northern telecom

Problem Determination Tools

Author(s):	Mikhail Khodosh
Manager:	Rod Bagg
Dept:	4Q22

Date:	July 23, 1991
Issue:	3.0
Project Name:	THOR
File Name:	pdt.3.0.ud
Activity Id:	DE0792
Release:	19

Keywords:	PDT, THOR, Debugger, Problem Determination, RA, HLA, HLD

Abstract:	This document describes the THOR problem determination strategy.

Revision History

ISSUE NO.	DATE	AUTHOR	REASON FOR ISSUE
0.1	11/12/90	M. Khodosh	RA initial release
1.0	12/21/90	M. Khodosh	Issue for review
1.1	01/20/91	M.Khodosh	Result from review
2.0	01/29/91	M.Khodosh	HLA added
2.2	06/03/91	M.Khodosh	Result from HLA review

References

(1) Debug Tool Proposal, 10/21/88, Karl Bernhardt, Bob Asdel, Robert Stagmier, NT MTV

(2) Field Support Overlay Debug, User Manual, January/89, J. Pancevich

(3) VxWorks Programmer's Guide, Volume 1, Wind River Systems, Inc.

(4) THOR High Level Debugger, User's Guide, January/91, Charlene Yu, Dept. 4Q22, NT MTV

(5) vxshell source code, Michael Thompson, Dept. 4Q21, NT MTV

(6) rlogind source code, Michael Thompson, Dept. 4Q21, NT MTV

(7) Pseudo tty driver, Software Design Document, Andy Phan, Dept. 4Q25, NT MTV

(8) SL1 Remote Function Call, Feature Specification, Denny Landaveri, Dept. 4Q21, NT MTV

(9) C / SL1 Interface, Programmer's Reference, Allen Aubuchon, Dept. 4Q22, NT MTV

1 Requirements Analysis (RA)

Problem Determination Tools (PDT) are intended to locate, examine and eliminate problems (malfunctions) in the THOR system. They can be used both locally and remotely at development time and run time. PDT also can be used for profiling and statistics collection to analyze and improve system characteristics.

1.1 Commercial Requirements

PDT should:

- provide current resident debugger functionality;
- support problem determination and fixing for all processes of THOR operation (setting operating environment, SYSLOAD, INITIALIZE, call processing, overlays);
- provide for remote operation the same functionality as for local operation;
- ensure different levels of authorized access (access restriction mechanism).

1.2 System Requirements

The primary function of PDT is to support THOR debugging and problem fixing. The rest of the functions, described below, can be considered as auxiliary functions, which are intended to provide services to the primary one. Some of them are quite independent and can be viewed as a separate tools. They are:

- remote operation support;
- macro capability;
- line editor;
- disk utilities;
- file transfer facilities.

Together with the debugging facilities these tools will constitute Problem Determination Toolkit.

1.2.1 Access to the VxWorks

VxWorks, a powerful development environment, has a lot of facilities which can be helpful in the problem determination process. It would be a good idea to provide PDT users with an access to all the VxWorks features. However, it is important to understand that the power of the feature makes it potentially dangerous. That is why the PDT should split all the VxWorks features into several groups and provide different levels of authorized access to these groups (similarly to the rest of the PDT facilities).

1.2.2 Debugging

As a regular debugger PDT are to provide facilities to examine and change THOR programs and data, and to manage THOR operating environment. For these purposes PDT will implement several levels of user interface:

- call processing object level (Call Registers, TN, DN, ASD queues etc.);
- symbolic level (only for global symbols);
- assembler level;
- hardware level.

PDT should include most of the current resident debugger functionality plus new facilities:

- SYSLOAD and INITIALIZE processes debugging;
- conditional breakpoints;
- extended macro capability;
- disassembler;
- one-line assembler;
- diagnostic patching;
- global symbol table maintaining;
- debugging session logging and backtracking;
- patch retention for emergency fixing.

1.2.3 Remote Operation

The following remote capabilities must be provided as part of PDT operation:

- SYSLOAD and INITIALIZE;
- faceplate Indicators (LCD, Hex display, LEDs) read;
- initiating peripheral software (PSDL).

1.2.4 Macro Capability

Macro here is a sequence of debugging commands which can be executed as one command at any break point, as part of the debugger starting procedure and as part of the debugger ending procedure. Macro capability must include:

- macro create and edit facility;
- macro library support facility;
- macro binding facility.

1.2.5 Line Editor

A simple line text editor should be provided to custom macro creation, emergency patching, etc.

1.2.6 Disk utilities

A set of disk utilities have to be provided as a part of PDT. They will include the following features:

- disk maintenance (FORMAT, INFO, LABEL, etc.);
- directory maintenance (MKDIR, CHDIR, RMDIR, etc.);
- file manipulation (COPY, DELETE, TYPE, etc.).

1.2.7 File Transfer and Software Distribution

PDT remote operation requires file transferring and manipulating activity. File transfer should be provided for globals, patches, overlays, customer database, etc.

File transferring together with the PDT remote operation extend significantly PDT capabilities and can also be used as a base to create Software Distribution System.

1.2.8 User Interface

Because the PDT will be invoked through VxWorks shell and will utilize some of its debugging facilities, they should preserve their interaction style and syntax.

1.3 Performance Requirements

Special provisions should be made to fit the following requirements:

- PDT should not degrade SL-1 performance;
- PDT must work on heavily loaded system.

1.4 Implementation Considerations

The PDT will be based on current resident debugger programs, which were written in SL-1, and on VxWorks shell debugging facilities, which have “C-oriented” program interface. Special provisions should be made to use them together.

The new PDT modules are going to be written in C or SL-1, depending on what is more efficient in each particular case.

The front-end routines should be isolated enough to be replaced when porting to a different operating environment.

1.5 PDT Questionnaire

The list of functions indicated below have been submitted for reviewing to the ETAS group in Richardson and to the FS group in Mountain View. They were asked to indicate priority (High, Medium or Low) for provided functionality. There was five people from Richardson and eight people from Mountain View who had answered this questionnaire. Their answers and summarized score for each function are shown here.

