Appendix-3

Brief Overview of Dedip functional components

1. Introduction .............................................................. 2
2. OPRINT ........................................................................ 2
2.1 VAX/VMS ................................................................. 2
2.2 UNIX ........................................................................ 5
3. Host Manager ............................................................... 8
3.1 VAX/VMS ................................................................. 10
3.2 UNIX ........................................................................ 11
4. Slave Manager ............................................................. 11
5. File Transfer ................................................................. 13
6. Message Passing Library ............................................. 15
1. Introduction

Following sections gives overview of analysis and design of DEDIP components; OPRINT, HostManager, SlaveManager, Server, File Transfer and Message passing library.

2. OPRINT

The structure analysis and design was followed for developing the OPRINT process on VAX/VMS operating system. The ISRO software Quality Assurance Standards (SQAG) was followed for the same [73]. The detailed analysis and design is given in DWPIIP-1. Object modeling Technique (OMT) was used for user interaction component on Unix platform.

2.1 VAX/VMS

The first order hierarchy structure is shown in figure A3.1. This component was developed in C using VAX/VMS SMG (Screen Management Graphic) and VAX/VMS system services. Mailbox along with AST (Asynchronous System Traps) is used for communication with host Manager.

OPRINT:

Main function calling all the functions.

Read Menu Files:

The Dedip GUI is user inputs driven. The menus are configured as per the user inputs. This module reads the user defined menus from various files.

Create Menu:

It creates various menus depending upon the information read from files. It creates the different boxes for main menus and boxes within boxes for sub menus. Two levels of menus (Menu and submenu) are supported for a group of applications. See screen shot 5.1A.
Brief overview of Dedip functional components

Fig 3A.1: First Order Hierarchy diagram for OPRINT

Create Mail boxes:
It assigns the channels to mailboxes to communicate with HostManager. It associates the handle_SCH_msg module with using AST to get message from HostManager. When ever Host Manager send the message using the mailbox, the function handle_SCH_msg will be executed.
Brief overview of Dedip functional components

Delete Mail boxes:

It deletes the mail boxes.

Display keys:

It creates nice looking menu for each key. It displays the control key name and the function associated with the key. See screen shot 7.1A.

Handle user CMD:

It monitors the key board. It changes the focus across the menus and submenus based on arrow key pressed by the user. Whenever user selects the application and presses the pre defined control key, it forms the required message associated with the key. It then informs the message to the Host Manager using the mailbox.

Handle SCH msg:

This modules is executed when ever the Host Manager writes the message to the mailbox. This function decodes the message and calls the required display routine to update the status on the user screen. If any process is terminated abnormally, it displays the error code and message to the user terminal and interacts for necessary action. See screen shot 7.1D.

package status display:

It displays the over all status of all the applications under execution. See screen shot 7.1B.

process status display:

It displays the detailed information about each process of an application. See screen shot 7.1C.
Brief overview of Dedip functional components

2.2 UNIX

The first order class diagram is shown in figure A3.2. This component was developed using C++ and Motif.

Figure A3.2: 1st level class diagram for OPRINT on UNIX.
Brief overview of Dedip functional components

Icon:
It is base class to generate the icon. The Motif 1.1 does not support the icon directly. Hence this class was required.

IconColor:
It decides the various types of icon color.

NewIcon:
It generates the new folder in the currently selected folder. A popup window is attached along with it to interact with user to get the required inputs.

NewApplicationIcon:
It generates a new folder as well as the new application.

GroupIcon:
It displays a GroupWindow for the currently selected folder. A GroupIcon is attached with each folder under the DEDIP environment.

BuildApplicationIcon:
It displays a window to interact with user incase he has built the application externally.

ApplicationIcon:
It is an application which is configured by the designer. On clicking, it will display popup menu that contains different actions will be disabled as per the application status. When user select an action, it sends an appropriate message to HostManager.

MasterScreen:
It displays various windows within it. It is a super container.
Brief overview of Dedip functional components

**MasterGroupWindow:**

It contains NewIcon using which user can create different folders. It points to the root folder of Dedip operational environment.

