested)
- o VAX-11 DBMS
- o ORACLE on VAX for the CDM AP Cluster

- o TOTAL on IBM
- o IMS on IBM (Limited Tests)
- o ISAM or VSAM (Future)
- o Other database management systems (Example DB2)

4d. Specify Data Aggregation Step

To specify the data aggregation step, the precompiler performs the following functions:

- o Formulate aggregation operations to be performed (equal join, not equal join, semi-join (future), and union)
- o Provide data aggregator operational information to the data aggregator
- o Specify the format of the data supplied by the query processors

The data aggregators use the data supplied by the precompiler to perform the following functions:

- o Formulate aggregation operations to be performed (equal join, not equal join, semi-join (future), and union)
- o Obtain required internal schemas to create work area to receive and process data
- o Provide error signaling logic (using standard error messages)
- o Obtain, aggregate and output data supplied by query processors
- o Generate the byte count of the aggregated data and report the byte count and status information to the query scheduler.
- o Accept format of data supplied by the query processors
- o Generate status code (error, completion) upon execution

4e. Specify Query Scheduling

To specify the query scheduling step, the precompiler performs the following functions:

- o Determine the possible data aggregation sequence and location

- Invoke the query scheduler from the Application Process

The query scheduler uses the data supplied by the precompiler to perform the following functions:

- Request initiation of query processors
- Request initiation of data aggregators
- Request initiation of conceptual/external transformer
- Select dynamically and control the data aggregation sequence
- Request time out service from the NTM
- Request the cancellation of all query processors, data aggregators, conceptual/external transformer, related to a given distributed query, in the event of errors
- Receive, interpret and report error messages received from the query processors, data aggregators and conceptual/external transformer

4f. Generate Conceptual External Transformer

To generate the Conceptual-to-External Transformer, the precompiler performs the following functions:

- Obtain the conceptual-to-external unit transformation from the CDM
- Obtain the conceptual-to-external format conversions from the CDM
- Invoke the conceptual-to-external transformer from the Query Scheduler

The Conceptual-to-External Transformer uses the data supplied by the precompiler to perform the following functions:

- Carry out the conceptual-to-external unit transformations
- Carry out the conceptual-to-external format conversion
- Report errors to the query scheduler

4g. Precompiler Characteristics and Constraints

- Precompiler is portable

- o Precompiler first written in COBOL 74
- o Precompiler is designed to allow extension to other non-COBOL high order languages
- o Precompiler initial implementation is the VAX, uses DML of CDM database manager and is controlled from native terminal
- o Precompiler issues error messages when required
- o Precompiler can be invoked by the NTM and make use of the NDML
- o Precompiler is structured to support ad-hoc query processing

3.1.3 The User Interface Configuration Item

3.1.3.1 User Interface Mission Statement

The IISS system requirement document identifies a need to provide users with an overall view of the system as an integrated application rather than as a collection of disjointed programs. Users need to be able to invoke the various Test Bed subsystems such as MCMM, MRP and IDSS from a single terminal in a consistent manner even though the applications may reside on a number of different computer systems.

Additionally, the User Interface should support the development of "User Friendly" application programs that are flexible and maintainable.

3.1.3.2 User Interface Functional Areas

The function of the Test Bed User Interface is two fold. First, the UI provides an environment that not only allows but encourages good user interface design. Secondly, the UI provides a run time environment that supports interactive dialogue. These two subsystems are called the User Interface Development System and the User Interface Management System. The various functional areas making up the UIDS and UIMS are shown on the User Interface Configuration Tree given in Figure 3-30.

Figure 3-30 shows two major functional areas:

- o User Interface Development System (UIDS)
- o User Interface Management System (UIMS)

The UIDS is a set of tools that assist application developers with forms maintenance, report writing and rapid application generation. These tools are available to the user as part of the UIMS.

The User Interface is a forms based system. This means that all communication between an application program and the user is done through electronic forms on a video terminal screen. Electronic forms are full screen template displays that act as prompts or memory joggers to users as they fill in input data items.

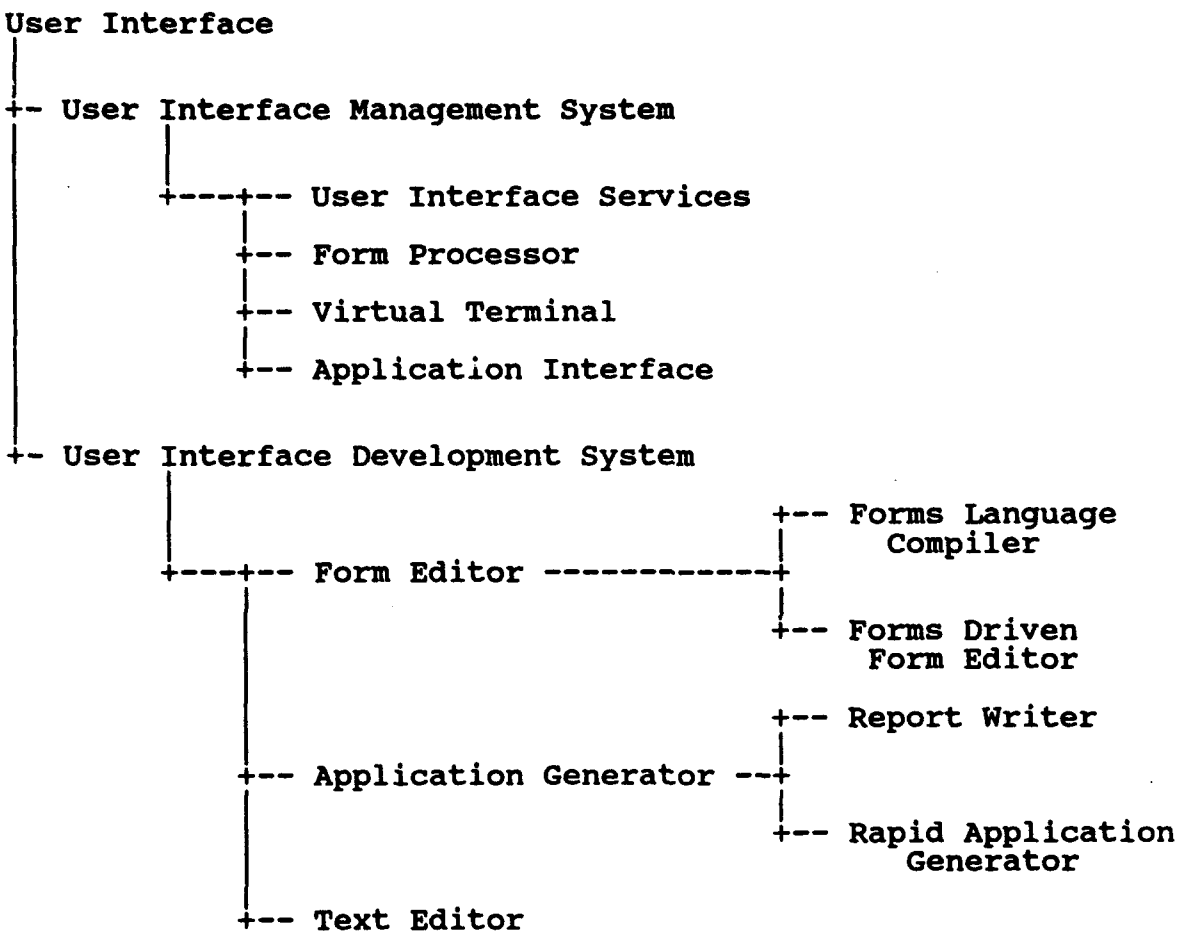


Figure 3-30. User Interface Configuration Tree

The UIDS provides two facilities for editing forms:

- The forms can be edited using a language called the Forms Definition Language which is compiled with the Forms Language Compiler.
- The forms can be edited through the use of an interactive application that prompts the user for form characteristics and allows for freestyle form layout.

The UIDS Report Writer and Application Generator are based on extensions to the Forms Definition Language for program generation.

The UIMS is the run time UI. It performs the functions which are interactive. As such, the UIMS interfaces with the user through an interface to the Virtual Terminal or neutral Test Bed terminal. The UIMS interfaces with application programs through the Application Interface to the Form Processor which handles screen processing commands. The Application Interface is bound to the Application Process and contains the necessary functions to extract data from a form sent to it, to fill out a form, and to initiate messages to the Form Processor as well as receive messages from the Form Processor.

Figure 3-31 illustrates the above concepts. This figure shows that the transactions exchanged between the Form Processor and the Application Interface are handled by the NTM. A description of the NTM itself can be found in section 3.1.1 of this System Design Specification.

3.1.3.3 User Interface Operational Scenerios

The User Interface Operational Scenarios presented here are introduced for the sole purpose of supporting the identification of the functional specifications to be met by the User Interface. These scenarios are not meant to imply a specific implementation of the functional specifications. Consequently, the final design may implement scenarios which differ from the scenarios in this section.

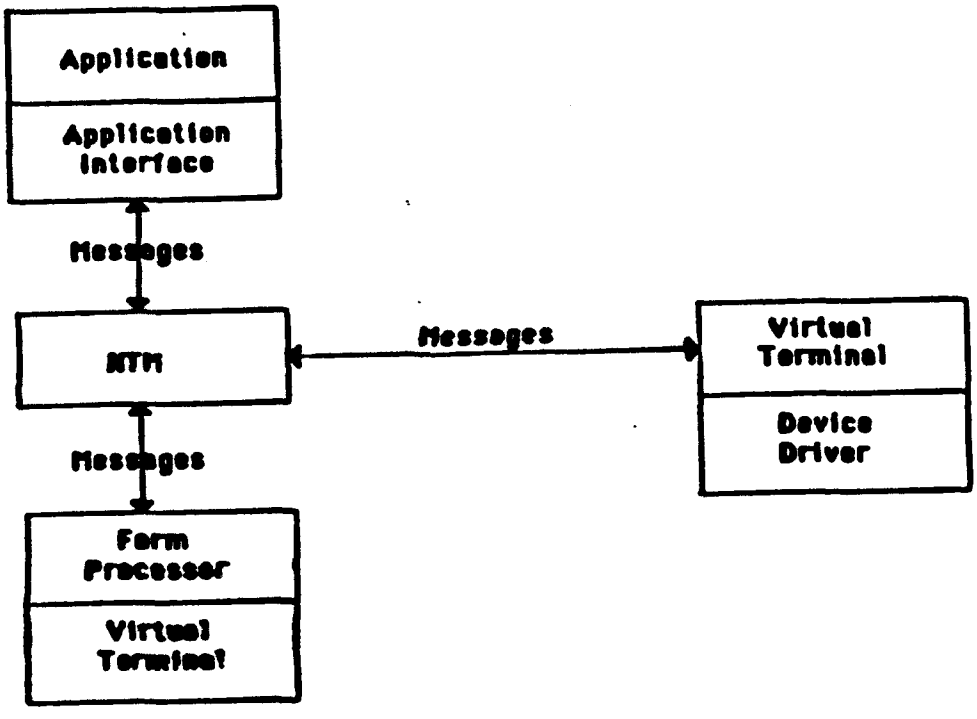


Figure 3-31. User Interface Management System

The Test Bed User Interface supports the following modes of operation:

- o Forms Mode
- o Form Editing Mode
- o Application Generation Mode

In the following sections is a description of these three (3) modes of operation.

3.1.3.3.1 Forms Mode Operational Scenario

The forms mode allows the displaying of predefined forms for the purpose of requesting data from the user or for the purpose of presenting data to the user.

Thus, forms can be used to display data, to input data or to control the execution of Application Processes on the Test Bed.

The forms which are displayed have been predefined and catalogued on the CDM. Each form can be uniquely retrieved by its name or ID.

The following definitions are convenient when discussing the Test Bed forms mode of operations.

- o A Form is viewed as a template defining fields and their attributes. The strength of the Test Bed forms processor is derived from the single and shared definition of all fields and variables referenced by forms. The forms are defined in the Common Data Model. (Future)
- o Form data are the actual values contained in the fields of the form. These values may either be input interactively by the user or may be supplied by the Application Process as output data. The Test Bed forms processing uses the attribute definitions available from the Common Data Model to perform data integrity checking at the level of the User Interface. This approach offers the following advantages:
  1. Errors are detected at the data entry point, where they may be corrected
  2. Errors are not propagated through the system
  3. Data integrity checking is consistent and transparent to the application
- o A form instance is a filled out form, that is the collection of a form and a single set of form data.

This definition can take two different representations; when viewed by the user a form instance is defined as above, when viewed by an Application Process a form instance reduces to form control data, form data, and form name. The format of the data is available to the Form Processor.

Displaying a Form To Be Filled Out

Consider Figure 3-32. A form to be used for user input is placed on the list of forms to be displayed (Display List). The form to be displayed is referenced by its form name. Frequently used forms may be downloaded from the CDM at start up time and placed on the list of open forms (Open List). Less frequently used forms are requested on an ad-hoc basis from the CDM. An error code is generated and routed to the Application Process when a requested form cannot be found.

Filling Out a Form

Consider Figure 3-33. The form to be filled out is presented on the IISS terminal. The user fills out the form, to the best of his abilities. In doing so, the user may perform some local editing procedures to correct typos or other errors. Typos are corrected by invoking the special function keys (backspace, etc.) defined and supported by the host terminal driver. When the form is filled out to the satisfaction of the user, the ENTER key is depressed on the terminal. This causes the data integrity checks to be performed.

Invoking the Help Facility

When filling out a form, the user may invoke the Help Facility. Help may be a message or an entire form. After invoking the Help Facility, the user may return to the original form.

View of Forms (FUTURE)

Form views allow customization of forms. A view is a form whose fields map to those of another form (the base form). If a field is changed on the view it is also changed on the base form and vice versa. However, the attributes of the view override those of the base form. This allows the view to have different initial values.

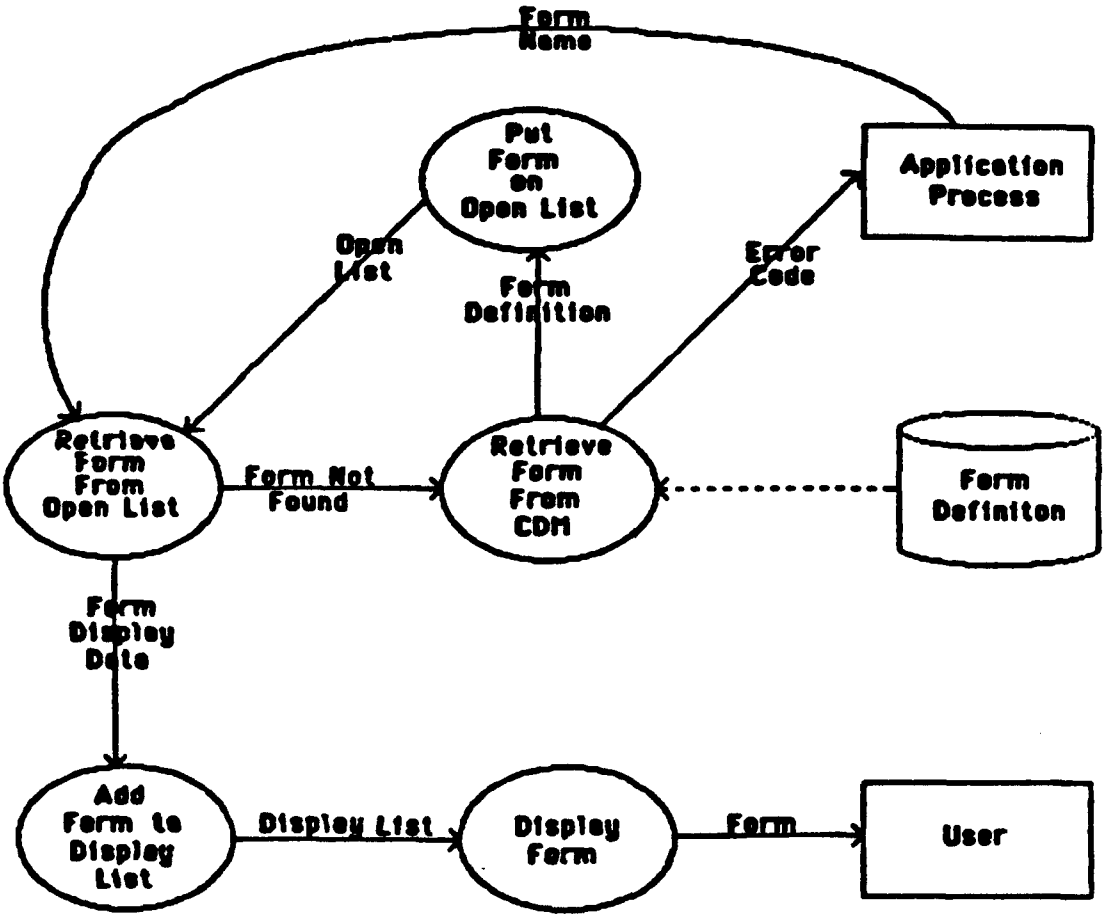


Figure 3-32. Display/Retrieve a Form

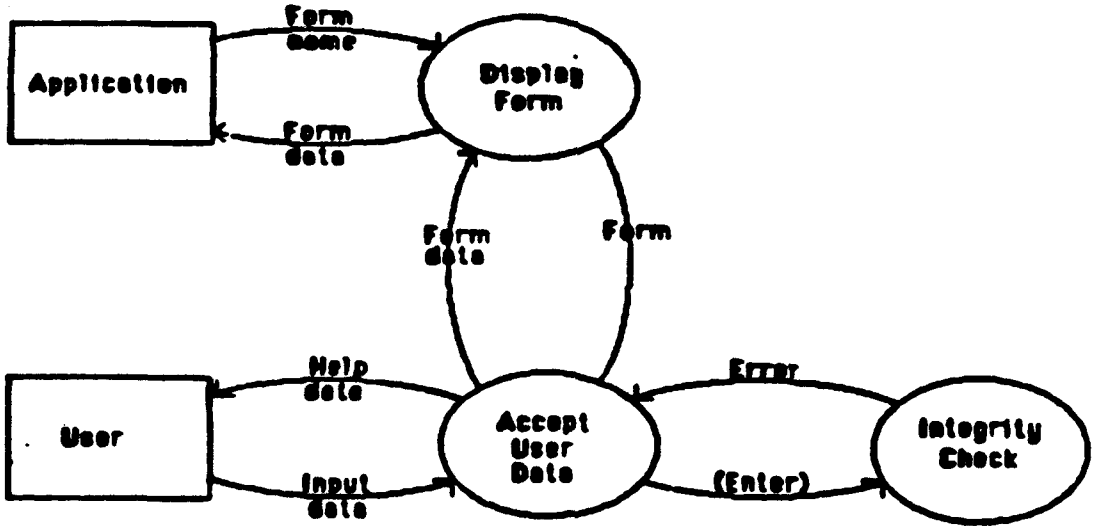


Figure 3-33. Filling out a Form - User Viewpoint

Transmitting a Form

Upon completing the filling out of a form, the user indicates his willingness to transmit a form by depressing a function key. This action causes the data integrity checks to be performed unless the function key depressed indicates that the user would prefer to quit processing the form. The data integrity checks are supported by the data contained in the form itself. This approach eliminates the need for additional CDM access. Error messages are issued to the user when appropriate. Only error free forms are transmitted, thus keeping network traffic to a minimum. The Test Bed is a message driven system. Data is transmitted from the User Interface to the Application by means of a message.

