* * * DRAFT * * *

DDN FTP/HP3000

Internal Design

Information Networks Division

Location Code: 66-7860

Project Number: 6651-1131

February 10, 1986

David St. John

PRODUCT IDENTIFICATION

SECTION 1

Name	HP/3000 DDN Services: FTP
Mnemonic	none
Project Number	6651-1131
Project Manager	Doug Heath
Project Engineer	David St. John
Product Manager	Dennis King
Product Assurance	Karen Dillon
Documentation	Mike Genevro
Off-Line Support	Bruna Byrne
On-Line Support	Susan Gennrich Lorraine Mehrtens Jim Zepp

DESIGN OVERVIEW

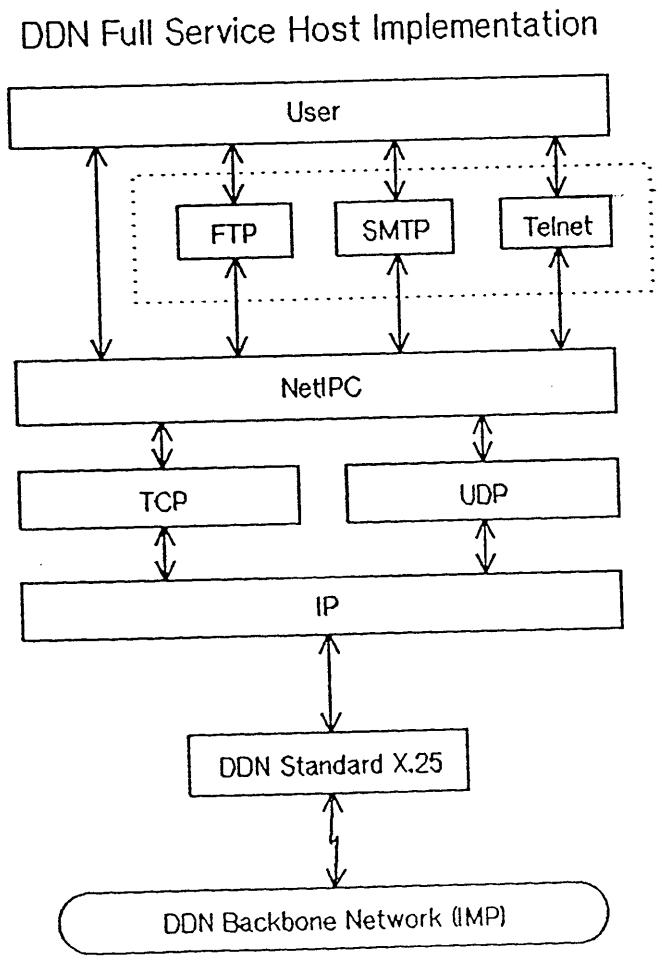
SECTION 2

2.1 DESIGN APPROACH

File Transfer Protocol (FTP) is one of three main required services protocols currently defined by the Department of Defence for the Defence Data Network (DDN). The other services are Simple Mail Transfer Protocol (SMTP) for electronic mail and Telnet, a network virtual terminal protocol.

2.1.1 DDN Model

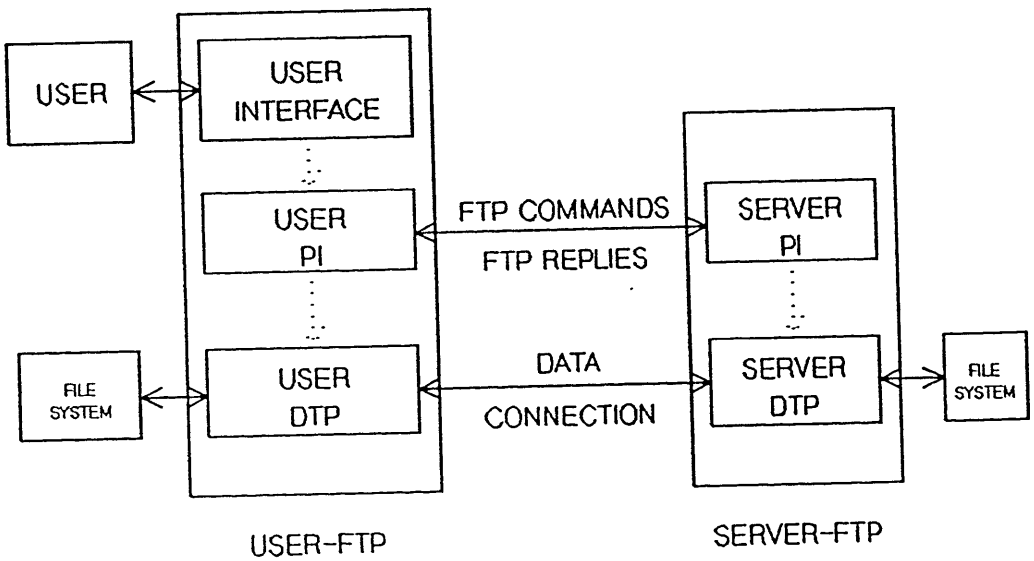
The DDN architecture model also has defined lower layer protocols, including Transmission Control Protocol (TCP), Internet Protocol (IP), and X.25. The figure below shows the standard defined model:



Hewlett-Packard's FTP service will provide file access from a 3000 to any of its other machines or to those of other vendors that follow the standard, Military Standard: File Transfer Protocol (Mil-Std). This design is intended for the HP3000 computer, but it will be designed with the objective to be portable to other systems with a minimum of effort. The design will follow the standard in all matters and the final product must prove that it has by qualifying for inclusion on the DDN (see Defense Data Network Host Interface Qualification Testing, Higher-Level Protocols). It is recommended that the Mil-Std as well as the External Specifications be read before this document.

2.1.2 FTP Model

There is also a model for the FTP service. This diagram is taken directly from the Mil-Std.



FTP MODEL

The USER-FTP and SERVER-FTP above are actually two processes and those items within these two large boxes will be within each of those processes. The user-FTP process receives the user commands via its interface. This interface has been defined in DDN FTP/HP3000: External Specifications and should be consulted for details. The User-PI is the protocol interpreter which translates those user commands into the standard FTP commands for transmission to the remote Server-FTP process. Some of these commands may also require that a data connection be established between the two processes for file transfer. The Server-FTP process is activated when an FTP command connection has been made from a remote User-FTP process. The Server-PI interprets the FTP commands as they are received across that connection and replies to them depending upon certain conditions in that host. If a file transfer is necessitated by one of those FTP commands, the Server-DTP opens the data connection, supervises the transfer, and closes the connection when the transfer is complete. This FTP model will serve as the design model for this implementation

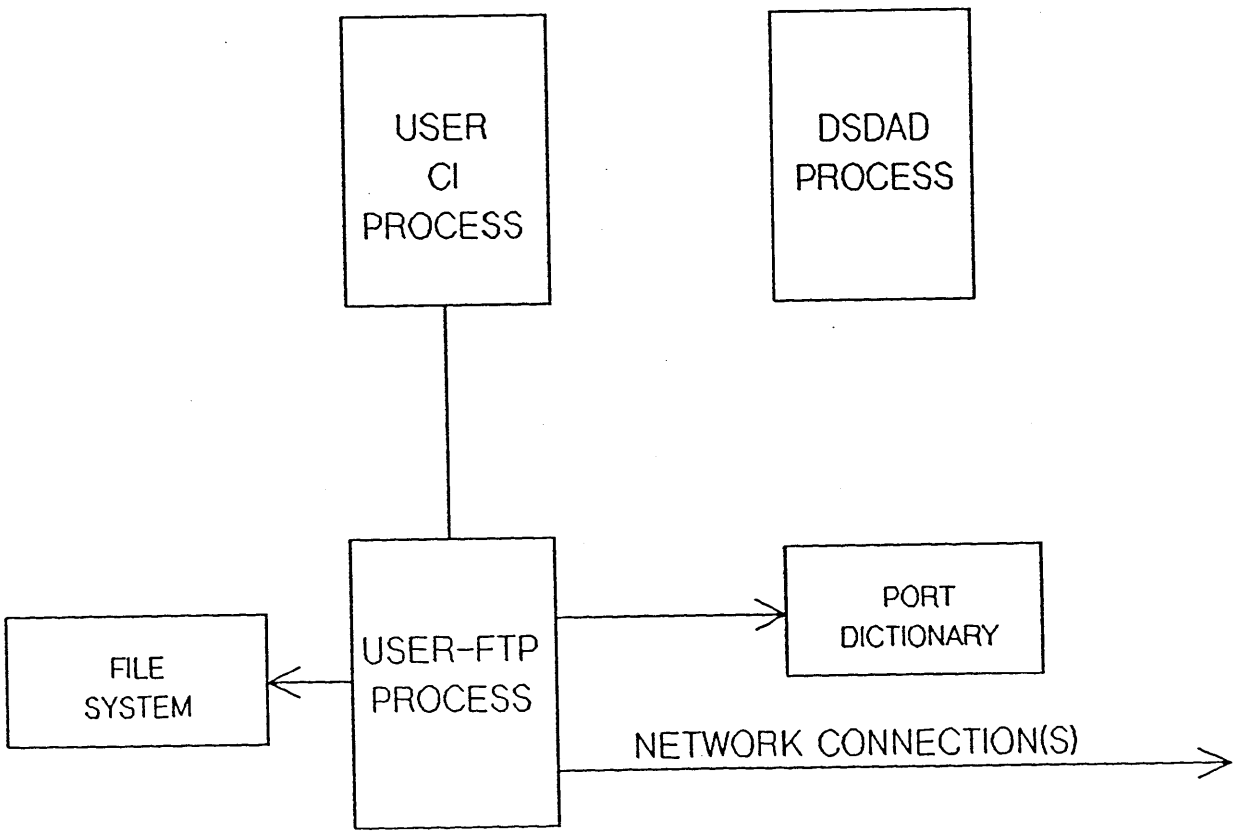
2.2 OVERVIEW OF OPERATION

This section includes discussion of other network products. It is assumed that the reader is familiar with NS Services, NS Transport, ports, NetIPC, and Network Manager; if not HP internal documents exist to aid the reader. The FTP product, as well as the other two services, will be designed to integrate into these and other existing products as much as possible. There are several advantages to this:

1) fewer system resources will be necessary for the operation of FTP e.g., if the user has NS, redundant data structures, processes, etc. will be present.
2) the release of FTP will be advanced by using these existing structures.
3) some of the products must be altered and included for DDN, e.g., TCP/IP.
4) the release of FTP will not be dependent upon any changes directly to MPE.
5) the Network Manager will have a network-independent interface for network supervision and administration.

Many of these products are making enhancements to include DDN Services. It is not the object of this document to discuss those changes. The reader should refer to DDN Services/HP3000: Investigation Report for details. We shall only include those that are directly involved in the operation of FTP.