PROBLEM DETERMINATION TOOLS QUESTIONNAIRE

FUNCTION	ETAS						FS									Σ
						Σ									Σ	
Diagnostic Patching	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H	H
Resdb functionality	H	H	H	H	H	H	H	H	H	H	H	H	H	M	H	H
Save Trap Data Block on init	H	H	H	H	H	H	H	H	H	H	H	H	H	M	H	H
Conditional breakpoints	H	H	H	H	H	H	H	H	L	M	H	M	H	H	H	H
Global procedure replacement	H	H	H	H	H	H	H	H	H	M	M	H	H	L	H	H
Print formatted Trap Data Block	H	H	H	H	H	H	M	L	H	H	M	M	H	H	M	H
Print Call Regs for ACTIVE CR & all keys & AUX CR	H	H	-	M	M	H	H	H	H	H	M	H	H	H	H	H
Global symbol table access	H	H	H	H	H	H	M	H	M	H	M	M	M	L	M	H
Secondary memory freeze	H	H	-	H	H	H	M	H	H	H	H	H	M	L	H	H
Macro capability	H	H	H	H	H	H	H	-	L	H	M	H	H	L	M	H
Patch retention for emergency patching	H	M	-	M	M	M	L	H	H	H	M	H	H	M	H	H
Print CR info for specific key on TN	H	H	-	M	M	H	M	H	H	M	M	H	H	H	H	H
Print ACD queues	H	H	H	H	M	H	M	M	M	M	M	M	M	M	M	M
Emergency Patching (680x0 opcode level)	H	M	H	M	M	M	L	-	H	M	H	H	H	M	M	M
Memory dump analysis tools	H	H	H	H	H	H	M	M	H	L	L	M	M	L	M	M
Set memory reference (prim/sec)	H	H	-	H	M	H	L	H	L	M	L	H	M	M	M	M
Memory dump to disk file	H	H	-	H	H	H	M	-	H	L	M	M	M	M	M	M
Debug Init	H	-	-	L	L	L	H	M	M	L	M	H	M	H	M	M
Print TTY port info	M	M	-	M	M	M	M	L	M	M	L	M	L	M	M	M
Database upload	L	-	-	M	M	M	M	M	L	L	L	M	L	M	M	M
Remote Init	L	L	-	L	L	L	M	L	L	M	L	L	H	H	M	L
Remote faceplate indicators manipulation	L	M	-	M	M	M	L	L	M	L	L	M	L	H	L	L
Debug Sysload	L	-	-	L	L	L	M	M	M	L	M	H	L	M	M	L
Remote switchover	H	-	-	L	L	L	M	M	L	L	L	H	L	M	M	L
Sysload patching	L	-	-	L	L	L	M	L	M	H	L	H	L	M	M	L
Remote Sysload	L	L	-	L	L	L	L	L	L	L	L	L	M	M	L	L

2 Project Description (HLA)

2.1 Functional Overview

The PDT, when activated, create a special environment, which allows to monitor any process in THOR operation. Conceptually, it can be viewed as some kind of a shell.

The PDT functions can be classified as follows:

- managing THOR operating environment;
- Basic debugging facilities;
- Extended debugging facilities;
- service (auxiliary) functions and utilities.

In accordance with this classification the PDT will be viewed as if they consist of four subsystems: Managing Operating Environment, Basic Debugging Level functions, Extended Debugging Level functions and Services.

2.1.1 Managing Operating Environment

This group will include the following functions:

- faceplate indicators read, write and clear;
- simulate Manual Reset for reload system;
- simulate Manual Interrupt for reinitialize call processing;
- check and change state of some of the hardware components.

2.1.2 Basic Debugging Level

This group will consist of the following subgroups:

- VxWorks shell facility;
- current RESDB facility, which is not call processing object oriented and not covered by the VxWorks shell;
- one-line symbolic assembler / disassembler.

2.1.3 Extended Debugging Level

This group can be divided into the following subgroups:

- call processing object level debugging;
- SYSLOAD and INITIALIZE debugging;
- conditional breakpoints and macro binding capability;
- diagnostic patching retention;
- debugging session logging and backtracking.

2.1.4 Services

This group will provide auxiliary functions to serve some other facilities from the groups above. It will also include several utilities which will also provide PDT related services. The functional subgroups of this group are listed below:

- line text editor;
- macro library support facility;
- global symbol table maintaining;
- customer database upload / download;
- software distribution support facility;
- disk utilities.

2.2 Normal Operation

2.2.1 Start-up

PDT should be spawned by the Root task prior SYSLOAD is spawned and immediately activated if Start / Restart debugging is required. During their initialization PDT should determine which serial port on CPU card will be used as the operator console. After the starting PDT will perform regular command interpreter functions and control all input from the operator console or via rlogin.

2.2.2 Configuration file

PDT start-up process can be almost completely determined by the special PDT configuration file. Among other things, configuration file allows to

- specify a serial port to be used as console;
- assign values to environment variables;
- specify names for macros to be bound with predefined breakpoints.

2.2.3 Predefined breakpoints

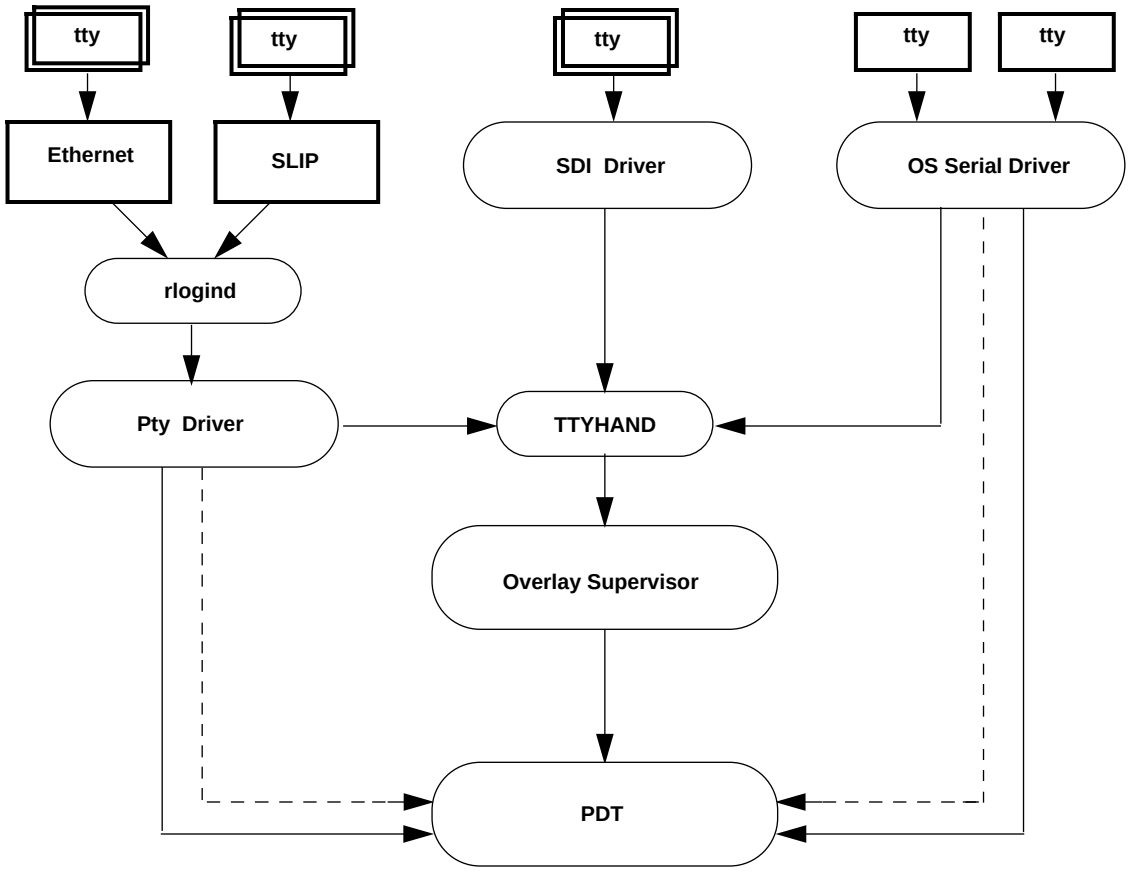
There are three predefined breakpoints known to PDT: at the SYSLOAD start point, INITIALIZE start point and Call Processing start point. These breakpoints are conventional means for the PDT to get control for performing appropriate actions before a proper process is started. PDT behavior at such breakpoints is defined by the macros, bound with them. Names of these macros are predefined (or defined in the configuration file).