Outputting a Form Instance

Data generated by an Application Process may be output as form data associated with a form. This form instance is displayed at the request of the Application Process. When the user has viewed the data, he may signal his willingness to view more data by entering a (next form) control character. This action allows more data to be displayed on the terminal.

Outputting a Form To Be Filled Out from an Application

The Application Process which has needs for input data may make its needs visible to the user by requesting that a form be displayed and filled out by the user. The specific Application Process issues a "display form" command for that purpose. The "display form" command contains the name of the form to be displayed.

Controlling the User Interface

The User Interface assumes one of three major states:

- ACCEPT USER DATA
- ACCEPT APPLICATION DATA
- PROCESS FORMS

  The PROCESS FORM state itself is made up of the following states:

  - DISPLAY FORM
  - ACCEPT USER DATA
  - INTEGRITY CHECKING
  - INITIATE MESSAGE

These states are shown on Figure 3-34.

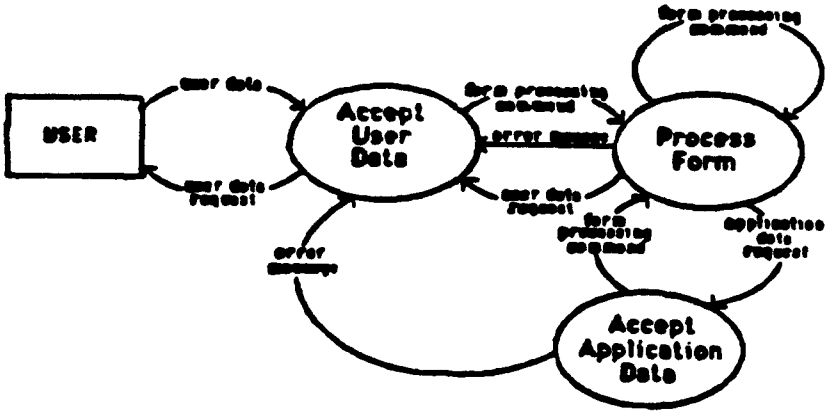


Figure 3-34. Control Diagram - Form Processor

Upon cold start or restart, after completing the startup procedure, the User Interface enters the ACCEPT USER DATA state and awaits user input. The PROCESS FORM state is entered whenever a command requesting form processing is received from the user or from the Application Process. Such command indicates the number of the form to be displayed or otherwise processed. Normal exit from the PROCESS FORM state can take three forms:

- Return to the ACCEPT USER DATA state
- Return to the ACCEPT APPLICATION DATA state
- Re-entry of the PROCESS FORM state

This last eventually allows the chaining of forms, as is encountered in menu processing. The information controlling normal exit from the PROCESS FORM state is either contained in the form being processed or is contained in the commands sent to the form processor by the user or by the Application Process. This approach is flexible and simple. Abnormal exit from the PROCESS FORMS and ACCEPT APPLICATION DATA must be accommodated. This is necessary in the event of a malfunction of an Application Process or in the event of a change of mind from the user. Abnormal exit from either state leads to the ACCEPT USER COMMAND state, leading the User Interface ready to accept user inputs and commands. The exit command is supplied by the user and acts as an interrupt.

The Run Time User Interface

Figure 3-31 depicted the interaction of the User Interface with an Application Process. As seen in that figure, an Application Process places calls to the User Interface Application Interface to which it is linked. In return it receives data. Requests made by the Application Process are

transformed into messages by the Application Interface routines. These messages are transmitted by the NTM to the Form Processor, which services the requests as explained in the previous paragraphs. Forms instances or user data are, in return, transmitted as messages by the Form Processor over the NTM. These messages are then interpreted by the Application Interface and the data is extracted in a form suitable to the Application Process. To reduce network traffic to a minimum, a complete forms instance is transmitted, and the data is kept by the Form Processor. Thus, subsequent calls by the Application Process on data available to the Form Processor can be answered locally without retransmission over the network.

This run time portion of the User Interface is called the User Interface Management System.

3.1.3.3.1.1 Form Processor Operational Scenario

Several operational scenarios are contemplated for the Test Bed Form Processor.

Cold Start/Restart Scenario

The Form Processor is cold started or restarted by a device driver following a User Logon request. The Form Processor then is ready for operation and displays the first logon form and enters the "ACCEPT USER COMMAND/DATA" state. The above concepts are illustrated on Figure 3-35 entitled "Form Processor Start Up".

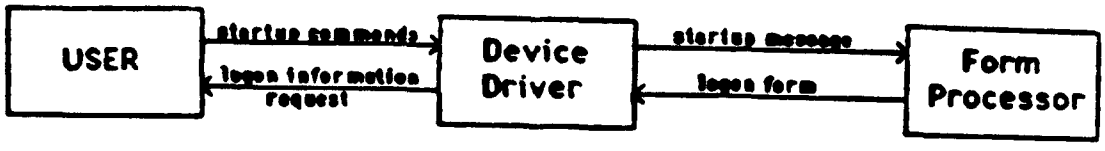


Figure 3-35. Form Processor Start Up

User Log On Scenario

Test Bed User Log On/Log Off scenario is viewed as a particular case of conducting data integrity checking on a form instance. In this case, the form used is a log on/log off form, and the form data is related to the log on/log off operations. When processing the logon form, the Form Processor requests CDM data from the CDM by sending a message requesting the start up data to the CDM request processor. The CDM contains the profile

data for each user. This table defines the password and roles that are valid for a user-i.d. The table is created and maintained by the CDM Administrator and is stored in the CDM. Failure to log on prevents the user from gaining access to the forms used to control the Application Processes and Test Bed services. These concepts are shown on Figure 3-36.

3.1.3.3.2 Form Edit Operational Scenario

The forms used by the User Interface are predefined and stored in the CDM. These forms are retrieved and transmitted to the Form Processor when a request is made for them by the application. The Form Editor is used to define forms and to control the storage, editing and deletion of forms which have been already defined. The Forms Driven Form Editor makes use of forms to facilitate the definition of new forms and to control maintenance operations on the form database segment of the CDM. Access to the Form Editor is controlled, as is the case of any program having CDM update privileges. The Form Editor is operated by any application developer with sufficient authority.

The following information is required to define a new form:

a. The name to be given to the form

b. The format of the form

c. The names of forms used within the new form (if any)

d. The fields contained in the form which are common data

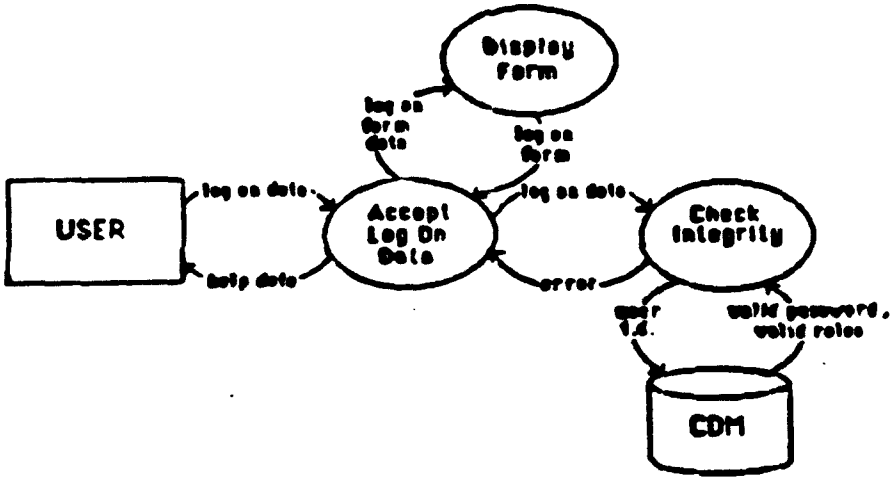


Figure 3-36. User Log on Scenario

With the above information, the application developer first identifies an existing schema or creates an external schema, usirg the Test Bed Neutral Data Definition Language (NDDL), to reference the fields to be defined to their respective CDM definitions. This linkage ensures an error free and effective

transfer of the data constraints associated with each element. Second, the application developer creates a new form description which is uniquely named, within the application. The new form description contains:

a. The name of the external schemas to be referenced

b. The dimensions of the form

c. For each subform invoked:

   - o The name of the subform
   - o The position (x,y) of the top, left corner of the subform

d. For each new field invoked:

   - o The name of the variable
   - o The position (x,y) of the left most characters of the variable
   - o The graphic attributes associated with the field
   - o The name of the indirect form associated with the field (help, etc.)

e. For each textual element included:

   - o The character string to be shown
   - o The position (x,y) of the left most character
   - o THE GRAPHIC ATTRIBUTES ASSOCIATED WITH THE TEXTUAL field

f. The name of the next form to be displayed or whether control is returned to the ACCEPT USER COMMANDS state or to the ACCEPT APPLICATION COMMANDS state (Future)

g. Security information associated with the form, that is, the role of the user having access to the form

The position (x,y) of each field can either be defined by its relative position or graphically (graphic definition: future). The Form Editor can be invoked from an IISS terminal. The Form Editor communicates with the CDM processor via the NTM. The CDM processor performs the CDM updates, edits.

3.1.3.3.3 Application Generation Operational Scenario

Application programs can be described using an Application Definition Language (ADL). The ADL is similar in syntax to the Forms Definition Language (FDL) and contains both FDL and Neutral Data Manipulation (NDML). The ADL can be compiled to produce an application program that accesses the CDM via forms. Figure 3-37 describes the application generation.

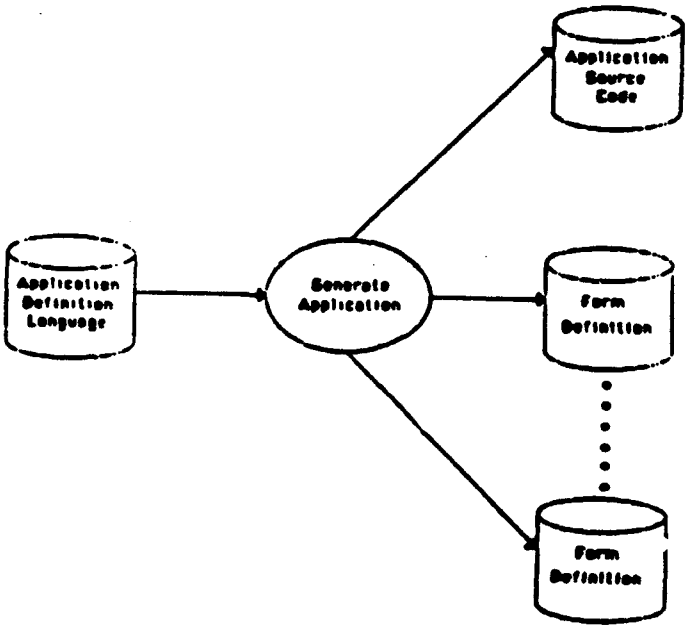


Figure 3-37. Application Generation

A report program is a special kind of program generated by the Application Generator. Paging and other formatting characteristics of the report depend on the data to be displayed within it. The language used to describe reports is called the Report Definition Language (RDL) and the portion of the Application Generator used to compile the RDL is called the Report Writer.

3.1.3.4 Functional Specifications

The functional specifications implied in the scenarios presented in Section 3.1.3.3 are identified and presented in this section.

3.1.3.4.1 User Interface Management System

Form Processor Start Up

Mission: To ready the Form Processor for user interaction upon user logon to the VAX host under an IISS account.

Functional Specifications:

1. To initialize the Form Processor, and all modules associated with it.

2. To initiate a message to the NTM. This message identifies the Form Processor.

3. To display the first form to be displayed on the Terminal.

4. To place the Form Processor in the "Accept User Command" State

User Logon

Mission: To ensure that only authorized users may gain access to the Test Bed Command forms.

Functional Specifications:

1. To display the Test Bed logon form. The Test Bed logon form requests that the following data be supplied:

   - o User-i.d.
   - o User role
   - o User password - not echoed

2. To validate the above information against access data supplied by the CDM

3. To reject unqualified answers by repeating step one

4. To record user statistics for qualified users

5. To display the first page of the Command selection menu

6. To place the Form Processor controller in the ACCEPT USER COMMAND state

Form Processing

Mission: To perform the operations required to process forms:

- o Retrieval of a predefined form
- o Display of the form
- o Filling out of the form (from the user point of view)
- o Data integrity checking
- o Help facility
- o Window management
- o Text editing

Functional Specifications:

1. Retrieval of a Predefined Form

   a. Accept the name of the form to be retrieved. The name of the form to be retrieved is supplied by an Application Process.

   b. Determine whether or not a given form name is stored in the Form Processor forms table.

   c. Retrieve a given form name from the Form Processor forms table.

   d. Query the CDM to request the downloading of a form. This capability is required whenever the form is not available locally. The form to be downloaded is specified by its name.

   e. Formulate and display appropriate error messages, if necessary, on the terminal.

   f. Clear form

2. Display of a Predefined Form

   a. Display a Blank Form - A form which is intended to be filled out by the user is first retrieved as explained in the operational scenario.

   b. Display of a Form Instance - A form instance is created by first retrieving the blank form as explained above, and by adding to it the data defined by the user or by the Application Process.

3. Filling Out a Form by the User

The form to be filled out is first displayed (see Display of a Blank Form - item 2a above). The user designates the field he wishes to fill by positioning the terminal cursor in the left most character position of the field to be filled. Characters entered by the user are added sequentially in the field.

   a. Cursor Position Control - The user may fill the field in the sequence he desires. To start entering data, the user positions the cursor in the first character position of the field to be filled out. The following cursor motions are supported:

      o One space motions (right, left, top, bottom)

      o Tab control

      o Next line

      o Beginning of line

   b. Data Entry - The user then types in the data, on a character basis.

The Form Processor detects the following conditions:

- o Field overflow
- o Data entry outside a field

The Form Processor recognizes a special character which indicates that the user has completed the data entry operations.

4. Data Integrity Checking

Once a form has been filled out by the user, it is checked for integrity. The following capabilities are provided:

a. Edit Check Each Field - Each field entered on the form is checked against the edit constraints defined for it in the External Schema attached with the form. The edit checks which are performed include:

- o Numeric fields
- o Alpha fields
- o Predefined number of characters

b. Range Check Each Field - Each field which has passed the edit check is then checked for range. Upper and lower limits of each data item is defined in the External Schema attached with the form. The range checks which are performed include:

- o Upper limit check
- o Lower limit check

Ranges are not restricted to be continuous. Range information may include combinations of continuous and discrete values. Ranges may be defined as tables of authorized numeric or non-numeric values.

c. Flag Errors - The errors detected by the edit check and range checks are reported to the user. This is done by signalling out the field(s) in error. The edit checks and range checks are conducted on the entire form before control is passed to the Application Process.

d. Form Processor Control - The Form Processor is controlled as follows:

- o No errors detected - control is given to the MESSAGE INITIATOR state, and the process of initiating a message begins
- o Errors have been detected - control is given to the ACCEPT USER DATA state and the user is given the opportunity to correct his errors.