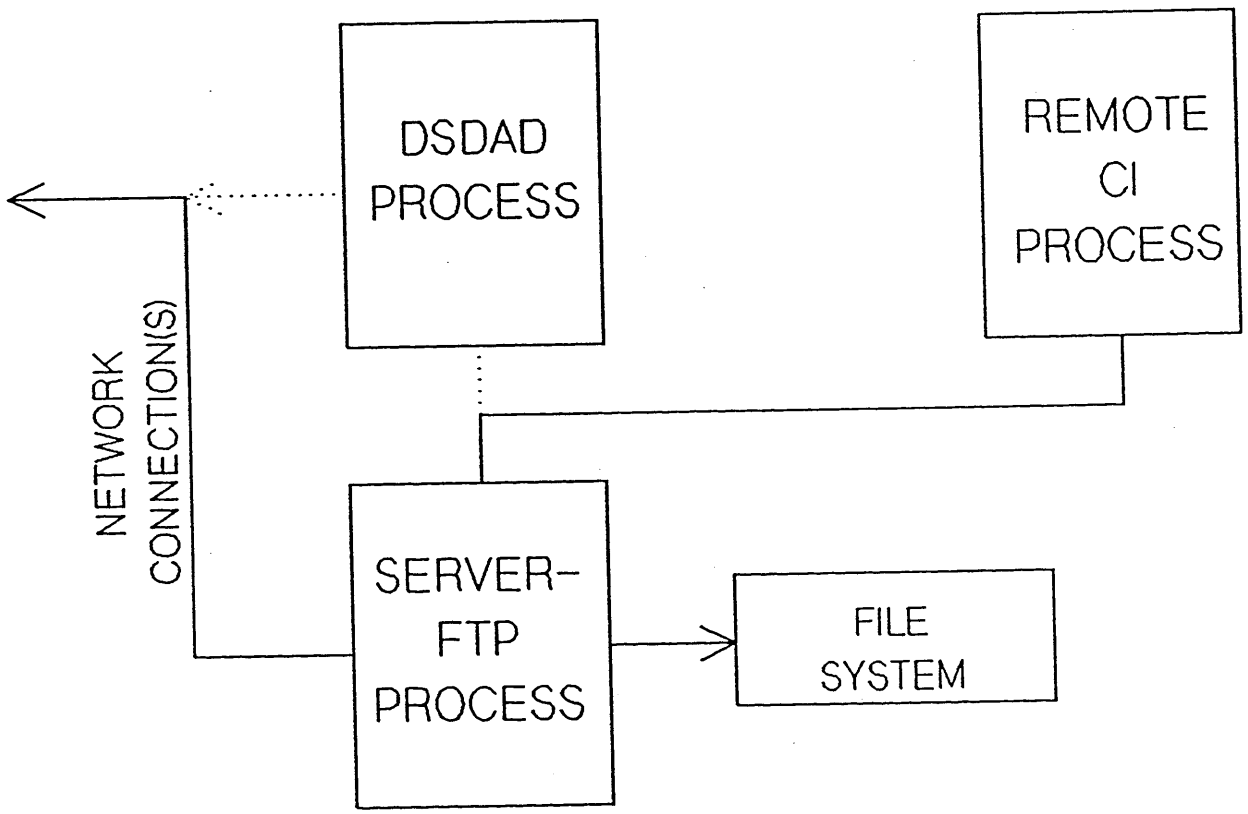
The models below are the general guide for the design of FTP/HP3000.



LOCAL FTP STRUCTURE

HOW IS THIS CONTROLLED (WHAT CHANGES WHEN DDN INSTALLED ?

1) The above state begins when the user asks the CI to run the FTP user program, thereby creating the FTP user process.
2) The FTP process will check the Port Dictionary to ascertain if the FTP service has been allowed. The "FTPL" entry in the Dictionary will be placed there by DSDAD when the Network Manager includes the service in the NSCONTROL START command. If the port entry is not found, the FTP program would exit in an error state. Note that unlike NS local server processes the user FTP process will not be a child of DSDAD at any time.
3) A NetIPC request will be made for a virtual connection. All interfacing with the lower layers of the DDN model will be through NetIPC.



REMOTE FTP STRUCTURE

The above diagram illustrates the structure on the remote 3000 and the following events.

1) If the FTP service is started in the NSCONTROL START command, DSDAD will listen on Port 21 for a connection request. When that request arrives it gives the connection to a FTP server process that it has created. The server owns the connection until it is terminated by the user. The FTP server process remains the child of DSDAD until the user logon string arrives from the FTP user process.
2) The first FTP command that should be received on the connection should be the USER command. That logon string will be used to establish a session on the 3000. A combination of MPE and NS intrinsics will be used to establish the session. A null pseudo-terminal will be obtained and the user information will be passed to DEVLOGON. When the remote session has been created by MPE the remote FTP

server will adopt itself into that session. The server will remain a part of that session until the connection is closed or a new USER command is received.

3) The FTP commands will be accepted from the command connection and will be responded to by the server. Any data transfer will occur through a data connection. That connection is only made when such a request is made and is not still open from a previous transfer. The data connection will be made on Port 20 to the user port or to any port previously indicated via a received PORT command. After completion of the data transfer the data connection may be closed according to the file structure.
4) When the QUIT command is received from the user FTP process the server should finish any data transfer that may be in progress. If no data transfer is active or when it is finished, the server process will begin to release gracefully the command connection. The server will adopt itself back under DSDAD to be either terminated or returned to a reserve pool. It will then ask MPE to log off the remote session and deallocate the remote null pseudo-terminal. DSDAD will be notified when these events have been accomplished.
5) As a consequence of the remote closing of the command connection the user FTP process will also close its side of the connection. The user process may not terminate depending upon the user command it has received. If the user FTP process is exited, the local user will return to a CI prompt after termination of the process.

2.3 MAJOR MODULES

Consistent with the operation overview given above, there will be two major modules involved in the design of FTP/HP3000.

2.3.1 User FTP Module

This process will be started when the user runs the FTP program or uses the CREATEPROCESS intrinsic. It must first check the port dictionary to ascertain if the local FTP server has been started via the NSCONTROL command. It will next initialize its data structures and all defaults as specified by the Mil-Std. If the user has included the name of the remote host or when the user does an open to a remote host, the process will request via NetIPC a connection to the remote. The process will then accept user commands through its user interface and translate those into the appropriate FTP command. All replies will be interpreted from the command connection received from the remote FTP server. Those replies will determine the next event which the FTP user must create. If a data transfer is necessary based upon the FTP command, a passive open will be done through NetIPC to await a connection request from the server process. After the data transfer the data connection may or may not be closed by the server. The process will return necessary information to the user and will return to the subsystem command interpreter.

It is the responsibility of the user process to notify the remote server that the user wishes to close the command connection and to wait for a reply to that request. It will then close its side of the connection. Upon termination of the program any open connection will be terminated before ending.

2.3.2 Server FTP Module

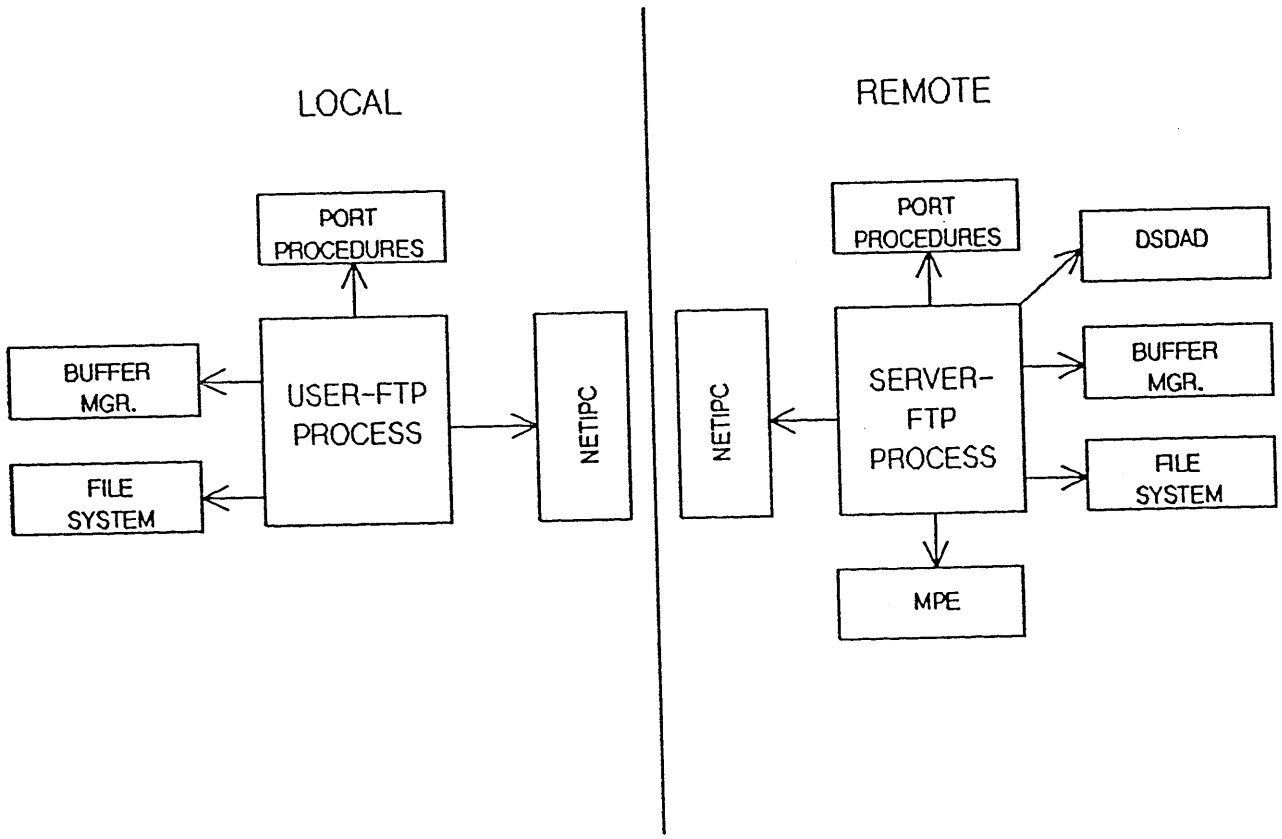
Upon creation of the remote FTP server by DSDAD it will initialize its data structures, create a port via which it can communicate with DSDAD, and await notification of a remote connection request. When the notification arrives it will respond and get the connection which DSDAD will give away. The server then awaits commands from the user process and replies according to the state on that system. The first executable FTP command must be a USER command at which point it will start operations necessary to create a session on that system and to adopt itself into that session. If any subsequent commands require

file transfer, it will establish a connection on Port 20 to the specified system, will transfer or receive the data, and may close the data connection. Upon a request to close the command connection the server must respond according to its current state and if appropriate close the connection. The server must then have the remote session logged off and readopt itself under DSDAD to be disposed of as it wishes.