One of the most evident uses for this breakpoints is for diagnostic patching purposes or for initiating the patch retention tool.

2.2.4 Command Input

Before SL-1 is started, PDT will use the serial port, chosen during initialization for communication purposes. If necessary, PDT will provide for switching to the other CPU card serial port with a special command.

After SL-1 is started, PDT can accept commands from the network serial ports (through overlay supervisor) as well.



PDT Input

2.2.5 Security Levels

The primary purpose of the security mechanism providing by PDT is information integrity and preventing denial of service. It will be achieved by providing several levels of controlled access to PDT functionality. The implementation model is based on assumption that each PDT command or

command option have fixed security attribute which can be are used to determine whether a user can access this function.

There will be four different levels of access:

Informational level.	Only information retrieval commands are allowed.
Debugging level.	Regular PDT level. Most of PDT commands can be used on this level.
System level.	This level includes commands which cause changes that would normally not be recoverable even with system restart.
Super user level.	PDT administrative level. This level allows to change security attributes, passwords, reset or restart PDT and so no.

2.3 Design Approach

As it can be seen PDT should include several different groups of functions. For the designing purposes, it would be convenient to split PDT into several independent subsystems, which will be integrated by the common and consistent user interface.

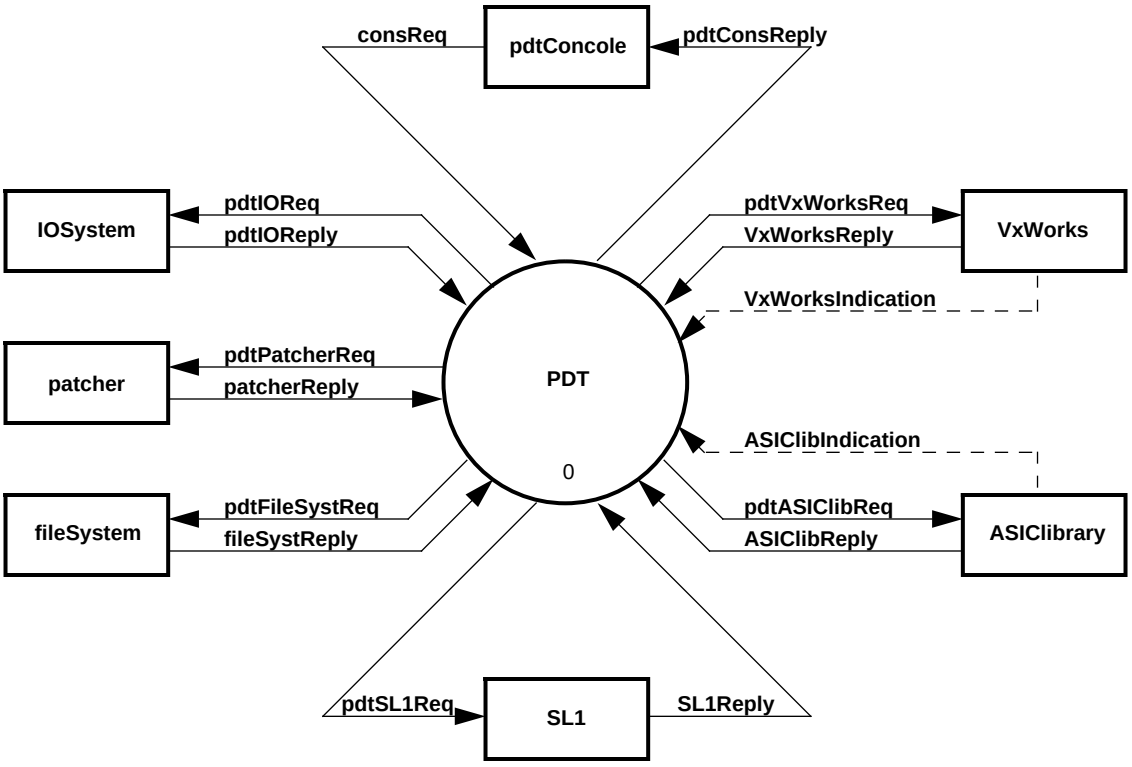
Each subsystem can be designed separately and independently of the others and will have its own implementation layers.

Nevertheless, it would be a good idea to have a common lowest-level layer for all of the PDT subsystems (PDT kernel). It will make possible to generalize functions and data structures, common for several subsystems.

3 Interfaces (HLA)

3.1 Context Diagrams

Context-Diagram
Problem Determination Tools



3.2 Description of Elements of Diagram

3.2.1 PDT Console

pdtConsole - terminator represents a logical device which will be used to interact with PDT. This is composite device, which is created by PDT, using standard input and standard output files.

consReq - data flow represents PDT commands or additional data, which can be requested by PDT to complete execution of the processing command.

pdtConsReply - data flow represents any data provided by PDT as a response to the entered request.

3.2.2 Input-Output System

IOSystem - represents input-output facilities, which are needed for the PDT operation. These facilities can be provided by IOP, SDI or Network software.

3.2.3 Patcher

patcher - represents a THOR Patcher program, which will be used for patch retentions.

3.2.4 File System

fileSystem - represents THOR file system, which will provide basic features for disk utilities operation, macro library support, and customer database upload / download.

3.2.5 ASIC library

ASIClibrary - terminator represents a library of services provided for the Application Specific Integrated Circuits (CMB, SRA, BIC).

3.2.6 SL-1 Software

SL1 - terminator represents call processing and related software (SYSLOAD, INITIALIZE, overlays) which needs to be debugged or monitored by the PDT**.**

3.2.7 VxWorks Operating System

VxWorks - terminator represents services provided by the VxWorks Operating System.

3.3 Man Machine Interface

PDT are command driven. They will preserve the VxWorks interaction style and syntax. As a temporary solution PDT can still allow to use an old-fashioned mnemonics and syntax for the existing RESDB commands to make a transition to a new interface easier.

3.3.1 Managing Operating Environment

Mnemonic		Description
QENV	[	show all environment variables or
	SECUR |	current authority level
	BRKPT |	current breakpoint number
	LOG |	current log file name
	MREF |	current memory reference
	MLIB |	current macro library name
	MECHO |	current macro echoing setting

	TTY]	current TTY port and USER list
SETENV		change value of an environment variable
	SECUR |	security level
	MREF |	memory reference
	LOG |	log file name
	MLIB |	macro library name
	MECHO |	macro echoing setting
	TTY	TTY port
MEM	[OS | CP | SM]	display memory map
CON	[PORTA | PORTB]	switch console to the required device
FIClear	LCD | LED | HXD	clear faceplate indicator
FIRead	LCD | LED | HXD	read faceplate indicator
FIWrite	LCD | LED | HXD <string>	write string on faceplate indicator

3.3.2 Basic Debugging Level

3.3.2.1 Breakpoints, Stepping and Continuing

Mnemonic		Description
BClear	<brkptno> | ALL	clear breakpoint(s)
BDisable	<brkptno> | ALL	disable break(s)
BEnable	<brkptno> | ALL	enable break(s)
BList	<brkptno> | ALL	list breakpoint(s) and its (their) attributes
BRet		return from break mode (continue)
BSet	<adr>	set breakpoint
SInstr		step (execute one machine instruction)
SOver		step over a subroutine

