

Pre-Release B Testbed Design Specification for the ECS Project

Technical Paper

July 1997

Prepared Under Contract NAS5-60000

PREPARED BY

D. Patel, J. Watson, and Testbed Team Members Date
EOSDIS Core System Project

APPROVED BY

Karl Leatham, Deputy Program Manager Date
EOSDIS Core System Project

Hughes Information Technology Systems
Upper Marlboro, Maryland

This page intentionally left blank.

Abstract

This "as-built" design documentation paper is prepared in response to the white paper "Implementation Plan for the Pre-Release B Testbed for the ECS Project", S416-WP-001-001. The document contains "as-built" information, at the "sell off", for the hardware and software for the delivered testbed at four DAAC sites, GSFC, LaRC, EDC, and NSIDC.

Keywords: as-built, Testbed, hardware, software, network, Planning, Processing, OMT, Pre-Release B

This page intentionally left blank.

Contents

Abstract

1. Introduction

1.1	Purpose.....	1-1
1.2	Scope.....	1-1
1.3	Document Organization.....	1-1

2. Related Documents

2.1	Parent Documents.....	2-1
2.2	Applicable Documents.....	2-1
2.3	Information Documents Referenced.....	2-3

3. Testbed Design Overview

3.1°	Background.....	3-1
3.1.1	Testbed Functional Overview.....	3-1
3.2°	Testbed Architecture.....	3-2
3.2.1	Subsystems and Functions.....	3-2
3.2.2	Subsystems Components and Functionality.....	3-8

4. Communications Subsystem (CSS) CSCI

4.1	DCI CSCI Overview.....	4-1
4.2	DCI CSCI Service Descriptions and Object Models.....	4-2

5. Systems Management Subsystem (MSS) CSCIs

5.1	Introduction and Context.....	5-1
5.2	MSS Services and CSCIs.....	5-1

6. Planning Subsystem (PLN) CSCIs

6.1	Overview.....	6-1
6.2	Subsystem Context.....	6-2
6.3	Subsystem Object Model.....	6-3

7. Data Processing Subsystem (DPS) CSCIs

7.1	PRONG CSCI Overview	7-1
7.2	Subsystem Context.....	7-13
7.3	PRONG Object Model.....	7-14
7.4	AITTL CSCI.....	7-143

8. Integrated Metastorage Factory (IMF)

8.1	Introduction.....	8-1
8.2	Overview.....	8-1
8.3	IMF Context.....	8-2
8.4	IMF File Structure and Description	8-2
8.5	IMF/PDPS Interfaces	8-4
8.6	IMF Components	8-29
8.7	IMF Services, Configuration File, and Key Parameters	8-32

9. Testbed Databases

9.1	Overview.....	9-1
9.2	Database Organization	9-2
9.3	Scripts	9-14
9.4	Stored Products	9-15

10. Network Architecture and Configuration

10.1 °	Common Elements and Connectivity	10-1
10.1.1	Network Protocols	10-1
10.1.2	Network Security	10-2
10.2 °	Testbed Network at GSFC	10-2
10.2.1	IP Address Assignment and Network Connectivity at GSFC	10-3
10.2.2	Testbed Routing at GSFC	10-6
10.2.3	Network Hardware at GSFC	10-6
10.2.4	Points of Contact for GSFC Testbed Network	10-7
10.3 °	Testbed Network at LaRC	10-7
10.3.1	IP Address Assignment and Network Connectivity at LaRC	10-8
10.3.2	Testbed Routing at LaRC	10-10
10.3.3	Network Hardware at LaRC	10-10
10.2.4	Points of Contact for the LaRC Testbed Network	10-11

10.4 °	Testbed Network at EDC	10-11
10.4.1	IP Address Assignment and Network Connectivity at EDC	10-12
10.4.2	Testbed Routing at EDC	10-16
10.4.3	Network Hardware at EDC	10-16
10.4.4	Points of Contact for the EDC Network	10-17
10.5 °	Testbed Network at NSIDC	10-17
10.5.1	IP Address Assignment and Network Connectivity at NSIDC	10-18
10.5.2	Testbed Routing at NSIDC	10-22
10.5.3	Network Hardware at NSIDC	10-22
10.5.4	Points of Contact for the NSIDC Network	10-23

11. Hardware Design and Configuration

11.1 °	Subsystems, Hardware Identification, and Functions.....	11-1
11.1.1	Severs, Work Stations, Peripherals, and Communication Equipments	11-3
11.1.2	RAID Disks	11-4
11.1.3	Network File System (NFS- Infrastructure)	11-4
11.1.4	DNS Servers	11-4
11.1.5	SMC hardware	11-5
11.1.6	Software COTS Category	11-5
11.1.4	CSCI to Executables	11-7
11.2 °	Hardware Design and Configuration at GSFC	11-8
11.2.1	Hardware Configuration at GSFC	11-8
11.2.2	Software Mapping onto Hardware at GSFC	11-10
11.3 °	Hardware Design and Configuration at LaRC	11-11
11.3.1	Hardware Configuration at LaRC	11-11
11.3.2	Software Mapping onto Hardware at LaRC	11-13
11.4 °	Hardware Design and Configuration at EDC	11-15
11.4.1	Hardware Configuration at EDC	11-16
11.4.2	Software Mapping onto Hardware at EDC	11-17
11.5 °	Hardware Design and Configuration at NSIDC	11-19
11.5.1	Hardware Configuration at NSIDC	11-19
11.5.2	Software Mapping onto Hardware at NSIDC	11-20

Figures

3.2.1.1-2	Testbed Infrastructure Services and Support	3-8
1	CMIService	4-5
2	TimeService	4-9
3	ConfigurationFile	4-16

4.2.4.2-1	ECS Process Classification	4-22
4.2.4.3.1	DCEPthreadMutex Class	4-24
4.2.4.4.1.2-1	Example of General Object Model for Client Applications	4-34
6.2-1	CSCI Event Flow Context Diagram	6-3
5	Core_Library_Part_1	6-5
6	Core_Library_Part_2	6-6
7	Core_Library_Part_3	6-7
8	Planning_Workbench_Ac	6-104
7.1.4.2-1	Scheduling Jobs Using AutoSys	7-11
7.1.4.2-2	Initiating Processing Components Using AutoSys	7-12
7.2-1	Processing CSCI Context Diagram.....	7-14
33	Database_Interface.....	7-15
34	Data_Management	7-36
35	Exception	7-48
36	Execution_Management	7-50
37	Job_Management	7-80
38	QA_Monitor.....	7-90
39	Main	7-97
40	Resource_Management.....	7-109
7.4.2-1	Algorithm Integration and Test Tools Context Diagram.....	7-147
7.4.4.1-1	View Documentation	7-148
7.4.4.2-1	Check Standard	7-149
7.4.4.3-1	Analyze Code.....	7-150
7.4.4.4-1	Examine Data.....	7-151
7.4.4.5-1	Compare Files	7-152
7.4.4.6-1	Measure Resource Requirements.....	7-153
7.4.4.7-1	Update IMF Data Server.....	7-154
7.4.4.8-1	Update PDPS Database.....	7-155
7.4.4.9-1	Manage Reports	7-156
41	Data_Server.....	7-157
42	Metadata_GUI.....	7-162
43	Metadata.....	7-182
44	IR1_Heritage_Common_Library	7-189
45	IR1_Heritage_File_Differencing_Tools	7-197
46	IR1_Heritage_Process_Control_File_Checker.....	7-211
47	IR1_Heritage_Prohibited_Function_Checker.....	7-218
48	IR1_Heritage_Prolog_Extractor	7-230
49	IR1_Heritage_Top_Level_GUI_Manager.....	7-235
8.3-1	IMS Context Diagram.....	8-2

8.4-1	Structure of IMF Architecture	8-3
8.5-1	IMF/PDPS Interface Diagram.....	8-5
9.1-1	Testbed Database	9-1
9.2.1-1	Execution and Data Management	9-3
9.2.1-2	Execution and Data Management	9-4
9.2.1-3	Production Plan.....	9-5
9.2.1-4	Production Request Edit	9-6
9.2.1-5	Resource Management.....	9-7
10.2-1	Network Topology at GSFC	10-3
10.2.1-1	Network Connectivity and Addressing for the Testbed at GSFC	10-4
10.2.1-2	Network Connectivity for the Testbed SMC at GSFC	10-5
10.2.2-1	GSFC Testbed Route Advisement	10-6
10.3-1	Network Topology at LaRC	10-8
10.3.1-1	Network Connectivity and Addressing for the Testbed at LaRC	10-9
10.3.2-1	LaRC Testbed Route Advisement	10-10
10.4-1	Network Topology at EDC	10-12
10.4.1-1	Network Connectivity at EDC	10-13
10.4.2-1	EDC Testbed Route Advisement	10-16
10.5-1	Network Topology at NSIDC	10-18
10.5.1-1	Network Connectivity at NSIDC	10-19
10.5.2-1	NSIDC Testbed Route Advisement	10-22

Tables

3.2.1-1	Testbed Functions to Subsystem Mapping	3-3
10.2.1-1	GSFC Testbed Network Address Space Information	10-3
10.2.3-1	GSFC LAN Hardware	10-7
10.2.4-1	Points of Contact for GSFC Network	10-7
10.3.1-1	LaRC Testbed Network Address Space Information	10-8
10.3.3-1	LaRC LAN Hardware	10-11
10.3.4-1	Points of Contact for LaRC Network	10-11
10.4.1-1	EDC Testbed Network Address Space Information	10-14
10.4.1-2	Network Address for the Host at EDC	10-14
10.4.1-3	Network Address for the Network Products at EDC	10-15
10.4.3-1	EDC LAN Hardware	10-17
10.4.4-1	Points of Contact for EDC Network	10-17
10.5.1-1	NSIDC Testbed Network Address Space Information	10-18
10.5.1-2	Network Address for the Host at NSIDC	10-20
10.5.1-3	Network Address for the Network Products at NSIDC	10-21
10.5.3-1	NSIDC LAN Hardware	10-23

10.2.4-1	Points of Contact for NSIDC Network	10-23
11-1	Hardware Subsystems and Functions	11-1
11.1.6 -1	COTS Categories	11-6
11.2.1-1	Testbed Hardware at GSFC	11-8
11.2.2-1	Hardware Software Mapping to Hardware Platforms at GSFC	11-10
11.3-1	Testbed Hardware at LaRC	11-12
11.4-1	Testbed Hardware at EDC	11-16
11.4.2-1	Hardware Software Mapping to Hardware Platforms at EDC	11-17
11.5.1-1	Testbed Hardware at NSIDC	11-19
11.5.2-1	Hardware Software Mapping to Hardware Platforms at NSIDC	11-21

Abbreviations and Acronyms

1. Introduction

1.1 Purpose

The purpose of this document is to capture the design of the Pre-Release B Testbed (hereafter referred to as the "Testbed") that meets all the functional capabilities as described in the Testbed implementation plan, S416-WP-001-001. This document describes the design of the Testbed hardware and software items in sufficient details for:

- the ECS M & O organization to perform the required corrective maintenance (design problem resolution), post Testbed deployment, that is consistent with the established maintenance concept. The maintenance concept for the Testbed is defined in the Section 9 of the Testbed implementation plan, S416-WP-001-001.
- the ECS M & O organization to perform the required operational functions at the four DAAC sites, post Testbed deployment, that are consistent with the established operational procedures. The operational concept for the Testbed is defined in the Section 7 of the Testbed implementation plan, S416-WP-001-001.
- DAAC personnel to become familiar with the design of the Testbed, it's inherent capabilities, and the limitations thereof.

1.2 Scope

This document describes the "as-built" architecture and design of the computer software and hardware, databases, and network items for each of the four Testbed systems - once each Testbed is hosted at the DAAC sites (GSFC, LaRC, EDC, and NSIDC). The design information contained herein is applicable to each DAAC site unless it has been otherwise stated. This design document assumes that the reader has access to a supplementary set of information for the performance his/her specific maintenance task(s), as mentioned in the relevant sections of this document. For example, M & O personnel would have access to the source code listing for the custom code and be able to obtain detailed information about the as-built custom software components of the Testbed.

1.3 Document Organization

This document is organized to describe the Testbed design as follows:

Section 1 provides information regarding the identification, purpose, scope, and organization of this document.

Section 2 provides a listing of the related documents, which were used as source information for this document, and lists the reference material.

Section 3 provides a functional overview of the Testbed system and its interfaces (function and data) with the external entities. This section primarily focuses on the structural overview of the Testbed and identifies constituent subsystems including their configurable items (both hardware and software) for each DAAC.

Section 4 describes the design of the Communication Subsystem's services. It also documents the applicable external interfaces and COTS/OTS dependencies.

Section 5 describes the design of the System Management Subsystem's services. It also documents the applicable external interfaces and COTS/OTS dependencies.