**GroupWindow:**

It displays NewIcon, NewApplicationIcon and different application icons. There may be many group window in hierarchical order under Master Group Window.

**ActiveApplicationWindow:**

It activates applications from ActiveApplication list and display ActiveApplications for user interaction.

**ActiveApplicationList:**

It contains list of active (running) applications.

**ActiveApplication:**

This icon is attached to each active application. A popup menu is attached to this icon which contain user actions.

**ApplicationProgressWindow:**

It displays the detailed information about each process of each application.

**MessageWindow:**

It displays different messages to the users. It is used by other classes for different purposes.

**FileDisplayWindow:**

It displays the files created by application to the user. This is also used by other classes for different purposes.

**DisplayError:**

It is used to display error message to the user and interacts for appropriate action. It is user by other classes for different purposes.
Brief overview of Dedip functional components

3. Host Manager

SQAG was followed for the structured analysis and designed of these components. The first order hierarchy structure for both the operating systems (VAX/VMS and Unix) remain nearly same. It is shown in figure A3.3.

Fig. A3.3: 1st order Hierarchy structure of HostManager
Brief overview of Dedip functional components

*init global var*:  
It initializes the global variables.

*create global section*:  
It creates global section (shared memory) for inter process communication.

*exit handler*:  
It handles the self-termination mode. It disables the various functions based on the termination type.

*start package*:  
It reads the inter-process dependency information from the application configuration file. It generates multi-branches link list. It starts the application by scheduling the first process. It maintains a counter for invoking an application for multiple times for different data sets.

*handle process termination*:  
On termination of any scheduled process, this module is executed. It informs the OPRINT about the process status. It checks the status of the terminated process and shoots the next process if terminated successfully.

*shoot next process*:  
on successful termination of any process, it finds out the dependent processes from the linked list and schedules all such processes.

*restart package*:  
It kills all the current processes of an application and restart the application from first process.

*abort package*:  
It kills all the current processes of an application.
Brief overview of Dedip functional components

**suspend package:**
It suspends all the current processes of an application.

**resume package:**
It resumes all the suspended processes of an application

**handle remote msg:**
It decodes the message from the SlaveManager and takes the necessary actions. The action may be informing the OPRINT about the latest status of a process, Scheduling next processes, etc.

**inform slave manager:**
This module makes a connection with the remote server and transfers the message to SlaveManager over the network using TCP/IP socket. This module is invoked by many modules as when interface with the SlaveManager is required, for example scheduling/terminating/suspending/resuming a process on remote node. It resolves the issues multiple synchronous connection to the slave server.

**create mailbox:**
It creates mailboxes on VAX/VMS for communication with OPRINT and attaches AST with it.

**load prev session:**
This module is invoked whenever the operator selects the restarting of the previous incomplete session. It loads all the required information from various intermediate files. It starts all the pending processes of all the applications of the previous session one by one based on its status.

3.1 VAX/VMS
This component was developed using C and VAX/VMS system services. Mailbox was used for communication with OPRINT. The system level process termination status was captured using mailbox and AST. Global section was used for user handled error codes.
Brief overview of Dedip functional components

3.2 UNIX

This component was developed using C and Unix system calls. Shared memory and interrupts were used for inter process communication among HostManager, OPRINT and scheduled processes.

4. Slave Manager

SQAG was followed for the structured analysis and designed of these components. The first order hierarchy structure is shown in figure A3.4. This component is developed using C and Unix system calls. This component was very much useful until the HostManager on Unix wasn’t developed.

![Hierarchy structure of Slave Manager](image)

Fig A3.4: Hierarchy structure of Slave Manager.
Brief overview of Dedip functional components

**read path:**

It reads the configuration file and set the DedipArea path as configured by the system administrator.

**shoot server:**

It schedules the server process that waits for message from HostManager. The server passes the message from HostManager to the SlaveManager using shared memory and interrupt.

**shoot process:**

It schedules the process as required by the HostManager. It creates the space in shared memory for inter-process communication with the scheduled process. It passes the process start time to the HostManager.