3.3.2.2 Event Monitoring

Mnemonic		Description
Event	READ [<adr1> [-< adr2>]]	define event to be monitored
	WRITE [<adr1> [-< adr2>]]	
	EXEC [<adr1> [-< adr2>]]	
EDisable	<eventno> | ALL	disable events(s)
EEnable	<eventno>| ALL [STOP | NSTOP]	enable event(s) with stop / nonstop attribute setting

EList	<eventno> | ALL	list event(s) and its (their) attributes
ERet		return from break mode (continue)
SInstr		step (execute one machine instruction)
SOver		step over a subroutine
CPEvent	<etype> [<adr1> [- adr2>]]	define CP event to be monitored

3.3.2.3 Display and Modify Memory

Mnemonic		Description
Display	[<adr> [<length>]]	display memory
Modify	<adr>	modify memory

3.3.2.4 Assemble, Unassemble

Mnemonic		Description
Assemble	<adr>	enter assembler mode
Unassemble	[<adr> [n]]	display raw program in assembler format

3.3.3 Call Processing Related Commands

3.3.3.1 Call Processing Loading and Initializing

Mnemonic		Description
CPLOAD	[DEBUG]	start CP SYSLOAD
CPINIT	[DEBUG]	start CP INITIALIZE

3.3.3.2 Call Processing Break, Stepping, Continuing

Mnemonic		Description
CPBRK	[NOW]	enter manual break mode immediately or
	[WHEN] <*event*>	set eventual breakpoint
CPSTEP		Call Processing step
CPGO		continue (return from CP break mode)

3.3.3.3 Call Processing Query

Mnemonic			Description
CPTN	<*tn*>	show	*tn* alternate form
	<*tn*> CRG		*tn* call register information
	<*tn*> TRM		*tn* transmission
	<*tn*> KEY <*key#*>		*tn* call register for *key#*
CPCST	<*cust. no.*> AQ	show	customer ACD queues
	<*cust. no.*> DN		customer DN translation

	<*cust. no.*> DP		customer data pointers
	<*cust. no.*> RP		customer route pointers
CPDSP	CODE <glblno.><offs>	display	program memory
	DATA		data memory
	MNS		main stack
	RAS		return address stack
	INI		trap information
CPNWK	<loop no.> [time slot]	display network information	

3.3.4 Macro Facility

Mnemonic		Description
MACRO	<*macro name*>	macro definition
MBIND	<*brkptno*> [<*macro name*> | MACRO]	macro binding
MECHO	[ON | OFF]	echo commands from executing macro
MIF	<*condition*>	macro conditional statement
MLIB	[<*macro library*>]	define macro library
MSYNX	<*macro name*>	check macro syntax
MEND		end of macro
MRUN	<*macro name*>	run specified macro

3.3.5 Services and Utilities

3.3.5.1 PDT Session Logging

Mnemonic		Description
LOG	[STATUS] |	logging status
	OPEN |	open log file
	CLOSE |	close log file
	ENABLE |	activate logging
	DISABLE |	deactivate logging
	FLASH]	write log buffer to the file

3.3.5.2 Disk Maintenance

Mnemonic		Description
FORMAT	<disk>	format disk space

CHKDSK	<disk>	show disk information
LABEL	<disk>	change volume label
VOL	<disk>	show volume label
BACKUP	<disk>	disk backup
RESTORE	<disk>	disk restore
DISKCOPY	<disk1> <disk2>	copy disk
DISKCOMP	<disk1> <disk2>	compare disks

3.3.5.3 Directory Maintenance

Mnemonic		Description
CHDIR	<path>	change current directory
MKDIR	<path>	create new directory
PDIR	<path>	protect (lock) directory
UDIR	<path>	unlock directory
RMDIR	<path>	remove directory
TREE	<path>	show directory tree

3.3.5.4 File Manipulation

Mnemonic		Description
COPY	<file> <file>	copy file
COMP	<file> <file>	compare files
DIR	[<path>]	list files
DELETE	<file>	delete file
MOVE	<file> <file>	move file
TYPE	<file>	type file

3.3.5.5 Miscellaneous

Mnemonic		Description
DATE	[<date>]	set current date
TIME	[<time>]	set current time
EDIT	[<filename>]	edit text file
HELP	[<command> | <topic>]	display Help information

3.4 Exported Interfaces

As a rule all PDT functions will have conventional procedure interface and can be called as an existing VxWorks routines, except if access to some of them will be restricted.

4 Global Interactions

4.1 Project Interactions

4.1.1 SL-1 Interaction

PDT will interact with SL-1 when they need to perform SYSLOAD, INITIALIZE or PSDL. Interaction with SL-1 will be required when conditional breakpoints are using and probably also for some of the object level debugging operation.

4.1.2 VxWorks Interaction

PDT will interact with SL-1 when they perform console read and write operation, when they use VxWorks debugging and task management facility, when they perform file transferring operation, and when they use or maintain system symbol table.

4.1.3 Input-Output System Interaction

PDT will interact with the input-output system to perform operations which involve I/O software and hardware.

4.1.4 File System Interaction

PDT will interact with the file system to perform all disk utility functions, to upload and download customer database, to maintain macro library and emergency patching files.

4.1.5 ASIC Library Interaction

PDT will use ASIC library functions for the following purposes:

- to read, write or clear faceplate indicators:
- to simulate Manual Reset and Interrupt events;
- to use hardware breakpoints facilities.

PDT will also use ASIC library functions if, for some reason, higher level software does not operate properly or does not operate at all.

4.1.6 Patcher Interaction

PDT will interact with Patcher to initiate patch retention process.

4.2 Field Support

To be able to use all PDT features support organizations have to:

- train their staff with the PDT;
- train their staff with the 68030 architecture and Assembler language;
- have SUN workstations;

- have modems with reasonable speed.

5 High-Level Design (HLD)

5.1 Design Approach

As mentioned above, PDT should include several almost independent groups of functions. For the design purposes, it would be convenient to split PDT into several subsystems which can be classified as follows:

Functional subsystems. Each functional subsystem implements one major group of PDT functions. The most important functional subsystems are:

pdtSL1Server - implements call processing related functions. As a rule, all related programs residing in SL-1, are called using RFC facilities and are executed during the SL-1 provided time slice.

pdtDbgServer - implements basic debugging facilities. It includes breakpoint functions, event monitoring functions, Display and Modify memory functions, Assemble and Unassemble functions.

pdtDiskUtilities-implements disk and file manipulation functions.

Shell subsystem. Implements user interface and serves as a main control program coordinating functional subsystems. Because PDT is a multiuser system, there can be several shell instances at a time.

Auxiliary subsystems. Auxiliary subsystems provide services for the rest of subsystems. Although all these subsystems are conceptually viewed as independent, sometimes it is more efficient to implement some of their features as part of other subsystems. Thus, some macro facilities should be parts of *Shell* or *Debugger*.

pdtShClassCtl - provides shell control facilities.

pdtStart - provides start-up and restart functions.

pdtLogin - provides login/logout facilities.

pdtMacro - implements PDT macro facilities.

pdtEnv - provides PDT common data structures and related functions.

