



Guide to Developing with the GCSS-AF Integration Framework

Prepared for:
Department of the Air Force
Headquarters Standard Systems Group (HQ SSG)
Maxwell Air Force Base - Gunter Annex
Montgomery, Alabama

Contract Number: **F01620-96-D-0004**

Document Number: **GCSS-REPORT-1997-0011**
Version: **3.8**
Date: **08/10/01**



-Lockheed Martin Systems Integration-Owego
1801 Route 17C
Owego, NY 13827-3998



This Page Intentionally Left Blank

DOCUMENT CHANGE HISTORY					
VERSION			CHANGE DESCRIPTION (Required for changes affecting TPM, Requirements, Configuration, Cost & Schedule)		
No.	Approval	Date	Change Doc.	Section	Narrative (of items affected)
1.0	CCB	12/15/97			Initial Release
1.1	CCB	07/02/98		3.3.2.1 5.1.6.3	SCR 69 - COM Inclusion Rules and Reuse Paradigm SCR 74 – REFERENCE Added to ANNEX B, “Integrated Development Environment”. SCR 75 – REFERENCE ADDED TO ANNEX C, “COTS Trade Study Architectural Criteria”.
2.0	CCB	03/25/99		All	SCR 060 – Incorporate Concepts and Processes for systems integration. SCR 083 – remove c4isr tables. Updates for Core and sstr. Obsolete Annexes A, B, and C and delete associated references.
2.1	CCB	06/09/99		All	SCR 097- Incorporate Customer Comments. Updates for Core Security Release.
2.2	CCB	01/07/00		All	SCR 0098 – Incorporate C4ISR Tables SCR 0111 – updates for IF release 1.2
3.0 DRAFT	IPT	07/19/00		All	CR-0009-Change Title from GCSS-AF Developer’s Guide Tab B – Design and Development to Guide to Developing With The GCSS-AF Integration Framework Major restructuring of the document
3.1 Draft	IPT	7/28/00		All	General clean-up: Expanded acronyms for clarity and corrected mapping of items in Section 4 to the IF system tree
3.2 Draft		11/30/00		All	Update for if 2.0 Release and add Section 5
3.3 Draft		02/02/01		All	Restructured Sections 4 and 5 and added Section 6-Security
3.4 Draft		02/20/01		All	Further work done on uniformity
3.5 Draft		03/09/01		All	Revision of material in Section 5.
3.6 Draft		03/16/01		All	Material added to Section 4. Updates added to Section 5.
3.7		03/23/01		All	Material added to Section 5.4. Updates added to Section 5.
3.8		08/10/01			Updates for IF 2.2 and 2.3. Updates to section 6 related to upgrading PD to v3.7 and PKI based user authentication. Remove references to PD version.



This Page Intentionally Left Blank

TABLE OF CONTENTS

1.	Introduction.....	9
1.1	Purpose.....	9
1.1.1	Overview.....	9
1.1.2	Purpose of this Document.....	11
1.2	Objectives.....	11
1.3	Scope.....	11
2.	Reference Documents.....	11
2.1	Government Documents.....	12
2.2	Applicable Standards.....	12
2.3	Contractor Documents.....	15
2.4	Product Documents.....	16
2.5	Other Documents.....	16
3.	GCSS-AF Architecture.....	17
3.1	Reference Architecture Overview.....	17
3.1.1	Integration Framework (IF).....	18
3.1.2	Application Framework.....	18
4.	Integration Framework Content and Brief Description.....	19
4.1	Infrastructure Layer.....	19
4.1.1	Platforms.....	20
4.1.2	Operating Systems and Services.....	21
4.1.3	Database Engines.....	21
4.1.4	Java Virtual Machine.....	22
4.1.5	DII COE.....	22
4.2	Integration Services Layer.....	22
4.2.1	Distributed Control.....	23
4.2.2	Messaging.....	28
4.2.3	Distributed Data Access.....	30
4.2.4	Distributed Communications Protocols.....	32
4.2.5	Network Time Protocol.....	33
4.3	Technical Services Layer.....	33
4.3.1	Services.....	34
4.3.2	Facilities.....	42
4.4	Reusable Business Component Support.....	56
4.4.1	Business Component Support.....	56
4.4.2	Web GUI Support.....	57
4.4.3	OAG Derived Components.....	57
4.4.4	Legacy Interface Component Support.....	57
4.4.5	External Interface Component Support.....	57
4.4.6	Transaction Coordination Support.....	58
4.5	Service to Product Mapping.....	58
4.6	How The IF Provides Services and Facilities.....	61
5.	Design Guidance for GCSS-AF Applications Using the IF.....	62
5.1	Mission Application Architecture Considerations.....	63
5.1.1	Reference Application Model.....	64
5.1.2	Reference Application Development Process.....	68
5.1.3	Application Integration.....	86
5.1.4	Application Security Responsibilities.....	93
5.1.5	Application and Integration Framework Integration Points.....	96
5.1.6	Development Environment Notes.....	103
5.1.7	Test Component Descriptions.....	112
5.1.8	Additional IBM Reference Material.....	114
5.2	Presentation.....	115
5.2.1	Overview.....	115

5.2.2	Service Use.....	116
5.2.3	Communications Between Layers.....	135
5.3	Business Logic.....	136
5.3.1	Overview.....	136
5.3.2	Service Use.....	137
5.3.3	Communications Between Layers.....	173
5.3.4	Design Issues Concerning Deployment.....	181
5.4	Data.....	185
5.4.1	Overview.....	185
5.4.2	Service Use.....	185
5.4.3	Communication Between Layers.....	198
5.5	Messaging Guidance.....	202
5.5.1	Message Transmission and Reception Types.....	202
5.5.2	Message Queue Configuration Considerations.....	204
5.5.3	Publish/Subscribe approach.....	214
5.5.4	Messaging from within the Application Server.....	215
5.5.5	Messaging from an Application outside the Application Server.....	216
5.6	USING THE INTEGRATION FRAMEWORK LOG FACILITY.....	217
5.6.1	Log4j Version 0.8.5b Package Overview.....	217
5.6.2	Integration Framework 2.0 Implementation Guidelines.....	223
5.6.3	A Note Regarding Tivoli Compatibility.....	226
6	Securing the Application.....	227
6.1	Overview of IF Security.....	227
6.1.1	Security Requirements.....	233
6.1.2	The ISO Security Model.....	236
6.1.3	Non-Technical Aspects of Security.....	239
6.2	Authentication.....	239
6.2.1	User Authentication.....	242
6.2.2	Application Authentication.....	246
6.3	Access Control.....	248
6.3.1	Planning for Access Control.....	250
6.3.2	Setting up ACLs.....	262
6.3.3	Setting Up Groups.....	266
6.3.4	Setting Up User Data.....	267
6.3.5	Access Control and Web Objects.....	268
6.4	Non-Repudiation.....	272
6.4.1	User Digital Signatures.....	273
6.4.2	Application Digital Signatures.....	273
6.5	Confidentiality.....	274
6.5.1	Protecting Static Data.....	274
6.6	Integrity.....	280
6.6.1	The GCSS-AF Enclave.....	280
6.6.2	The DMZ.....	282
6.6.3	Firewall Considerations.....	283
6.6.4	Multi-Level Security.....	286
6.7	Audit and Alarms.....	286
6.7.1	Logging Framework Security Events.....	287
6.8	PKI and Key Management.....	302
6.8.1	User PKI.....	302
6.8.2	Application PKI.....	302

TABLE OF FIGURES

Figure 1: GCSS-AF Document and Model Inter-relationships	10
Figure 2: GCSS-AF Reference Architecture	17
Figure 3: Integration Framework Top-Level Packages	19
Figure 4: Infrastructure Layer Packages of the Integration Framework	20
Figure 5: Integration Services Packages of the Integration Framework	23
Figure 6: Technical Services Packages of the Integration Framework	34
Figure 7: Service Packages of the IF Technical Services	34
Figure 8: High-Level Packages Under Facilities of the Integration Framework	42
Figure 9: Packages of the User Interface Common Facilities	43
Figure 10: Packages of the Information Management Common Facilities	51
Figure 11: Internationalization Package	54
Figure 12: Packages of Task Management Common Facilities	55
Figure 13: Packages of the Reusable Business Component Support	56
Figure 14: N-Tier (Mission Application) Architecture	64
Figure 15: Top-Level Mission Application Reference Design	66
Figure 16: Model-View-Controller Design Pattern	66
Figure 17: Model-View-Controller Extended Design Pattern	68
Figure 18: Reference Mission Application Development Process	70
Figure 19: Mission Application - Analysis Model View	71
Figure 20: Mission Application - Design Model View	75
Figure 21: IF Naming Conventions Relative to Application Interaction	90
Figure 22: Namespace and Naming Conventions Mapping Example	91
Figure 23: IF Routing Relative to Application Interaction	93
Figure 24: Application Security Integration Points and Considerations	94
Figure 25: Integration Framework Interface and Configuration Integration Points	98
Figure 26: Test Components UML Analysis Class Diagram	112
Figure 27: Test Component UML Design Packages	114
Figure 28: Example Servlet Hierarchy	118
Figure 29: IFServlet Source View	120
Figure 30: Add Part Control Flow Scenario	121
Figure 31: Example of IFPropertyManager Abstract Method	122
Figure 32: Example of Abstract Method Implementation of initializeServerConnection	124
Figure 33 Example of SessionInfoStruct Code	125
Figure 34: Example of PDCSessionAO Interface Code	125
Figure 35: Example of findByPrimaryKey code	130
Figure 36: Example of getInitalContext code	131
Figure 37: Example of Servlet doPostMethod Usage	132
Figure 38: Example Code taken from JSP Page	134
Figure 39: Pictorial Representation of an Enterprise JavaBean Component	140
Figure 40: Locating and creating an EJB Home	146
Figure 41: Locating a WAS-EE deployed component	147
Figure 42: Bean managed transaction	149
Figure 43: Example of host/resources/factory-finders/<ServeName>-server-scope-widened Code	153
Figure 44: BOD Structure	155
Figure 45: IFCBSecurityInfoSFigure 8: gcssafAuthenticateUser Code Example	161
Figure 46: Policy Director Object Space for PDC Test Component	164
Figure 47: Policy Director Object Space for PDC Test Component (Part A)	164
Figure 48: Policy Director Object Space for PDC Test Component (Part B)	165
Figure 49: Example IDL from PDCSessionModule of the PDC Test Component	167
Figure 50: Example IDL from com_lmfs_framework_testcomponents_requisition_OrderingTie.java of the Requisition Component Test Component	168