2.4 MAJOR DATA STRUCTURES

No major external data structures will be directly required for FTP. No system tables will be accessed. No extra data segments will be required. User files may be accessed for transfer through the MPE file system. Other files will be accessed internally for the operation of FTP, including Network Directory, DDN catalog file, etc. Internal data structures will have to be used for information about the current state of the processes. Any file buffering will be done in the process stack via NS buffer management intrinsics.

2.5 MAJOR INTERFACES



MAJOR INTERFACES

The above illustration shows the major interfaces required for both user and server FTP processes. The

common interfaces will be addressed first and then the process-particular ones.

1) Buffer Management: NS buffer management routines will be used when they are required. No extra data segments will be necessary. These intrinsics include creating, deleting, and transferring data to and from them. The interface is well described in NS/3000: Overview Internal Maintenance Specifications.
2) File System: MPE file system intrinsics will be used to access any user files which must be transferred. These include FOPEN, FCLOSE, FWRITE, FREAD, and FCONTROL to mention only a few. The interface is well defined in MPE Intrinsics Reference Manual.
3) NetIPC: All circuit connections, controls, sends, receives, and closures will be done through NetIPC intrinsics which are defined in NS/3000 User/Programmer Reference Manual. No exception will be made for communication to lower layers.
4) Port Procedures: The user FTP process will only do a port dictionary lookup to ascertain if the service has been allowed. The server process will establish a port for communication with DSDAD. The procedure interface is outlined in Port Procedures: External Specifications.
5) DSDAD: The server FTP process will have to communicate to DSDAD using the standard port messages outlined in NS/3000: Overview Internal Maintenance Specifications. No changes to port messages should be necessary with the addition of FTP.
6) MPE: The server will also have to cause the creation and deletion of remote sessions on the system. These will be done through a combination of MPE and NS intrinsics which are defined in the above references. It will also use the COMMAND intrinsic to execute certain user commands.

2.6 PERFORMANCE CONSIDERATIONS

Performance will be a key issue in the design of FTP. As stated in earlier Product Life Cycle documents the goal is to make it comparable to that on machines similar to the HP3000. We are projecting that the performance will be equal to DSCOPY in interchange mode over a comparable connection. The Mil-Std specifies that the FTP command connection is a dialog between the user process and the server process. This means that user process must wait for a reply or replies to a previous command before sending the next. This will make the performance at the command level largely dependent upon that of the network.

There are some design features which can be incorporated within the strictures of the specification, especially for data transfer.

1) The choice of FTP Stream Mode was chosen because it calls for little file processing before transmission. The other modes call for adding headers or compressing the data. For "File" Structure it is a steady stream of bytes with implied EOF by closure of the data connection. This means that multiple send buffers can be used within the limits of the lower layers. Multiple send buffers can also be used in Record Structures since only EOF and EOR must have special handling.
2) No-wait I/O will be used as much as possible to allow the processes to be event-driven. These include IPCRECV, IPCSEND, DSDAD port, etc. This mechanism will also allow the processes to continue processing new file records while previous ones are in transit.
3) Event look-ahead will be used as much as possible. If time can be saved by using potentially idle time for other purposes, it will be so used.
4) The file transfer operations will especially be the focus of concentration for increased performance.

2.7 LOCALIZATION CONSIDERATIONS

The current plan for DDN Services is to include a catalog with the product. All keywords, delimiters, replies, and error messages will be accessed from that catalog. This mechanism will allow one to change

those things in the file and to use the MAKECAT program to make it a valid catalog file. FTP will not have any assumptions about any of the above. Keywords and delimiters will be loaded upon initialization of the processes. This only applies to the user interface; the wording and structure of FTP commands are set by the specification. No direct warning, error, or reply messages will be included in the code. All of these will be generated from the message catalog. An exception for this will be the three digit FTP reply code which is set by the Mil-Std. There will be no translation of the ascii string following the FTP reply code from a remote server.

MODULE DESIGN

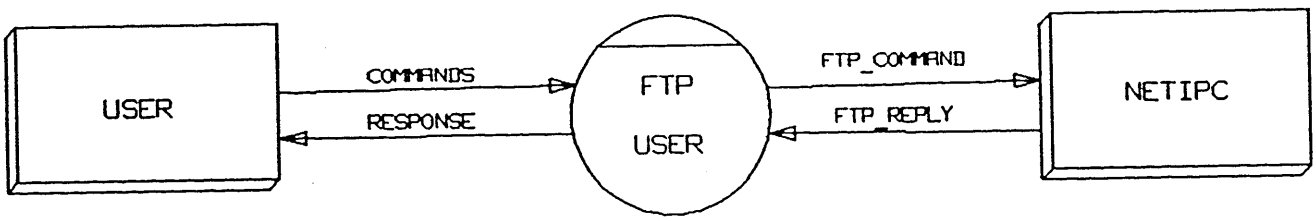
SECTION 3

The two major modules, user and server, discussed in the previous section will be outlined in greater detail. There are several transforms which will be discussed under the appropriate major module.

Structured analysis was used for this section. The figures are based on Data Flow Diagrams. The data dictionary for the user module DFD's are located in Appendix A. A separate one is available for the server module in Appendix B.

If the reader is not familiar with structured analysis of DFD, a good reference is The Pratical Guide to Structured Systems Design, by Meilir Page-Jones. His definition of DFD is "a network representation of a system, and shows the active components of the system and the data interfaces between them."[p. 59] Data flows are arrows that show movement of data between other elements of the DFD. A process is shown as a bubble which transforms the structure of the data or the information within the data. Data stores are symbolized by two parallel lines and are time-delayed repositories of data. Sources and sinks are shown as rectangles. They are the source and/or sinks of certain data flows.

3.1 FTP USER MODULE



Function: The FTP user module is the program file which will allow the user to transfer files between itself and a remote FTP server.

Interface: There are two main external interfaces for the user FTP process: 1) the user interface and 2) the NetIPC interface which need to be addressed. The external user interface has been discussed in the External Specifications for this service. This describes how the FTP program will look to the user and the subsystem commands available and should be referred to for that level of interface.

1) The FTP program can be entered only through the MPE command, RUN, for the interactive user. There will be no intrinsic as there is with DSCOPY. The user may include an optional ";INFO=<host_name>" with the RUN command, as outlined in MPE V COMMANDS Reference Manual. The host_name will be used to open a connection to the specified host. The PARM option will not be used by the FTP program. There will be no entrypoint other than the primary one.

   In addition there will be programmatic access to the FTP program through the MPE intrinsic, CREATEPROCESS, as discussed in MPE V Intrinsics Reference Manual. The INFO string can still be passed to the newly created process by using the "itemnums" options 11 and 12. Any other parameters used in either the RUN command or CREATEPROCESS intrinsic will not be directly used by the program once it has started.

2) All connection requests, connection controls, and data transmissions will be done through NetIPC procedures as outlined in NS/3000 User/Programmer Reference Manual. These include ADDOPT, INITOPT, IPCCHECK, IPCCONNECT, IPCCONTROL, IPCCREATE, IPCDEST, IPCERRMSG, IPCRECV, IPCRECVCN, IPCSEND, and IPCSHUTDOWN. This list may change depending upon the implementation of NetIPC dependencies as stated in DDN Services/HP3000 Investigation Report. There will be no other method used to communicate with the lower layers, including X.25.

Algorithm:

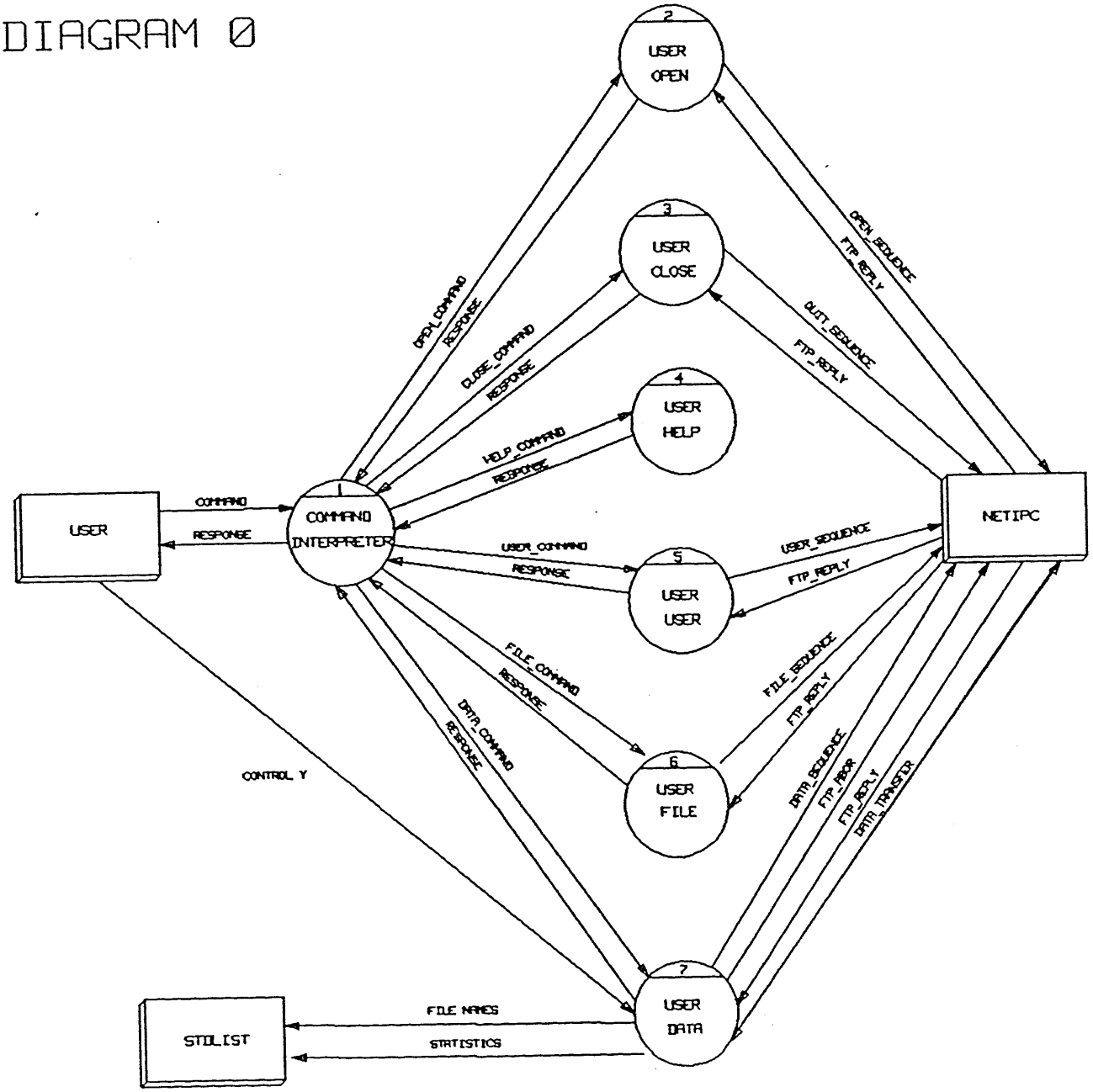
If "FTPL" not found in port dictionary then terminate with error message
Create port to receive ServerStop message in case service is aborted by Network Management
Initialize all variables and data structures
Load keywords, control Y responses, and configurable IPCSEND, IPCRECV, etc. buffers from DDN catalog file
If HOST_NAME specified in info string then request the command connection via OPEN_SEQUENCE
Enter into Command Interpreter transform (see Diagram 0)
Program terminates upon return from the Command Interpreter

Local Data Structures: The FTP user process will have a record of information necessary for opening and maintaining connections to the remote host server. This will include:

1) Remote host name
2) Command connection virtual circuit descriptor
3) Command connection state
4) Data connection virtual circuit descriptor
5) Data connection state
6) Command connection destination IP address
7) Parameter values for Type, Structure, and Mode commands
8) Internal error number
9) Last FTP command
10) Last FTP reply from remote server

This record will be accessed and possibly changed in all following transforms.