Section 6 describes the design of Planning Subsystem's "PLANG" CSCI. It also documents the applicable external interfaces and COTS/OTS dependencies. Note: The design of the Client Subsystem CSCI's component, EOSView, is documented separately. Thus, it is only referenced in this specification. Refer to Section 2 for the applicable document number(s).

Section 7 describes the design of Data Processing Subsystem's "PRONG" and "AITTL" CSCIs. It also documents the applicable external interfaces and COTS/OTS dependencies. Note: The design of the PRONG CSCI's components, the HDF-EOS and the SDP Toolkit, are documented separately. Therefore, they are only referenced in this specification. Refer to Section 2 for the applicable document number(s).

Section 8 describes the design of Integrated Metastorage Factory (IMF) CSCI.

Section 9 describes the design of the Testbed databases.

Section 10 provides a brief description of the network configuration and LAN architecture at each DAAC site.

Section 11 provides the hardware configuration for the processing nodes, including the storage, and the peripheral devices for each DAAC site. It also provides the hardware to software mapping for COTS, and the custom software (executables) mapping to each hardware platform.

Acronyms and Abbreviation section contains an alphabetical listing of the acronyms and abbreviations used in this document.

The design of the Client Subsystem (CLS) component, EOSView, is documented separately and therefore is only referenced in this specification. Refer to Section 2 for the applicable document number(s).

2. Related Documents

2.1 Parent Documents

The parent document is the document from which the scope and content of this "as-built" Testbed design specification is derived.

S416-WP-001-001 Implementation plan for the Pre-Release B Testbed for the ECS Project

2.2 Applicable Documents

The following documents are ECS controlled documents and they are directly applicable to this "as-built" Testbed design specification.

456-TP-013-001	HDF-EOS Design for the ECS Project
456-TP-012-001	EOSView Design for the ECS Project
445-TP-001-001	SDP Toolkit Design Specification
162-TD-001-002	Science Software I & T Operational Procedure
205-CD-002-003	Software Developer's Guide to Preparation, Debug, Integration, and Test with ECS
420-TD-010-003	GSFC Pre-Release B Testbed HW/Network
420-TD-002-004	Pre-Release B Testbed HW diagram for GSFC
420-TD-006-009	GSFC Pre-Release B Testbed Hardware-Software mapping baseline
420-TD-037-002	Pre-Release B Testbed baseline for Science Processing RAID at GSFC
420-TD-054-001	Pre-Release B IMF RAID Configuration baseline for GSFC
420-TD-051-001	Pre-Release B IMF RAID Configuration baseline for GSFC
420-TD-066-001	Pre-Release B Testbed Staging RAID configuration baseline for GSFC
420-TD-004-004	Pre-Release B Testbed HW diagram for SMC
420-TD-025-005	Pre-Release B Testbed MSS/CSS (Primary) RAID partitions for SMC, GSFC, and LaRC
420-TD-024-004	Pre-Release B Testbed MSS/CSS (Secondary) RAID partitions for SMC, GSFC and LaRC
420-TD-011-004	Pre-Release B Testbed Network Diagram for LaRC
420-TD-003-002	Pre-Release B Testbed HW diagram for LaRC
420-TD-007-005	Pre-Release B Testbed HW/SW mapping baseline for LaRC
420-TD-038-002	Pre-Release B Testbed baseline for Science Processing RAID at LaRC
420-TD-055-001	Pre-Release B Testbed IMF RAID Configuration baseline for LaRC
420-TD-043-002	Pre-Release B Testbed Planning Server Disk partitions for LaRC
420-TD-067-001	Pre-Release B Testbed Staging RAID configuration baseline for LaRC

420-TD- 048-001	Pre-Release B Testbed Network Diagram for EDC
420-TD-050-001	Pre-Release B Testbed Hardware Diagram for EDC
420-TD-046-002	Pre-Release B Testbed HW/SW mapping baseline for EDC
420-TD-056-001	Pre-Release B Testbed Science Processing RAID Configuration for EDC
420-TD-060-001	Pre-Release B Testbed NFS RAID configuration baseline for EDC
420-TD-058-001	Pre-Release B Testbed Staging RAID configuration baseline for EDC
420-TD-064-001	Pre-Release B Testbed MSS/CSS RAID Configuration baseline for EDC
420-TD-062-001	Pre-Release B Testbed Planning/Queuing Server RAID for EDC
420-TD-052-001	Pre-Release B Testbed IMF RAID Configuration baseline for EDC
420-TD-049-001	Pre-Release B Testbed Network Diagram for NSIDC
420-TD-051-001	Pre-Release B Testbed HW Diagram for NSIDC
420-TD-047-002	Pre-Release B Hardware-Software mapping baseline for NSIDC
420-TD-061-001	Pre-Release B Testbed NFS RAID configuration baseline for NSIDC
420-TD-059-001	Pre-Release B Testbed Staging RAID configuration baseline for NSIDC
420-TD-053-001	Pre-Release B IMF RAID Configuration baseline for NSIDC
420-TD-063-001	Pre-Release B Testbed Planning/Queuing Server RAID for NSIDC
420-TD-065-001	Pre-Release B Testbed MSS/CSS RAID Configuration baseline for NSIDC
420-TD-019-006	Pre-Release B Testbed portmaps baseline
420-TD-014-002	Pre-Release B Testbed SGI IRIX 5.3 Operating system Patch baseline
420-TD-031-002	Pre-Release B Testbed SGI IRIX 6.2 Operating system Patch baseline
420-TD- 014-002	Pre-Release B Testbed HP Operating system Patch Baseline
420-TD-012-002	Pre-Release B Testbed Sun Solaris Operating system Patch baseline
420-TD-068-001	Pre-Release B Testbed CSCI-Software executable mapping baseline
420- TD-029-003	Pre-Release B Filenames and guidelines and standards baseline
456-TR-001-001	GSFC Pre-Release B Testbed Version Description Document(VDD)
456-TR-002-001	LaRC Pre-Release B Testbed Version Description Document(VDD)
456-TR-003-001	EDC Pre-Release B Testbed Version Description Document(VDD)
456-TR-004-001	NSIDC Pre-Release B Testbed Version Description Document(VDD)

2.3 Information Documents Referenced

List of software COTS/OTS documents:

2.3.1 Autosys

AutoSys User Manual Version 3.2

2.3.2 HPOV

Using Network Node Manager

Network Node Manager Reference

A Guide to Scalability and Distribution for NNM

Using Relational Databases with Network Node Manager for
Integration Utilities for HPOV

Network Node Manager Products Installation Guide

2.3.3 Tivoli

Tivoli Management Platform Reference Manual

Tivoli Management Platform Planning and Installation Guide

Tivoli Management Platform User's Guide

Tivoli User and Group Reference Manual

Tivoli User and Group Installation Guide

Tivoli User and Group Management Guide

Tivoli Host Management Guide

Tivoli/Sentry User's Guide

Tivoli/Sentry Monitoring Collection Reference

Tivoli/Courier User's Guide

Tivoli/Courier Reference Manual

Tivoli NIS Management Guide

2.3.4 Remedy (Action Request System)

Programmer's Guide

Administrator's Guide for OSF/Motif

User's Guide for OSF/Motif

Troubleshooting and Error Message Guide

2.3.5 PNM

Network/Communications Management Volume 1

2.3.6 ESSM

Enterprise SQL Server Manager User's Guide

2.3.7 Networker

Legato Networker Administrator's Guide

Legato Networker User's Guide

Legato Networker Installation and Maintenance Guide

2.3.8 IQ

Intelligent Query and IQ Access User's Guide

IQ System Manager's Guide

2.3.9 XRP-II

XRP-II Datalook - Datarite Reference Manual

XRP-II Inventory Management Reference Manual

XRP-II Material Requirements Reference Manual

XRP-II Product Information Reference Manual

XRP-II Purchasing Management Reference Manual

XRP-II Shop Floor Control Reference Manual

XRP-II System Reference Manual

XRP-II Tools, Techniques, and Conventions Manual

XRP-II Work Order Processing Reference Manual

XRP-II Tools, Techniques, and Conventions Manual

XRP-II Work Order Processing Reference Manual

UNIFY Developer's Tutorial

UNIFY Direct HLI Programmer's Manual

UNIFY Developer's Reference Product Enhancements Manual

2.3.10 DDTS

PureDDTS Administrator's Manual

PureDDTS User's Manual

PureDDTS Manual Pages Reference Guide

2.3.11 ClearCase

ClearCase Administrator's Manual

ClearCase User's Manual

ClearCase Reference Manual

2.3.12 Network COTS

Cabletron Ethernet Hub	User's Guide for MicroMAAC-24E, MicroMMAC-32E, MicroMAAC-34E, 10Base-T Intelligent Stackable Hub
Cabletron Ethernet Hub	On-line documentation from Cabletron's Web Site at http://www.cabletron.com
Cabletron Systems	MicroMMAC-22E User's Guide, dated April 1994
Cabletron Systems	BRIM-F6 User's Guide, dated October 1994
SynOptics (Bay Networks)	Using the 291x FDDI Workgroup Concentrator, dated May 1994 FDDI Concentrator)
Fore Systems PowerHub	PowerHub 7000 Installation and Configuration Manual
Fore Systems PowerHub	PowerHub Software Manual
Hayes Modem	Manual of floppy disk for Windows 3.1 or 95 platform
Hayes Modem	On-line documentation from Hayes' Web Site at http://www.hayes.com
Cisco Remote Access Server	CISCO 2500 Series Access Server User Guide
Cisco	Internetworking Operating System (IOS) software and hardware manuals on CD for Windows 3.1 or 95, Mac, or Unix platform for CISCO 2500
Cisco	On-line documentation from Cisco's Web Site at http://www.cisco.com for Cisco 7507 Router
Cisco	Cisco 7507 Installation and Maintenance manual

This page intentionally left blank.

3. Testbed Design Overview

3.1 Background

The primary goal for the Testbed is to support AM-1 and SAGE III Science Software Integration and Test (SSIT) activity. The SSI&T will be supported at the GSFC, LaRC, EDC, and NSIDC DAACs for science software delivered by the ASTER, CERES, MISR, MODIS, MOPITT, and SAGE III instrument teams.

SSI&T is defined as the process by which the science software developed by the instrument teams (ITs) at local SCFs is tested and integrated into the ECS at the DAACs. The activities associated with SSI&T can be logically separated into two categories: pre-SSI&T and formal SSI&T. Pre-SSI&T activities are those that do not require Pre-Release B Testbed unique capabilities. Inspection, standards checking, and building the science software with the SCF Toolkit are pre-SSI&T activities. Formal SSI&T activities are those that require Pre-Release B Testbed capabilities. These activities include populating the ECS Testbed databases, building the science software with the DAAC Toolkit, integration of each science software PGE into the ECS Pre-Release B Testbed, and successful execution using the Testbed software.

The primary focus of the Testbed is SSI&T. The instrument teams, with the support of the ECS and DAAC SSI&T personnel, will perform a sequence of activities for each delivered PGE. The goals of the SSI&T process are:

1. To ensure that the delivered PGEs conform to ESDIS project standards
2. To port the PGEs to computer platforms at the DAACs
3. To integrate the PGEs with the DAAC version of the SDP Toolkit and execute them using the ECS Testbed software
4. To verify that the data products and results are the same as those produced at the SCFs

Refer to the "Implementation Plan for the Pre-Release B Testbed" (S416-WP-001-001) for the details on SSI&T activities, and remote access and data distribution for allowing the instrument team to participate from their remote location (such as their local SCFs).

3.1.1 Testbed Functional Overview

The Testbed provides the following major functions;

- provides comprehensive SSI&T support including
 - acquire input data files (HDF-EOS and non-HDF-EOS) from the science archive and stage them in PDPS
 - register PGEs
 - set up subscriptions for planning
 - enter production requests
 - develop and activate a production plan
 - schedule and monitor jobs with the AutoSys COTS

- manage PGE execution and insertion into to the science archive (the data store) output data files (HDF-EOS and non-HDF-EOS)
- Q/A monitoring of products including use of EOSView for visualization
- acquire multiple data granules (HDF-EOS and non-HDF-EOS) having the same ESDT and different temporal coverage
- provides data store components that will
 - interface with the Planning and Data Processing components
 - support metadata validation
 - support subscription registration and notification,
 - allow ESDT additions via a GUI interface
- monitors Testbed failures (e.g., hardware and network failures)

3.2 Testbed Architecture

The Testbed is installed and configured at four DAAC sites, GSFC, LaRC, EDC, and NSIDC. The Testbed is configured using single cell DCE (OSF 1.0.3) architecture which encompasses all four Testbed resources and cross site boundaries. The single cell for the Testbed represents an optimum infrastructure and associated security. The Testbed provides checking the privilege of a principle (i.e., user authorization). The Sybase RDBMS and the UNIX operating system provide the integrity of data and are responsible for data access privileges in storage. The master cell for the Testbed is the SMC Testbed at GSFC, while each Testbed site contains the replica. However, for performing normal routine functions, each site is autonomous.