Figure 51: Example of Obtaining Parameters Required by the gcssAccessDecisionAllowed Method.....	170
Figure 52: Code Example of Finding IFCBSecInfo Home	170
Figure 53: Code Example of Creating IFCBSecurityInfo Instance	171
Figure 54: Code Example of Invoking the gcsafAccessDecisionAllowed Method.....	172
Figure 55: Use of the MQSeries application adaptor by a Component Broker application.....	175
Figure 56: Sending a message using TMOutbound.....	177
Figure 57: Getting the Outbound Message home	177
Figure 58: Receiving a correlated message using TMOutbound.....	179
Figure 59: Example of Trigger Monitor Application Code.....	181
Figure 60: Illustration of Workload Managed Configuration	184
Figure 61: Business, data, and Persistent Object Relationships.....	188
Figure 62: Example Transaction Emulation Across Components.....	201
Figure 63: DISA Naming Conventions Model.....	211
Figure 64: Parent/Child Hierarchy	214
Figure 65: Model of Communication Between MQSeries Applications and Component Broker CORBA Components	215
Figure 66: Example of Log Output Code Using PatternLayout.....	220
Figure 67: Compile-Time Directive Usage Example	221
Figure 68: Example Code of Compile-Time Follow up	221
Figure 69: Example Code for Assigning Foo Class to the cat Category	224
Figure 70: Example Code to Create a Sub-Category in the Class Foo.....	224
Figure 71: Example Code for Log Output Appender A1.....	225
Figure 72: Example Code for Log Output Appender A2.....	225
Figure 73: Operational Diagram: Authentication & Authorization	228
Figure 74: Example of PDCAddServlet Code	244
Figure 75: Example Queue Manager Channel Exits Code	247
Figure 76: IF Test Applications Actions.....	264
Figure 77: Example Policy Director Object Space WebSeal branch.....	272
Figure 78: GCSS-AF Enclave Model I.....	276
Figure 79: GCSS-AF Enclave Model II.....	281
Figure 80: Typical One-router, One-firewall Internet Connection Configuration	284
Figure 81: Model of Log File Generation.....	290

TABLE OF TABLES

Table 1: Service to Product Mapping: Infrastructure	58
Table 2: Service to Product Mapping: Integration Services	58
Table 3: Service to Product Mapping: Technical Services/Services	59
Table 4: Service to Product Mapping: Technical Services/Facilities	59
Table 5: Service to Product Mapping: Reusable Business Component	60
Table 6: Sample Identification of Code Template, Base Class, and Helper Class.....	62
Table 7: Presentation - Analysis Component Type to Design Component Type Mapping.....	81
Table 8: Business Logic - Analysis Component Type to Design Component Type Mapping.....	82
Table 9: Data - Analysis Component Type to Design Component Type Mapping.....	83
Table 10: Design Component Type to Unique Implementation Component Type Mapping.....	85
Table 11: Analysis Phase Development Environment Items	104
Table 12: Design Phase Development Environment Items	105
Table 13: Implementation Phase Development Environment Items	107
Table 14: Integrated Development Environment Items	108
Table 15: Test Tool Categories	111
Table 16: Servlet Property File Contents.....	123
Table 17: Servlet Development Dependencies.....	126
Table 18: IF Business Logic Component Services	137
Table 19: EJB & CORBA Server Components Security Activities.....	159
Table 20: MA Authentication Scenarios.....	160
Table 21: IF Policy Director Application Object Namespace (Recommendation).....	163
Table 22: Example Policy Director Access Control Lists (ACL)	166
Table 23: Parameters of the GCSSAF/AccessDecisionAllowed Method.....	168
Table 24: IF Data Layer Component Services.....	185
Table 25: Cardinality Example	191
Table 26: Log4j Packages	217
Table 27: Chained Priorities Examples	219
Table 28: Multiple Appender Example	223
Table 29: Message Priorities.....	224
Table 30: Authentication Matrix.....	240
Table 31: Pseudo-Supply Test ACLs	259
Table 32: Access Control List.....	264
Table 33: Policy Director Built-in Permissions.....	264
Table 34: Policy Director Object Space WebSEAL Branch.....	271
Table 35: Confidentiality Network Traffic Matrix	277
Table 36: Explanation of DISA Prioritization Scheme	293
Table 37: Recommendation of Events for IF Use Cases	294
Table 38: Application PKI	303

1. Introduction

GCSS-AF provides a component-based Reference Architecture Framework that serves as the Integration and Application Framework Layers for GCSS-AF functional capabilities consistent with the Defense Information Infrastructure Common Operating Environment (DII COE), the Joint Technical Architecture - Air Force (JTA-AF), and based on commercial open standards. The GCSS-AF Reference Architecture Framework also provides common interfaces for those functions that either directly or indirectly support Command and Control (C2) or share information with C2 Systems.

It is assumed that the reader is cognizant of Object Oriented Analysis and Design methods, the Unified Modeling Language (UML), Open Applications Group (OAG) concepts and specifications, Object Management Group (OMG) concepts and specifications, and GCSS-AF Requirements Specifications. Specific reference documents are provided in Section 1.

1.1 Purpose

1.1.1 Overview

Application Developers associated with GCSS-AF will be performing development in a new environment with new processes, techniques, and constraints. Guidance is needed to understand the overall integration environment. This document is one of an interrelated set of four primary sources of information for developing applications within GCSS-AF:

- A. Global Combat Support System - Air Force (GCSS-AF) Architecture Overview and Description
- B. Global Combat Support System - Air Force (GCSS-AF) Application Framework Developer's Guide
- C. Global Combat Support System - Air Force (GCSS-AF) Guide to Developing with the GCSS-AF Integration Framework
- D. Global Combat Support System - Air Force (GCSS-AF) Systems Solutions UML Model

In addition, there should be a Developer's Guide unique to the Business Area under development. The document and model interrelationships are depicted in Figure 1.

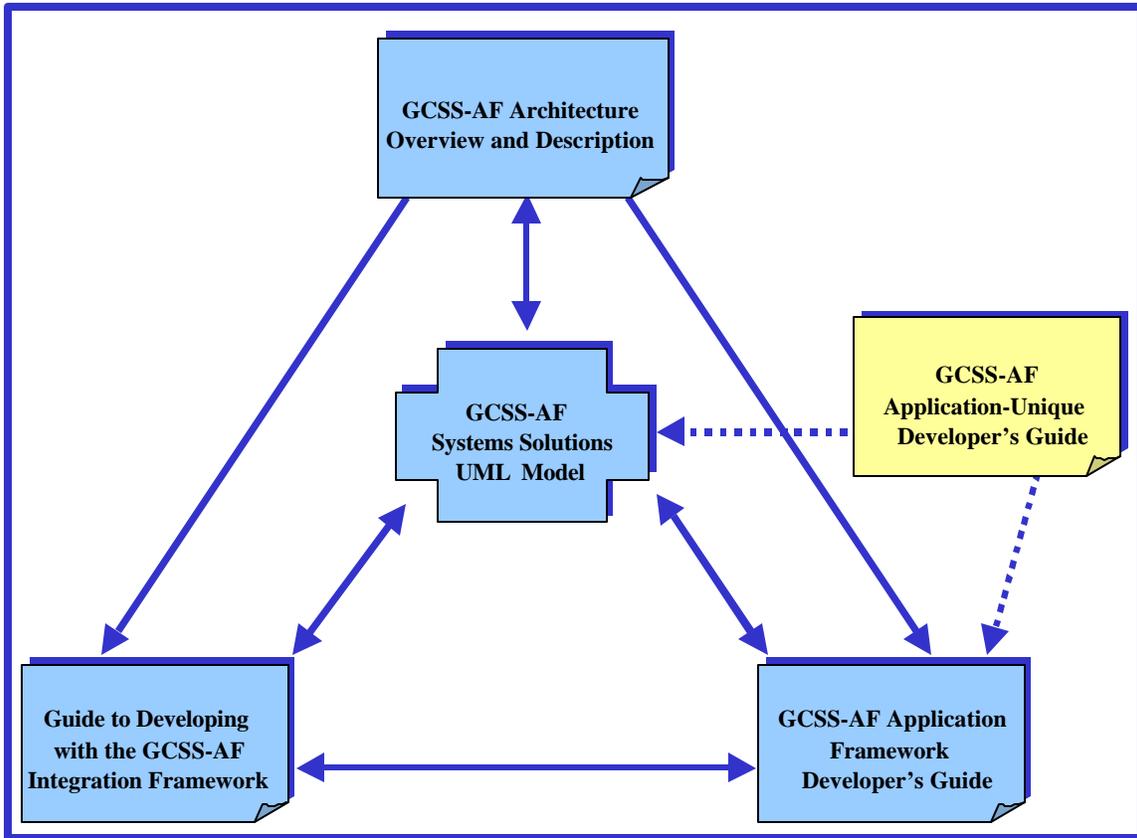


Figure 1: GCSS-AF Document and Model Inter-relationships

For the Application Developer to obtain a complete understanding of the definitions, concepts and processes, the information above should be read/used in the order above (A through C with references to D, as necessary) and sequentially within each document to build a complete understanding of the development methodology.