3.1.1 Diagram 0



The above figure shows the first level of the FTP user process. Each of these 7 transforms will be explained separately. The seventh, USER DATA, has its own diagram which will follow the discussion of this one.

3.1.1.1 Command Interpreter (1)

Function: The command interpreter will handle the processing of the FTP subsystem user commands. Those commands are explained in the External Specifications. The data dictionary in Appendix A contains the definitions of the grouping of subsystem commands by types. Based upon the command type, the appropriate transform will be called. If the user enters an invalid command, it will be rejected at this level. The command interpreter will not accept another subsystem command until the previous one has been finished.

Interface: The user interface has already been discussed in the External Specifications and is defined in the data dictionary. The command line will be stored in a local array which will be shared with the tranforms called by the CI. The CI will not change the connection record discussed above.

Algorithm:

Issue prompt to user and accept COMMAND
Update connection record's last FTP command
If command keyword matches one of keywords from DDN catalog then call appropriate transform
  else print error message to user
Set FTPJCW from connection record
Continue until exit command is given

Data Structures: None exist that are private to this transform.

3.1.1.2 User Open (2)

Function: It is the responsibility of this transform to open the command connection to the remote FTP server, based upon the host name, via NetIPC intrinsics.

Interface: There will be no input or output parameters for this transform. In order to accomplish its function, it will deal mostly with IPC intrinsics. IPCDEST will be called for a destination descriptor, based upon the host name parameter of the command and the reserved TCP port for FTP command connection. IPCCREATE will be called to create a call socket. IPCCONNECT will be given the call socket descriptor for a destination descriptor in order to make a connection; it will return the virtual circuit descriptor. IPCSHUTDOWN will release the call and destination descriptors. IPCRECV will return the connection reply for the virtual circuit.

Algorithm:

Create call socket if necessary and obtain destination descriptor
Request connection to remote host
Close call socket and destination descriptor
Receive reply from connection request
Update record host name, command connection vc, state, dest. IP

Data Structures: Call socket and destination descriptors will be local variables since they will not be needed outside this transform. The parts of the host record updated by the module are stated in the algorithm.

3.1.1.3 User Close (3)

Function: There are two user commands which will invoke this module. It will close the command connection if it is open. It will notify the remote server that it wishes to terminate the session and will then close all connections via IPCSHUTDOWN. It will have no input or output parameters.

Interface: NetIPC intrinsics will be used with the virtual circuit descriptor including IPCSEND, IPCRECV, and IPCSHUTDOWN.

Algorithm:

If logged onto remote system then send FTP_QUIT and analyze FTP_REPLY
If data connection open then close it
If command connection open then close it
Update record host name, command and data vc, IP, and states unless EXIT command

Data Structures: There are no private variables for this transform. It will update the connection record as noted above.

3.1.1.4 User Help (4)

Function: The help module will handle a local request for program information. This is not to be confused with the FTP HELP command, which is used to gain information about the remote server. A description of the program, of the keywords, and of the syntax for commands will be printed for the user.

Interface: There are no input or output parameters for this transform.

Algorithm:

Print help text from catalog file to output

Data Structures: There are no data structures exclusive to this transform.

3.1.1.5 User User (5)

Function: This transform will be called by the CI when a USER command has been entered by the user. It will format the proper FTP command to start a session on the remote system via the remote server. The log on string specified by the user will be passed to the remote server. It will ask the user for any passwords or account information requested from the remote server.

Interface: It has no input or output parameters. IPCSEND and IPCRECV will be used to send and receive data from the remote server on the command connection virtual circuit.

Algorithm:

```
Format FTP_USER with USER_NAME
If FTP_REPLY indicates password needed then
   Turn off echo and prompt user for PASSWORD
   Format FTP_PASS command and send to server
If FTP_REPLY indicates account needed then
   Prompt user for ACCOUNT_NAME
   Format FTP_ACCT command and send to server
Update record command state
```

Data Structures: A local character array will be used to hold password(s) and account name.

3.1.1.6 User File (6)

Function: This transform will handle all file related commands which will not require any transfer across a data connection. These are reserved for the User Data transform. These commands both include setting certain FTP parameters and remote file manipulation (i.e., renaming and purging files).

Interface: The transfer of FTP commands and replies will be done by IPCSEND and IPCRECV on the command connection.

Algorithm:

```
If TYPE_COMMAND then
   Send FTP_TYPE command
   Update record type parameter
If RENAME_COMMAND then
   Send FTP_RNFR command
   Analyze FTP_REPLY
   Send FTP_RNTO command
If PURGE_COMMAND then send FTP_DELE command
Analyze FTP_REPLY
```

Data Structures: No internal data structures are necessary for this transform. The connection record will only be updated for the TYPE command.

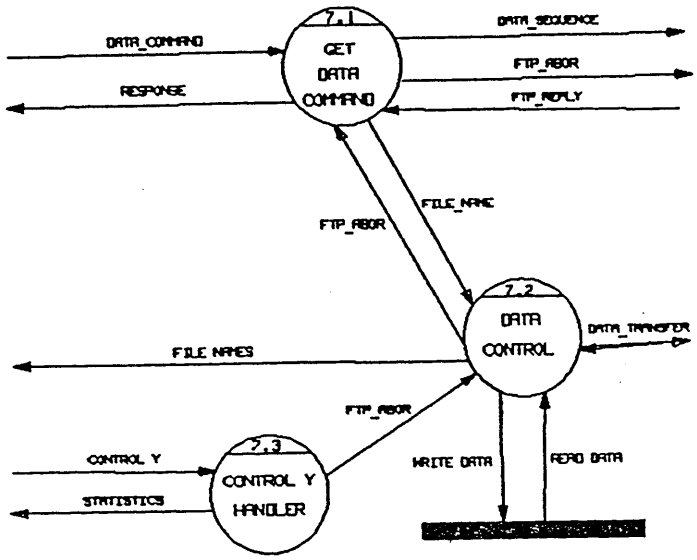
3.1.1.7 User Data (7)

Function: This transform will be called when the CI receives a command which will involve data transfer over a data connection. This will involve either sending a file to the remote, receiving a file, or receiving a list of files as specified by the user. This transform contains the local file system interface, the translation of file types if needed, and control of the data connection. The control Y handler will also be contained within this transform.

This transform is more complex than the other modules at level 0. The details will be reserved for the following section.

3.1.2 Diagram 7

DIAGRAM 7



As mentioned above the User Data transform will handle all file transfer requests including remote file listing. Because of the complexity, it has been divided into three separate transforms which will be discussed below.

3.1.2.1 Data Command Interpreter (7.1)

Function: This module will be the one called by the subsystem command interpreter when a data command is received from the user. It will request a data connection and send the necessary FTP commands on the command connection. It will enable the control Y handler (7.3) and call the data control transform (7.2) to handle the data transfer.

Interface: NetIPC intrinsics used will be the same as for User Open above in order to establish the data connection. One modification in this is that the IPCRECVCN will be done with a fully specified passive open using the destination IP address of the command connection and the TCP dedicated port for FTP data transfer (20). File system intrinsics will also be necessary. FOPEN will be used to either open a file on the local system or create a new file. Once the file has been successively opened, FGETINFO may be used to extract certain information about the file. Once the data transfer is finished, FCLOSE will be called in order to close the local file. The file system interface is described in more detail in Section 4.

NS buffer management routines will be used to allocate space, move data, and deallocate space. The NS buffer management will be discussed in Section 4.

Algorithm:

If structure parameter is not Record then send FTP_STRU command with "R" parameter and analyze FTP_REPLY
FOPEN local file if store, retrieve, or append

If retrieve or append file non-existant then FOPEN to create new file
If store file then return in error
If store operation then
  Call FGETINFO for foptions and file and record sizes
  Send FTP_TYPE if parameter not properly set and analyze FTP_REPLY
  Send FTP_ALLO command for disc allocation, record size and analyze FTP_REPLY
If necessary allocate buffer space for file transfer
If necessary open data connection with fully specified passive
Send DATA_SEQUENCE and analyze FTP_REPLY
Enable control Y for transform (7.3)
Receive connection request from remote server
Update record data vc descriptor and state
Pass control to the data control transform (7.2)
Analyze FTP_REPLY
Disable Control Y handler (7.3)
If abort requested by user through control Y then send FTP_ABOR and analyze FTP_REPLY
If STRU parameter is "F" then close IPC data connection
Close the local file

Data Structures: Local variables for this transform will be the call socket and destination descriptors. Other variables will be shared between this transform and the others at this level, including file number, transfer type (i.e., store, retrieve, or append), and file system information. Parts of the connection record will be updated as noted above.

3.1.2.2 Data Control (7.2)

Function: The Data Control transform will supervise the flow of data across the data connection. It will read or write data across the connection dependent upon the type of transfer required. Locally that may either translate into reading or writing to a local file or writing a list of remote files to the standard output device of the program. This transform will also perform any data transformation which is necessary (e.g., stripping <CRLF>) before writing a record to the file in question or adding a control byte for EOR or EOF from an FREAD before transmitting the record, per the Mil-Std. It is also the function of this transform to check the data connection state to see if it should be aborted. If that is the case, it will cease sending or receiving data and pass control back to 7.1 which is responsible for sending FTP commands across the command connection. After the successful completion of the data transfer, control will be passed back to 7.1.