5.1.1 Assumptions

1. The existing VxWorks rlogin support facility must be extended to provide multiple shell support in the same way as it was done by Michael Thompson in [6].

5.1.2 Dependencies

1. The VxWorks debugging facilities must be defined in the VxWorks configuration file *configAll.h*.

5.2 Flow Diagram

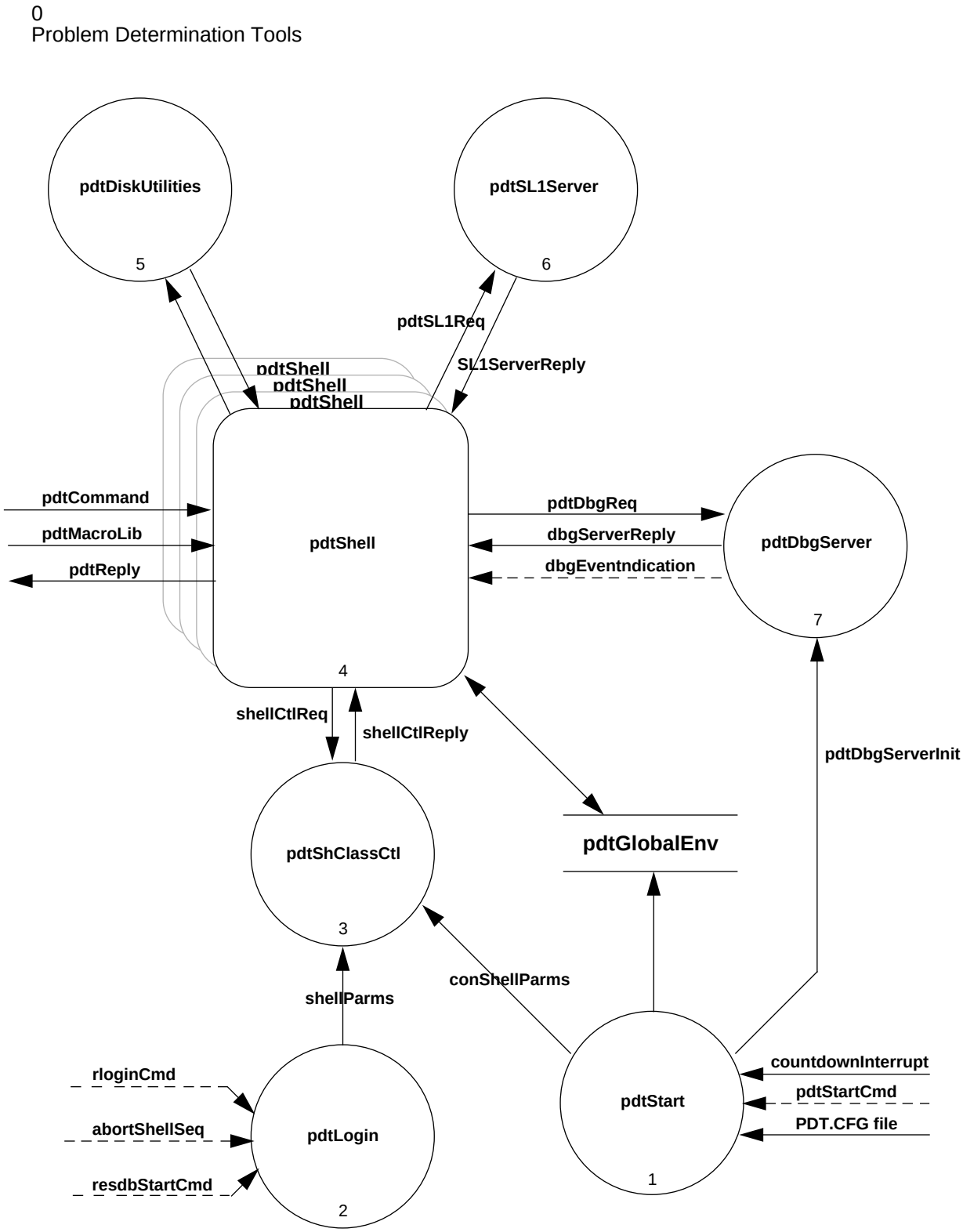


Figure 1. PDT - Flow Diagram

5.3 Common Data Stores

5.3.1 pdtGlobalEnv

5.3.1.1 Data Entities and Interfaces

```
/* pdtGlobalEnv.h */
/* Declarations for the PDT global data */

#define   INIT_START        1       /* initial (booting) start */
#define   COLD_START        2       /* cold restart */
#define   WARM_START        3       /* warm restart */

#define   PRIME_MEM         1       /* primary memory reference*/
#define   SECND_MEM         2       /* secondary memory reference*/

#define   DEFAULT_MLIB      /pdt/maclib
#define   DEFAULT_LOG       /pdt/PDT.LOG

struct pdtGlobalEnv
    {
    int         pdtGblStartType;        /* type of PDT start / restart */
    int         pdtGblMemRef;           /* current memory reference  */
    FNAME       pdtGblLogFname;         /* PDT log file name */
    FD          pdtGblLogFd;            /* PDT log file Fd */
    PATH        pdtGblMlibPath;         /* PDT maclib directory path */
    };

VOID        pdtStartTypeSet();
int         pdtStartTypeGet();
VOID        pdtMemRefSet();
int         pdtMemRefGet();
int         pdtLogFd();
char        *pdtLogName();
char        *pdtMlibPath();
```

5.4 pdtStart Subsystem

5.4.1 Data Entities

shellParms - Input fd, Output fd, Error fd and security level value.

5.4.2 Processes

5.4.2.1 pdtStart

This routine is used to perform initial PDT startup and to restart PDT when a RESTART command is issued or a special watchdog interrupt event occurs.

int pdtStart (startType)

startType = {INIT, COLD, WARM};

INPUT CONTROL FLOWS:

pdtStartCmd : functionCall

OUTPUT CONTROL FLOWS:

pdtShClassInit : functionCall

pdtShCreate : functionCall

pdtSL1ServerInit : functionCall

INPUT DATA FLOWS:

countDownInterrupt : countDownSequence

PDT.CFG file : file

startType : pdtStartTypes

OUTPUT DATA FLOWS:

pdtGblEnv : structure

DESCRIPTION:

```
switch (startType)
    {
    case WARM_START:
        break;
    case INIT_START:
        if <pdtCountDowned>   {<suppress SL1 start>};
        pdtSL1ServerStart();                    /* initialize SL1 server */
        pdtShClassInit(maxShells);              /* initialize shell Class */
        startRlogindTask();
        startDbgServer();
```

```
                startSL!Server();
                readPdtConfigFile();
                setPdtAbortShellFunc();
                startFirstShell();
                break;
        case COLD_START:
                stopRlogindTask();
                stopAllShellTasks();
                stopDbgServer();
                stopSL1Server();
                readPdtConfigFile();
                setPdtAbortShellFunc();
                startRllogind();
                startFirstShell();
                break;
        }
END
```

5.4.3 Issues for the pdtStart Subsystem

1. What are the differences between INIT, COLD and WARM start of PDT? What if environment variables were changed an it is necessary to restart PDT (should we use actual values original PDT.CFG file?
2. Do we need a REBOOT PDT command to initiate OS reboot (manual Reset simulation)?
3. Do we need a CPRESTART (CPINIT) command to initiate call processing restart (manual Interrupt simulation)?
4. Is SYSLOAD going to be separate task? Do we still need a CPLOAD command?
5. Countdown interrupt? Is it possible to suppress SL1 starting during OS reboot?
6. Watchdog interrupt? How it will affect PDT?