5. Help Facility

A help form is a form associated with another form or field which provides additional information or explanation of the data to be entered.

a. Invoking the Help Form - A means by which the user can invoke the help forms associated with the form or with the given field within the form is provided. The operational procedure shall, in either case, be similar.

b. Nesting Levels - A help form may be associated with any form, even if it is a help form. However, for practical considerations, five level nesting is considered.

c. Returning to the Original Form - A facility shall be provided to enable the user to return directly to the original form from any level of help, without backtracking through the help forms invoked.

d. Defining Data on a Help Form - Data may be directly entered on the help form, and automatically be inserted on the original form. (FUTURE)

6. Window Management

Windows are spaces that are rectangular in shape that are reserved for form placement. Windows are contained in forms. Application Processes and users manipulate windows.

Application Processes must be able to perform the following activities at run time:

a. Add a form to a window

b. Replace a form in a window

c. Remove a form from a window

Users must be able to perform these activities at run time:

a. Scroll a form within a window

b. Move a window

c. Shrink a window

7. Text Editing

Users must be able to perform these text editing functions on data items:

a. Cut and paste

b. Global search and replace

c. Repeat

d. Set margin

Message Initiation

Mission: To initiate the messages required to:

a. Request data from the CDM

b. Send data to an Application Process

c. Send data to the Virtual Terminal

Functional Specifications:

1. Formulate the following CDM requests:

   - o Request for user profile data (password, role)
   - o Request for specific form data

2. Formulate the following Application Process requests:

   - o Upload of the form data, form name to the Application Process
   - o Upload of the Form Processor status and error conditions

3. Formulate the following Virtual Terminal requests:

   - o Request for user entered data
   - o Request for cursor position
   - o Request for function key pressed by user

4. The messages formulated by the Message Initiator shall be in a well defined and structured format. The format exhibits:

   - o A header
   - o A data body

5. Message header contains the information required to control the User Interface Management System. The data body contains the information which may be required to support processing.

Form Processor Monitor

Mission: To control the operations of the Form Processor and to coordinate the actions of the Form Processor in response to:

a. User commands

b. Application Process Commands

Functional Specifications:

1. The Form Processor Monitor allocates control of the Process Forms functions to the Virtual Terminal, or to the Application Process.

2. The Form Processor Monitor allows the Virtual Terminal to regain control of the Form Processor at anytime. This is done on an interrupt like basis. In this event, the Application Process, if any, is terminated.

3. The Form Processor Monitor waits on an acknowledgment from the Virtual Terminal to go to the next form or next state. Standard function keys are defined for that purpose.

4. The Form Processor Monitor recognizes a set of function keys (or combination of keys). These keys are consistent, and always produce the same effect. The following keys are recognized:

   o Quit

   o Enter

   o Help

Application Interface

Mission: To facilitate the control and the manipulation of forms and forms data by an Application Process.

Functional Specifications:

1. Control of the Form Processor

   The Application Interface facilitates the control of the Form Processor by the Application Process. To that effect, the Application Interface offers a set of commands which can be invoked by the Application Process. The set of commands supported by the Application Interface is given below:

- Remove form
- Display form
- Display form instance
- Read form
- Read form instance
- Display error message

2. Data Manipulation Primitives

In addition, the Application Interface allows the manipulation of data forms by the Application Process. To that effect, the Application Interface supports the following data manipulation primitives:

- Read field value by name
- Read previous value by name
- Write field value by name
- Query field attribute by name
- Set field attribute by name

The above operations are performed locally by the Form Processor which maintains a local copy of the form instance displayed by the Virtual Terminal.

3. Stored Forms Instance Primitives

Form instances can be saved locally. This capability provides memory to the Form Processor. Previously defined data can be selectively retained for further processing. To that effect, the Application Interface supports the following stored forms instance operations:

- Save form instance (future)
- Get form instance
- Delete form instance (future)

3.1.3.4.2 User Interface Development System

Forms Definition

The definition of the forms include the following sections:

1. Forms Identification

- Application name
- Form name

The system wide form name is made up by the concatenation of the application name and of the form name.

2. Form Control Definition

   - Name of the help form associated with this form
   - Relative position of the form on the screen, with respect to the origin: Origin is taken to be the top, left corner of the screen, for the top level form. The location of the origin of any other form is controlled by the containing form.

3. Textual Element Definition

   - Relative position of the top left character of the Textual element being defined.
   - Length and width of the text being defined (length and width are computed automatically).
   - Display attributes of the Textual element.

4. Field Definition

   - Length and width of the field
   - Relative position of the field top left most character with respect to the origin of the form in which the field is defined.
   - Name of the variable to be stored in the field. Variable is referenced by its name as shown in the External Schema.
   - Display attributes granted to the field.
   - Field name (optional).

5. Field Information for an Existing Field

   - Set a field equal to another (defined) field.

6. Prompt Field

   - Name of textual field to be prompted, or
   - Name of the field to be displayed

Form Editor

The following functions are provided by the Form Editor:

1. Define Forms

   Forms definition is forms oriented. A form is predefined for each of the sections of a form:

   - o Form identification
   - o Form control information
   - o Textual element definition
   - o Field definition
   - o Prompt field definition

2. Add Field

   Fields can be added to an existing form by using the add field capability. The add field utility allows the definition of a field as defined previously.

3. Delete Field

   A field defined in an existing form can be deleted via this utility.

4. Set Field Attribute

   An attribute of a field can be set (or reset) by this utility. The user needs to define the name of the attribute and its value (Boolean, integer).

5. Save Form

   This utility is used to save on secondary storage (in the CDM) a form which has been defined by the user.

6. Get Form

   This utility is used to retrieve a form. The form may either be retrieved from the CDM or from the Form processor table (if available).

7. Delete Form

   This utility is used to delete a form stored in the CDM.

8. Error Processing

   The Form Editor provides the following type of error processing:

   - o Check for uniqueness of names of forms
   - o Check for dimension constraint

- Check for undefined attribute constraints

The Form Editor is viewed as any other process using the forms capability of the User Interface. Hence, it can also invoke all of the following Active Instance operations:

- Show form
- Clear form
- Query field attribute
- Save instance
- Get instance
- Delete instance

Report Writer

Mission: To provide a means to translate textual definitions of reports into programs that generate the reports.

Functional Specifications:

1. Textual Output

   The background template for the report is a form. It is specified using the Form Definition Language.

2. Database Operations

   All database operations are described using the Neutral Data Manipulation Language (NDML) and are performed by the CDMP.

3. Statistical Summarization

   Simple statistical computations may be performed upon item values and included in the output. The computations include:

   - Count
   - Sum
   - Average
   - Minimum

4. Picture Specifications

   Each item field that is mapped to CDM data may have a picture specified to define the output format. The editing provided includes numeric and alphanumeric specification, leading zero suppression, decimal

placement, leading sign indicators, currency symbols, and placement of embedded commas.

5. Exceptional Conditions

Tests can be established for certain exceptional conditions. An action or group of actions can be specified to occur when the condition arises. Practical applications of this facility include output of headers, footers, statistical summaries and paging.

The following exceptional conditions are recognized:

- o Change in the value of an item
- o Page overflow
- o Startup of report

6. Exception Actions

The following actions can be taken when an exceptional condition occurs:

- o Page ejection
- o Change of form
- o Setting of a data item
- o Database query

Rapid Application Generator

Mission: To provide a means to translate textual definitions of interactive database applications into programs that access databases via the CDM.

Functional Specifications:

1. Application Definition

The application is defined by using an Application Definition Language that subsumes the Forms Definition Language.

2. User Interaction

The user of the generated application may perform the following activities:

- o Filling out form templates
- o Menu picking
- o Item selection by cursor position

- Function key Selection

As a result of the user interaction the following activities are performed by the generated application:

- Switching between functions
- Database modification

3. Database Operations

All database operations are described using the Neutral Data Manipulation Language (NDML) and are performed by the CDMP.

4. Security and Access Control

Security and access control is provided through the mechanism of conditional forms and triggers sensitive to particular item field values. Thus, the functionality offered to a particular user can be restricted by not displaying a form containing options for certain database operations or by not enabling certain function keys. Further, other forms informing the user of restrictions on access can be displayed instead.

5. Conditional Actions

Conditional actions are those triggered by the occurrence of an event defined in an ON statement. Any number of actions can be triggered by a single event. They involve both form processor actions and database transactions. It is primarily the conditional actions which determine the course of the execution of the application.

3.1.4 The Virtual Terminal Interface Configuration Item

3.1.4.1 Virtual Terminal Mission Statement

The mission of the Virtual Terminal Interface is to afford terminal independence to the Application Process and Test Bed System Services. More specifically, the Virtual Terminal Interface provides:

- Terminal protocol independence
- Terminal character independence
- Terminal feature independence to the Test Bed Applications and Services.

3.1.4.2 Virtual Terminal Functional Areas

The various functional areas making up the Test Bed Virtual Terminal Interface are shown on the Virtual Terminal Interface configuration tree given on Figure 3-38.

Figure 3-38 shows two major functional areas:

- o Virtual Terminal Definition
- o Virtual Terminal Implementation

Virtual Terminal Definition

The Virtual Terminal defines a set of characters, terminal features and data exchange protocols which are considered standards in the Test Bed System. Any Application Subsystem or Test Bed Service written specifically for the Test Bed assumes that it is interfaced to a terminal offering the character set, terminal features and data exchange protocols defined for the Virtual Terminal Interface.

Virtual Terminal Implementation

The Test Bed Applications must however ultimately communicate with a hardware terminal. It is the function of the Virtual Terminal Interface (VTI) to provide the necessary character and protocol conversion procedures. Likewise, the VTI provides the mappings that may be required to implement any Virtual Terminal Features on a specific hardware terminal.

3.1.4.3 Virtual Terminal Operational Scenarios

The Virtual Terminal Interface operational scenarios presented here are introduced for the sole purpose of supporting the identification of the functional specifications to be met by the Virtual Terminal Interface. These scenarios are not meant to imply a specific implementation of the functional specifications. Consequently, the final design may implement scenarios which differ from the scenarios shown in this Section.

3.1.4.3.1 Interface to a Real Terminal

The most natural role for the Virtual Terminal is to interface with a Real Terminal. This scenario is illustrated on Figure 3-39. This figure shows the VTI interfacing with the

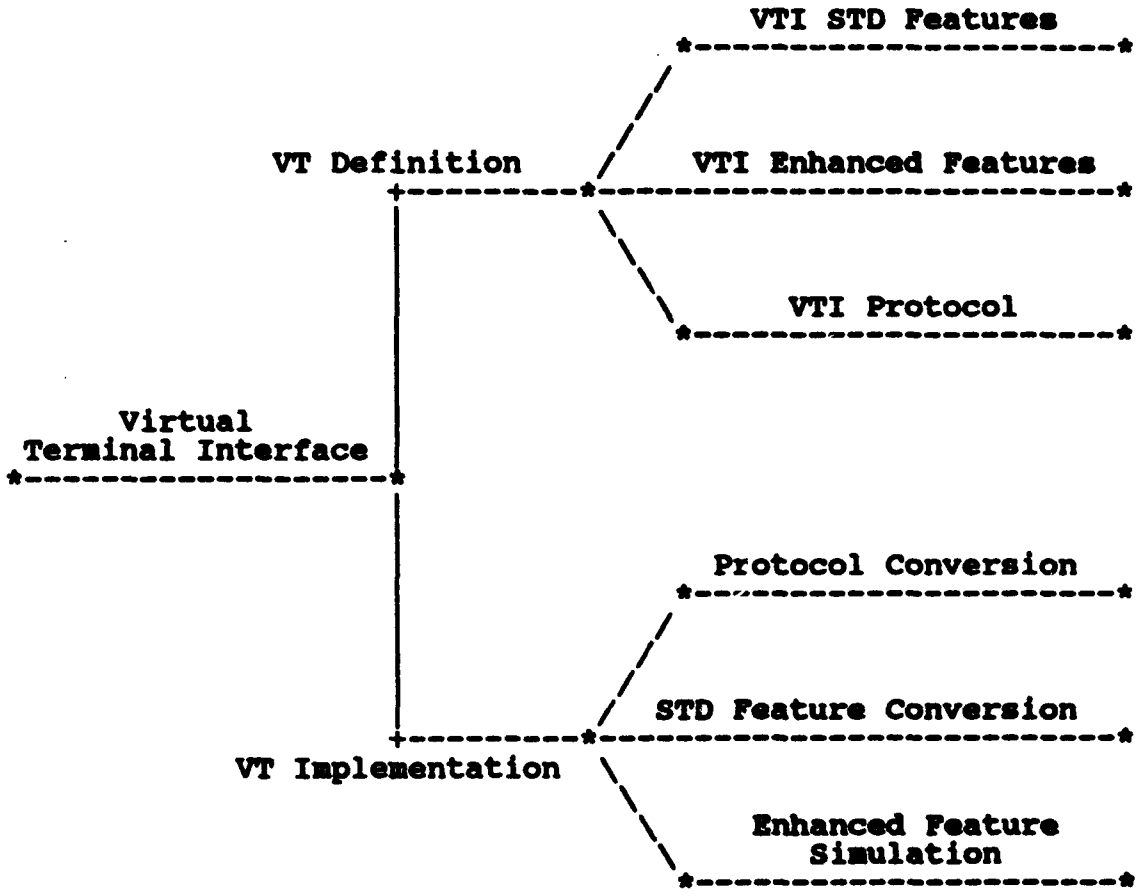


Figure 3-38. VTI Configuration Tree

hardware terminal through the terminal driver provided by the host operating system. The VTI is also interfaced with the User Interface Form Processor.

In this role, the Virtual Terminal performs the following functions:

1. Data Acquisition from the Real Terminal

   The VTI receives data from the real terminal according to a protocol. This protocol can either be character, line, or block oriented.

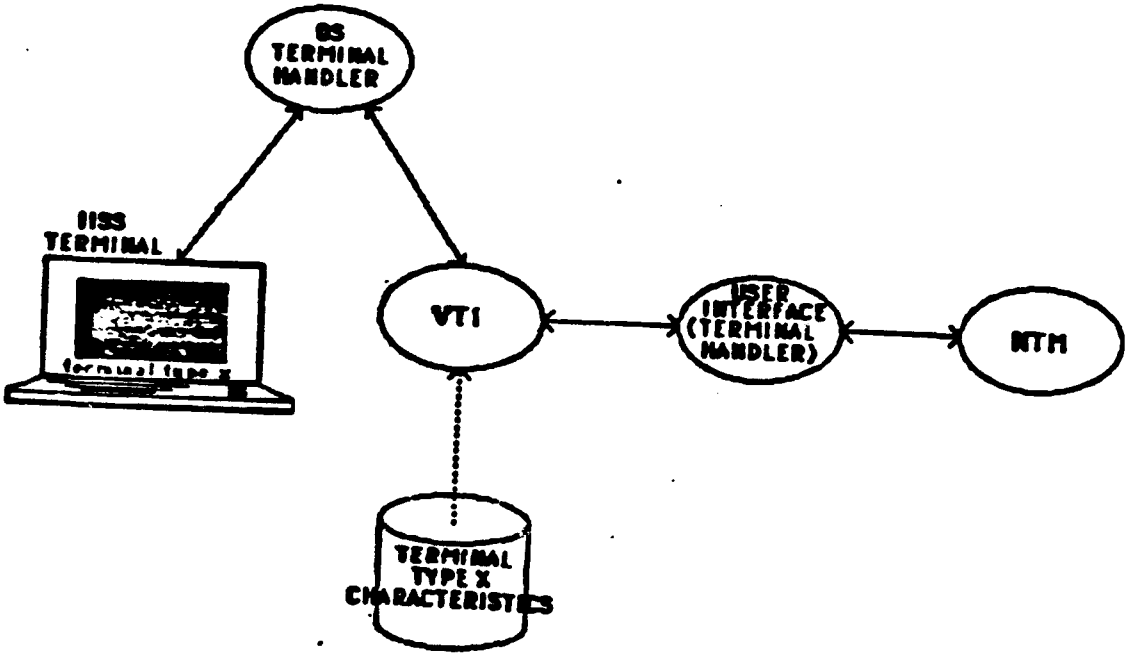


Figure 3-39. Interface to a Real Terminal

2. Data Conversion from Real Terminal to VTI

   Once the VTI has acquired data from the Terminal Handler, it may proceed to convert the characters received into the VTI character set. At this point, it must be noted that character conversion cannot be assumed to be context free. Some terminals (like the VIP 7200) use escape sequences which are several characters long. The sequence of characters indicates the type of the user interrupt, and the characters included in the sequence are context dependent.