Interface: The main IPC interface will be through IPCSEND or IPCRECV on the data connection virtual circuit as set up by the previous transform. Only one IPCRECV will be posted at a time, but there may be several outstanding IPCSEND's. The number of IPCSEND's will be configurable in the DDN catalog file (see section 4 on Interface). There will also be a file system interface using FWRITE and FREAD based upon the file number of the file that was opened in 7.1. A further discussion of the file system interface will be addressed in the next section.

The input parameters for this transform will be the file number of the file in question, type of transfer operation (i.e., store, retrieve, list, append), and error parameter which will be the output parameter for the module.

Algorithm:

If transfer type is for local storage or append then
  Post IPCRECV
  Transform data from Mil-Std format into record format

```
    Write data into file
  If transfer type is for local retrieve
    Read file record
    Transform as per Mil-Std
    If buffer space still available then continue
    Else send data across data connection
  If transfer type is list then
    Post IPCRECV
    Write record to output device
  Update data transfer count in connection record
  If data state indicates abort then
    Set error output parameter
    Exit to caller (7.1)
  Else continue as above until finished
```

Data Structures: The input parameters will be used, but not changed during the operation of the module. The error output parameter will be altered. There may be some minor variables involved in the module. The connection record will be checked for the abort data state and the transfer byte count will be changed after every transfer block.

3.1.2.3 Control Y Handler (7.3)

Function: This module will handle the subsystem break when the user hits Control Y. It will be enabled by 7.1 just before calling data control (7.2). It will also be disabled by the same module after completion of the data transfer module. It will print a choice of options for the user to select. These choices will be to 1) continue with the operation, 2) abort the data transfer, and 3) return status about the data transfer. The status will return the local and remote file names and the current transfer count in bytes. Asking for the abort option will set the connection record data connection state to the abort state. The data transfer module will check this state after the completion of every IPCSEND or IPCRECV. If there was a request to abort the operation, the error parameter will be set to signal it by 7.2. Transform 7.1 will actually format the FTP__ABOR request and send it on the command connection.

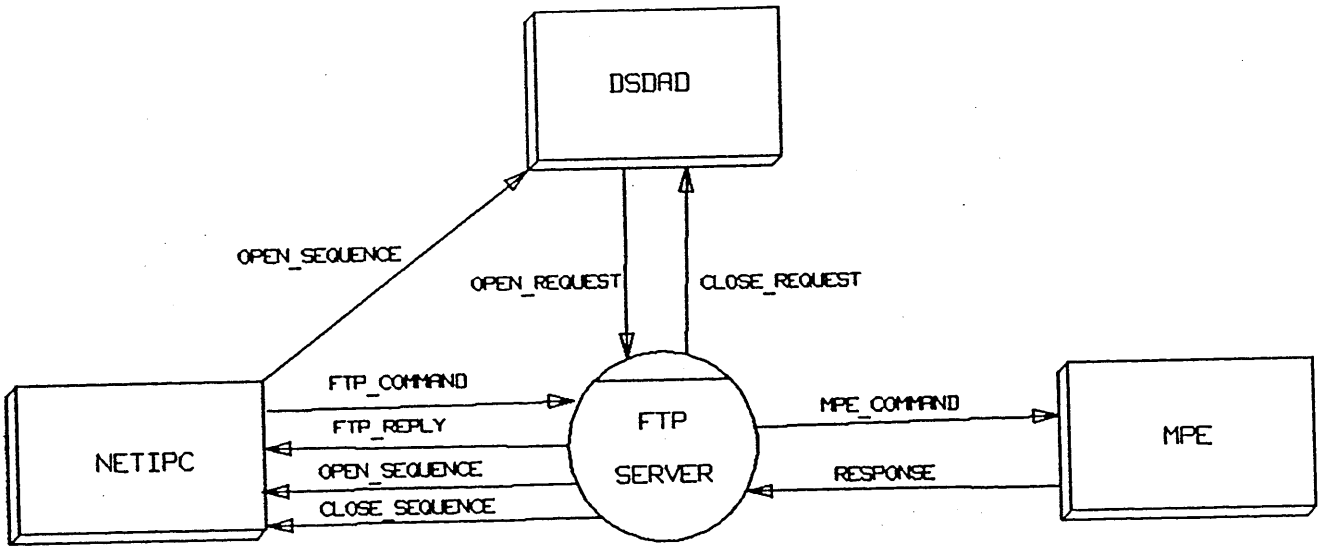
Interface: The only interface will be to the interactive user who has hit the control Y key of the terminal. The transform will print the options to the user, get the subsystem break command, and react accordingly.

Algorithm:

```
  Print 3 options to output device
  If continue option then send FTP__NOOP and exit module
  If status option then send FTP__NOOP and print status
  If abort option then set data abort state
```

Data Structures: There will be a local variable to read the user option command. The transfer byte count and data state are parts of the connection record; the file names are a part of the global command array.

3.2 FTP SERVER MODULE



Function: The server module will not be directly run by the user as was the user module. As the complement of the user program, it will serve as the program to be run for remote users on the remote system. It will be started as a child of DSDAD and will await a remote connection request from that process. It will take the command connection from DSDAD after the request has been received from the network. It will then accept FTP commands across the connection, act accordingly, and send a reply to the command. If any of those commands require data transfer, it will open a data connection, transfer the data, and eventually close the data connection.

Interface: The externals for the server are three:

1) NetIPC intrinsics will be used for communication through the network. The same procedures as outlined in the user module will be used here as well for connection opening, sending and receiving data across both connections, and connection closing. One other intrinsic, IPCGET, will also be used to accept the command connection when DSDAD has received a server request.
2) DSDAD will serve as the original parent of the server module. It will create the server when a connection request arrives to port 21 on which it should be listening, if the system manager has allowed the service to be started. It and the server will communicate to each other using port procedures. Once the server module has taken the command connection and adopted itself under the session which it will request, it will no longer be a child of DSDAD. Once the remote user has requested that the command connection be closed, the server will adopt itself back under DSDAD. All communications with DSDAD will be through NS port messages.
3) MPE will also be a major interface for the FTP server. It will include session creation and deletion, MPE commands (e.g., rename and purge), and file system intrinsics. Each of these will be addressed in the following subsections.

Algorithm:

Initialize all variables and data structures
Create port for communication with DSDAD
Enter name of server (FTPSERxx) in Port Dictionary

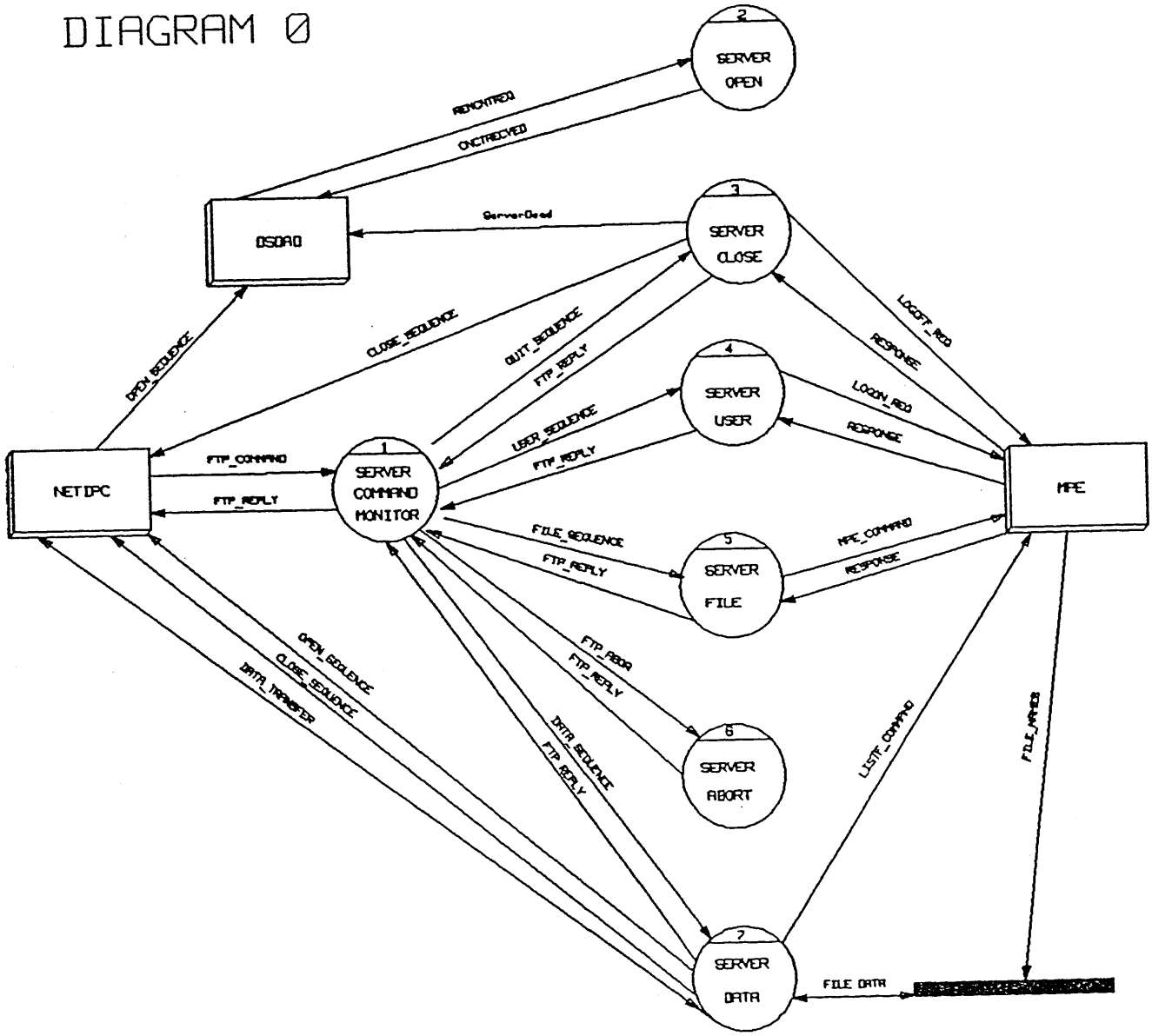
Load keywords from DDN catalog file
Call Server Open to await REMCNCTREQ from DSDAD
Call Server Command Monitor
When Command Monitor returns
  Close command connection
  Send DSDAD a SERVERDONE message for termination

Data Structures: The server will have a connection record similar to that of the user module with a few changes. This will be shared by all transforms. These include:

1) Command VC descriptor
2) Command VC source IP and port address
3) Command connection state
4) Data VC descriptor
5) Data VC destination IP address and port
6) Data connection state
7) Parameter values for Type, Structure, and Mode commands
8) Internal error number
9) Last FTP reply error code
10) Last FTP command received
11) Local user process id number
12) Rename from file name
13) User session ID number
14) Pseudo-terminal ldev
15) FTP_ALLO parameters

This record can be altered by all of the following transforms. There will also be a global array to hold the command line.