**terminate process:**

This function is called when the HostManager request to terminate a process. It checks whether the process is currently running. It kills the process, if found running. If the process is suspended then it first resumes it and then terminates the same. It passes the process termination time to the HostManager.

**suspend process:**

This function is called when the HostManager request to suspend a process. It checks whether the process is currently running. It suspends the process, if found running. It passes the status to the HostManager.

**resume process:**

This function is called when the HostManager request to resume a process. It checks whether the process is suspended. It resumes the process, if found suspended. It passes the status to the HostManager.

**inform scheduler:**

This function makes the connection with the server running on the host using socket and passes the message to the HostManager. It makes three attempts at predefined time interval incase of failure. It resolves the issues multiple synchronous connection to the host server.

A3.12
5. **File Transfer**

SQAG was followed for the structured analysis and designed of these components. This component has multiple flavors. It has interfaces with FTP and RCP using command procedures on VAX/VMS for pushing and pulling data from remote machine. The drivers programs were developed in C for creating the command procedures dynamically based on the user requirement. The drivers were capable of scheduling and monitoring the command procedures.

A client server application is developed using C and socket library. The first order hierarchy structure is shown in figure A3.5 and A3.6.

**Fig A3.5: Hierarchy chart of File Sender**

- **env set for send:**
  - It opens a server socket using TCP/IP.

- **send new port:**
  - When client requests connection, it reads the new port from the configuration file. It forks new process that opens a new socket using C.
Brief overview of Dedip functional components

the port for communication with client receiving the data. It sends the new port to the main connection and closes the main connection.

**transfer file:**

It communicates with the client to decide the list of files to be transferred to the remote node. It calls either raw_file_transf or ascii_file_transfer for each of the file requested.

**raw file transfer:**

It opens the user-demanded file using raw mode. It reads the file in chunks of bytes and sends the packets to the remote node. On completion, it closes the file.

**ascii file transfer:**

It opens the user-demanded file using ascii mode. It reads the file line by line and sends the packets to the remote node. On completion, it closes the file.

![Hierarchy chart of File Receiver](image)

Fig A3.6: Hierarchy chart of File Receiver
Brief overview of Dedip functional components

**env set for rec:**

It makes connection to server using TCP/IP.

**rec new port:**

It reads the new port from server. It then closes the main connection and makes the connection with server using the new port.

**receive file:**

It communicates with the server to decide list of files to be transferred from the remote node. It calls either raw_file_rec or ascii_file_rec for each of the file. It preserves intermediate status that helps in retransmission in case the connection is reset due to some reason.

**raw_file_rec:**

It opens the new file using raw mode. It reads the data packets from server and writes to the file. It preserves intermediate status that helps for retransmission if connection is reset due to some reason at minimum loss of data transfer. On completion, it closes the file.

**ascii_file_rec:**

It opens the new file using ascii mode. It reads the data line by line from server and writes to the file. It inserts the line termination code based on the operating system requirement. It preserves intermediate status that helps for retransmission if connection is resent due to some reason at minimum loss of data transfer. On completion, it closes the file.

6. Message Passing Library

Different routines were developed at different time slots based on the requirements. No systematic analysis and design procedures were followed for this library due to smaller size of each routine. SWAQ standards were followed for coding. Few important routines are listed below.
Brief overview of Dedip functional components

**Allocate-resource:**

It allocates a resource for an application.

**Release-resource:**

It releases the allocated resource to other application

**Reserve-resource:**

It reserves the resource for an application that needs the resource at later (known) time. It routine right now is implemented only in simple mode that is very similar to that of allocate-resource. However aim of designing this routine is to utilize the resources optimally.

**Resource-status:**

It returns the current status of a resource.

**Post-error:**

This module passes the used handled error code to the slave manager / hostmanager / agent.

**Create xfer file:**

It creates a empty file that contains file transfer information required by a process.

**Append xfer file:**

It appends a record contain file transfer information into the corresponding file.

**File-open:**

This routine is very much useful for applications that support the simultaneous execution on multiple data set. The application designer should use this function rather than using directly supported by the language for temporary files. It creates / opens a file job identification number (counter + process number). It discriminates the temporary file creatd for different data sets.