The Testbed is composed of seven (7) subsystems as defined in Section 3.2.1. The design of each Testbed employs FDDI based LAN networks, OODCE, and client/server architecture. It consists of heterogeneous platforms from HP, SUN, and SGI. The Testbed architecture contains the universal reference (UR) mechanism, which provides a means for identifying and referencing the data and service objects throughout the Testbed in a standardized manner. The functionality of the Testbed is distributed among the various subsystems, and they are described in Section 3.2.2 of this document.

In order to provide the reliable production and archiving of science data, the Testbed utilizes field-proven, robust COTS hardware. It employs reliable technology such as RAID storage for staging data, FDDI fault tolerant LAN and interface devices for intra-DAAC communications, and workstation and server hardware from OEM manufacturers such as SUN, SGI, and HP.

The following sections provide an overview, functional allocation, and design information for the Testbed architecture and its major constituent design elements.

3.2.1 Subsystems and Functions

The Testbed consists of seven (7) subsystems based on its physical and logical structure. They provide hardware and software resources needed to implement the Testbed functionality. Each subsystem consists of one or more computer software configuration items (CSCIs) and/or hardware configuration items (HWCI). These CIs are a combination of custom software and/or a

custom (“as is”) configuration of COTS/OTS hardware and software items. Detailed design information of the custom software items is provided in Sections 4 through 9.

List of the Testbed subsystems:

- Communication Subsystem (CSS); custom and COTS
- System Management Subsystem (MSS); custom and COTS
- Data Processing Subsystem (DPS); custom and COTS
- Planning Subsystem (PLS); custom and COTS
- Integrated Metastorage Factory (IMF); custom software only
- Infrastructure-Network File System (NFS); all COTS
- Internetworking Subsystem (ISS); all COTS

A detailed list of functional capabilities for the Testbed subsystems is provided in Table 3.2.1-1.

Table 3.2.1-1. Testbed Functions to Subsystem Mapping (1 of 4)

Activity Number	SSI&T Activity	Testbed Entity and Functions	Testbed Subsystem
Pre-SSI&T:			
1	Accessing the delivered algorithm package (DAP)	Network File System (NFS)	Infrastructure
2	Inspection of DAP contents	UNIX text editors, SSIT Manager, Printer	DPS
3	Review of delivered documentation	Sun, SSIT Manager GUI, Document viewers (WABI, ghostview, Acrobat Reader, UNIX text editors), Printer	DPS
4	Placing of DAPs under configuration management	Sun, ClearCase and scripts (from M&O)	N/A
5	Checking of science software for standards compliance	Sun, SGI, SSIT Manager GUI, SSIT Tools (Prohibited Function Checker, Prolog Extractor, File comparison tools), FORCHECK, C, Fortran 90, and Ada SGI compilers	DPS
6	Building of the science software into PGEs with the SCF Toolkit	SGI, Release A SCF Toolkit, C, FORTRAN 77, Fortran 90, and Ada SGI compilers	Toolkit Compilers

Table 3.2.1-1. Testbed Functions to Subsystem Mapping (2 of 4)

Activity Number	SSI&T Activity	Testbed Entity and Functions	Testbed Subsystem
7	Running of the PGEs from the command line	SGI	N/A
8	Collection of performance statistics for the PGEs	SGI, Rusage profiling (command line)	N/A
9	Examination of output log files from PGE runs	SGI, UNIX text editors, Printer	N/A
10	Viewing of output products and comparison to delivered test data	EOSView SSIT Manager, hdiff (HDF command-line tool), HDF GUI File Comparison Tool, ASCII File Comparison Tool, Binary File Assistant Comparison Tool, IDL	DPS (Note: EOSView is hosted on PLS and DPS hardware)
11	Reporting of science software problems	DDTS (Distributed Defect Tracking System) (COTS)	DDTS (hosted on MSS)
12	Reporting of ECS problems	DDTS	N/A
Formal SSI&T:			
13	Registration of ESDTs	Add new or /new version ESDTs, ESDT generation scripts	DPS/IMF
14A	Registration of the PGEs in the PDPS database	Sun, SSIT Manager GUI, PGE registration tools, Update PDPS/SSIT DB GUI	DPS
14B	Registration subscriptions for dynamic inputs and/or outputs that are input to downstream PGEs	Subscription Editor (command line interface)	PLS
15	Inserting test data into the data store	Insert PGE .exe tar file, Insert static and dynamic test data granules	DPS
16	Building of the science software into PGEs with the DAAC Toolkit	SGI, Testbed DAAC Toolkit consistent with the SDP toolkit, C, FORTRAN 77, Fortran 90, and Ada SGI compilers per Release A baseline	Toolkit and Compilers
17A	Set up resource reservations for SSIT resources	PLANG (PLANNING) CI Resource Planning Workbench	PLS
17B	Submit individual production requests (PRs) to run single PGEs within PDPS	PLANG (Planning) CI Production Request Editor	PLS

Table 3.2.1-1. Testbed Functions to Subsystem Mapping (3 of 4)

Activity Number	SSI&T Activity	Testbed Entity and Functions	Testbed Subsystem
17C	Create candidate plan to run PGE	PLANG (Planning)CI Planning workbench	PLS
17D	Activate plan	PLANG (Planning)CI Planning workbench	PLS
17E	Run PGE	PLANG (Planning) CI Subscription Manager Manage subscription notatification. PRONG (Processing) CI: All functions except alternate processing depending on PGE exit codes. Job Management, Add/Modify/Cancel Jobs, Release Jobs. AutoSys Scheduling COTS. Resource Management Allocate/Deallocate resources. PGE Execution Management, Stage PGE, Execute PGE, Manage PGE execution, Generate Production	PLS DPS DPS DPS DPS DPS
18	Examination of output log files and production history from PGE runs	SGI, UNIX text editors, Printer, QA Monitor	DPS
19	Viewing of output products and comparison to delivered test data	QA Monitor, SSIT Manager GUI Selections	DPS
20A	Using the Planning subsystem to plan PGE chains and activate plan	PLANG (Planning) CI: Production Request (PR) Editor, Enter PR, Review/Modify PR, Generate DPRs for future and past data, Review/Modify DPR, Schedule PR and DPR, Generate Plans and Reports. Production Planning Workbench, Allocate DAAC Resources to DPRs, Develop candidate plans, Activate Plan, Update Active Plan, Production Planning Timeline	PLS PLS
20B	Run PGE chains	Same as 17E	Same as17E
20C	Examination of output log files and production history from PGE runs	SGI, UNIX text editors, Printer QA Monitor	DPS

Table 3.2.1-1. Testbed Functions to Subsystem Mapping (4 of 4)

Activity Number	SSI&T Activity	Testbed Entity and Functions	Testbed Subsystem
20D	Viewing of output products and comparison to delivered test data	QA Monitor, SSIT Manager GUI Selections	DPS
21	Ensuring consistency and correctness of output products	ECS databases, Tools/scripts for examining databases	N/A
22	Distributing data products to instrument teams	FTP	N/A
23	Reporting and tracking of science software and other Testbed problems, i.e., Non-Conformance Reports (NCRs)	DDTS	N/A
Other:			
24	Configuration Management	Clearcase	MSS
25	Remote access to authorized users	DDTS	N/A

3.2.1.1 Testbed Data Flow and Context

Figure 3.2.1.1-1 depicts the Testbed system level data flow and the interactions of the as-built application CIs for the Testbed. Figure 3.2.1.1-2 is included to show the Testbed infrastructure services and support functions.

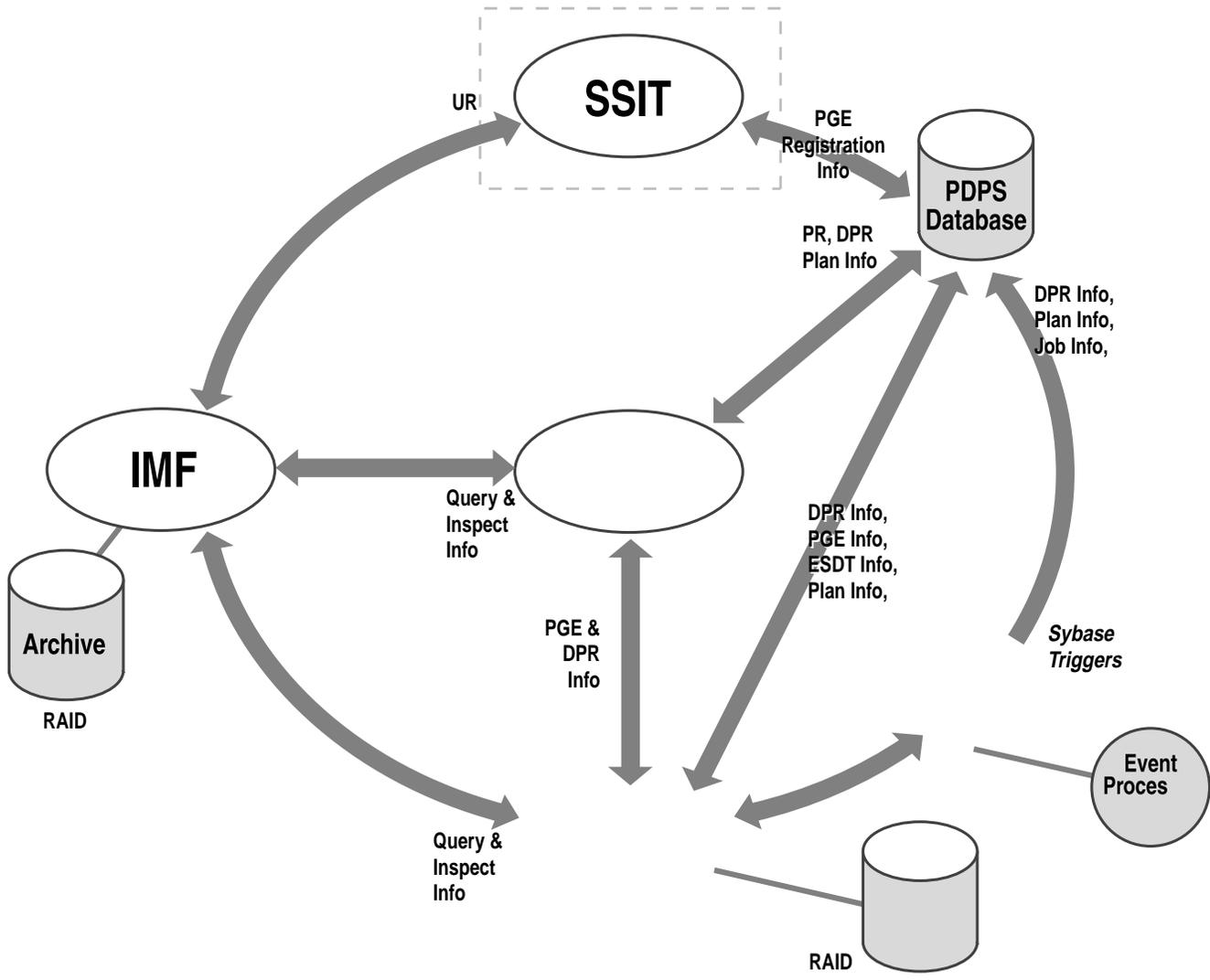


Figure 3.2.1.1-1 Testbed System Data Flow

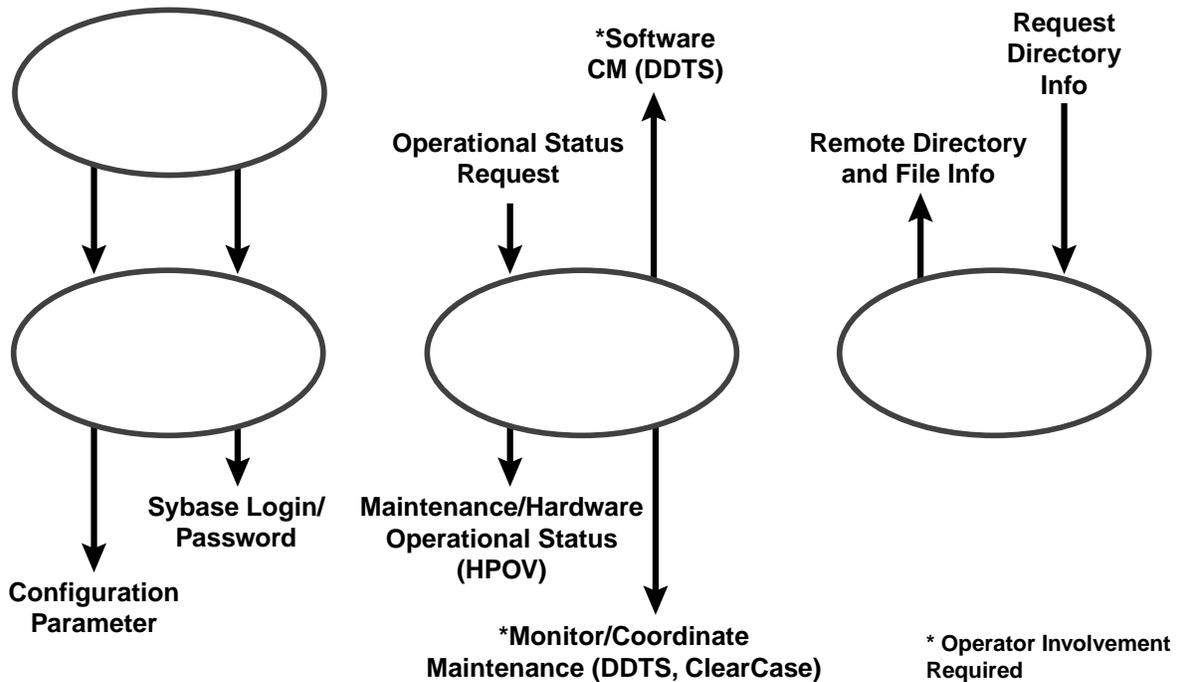


Figure 3.2.1.1-2. Testbed Infrastructure Services and Support

3.2.2 Subsystem Components and Functionality

Following subsections describe the static structure of the Testbed subsystems and their functionalities. More detailed discussions of their design breakdowns are provided in Sections 4 through 10 of this document.