In addition, the Application Developer should review the following documents to understand the overall GCSS-AF Requirements as well as the specific Integration Framework requirements:

- E. Global Combat Support Systems – Air Force System Requirements Specification
- F. Global Combat Support System - Air Force (GCSS-AF) Integration Framework Enterprise Systems Management (ESM) Requirements Subsystem Specification
- G. Global Combat Support System - Air Force (GCSS-AF) Integration Framework Security Requirements Subsystem Specification
- H. Global Combat Support System - Air Force (GCSS-AF) Integration Framework Data Warehouse Services Requirements Subsystem Specification

1.1.2 Purpose of this Document

The purpose of this Guide to Developing with the GCSS-AF Integration Framework is to provide the Application Developers with information to enable them to understand and utilize the capabilities provided by the GCSS-AF Integration Framework (IF). This guide is also intended to direct developers to additional information required to develop applications employing the IF capabilities.

1.2 Objectives

The primary objective of this document is to enable the Application Developers to determine the services/facilities/capabilities of the GCSS-AF IF, which can be employed in the application and which are needed for the Target State Application Architecture.

1.3 Scope

This document is composed of the following sections:

- Section 1: This section provides an Introduction, Purpose and Scope of this document.
- Section 2: This section contains lists of Reference Documents.
- Section 3: This section contains an overview description of the GCSS-AF Architecture, including a Systems View of the Integration and Application Framework layers.
- Section 4: This section describes the GCSS-AF IF content and description.
- Section 5: This section contains Design Guidance for GCSS-AF Applications.
- Section 6: This section describes the Security solution and services provided by the GCSS-AF IF.
- Appendix A: Stand alone document GCSS-REPORT-1997-0011 Appendix A that contains a description and intended use of each helper and utility class that is delivered with the Integration Framework.

For a list of Acronyms and Glossary of Terms, reference the GCSS-AF Developer's Guide – Architecture Dictionary and Acronyms; GCSS-REPORT-1999-0100.

2. Reference Documents

The documents in this section are not all explicitly referenced in this document. However, it is important to review and use these documents as they provide pre-requisite information relevant to understanding the overall GCSS-AF.

2.1 Government Documents

AF Policy Directive 16-5; Planning, Programming, and Budgeting System, 29 July 1994
C4ISR Architecture Framework; Version 2.0; 18 December 1997

Defense Information Infrastructure (DII) Common Operating Environment (COE), Developer
Documentation Requirements; Version 2.0; 23 January 1998

Defense Information Infrastructure (DII) Common Operating Environment (COE), How to
Segment Guide.

Defense Information Infrastructure (DII) Common Operating Environment (COE), Integration
and Runtime Specification (I&RTS); Version 4.0; October 1999

Defense Information Infrastructure (DII) Common Operating Environment (COE), Office
Automation Software Requirements Specification (SRS); V3.3; 11 January 1998

Defense Information Infrastructure (DII) Common Operating Environment (COE), Security
Software Requirements Specification (SRS); Version 4.0; 20 October 1998

Defense Information Infrastructure (DII) Common Operating Environment (COE), Software
Quality Compliance Plan.

Defense Information Infrastructure (DII) Common Operating Environment (COE) User Interface
Specifications v 3.0 (incl Style Requirements of DII Compliance as Appendix I); 8 March 1998

Department of Defense (DoD) Joint Technical Architecture; Version 3.0; 15 November 1999

DoD 5200.28-STD, Department of Defense Trusted Computer System Evaluation Criteria;
December 1985 [Orange Book]

DoDD 5400.4, Provision of Information to Congress, 30 January 1978

DID DI-IPSC-81433 (MIL-STD-498), System/Subsystem Specification (SSS); 94-12-05
DII-AF Chief Architects' Office Position Paper No. CAO-002; Use of Windows '95 & '98;
06/26/98

2.2 Applicable Standards

AFI 33-129; Transmission of Information Via The Internet; 1 August 1999

AFI 33-202; Computer Security; 1 February 1999 (replaces AFSSI 5102 COMPUSEC For
Operational Systems, 23 Sept 1996)

AFI 35-119; Electronic Mail Management and Use

AFI 35-205; Air Force Security and Policy Review Program

AFM 33-229, Controlled Access Protection, 1 Nov 1999

AFMAN 33-229, Controlled Access Protection; 1 November 1997

AFSSI 5024, Volume 1, The Accreditation and Certification Process, 1 Sep 97

AFSSI 5024, Volume 2, The Certifying Official's Handbook, 1 Sep 1997

AFSSI 5024, Volume 3, Designated Approving Authority Handbook, 20 Nov 1998

AFSSI 5024, Volume 4, Type Accreditation, 10 Sep 97

AFSSI 5027, Network Security Policy, 27 Feb 1998

AFSSM 5018, Risk Analysis, 1 Feb 1997

ANSI-X3.135-1992, 1992, Information Systems - Database Language - SQL with Integrity Enhancements)

CJCS Manual 6231.03, 2 Jun 95, Manual for Employing Joint Data Communications Systems - Joint Record Data Communications

Component Object Model (COM); Microsoft Corp.

Common Object Request Broker: Architecture and Specification; Revision 2.3.1; October 1999 [OMG]

Deputy Secretary of Defense (OSD) Memorandum, Subject: Management Reform Memorandum #16 - Identifying Requirements for the Design, Development, and Implementation of a DoD Public Key Infrastructure; 08 August 1997

FIPS PUB 113, Computer Data Authentication, 30 May 1985

FIPS PUB 127-2, Standard Query Language (SQL), 2 June 1993

FIPS PUB 140-1, Security Requirements for Cryptographic Modules; 11 January 1994

FIPS PUB 161-2, Electronic Data Interchange (EDI), 29 April 1996

FIPS PUB 180-1, Secure Hash Standard, 17 April 1995

FIPS PUB 186-2, Digital Signature Standard (DSS), 27 January 2000

GAO/AIMD-00-21.3.1, Standards for Internal Control in the Federal Government, November 1999

IEEE 1003.1-1990, ISO/IEC IS 9945-1:1990, 1990, IEEE Standard for Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language]

IEEE 1003.1b-1993, 1993, IEEE Standard for Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) - Amendment 1: Real-Time Extension [C Language]

IEEE 1003.2-1994, 1994, IEEE Standard for Information Technology - Portable Operating System Interface (POSIX) - Part 2: Shell and Utilities

ISO IS 9579, Aug 1992, Generic Remote Database Access

ISO/IEC 9075:1992, Information Technology --- Database Languages --- SQL

ISO/IEC 9075-3:1995 Information Technology—Database Languages—SQL—Part 3: Call-Level Interface (SQL/CLI)

Microsoft® OLE DB 2.0 Programmer's Reference and Data Access SDK

MIL-STD-187-700, 01 Jun 92, Interoperability and Performance Standards for the Defense Information System

MIL-STD-2045-14502, Internet Transport Profile for DOD Communications

MIL-STD-2045-17503, Internet Message Transfer Profile for DOD Communications

MIL-STD-2045-17505, Internet Domain Name System (DNS) Profile for DOD Communications, 29 Jul 1994

MIL-STD-2045-17506, Internet Remote Login Profile for DOD Communications, 29 Jul 1994

MIL-STD-2045-17507, Internet Network Management Profile for DOD Communication

OMB Circular A-127, Financial Management Systems

Open Applications Group Common Middleware API Specification (OAMAS), Release 1.0

Open Applications Group Integration Specification (OAGIS), Release 6.2

Open Document Management API (ODMA), Version 2.0, September 19, 1997

OSD Memorandum, Subject: Interim Guidance for the Department of Defense (DoD) Public Key Infrastructure (PKI); 11 August 1998

OSD Memorandum, Subject: Public Key Infrastructure (PKI) Services for the Defense Information Infrastructure (DII); 19 August 1997

Programming Interface (Interface 2 & 3) Specification; (WFMC-TC-1009 - Specification V 2.0); (WFMC-TC-1013 - Naming Conventions V 1.4)

RFC 1094, Mar 89, Network File System Protocol Specification

RFC 1144, Feb 90, Serial Line Interface Protocol (SLIP) Specification

RFC 1305, Mar 92, Network Time Protocol (Version 3) – Specification

RFC 1661, Jul 94, Point-to-Point Protocol (PPP) Specification

RFC 2030, Oct 96, Simple Network Time Protocol (SNTP)

RFC 768, Aug 80, User Datagram Protocol (UDP) Specification RFC 791, Sep 81, Internet Protocol (IP)

RFC 792, Sep 81, Internet Control Message Protocol Specification

RFC 793, Sep 81, Transmission Control Protocol (TCP)

RFC 822; Aug 82, Standard for the Format of ARPA Internet Text Messages
The X/Open CAE Specification “Data Management: SQL Call-Level Interface (CLI)”
Workflow Management Coalition (WfMC) Interface 1 – Process Definition Interchange V 1.0 Beta; (WfMC-TC-1016-P)

Workflow Management Coalition (WfMC) Interface 2 - Workflow Client Application

Workflow Management Coalition (WfMC) Interface 4 – Interoperability; Abstract Specification (WFMC-TC-1012, 20-Oct-96, 1.0); Internet E-mail MIME Binding (WFMC-TC-1018, 18 September 98, 1.1)

Workflow Management Coalition (WfMC) Interface 5 - Audit Data Specification; (WFMC-TC-1015, 1-Nov-96, 1.0)

2.3 Contractor Documents

Global Combat Support System - Air Force (GCSS-AF) Architecture Overview and Description GCSS-REPORT-1997-0010.