3.2.1 Diagram 0



Each of the seven transforms above will be addressed separately in this section. They roughly correspond to the seven transforms outlined for the user module in its Diagram 0.

3.2.1.1 Server Command Monitor (1)

Function: This transform is similar to the command interpreter of the user module. Its commands, however, come from the command connection and consist of FTP commands rather than user commands. It will determine which of the transforms numbered 3-7 will be called to handle the request. It will always be the transform receiving requests from the command connection. Transform 2, Server Open, is

not called by the Command Monitor since it is not designed to handle any particular FTP command. It is only used to handle communication with DSDAD in setting up the command connection.

Interface: The server external interface will be through NetIPC intrinsics as outlined earlier. It will only communicate over the command connection and, in no circumstances, over the data connection.

Algorithm:

Issue nowait IPCRECV in order to receive an FTP_COMMAND
If Telnet negotiation then refuse request
If not a valid or supported command then reject
Else call transform 3-7 based on FTP_COMMAND
Continue after transforms return until FTP_QUIT processed

Data Structures: No private major variables will be needed by this module. It will have no input or output parameters. It will have the FTP commands hard-coded unlike the user commands, because they are set by the standard and should not be changed by customers.

3.2.1.2 Server Open (2)

Function: This transform will receive the command connection from DSDAD in preparation for the server monitor module. It will communicate with DSDAD; the server command monitor will communicate with the network. This was done to isolate these two functions.

Interface: Port procedures will be used to communicate with DSDAD through NS port messages. The NetIPC intrinsic, IPCGET, will be used to get the connection being given away by DSDAD.

Algorithm:

Wait for REMCNCTREQ port message from DSDAD
Get command connection based upon give name in port message
Update connection record command connection VC, state, source IP and port
Send DSDAD CNCTRECVED port message
Go to Server Command Monitor (1)

Data Structures: There are no private major variables for this transform. It will update parts of the connection record as outlined above.

3.2.1.3 Server Close (3)

Function: This transform will be called by the command monitor whenever an FTP_QUIT command is received from a user process. It will handle the request as stated in the Mil-Std, i.e. the command connection will not be closed if file transfer is taking place. When the module is ready to act upon the request to close the command connection, it will adopt itself back under DSDAD, kill the local session, and notify DSDAD that it has finished its function.

Interface: NetIPC IPCSEND will be used to transmit the appropriate FTP replies and messages. NS session handling procedures will be used as well as port messages. MPE will be requested to stop the user session via the ABORTSESS intrinsic.

Algorithm:

If file transfer in progress then update connection record command state

Else
Close data connection if open
Send FTP_REPLY for FTP_QUIT command
Close command connection gracefully
Adopt server back under DSDAD process
Call Abortsess to destroy session
Deallocate null pseudo-terminal
Send SERVERDONE port message to DSDAD
Update connection record

Data Structures: No private variables will be used. It will use the connection record and update it.

3.2.1.4 Server User (4)

Function: Server User will handle the FTP_USER command. It is this transforms's responsibility to have a user session created on the 3000. The user_name string passed with the user should have enough information to create a session on the 3000. The only thing that can be omitted are passwords for account, group, and/or user. Once the session has been successfully created, the transform will adopt the server into that session.

Interface: NetIPC intrinisics will be used to send replies to the user program. The transform must have some method of getting a null pseudo-terminal through Telnet (these details have not been designed by the Telnet engineer). The MPE procedure DEVLOGON will be used to create the session. Any requests for passwords will be processed and requested from the user as per the Mil-Std. ADOPT' procedure will be used to change the parent of the FTP server.

Algorithm:

If process is not already logged on then
Get a null pseudo-terminal from Telnet
Else
Adopt back under DSDAD
Abortsess existing user process
Call Devlogon with USER_NAME
If passwords required then send the FTP_REPLY and string to specify which password
Receive FTP_PASS and call Devlogon
Adopt server under new user process from DSDAD
Send completion FTP_REPLY
Update connection record

Data Structures: There will be some local variables for this transform, which will not be needed later, including password strings, etc. Parts of the connection record will be altered, especially the command connection state, pseudo-terminal ldev number, and user session id number.

3.2.1.5 Server File (5)

Function: This transform will handle those commands that do not require file transfer over the data connection or those already covered. Those include setting the type, structure, and mode parameters as well as some commands that will interface to MPE/file system.

Interface: The interface with the network will be through NetIPC intrinsics over the command connection. The MPE COMMAND intrinsic will be used to perform the renaming of a file and the purging of a local file.

Algorithm:

If FTP_RNFR command then Update connection record
If FTP_RNTO command then
  Check that preceding command was FTP_RNFR
  Call Command intrinsic using Rename with FILE_NAMEs
If FTP_DELE command then call Command intrinsic using Purge command with FILE_NAME
If FTP_TYPE, FTP_STRU, or FTP_MODE command then update connection record if parameter is supported
If FTP_ALLO command then update connection record
If FTP_PORT then update connection record
Send FTP_REPLY

Data Structures: There will be no internal variables. The connection record state, rename from file name, reply code, etc. will be altered.

3.2.1.6 Server Abort (6)

Function: This transform will mark the connection record in order to signal the following transform that the user has requested that the data transfer be halted. The record will be checked before each IPC transfer on the data connection.

Interface: The only interface for this transform is to send an FTP reply back for the FTP_ABOR command.

Algorithm:

If data transfer in progress then mark abort in connection record data state
Send FTP_REPLY

Data Structures: There will be no internal variables for this transform. It may alter the connection record as noted above.

3.2.1.7 Server Data (7)

There will be a brief description of this tranform. Most of the discussion will be found in the following discussion under Diagram 7.

Function: All data transfer requests and the resulting file transfers will be performed here within.

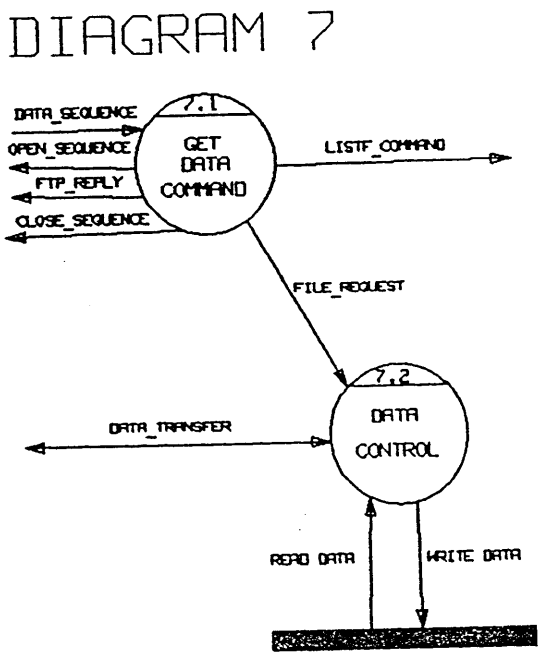
Interface: Both the command and data connections will be used via NetIPC intrinsics. File system intrinsics will also be used (see discussion under Interface Design section).

Algorithm:

Receive file transfer request
Open data connection if not already opened
Open file and transfer
Close data connection if "file" structure
Close file

Data Structures: These will be discussed in the following section.

3.2.2 Diagram 7



As mentioned in the previous section the Server Data transform has more than one part. The two in question are shown above. One is in charge of the command connection and the other of the data connection. Both will interface to MPE and file system.

3.2.2.1 Data Command Interpreter (7.1)

Function: Transform 7.1 will be called by the Server Command Monitor for any of the data sequence commands requiring data transfer over a data connection. It will analyze the command and set up the necessary steps to complete it. It will open the files, request transform 7.2 to transfer the file in question, and close the file. It may open and close the data connection. It will also ask MPE to make a list of files in a temporary file for transfer to the user module.

Interface: There are no input or output parameters for this transform. Its main interface will be with the following module, 7.2. It will also be receiving and sending replies through NetIPC intrinsics over the command connection only. It will use the MPE intrinsic, COMMAND, with the LISTF command. It will provide a temporary file name for the optional listfile. In that way the following module can transfer it as it would any other file. It will also use NS buffer management routines (see Interface Design section following this one).

Algorithm:

```
If FTP_NLST then call Command using Listf with FILE_NAME and listfile
FOPEN file name or temporary listfile
If retrieve command and file non-existant then
   Send FTP_REPLY
   Return from transform
If file non-existant then create file using parameters from connection record
Establish buffer space if not already done
Open data connection with source TCP port and destination IP/port indicated if not already open
```

Send intermediary FTP_REPLY
Call Data Control (7.2)
Close local file
Send final FTP_REPLY

Data Structures: The transform will use the connection record for establishing the data connection. Some internal, unshared variables will be used to set up a call socket and destination descriptors.

3.2.2.2 Data Control (7.2)

Function: It will supervise the file transfer over the data connection. It will not use the command connection. It will supervise the transfer, transform the data if necessary, and abort the data transfer if necessary.

Interface: It will use NetIPC intrinsics to send and receive the data depending upon the direction of transfer. It will also use NS buffer management procedures to handle the buffering of file records. File system intrinsics will be used to read records to and from the local file. The input parameters to this module will be the file number for the file involved in this transfer, the transfer type (i.e., store, retrieve, append), and a variable in which to return the status of the transfer back to module 7.1.

Algorithm:

If abort data state then return with error condition
If retrieve type then
  Read record
  If type parameter is File then add <CRLF> to end of record
  Else add Record control bytes for EOR and EOF
  Send buffer to remote when full
Else
  Issue IPCRECV for file data
  Transform data to 3000 file structure
  Write record to file
Continue until EOF or data connection closed from remote

Data Structures: A few local variables will be necessary. Other than that, the connection record will be used.

INTERFACE DESIGN

SECTION 4

There are three major interfaces which will be discussed in this section. They are:

1) DDN Catalog
2) NS Buffer Management Routines
3) File System

4.1 DDN CATALOG

There will be several message sets in the catalog file for all DDN services (SMTP, FTP, and Telnet). FTP will require 3 message sets. These include:

1) User keywords, delimitors, configurable parameters, control Y response/action, and user module error messages
2) User module help facility text
3) Server module reply messages

The catalog file will be opened by both processes. The user module will load the keywords, delimitors, and control Y information into a search dictionary in the process stack. These will be used to parse the FTP user subsystem commands as well as the control Y response. The configurable parameters will include number and size of data buffers; this will allow for changing easily during testing and for optimizing performance at customer sites.

The help set will only be accessed when the user asks for it. The GENMESSAGE intrinsic will be used to read the text from the catalog file. The same procedure will be used for FTP error messages for the user module.