3.2.2.1 Communication Subsystem (CSS) Structure and Functionality

The Communication Subsystem consists of two CIs: the Distributed Computing Software (DCI) CSCI, and the DCHI Hardware HWCI. The DCI CSCI is comprised of custom software and DCE OSF 1.0.3 COTS software. It provides following functionality:

Software (DCI)	Function
OSF DCE 1.0.3	Commensurable with OSF DCE 1.0.3 (except note 1)
COTS	E-Mail, FTP, virtual terminal
Custom Software	<ul style="list-style-type: none"> - DCE login, SYBASE login (in conjunction with CMI services) - Time Service (obtain time and format translation) - Configuration file (CMI mechanism) - Process Framework

Note 1: The DCE security features (with the use of Kerberos and the POSIX 1003.6 Access Control List (ACL) service) are stubbed in.

Note 2: FTP is used for the manual data distribution "push" for the delivered algorithm package (DAP) and for the "pull" for the test output and related products.

3.2.2.2 System Management Subsystem (MSS) Structure and Functionality

The System Management Subsystem consists of three CIs: Management Software CI (MCI) CSCI, Management Logistic Software (MLCI) CSCI, and Management Hardware CI (MHCI) HWCI. The MCI and MLCI consist of the COTS software components needed to meet the allocated subsystem functionality listed below.

COTS Software	Function
HP OpenView/MCI	Hardware/Network operational status monitoring without custom software
DDTS/MLCI	NCR tracking
Clearcase/MLCI	Configuration Management

3.2.2.3 Data Processing Subsystem (DPS) Structure and Functionality

The Data Processing Subsystem consists of four CIs: the Data Processing (PRONG) CSCI, the Algorithm Integration and Test Tools (AIITL) CSCI, the Science Processing Hardware (SPRHW) HWCI, and the Algorithm Integration and Test Hardware (AITHW) HWCI. These CSCIs consist of the software components needed to meet the allocated subsystem functionality as listed below.

Software	Function
PRONG	Job Management - Add jobs - Modify jobs - Cancel jobs - Implement job dependencies - Release jobs
AutoSys - COTS/PRONG	Scheduling - Manual startup - Manual shutdown - Full COTS functionality
AutoSys - COTS/PRONG	Resource Management - Allocate resources - Deallocate resources
AutoSys - COTS/PRONG	PGE Execution Management - Stage PGE - Execute PGE - Manage PGE execution - Generate production history (PH)
PRONG	Data Management - Manage data on local disks - Retain data on local disk - Stage science data products - Destage science data products

Software	Function
PRONG/AITTL	QA Monitor - Access product data - Update DAAC QA metadata - Visualize Product Data
HDF-EOS/PRONG	Provide extensions of HDF for EOS data products, which standardize the conventions for writing HDF files
SDP Toolkit/PRONG	Build the science software into PGEs with the DAAC Toolkit
AITTL	SSIT Manager - Enter Subscriptions - Register PGE - Archive PGE .exe tar file - Update PDPS/SSIT DB GUI
AITTL	SSIT Tools - Prohibited function checker - Binary file differencing tool - HDF comparison tool - Profiling tool - SSIT GUI support tools - IDL IMSL

Note: The EOS-View software tool is hosted on AITHW, which provides the viewing and verification functions for HDF-EOS and the HDF data files.

3.2.2.4 Planning Subsystem (PLS) Structure and Functionality

The Planning Subsystem consists of two CIs: the Production Planning Software (PLANG) CSCI, and the Production Planning Hardware (PLNHW) HWCI. The Production Planning CSCI consists of the software components to meet the allocated subsystem functionality as listed below.

Software	Function
PLANG	Production Request Editor - Enter production request - Review/modify PR - Generate DPRs for future and past data - Review/modify DPR - Schedule PR, DPR - Production request editor GUI - Generate plans and reports
PLANG	Production Planning Workbench - Allocate DAAC resources to production processing (DPRs) - Develop candidate plan(s) according to Release A production rules - Activate plan - Update active plan - Production planning timeline

Software	Function
PLANG	Resource Planning Workbench - Resource editor GUI
PLANG	Subscription Editor - Enter/withdraw subscriptions based on advertised events and services for the static database only (IOS Stubbed Version integrated)
PLANG	Subscription Manager - Manage subscription notification (monitor for data availability before submitting DPR)

Note: The EOS-View software tool is hosted on PLNHW, which provides the viewing and verification function for HDF-EOS and the HDF data files.

3.2.2.5 Integrated Metastorage Factory (IMF) Structure and Functionality

The Integrated Metastorage Factory (IMF) software consists of the Release A Data Server software “as-is”, along with additional hooks to the UNIX file system which have been implemented by overriding the original Data Server class methods. The IMF software library is linked with the Planning and Data Processing Subsystems and provides following functionality:

Software	Function
IMF	Insert Science Product - Browse - QA - Production history
IMF	Acquire products (Production History)
IMF	Query
IMF	Insert metadata (into metadata inventory) - Validate metadata - Acquire metadata
IMF	Update metadata - Update QA metadata
IMF	Add new ESDTs via GUI
IMF	Update, status, delete, add subscriptions via GUI
IMF	Trigger subscriptions based on events

3.2.2.6 Infrastructure Structure and Functionality

The Network File System (NFS) (i.e., the infrastructure) server automatically mounts and allows its clients to access remote directories across the Testbed platforms. It also provides DNS master mail functionality.

3.2.2.7 Internetworking Subsystem (ISS) Structure and Functionality

The Internetworking Subsystem (ISS) consists of COTS hardware and software items. It provides the networking services based on protocols and standards corresponding to the lower four layers of the OSI reference model. Section 10 of this document contains detailed description of the ISS.

This page intentionally left blank.

4. Communications Subsystem (CSS) CSCI

4.1 DCI CSCI Overview

The software for the Communications Subsystem (CSS) consists of the Distributed Computing Software CI (DCI). The following table lists the services provided by the DCI CSCI:

Table 4.1-1 DCI Services

Service	Remarks
Object Services	
Directory Naming (DNS); includes both Cell Directory Service and Global Directory Service	OSF DCE 1.0.3
Security	OSF DCE 1.0.3
Message Passing and RPC Calls	OSF DCE 1.0.3
Thread	OSF DCE 1.0.3
Time	OSF DCE 1.0.3
Common Facility Services	
Electronic Mail (E-Mail)	Standard component of internet
File Access (ftp)	Internet standard application for file transfer
Virtual Terminal	Remote logon to a machine
CMI	For logging into system
Configuration File	
Process Framework	

The Distributed Computing Environment (DCE) from the Open Software Foundation (OSF), version 1.0.3, is a CSS baseline COTS product for the Testbed. DCE supports client/server applications development, is a heterogeneous platform environment (including SUN, HP, and Silicon Graphics (SGI) for the Testbed, and is available from multiple vendors (including HP, Transarc, SUN, and SGI). The Testbed design is a single cell DCE architecture. Multicell security authorization and cross-cell authentication including Access Control List (ACL) facilities are not used. The Testbed makes use of the UNIX provided password authorization facilities.

Another COTS product, OODCE from HP, was selected as the object oriented encapsulation method. OODCE provides an object-oriented layer on top of DCE by providing a set of class libraries and an Interface Definition Language compiler, IDL++, to generate stub code in the C++ language. With OODCE comes the ability for the applications developer to use object oriented methodology in their client/server development. This encapsulation method also entails CSS development of custom APIs as application developer interfaces used to access OODCE and DCE functionality. Note that each CSS service provides its service specific DCE encapsulating API.

A number of CSS services are used only to establish a client/server session (i.e., find and bind from the directory service, authentication and authorization from the security service). After the client/server session has been established, the client and the server can agree to other interfaces (e.g., the DBMS client to DBMS server native protocol).

The following lists the COTS products that are configured for the Testbed platforms:

- DCE Client DCE Application version 1.0.3.a; Transarc/TAR
- DCE Application Development Kit version 1.0.3.a/1.1; Transarc/TAR
- OODCE Libraries Version 1.0.3.a; HP
- TCPWrapper Version 7.3; HP
- Tripwire Version 1.2; HP
- DCE Replica Library HP
- DCE Timeserver
- DNS Master
- DNS Slave
- OODCE Libraries (HP DCE/9000) Rev 1.2.1; for binding custom C++ code with non-OO DCE
- Kerberos Client Version CNS9691; MIT

Detailed configuration information is provided in Section 11 of this document as well as in the version description documents (VDDs) for each Testbed site.

4.2 DCI CSCI Service Descriptions and Object Models

This page intentionally left blank.

This page intentionally left blank.

4.2.1 CMIService Class Category

4.2.1.1 Overview

The EcSeCmi class was designed to be used for logging into a system (i.e. DCE login, sybase login) without hard coding a user ID and password. The user of this class may obtain a user ID and password using a seed and a random data file. Prior to using this class in application software, the user should obtain a user ID and password using the GUI program, EcSeAuthnProg that utilizes the same data file and seed. The EcSeAuthnProg uses the same library that applications may use to obtain the user ID and password. Once a user ID and password are obtained, an account needs to be established by the system administrator. The application may then use the same seed and data file in the software to obtain entry into the system. Applications using the process framework may place the seed and data file name in the process framework configuration file.

4.2.1.2 Object Model

public

EcSeCmi
myDataFile : RWFile* myLocalIP : RWCString myDatafileSize : EcTLongInt seed : EcTULongInt
ConnectAuth() srand() rand() ReadChar()

4.2.1.2.1 EcSeCmi Class

Overview:

This class will deliver an application its user ID and password based on an application key. NOTE: The application key must be a configuration parameter provided by an application configuration file.

Export Control: Public

Inheritance Relationships:

Attributes:

myDatafileSize: EcTLongInt

myDatafileSize is the read file size in bytes.

myDataFile: RWFile*

myDataFile is the file to read data from to obtain the random user IDs/passwords.

myLocalIP: RWCString

myLocalIP is the IP address of local host. IPaddress has one of the following forms. a, a.b, a.b.c a.b.c.d (e.g. 155.157.114.56).

seed: EcTULongInt

seed is used for random generator and defaults to 1.

Constructors and Destructor:

```
EcSeCmi(const EcTChar* datafile);
```

The constructor EcSeCmi opens the datafile argument for binary read using Rogue Wave. If the file is not exist, or the file is empty the constructor will set the error status and throw an exception. The constructor finds the size of the file by position and stores the file size in the private member myDatafileSize.

```
~EcSeCmi ();
```

~EcSeCmi is the destructor.

Operations:

- ConnectAuth

```
EcUtStatus ConnectAuth(const EcTLongInt applicationKey,  
RWCString& name, RWCString& password, const RWCString  
IPaddress);
```

ConnectAuth is the only public interface to the class to be called by application to obtain for the application its user ID and password based on an application key, which is a configuration parameter provided by an application configuration file.

Input - applicationKey is the seed to be used for determining the user ID and password. Output - name is the login name to be used. Output - password is the password to be used. Input - IPaddress is the Internet Address of the host from which connection will be made. This argument is optional and has a default of NULL.

- GetIPAddress

```
EcTChar* GetIPAddress();
```

The GetIPAddress method get the default IP of the connecting host.