Global Combat Support System - Air Force (GCSS-AF) Application Framework Developer’s Guide, PROJ-2000-GCSSAF-0371.

Global Combat Support System - Air Force (GCSS-AF) Guide to Developing with the GCSS-AF Integration Framework, GCSS-REPORT-1997-0011.

Global Combat Support System - Air Force (GCSS-AF) Systems Solutions UML Model

Global Combat Support System - Air Force (GCSS-AF) System Requirements Specification; GCSS-REQ-1997-0001.

Global Combat Support System - Air Force (GCSS-AF) Integration Framework Enterprise Systems Management (ESM) Requirements Subsystem Specification, GCSS-SPEC-1999-0110.

Global Combat Support System - Air Force (GCSS-AF) Integration Framework Security Requirements Subsystem Specification, GCSS-SPEC-1999-0111.

Global Combat Support System - Air Force (GCSS-AF) Integration Framework Data Warehouse Services Requirements Subsystem Specification, GCSS-SPEC-1999-0112.

Global Combat Support System - Air Force (GCSS-AF) Developer's Guide – Architecture Dictionary and Acronyms; GCSS-REPORT-1999-0100.

Global Combat Support System - Air Force (GCSS-AF) Guide to GCSS-AF Compliance; PROJ-2000-GCSSAF-0315.

Global Combat Support System – Air Force (GCSS-AF) Systems Security Policy; 10 Oct 97, HQ SSG/ENI, Maxwell Air Force Base-Gunter Annex.

Integration Framework Software Version Description Document, GCSS-REPORT-1998-0098.

2.4 Product Documents

IBM MQSeries V5.1 Documentation

IBM Policy Director V3.0 Documentation

IBM WebSphere Advanced Edition V3.0 Documentation

IBM WebSphere Enterprise Edition V3.0 Documentation

ORACLE Database Documentation

Unified Modeling Language (UML), Rational Software Corporation,
<http://www.rational.com/uml/resources/index.jttml>

2.5 Other Documents

Unified Software Development Process by Ivar Jacobson, Grady Booch, and James Rumbaugh

Component Software *Beyond Object-Oriented Programming*, Clemens Szyperski, Addison-Wesley, 1998

3. GCSS-AF Architecture

3.1 Reference Architecture Overview

The Layered System View of the Reference Architecture Model, shown in Figure 2: GCSS-AF Reference Architecture, is defined to support distributed component-based applications developed for a distributed environment. This architecture also provides a structure that will enable interfacing with the monolithic applications that exist as Legacy systems as well as Legacy client-server applications. Each layer of the Reference Architecture is built using capabilities from the layers below it as needed.

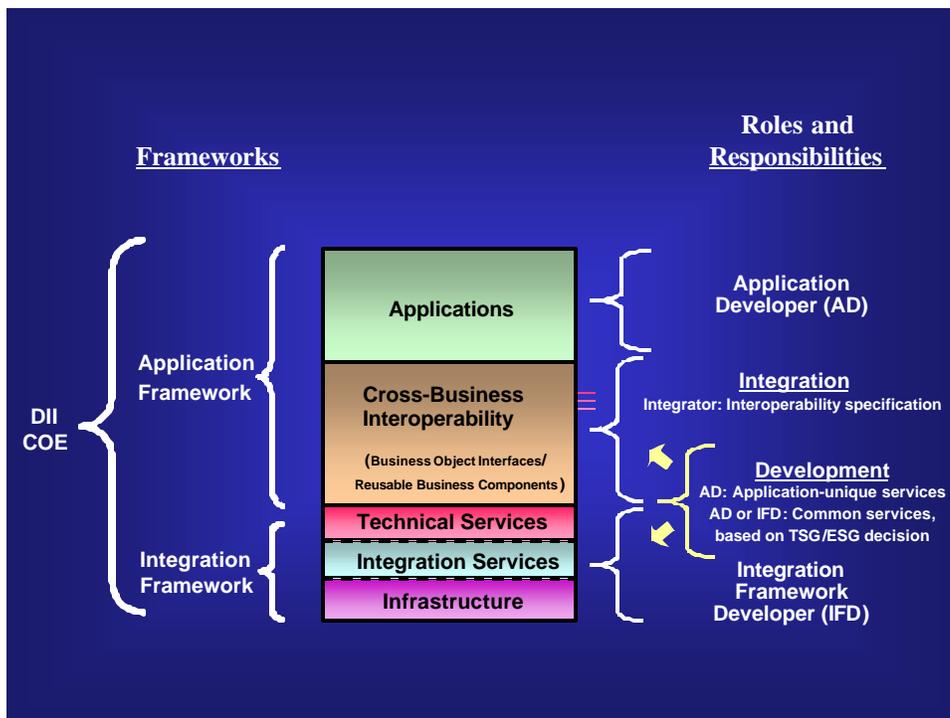


Figure 2: GCSS-AF Reference Architecture

The GCSS-AF Reference Architecture is composed of 5 layers grouped into two major frameworks. The Integration Framework supplies the facilities and services that are utilized to build and execute mission applications and are DII COE level 6 compliant (with a goal of level 7). The Application Framework provides reusable business components and the business object interfaces that implement the mechanisms for communication among business components and should also be DII COE level 6 compliant (with a goal of level 7).

A Mission Application (MA) implemented using this Reference Architecture is composed of pieces of each layer starting at the bottom, building the system by identifying and using capabilities from each layer, in turn, to satisfy the system requirements.

The Roles and Responsibilities in Figure 2: GCSS-AF Reference Architecture indicate which portion of the Reference Architecture is being/will be developed by the Integration Framework Developer (IFD), the Integrator, and the Applications Developer (AD). The IFD is responsible for all aspects of the GCSS-AF IF. The Integrator is responsible for the Interoperability Specification. The AD is responsible for all Application-unique aspects. If there are Application-unique services that are required for Cross-Business Interoperability, then the AD will develop that service. If there are common services in the Cross-Business area, then the Technical Steering Group (TSG) and Executive Steering Group (ESG) will decide on whether the IFD or AD will accomplish this development. For specific details on Methodology of GCSS-AF IF Compliance, the process for enabling GCSS-AF IF Compliance, Exception Conditions, and Application Validation and Integration, reference the [Guide to GCSS-AF Compliance](#).

3.1.1 Integration Framework (IF)

The IF provides the foundation and building blocks upon which all GCSS-AF applications should be built. The availability of this foundation enables cost and schedule savings through shared use of developed and documented facilities and services and reduces the effort required to integrate modernized and newly developed systems.

The IF is composed of the Infrastructure, Integration Services and Technical Services Layers. The facilities and services provided by these layers are being centrally developed and implemented for the Combat Support community as the GCSS-AF IF. The current and future IF services are described in Section 4. A summary of these layers of the architecture is provided below.

The lowest layer, **Infrastructure**, provides the Operating System (OS) and major system level COTS packages like the Database Engine. This layer also contains the hardware, such as clients, servers, Local Area Network (LAN)/Wide Area Network (WAN), network devices, and cabling.

Moving up to the next layer, **Integration Services** provides the communication protocols and methods such as CORBA, MOM or COM+ that are most often identified as Middleware.

The next layer is the **Technical Services** which provides distribution, presentation, data and security as well as enterprise system management services and facilities required to enable the construction and operation of component based systems.

3.1.2 Application Framework

The Application Framework is composed of the **Cross-Business Interoperability** layer and the **Applications** layer. A summary of these layers is provided below. The capabilities of these layers are implemented via individual mission applications.

The **Cross-Business Interoperability** layer defines the cross business area and business area specific functional components and the associated data model. Business areas such as financial, logistics, personnel and medical are represented here. This level also defines and implements the rules for communications, interoperability capabilities and constraints among the Business Components within the architecture.

Finally, the **Applications** layer contains the typically coarse-grained Business Components, which implement the business logic that is specific and unique to the functionality being provided to the user. These components implement what is not available in any other layer. These components shall also be DII COE compliant. This layer also encompasses the development tools required to construct and assemble components.

4. Integration Framework Content and Brief Description

This section provides a brief high level description of each of the categories and capabilities provided by the GCSS-AF IF. The intent of this section is to provide an orientation to the capabilities and services provided by the Integration Framework (IF). The specific versions of IF products relative to an IF release can be found in the Integration Framework Software Version Description Document.

It is important to recognize that the following represents both current and anticipated future IF services and capabilities. The description for each category, service, or capability will identify the release in which they appear or as a Future Capability. Any items which are Future Capabilities will be marked “**Note Future Capability:**” and in **Violet text. Bold Face font** throughout the document indicates an example of code as it needs to appear when it is implemented.

The GCSS-AF Systems Solution UML Model documents the extent to which each service is provided in the current release. Figure 3: Integration Framework Top-Level Packages illustrates the top-level system/software packages provided and/or addressed by the Integration Framework.