3. Data Transmission to the User Interface Form Processor
   The VTI data, which has now been converted to the VTI character set, with all terminal dependent character sequences eliminated is then transferred to the User Interface. This data transfer is done on a pre-negotiated protocol (character, line, block). The data is now punctuated by control characters taken from the set of VTI control characters. These control

characters are used to indicate the occurence of things such as: line feed, carriage return, cursor control, etc.

4. Receiving Data from the Form Processor

The Virtual Terminal receives data from the User Interface Form Processor whenever the User Interface Form Processor wishes to communicate with the hardware terminal. The Virtual Terminal Interface is invoked to that effect by the Form Processor. The data supplied by the Form Processor is, by definition, in the standard Virtual Terminal format.

5. Data Conversion from VTI to Real Terminal

The data presented to the Virtual Terminal Interface is converted so the character set of target hardware terminal and the VTI control characters are replaced by the control characters of the target terminal. Once this conversion has been completed, the VTI invokes the Terminal Handler of the host operating system. Figure 3-40 shows the mechanics involved in acquiring, converting and transmitting data to the User Interface Form Processor. This figure assumes that the VTI is not directly controlling the I/O Terminal Handler. The

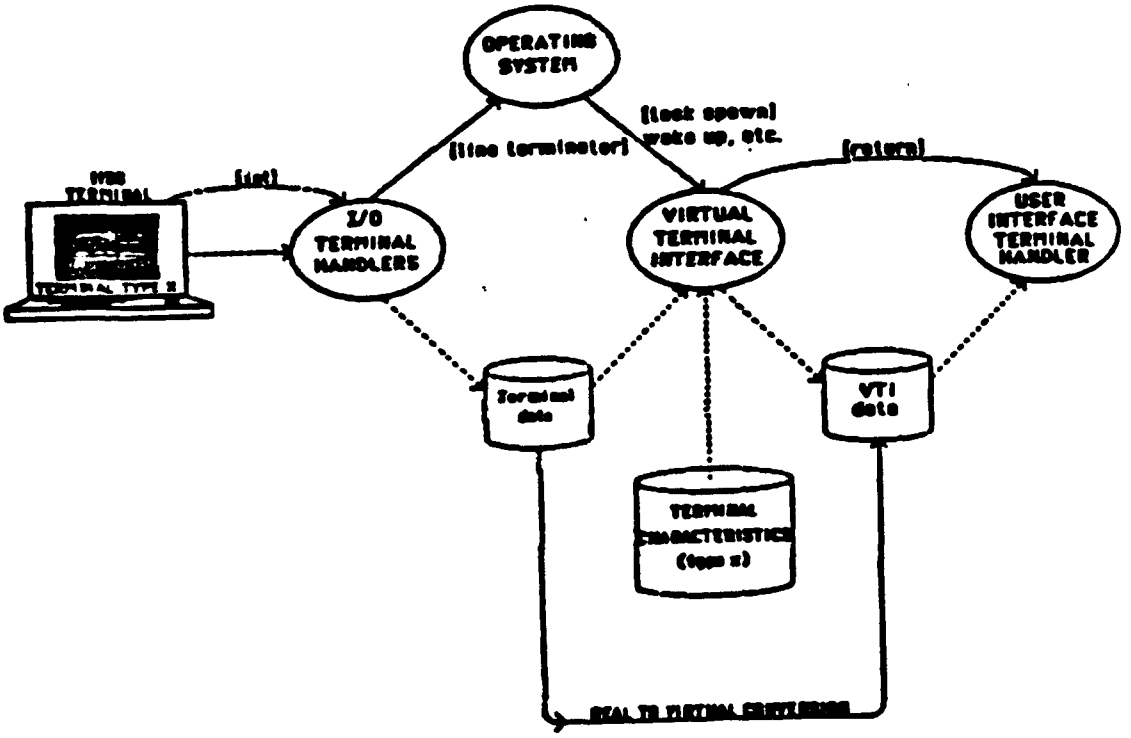


Figure 3-40. Data Acquisition, Conversion, Transmission

I/O handler is shown to be under the control of the host operating system.

6. Feature Mapping

Some hardware terminals may not offer any hardware implementation support for all of the standard terminal features assumed for the Virtual Terminal Interface. For such terminals, it may then be necessary to simulate some of the desired Virtual Terminal feature by a well thought out sequence of operations which are implementable on the hardware terminal. This process is referred to as feature mapping.

3.1.4.3.2 Interface to an Existing Application Program

Terminals are not the only entities in the Test Bed which offer or assume specific terminal features. Programs written around a specific terminal hardware exhibit characteristics of that terminal. Such programs are said to be terminal dependent, and in fact are the general case, rather than the exception. The Virtual Terminal is also used to grant to those existing programs the terminal independence required for integration in the Test Bed. In this role, the Virtual Terminal performs all of the functions and services described in the previous section (Section 3.1.4.3.1). Refer to Figure 3-41. This figure shows that an existing Application Process invokes the Virtual Terminal Interface via a procedure call. Data is also passed to the Virtual Terminal Interface. This data is stored in a table, or perhaps in a file. The Virtual Terminal Interface performs the necessary conversion, and generates the equivalent VTI data. Once the conversion process is completed, the VTI transfers the data to the Network Transaction Manager. Invoking the Virtual Terminal Interface from one existing Application Process requires that either one of the following methods be used:

1. Modify the existing Application Process and replace the I/O statements with calls to the Virtual Terminal Interface subroutines. This can be done manually or automatically with a suitable precompiler.

2. At link time, replace the host standard I/O library with a library of functions including the Virtual Terminal Interface subroutines. These subroutines have the same names than the host I/O subroutines. Figure 3-41 also describes the process by which data is transmitted from the Network Transaction Manager to the existing Application Process. This process parallels the process used to transfer data from the existing Application Process to the NTM.

3.1.4.3.3 Interface to a New Application Program

In this context, new Application Program means an Application Program specifically written for the Test Bed. New Application Programs are using, by definition, the terminal features offered by the Virtual Terminal Interface. Thus, new Application Programs do not require any of the conversion otherwise performed by the Virtual Terminal Interface.

3.1.4.4 Virtual Terminal Functional Specifications

The functional specifications implied in the scenario presented in Section 3.1.4.3 are identified and presented in this section.

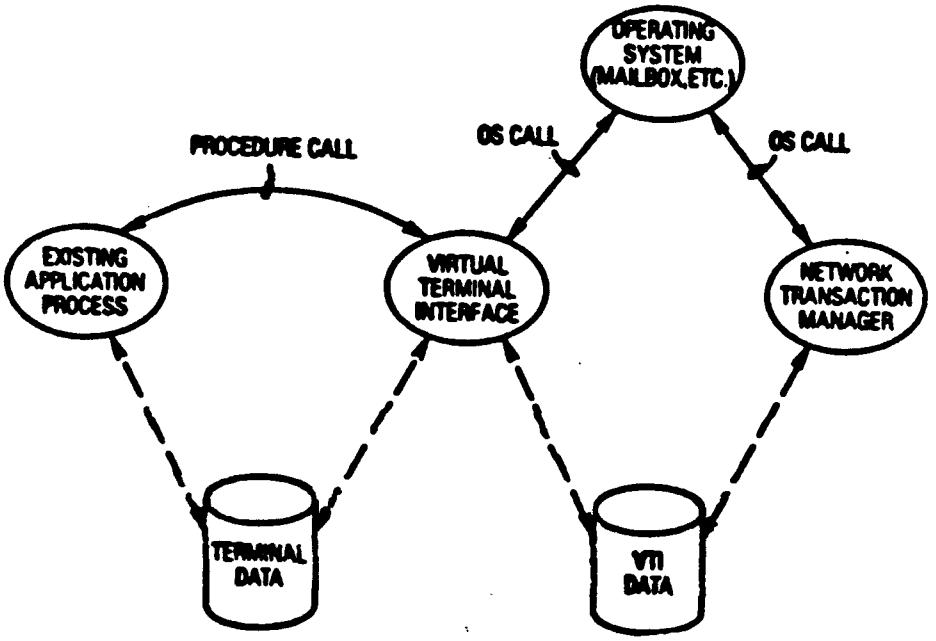


Figure 3-41. Interface to an Existing Application Program

3.1.4.4.1 Virtual Terminal Feature Definition

Mission: To provide the definition of a set of common terminal features and protocols used throughout the Test Bed.

Functional Specifications:

The Virtual Terminal Interface definition includes:

1. ASCII character set (lower case and upper case) as standard VTI characters
2. The following minimum set of VTI Control characters or sequences:

- o To indicate the end of line
- o To indicate the feeding of a new line
- o To position the cursor
- o To clear the screen
- o To indicate the end of a data block
- o To indicate a user interrupt
- o To indicate any of the VTI features listed below in paragraph 4

3. The following minimum set of VTI features:

- o Block mode display/input
- o Reverse video
- o Blinking
- o Bold (bright)
- o Dim (half-bright)
- o Underscore
- o Bell
- o No echo
- o Upper case
- o Lower case
- o Size of screen

4. The following minimum set of VTI protocols:

- o Block VTI/Real Terminal
- o Block VTI/VTI

3.1.4.4.2 Virtual Terminal Implementation

Mission: To implement the Real Terminal/Virtual Terminal protocol, character set and feature transformation required to achieve terminal independence.

Functional Specifications:

1. The implementation of the Virtual Terminal supports the following functions:

- o Character set conversion
- o Feature mapping
- o Protocol conversion

2. The following real terminals are used for demonstration of the VTI concept:
   - o DEC VT-100
   - o Honeywell VIP 7200
   - o Lear Siegler ADM-3
3. An instance of the Virtual Terminal Interface supports one instance of the real terminal type.
4. The Virtual Terminal Interface can be expanded to include new features, and to support via software simulation functionality limited terminal hardware.

3.1.5 The Communication Subsystem Configuration Item

3.1.5.1 Communication Subsystem Mission Statement

The Communication Subsystem provides Communication Services to the Test Bed Subsystems. The Communication Services allow on host interprocess communication and inter host communication between the various Test Bed Subsystems.

3.1.5.2 Communication Subsystem Functional Areas

The configuration tree shown on Figure 3-42 identifies the following functional areas:

Communication Hardware

- o Local Area Network
- o Wide Area Network

Communication Software

- o Inter Process Communication
- o Inter Host Communication
- o Configuration & Maintenance

3.1.5.3 Communication Subsystem Operational Scenarios

Figure 3-43 shows the demonstration hardware environment of the Test Bed. This figure shows the Local Area Network and the Wide Area Network services used to interconnect the IBM 3033 computer to the Local Area Network. The same figure also shows

the wide area services used to support remote software development on each of the Test Bed hosts, as well as on the CIDS development system (CIDS during initial phase only).

3.1.5.3.1 Local Area Network

Refer to Figure 3-43. The Local Area Network is composed of the BUS Interface Units of the GENET Local Area Network. These units are shown interconnected by a coaxial cable, with each bus interface unit properly tapped into the cable.

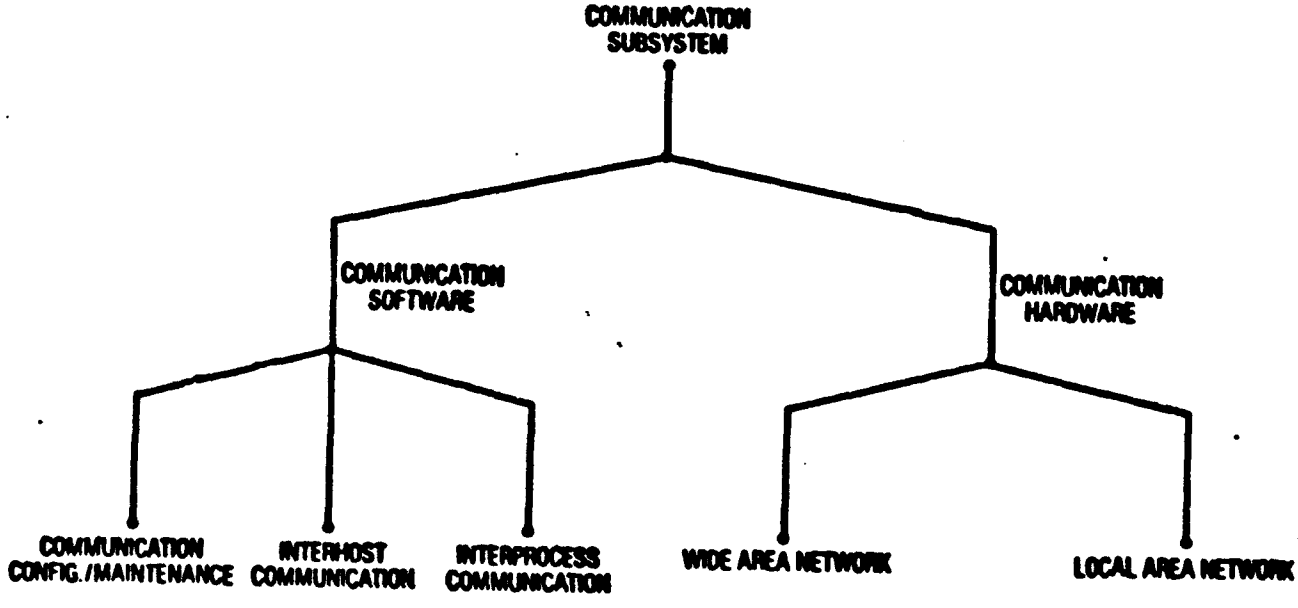


Figure 3-42. Communication Subsystem: Configuration Tree

The Honeywell Level 6 and the VAX are shown to communicate with their respective bus interface units via two RS-232-C communication lines.

The IBM 3081 is shown to be connected via synchronous modems and a leased telephone line to a cluster controller located in the room containing the Honeywell Level 6 and the VAX. The cluster controller communicates to a bus interface unit via 2 RS-232-C lines. A third RS-232-C is used to connect an asynchronous development terminal to the IBM 3081.

The cluster controller performs the multiplexing and demultiplexing of the traffic carried by the synchronous line into the two asynchronous lines shown. In addition, the cluster controller also performs the synchronous/asynchronous protocol conversion as well as the ASCII/EBCDIC character conversions.

The GENET Local Area Network is used in the Permanent Virtual Circuit Mode (PVC). The Local Area Network supports the 3 Virtual Circuits shown on Figure 3-44.

The Virtual Circuits shown on Figure 3-44 are permanent, that is these circuits are set up when the Local Area Network is powered up, and are maintained until the system is shut down. The configuration data required to set up these virtual circuits and to configure the six RS-232-C ports is stored in the GENET Configuration ROM Memory.

Figure 3-44 shows clearly that with the permanent Virtual Circuit approach outlined above, one RS-232-C port on each machine is dedicated to the bidirectional communication operations with a given computer. This port/host assignment is known to the communication software.

The Local Area Network described above clearly allows for the distributed processing required to support the operation of the Test Bed. Each host is capable of transferring information bidirectionally with every other host. The approach described here is used since no overhead is incurred to set up and tear down Virtual Circuits.