- GetNumber

```
EcTLongInt GetNumber(const EcTChar* IPaddress);
```

The GetNumber method generates a random offset of the file pointed to by myDataFile in the range of the file size. The offset generated relates to the IP address of the connecting host. It has one of the forms a, a.b, a.b.c, or a.b.c.d

- GetString

```
EcTInt GetString(RWCString& string, const RWCString& IPaddress);
```

The GetString method reads 8 bytes alpha-numeric characters from the file starting from the specified offset.

- rand

```
EcTULongInt rand(void);
```

The rand method is the random number generator.

- ReadChar

```
EcTInt ReadChar(EcTLongInt rnum, EcTChar& ch);
```

The ReadChar method searches for an alpha-numeric character from datafile at a specified offset.

- srand

```
void srand(EcTULongInt initial_seed);
```

The srand method sets the seed, the random number generator.

4.2.2 Time Service

4.2.2.1 Overview

The Time Service Object Model provides a more detailed view of the interaction possible using the Time Service. Time Service provides operations to obtain timestamps based on Coordinated Universal Time (UTC). The Time Service also translates different timestamp formats and performs calculations on timestamps.

4.2.2.2 Context

All are expected to use the Time Service for the Testbed. The CSS Time Service will provide distributed time with millisecond resolution. Applications utilize the Time Service when they need to obtain the time in various formats. The Time Service provides APIs to perform these categories of functionality.

The CSS time service will not provide a method to set time but will provide methods to obtain the time in various formats.

Some applications may need to simulate the current time by applying a delta to the current time. The time class allows application developers to obtain the current time in various formats and optionally lets them apply a predetermined delta to those values.

4.2.2.3 Object Model

public

EcTiTimeService
<code>_delta_value : utc_t</code> <code>_delta_indicator : EcTInt</code>
<code>EcTiTimeService()</code> <code>EcTiTimeService()</code> <code>~EcTiTimeService()</code> <code>GetAscGmtTime()</code> <code>GetAscGmtTime()</code> <code>GetSecNanoTime()</code> <code>GetTimeValues()</code> <code>GetRWTime()</code> <code>GetTime()</code> <code>GmTime()</code> <code>BinToAscGmt()</code> <code>MkBinTime()</code> <code>MkBinRelTime()</code> <code>AddTime()</code> <code>SubTime()</code> <code>GetLocalZone()</code> <code>MkGMTTime()</code> <code>CmpMidTime()</code> <code>CmpIntTime()</code> <code>GetLocalTime()</code> <code>CvtStrToBin()</code> <code>ApplyDelta()</code> <code>CalculateDelta()</code> <code>CvtDeltaToBinary()</code>

4.2.2.3.1 EcTiTimeService

Overview:

This class is used to obtain the current time in various formats, and manipulate the time. There will be no capability for the programmer to set the time.

In order to provide for simulated time when an object of this class is instantiated the parameters passed to the constructor will be a string which is a name in the namespace, a string containing an absolute time or delta time, or a null string, and a flag indicating whether the string is a namespace

containing an absolute time value or a delta value, or the string is an absolute time or delta time itself.

When a NULL string is passed as a parameter in the constructor the current time will be obtained.

When a string passed as a parameter in the constructor is not NULL it is assumed to be a name in the namespace, when the delta value is equal to 1 or is equal to 2. The namespace should contain either a delta value or an absolute time value.

When the delta value in the constructor is equal to a 3, the string is assumed to be the absolute time and when the delta value is equal to 4 the string is assumed to be a delta value.

When the flag passed in the constructor indicates the time in the namespace or in the constructor is an absolute time a delta value will be calculated by subtracting the current time with the absolute time in the namespace. The delta value will be applied to the current time whenever a call is made to obtain the current time.

The absolute time in the namespace or in the constructor should be a NULL-terminated character string in the following format:

```
1995-11-04-12:10:30.000
```

where 1995 is the year - separator between year and month 11 is the month - separator between month and day 04 is the day of month - separator between day and hours 12 is the hour : separator between hours and minutes 10 is the minutes : separator between minutes and seconds 30 is the seconds . separator between seconds and thousandths of seconds 000 is the thousandths of seconds

When the flag passed in the constructor indicates the time in the namespace is a delta value the delta value will be applied to the current time whenever a call is made to obtain the current time.

The delta time in the namespace or in the constructor should be a NULL-terminated character string in the following format:

```
[+|-]dd:hh:mm:ss where [+|-] indicates plus or minus dd indicates number of days - separator between day and time hh indicates hours mm indicates minutes ss indicates seconds
```

The only time a delta will be applied is when obtaining the time (GetAscGmtTime, GetSecNanoTime, GetTimeValues, GetTime). The other methods in this class will not apply the delta.

Export Control: Public

Inheritance Relationships:

Attributes:

`_delta_indicator: EctInt`

The `_delta_indicator` indicates what to do with the delta value. 0 = Obtain current time and 1 = Add delta to current time.

`_delta_value: utc_t`

The `_delta_value` attribute is a delta value maintained as binary timestamp.

Constructors and Destructor:

```
EcTiTimeService();
```

Default constructor.

```
EcTiTimeService(EcUtStatus* status, const EcTChar* a_Time,  
EcTInt a_DeltaType);
```

Constructor.

Output - status (EcUtStatus) Input - a_Time is a string in the name space, if null delta value of zero will be used or can be the absolute/delta time string. Input - a_DeltaType is a type of delta where 1 = a_Time is absolute time (default), 2 = a_Time is delta time, 3 = a_Time is absolute time, and 4 = a_Time is delta time.

```
virtual ~EcTiTimeService();
```

Destructor.

Operations:

- AddTime

```
EcUtStatus AddTime(const utc_t& a_utc1, const utc_t& a_utc2,  
utc_t& ao_result);
```

AddTime computes the sum of two binary timestamps.

The AddTime routine adds two binary timestamps, producing a third binary timestamp whose inaccuracy is the sum of the two input inaccuracies. One or both of the input timestamps typically represents a relative (delta) time. The TDF in the first input timestamp is copied to the output. The timestamps can be two relative times or a relative time and an absolute time.

Although no error is returned, the combination absolute time + absolute time should not be used.

Input - a_utc1 is a binary timestamp. Input - a_utc2 is a binary timestamp. Output - ao_result is a binary timestamp. Return value is EcUtStatus.

- ApplyDelta

```
EcUtStatus ApplyDelta(const utc_t& a_utc, utc_t& ao_utc);
```

ApplyDelta applies the delta if necessary to the current time.

Input - a_utc is a binary timestamp. Output - ao_utc is a binary timestamp. Return value is EcUtStatus.

- BinToAscGmt

```
EcUtStatus BinToAscGmt(const utc_t& a_utc, EcTChar*  
ao_TimeString);
```

BinToAscGmt converts a binary timestamp to an ASCII GMT String.

Input - a_utc is a binary timestamp. Output - ao_TimeString[] is an ASCII GMT time string.
Return value is EcUtStatus.

- CalculateDelta

```
EcUtStatus CalculateDelta(const utc_t& a_testtime);
```

CalculateDelta will calculate a delta value using the current time and an absolute time from the namespace.

Input - a_testtime is a binary timestamp. Return value is EcUtStatus.

- CmpIntTime

```
EcUtStatus CmpIntTime(const utc_t& a_utc1, const utc_t& a_utc2,  
utc_cmptype& ao_relation);
```

CmpIntTime compares two binary timestamps or two relative binary timestamps, not ignoring inaccuracies.

The method will returns a flag indicating that the first timestamp is greater than, less than, or equal to the second timestamp.

Input - a_utc1 is a binary timestamp. Input - a_utc2 is a binary timestamp. Output - ao_relation is the relationship. Return value is EcUtStatus.

- CmpMidTime

```
EcUtStatus CmpMidTime(const utc_t& a_utc1, const utc_t& a_utc2,  
utc_cmptype& ao_relation);
```

CmpMidTime compares two binary timestamps or two relative binary timestamps, ignoring inaccuracies.

The method will returns a flag indicating that the first timestamp is greater than, less than, or equal to the second timestamp.

Input - a_utc1 is a binary timestamp. Input - a_utc2 is a binary timestamp. Output - ao_relation is the relationship. Return value is EcUtStatus.

- CvtDeltaToBinary

```
EcUtStatus CvtDeltaToBinary(const EcTChar* a_deltastring);
```

CvtDeltaToBinary will convert the delta time given as an ASCII string from the namespace into a binary timestamp.

Input - a_deltastring is the delta string. Return value is EcUtStatus.

- CvtStrToBin

```
EcUtStatus CvtStrToBin(const EcTChar* a_TimeString, utc_t&  
ao_utc);
```

CvtStrToBin converts a NULL-terminated character string that represents an absolute timestamp to a binary timestamp.

Input - a_TimeString is a character time string. Output - ao_utc is a binary timestamp. Return value is EcUtStatus.

- GetAscGmtTime

```
EcUtStatus GetAscGmtTime(RWCString& ao_TimeString);
```

GetAscGmtTime obtains current GMT Time as a RWCString.

Output - ao_TimeString is a string containing the ASCII time. Return value is EcUtStatus.

- GetAscGmtTime

```
EcUtStatus GetAscGmtTime(EcTChar* ao_TimeString);
```

GetAscGmtTime will return an ASCII string containing the current GMT/UTC time. The ASCII string will appear as follows:

```
1995-09-21-19:01:10.253I-----
```

Output - ao_TimeString[] is a string containing the ASCII time. Return value is EcUtStatus.

- GetLocalTime

```
EcUtStatus GetLocalTime(const utc_t& a_utc, tm& ao_timetm,  
EcTLongInt& ao_tns, tm& ao_inacctm, EcTLongInt& ao_ins);
```

GetLocalTime converts a binary timestamp to a "tm" structure that expresses local time.

The user's environment determines the time zone rule (details are system dependent).

If the user's environment does not specify a time zone rule, the system's rules are used (details of the rule are system dependent).

Additional returns include nanoseconds since the last second of Time and nanoseconds of inaccuracy.

Input - a_utc is a binary timestamp. Output - ao_timetm is a local time. Output - ao_tns is nanoseconds since local time. Output - ao_inacctm is the seconds of the inaccuracy. Output - ao_ins is the nanoseconds of the inaccuracy. Return value is EcUtStatus.

- GetLocalZone

```
EcUtStatus GetLocalZone(const utc_t& a_utc, EcTInt& a_tzlen,  
EcTChar* ao_tzname, EcTLongInt& ao_tdf, EcTInt& ao_isdst);
```

GetLocalZone obtains the local time zone label and offset from GMT, given the UTC.

The user's environment determines the time zone rule (details are system dependent).

If the user's environment does not specify a time zone rule, the system's rules are used (details of the rule are system dependent).

Input - a_utc is a binary timestamp. Input - a_tzlen is the length of the tzname. Output - ao_tzname[] is a character string long enough to hold the time zone label. Output - ao_tdf is a

longword with differential in seconds east of GMT. Output - ao_isdst is the value of 0 (zero) if standard time is in effect or a value of 1 if daylight savings time is in effect. Return value is EcUtStatus.

- GetRWTime

```
RWTime GetRWTime(EcUtStatus* retvalue);
```

GetRWTime will return the current system time as a RW Time object.

Output - retvalue is an EcUtStatus object. Return value is RWTime object.

- GetSecNanoTime

```
EcUtStatus GetSecNanoTime(timespec_t& ao_timesp, timespec_t& ao_inaccsp, EcTLongInt& ao_tdf);
```

GetSecNanoTime will obtain the current time in the format of timespec_t structure. The first timespec_t structure contains the time component in form of seconds and nanoseconds since the base time. The second timespec_t structure contains the inaccuracy component in the form of seconds and nanoseconds. The TDF (Time Differential Factor) returned is in the form of signed number of seconds east of GMT.

The timespec_t structure is as follows:

```
struct timespec { unsigned long tv_sec; Seconds since 00:00:00 GMT, January 1, 1970 long tv_nsec; Additional nanoseconds since tv_sec } timespec_t;
```

Output - ao_timesp is the time component in form of seconds and nanoseconds. Output - ao_inaccsp is the Inaccuracy component in the form of seconds and nanoseconds. Output - ao_tdf is the TDF component in the form of signed number of seconds east of GMT. Return value is EcUtStatus.

- GetTimeValues

```
EcUtStatus GetTimeValues(tm& ao_tm);
```

GetTimeValues will obtain the current local time in the form of a tm structure.

The tm structure has the following format:

```
struct tm { int tm_sec; Seconds (0-59) int tm_min; Minutes (0-59) int tm_hour; Hours (0-23) int tm_mday; Day of month (1-31) int tm_mon; Month of year (0-11) int tm_year; Year - 1900 int tm_wday; Day of Week (Sunday = 0) int tm_yday; Day of Year (0 - 364) int tm_isdst; Nonzero if daylight savings is in effect
```

Output - ao_tm is a tm structure. Return value is EcUtStatus.