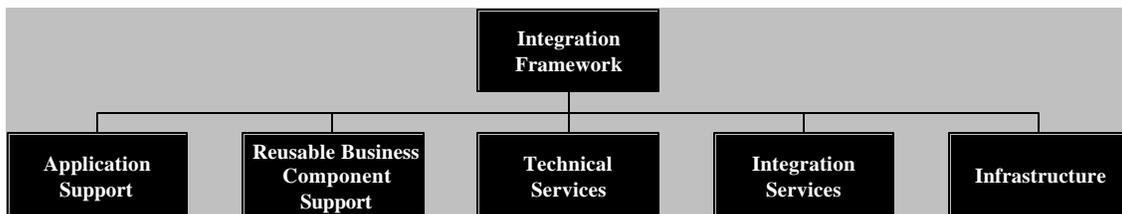


Figure 3: Integration Framework Top-Level Packages

4.1 Infrastructure Layer

The Infrastructure Layer defines the baseline hardware and software technology required to enable rapid development and deployment of scalable component-based applications within performance, cost, and reliability constraints (e.g. operating systems, database engines, workstations, servers, networking hardware and software). The products selected for this level are compliant with the standards and constraints defined in the Technical View of the GCSS-AF Reference Architecture. These products establish the hardware and system software baseline for the Reference Architecture.

Figure 4: Infrastructure Layer Packages of the Integration Framework identifies the system/software packages provided and/or addressed by the Infrastructure layer of the Integration Framework.

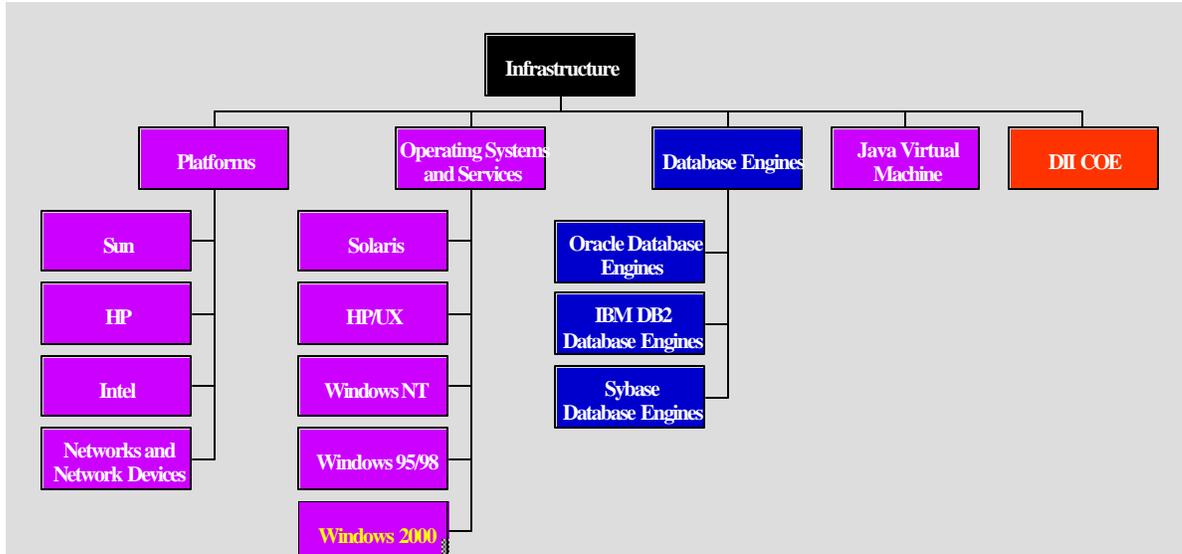


Figure 4: Infrastructure Layer Packages of the Integration Framework

4.1.1 Platforms

This category identifies the hardware platforms the IF supports. The developer shall remember to consider specifics of the platform in regards to the Operating System employed on the platform. Note that the platforms identified are, by **GCSS-AF System Specification**, restricted to those supported by the DII COE.

A Unix operating system running on an expandable multi-processor Sun or HP computer should be strongly considered where significant future expansion of processing capacity is anticipated or high processing capacities are required.

4.1.1.1 Sun Microsystems

Computers from Sun Microsystems are supported as servers.

4.1.1.2 Hewlett-Packard (HP)

HP/UX based computers employed as database servers support both the HP/UX and Windows based computers from HP. Windows-based servers from HP are in the Intel platform category.

Note: HP/UX web and application servers will not be available until a future release.

4.1.1.3 Intel (includes x86-compatible)

Intel and x86-compatible computers are supported as both client and servers. Servers require Windows NT Server while clients can use Windows NT, Windows 95, or Windows 98.

4.1.1.4 Networks and Network Devices

The processing center and supporting agency typically provide and specify the networks and network devices for that center (e.g. DISA, AFPCA, and BNCC). However, the IF specifies the use of CISCO Local Director to front end the Security Services Web Proxies to provide load balancing and fault tolerance.

4.1.2 Operating Systems and Services

This category identifies the Operating Systems (OS) and associated services supported by the IF. Also note, that the Operating Systems identified are, by **GCSS-AF System Specification**, restricted to those supported by the DII COE. The IF supports both Unix and Windows Operating Systems.

Note Future Capability: These Operating Systems provide POSIX 1003.1 APIs, which C++ components employ through developer created “wrappers,” accessed by Java components. The current IF does not provide “wrappers” for Java access.

4.1.2.1 Solaris

The IF supports the Solaris (Unix) operating system.

4.1.2.2 HP/UX

HP/UX based computers employed as database servers supporting the HP/UX (Unix) operating system.

Note Future Capability: HP/UX web and application servers will not be available until a future release.

4.1.2.3 Windows NT

The IF supports the Windows NT operating system as a server and as a User Workstation Client.

4.1.2.4 Windows 95/98

The IF supports Windows 95 and Windows 98 operating systems as User Workstation Clients.

4.1.2.5 Windows 2000

Note Future Capability: The Windows 2000 operating system is anticipated to be supported in a future IF release.

4.1.3 Database Engines

This category identifies the database “engines” which are supported or planned for support. The data access services of the products identified here are listed under Technical Services/Facilities/Information Management Common Facilities/Information Storage and Retrieval Facility/Data Access Services in **the GCSS-AF Systems Solutions UML Model**. Note that the actual database products provide both the engines and the data access services.

Oracle

The Oracle RDBMS product is supported. The Oracle product identified here is only the RDBMS the developer should not construe it as other than the RDBMS.

IBM DB2

The IF supports the IBM DB2 RDBMS product. It is included with the IBM WebSphere Advanced and Enterprise Editions.

Sybase

The IF will support the Sybase RDBMS, if required in a future release.

Other

Other databases, including Object Databases, shall fall under consideration for future releases as required.

4.1.4 Java Virtual Machine

The Java Virtual Machine (JVM) allows a Java application to run on any operating system. Java Virtual Machines operate on the supported Servlet engines and application servers. While the GCSS-AF supported Web Browsers support a JVM, for security reasons, the use of Applets is strongly discouraged.

IBMs JVM

The JVM, provided with the IBM WebSphere product line, is required on those servers running the WebSphere Servlet engine and application server.

Sun Microsystems JVM

The Sun Microsystems JVM is supported with the Netscape Web Browser.

4.1.5 DII COE

The DII COE follows a mandate that specifies installation on all servers employed for GCSS-AF as well as required by DISA on servers managed by an RSA. The primary use of the DII COE within GCSS-AF is segmentation, the use of the COE installer, and some of the security-related utilities.

4.2 Integration Services Layer

The Integration Services layer defines the communications protocols and methods used to support communications among components as well as Legacy systems (e.g., CORBA, MOM, COM+, JDBC, HTTP, IIOP). The Integration Services include Distributed Control, Messaging, Distributed Data Access Mechanisms, and Distributed Communications Protocols. Products typically referred to as Middleware fall into this category.

Figure 5: Integration Services Packages of the Integration Framework identifies the Integration Services related packages provided and/or addressed by the Integration Framework.