3.1.5.3.2 Wide Area Network

Consider Figure 3-43. The IBM 3081, which is located some three miles away from the other computers, is interfaced to the cluster controller via synchronous modems and a leased telephone

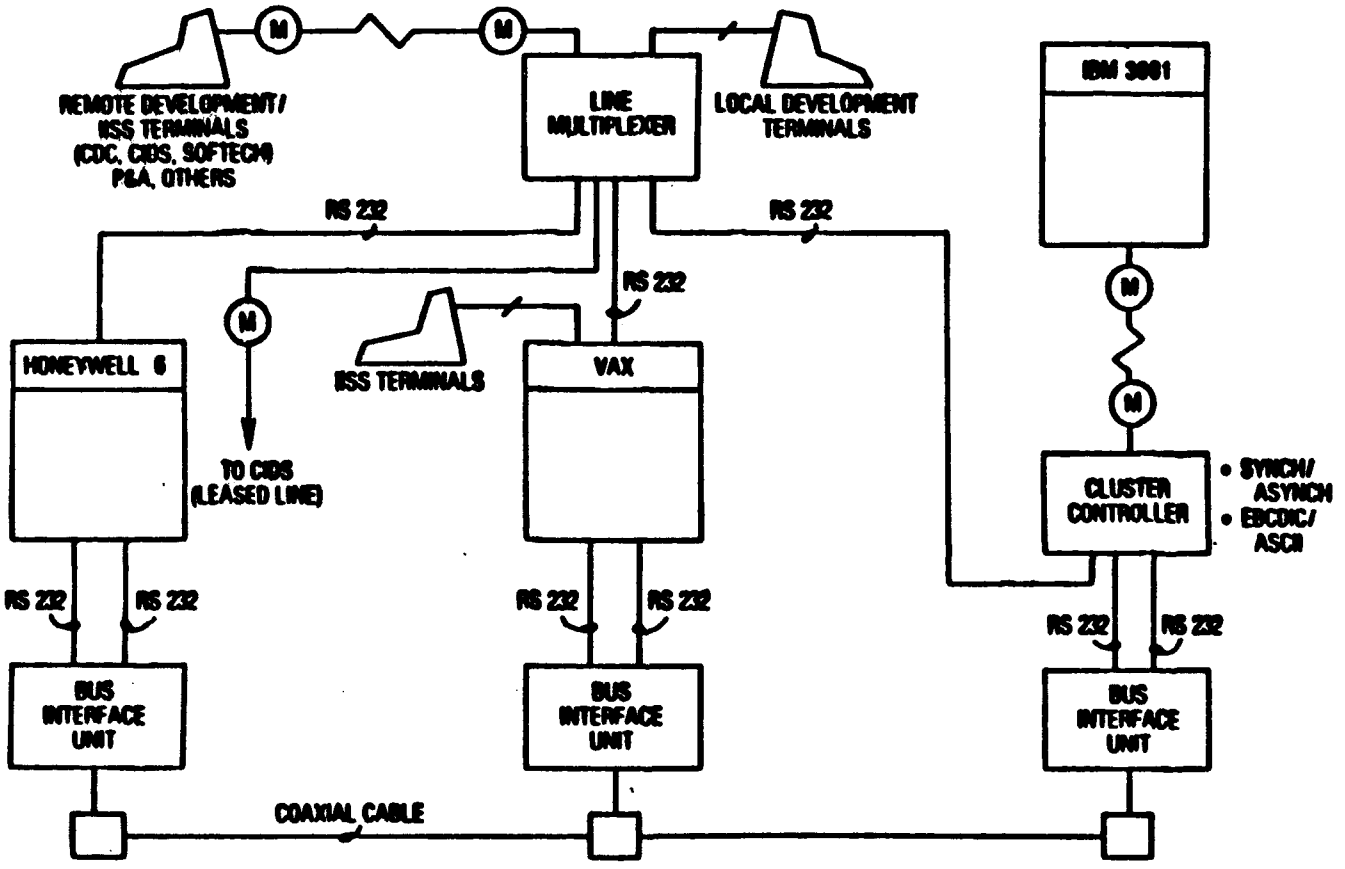


Figure 3-43. Test Bed Local & Wide Area Network Configuration

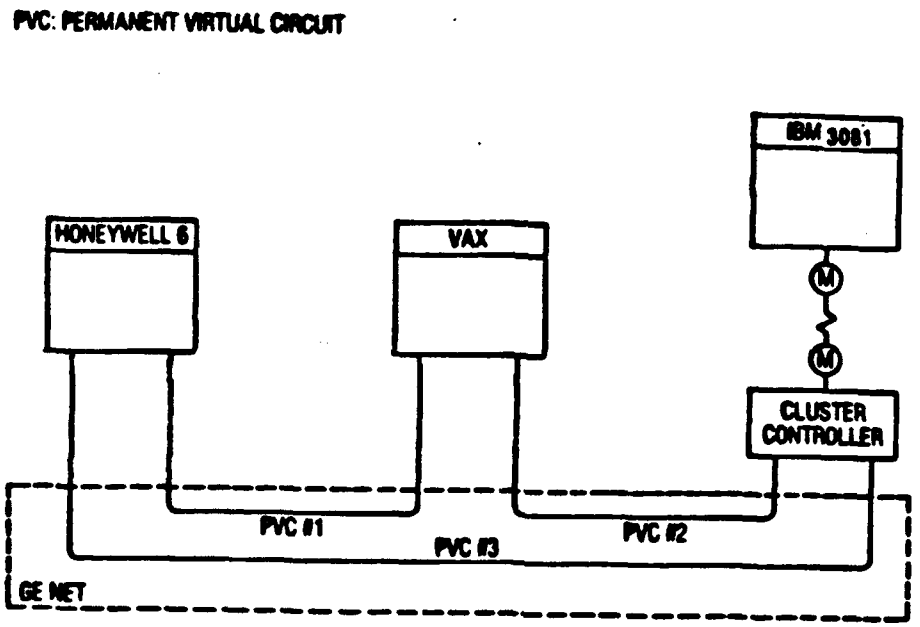


Figure 3-44. Genet Permanent Virtual Circuit

line.

A line multiplexer allows the multiplexing of local and remote terminals to any of the Test Bed hosts. This convenience is of major significance when considering that software development can be carried out remotely by the various ICAM subcontractors having need to access the Test Bed. This configuration allows SofTech, CDC, P&A and other ICAM subcontractors to gain access to any host of the Test Bed, in native mode. Likewise, this configuration allows any remote terminal to become an IISS terminal.

The line multiplexer is also used to multiplex terminals located at SofTech or at General Electric to the Central ICAM Development System (CIDS). This allows SofTech and General Electric developer to share a leased telephone line to CIDS, located in Cushing, Oklahoma (Initial phase).

The line multiplexer arrangement described above thus supports development activities and forseeable Test Bed experimentation scenarios.

3.1.5.3.3 Inter Process Communication Scenario

The Test Bed software is composed of many processes. These processes are either System Services or Application Processes, and are in general highly independent. Processes which are

coresident on one host communicate with one another via the Interprocess Communication (IPC) Primitives provided by the Test Bed system software. Processes which reside on different Test Bed hosts communicate with one another via the Interhost Communication (IHC) services provided by the Test Bed system software.

The following scenario describes the sequence of operations typical of a communication between two processes coresident on the same processor. This scenario describes the communication activities which take place between the Network Transaction Manager (NTM) and a typical process called Process X. This scenario is presented here for the following reasons:

a. In the Test Bed, all communications are routed via the Message Manager portion of the NTM. This rule is convenient since it allows the grouping of the manage message functions in one module, and since it also allows for a highly structured design. The problem of supporting communication between n times n applications is thus reduced to a much simpler problem, namely that of communicating between any process and the NTM.

b. Communications between the NTM and the COMM process used for interhost communication is a particular case of the above case. The simplification encountered in this very important case of interprocess communication are pointed out in the discussion of interhost communication.

The following scenario is implemented when the NTM on the AP Cluster where Process X resides receives a message requesting that Process X be initiated. The discussion presented here applies to the initiation of the first or any additional instance of Process X on the AP Cluster.

3.1.5.3.3.1 Establishing Mail Boxes between Process X and the NTM

a. On AP Cluster NTM operations:

1. The NTM receives a message requesting that an additional instance of Process X be initiated.

2. The NTM creates a unique name for the instance of Process X.

3. The NTM calls upon the local host operating system to create a new instance of Process X.

b. New Process X operations

The following actions are taken by the newly created instance of Process X. The new instance of Process X is created by the local host operating system in response to the initiate Process X request placed by the NTM in Step a.3 above.

1. The newly created instance of Process X obtains the name of the mail boxes it must use to communicate with the NTM on AP Cluster. The names of the mailboxes are predefined to the NTM Run Time Routines bound to the Process X.

2. The newly created instance of Process X creates the input mail box it uses to obtain data from the NTM. This input mail box is named according to the name given to the AP which is obtained from the local operating system.

3. The newly created instance of Process X signals to the NTM that it is running normally and is ready to accept data from the NTM by writing into the NTM input mail box.

3.1.5.3.3.2 Writing and Reading into the Mail Box

a. Writing into the NTM input mail box

Consider Figure 3-45. Once the input and output mail boxes have been identified and created per the above procedure, Process X writes into the input mail box of the NTM by invoking the communication service used to write into a mail box. The call is placed by Process X, and the call conveys to that communication service the following data:

- Name of the buffer (in Process X) which contains the data to be transmitted
- Name of the NTM input mail box to be written into

Messages written into a mail box are queued on a FIFO basis.

The logic of the host operating system or interprocess primitives prevents overwriting a mail box which has not been read by the NTM.

If the attempt to write in the input mail box was successful, Process X continues processing.

b. Reading the NTM input mail box

Consider Figure 3-45. The host operating system (or IPC primitive) detects the fact that Process X wrote into the input mail box of the NTM.

This allows the Communication Service for Reading Data from input mail box to proceed with the reading of the data contained in the mail box. This communication service also copies the input mail box data into the NTM buffer.

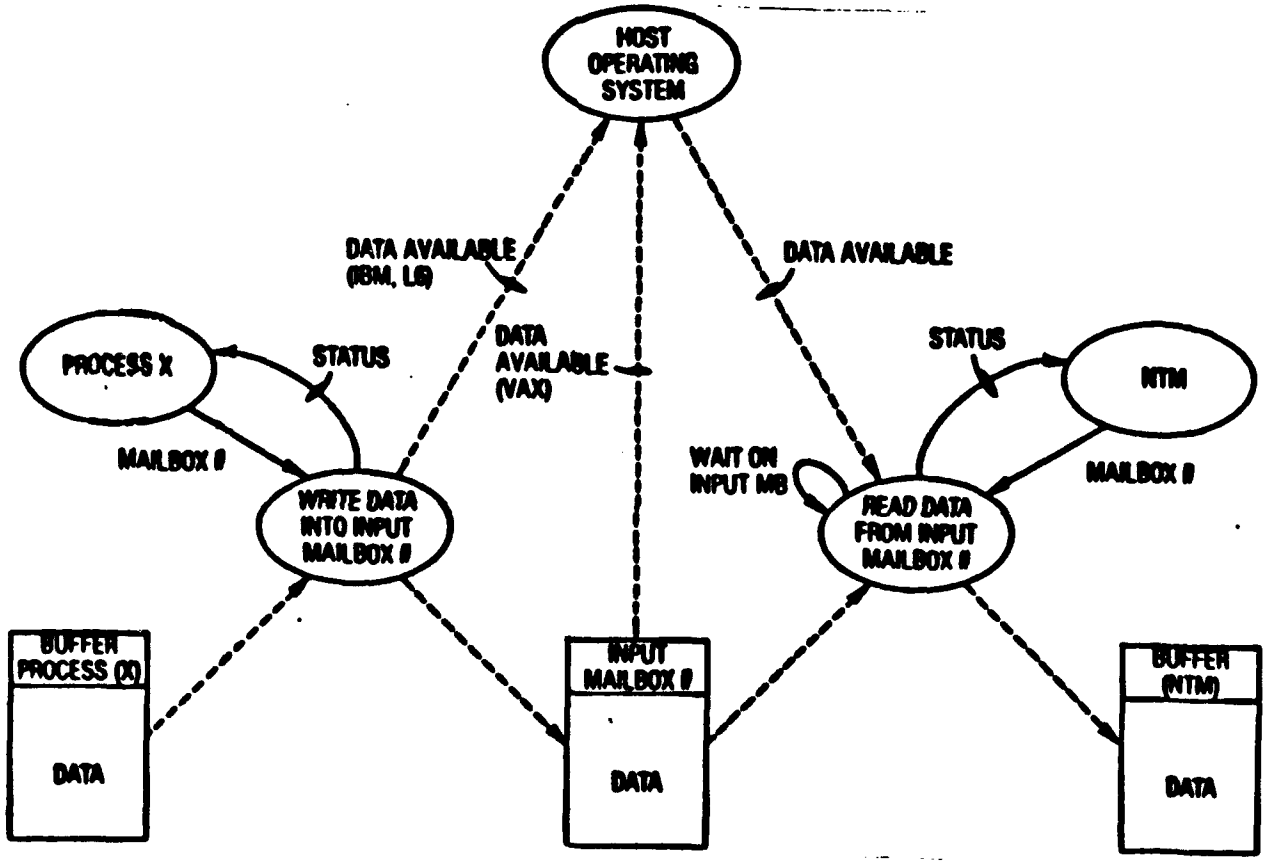


Figure 3-45. Writing & Reading into Mail Box

This Communication Service also returns status information to the NTM. In the event of a successful transmission, normal NTM processing continues. In the event of errors in the transmission, the NTM logs the error code in the error log and takes appropriate action.

3.1.5.3.3.3 Clearing Mail Boxes

The mail boxes created by Process X to communicate with the NTM must be cleared when Process X terminates. Clearing the mail boxes returns memory storage to the buffer pool.

Two eventualities must be considered when clearing the mail boxes used by Process X to communicate with the NTM. The first eventuality is the normal termination of Process X, and the second is the abnormal termination of Process X.

3.1.5.3.4 Process Synchronization

Process synchronization capabilities are required in any system made up of cooperating processes. In the Test Bed, process synchronization is achieved via the WAIT primitive.

A process which needs to be synchronized with another process utilize a combination of the Receive, Set-Timer, and Wait primitives. The Receive primitive accepts data from the mail box of the process with which synchronization is desired. The Set-Time primitive is used to detect a synchronization failure and the Wait primicive returns to the calling programs when either the timer elapses or the expected data is received.

The above scenario may lead to anyone of the following outcomes:

1. The program target of the synchronization request supplies a synchronization message and the WAIT primitive returns to its calling program thereby achieving synchronization.

2. The target program fails to supply a message and a count down timer terminates the WAIT primitive which returns an error message to its calling program.

3.1.5.3.5 Inter Host Communication

Figure 3-46 shows an overview of the Inter Host Communication Subsystem. This figure illustrates the following concepts:

1. The NTM on the COMM AP Cluster routes all inter host traffic

2. A COMM Subsystem is dedicated to communication with a particular host, and as such transmits and receives on an RS-232 port dedicated to a specific host.

3. Each COMM Subsystem uses two queues to communicate with the NTM on the COMM AP Cluster. One queue is dedicated to inbound messages, whereas the other queue is dedicated to outbound messages.

4. The COMM Subsystems invoke a count down timer to detect line failures.

5. The COMM Subsystems use the local host operating system I/O handler to control their respective ports.

6. The inbound and outbound queues can contain more than one message at a time. From COMM point-of-view, the queues are handled on a FIFO basis. Any scheduling of the messages (both in and outbound) is performed by the NTM on the COMM AP Cluster. COMM is responsible for segmenting messages which exceed the size of the communications data blocks. The COMM is also responsible for assembling the various data message segments into a message prior to routing to the NTM.

3.1.5.3.5.1 Message Scheduling (Future)

(Initial message scheduling is on a FIFO basis.)
The NTM is responsible for scheduling the messages to be transmitted by COMM. The scheduling is based on the following rules:

1. The messages from all Application Processes on an AP Cluster are examined for the highest priority on a round robin basis.

2. Consecutive messages to be transmitted to COMM do not belong to the same Application Process queue unless all

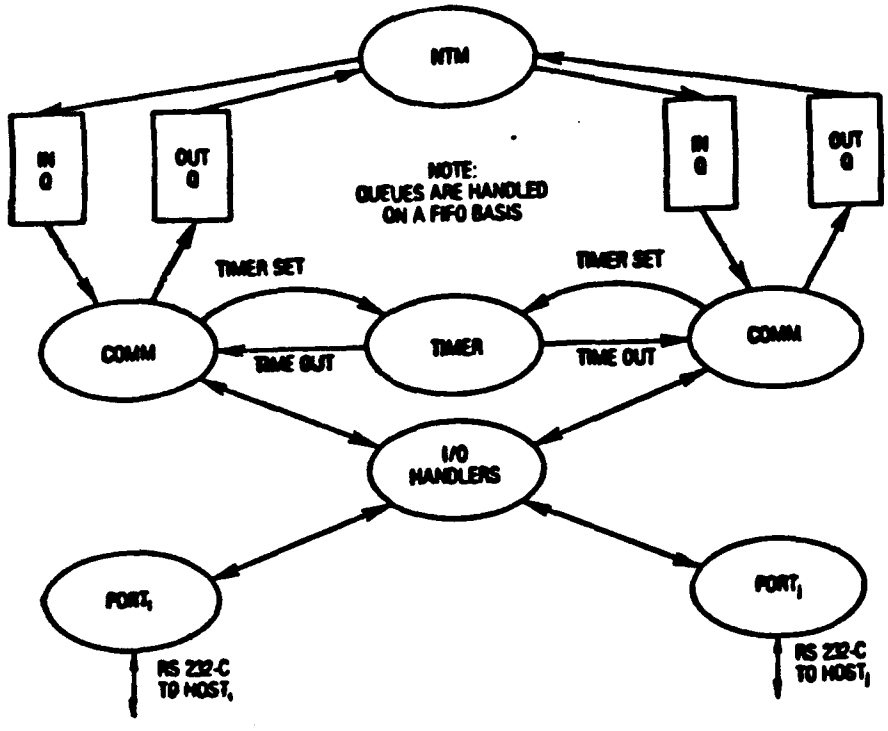


Figure 3-46. Interhost Communication & Overview

other queues are empty.

3. Messages which are not selected (passed over) for transmission because of insufficient priority are aged to ensure that they are not indefinitely delayed in a busy, higher priority environment.

4. Back pressure is applied, if necessary, through the round robin scheduling by skipping those queues which cannot be transmitted because of insufficient space on the corresponding queues. This implies status feedback from the receiving AP Cluster.

Figure 3-47 supports an illustration of the message priority scheme implementing the above rules. A discussion of the message scheduling scenario follows:

Messages issued by Application Processes AP1, AP2, and AP3, are transmitted via mail boxes to the NTM. These messages are queued by the NTM. The queue is processed according to the above rules, and messages selected are either send to COMM or to another NTM on the same host.