- GetTime

```
EcUtStatus GetTime(utc_t& ao_utc);
```

GetTime obtains current binary timestamp.

Output - ao_utc is a binary timestamp. Return value is EcUtStatus.

- GmTime

```
EcUtStatus GmTime(const utc_t& a_utc, tm& ao_tm);
```

GmTime converts a binary timestamp to time values.

Input - a_utc is a binary timestamp. Output - ao_tm is a tm structure. Return value is EcUtStatus.

- MkBinRelTime

```
EcUtStatus MkBinRelTime(const reltimespec_t& a_timesp, const timespec_t& a_iaccsp, utc_t& ao_utc);
```

MkBinRelTime makes a binary relative time from timespec_t structure.

Input - a_timesp is a reltimespec_t structure. Input - a_iaccsp is a timespec_t structure. Output - ao_utc is a binary timestamp. Return value is EcUtStatus.

- MkBinTime

```
EcUtStatus MkBinTime(const timespec_t& a_timesp, const timespec_t& a_iaccsp, EcTLongInt& a_tdf, utc_t& ao_utc);
```

MkBinTime makes Binary Time from timespec_t structure.

Input - a_timesp is a timespec_t structure containing the time. Input - a_iaccsp is a timespec_t structure containing the inaccuracy. Input - a_tdf is time differential factor (TDF). Output - ao_utc is a binary timestamp. Return value is EcUtStatus.

- MkGMTTime

```
EcUtStatus MkGMTTime(const tm& a_timetm, EcTLongInt& a_tns, utc_t& ao_utc);
```

MkGMTTime converts a "tm" structure that expresses GMT or UTC to a binary timestamp.

Additional inputs include nanoseconds since the last second of Time and nanoseconds of inaccuracy.

Input - a_timetm is a tm structure. Input - a_tns is the nanoseconds since the time component. Output - ao_utc is a binary timestamp. Return value is EcUtStatus.

- SubTime

```
EcUtStatus SubTime(const utc_t& a_utc1, const utc_t& a_utc2, utc_t& ao_result);
```

SubTime computes the difference between two binary timestamps.

The two binary timestamps express either an absolute time and a relative time, two relative times, or two absolute times. The resulting timestamp is utc1 minus utc2. The inaccuracies of the two input timestamps are combined and included in the output timestamp. The TDF in the first timestamp is copied to the output.

Although no error is returned, the combination relative time - absolute time should not be used.

Input - a_utc1 is a binary timestamp. Input - a_utc2 is a binary timestamp. Output - ao_result is a binary timestamp. Return value is EcUtStatus.

4.2.3 Configuration File

4.2.3.1 Overview

The configuration file service provided by CSS consists of two classes, EcPfConfigFile and ListAndNameCollect. This service will read attributes and their values from a configuration file and creates a linked list of attributes and a linked list of the attributes values. An application may obtain the values for a particular attribute using this service. This service is used by the process framework and can be used on any configuration file for an application.

4.2.3.2 Object Model

public

EcPfConfigFile
myglobalRWSlist : RWSlistCollectables myglobalRWSlistPtr : RWSlistCollectables*
CreateLineList() CreateListOfLineLists()

public-

ListAndNameCollect
-attribName : RWCString -listP : RWSlistCollectables*
-RWDECLARE_COLLECTABLE() +ListAndNameCollect() +ListAndNameCollect() +~ListAndNameCollect() +ConvertListToString() +isEqual()

4.2.3.3 EcPfConfigFile Class

Overview:

This class is used to read the Configuration file that holds the values of attributes needed for the process Framework. For this it creates a linked list each node of which stores an attribute name and a linked list of values for this attribute given in the Configuration file.

Export Control: Public

Inheritance Relationships:

Attributes:

myglobalRWSlistPtr: RWSlistCollectables*

Pointer to the pointer of the global RWSlist.

myglobalRWSlist: RWSlistCollectables

Pointer to the global RWSlist.

Constructors and Destructor:

EcPfConfigFile(RWCString rsNewFileName, EcUtStatus* ao_status);

Constructor.

Input - rsNewFileName is the name of the Configuration File to be read. Output - ao_status is the status indicating whether the constructor failed or succeeded.

~EcPfConfigFile();

Destructor.

Operations:

- CreateLineList

ListAndNameCollect* CreateLineList(RWCString line_string, EcUtStatus* ao_status);

CreateLineList creates the list of values in the right side of the assignment in a line. It goes through checking whether the line is a comment line and hence needs to be ignored, or otherwise checks for the correct structure of the line (name = value1 value2 ..). In this case creates a collectable with the name of the attribute and the linked list of attribute values and inserts it in the main list.

Input - line_string is a RWCString. Output - ao_status is the status. Returns ListAndNameCollect* a list of values.

- CreateListOfLineLists

```
EcUtStatus CreateListOfLineLists(RWCString rsNewFileName);
```

CreateListOfLineLists creates the stream from the file name provided. Goes through the lines of that file and for each line creates a node to be inserted into the list. The node stores the name of the attribute and the list of values.

Input - rsNewFileName is the name of the configuration files to be read. Returns EcUtStatus indicating the status of the method.

- GetAttributesList

```
ListAndNameCollect* GetAttributesList(RWCString attribute,  
EcUtStatus* ao_status);
```

GetAttributesList gets all the attribute values linked list for a given attribute name.

Input - attribute is the attribute for which the values are requested. Output - ao_status is the status. Returns the linked list holding the values for an attribute name.

- GetGlobalPointer

```
RWSlistCollectables* GetGlobalPointer();
```

GetGlobalPointer obtains a global pointer to the RWSlist. Returns value of type RWSlistCollectables.

4.2.3.3.1 ListAndNameCollect Class

Overview:

This class is used to create collectable nodes whose fields are the attribute name and the linked list of the values for the respective attribute.

Export Control: Public

Inheritance Relationships:

Attributes:

```
attribName: RWCString
```

The Attribute Name that is found in the configuration file.

```
listP: RWSlistCollectables*
```

Pointer to the list of attributes in the configuration file.

Constructors and Destructor:

```
ListAndNameCollect();
```

Constructor.

```
ListAndNameCollect(RWCString rsNewName, RWSlistCollectables*  
pList, EcUtStatus* ao_status);
```

Constructor.

Input - rsNewName is the name to be added. Input - *pList is the list to be added to node. Output - ao_status is the status indicating whether the constructor failed.

```
~ListAndNameCollect();
```

Destructor.

Operations:

- ConvertListToString

```
RWCString ConvertListToString(EcUtStatus* ao_status);
```

The ConvertListToString method converts the list of values to a string.

Output - ao_status is the status indicating whether the constructor failed. Returns RWCString containing the list of values.

- GetList

```
RWSlistCollectables* GetList(EcUtStatus* ao_status);
```

The GetList method gets the list of the values for the attribute.

Output - ao_status is the status indicating whether the method failed. Returns a RWSlistCollectables pointer.

- GetName

```
RWCString& GetName(EcUtStatus* ao_status);
```

The GetName method gets the name of the attribute.

Output - ao_status is the status indicating whether the method failed. Returns the name of the attribute as a RWCString.

- isEqual

```
RWBoolean isEqual(const RWCollectable* ) const;
```

The isEqual method checks to see if the attribute is equal to one in the list.

- RWDECLARE_COLLECTABLE

```
int RWDECLARE_COLLECTABLE(ListAndNameCollect );
```

Declaration for the ListAndNameCollect RWCollectable.

- SetList

```
EcTVoid SetList(RWslstCollectables* pList, EcUtStatus*  
ao_status);
```

The SetList method sets the list of the values for the attribute.

Input - *pList is the list to be added to node. Output - ao_status is the status indicating whether the method failed.

- SetName

```
EcTVoid SetName(RWCString rsNewName, EcUtStatus* ao_status);
```

The SetName method sets the name of the attribute.

Input - rsNewName is the name to be added. Output - ao_status is the status indicating whether the method failed.

4.2.4 Process Framework

4.2.4.1 Overview

The ECS contains several infrastructure features which facilitate the implementation of client-server applications. The framework provides an extensible mechanism for ECS Client and Server applications to transparently include these infrastructure features. Therefore, their importance grows with future releases of ECS. Furthermore, the framework is used solely by ECS custom developed applications and as such is not meant for COTS applications.

The objective of the process framework is to ensure design and implementation consistency for all ECS Client and Server applications. It encapsulates implementation details of ECS infrastructure services and removes the need for programmers to rewrite common initialization code. It should be noted that the testbed only is using the client side of the process framework and the discussion in the document is limited to the client side.

This section presents the design of the framework. It can be noted that the design is capable of providing a different kind of framework for each type of ECS process presented in the classification in the object model section. As said in that section, a framework for the client will be provided only for DCE clients.

4.2.4.2 Context

A two step approach is used to develop the process framework. First, a process classification for the ECS project is developed from the client/server perspective. Then, the required capabilities are allocated at different levels of abstraction for each process type. The details of the above steps are presented below.

Figure 4.2.4.2-1 presents a first glance of the process classification for ECS application processes. A generic process can be specialized in client process and in server process. The former can be specialized in gateway client (client external to the ECS system and connected to it through a gateway) and DCE client (client in the ECS system which uses DCE as communication mechanism). A server process can be specialized in managed server process and managed server process.

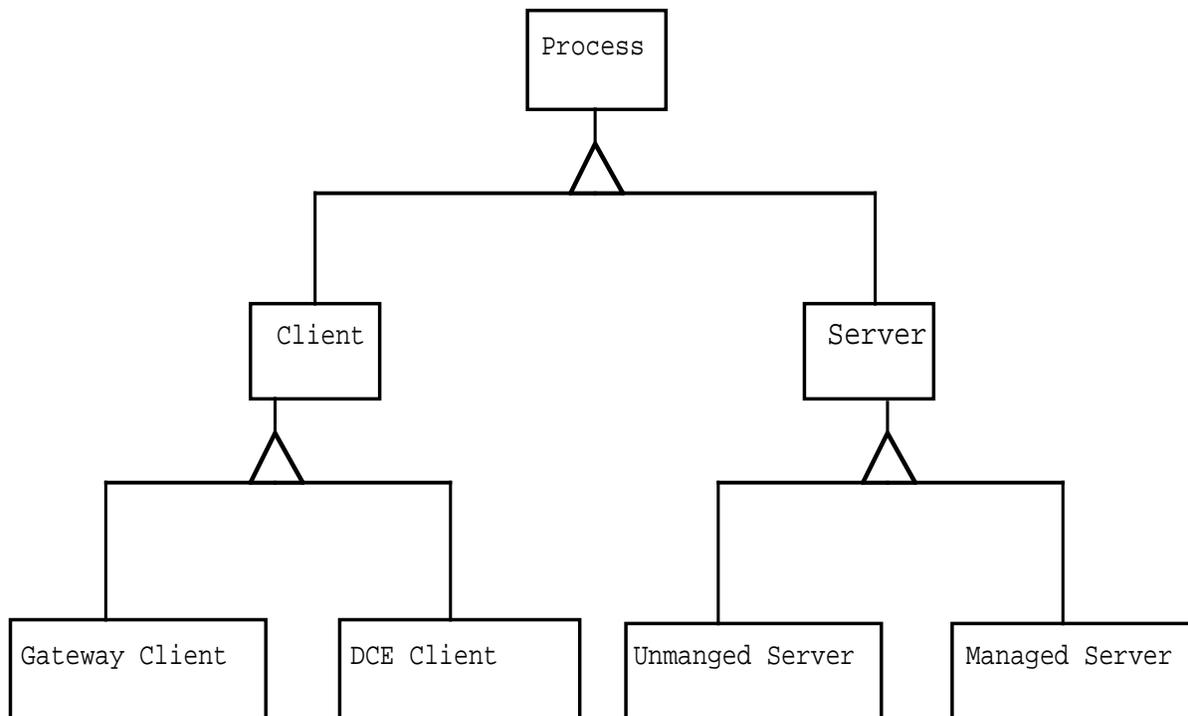


Figure 4.2.4.2-1. ECS Process Classification

In this context, the Managed Server Process is a process controlled by MSS (by the operator through HP Openview). An unmanaged server process is a process created by another process for its specific use and is out of MSS's direct control. The name must not confuse the reader. An unmanaged process is managed as well, but it is managed by its parent process instead of MSS. In the testbed release, the only types of processes that are being used are clients.

4.2.4.3 Object Model

The following figure presents the object model for the framework. The class names, in accordance with ECS policy, start with the code "Ec" and are followed by a component code, in this case we choose "Pf" for Process framework.