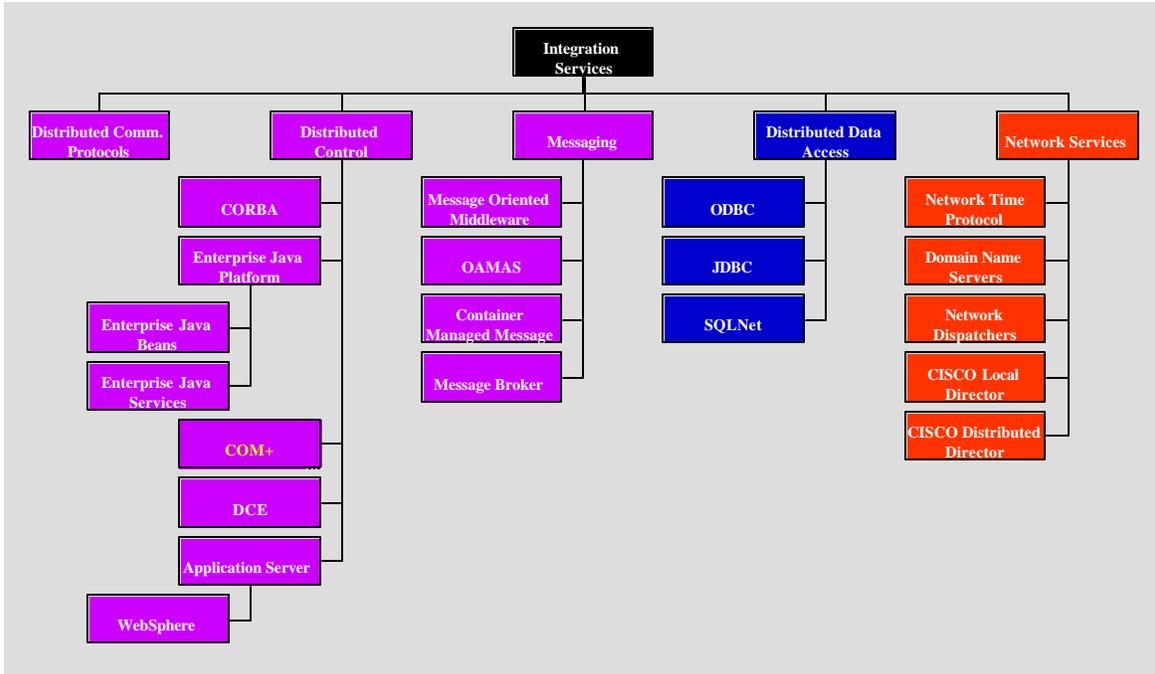


Figure 5: Integration Services Packages of the Integration Framework

4.2.1 Distributed Control

This category of services provides the basic capabilities required for distributed computing. It includes both complimentary [*CORBA and Enterprise JavaBeans (EJB) - Java 2 Platform Enterprise Edition (J2EE)*] and competing [*CORBA - EJB/J2EE and COM+*] approaches to distributed computing. For Java-based applications, the use of the EJB/J2EE is recommended. For C⁺⁺-based applications, CORBA is recommended. Note that for interoperability, the IF supported EJB/J2EE server product provides an underlying CORBA implementation. Rapidly emerging application server technologies supply both the CORBA and EJB/J2EE capabilities.

Note Future Capability: The applicability and use of COM+ has not yet been defined for GCSS-AF. However, its use may be restricted to Windows client workstations due to the unavailability of COM+ on Unix platforms. In addition, the selection of a bridge product to support interoperability between CORBA and COM+ has not been made. It will be a decision in future releases when required for GCSS-AF.

In addition, the use of *Distributed Computing Environment (DCE)* is included within the Integration Framework only because several of the products comprising the Integration Framework employ DCE now.

4.2.1.1 CORBA

CORBA is an architecture and specification for creating, distributing, and managing distributed program object in a network. It allows programs at different locations and developed by different vendors to communicate in a network through an “Interface Broker.” A consortium of vendors through the Object Management Group developed CORBA.

The essential concept in CORBA is the *Object Request Broker* (ORB). ORB support in a network of clients and servers on different computers means that a client program (which may itself be an object) can request services from a server program or object. It does not have to understand where the server is in a distributed network or what the interface to the server program looks like. To make requests or return replies between the ORBs, programs use the *Internet Inter-ORB Protocol* (Internet Inter-ORB Protocol) that maps requests and replies to the Internet’s Transmission Control Protocol layer in each computer.

As supported within the GCSS-AF enterprise, CORBA is intended for distributed computing *within* an application (or business component) and for interaction with IF services and facilities. Use of CORBA as the communications mechanism *between* applications, especially those located at different processing centers is discouraged due to significant firewall issues. In addition, (synchronous) CORBA invocations across a Wide Area Network bring along significant latency and reliability issues. Use of Messaging Services mitigates these issues. The CORBA implementation supported by the current IF is the IBM ORB included with the IBM WebSphere Enterprise Edition (Component Broker) product. This generally corresponds to the OMG CORBA 2.3 version.

To obtain more detail on OMG CORBA access the OMG web site at <http://www.omg.org>. For specific, CORBA and services implementation details refer to the associated Application Server vendor web sites identified in Section 4.2.1.5 Application Server and subsections.

4.2.1.2 Java 2 Platform, Enterprise Edition

Enterprise JavaBeans[™] (EJB) technology defines a model for the development and deployment of reusable Java[™] server components. Components are pre-developed pieces of application code that the developer assembles into a working application system. Java technology currently has a component model called JavaBeans[™], which supports reusable development components. The EJB architecture logically extends the JavaBeans component model to support server components. Enterprise Java is the preferred architecture for new and re-engineered applications.

Enterprise JavaBeans

This category encompasses Entity Beans, Session Beans, Servlets, and *Java Server Pages* (JSPs). The Enterprise JavaBeans API defines a server component model that provides portability for application servers and implements automatic services on behalf of the application components. The Java Servlets and Java Server Pages APIs support dynamic HTML generation and management for browser clients. The API implementations supported by the current IF are those included with the IBM WebSphere product.

Enterprise Java Services

The Enterprise Java platform consists of several standard Java *Application Programming Interfaces* (APIs) that provide access to a core set of enterprise-class infrastructure services. These include Distributed Communication services, Naming and Directory services, Transaction services, Messaging services, Data Access and Persistence services, and Resource-sharing services. The Enterprise Java APIs provide a common platform and vendor neutral interface to the underlying infrastructure services, regardless of the actual implementation. The API implementations supported by the current IF are those included with the IBM WebSphere product.

The current version of the Java 2-Enterprise Edition (J2EE) supported by the IF is the J2EE 1.0. More detailed information on the J2EE can be obtained through the Sun Microsystems web site at <http://java.sun.com>. For specific, J2EE and services implementation details refer to the associated Application Server vendor web sites identified in Section 4.2.1.5 Application Server and subsections.

4.2.1.3 COM+

Note Future Capability: COM+ (Component Object Model +) is in essence Microsoft's Distributed Component Architecture. COM+ is an extension of DCOM with "lightweight" object models for utilization by a Java virtual machine. Additionally, COM+ will include standard services such as security and transactions. COM+ is scheduled for release with Windows 2000. As previously identified, the use of COM+ may well be restricted to use within a client workstation.

4.2.1.4 Distributed Computing Environment (DCE)

The use of DCE in the GCSS-AF enterprise is limited to that required by IF supported IBM WebSphere Application Server and IBM Policy Director security products. It is *not* for use by GCSS-AF applications. The required DCE is bundled with both the IBM WebSphere Application Server and IBM Policy Director security products.

In network computing, DCE, developed by the Open Software, is an industry-standard software technology for setting up and managing computing and data exchange in a system of distributed computers. DCE is part of a larger network of computing systems that include different size servers scattered geographically. DCE uses the client/server model. Much of DCE setup requires the preparation of distributed directories so that DCE applications and related data can be located while they are in use.

4.2.1.5 Application Server

An **Application Server** is, by definition, "a computer server that serves applications." More precisely, an application server serves up application services. The main purpose of an Application Server is to reduce the workload of applications by taking over the responsibility of mundane activities involved in executing the application and making the application's services available to external modules in a reliable manner.

An Application Server is a computer program that resides on a server in a distributed network and whose main function is to provide the business logic for an application program. An Application Server provides a customizable and flexible execution environment for hosting business logic components, thus providing distributed services and integrity for application execution.

An Application Server provides an environment for developing scalable, multi-tier applications. Application servers integrate databases, object persistence, Java, Enterprise JavaBeans, C++, and standard CORBA services such as Transactions, Security, Events, Concurrency and Locking. Many also provide support for Load Balancing and Fault Recovery.

For GCSS-AF, application server products will be qualified at the request of and under the direction of the GCSS-AF Enterprise Authority. The subsection that follows provides a brief description of the current qualified application server products.

4.2.1.5.1 IBM WebSphere Application Server

IBM WebSphere Application Server is an e-business application deployment environment built on open standards-based technology. It is the cornerstone of WebSphere application offerings and services. The Standard Edition lets the developer use Java Servlets, Java Server Pages, and XML to quickly transform static web sites into vital sources of dynamic web content. The Advanced Edition is a high-performance EJB server for implementing EJB components that incorporate business logic; it includes the Standard Edition. The Enterprise Edition integrates EJB and CORBA components to build high-transaction, high-volume e-business applications logic; it includes the Standard Edition and Advanced Edition. In order to address the GCSS-AF Enterprise requirements, the Enterprise Edition is required for the GCSS-AF Enterprise.

WebSphere features at a glance include:

Standard Edition

The features of Standard Edition for Web developers and content authors includes:

Enhanced Java, leveraging Java 2 Software Development Kit V1.2.2 across all supported operating systems

Support for JavaServer™ Pages, including:

- Support for specifications .91 and 1.0
- Extended tagging support for queries and connection management
- An XML-compliant DTD for JSPs

Support for the Java™ Servlet 2.1 specification including a graphical interface, automatic user session management and user state management

High speed pooled database access using JDBC for DB2® Universal Database™ and Oracle XML server tools, including a parser and data transformation tools

XSL support

V3.5 supports Windows NT, Windows 2000, Solaris, AIX, AS/400, and HP-UX.

IBM HTTP server, based on Apache Web Server, including:

- An administration GUI
- Support for LDAP and SNMP connectivity

Integration with IBM VisualAge® for Java to help reduce development time by allowing developers to remotely test and debug Web-based applications
Tivoli Ready Modules

Advanced Edition

The features of Advanced Edition for Web application programmers, provides all the features of the Standard Edition, plus:

Integration with Lotus Domino, IBM Visual Age for Java and IBM WebSphere Commerce Suite

Full support for the Enterprise JavaBeans™ (EJB) 1.0 specification, including both SessionBeans and EntityBeans (Container-Managed and Bean-Managed Persistence)

Deployment support for EJBs, Java Servlets and JSPs with performance and scale improvements, including:

- Application-level Partitioning
- Load Balancing

V3.5 supports Windows NT, Windows 2000, Solaris, AIX, AS/400 and HP-UX.