The server module will have the 3 digit reply codes hard-coded into the body of the program. The accompanying ASCII string message will be pulled from the catalog file when needed.

MPE error messages will be taken from the file CATALOG.PUB.SYS based upon the CIERROR number. File system error messages will be returned from the FERRMSG intrinsic after FCHECK has been called.

The catalog file will be closed before termination of either the user or server modules.

4.2 NS BUFFER MANAGEMENT

NS buffer management routines will be present on systems with DDN services. These routines are discussed in NS/3000 Overview Internal Maintenance Specifications. It is not our object to discuss these in this document. We wish to limit ourselves to discussion of buffer space usage.

Buffers will be located in the DL-DB area of the user or server stack. An extra data segment buffer will be avoided. The buffer space will include buffers for IPCRECV and IPCSEND as well as for data transformations necessary before the send or after the receive (FTP Mil-Std, p. 12-13,15). The latter will consist of one buffer based upon the record size of the file in question. If the record size is unknown, as

may often be the case, certain assumptions will have to be made to compensate for the shortcomings of the standard. The discussion of file system interface follows.

As far as concerns the buffer space for the IPC intrinsics, there will be several factors to determine the size and number. The number and size of send buffers will be determined by configured parameters in the catalog file. Defaults will be specified in the code in case they are missing, they will be 2 for the number of outstanding sends and 4096 bytes for the size of each buffer. These two values may be changed before product release, if it is determined that performance would improve.

The receive buffers are a different matter since only one IPCRECV can be outstanding at one time in nowait mode. The same buffer size as above will be used. The same number of buffers will also be used. This will allow the processes to post another receive before emptying the buffer from the previous one. If it is determined that performance will not be improved by this method, the number of buffers will be reduced to the fixed number of 1.

4.3 FILE SYSTEM

FTP was designed to be a file transfer program between heterogenous systems as well as homogenous. In consideration of different implementations of systems' file systems, the Mil-Std has allowed little opportunity for one system to describe the file which is to be transferred. The NS version of NFT follows an HP internal standard which allows one HP system to describe the file in question in great detail. Among these are record size, file size, file code, file disposition, etc. None of these are required for minimum implementation of FTP, and, therefore, we can not assume that all implementations will have them. Indeed, there is no mechanism in the standard for passing any of this information save the file and record size. As a consequence several assumptions will have to be made about the file in advance. These assumptions can be overwritten by the user via the MPE BUILD command, if it is seen that the assumptions are not adequate for a particular file. The rest of this section will not direct itself to the discussion about the interface with the file system and the intrinsics that will be used for file transfer. That was discussed in Section 3 where applicable. We shall devote ourselves to discussing the assumptions and the impact that they will have on the program and the 3000.

4.3.1 File Structure

The Mil-Std has made allowances for three types of file structures. Those are 1) file, 2) record, and 3) page. File and Record are part of the minimal implementation and are supported by this product. The 3000 file structure is based upon records. We must, however, be able to accept files that are not structured into records. Both the user and server modules will have the capability to transform records into a "file" structure and to reverse the process. End of records will be changed into <CRLF> and the reverse will be done upon receipt of file data. EOF will be done by closing the data connection. Whenever the file structure is "record" based, which is the case of 3000-3000 transfer, a control byte followed by either a EOR or EOF marker will be inserted into the record before transmission. The control byte and marker will be removed before the file is stored. The transition should be reversable, that is a data stored with either record or file structure can be retrieved, sent to the original producer, and match the original file, although the file structure may be different.

File structure will also affect the buffering and data transformations. File structure requires only a <CRLF> placed at the end of the record. For that reason we shall only use the send buffer directly for the FREAD target. If it is determined that the next record will not fit into the remaining space of the send buffer, the FREAD target will be a record-size buffer and the remaining send buffer filled from the beginning of the record data. The rest of the record will be moved into the next send buffer before the next file read if one is available. Upon receiving data into the receive buffer, a scan will be done for the

<CRLF>, the preceding data written to the file by a FWRITE, and skip the <CRLF> to the next record. An extra record-size buffer will be used to hold any partial records that may be received in one IPCRECV. If no <CRLF> is discovered before the record-size buffer is filled, the data will be written as one record.

A scan must be done of the file data in record structure. For that reason a record-size buffer will be used for the FREAD. If a data byte is the same as a control byte, it and the preceding data will be moved to the send buffer, the control byte added to the send buffer, and the scan continues. If the EOR or EOF is encountered, the preceding data will be moved to the send buffer and the control byte/marker sequence added. Data in the receive buffer may not be moved to the record-size buffer before the FWRITE, if a double control byte is not encountered. The record-size buffer will be used for a split record as mentioned above. If the EOR is not encountered before the record buffer is filled, the data will be written.

It has been decided to not allow the user to set the file structure parameter. The fact that the 3000 file system is based upon a record format can not be changed. The user module will always send a "STRU R" command to the remote server. Although both record and file are required for all implementations, this author has seen some implementations which will not accept this command. For that reason this implementation will be designed, as mentioned above, to transform that data within the user module if Record is refused by the server. There is no mechanism within the standard for the server process to notify the user process of the file structure, if it is to be the producer in the transaction. Therefore we will design the server module to perform both transformations as well.

4.3.2 File Mode

It has been decided to support only the STREAM type of transmission mode. For that reason the user will not have the option of setting it to different modes. All requests for other options will be refused by the server. The ramifications of this upon the data transfer and the file structure have been discussed above. The user module will have the capability to send a MODE command, but there will be no opportunity since stream mode is the default. It will be included in the design for completeness in case of future enhancements.

4.3.3 File Type

Two file types will be supported by this implementation, ASCII and binary. We have made the decision to allow the user to set this parameter. This has been done only so the user process might have at least some information about the file as it exists on the remote server. Like so many of the file commands, this one is only allowed from the user process to the server, which does not allow the user to have any idea about the remote file it may be receiving. The user will be encouraged to set this parameter in case of remote-to-local transfer, in case the default, ASCII, should not be used. For local-to-remote files, the user module will use the FGETINFO intrinsic to determine whether the file is ASCII or binary and send a TYPE command, if it is not the current parameter value.

4.3.4 FOPEN Assumptions

Since we may have little information important for accepting a file from a remote file system, certain assumptions or defaults must be used. The standard does specify that an existing file must be overwritten if it does exist. That will provide the user with a mechanism for setting many of the parameter specifications as set in the FOPEN intrinsic. The BUILD command may be the best method of accomplishing this.

The following subsections will discuss this for user as source, user as target, server as source, and server as target. This is necessary because the relationship between the user and server is not a balanced one. We will then address the consequences of the append command for the server process.

4.3.4.1 USER AS SOURCE. This is the best of the four options. The user process will have information about the file and the ability to pass some of that information on to the server. The file will be opened with the name specified by the user and foptions=%2 (old permanent or temporary file). If the file does not exist or can not be accessed, an error condition will be reached. FGETINFO intrinsic will then be called with the file number to ascertain the following:

1) FOPTIONS - (13:1) will tell us if the file is ASCII or binary. If the current TYPE parameter is not set to that of the file, a new TYPE command will be sent to the server process.
2) FLIMIT - The maximum number of records which could exist in this file. This with the following information will be used to calculate the space that should be allocated for this file.
3) RECSIZE - The size of each record (positive for words, negative for bytes). This and the preceding parameter can be used to calculate the first parameter for an ALLO command.

   If recsize is positive: RECSIZE * 2 * FLIMIT
   If recsize is negative: -(RECSIZE) * FLIMIT

   The second parameter of the ALLO command will be RECSIZE, converted to positive number of bytes. RECSIZE will also be used to calculate the size of the record buffer, discussed under buffer management above, for data target in calls to FREAD.

The source file will be closed and unchanged.

4.3.4.2 USER AS TARGET. Unfortunately the remote server process has no way to change the STRU or TYPE parameters and can not pass any information similar to the ALLO command. The STRU will be set to Record by the 3000 user process. The type parameter will be left as is since we have no way to know what the type will be. The user should be made aware of the advantage of setting the type in advance.

FOPEN will be called using the target file name, or the source file name if the target name is to default to that. The foptions will be %3 to ascertain if the file already exists. If it does then the existant file will be overwritten without changing any of the other parameters that were set when the file was created. Aoptions will be %101 to claim exclusive access and to write over any data which may already be in the file. FGETINFO will be called to return the RECSIZE parameter. That information will be used as above to determine the size of the record buffer.

If the previous FOPEN returns an error code indicating that the file is non-existant, a second FOPEN will be done to create the file. The following parameters will be specified:

1) FORMALDESIGNATOR - The target file name or the source file name if the optional target name was not included.
2) FOPTIONS - This parameter will vary according to the TYPE parameter set in the connection record. If the parameter is ASCII, foption will be %104; if binary, %100. The other bit that is set in both cases is the record format specifications. If the user process has no information about the file it will be receiving, it will open the file using variable-length records. The actual useful data size of the record will be determined by the FWRITE's to the file. This mechanism is the best option available to us.
3) AOPTIONS - This parameter will be %101. The first bit opens the file exclusively for this process. That will be done to insure that no other process is accessing the file since we must overwrite the information that is in the file before accepting the new file data. The second bit is write access

only. This will purge the current contents of the file, set EOF to 0, record pointer to 0, and accept only writes to the file.

4) RECSIZE - This value will be determined by the TYPE parameter. If the file type is ASCII, the record size will be 256 bytes. If the file type is binary, it will be 128 words. Although the record size is the same in both cases, binary files are always expressed in words and ascii files in bytes.

The rest of the FOPEN parameters will not be specified in the creation of the file. All the defaults will be in effect, e.g., filecode=0, filesize=1023.

4.3.4.3 SERVER AS SOURCE. Although the server will have as much information about its file as the user process does when it is the producer, there is no mechanism in FTP for the server to share that information with the user file system. FOPEN will be used with source file name and foption %3. If the file is non-existant or unavailable, it will be closed and a negative reply will be sent the user. FGETINFO will be called to determine the record size of the file; this will be used to determine the size of the record buffer.

4.3.4.4 SERVER AS TARGET. The first thing to determine is whether the file exists before the data transfer commences. FOPEN will be called with the specified file name, foption=%3, aoption=%101. If the file exists, it will simply be overwritten. FGETINFO will be called to extract the record size for record buffer allocation.

If the file does not exist, certain different parameters and commands can determine the way in which we will want to open the file. The TYPE parameter will tell us if the file will be ASCII or binary. STRUCTURE and MODE parameters will not influence the way in which the file is created, only in the transformation we must do before writing to the file. The server may or may not have been sent an ALLO command by the user process. We shall examine these different possibilities in the following parameter discussion.