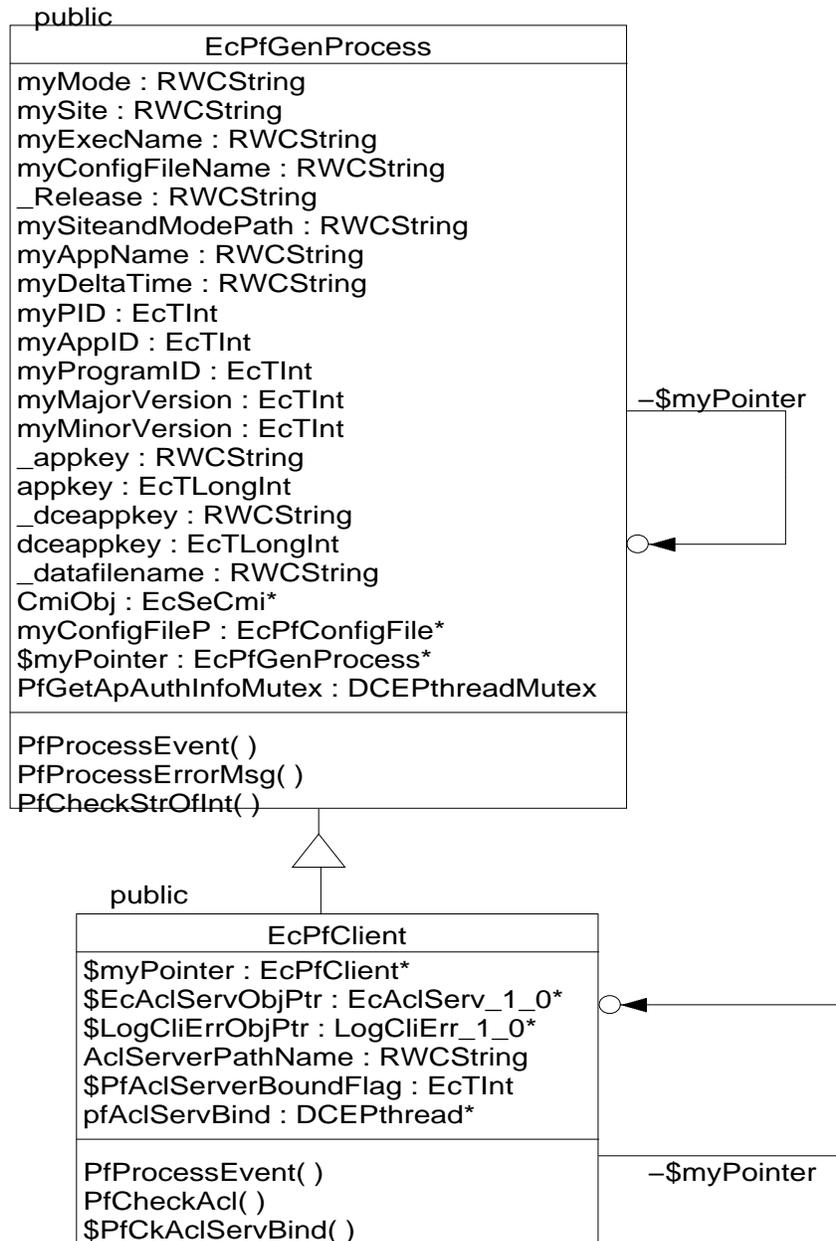
Although different type of processes need different framework functionality, the object model closely maps the process classification so that the reuse of common functionality is maximized.

Since the testbed only utilizes the client side of the process framework, the discussion here is limited to only the client features of the process framework.

The class EcPfGenProcess represents the process framework for a generic process. It has all the common functionality for all the processes. It is a class which provides a common interface to access the common methods. Moreover, every process needs to have available some basic

information about itself. This functionality can work in the same way for every process and therefore is defined in this class. On the other hand, every process needs to have Event-Error Handling capability, but its implementation must be different for client and server processes, although its interface could be the same. The method `PfProcessEvent` is therefore declared as virtual forcing the subclasses to implement this capability.

The class `EcPfClient` defines the framework for client processes. This class has essentially only two methods, the `PfProcessEvent` method and the `PfCheckAcl` method. At the moment, it doesn't seem that there are other differences with a generic process. This class is defined to satisfy a future need for specific functionality needed by client processes. In the testbed version of the `EcPfClient`, the capability to process events and to check acls does not exist. The testbed is not utilizing DCE.



4.2.4.3.1 DCEPthreadMutex Class

Overview:

Export Control: Public

Inheritance Relationships:

Attributes:

Constructors and Destructor:

Operations:

4.2.4.3.2 EcPfClient Class

Overview:

This class inherits from the EcPfGenProcess class and should be inherited from by the application. This class will provide the following: 1) create the client object to log events(errors) to the EcAclServer, 2) the client may be logged into DCE via software if the application client provides a seed in their configuration file, and 3) when the client is logged into DCE, a client object will be created for clients who which to call the method PfCheckAcl(method provided to check permissions for an acl).

Export Control: Public

Inheritance Relationships:

Inherits from EcPfGenProcess

Attributes:

AclServerPathName: RWCString

AclServerPathName is the EcAclServer path name in the CDS.

EcAclServObjPtr: EcAclServ_1_0*

EcAclServObjPtr is a pointer to the client EcAclServ object.

LogCliErrObjPtr: LogCliErr_1_0*

LogCliErrObjPtr is a pointer to the client EcAclServ object.

myPointer: EcPfClient*

myPointer is a pointer to this object.

PfAclServBind: DCEPthread*

PfAclServBind is a pointer to the PfCkAclServBind thread.

PfAclServerBoundFlag: EcTInt

PfAclServerBoundFlag indicates whether binding has occurred for the EcAclServObjPtr object.

Constructors and Destructor:

```
EcPfClient(EcTInt a_argc, EcTChar** a_argv, EcUtStatus*  
ao_status);
```

Constructor initializes inherited classes and private state.

Input - a_argc is the number of arguments on the execution line. Input - a_argv is the array of the arguments on the execution line. Output - ao_status is the status indicating whether the constructor failed.

```
virtual ~EcPfClient();
```

Destructor.

Operations:

- PfCheckAcl

```
EcUtStatus PfCheckAcl(const RWCString& a_dbname, const  
RWCString& a_aclname, const RWCString& a_permissions,  
EcTBoolean* ao_flag);
```

The PfCheckAcl checks for permissions on an acl. This method makes an RPC to the EcAclServer. The client needs to be logged into DCE to make this call, otherwise an error will occur.

Input - a_dbname is the acldbname. Input - a_aclname is the name of the acl. Input - a_permissions is the permissions to be checked. Output - ao_flag indicates if permission is granted or not granted. Returns an EcUtStatus indicating the status of the method.

- PfCkAclServBind

```
static DCEPthreadProc PfCkAclServBind(DCEPthreadParam a_param);
```

PfCkAclServBind is a thread method that checks to see if the bounding can be obtained for the EcAclServer object to interface to the server.

Input - a_param is a parameter to the thread. Returns DCEPthreadProc.

- PfGetAclServerPathName

```
RWCString PfGetAclServerPathName(EcUtStatus* ErrStatus);
```

The PfGetAclServerPathName obtains the EcAclServer path name in the CDS.

Output - ErrStatus indicates the status of the method. Returns the CDS path name to find the EcAclServer.

- PfGetAclServObjPtr

```
EcAclServ_1_0* PfGetAclServObjPtr(EcUtStatus* ErrStatus);
```

The PfGetAclServObjPtr obtains a pointer to the EcAclServObj.

Output - ErrStatus indicates the status of the method. Returns an EcAclServ_1_0 pointer.

- PfGetGlobalPtr

```
static EcPfClient* PfGetGlobalPtr();
```

The PfGetGlobalPtr method obtains the global pointer to this object.

Returns a pointer to the EcPfClient object.

- PfProcessEvent

```
virtual EcUtStatus PfProcessEvent(EcAgEvent* a_event,  
EcTAgLogType a_logtype);
```

The PfProcessEvent method flattens the EcAgEvent and make a call to the EcAclServer to log the event. If the client cannot bind to the EcAclServer the event will be written to the system log. The client does not have to be logged into DCE to make this call.

Input - a_event which is an EcAgEvent object. Input - a_logtype which indicates the log, application level or MSS level. Returns a EcUtStatus indicating the status of the method.

4.2.4.3.3 EcPfGenProcess Class

Overview:

EcPfGenProcess is a class which represents the process framework for a generic process. It has all the common functionality for all the process frameworks. It is mainly a container of attributes needed by every process. It obtains attribute values from the command line parameters or from a configuration file.

The command line argument list is passed to the constructor. It is required that the configuration file and the mode are provided from the command line. The constructor will set all attribute values and return the status.

Export Control: Public

Inheritance Relationships:

Attributes:

```
appkey: EcTLongInt
```

Application Start Number (Seed) as an integer. AppStrtNum converted to long integer.

CmiObj: EcSeCmi*

Pointer to the EcSeCmi object used for created userID and Password.

dceappkey: EcTLongInt

DCE Application Start Number (Seed) as an integer. DCEAppStrtNum converted to long integer.

myAppID: EcTInt

Application ID.

myAppName: RWCString

Name of your application.

myConfigFileName: RWCString

Name of the process framework configuration file.

myConfigFileP: EcPfConfigFile*

Pointer to the EcPfConfigFile Object used for reading from the Configuration file.

myDeltaTime: RWCString

Delta time used for simulation of time.

myExecName: RWCString

Executable name of your process.

myMajorVersion: EcTInt

Major Version number.

myMinorVersion: EcTInt

Minor Version number.

myMode: RWCString

Mode in which your process is running (ops, ts1, tr1, etc.).

myPID: EcTInt

Process identifier.

myPointer: EcPfGenProcess*

Pointer to this EcPfGenProcess object.

myProgramID: EcTInt

Program ID.

mySiteandModePath: RWCString

The site and mode path for the CDS entry.

mySite: RWCString

Site where your process is running (gsfc, larc, etc.).

PfGetApAuthInfoMutex: DCEPthreadMutex

Mutex that is locked before calling CmiObj->ConnectAuth and unlocked after the call to CmiObj->ConnectAuth. Done to prevent problems during the method if call from multiple threads at the same time.

_appkey: RWCString

Optional attribute, AppStrtNum, used for call PfGetApAuthInfo Application Start Number (Seed) as a RWCstring.

_datafilename: RWCString

Datafilename needed to instantiate EcSeCmi object.

_dceappkey: RWCString

Optional attribute, DCEAppStrtNum, used for call PfGetApAuthInfo DCE Application Start Number (Seed) as a RWCstring.

_Release: RWCString

Release of your process (A or B).

Constructors and Destructor:

```
EcPfGenProcess(EcTInt a_argc, EcTChar** a_argv, EcUtStatus*  
ao_status);
```

Constructor.

Input - a_argc is the number of arguments on the execution line. Input - a_argv is the array of the arguments on the execution line. Output - ao_status is the status indicating whether the constructor failed.

```
virtual ~EcPfGenProcess();
```

Destructor.

Operations:

- PfCheckStrOfInt

```
EcUtStatus PfCheckStrOfInt(RWCString a_IntStra);
```

The PfCheckStrOfInt method checks a string to see if the string consists only of integers.

Input - a_IntStra is the string that is to be checked. Returns a EcUtStatus indicating if the string consisted only of integers.

- PfGetApAuthInfo

```
EcUtStatus PfGetApAuthInfo(RWCString& user_name, RWCString& user_password, EcTLongInt ap_auth_info_seed);
```

PfGetApAuthInfo obtains a username and password using a seed. If the default is used for the seed argument, the seed from the configuration file is used.

Input - ap_auth_info_seed is the Application authorization information seed. Output - user_name is a RWCString containing the user name. Output - user_password is a RWCString containing the user password. Returns a EcUtStatus indicating the status of the method.

- PfGetAppID

```
EcTInt PfGetAppID(EcUtStatus* status);
```

The PfGetAppID method obtains the Application ID class attribute.

Output - status is the status indicating whether the method failed. Returns a EcTInt containing the application ID.

- PfGetAppName

```
RWCString PfGetAppName(EcUtStatus* status);
```

The PfGetAppName obtains the application name class attribute.

Output - status is the status indicating whether the method failed. Returns a RWCString containing the application name.

- PfGetConfigFileName

```
RWCString PfGetConfigFileName(EcUtStatus* status);
```

The PfGetConfigFileName method obtains the configuration file name class attribute which was set by reading the command line.

Output - status is the status indicating whether the method failed. Returns a RWCString containing the configuration file name.

- PfGetConfigFileP

```
EcPfConfigFile* PfGetConfigFileP(EcUtStatus* status);
```

The PfGetConfigFileP obtains the pointer to the EcPfConfigFile object used in the constructor.

Output - status is the status indicating whether the method failed. Returns pointer to the EcPfConfigFile object.

- PfGetDeltaTime

```
RWCString PfGetDeltaTime(EcUtStatus* status);
```

The PfGetDeltaTime method obtains the delta time class attribute.