IBM LDAP Directory, which can be optionally installed

A DB2 server is automatically installed as part of the runtime environment

Support for Distributed Transactions and Transaction Processing

Management and security controls, including:

- User and group level setup
- Method level policy and control

Enterprise Edition

The features of Enterprise Edition for enterprise e-business application developers and architects, includes all the features of the Advanced Edition, plus:

Full distributed object and business process integration capabilities

IBMs transactional application environment integration (from TXSeries™)

Complete Object Distribution and Persistence (from Component Broker)

Support for MQSeries®

Complete Component Backup and Restore support

XML-based team development functions

Integrated Encina application development kit

V3.5 supports Windows NT, Solaris, and AIX.

For more detail on the IBM WebSphere product suite, refer to IBMs on-line documentation available through the <http://www.ibm.com> web site. In addition, to obtain IBM Redbooks on the subject, access the web site at <http://www.redbooks.ibm.com>.

4.2.2 Messaging

Enterprise Messaging is an essential tool for building Enterprise Applications. Messaging provides application-programming services that applications can use to communicate with each other using messages and queues. It provides guaranteed, once-only delivery of messages. Using asynchronous Messaging allows separation of application programs, such that the program sending a message can continue processing without having to wait for a reply from the receiver. In addition to the point-to-point *send/receive* capability, messaging also supports the *publish/subscribe* Messaging paradigm. Messaging is recommended as the communications mechanism between applications, especially those located at different processing centers. Developers shall consider the asynchronous nature of Messaging when designing an application, particularly as it relates to transactions.

The Integration Framework provides multiple mechanisms to send and receive messages. The mechanism the IF uses is dependent upon the nature of the application's implementation:

Application server based applications should employ Container-Managed Messaging. Stand-alone applications written in either Java or C++ should use the Open Application Middleware API Specification (OAMAS).

The use of JMS is pending for a future Integration Framework release.

Note that while the native MQSeries API is exposed to applications, the use of this API is highly discouraged as the higher-level abstracted interfaces identified above serve to simplify the use of messaging as well as providing (to the application) a high degree of independence from the underlying Messaging implementation. Exceptions to this for Legacy systems interfaces will fall under evaluation on a case-by-case basis.

Firewall issues have been addressed for the Integrated Framework supported Message Oriented Middleware (MOM) COTS product.

4.2.2.1 Message Oriented Middleware

MOM is the accepted term used to categorize existing Middleware products that implement the Messaging described in the above section. These products may have more than their native APIs built on top of the product (e.g. Java Messaging Service - JMS).

MQSeries (IBM)

The MOM product supported by the IF. In addition to the guaranteed delivery of messages, MQSeries supports Transactional Messaging, which means that operations on messages can be grouped into 'units of work'. A unit of work is either committed in its entirety, or backed-out, so that it is as if none of the operations took place. MQSeries can also coordinate units of Messaging work with other Transactional work, like database updates, so that message data and database data remain completely in-sync at all times. MQSeries provide a native API (MQI), a Java Messaging Service API, and an Application Messaging Interface (AMI).

For more detail on the IBM MQSeries product, refer to IBMs on-line documentation available through the <http://www.ibm.com> web site. In addition, to obtain IBM Redbooks on the subject, access the web site at <http://www.redbooks.ibm.com>.

4.2.2.2 Container-Managed Messaging

Container-based messaging is analogous to that of container managed persistence in that the container provides the underlying services to business object developers to:

Treat messages like business data and not specifically a message to provide (nearly) the same view of messages to a business object as any other data object.

Move messages on and off queues.

Manage the movement of messages between business logic and queues within the transaction configured for the container.

As an application server provides and manages containers, the capabilities the container provides are currently specific to the container as supplied by the application server vendor. A standard such as that specified by the J2EE is not yet available for container-managed messaging. However, proposals for Container-Based Messaging are now being made for the J2EE.

For more detail on the MQSeries Adapter provided by the IBM WebSphere product, refer to IBMs on-line documentation available through the <http://www.ibm.com> web site. In addition, to obtain IBM Redbooks on the subject, access the web site at <http://www.redbooks.ibm.com>. Note that to minimize the repeatable code needed to employ the *publish/subscribe* capability from within WebSphere (application) components, the Integration Framework provides additional classes to perform the necessary operations. These descriptions appear in the section 4.2.1.5.1 IBM WebSphere Application Server

4.2.2.3 Open Application Middleware API Specification (OAMAS)

The OAMAS is a high level API that greatly simplifies programming for application messaging and *publish/subscribe*. The *Open Application Group Inc.* (OAGI) has now adopted the IBM specified and developed *Application Middleware Interface* (AMI) API for the OAMAS. The IF supported OAMAS (*AMI from IBM*) provides bindings for standard programming languages including Java, C, and C++. By providing a high level of abstraction and moving Message-handling logic from the application into the Middleware, using the OAMAS reduces the amount of written code for new applications. Connectivity code specifies a service and a policy that the IF will use when *sending* or *receiving* messages. OAMAS is currently the recommended API for use by applications and components. Note that the current OAMAS 1.0 release includes only the AMI procedural interface; the next release of OAMAS will add the AMI supported object interfaces. Currently, the full AMI is available to the developer with the assumption that it will be included in the next OAMAS release.

The OAMAS specification is available from the Open Application Group's web site at <http://www.openapplications.org>.

4.2.2.4 Java Messaging Service (JMS)

The JMS API improves programmer productivity by defining a common set of messaging concepts and programming strategies that will be supported by all JMS technology-compliant messaging systems. JMS is one of the standards APIs defined for Enterprise Java. The JMS provided with the IF is included with the IBM WebSphere product.

Note Future Capability: As the EJB specification evolves to include adequate support for events (including asynchronous messages), the benefits of using JMS should increase significantly. While the AMI API is currently identified as the preferred GCSS-AF messaging interface, investigation is on going as to the use of Java Messaging Service by Java applications; this capability may be made available in a future release of the IF.

The JMS specification, as well as additional information, is available through the Sun Microsystems web site at <http://java.sun.com>. For specific JMS implementation details refer to the associated Application Server vendor web sites identified in Section 4.2.1.5 Application Server and subsections.

4.2.2.5 CORBA 3.0

Note Future Capability: CORBA 3.0 Asynchronous Messaging and Quality of Service Control - This new CORBA 3.0 Specification along with availability of products which implement the specification is being monitored. It is not supported by the IF but will be considered as products become available. The CORBA 3.0 Messaging Specification defines a number of asynchronous and time-independent invocation modes for CORBA. The specification allows both static and dynamic invocations to use every mode. Asynchronous invocations results may be retrieved by either polling or callback, with the choice made by the form used by the client in the original invocation. Policies allow control of Quality of Service of invocations. Clients and objects may control ordering (by time, priority, or deadline); set priority, deadlines, and time-to-live; set a start time and end time for time-sensitive invocations, and control routing policy and network routing hop count.

More detail information on the OMG CORBA 3.0 specification and it is asynchronous messaging can be obtained through the OMG web site at <http://www.omg.org>.

4.2.3 Distributed Data Access

The Integration Services layer provides the capability for components to access databases that are distributed across the enterprise. These are all based on industry standards and are widely supported.

4.2.3.1 Open Database Connectivity (ODBC)

Open Database Connectivity is a Microsoft established standard that enables software developers to create applications that can work with a number of SQL-based data sources. ODBC is a C-level application-programming interface (API) for SQL data stores and provides a common interface for accessing heterogeneous SQL databases. ODBC is based on SQL as a standard for

accessing data. ODBC's consistent interface provides maximum interoperability: A single application can access different database management systems (DBMSs) through a common set of code. ODBC is meant for use by C++ components only; Java components should use JDBC. This enables a developer to build and distribute a client/server application without targeting a specific DBMS or having to know specific details of various back-end data stores. When an application needs to get data from a data store, the application sends a SQL statement to the ODBC Driver Manager, which then loads the ODBC drivers, required to talk to the data. The driver then translates the SQL sent by the application into the SQL used by the DBMS and sends it to the back-end database. The DBMS retrieves the data and passes it back to the application through the driver and the Driver Manager.

Developers can code directly to the ODBC API by declaring various functions and then using them to connect, send SQL statements, retrieve results, get errors, disconnect, and so on. However, it is difficult and involves a lot of code. Because of this, the developers use ADO, RDO, and DAO more often to access ODBC data from applications.

ODBC has been the data access standard since 1992 and has played a very important role in enabling client/server applications. There are more than 170 ODBC drivers available. With well-written drivers, ODBC performance is excellent. In the short and medium term, ODBC is the best way for C++ programs to access a broad range of relational data due to the high number of available drivers.

The ODBC specification as well as additional information is available from the Microsoft web site at <http://www.microsoft.com>.

4.2.3.2 Java Database Connectivity (JDBC)

The JDBC™ API provides universal data access from the Java™ programming language. Using the JDBC 2.0 API, a user can access virtually any data source, from relational databases to spreadsheets and flat files. JDBC technology also provides a common base to build tools and alternate interfaces. JDBC drivers supported by the IF are the IBM WebSphere provided JDBC driver for DB2 and the Oracle JDBC driver for the Oracle RDBMS.

Note Future Capability: If, in the future, the Sybase RDBMS is required, a JDBC driver for it will be supported.