1) FORMALDESIGNATOR - The file name sent with the store command.
2) FOPTIONS - %100 for binary files and %104 for ASCII. See "User as Target" above for details.
3) AOPTIONS - %101 (see above).
4) RECSIZE - 128 words or 256 bytes will be the default (see above). If an ALLO command preceded the STORE command, the record size parameter will be used for this parameter. The file will still remain in variable-length format.
5) FILESIZE - This can only be specified when an ALLO command has preceded the STORE command. This will be computed in reverse of the formula above. Blockfactor of 1 will be assumed as well.

The other FOPEN parameters will use the defaults provided by the file system.

4.3.4.5 SERVER APPEND. Append file actions are very similar to the those taken for the store command. If the file does not exist before the transfer, the actions and parameters are exactly the same as above, basically an append to a non-existant file is the same as a store to that file. If the file does exist, aoptions will change to %103 (append access only).

IMPLEMENTATION PLAN

SECTION 5

5.1 CODING CONVENTIONS

Pascal will be the language used for the FTP modules. In-code comments will constitute about 40-50% of the final finished source code. As mentioned in other documents, all 3000-specific type instructions will be avoided in order to provide easier portability of the finished product. Standard ANSI Pascal will be used as much as possible, avoiding the Pascal/3000 adaptations. Each transform will be well-documented at the beginning with global data structures used and/or changed, input and output parameters, and a discussion or overview of the transform's function within the program. Further comments will be added within the transforms where further elucidation is considered important.

5.2 DESIGN ORDER

The basic philosophy behind the order of design will be to divide the user process into manageable parts based on the Data Flow Diagrams starting at the top and working down to lower sub-modules. Upon the completion of one part the corresponding part of the server process will be developed, if there is one. In this way the interplay of the parts can be tested before the whole of one process is finished. The following list gives the order in which the transforms will be developed using "(U1)" to signify User Transform 1.

1) User interface and initialization of user process (U1)
2) Outer block of server (S1)
3) FTP commands for opening and closing connections (U2&U3)
4) Server processing of opening and closing connections and with proper reply codes (S2&S3)
5) FTP user creation command (U5)
6) Session creation and deletion for server process (S4)
7) Other FTP commands which will not require data transfer across the data connection (U6)
8) Server processing of commands from 7) above (S5)
9) User commands involving eventual data transfer including file system interface (U7)
10) Server handling of data transfer requests and file system interface (S7)
11) Server handling of data transfer abort (S6)
12) User's help facility (U4)

5.3 DESIGN SCHEDULE

The schedule is based upon the above 12 parts of the design order.

1) 2/3/86 - 2/17
2) 2/17 - 3/3
3) 3/3 - 3/17
4) 3/17 - 3/31
5) 3/31 - 4/14

6) 4/14 - 4/28
7) 4/28 - 5/5
8) 5/5 - 5/19
9) 5/19 - 6/9
10) 6/9 - 7/14
11) 7/14 - 7/28
12) 7/28 - 8/4

USER DATA DICTIONARY

APPENDIX

A

COMMAND = [OPEN__COMMAND | CLOSE__COMMAND | HELP__COMMAND | USER__COMMAND | FILE__COMMAND | DATA__COMMAND]

OPEN__COMMAND = "OPEN" + HOST__NAME

USER__COMMAND = "USER" + USER__NAME

CLOSE__COMMAND = "CLOSE" | "EXIT"

HELP__COMMAND = "HELP"

FILE__COMMAND = [TYPE__COMMAND | RENAME__COMMAND | PURGE__COMMAND]

DATA__COMMAND = [LISTF__COMMAND | GET__COMMAND | PUT__COMMAND | APPEND__COMMAND]

TYPE__COMMAND = "TYPE" + ["ASCII" | "BINARY"]

GET__COMMAND = "GET" + FILE__NAME (+ FILE__NAME)

PUT__COMMAND = "PUT" + FILE__NAME (+ FILE__NAME)

APPEND__COMMAND = "APP"("END") + FILE__NAME (+ ":" + HOST__NAME) + " " + FILE__NAME

RENAME__COMMAND = "RENAME" + FILE__NAME + "," + FILE__NAME

PURGE__COMMAND = "PURGE" + FILE__NAME

LISTF__COMMAND = "LISTF" (+ FILE__NAME)

HOST__NAME = 1{alpha} (+ 0{alpha | digit | "." | "-"}61 + 1{alpha | digit})

FTP__COMMAND = [OPEN__SEQUENCE | QUIT__SEQUENCE | USER__SEQUENCE FILE__SEQUENCE | DATA__SEQUENCE | FTP__ABOR]

OPEN__SEQUENCE = IPC connection

QUIT__SEQUENCE = (FTP__QUIT) + IPC shutdown

USER__SEQUENCE = FTP__USER (+ FTP__PASS) (+ FTP__ACCT)

FILE__SEQUENCE = [FTP__DELE | FTP__RENAME | FTP__STRU | FTP__MODE | FTP__TYPE]

DATA__SEQUENCE = [FTP__STOR | FTP__RETR | FTP__APPE | FTP__NLST] + (FTP__PORT) + (FTP__ALLO) + (FTP__NOOP)

FTP__USER = "USER" + USER__NAME

FTP__PASS = "PASS" + PASSWORD

FTP_ACCT = "ACCT" + ACCOUNT_NAME

FTP_DELE = "DELE" + FILE_NAME

FTP_RENAME = FTP_RNFR + FTP_RNTO

FTP_REFR = "RNFR" + FILE_NAME

FTP_RNTO = "RNTO" + FILE_NAME

FTP_STRU = "STRU" + ["F" | "R"]

FTP_MODE = "MODE S"

FTP_TYPE = "TYPE" + ["A" | "I"]

FTP_STOR = "STOR" + FILE_NAME

FTP_RETR = "RETR" + FILE_NAME

FTP_APPE = "APPE" + FILE_NAME

FTP_NLST = "NLST" + (FILE_NAME)

FTP_PORT = "PORT" + IP_ADDRESS + PORT_NUMBER

FTP_ALLO = "ALLO" + integer (+ " R " + integer)

FTP_ABOR = "ABOR"

FTP_NOOP = "NOOP"

FTP_QUIT = "QUIT"

IP_ADDRESS = 4{0..255}

PORT_NUMBER = 1..64K

PASSWORD = 1{ascii printable characters}*

ACCOUNT_NAME = 1{ascii printable characters}*

FILE_NAME = 1{ascii printable characters}*

USER_NAME = 1{ascii printable characters}*

FTP_REPLY = 1{1..5} + 1{0..5} + 1{0..9} + ASCII_STRING

ASCII_STRING = 128 printable ASCII characters, except <CR> and <LF>

SERVER DATA DICTIONARY

APPENDIX B

FTP_COMMAND = [OPEN_SEQUENCE | CLOSE_SEQUENCE | USER_SEQUENCE | FILE_SEQUENCE | DATA_SEQUENCE | QUIT_SEQUENCE | FTP_ABOR]

OPEN_SEQUENCE = IPC connection

QUIT_SEQUENCE = FTP_QUIT + IPC shutdown

CLOSE_SEQUENCE = IPC shutdown

USER_SEQUENCE = FTP_USER + (FTP_PASS)

FILE_SEQUENCE = [FTP_DELE | FTP_RENAME | FTP_STRU | FTP_MODE | FTP_TYPE FTP_PORT | FTP_ALLO]

DATA_SEQUENCE = [FTP_STOR | FTP_RETR | FTP_APPE | FTP_NLST] + (FTP_ABOR) + (FTP_NOOP)

FTP_USER = "USER" + USER_NAME

FTP_PASS = "PASS" + PASSWORD

FTP_DELE = "DELE" + FILE_NAME

FTP_RENAME = FTP_RNFR + FTP_RNTO

FTP_RNFR = "RNFR" + FILE_NAME

FTP_RNTO = "RNTO" + FILE_NAME

FTP_STRU = "STRU" + ["F" | "R"]

FTP_MODE = "MODE S"

FTP_TYPE = "TYPE" + ["A" + ("N") | "I"]

FTP_STOR = "STOR" + FILE_NAME

FTP_RETR = "RETR" + FILE_NAME

FTP_APPE = "APPE" + FILE_NAME

FTP_NLST = "NLST" + (FILE_NAME)

FTP_PORT = "PORT" + IP_ADDRESS + PORT_NUMBER

FTP_ALLO = "ALLO" + integer + "R" + integer

FTP_QUIT = "QUIT"

FTP__ABOR = "ABOR"

FTP__NOOP = "NOOP"

IP__ADDRESS = 4{0..255}

PORT__NUMBER = {1..64K}

FTP__REPLY = 1{1..5} + 1{0..5} + 1{0..9} + ASCII__STRING

ASCII__STRING = 128 printable ASCII characters, except <CR> and <LF>

PASSWORD = 1{ascii printable characters}8

FILE__NAME = 1{ascii printable characters}26

USER__NAME = 1{ascii printable characters}*

Section 1
PRODUCT IDENTIFICATION

Section 2
DESIGN OVERVIEW

Section 3
MODULE DESIGN

INTERFACE DESIGN

Section 5
IMPLEMENTATION PLAN

Appendix A
USER DATA DICTIONARY

Appendix B
SERVER DATA DICTIONARY

** END OF FORMATTING **
TDP/3000 (A.03.11) HP36578 Formatter
MON, FEB 10, 1986, 6:30 PM
NO ERRORS
INPUT = ID.FTP.DDN
OUTPUT = *HP2680

#J183; #O2621 * ID, DAVID.DDN; SLP * MON, FEB 10, 1986, 6:58 PM
#J183; #O2621 * ID, DAVID.DDN; SLP * MON, FEB 10, 1986, 6:58 PM
#J183; #O2621 * ID, DAVID.DDN; SLP * MON, FEB 10, 1986, 6:58 PM

#J183; #O2621 * ID, DAVID.DDN; SLP * MON, FEB 10, 1986, 6:58 PM
#J183; #O2621 * ID, DAVID.DDN; SLP * MON, FEB 10, 1986, 6:58 PM
#J183; #O2621 * ID, DAVID.DDN; SLP * MON, FEB 10, 1986, 6:58 PM

#J183 #O2621 * ID, DAVID.DDN; SLP * MON, FEB 10, 1986, 6:58 PM
#J183; #O2621 * ID, DAVID.DDN; SLP * MON, FEB 10, 1986, 6:58 PM
#J183; #O2621 * ID, DAVID.DDN; SLP * MON, FEB 10, 1986, 6:58 PM

#J183; #O2621 * ID, DAVID.DDN; SLP * MON, FEB 10, 1986, 6:58 PM
#J183; #O2621 * ID, DAVID.DDN; SLP * MON, FEB 10, 1986, 6:58 PM
#J183; #O2621 * ID, DAVID.DDN; SLP * MON, FEB 10, 1986, 6:58 PM