Output - status is the status indicating whether the method failed. Returns a RWCString containing the delta time.

- PfGetExecName

```
RWCString PfGetExecName(EcUtStatus* status);
```

The PfGetExecName method obtains the executable name class attribute.

Output - status is the status indicating whether the method failed. Returns a RWCString containing the executable name.

- PfGetGlobalPtr

```
static EcPfGenProcess* PfGetGlobalPtr();
```

The PfGetGlobalPtr obtains the global pointer to this EcPfGenProcess object.

Returns a pointer to the EcPfGenProcess object.

- PfGetMajorVersion

```
EcTInt PfGetMajorVersion(EcUtStatus* status);
```

The PfGetMajorVersion method obtains the major version class attribute.

Output - status is the status indicating whether the method failed. Returns a EcTInt containing the major version number.

- PfGetMinorVersion

```
EcTInt PfGetMinorVersion(EcUtStatus* status);
```

The PfGetMinorVersion method obtains the minor version class attribute.

Output - status is the status indicating whether the method failed. Returns a EcTInt containing the minor version number.

- PfGetMode

```
RWCString PfGetMode(EcUtStatus* status);
```

The PfGetMode method obtains the mode attribute.

Output - status is the status indicating whether the method failed. Returns a RWCString containing the mode.

- PfGetPath

```
RWCString PfGetPath(EcUtStatus* status, const EcTChar*  
a_ServerName, const EcTChar* a_Site, const EcTChar* a_Mode);
```

The PfGetPath method creates a CDS path name using servername, site, mode. If defaults are taken for site and mode, values from the configuration file will be used.

Input - a_ServerName is the Server Name. Input - a_Site is the site. Input - a_Mode is the mode (tr1, ts1, ops, etc.). Output - status is the status indicating whether the method failed. Returns a RWCString containing a full CDS path.

- PfGetPID

EcTInt PfGetPID(EcUtStatus* status);

The PfGetPID method obtains the process ID.

Output - status is the status indicating whether the method failed. Returns a EcTInt containing the process ID.

- PfGetProgramID

EcTInt PfGetProgramID(EcUtStatus* status);

The PfGetProgramID method obtains the program ID class attribute.

Output - status is the status indicating whether the method failed. Returns a EcTInt containing the program ID.

- PfGetSiteandModePath

RWCString PfGetSiteandModePath(EcUtStatus* status);

The PfGetSiteandModePath method constructs the value of the attribute mySiteandModePath which has the format: ././ecs/mySite/myMode and returns it.

Output - status is the status indicating whether the method failed. Returns a RWCString containing the site mode path.

- PfGetSitePath

RWCString PfGetSitePath(EcUtStatus* status);

The PfGetSitePath constructs a path of the format: ././ecs/mySite and returns it.

Output - status is the status indicating whether the method failed. Returns a RWCString containing the site path.

- PfGetSite

RWCString PfGetSite(EcUtStatus* status);

The PfGetSite method obtains the site class attribute.

Output - status is the status indicating whether the method failed. Returns a RWCString containing the site.

- Pfget_appkey

EcTLongInt Pfget_appkey(EcUtStatus* status);

The Pfget_appkey obtains the seed value class attribute which was read from the configuration file/command line.

Output - status is the status indicating whether the method failed. Returns a EcTLongInt containing the seed value.

- Pfget_DataFileName

```
RWCString Pfget_DataFileName(EcUtStatus* ErrStatus);
```

The Pfget_DataFileName method obtains the DataFileName file name class attribute which was set by reading the command line/configuration file. DataFileName is used when creating the EcSeCmi object.

Output - status is the status indicating whether the method failed. Returns a RWCString containing the DataFileName.

- Pfget_dceappkey

```
EcTLongInt Pfget_dceappkey(EcUtStatus* status);
```

The Pfget_dceappkey obtains the seed value class attribute which was read from the configuration file/command line.

Output - status is the status indicating whether the method failed. Returns a EcTLongInt containing the seed value.

- PfProcessErrorMsg

```
EcUtStatus PfProcessErrorMsg(const EcLgErrorMsg& );
```

The PfProcessErrorMsg method processes an error message using PfProcessEvent method.

Input - EcLgErrorMsg is a structure containing the error to be logged. Returns a EcUtStatus indicating the status of the method.

- PfProcessEvent

```
virtual EcUtStatus PfProcessEvent(EcAgEvent* a_event,  
EcTAGLogType a_logtype);
```

The PfProcessEvent method for logging of events is not defined at this time in this level.

Input - a_event which is an EcAgEvent object. Input - a_logtype which indicates the log, application level or MSS level. Returns a EcUtStatus indicating the status of the method.

- PfSetAttrFromArgv

```
virtual EcUtStatus PfSetAttrFromArgv(EcTInt a_argc, EcTChar**  
a_argv);
```

The PfSetAttrFromArgv method sets the attributes from the argument line for this class, EcPfGenProcess.

Input - a_argc is the number of arguments on the execution line. Input - a_argv is the array of the arguments on the execution line. Returns a EcUtStatus indicating the status of the method.

4.2.4.4 Implementation Description

This section presents an implementation description of the Process Framework with implementation instruction and suggestion for ECS client developers since clients are the only process framework processes in the testbed.

4.2.4.4.1 Client Process

4.2.4.4.1.1 Description

Every client that wishes to be a process framework client should inherit from the EcPfClient class. Process framework clients in the testbed will not have the ability to login to DCE, log events to the EcAclServer, and check acls if needed. An example of the client using a DCE interface is shown for completion even though no DCE interfaces are contained in the testbed.

4.2.4.4.1.2 Example of a Client

Figure 4.2.4.4.1.2-1 shows an example of the connection between all the class and modules involved in a client application. The client application main program instantiates an object, in this example MyClientProc. This object inherits its framework and is connected to the class generated by the idl++ compiler. This class, in the example Widget_1_0, simulates a local call for a method which really is implemented in a server. DCE takes care of all the actions to call the distributed method in the server side.

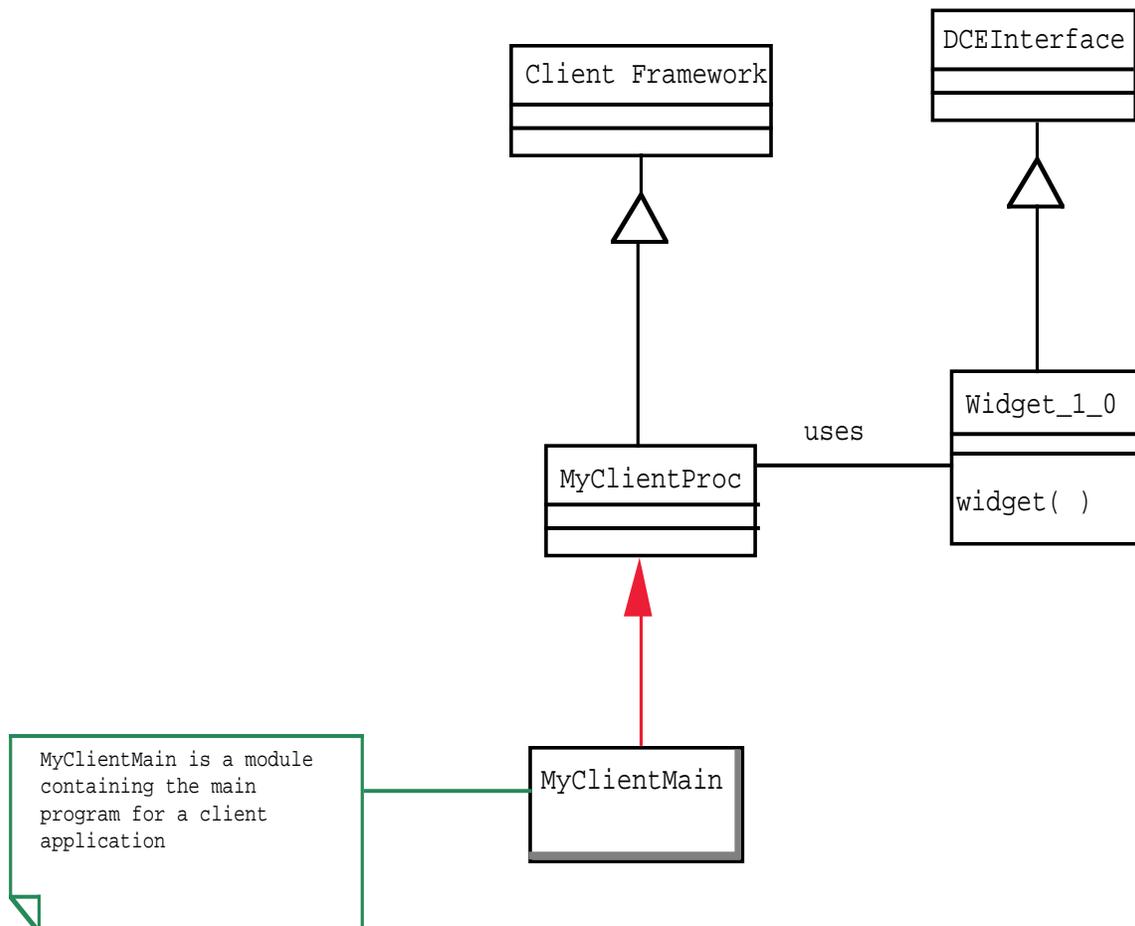


Figure 4.2.4.4.1.2-1. Example of General Object Model for Client Applications

All the DCE capability is provided in the class Widget_1_0 and its ancestor classes as DCEInterface. The client application developer doesn't use this DCE capabilities directly but only through this IDL++ generated class.

4.2.4.4.1.3 Configuration File for Process Framework Client

Set up a Process Framework configuration file. See example below:

- # The comment line should start with '#' - a # sign followed by space.
- # Please surround with space the equal sign and attribute value(s).
- # Use of a semicolon (preceded by an empty space) at the end of the line is optional.
- # Optional fields you are not using should be left blank.
- # You can give a string of different values as follows:

ProgramID = 12312

Program ID. Refer to Event Handling Standards

ApplicationID = 123

Application ID. Refer to Event Handling Standards.

Site = larc

Used in pathname for CDS to create a path for finding any server. This is used when creating the client object to find the EcAclServer for logging of event and checking acks if desired.

DeltaTime = 3600

Is for simulated time.

MajorVersion = 1

MinorVersion = 0

Major and Minor Version numbers of the delivered code.

Release = A or B

If 'A' then ././ecs/Site/Mode, since it is a single cell.

If 'B' then ././Site/ecs/Mode, since B is going to have a multicell architecture.

We need to specify in the config file.

A note: tried using B server stopped running. If we use A then clients will look in the CDS for a path with release A.

DataFileName = NameOfRandomDataFile

Name of the data file that is to be used to generate userids and passwords.

AppStrtNum = ApplicationSeedNumber

Seed that is to be used to generate user-ids and passwords.

DCEAppStrtNum = DCE login Seed Number

Seed that is to be used to generate user-id and password for logging into DCE.

5. Systems Management Subsystem (MSS) CSCIs

5.1 Introduction and Context

The MSS provides the ECS Maintenance and Operations (M&O) staff with the capability to manage the Testbed, perform network and system management services for the Testbed hardware and software resources. The software CSCIs of the MSS are composed of commercial off the shelf (COTS) software. The software contains the Management Software CSCI (MCI) and the Management Logistic Software CSCI (MLCI).

5.2 MSS Services and CSCIs

The following lists the MSS provided services and CSCI constituent elements:

Table 5.2-1 MSS Service to CSCI Mapping

Management Service	Function	CSCI
Fault Management	Monitor, detect, isolate, diagnose, and recover manually from faults within domain; largely COTS capabilities (HP OpenView) and SNMP agent software	MCI
Track non-conformance Reports (NCRs) to locate and record resources. Detects changes to approved configuration	Policy flowdown, system-wide monitoring and analysis; COTS capability provided by Distributed Defect Tracking System (DDTS)	MLCI
Site inventory data maintenance and management	Maintenance through the use of office automation tools	N/A
Site-level monitoring of spares and consumables including replenishment	Maintenance through the use of office automation tools	N/A
Site physical configuration	Configuration Management (CM) process and office automation tools	N/A
Establish and maintain PM schedules, monitor and coordinate off-site maintenance	Configuration Management (CM) process; site VDD document; COTS software DDTS; ClearCase	MLCI
Software CM of the Science Data Processing Testbed baseline	Clearcase selected for Software CM	MLCI
Maintain system wide status of change requests	COTS software DDTS	MLCI

Since HP Openview, DDTS, and Clearcase are COTS products, their designs are not modeled and documented in this “as-built” documentation. However, the operations and products’ architecture are fully documented in their vendor provided documentation as listed in Section 2 of this document.

This page intentionally left blank.