5.5 pdtLogin Subsystem

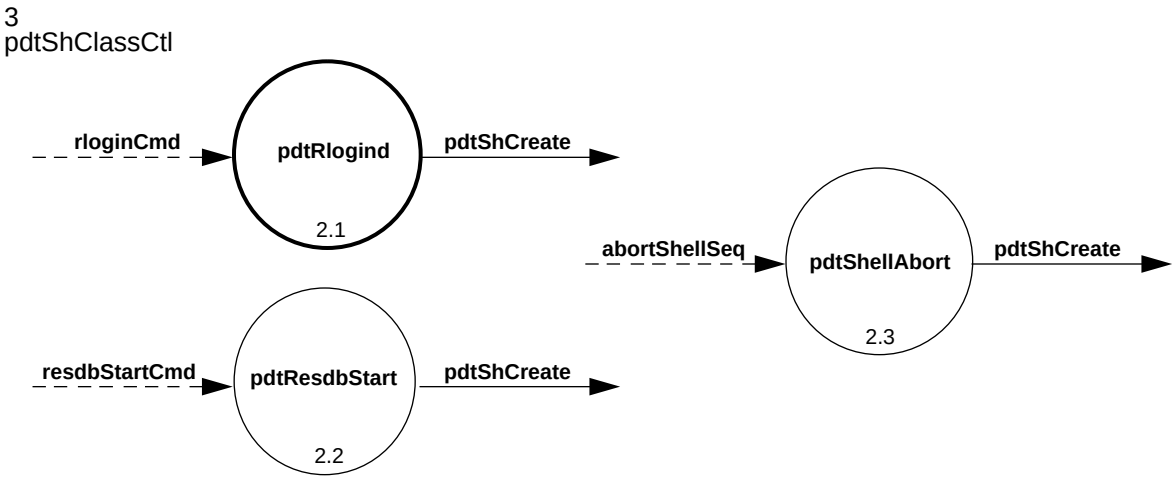


Figure 2. PDT - pdtLogin Subsystem

5.5.1 Data Entities

rloginCmd - UNIX rlogin command.

resdbStartCmd - '$$' - former RESDB start sequence.

abortShellSeq - '^C' - character sequence, which causes shell abort and restart.

shellParms - Input fd, Output fd, Error fd and security level value.

5.5.2 Processes

5.5.2.1 pdtRlogind

This routine will run as an independent process and must replace the existing VxWorks rlogin support. The most important new features are multiple shell support facilities. This routine seems to be very similar to the rlogin daemon "rlogind" written by Michael Thompson [6].

5.5.2.2 pdtResdbStart

This routine will be called by the pdtSL1Server, when '$$' character sequence is entered from any SL1 terminal. In its turn, it will call a pdtShCreate function to create shell with this terminal as a communication device.

5.5.2.3 pdtShellAbort

This routine will be called by the device driver when the shell abort character sequence (^C) is entered. In its turn, it will call a pdtShCreate

function to create new shell for the corresponding device. If any shell has been created for this device, it will be aborted before new shell starts.

5.5.3 Issues for the pdtLogin Subsystem

1. Is it necessary to have a meaningful user ID in rlogin command?
2. Is it necessary to ask for a password in rlogin command?
3. How and when should passwords for different security levels be set and changed?
4. How can the SL1 server get ty fd and provide them to PDT?
5. How can pdtShellAbort function get fd for the corresponding device?

5.6 pdtShClassCtl Subsystem

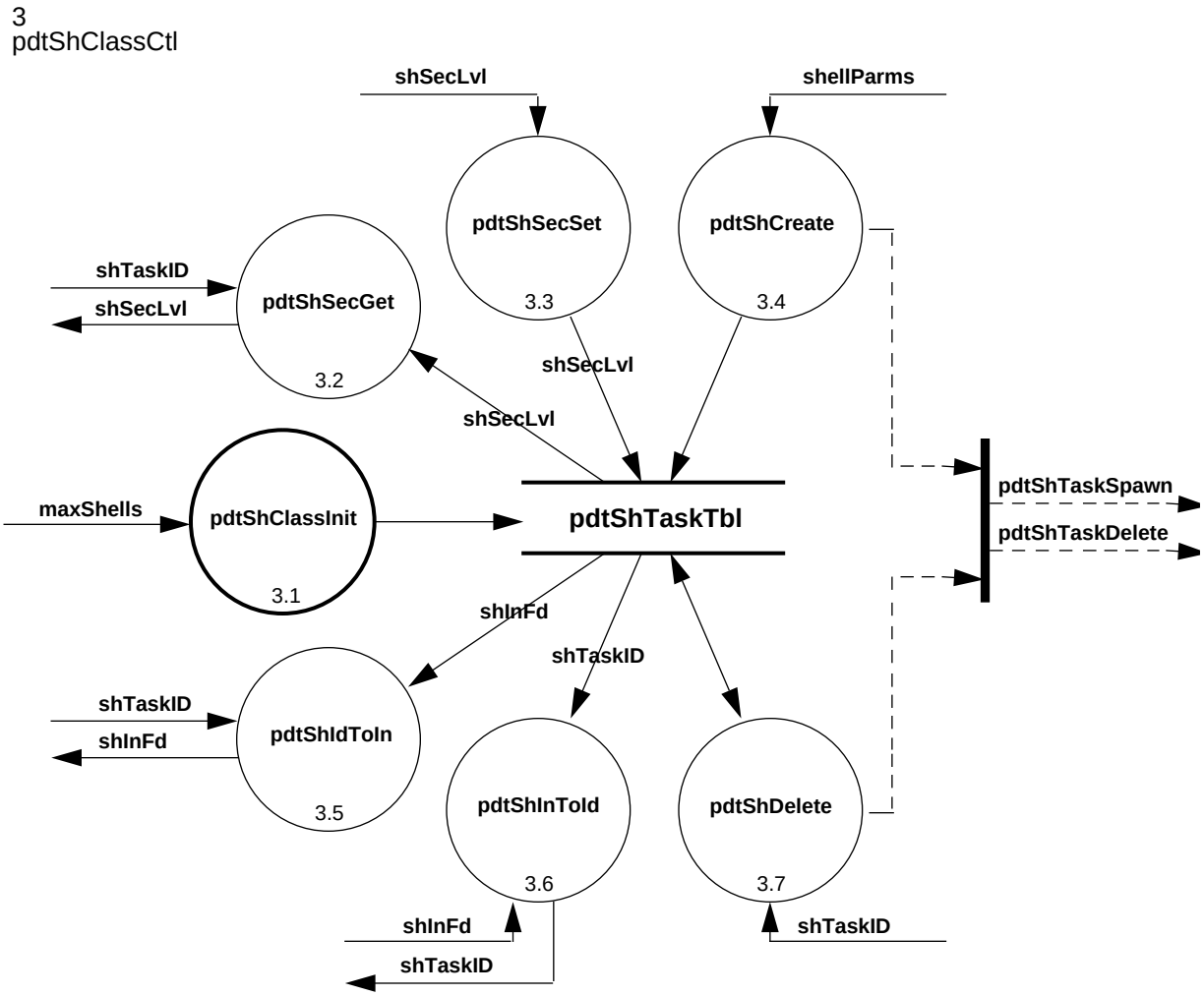


Figure 3. PDT - pdtShClassCtl Subsystem

5.6.1 pdtShell Object Class

At this level it is convinient to build an object class with the pdtShell as a class member. The following is this object class description:

CLASS pdtShell

{

PRIVATE:

struct pdtShTask

{

```
        int pdtShTaskID;            /* shell task ID                    */
        int pdtShInFd;              /* shell task Standard Input Fd     */
        int pdtShOutFd;             /* shell task Standard Output Fd    */
        int pdtShErrFd;             /* shell task Standard Error Fd     */
        int pdtShSecLvl;            /* shell current security level     */
        };
int PDTSH_TBL_LEN       10      /* shell table length  !!magic          */
int PDTSH_MAX           4       /* max number of shells !!magic         */
structure
    pdtShTbl    pdtShTask [PDTSH_TBL_LEN]    /* shell task table */

PUBLIC:
pdtShClassInit (maxShells);                 // class initialization function
pdtShCreate (inFd, outFd, errFd, secLvl);   // constructor function
pdtShDelete (shTaskID);                     // destructor function
pdtShSecSet (shTaskID, shSecLvl);           // set security level function
pdtShSecGet (shTaskID);                     // get security level function
pdtShIdToIn (shTaskID);                     // get shell input fd function
pdtShInToID (shInFd);                       // get input fd for the shell
}
```

Notes.

1. PDTSH_MAX defines the maximum number of shells by default. This value can be redefined in the PDT.CFG file and must be used in the pdtShClassInit function call.
2. Security level 0 will cause VxWorks shell spawning.

5.7 pdtShell Subsystem

PDT will allow multiple shells running simultaneously. These shells are running almost independently and the only thing to care about is the mutual exclusion in using some of the debugging facilities. The mutual shell arbitration will be based on the shell security level attribute. The shell routines code seems to be very similar to the "vxshell" code written by Michael Thompson [5].

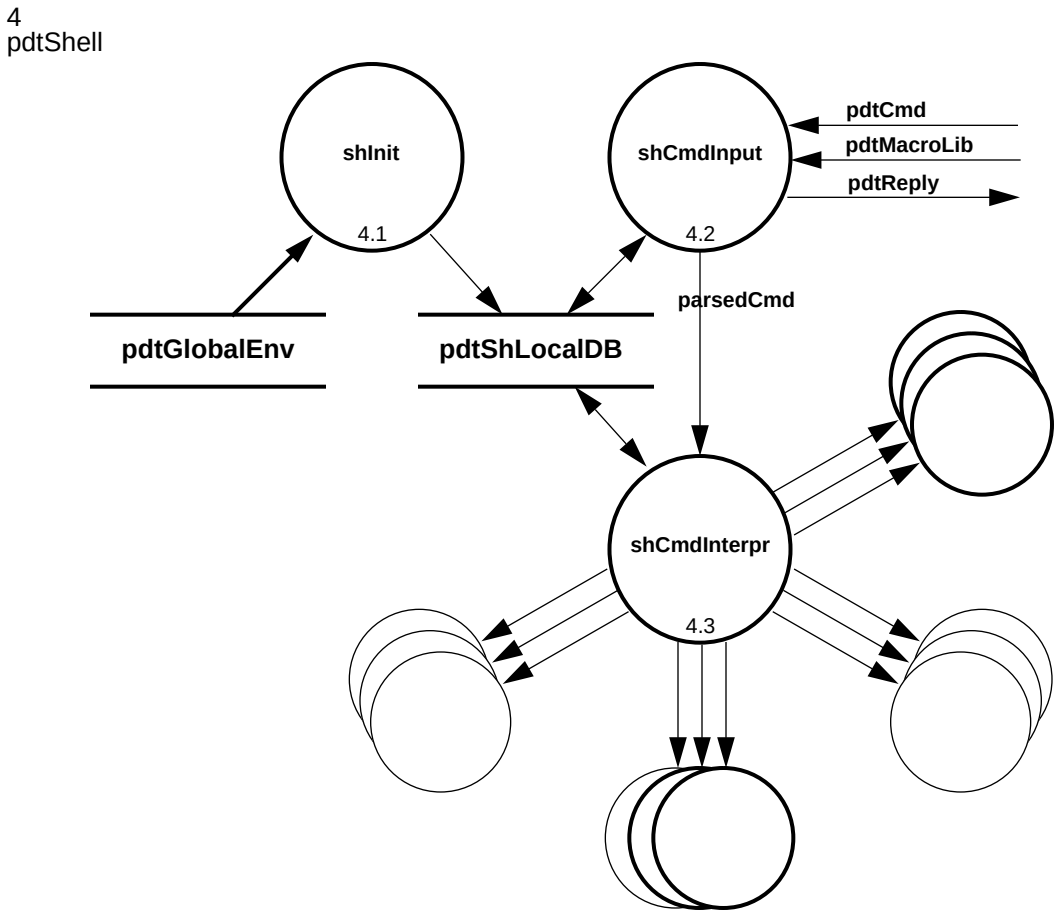


Figure 4. PDT - pdtShell Subsystem

5.7.1 Data Entities

pdtGlobalEnv - This data store is described above in the Common Data Stores section. It will be used to initialize corresponding fields in the shell local data base. Later

on, these local values can be changed while global values have to remain unchanged.

pdtLocalDb - Contains all shell variables. It also includes copies of global variables which can be changed locally by the shell user.

Because all functional subsystems are going to be implemented as reentrant programs, the local data base must allocate memory for their variables.

pdtCmd - Any PDT command as specified in the PDT Command Reference. See Appendix A for details.

pdtMacroLib - Directory which will be used to look for PDT shell scripts.

pdtReply - Results of the PDT command execution or error message.

parsedCmd - Lexically correct PDT command which is broken into tokens. It is a good idea to have all expression calculations to be already done at this point.

5.7.2 Processes

5.7.2.1 shInit

This routine will process shell input parameters, initialize shell local variables and copy values of global variables to the local data base.

5.7.2.2 shCmdInput

This routine provides a main shell loop which includes:

- issuing of the PDT prompt;
- entering the next PDT command;
- command parsing and token list creating (expressions calculation should be done here);
- command interpreter calling to perform the entered command;

A normal exit from the command input loop is made by the shell exit command.

5.7.2.3 shCmdInterpr

This routine accepts a orrect PDT command and, based on the command name will choose proper subroutine (*executive subroutine*) to perform the required action. Sometimes it is necessary to take in account not only the command name but also one or more command parameters to make proper selection. Usually it may happen when different command options require different security levels for execution.

Executive subroutines can reside in the same module (*pdtCmdIntrpr*) or be part of one of the functional subsystem. In any case, the *executive subroutine* returns control to the main loop program. All data exchanges should be done by means of the function call parameters and the shell local data base.