The messages are transmitted to the COMM AP Cluster NTM via shared memory. The messages are segmented into message segments when required and queued for transmission by COMM.

Upon reception by COMM (on the other host), the message segments are reassembled and then transmitted to the COMM AP Cluster NTM. The messages are then placed in the mail boxes of the Application Processes. When a mail box is filled to the point that additional messages cannot be delivered, back pressure is applied to the sending NTM via message indicating the mail boxes to be skipped by the round robbin scheduler. This message is sent by the receiving AP Cluster to the transmitting AP Clusters.

3.1.5.3.5.2 Transmission Error Detection

To improve the reliability of the messages received by the receiving COMM program and forward to the NTM, the COMM program performs error detection and provides acknowledgement (positive and negative) to the sending COMM program.

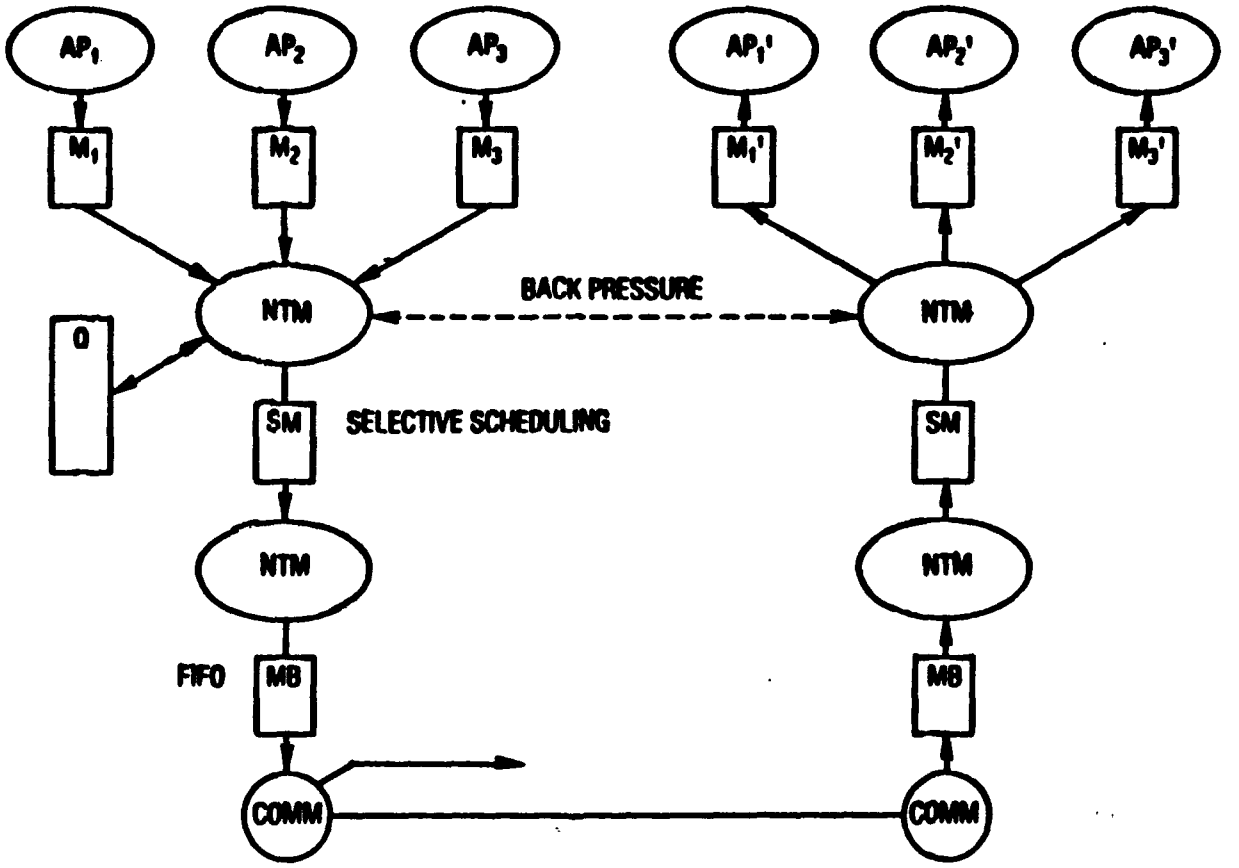


Figure 3-47. Message Scheduling (Interhost)

3.1.5.3.5.3 Line Protocol Handling

The line protocol implemented in the Test Bed establishes a master slave relationship between the sender and the receiver. The master is keeping track of the time out events and is responsible for terminating the transmission.

It must be noted that the Honeywell Level 6 and IBM 3081 computers do not support full duplex communication through their line I/O handlers.

Figure 3-48 shows the line protocol contemplated for the Test Bed, and supports the following discussion.

1. Idle state

   The COMM program enters the Idle state when:

   a. First started

   b. When no message segment was received, no message segment remains to be transmitted and no message is queued for transmission

2. Make line bid

   The COMM program indicates its willingness to act as a master by sending a message to the other host. If the other host is able to act as a slave, it will acknowledge the request by sending a line granted message.

3. Compute & wait backoff time

   If the other host was in the process of bidding for the line, a backoff time is computed to delay by a fixed amount of time the retransmission of the next line bid message. In the Test Bed, only two COMM programs may collide. These programs are configured with two markedly different backoff time constants. This simple scheme ensures that the second line bid attempt will not collide. It also implies a primary or favored role for one of the two COMM programs on one transmission line.

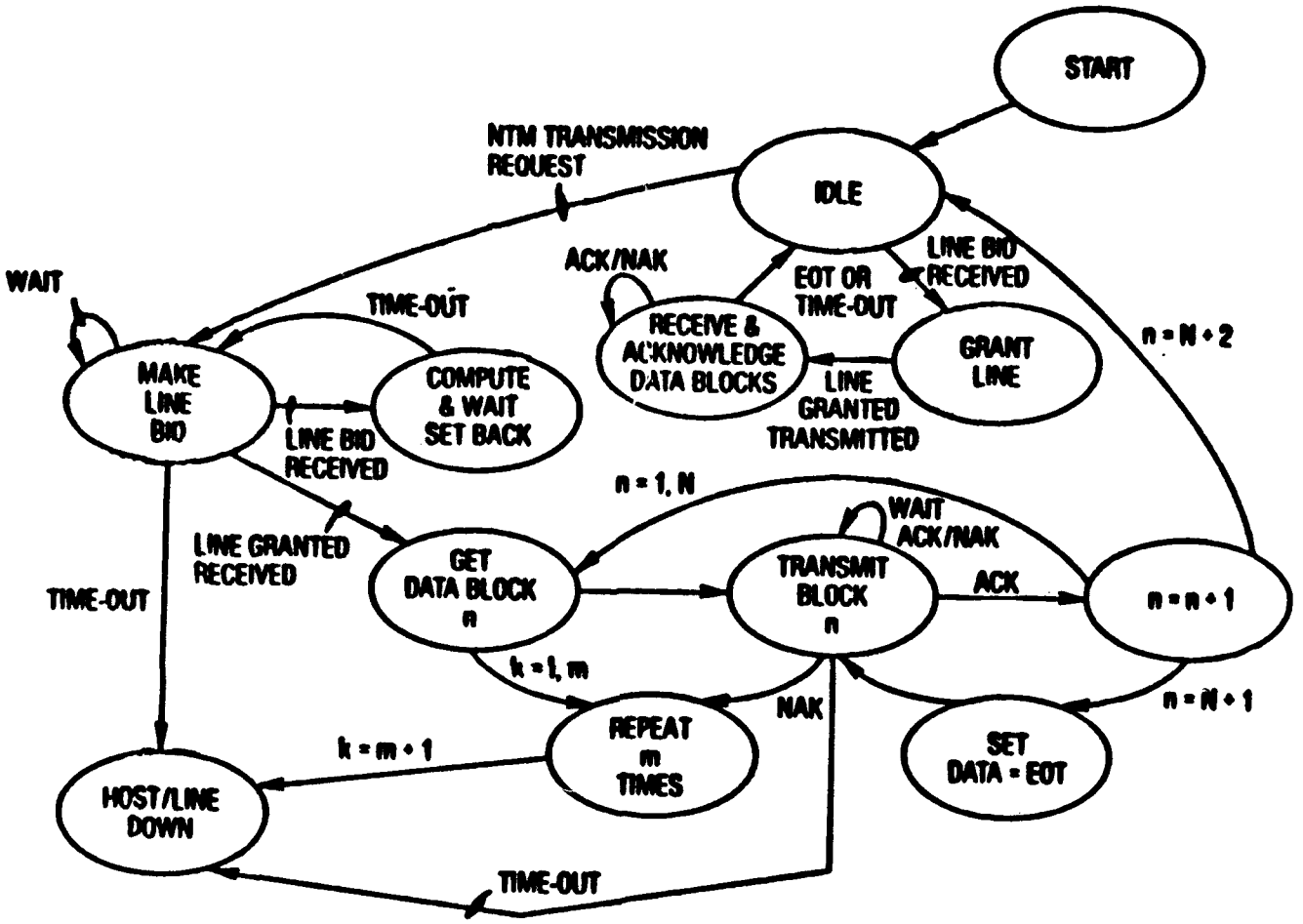


Figure 3-48. Line Protocol Transmission

4. Time Out

   A fixed time out timer is initiated by the Master after every transmission. Failure for the slave to respond within the time interval causes the master to retransmit. After a fixed number of retries, the line is assumed to be down and the fault is reported to the NTM. This time out technique applies to the line bidding as well.

5. Get first message segment n

   The COMM program enters this state when a line-granted message is received from the other COMM program. The

COMM program under discussion has thus gained control of the line and is ready for transmission. The message to be transmitted is assumed to be N message segment long, with the first segment to be transmitted being number one.

6. Transmit message segment n

   The COMM program proceeds with the transmission of message segment n. The COMM program appends a transmission header and trailer to the data block. This header contains the following information:

   - Cyclic transmit sequence number (1, 2, 3) which allows the detection of duplicate and missed blocks.
   - Cyclic receive sequence number
   - Control byte containing:
     - Line bid indicator
     - EOT marker
     - Continued message flag
     - ACK or NAK mark and cyclic transmission indicator of faulty segment
     - Binary/native flag

     The trailer contains the message segment check sum.

     Once the transmission of the message segment has been completed, the COMM program waits for an acknowledgement from the receiver.

7. Get next message segment (n = 2,N)

   A positive acknowledge from the receiver indicates that no errors were detected and that the transmission may proceed with the next message segment.

8. Repeat m times

   A negative acknowledge from the receiver indicates that an error was detected. The receiver indicates in its negative acknowledge the number of the message segment to be retransmitted. The retransmission sequence is attempted m times before the COMM program assumes that a hard failure is present in the system.

9. Send EOT

   The master sends a message containing the EOT control flag when transmission is complete. This message does

not contain data, and is used by the receiver to detect the end of the transmission.

10. Grant line

When the COMM program is in the Idle state, it responds to a line bid from its partner by a line granted message to indicate that it can assume the role of the slave.

11. Receive & acknowledge message segments

When in the slave mode, COMM proceeds with receiving and checking the message segments sent to it. COMM provides a negative acknowledge when:

- o The NTM input mail box is full
- o The message segment is out of sequence
- o The message checksum is in error

The acknowledge messages may be accompanied by message segments when there are messages to be transmitted by the slave to the master. The slave to master transmission proceeds as described above. A slave time out is required to detect failures in the master or in the Local Area Network. In this event, the slave returns a fatal error message to the NTM.

Configuration & Maintenance

The Configuration of the Communication Subsystem includes:

- o Port/Host Assignment
- o Port Configuration
- o Line Back Off Time Constraint (primary, secondary)

The above assignments are data driven. Thus reconfiguration does not imply recompilation. Whenever feasible, the configuration data is kept in the CDM and is downloaded at startup time. This approach does not apply to the minimum communication capabilities required to boot the IISS System. The configuration data for the minimum boot system is however data driven, and is contained in tables supported by the local hosts.

3.1.5.4 Communication Subsystem Functional Specifications

The functional specifications implied in the scenarios presented in Section 3.1.5.3 are identified and presented in this Section.

Local Area Network

Mission: To interconnect the VAX, Honeywell and cluster controller via Permanent Virtual Circuits.

Functional Specifications:

- o A minimum of 3 permanent Virtual Circuits
- o A minimum of 6 RS-232-C configurable ports
- o Error detection
- o Hardware diagnostics
- o Configuration data is stored in ROM memory

Wide Area Network

Mission: (1) to interconnect the IBM 3081 with the cluster controller, and (2) to multiplex 4 lines to the VAX, Honeywell Level 6 and to the cluster controller.

Functional Specifications

1. Cluster Controller
   - o EBCDIC to ASCII conversion
   - o Synchronous to Asynchronous protocol conversion
   - o Single Line 4800 bauds modem to support leased telephone line
   - o Protocol compatibility with GE and Boeing IBM 30xx computers
   - o A minimum of 7 RS-232-C ports to be multiplexed/demultiplexed into the synchronous line
2. Line Multiplexer
   - o Multiplex switched network telephone lines into the following equipment:
     - VAX RS-232-C port
     - Honeywell RS-232-C port
     - Cluster Controller RS-232-C port
     - CIDS leased telephone line (Initial)
     - Provide MODEM capabilities for dialup lines (Bell 103)
     - Multiplex local terminals into the above listed equipment

Inter Process Communication

Mission: To provide Communication Services supporting on host, process to process communications operations. The primitives support setting up, operating, tearing down the communication resources under normal and abnormal termination modes and error processing.

Functional Specifications

- o Generate unique mail box identifiers for communication between a given instance of a process and the NTM.
- o Create the input and output mail boxes bearing identifiers developed above
- o Communicate identification of mail boxes to the NTM
- o Write into mail boxes
- o Read from mail box when mail box has been written into
- o Notify NTM of the nature of the errors detected by the IPC services
- o Obtain name and size of buffer where data is to be stored
- o Detect buffer overrun
- o Detect reading from an empty buffer
- o Clear mail boxes when terminating

Interprocess Communication

Mission: To provide Communication Services supporting inter-host, NTM to NTM communication. The primitives support setting up, operating, tearing down the communication resources under normal and abnormal termination modes and error processing.

Functional Specifications

- o Maintain inbound and outbound message queues
- o Provide FIFO message processing with same priority level
- o Detect failures in Local Area Network
- o Provide bidirectional communications between any two hosts
- o Detect transmission errors (incomplete messages, bit drop out)

- o Detect message duplication
- o Support master/slave line protocol described in 3.1.5.4.3
- o Provide message segement check sum
- o Append transmission header
- o Append message end flag
- o Provide ACK/NACK on block receive
- o Retransmit n times before declaring a hard failure. Number of retries is defined with configuration data

Configuration & Maintenance

Mission: (1) to configure the Communication Subsystem to allow booting the IISS software, (2) to download the configuration data not required for booting, and (3) to maintain the configuration data.

Functional Specifications

- o Create and maintain local tables containing minimum configuration data for booting
- o Download CDM supported configuration data upon request from the hosts
- o Create and maintain the CDM supported configuration data tables

3.2 Interfaces

This section describes the system-level interfaces between the principal IISS Test Bed software subsystems. An overview of the Test Bed is shown in Figure 3-49. The major software components identified in this figure are:

- o Integrated Application Programs (AP's)
- o Non-Integrated Application Programs (MCMM, MRP)
- o Common Data Model (CDM)
- o Distributed Database System (DDBS) Processes
- o Local Database Management Systems (DBMS)
- o Network Transaction Manager (NTM)
- o User Interface (UI)
- o Communication Subsystem (COMM)
- o Test Bed Monitor

The obvious interfaces are the lines drawn between components in Figure 3-49. However, there are several levels, or "layers", of interfaces and there are also "protocols" between components that are not shown in this figure. For example, three important components not shown in the figure are:

- o Interprocess Communications Subsystem (IPC)
- o Virtual Terminal Interface (VTI)
- o Host Operating Systems

The following subsections describe the system-level information interfaces, the services to be provided, and the protocols to be established between software subsystems in the IISS Test Bed. Additional detail on interfaces can be found in the individual Configuration Item Development Specifications (DS's) and Product Specifications (PS's) to be developed for each software subsystem.

3.2.1 Information Interfaces

This section describes more detail about the kinds of information exchanged between software subsystems. Since information could potentially be exchanged between any pair of software components, a matrix is used to show all of the possible connections. A matrix showing the 12 categories of software components identified above is presented in Figure 3-50. The software components are listed along the diagonal. Each row indicates that output and the corresponding column indicates the input it may receive. Hence, information flows clockwise. For example, an Integrated Application Process (row 1) may send query requests to Distributed Database Processes (column 4) and will receive files of requested data in return (row 4, column 1). Where no interface exists between a pair of software components, large X's are placed in the corresponding boxes.

Figure 3-50 shows how each of the software components fit into a system framework. Components with many interconnections such as the NTM are clearly shown to be critical elements in the system. Other components such as the User Interface, Local DBMS's, and Telecommunications are shown to be relatively independent subsystems.