5.8 pdtSL1Server Subsystem

In terms of the “SL1 Remote Function Call” document written by Denny Landaveri [8], this subsystem is part of the SL1 Server, and PDT shells are clients requesting services from this server.

5.8.1 Data Entities

5.8.2 Processes

5.8.2.1 On the SL1 side

A Global Procedure name is **pdtServer**.

An Application ID is **PDT**.

There will be the following major functions (operations) provided by the Server to the PDT Application:

- pdtShowTN (function ID is **SHOW_TN**);
- pdtShowCust (function ID is **SHOW_CUST**);
- pdtShowNwk (function ID is **SHOW_NWK**);
- pdtDisplay (function ID is **SHOW_MISC**).

It looks like that the former RESDB code could be used to implement most of these functions. The only change required is to replace former output procedures. The new procedures should accept output device Fd as an input parameter and direct all output to that device.

5.8.2.2 On the PDT side

There is a corresponding routine for each SL1 Server major function on the PDT side. All these routines perform additional command parameters checking, create an input parameter list for the server call and then call SL1 Server to perform a required operation.

5.8.3 Issues for the pdtSL1Server Subsystem

1. Is it necessary to call pdtSL1ServerInit, during PDT restart? Does any type of PDT restart should do it?
2. It is necessary to rewrite RESDB output procedures to make them write data directly to the shell standard output device. This device Fd will be passed to the pdtSL1Server by the pdtShell when it requests any service. (This information should be placed into the shell local DB).
3. How can SL1Server provide for PDT the following operations: CPBRK, CPSTEP, CPGO?

5.9 pdtDebug Subsystem

This subsystem is to provide facilities which allow to suspend (break) regular task execution, perform necessary debugging and diagnostic actions, and then continue regular task execution again. For this purposes PDT uses *breakpoints*. Breakpoint is defined by the pair (event, action) and usually acts as follows:

1. When the specified event occurs, task is suspended.
2. An action is performed, if any was bound with this event.
3. Some additional actions can be specified manually by the user (operator).
4. Task continues its execution.

5.9.1 pdtDbgEvent Object Class

It is convinient to build an object class with the pdtDbgEvent as a class member. The following is this object class description:

```
CLASS pdtDbgEvent
{
PRIVATE:
struct pdtEvent
        {
        int         pdtEvtType;         /* event type: READ, WRITE, EXE      */
        int         pdtEvtTask;         /* task ID, for task related events, or 0 */
        int         pdtEvtAddr1;        /* event region starting address     */
        int         pdtEvtAddr2;        /* event region ending address       */
        int         pdtEvtInstr;        /* instruction, substituted by trap code */
        int         pdtEvtShell;        /* shell ID to interact to           */
        MNAME pdtEvtMacro;              /* macro name if defined             */
        ENAME  pdtEvtName;              /* event optional name               */
        short       pdtEvtFlags         /* flags                             */
        };
int EVT_TBL_LEN             16      /* event table length  !!magic           */
int EVT_MAX                 16      /* max number of events by default       */
structure pdtEvtTbl  pdtEvent [EVT_TBL_LEN]     /* event table */
PUBLIC:
pdtEvtClassInit (maxEvents, evtHandler);            /* class initialization function    */
```

```
pdtEvtDefine (evtType, evtAddr1, evtAddr2);     /* constructor function          */
pdtEvtDelete (evtNo);                           /* destructor function           */
pdtEvtEnable (evtNo);                           /* event enable function         */
pdtEvtDisable (evtNo);                          /* event disable function        */
pdtEvtActionBind(evtNo, macName);               /* bind action function          */
pdtEvtActionGet (evtNo);                        /* get  action macro function    */
pdtEvtNameSet (shInFd);                         /* define event symbolic name    */
pdtEvtNameToNo (shInFd);                        /* get event number              */
pdtEvtNoToName (shInFd);                        /* get event symbolic name       */
}
```

5.9.2 Issues for the pdtDebug Subsystem

1. New PDT command should be added to specify which task will be under debugging. Does it mean that only one task can be debugged at a time by PDT?

6 Testing (HLD)

6.1 Unit test

6.1.1 PDT Starting

1. Start PDT from Root Task.
2. Start PDT from VxWorks shell.
3. Use .CFG files to setup PDT environment.

6.1.2 Expression Evaluation and Syntax Checking

1. Use arithmetic expressions and check reported results.
2. Use expressions involve names from Global symbol table. Verify results.
3. Use expressions with indirect references to values. Verify results.
4. Use expressions with incorrect syntax. Verify error reported.
5. Use undefined PDT command. Verify error reported.
6. Use PDT commands with incorrect operands type. Verify errors reported.
7. Use PDT commands with wrong number of operands. Verify errors reported.

6.1.3 Managing Operating Environment

1. Use QENV command to display all environment variables values.
2. Use SETENV command to change value of several environment variables. To verify changes display values of these variables.
3. Use MEM command to display memory map.

6.1.4 Basic Debugging Level

1. Use BSet command to set breakpoint. Verify result by using of the BList command.
2. Use BEnable, BDIsable, BClear commands and verify results by using of the BList command.
3. Use Display command to display contents of memory.
4. Use Modify command to change contents of memory. Verify result by using the Display command.
5. Use Unassemble command to display program code.
6. Use Assemble command to create a piece of program code. Verify result by using of the Unassemble command.

6.1.5 PDT Session Logging

1. Use LOG STATUS command to check name and status of log file before and after each logging related commands.
2. Use LOG OPEN command to open log file. Enter several PDT commands which have to be placed to the log file.
3. Use LOG DISABLE command to deactivate logging. Enter several PDT commands which should not be placed to the log file.
4. Use LOG ENABLE command to activate logging. Enter several PDT commands which have to be placed to the log file.
5. Use LOG CLOSE command to close log file. Check if all previous commands was performed properly by using TYPE utility to display log file.

6.1.6 Disk Maintenance

1. Use CHKDSK command to check disk information after FORMAT and LABEL commands to verify results.
2. Use FORMAT command to format floppy disk.
3. Use VOL command to display floppy disk volume name.
4. Use LABEL command to change floppy disk volume name. Verify result by using of VOL or CHKDSK command.

6.1.7 Directory Maintenance

1. Use TREE command to check directory tree after CHDIR, MKDIR and RMDIR commands to verify results.
2. Use CHDIR command to establish pathes to different directories.
3. Use MKDIR command to create new directories.
4. Use RMDIR command to remove directories created.

6.1.8 File Manipulation

1. Use DIR command to check disk files after COPY, DELETE and MOVE commands to verify results.
2. Use COPY command for any existing file to create new one with different name.
3. Use MOVE command to move file to another directory and to rename it.
4. Use DELETE command to delete file.

6.1.9 Macro Facility

1. Use MBIND command to bind breakpoint with an existing macro. Verify result by using BList command.
2. Use MBIND command to create and bind in-line macro with the breakpoint. Verify result by using BList command.
3. Use MLIB command to change current using macro library. Verify result by using QENV MLIB command.
4. Use MSYNX command to check syntax of the macro in macro library.
5. Use MRUN command to execute macro from macro library.

6.1.10 Miscellaneous

1. Use EDIT command to create new text file. Use TYPE command to verify result.
2. Use HELP command to get common help information (help topics) and to get help information on single commands.