3.2.2 Services Provided (Internal Interfaces)

This section presents the IISS system interface requirements by describing the services to be provided by each of the major software components. The services provided by a software component are typically a subset of the functions it must perform. These are the functions it will be called upon directly by other software components to perform. For example, the NTM will be called upon to send messages between applicatin processes. This is a service. The NTM must also validate message header information and route messages to their

destinations, but these are not considered services in this context.

3.2.2.1 Integrated Application Programs

Integrated application programs provide "external" services to IISS users and may cooperate with other Test Bed programs. However, application programs are not considered to serve in any subordinate role with respect to other Test Bed software. Hence, no "internal" services are associated with application programs.

The service interfaces supported by other Test Bed software components and used directly by application programs are summarized in Figure 3-51. This figure shows the structure of a typical application program as consisting of COBOL source code and several layers of service routines which will be provided

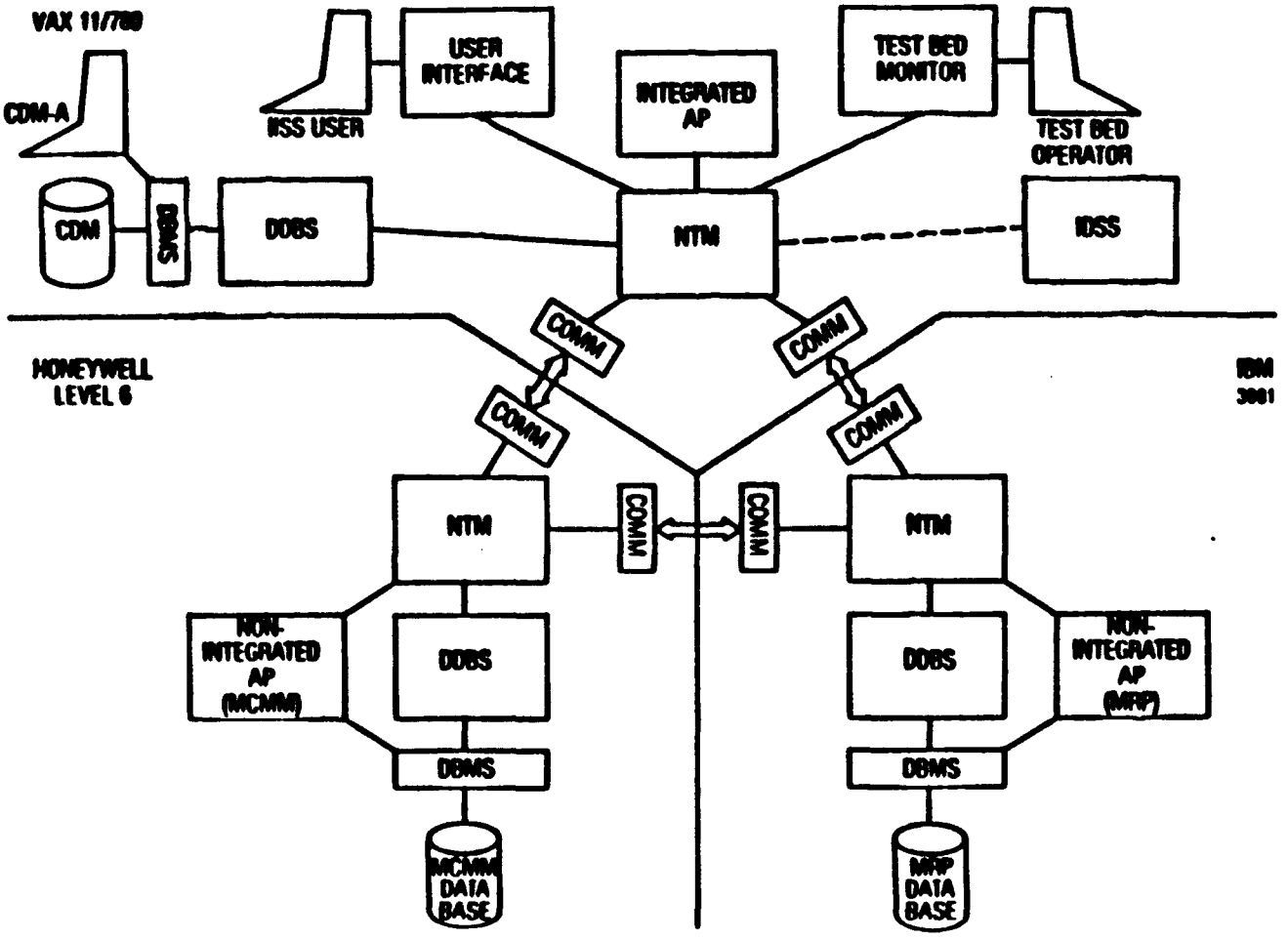


Figure 3-49. IISS Test Bed System Overview

from Test Bed software libraries. The first-level interfaces connect the application program with the User Interface, Distributed Database Processes (through precompiled Neutral Data Manipulation Language (NDML) statements), and the Network Transaction Manager. The lower-level interfaces show the connections with Interprocess Communications and the host operating system. Hence, Figure 3-51 reiterates the interfaces described by the top row and first column of the N-squared matrix shown in Figure 3-50.

3.2.2.2 Non-Integrated Application Processes

Non-Integrated application programs neither provide nor use Test Bed related services.

3.2.2.3 Common Data Model

The CDM has two principal roles in the Test Bed environment. One is maintaining an accurate picture of the data stored throughout the Test Bed computer network. The other is making this information available to Test Bed system and user processes. Maintenance of the CDM database is the responsibility of the CDM database administrator and is not considered a service. Providing data to Test Bed processes, however, is a service. The mechanism for calling upon CDM services is through NDML statements. Translation of NDML statements is a CDM function, but it is not considered a service. The requirements for the NDML syntax are shown in the CDM Processor Development Specifications. The syntax of NDML statements and the techniques for embedding queries in COBOL and Fortran programs form the CDM interface, and are fully described in the NDML Precompiler Development Specifications.

3.2.2.4 Distributed Database Processes

Figure 3-52 depicts the configuration of processes required to perform a query for distributed data. All of these processes (except the application process initiating the request) are "owned" by the CDM and only provide services for query processing. However, distributed queries are the most complex scenarios of communicating processes considered for the Test Bed environment, and they have "driven" the requirements for the NTM. Hence, the top layer of CDM internals has been exposed at the system level. The services provided by each of the components shown in Figure 3-52 are outlined below:

- Distributed Request Supervisor (Stager Scheduler)

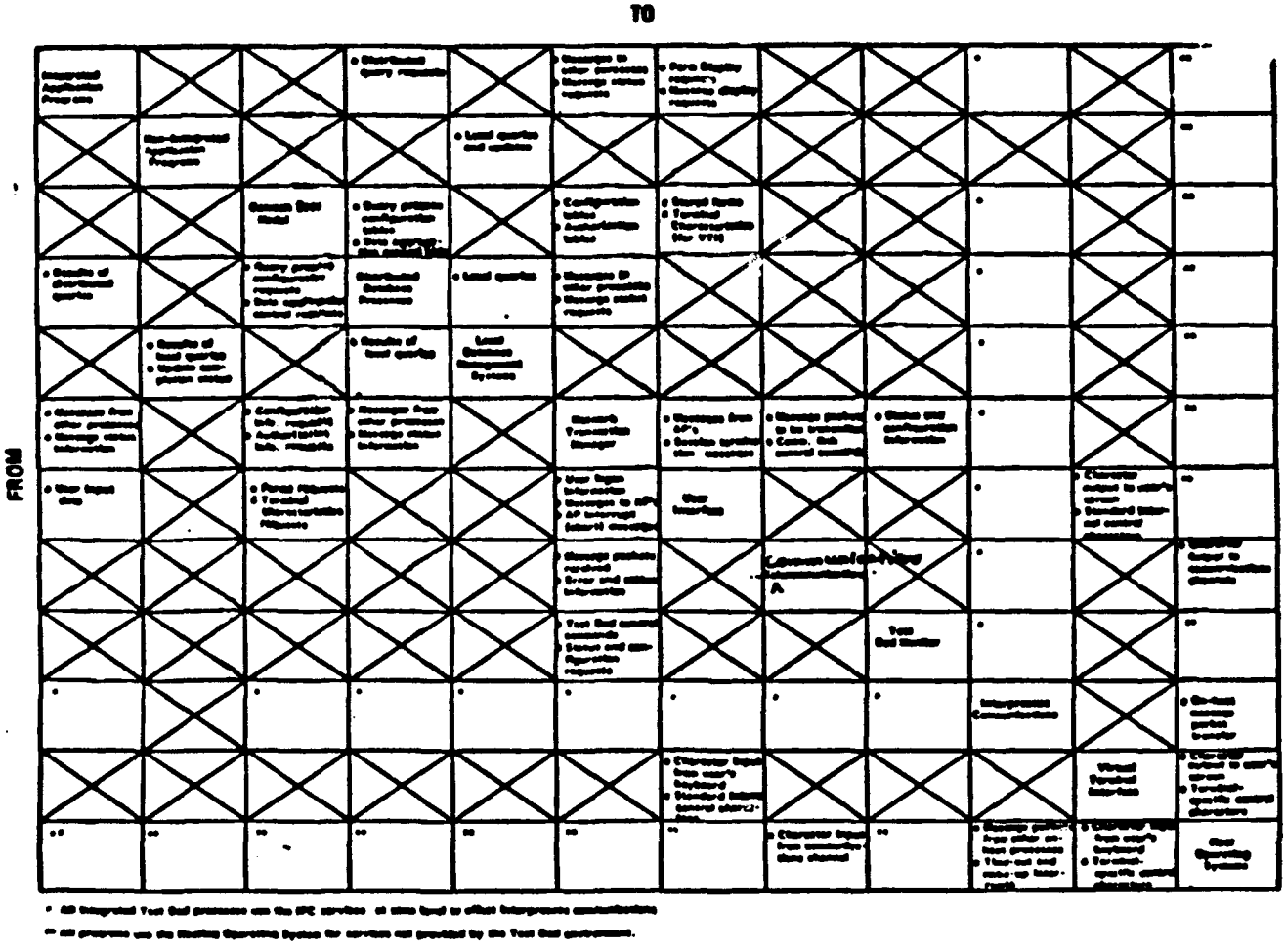


Figure 3-50. IISS Test Bed Software Information Interfaces

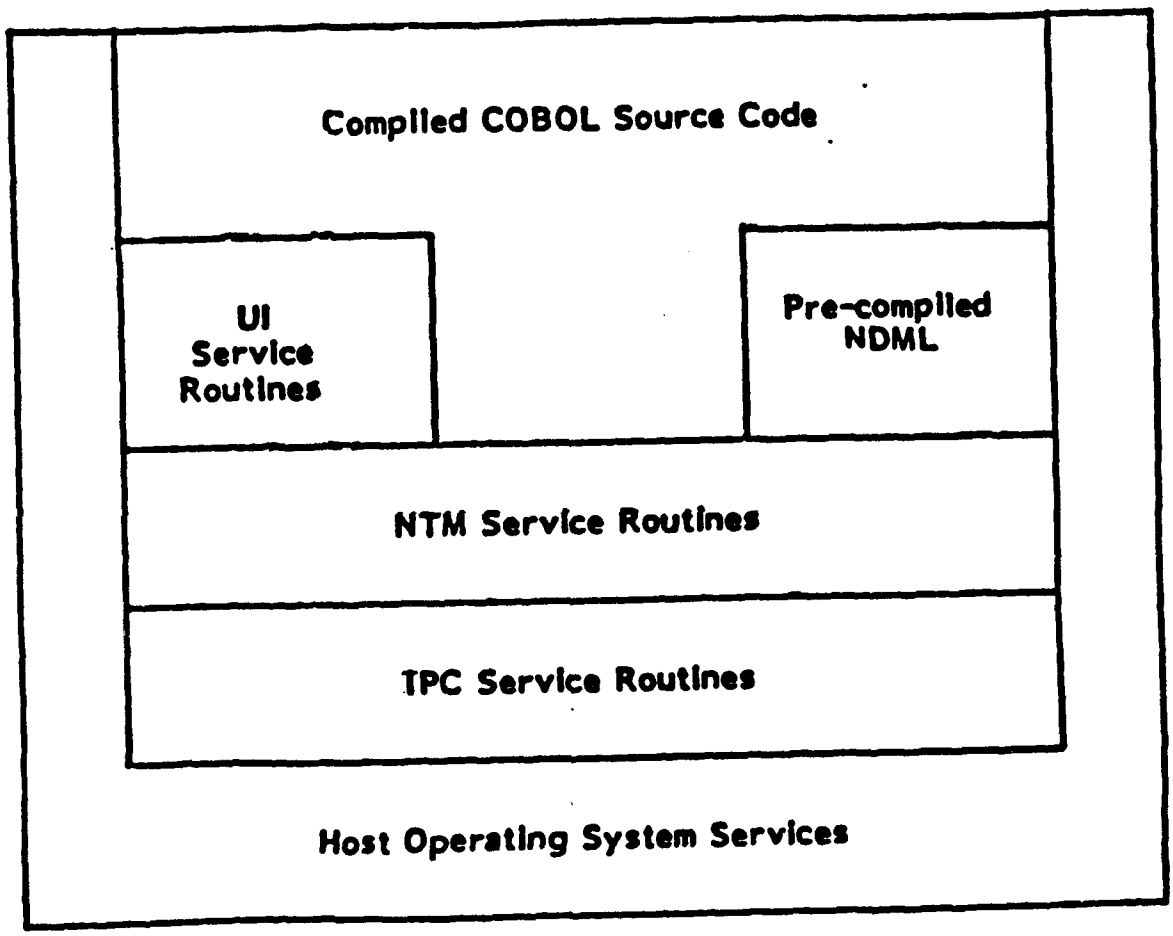


Figure 3-51. Structure of a Typical IISS Application Process

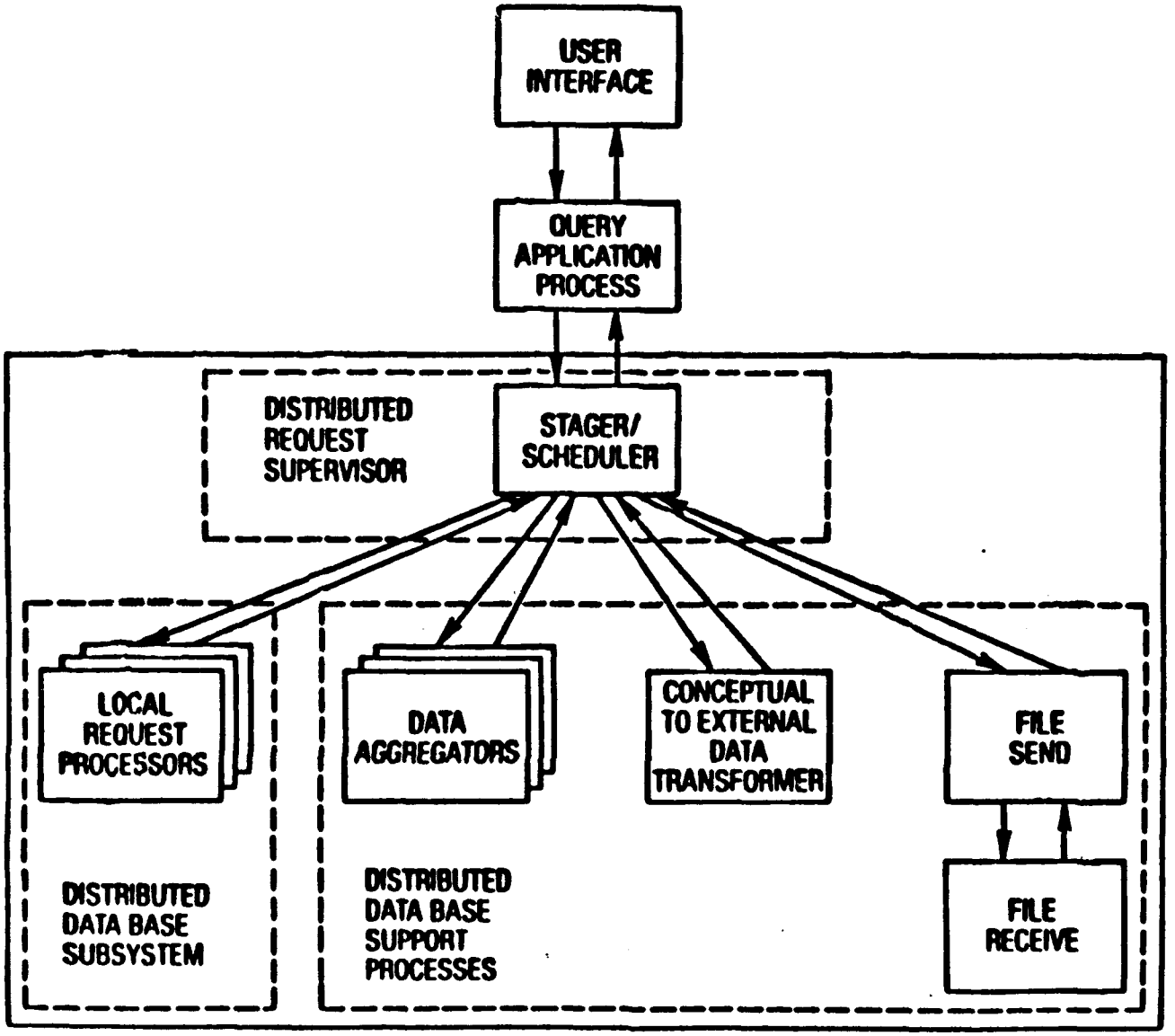


Figure 3-52. Predefined Query Processing

- o Local Request Processors
- o Data Aggregators
- o Conceptual to External Schema Transformer

3.2.2.5 Local Database Management Systems

Each local DBMS represents a unique interface for the local database request processes discussed above. These interfaces and the services a DBMS must provide for query execution shall be addressed in the NDML Precompiler and Local Request Process Generator design documentation.

3.2.2.6 Network Transaction Manager

Application Processes, Distributed Database Processes (DDP's), the User Interface, and other Test Bed programs will call upon the NTM for 4 major categories of services:

- o Process Logon and Logoff
- o Initiating and Terminating Processes
- o Sending and Receiving Messages
- o Obtaining Status of Messages and Processes

Each of these categories is expanded in following paragraphs.

The mechanism for invoking these services is through a library of service routines which is to be linked into each calling program (see Figure 3-51). This technique allows the NTM to view APs, database processes, and the UI as simply "processes." It also allows the NTM to effectively hide the details of NTM message headers, packet size, and the IPC interface from NTM users. The NTM must therefore provide the following services.

3.2.2.6.1 Process Logon and Logoff

1. Enable programs initiated under local operating systems to establish connections to the IISS Test Bed.

3.2.2.6.2 Process Initiation and Termination

1. Enable authorized processes to initiate other processes. Initiated processes may reside in the same application cluster, in different application clusters on the same host, or in application clusters on other hosts.
2. Enable delayed process initiation. The delay may be until a specified time in the future or for a specified elapsed time from the time of the request.
3. Enable initiation of multiple instances of a process -- without having to wait for one instance to complete before initiating the next.
4. Enable an authorized process to abort another process.
5. Enable a process to wait until another process or application cluster becomes available.

3.2.2.6.3 Send and Receive Messages

1. Enable a process to send a message over a logical channel to another process. Messages may require a

response (i.e., paired messages and guaranteed-delivery messages). Messages may also be either binary data which is to be received exactly as sent, or text which may require character code conversion when transmitted between hosts.

2. Enable a process to receive a message from a specified logical channel.

3. Enable a process to receive the next message arriving on any channel.

4. Enable a process to wait until a message arrives (Receive with a wait).

5. Enable a process to check for the presence of an incoming message (with or without actually obtaining the message) either from a specific channel or from any channel.

6. Enable a process to acknowledge the successful completion of processing in response to a guaranteed-delivery message.

7. Enable a process to indicate that it is under test and that its messages should not be allowed to corrupt normal system operation. (Future - Test mode indicator can be set/reset, but interpretation is AP specific)

3.2.2.6.4 Obtain Status of Messages and Processes

1. Enable an authorized process to obtain the logical names of the host and application cluster in which other processes reside, as well as the names of its own host and application cluster.

2. Enable an authorized process to obtain the status of hosts, application clusters, and processes.

3. Enable a process to obtain all necessary information (e.g., user's identification, the User Interface process name, and a suitable logical channel) to send informative messages to the user whose initial request caused its execution.

4. Enable a process to check the status of any guaranteed-delivery messages it has issued. (Future)

5. Enable a process to check the status of any messages for which replies are expected. (Future)

3.2.2.7 User Interface

The User Interface provides a number of services to IISS users such as simple menu-driven control of programs and a convenient "help" function. However, the interface described in this section focuses on the services provided by the UI to other

Test Bed software components -- principally Application Programs. The UI must provide the following services to control the display of forms and data, and retrieve user input:

1. Select forms to the displayed from a collection of previously defined, stored forms
2. Insert data into fields before they are displayed
3. Display part or all of a form on the user's screen
4. Allow field values to be updated on the screen
5. Erase part or all of a display
6. Accept data from the user's keyboard

3.2.2.8 Communications

The principal role of the Communications Subsystem is to provide host-to-host message (packet transfer). The Test Bed software architecture has allocated two COMM processes (one at either end) for each pair of communicating machines, as shown in Figure 3-49. Hence, messages sent via a given COMM process go to only a single destination. The only routing and distribution function performed by COMM processes is in handling high- and low-priority messages which are sent and received on different IPC channels but are transmitted between machines over a common communications channel (Message priority = Future). The services provided by COMM, therefore, are:

1. Transmission of messages received from its associated NTM process to its COMM counterpart on another host
2. Forwarding of messages received from its corresponding COMM process to its associated NTM process

3.2.2.9 Test Bed Monitor

- Performance monitoring
- Nonrecoverable failure handling

3.2.2.10 Interprocess Communications

The objectives of the Interprocess Communications Primitives (IPCs) are to provide a machine-independent, COBOL interface for communication between cooperating concurrent processes. In the IISS Test Bed environment, application programs will be provided higher-level message-passing services by the NTM. IPC services are expected to be used, directly, only by Test Bed "system" software such as the NTM and COMM. The services the IPC Primitives must provide for these subsystems are the following:

1. Establish communication channels between processes.

2. Send messages over established communication channels.

3. Receive messages over established communication channels.

4. Suspend a calling process for a specified period of time or until an incoming message arrives.

   [Note: "Communication channels" are implemented by a "Mailbox" concept.]

3.2.2.11 Virtual Terminal Interface

The function of the Virtual Terminal Interface (VTI) is to insulate application programs and other Test Bed software from the special characteristics of individual display terminals. Each brand of computer terminal uses a different set of control characters and different control sequences to clear the screen, position the cursor, highlight text, scroll displayed information, etc. The service provided by the VTI is the conversion of standard control characters and control sequences as needed to support the terminals connected to Test Bed computers. The standard internal character set and control sequences will be described in the VTI Development Specification.

3.2.2.12 Host Operating Systems

Host operating systems provide the run-time environment for all of the software components described above. Some of the specific run-time services they provide include:

1. System calls for interprocess communication

2. System calls for process initiation and termination

3. I/O primitives for input and output to user terminals, the local area network, and other peripherals.

3.2.3 Protocols and Messages

The services described above are implemented in a distributed processing environment like the IISS by exchanging "messages" between programs. When two programs of processes cooperate to deliver services, they must establish a set of rules and agreed-upon messages which, together, are called a "protocol". The message distribution services supported by the NTM, IPC, and COMM subsystems provide a basis for implementing these protocols.

The complexity of Test Bed process interconnections causes confusion about the distinction between interfaces and protocols. Figure 3-53 shows two views of process interaction. On the left is shown a "macro-level" view of a protocol between two application processes. This protocol is implemented by interfaces with the NTM. A similar protocol exists between the NTM processes and, at the bottom level, a telecommunications

protocol implements the actual transfer of the data. On the right-hand side of Figure 3-53 is shown a "microscopic" view of the interface between the application and NTM processes at the left. This view depicts a protocol between that application process and the NTM that is implemented by interfaces with the IPC. The IISS Test Bed macro-level protocols are described in the following paragraphs.

3.2.3.1 AP to AP

Protocols and messages exchanged between cooperating Application Programs (e.g., in a distributed AP) must be defined by the application designers. This information could ultimately reside in the CDM to allow the NTM to validate the exchange of messages at run-time.

3.2.3.2 AP to CDM

The protocol between an Application Program and the CDM's distributed database processes is embedded and effectively hidden within the NDML. This allows an Application Program to completely avoid direct dealings with the messages exchanged between it and the CDM processes. The content and format of these messages are defined by the CDM Precompiler.

3.2.3.3 AP to NTM

The protocol between Application Programs and the NTM is embedded within the library of NTM service routines which must be linked into every Test Bed program. This minimizes the dependencies on NTM message characteristics such as header information and message (packet) length within programs. The content and format of the NTM "message envelope" are defined by the NTM.

3.2.3.4 AP to UI

The protocol between Application Programs and the User

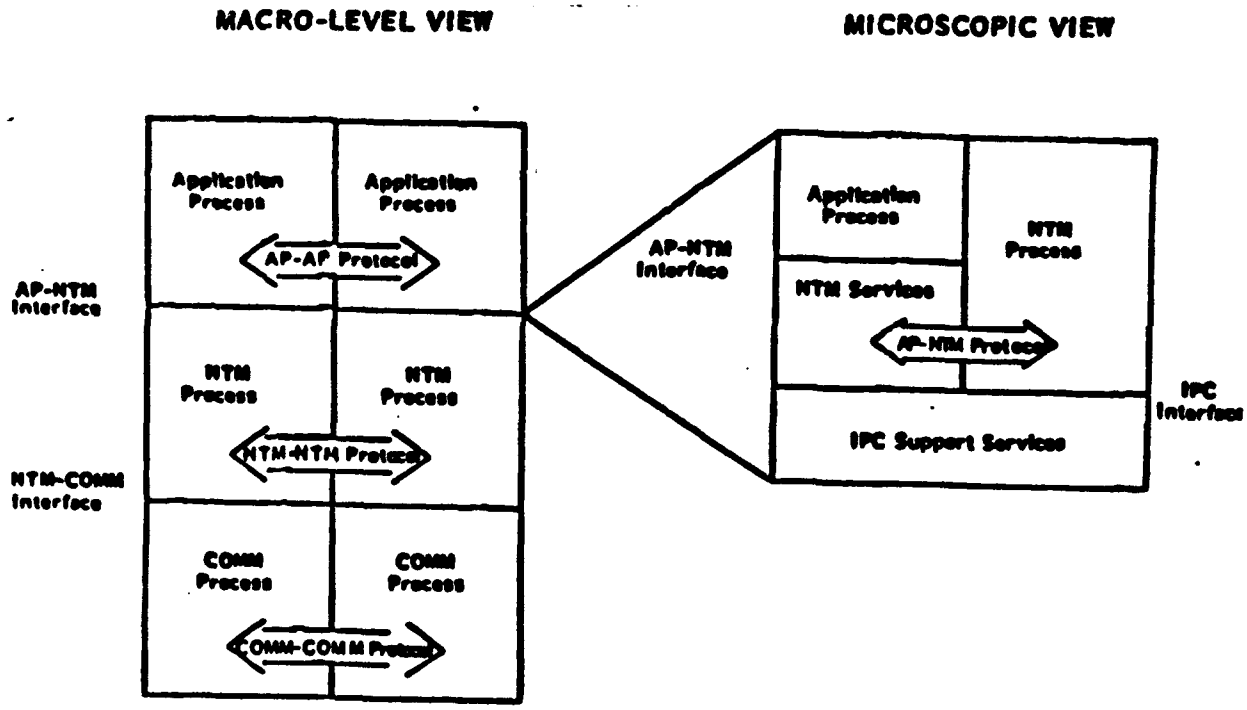


Figure 3-53. Example Protocols Between IISS Test Bed Processes

Interface are embedded within the library of UI service routines to minimize dependencies on the types of messages exchanged with the UI. The form and content of these messages are defined by the UI.

3.2.3.5 CDM to CDM

Messages exchanged between distributed database processes, including local database request processes, are defined by the CDM. Further detail on these messages will be presented in the CDM Development Specification.

3.2.3.6 NTM to NTM

Messages exchanged between NTM processes, either to forward messages between Application Process Clusters or for NTM process management, are defined by the NTM. Further detail on these messages will be presented in the NTM Development Specification.

SECTION 4

QUALITY ASSURANCE PROVISIONS

Requirements for formal tests/verifications of the IISS system design characteristics and operability will be specified based on the hardware used, combinations of developed softwre, as well as the developed implementation schedule. While a detailed Quality Assurance Plan cannot be developed at this time, critical minimum requirements have been identified.

4.1 General

A simplified breakdown of a system build and validation against functionality requirements would include the following major activities:

- o Develop and Verify Program Modules
- o Develop and Verify Programs
- o Develop and Verify Subsystem Modules
- o Develop and Verify Subsystems
- o Develop and Verify System
- o Volume Test System

Testing/validating will be performed as required as part of each of these activities. The testing procedures will be designed to validate that the functionality developed in the design has been fulfilled.

4.1.1 Responsibility for Test

Test/validation responsibility will be divided based on the contractor or subcontractor involved in the program module, program, and subsystem module.

Responsibility for testing/validating subsystems and systems will fall to the involved contractor and the user.

4.2 Special Tests and Examinations

Whenever software has been purchased, the vendor shall be responsible for insuring that the package will match the functional specifications previously developed and provided to the vendor. (It is expected that vendors will be responsible for insuring the packages perform whatever functions are necessary to support the functionality requested.)

SECTION 5

PREPARATION FOR DELIVERY

5.1 Hardware

The Test Bed hardware owned by the Air Force is scheduled to be delivered to the Air Foprce when it is no longer required to support the TestBed programmatic activities. The hardware shall be delivered in accordance with Air Force instructions, schedule, and at Air Force expense.

5.2 Software

The software developed under Project 6201M shall be delivered to the ICAM program office in accordance with the Test Bed Configuration Management Plan. The Software Configuration Management Plan for is published in the Final Technical Report Volume I (FTR620100001) and in the documents for the Software Configuration Management (SCM) system listed in the Appendix of the final report.

5.3 Documentation

The following Users Manuals have been prepared under Project 6201M and will be delivered to the ICAM program office in accordance with the Test Bed Configuration Management Plan. See the Final Technical Report Volume 1 (FTR620100001) for a complete list of all documents delivered under this contract.

5.3.1 NTM Programmer's Manual (Services)

This manual describes the services provided to IISS programmers by the Network Transaction Manager. These services are used by IISS Application Programs to send messages to and receive messages from other programs in IISS. This document is useful to programmers who are building IISS component programs and need to embed currently available NTM service calls in their programs. The document includes notes on restrictions, helpful hints, and reports on experience gained on the NTM.

5.3.2 NTM Operator's Manual

The NTM Operator's Manual describes the procedures and message exchanges taking place during the various phases of the NTM operational life cycle. Operator commands, IISS error codes, NTM table maintenance procedures, and NTM troubleshooting procedures are also given.

5.3.3. Common Data Model (CDM) Administrator's Manual

The primary focus of the CDM Administrator's Manual is placed upon the Scheme Intergration Methodology which is intended to guide a CDM administrator in building and maintaining the CDM database. Four models are provided as

components of the methodology. Two of the models are IDEF0 models. These models address building the initial conceptual schema and its incremental expansion. The CDM Administrator's Manual reflects the experience gained in integrating the MRP and MC-MM subsystems.

5.3.4 Precompiler User's Guide

The Precompiler User's Guide describes the procedures to be followed to precompile the NDML statements embedded in a COBOL source program. This manual describes the functions to be invoked, the naming conventions of the intermediate files, and the commands to be executed to review the source listing and the listing of the code generated by the Precompiler.

5.3.5 NDML User's Manual

This manual describes the requirements, theoretical foundation, commands and syntax of the IISS Neutral Data Manupulation Language. The Commands section of the manual consists of syntax and examples of the stand-alone NDML statements. Each NDML clasue is described. The Embedded NDML section describes the use of loopin, constructs containing NDML statements, and states restrictions on the use of looping, constructs contaiing NDML statements within a COBOL program. The BNF (Backus-Naur Form) includes a formal BNF description of the stand-alone form of the NDML.

5.3.6 UIMS User's Manual

The User Interface Management System Services (UIMS) Manual describes the forms and services availabe to the IISS user. In particular, the manual describes how to choose a function, to change password, to define an application, to define a command, to update a command, to define command parameters, and to execute an application. The manual contains examples and error messages.

5.3.7 VTI Programmer's Manual

The Virtual Terminal Interface (BTI) Programmer's Manual describes how to add a new terminal type to the Virtual Terminal Interfaces. The manual describes, in detail, the four files making up the VTI terminal type definition. These files are: the Definition File, the Lexical Analyzer Table, the Parser Table, and a Command Generation Table. The naming convention for each file is also given.

5.3.8 Forms Processor Application Programmer Manual

This document describes the callable interface to the Form Processor and is intended for the IISS application Programmers. The Form Processor routines use predefined forms to give programs the ability to read input and write data to terminals. The manual describes in detail each of the services provided and callable routines.

5.3.9 Interim Forms Editor Manual

The Interim Forms Editor is besed on a Digital Equipment Corporation IMS Forms Editor. The Interim Forms Editor Manual describes the data required to define a form and the restrictions bearing on the form definition process.

5.3.10 NDDL User's Guide

The NDDL Users Guide describes the syntax and semantics of each NDDL command. It describes how to use the language. It does not describe the role of maintaining the CDM which is found in the CDM Administrators Manual.

☆U.S. Government Printing Office:1992—648-127/62284