The JDBC™ specification as well as additional information is available through the Sun Microsystems web site at <http://java.sun.com>. For specific JDBC™ implementation details refer to the associated Application Server vendor web sites identified in Section 4.2.1.5, Application Server and subsections and/or the database vendor web sites identified in Application Server

4.2.3.3 Net8 / SQLNet

Note Future Capability: Net8 (formerly known as SQLNet) is the foundation of Oracle's family of networking features, providing an Enterprise-wide data access solution for heterogeneous, Distributed Computing Environments. It enables client-server and

server-server communications across any network, allowing Oracle databases and their applications to reside on different computers and communicate as peer applications. Net8 eases the complexities of network configuration and management, maximizes performance, and improves network diagnostic capabilities; while at the same time introducing distributed access for Java-based applications. Note that in the current release, while Net8 is available as part of the Oracle installation, it is not supported as no actual testing of Net8 has been undertaken. Support of Net8 could be provided if a valid need is identified to the TSG. However as Net8 is a proprietary access mechanism, its use is discouraged and use of the open standards based Integrated Framework supported data access products should be employed.

The Net8/SQLNet specification as well as additional information is available from the Oracle web site at <http://www.oracle.com>.

4.2.4 Distributed Communications Protocols

This category of IF services provide the underlying communication protocols upon which implements the higher level services and Middleware. They are typically provided either by the Operating System supplier or with a particular Middleware product such as IIOP with the ORB product or RMI with the Enterprise Java provider.

TCP/IP

The Operating System vendor provides.

UDP

The Operating System vendor provides

FTP

The Operating System vendor provides

HTTP

Hypertext Transfer Protocol (HTTP) is the set of rules for exchanging files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. Relative to the TCP/IP suite of protocols (which are the basis for information exchange on the Internet), HTTP is an application protocol. *HTTP is provided by the products that utilize it (e.g. Web Browser, Web Server, Web proxy, XML parser, etc.).*

IIOP

The CORBA specified protocol over TCP/IP is as provided as part the CORBA product. Products that utilize IIOP provide it (e.g. CORBA, Java using RMI over IIOP etc.). For the current release, this is the WebSphere (Enterprise Edition - Component Broker) product.

RMI

Remote Method Invocation (RMI) is the Java version of what is generally known as a *Remote Procedure Call (RPC)*, but with the ability to pass one or more objects along with the request. The IF specifies that RMI over IIOP is required. RMI is as supplied by the EJB vendor. For the

current release, RMI is included in the WebSphere (Enterprise Edition - Component Broker) product.

RPC

Remote Procedure Call (RPC) is a protocol that one program can use to request a service from a program located in another computer in a network without having to understand network details. The RPC provided is the Open Software Foundations' *Distributed Computing Environment* (DCE) implementation. As with the DCE described above, the use of RPCs on GCSS-AF is limited to that required by IF supported Security COTS product (IBM Policy Director); it is **not** for use by GCSS-AF applications. The required RPC/DCE is bundled with the IBM Policy Director security COTS product.

4.2.5 Network Time Protocol

Network Time Protocol (NTP), now an Internet standard, is a protocol that synchronizes computer clock times in a network of computers. In common with similar protocols, NTP uses Coordinated Universal Time to synchronize computer clock times to a millisecond, and sometimes to a fraction of a millisecond.

Accurate time across a network is important for many reasons; even small fractions of a second can cause problems. For example, distributed procedures depend on coordinated times to ensure that the computer follows proper sequences. Security mechanisms depend on coordinated times across the network. File system updates carried out by a number of computers also depend on synchronized clock times.

For GCSS-AF, the Integration Framework has specified that it will obtain UTC time through the *Global Positioning System* (GPS). DISA, at the RSAs, are to provide computers designated as *Primary Time Servers* outfitted with GPS receivers and using as NTP to synchronize the clock times of GCSS-AF computers. The term NTP applies to both the protocol and the client/server programs that run on computers to be time synchronized. GCSS-AF computers will be equipped with an NTP client to initiate a time request exchange with the timeserver. DISA is responsible for equipping fielded GCSS-AF with the client NTP as well as integrating it with the NTP timeserver.

The NTP specification as well as additional information can be obtained through the University of Delaware web site at <http://www.eecis.udel.edu/~ntp>.

4.3 Technical Services Layer

The Technical Services Layer defines the services and facilities (Figure 6: Technical Services Packages of the Integration Framework) of the Integration Framework. They provide the distribution, presentation; security, data, and enterprise system management capabilities in support of component based systems. Examples of these are Naming, Event, Persistence, Transaction, Licensing, & Trader services as well as User Interface, System Management, and Internationalization facilities, and a Metadata Repository).

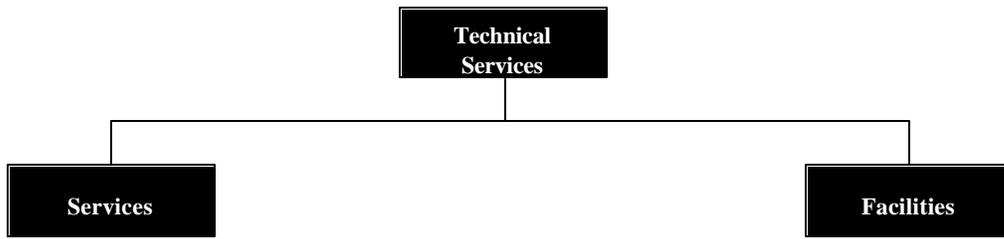


Figure 6: Technical Services Packages of the Integration Framework

4.3.1 Services

Many Distributed Object programs use these domain-independent interfaces. These components standardize the life-cycle management of objects. The IF provides interfaces to objects, to control access to objects, to keep track of relocated objects, and to control the relationship between styles of objects (class management). These services are illustrated in

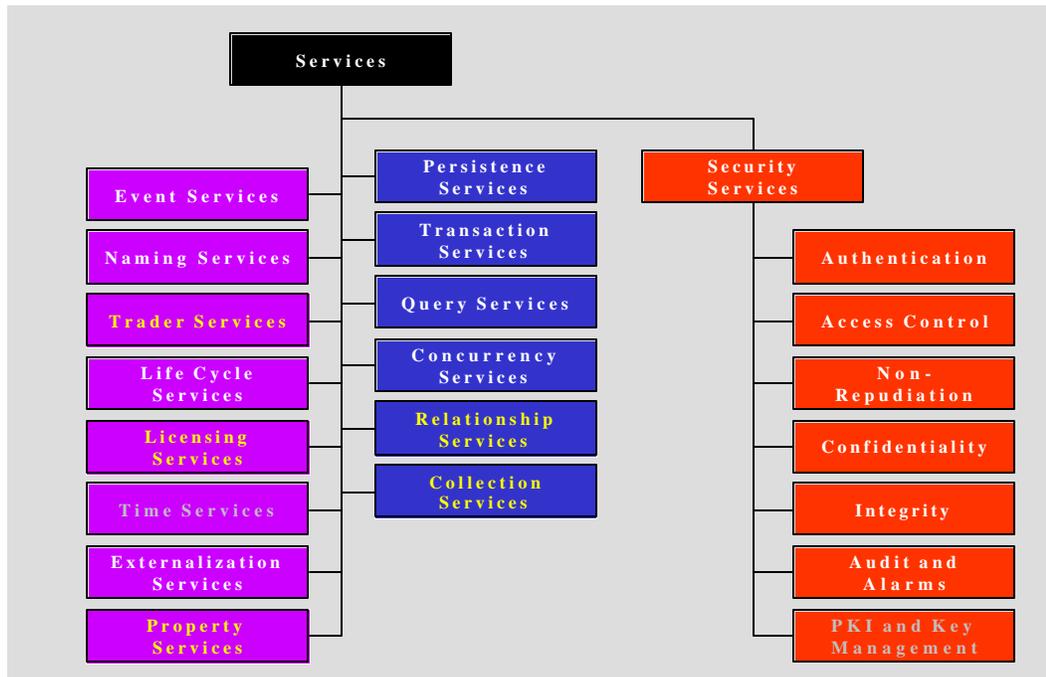


Figure 7: Service Packages of the IF Technical Services

Also provided are the generic environments in which single objects can perform their tasks. Object Services provide for application consistency and help to increase programmer productivity.

While from an OMA perspective, these are intended to be CORBA services, from the IF (and GCSS-AF) perspective these encompass not only CORBA services but also those services required and/or employed by non-CORBA elements of the IF. Examples are those services

required for securing MOM data exchange and Web access, Enterprise Java services such as Naming and Directory services, Transaction services, Messaging services, Data Access and Persistence services, and Resource-Sharing services.

4.3.1.1 Event (and Notification) Service

The Event service and Notification service (*currently in OMG selection*) allows components to establish events and for components to register or unregister their events in specific events. Event services allow the distribution of events among components that know nothing of each other. The IF Event service and Notification service are those specified by CORBA. The Notification service extends the Event service including the addition of Quality of service.

Note Future Capability: These services are available to CORBA components but are not for use by EJB components. The EJB specification is very weak in the area of event handling and services and is likely to expand in a future update. The IF, in a future release, will most likely provide services and/or templates to support event invocation of EJBs.

The Event service and Notification service provided with the current IF release is that provided with the IBM WebSphere (Enterprise Edition - Component Broker) product. The provided Notification service bases itself on the Notification services proposal submitted by the team that includes IBM. While these services are available, their use is discouraged, as the services may become unsupportable.

4.3.1.2 Naming Service

The Naming service is a generic directory service that allows components to locate other components by name. **CosNaming**, through *Java Naming and Directory Interface* (JNDI) for Enterprise JavaBeans, provide services to CORBA components. The IF Naming Service supported by the IF maintains both CORBA and EJB names within the same naming service (directory), thus eliminating the need to search different directories depending upon the type of component being “looked-up” or having to federate and manage two separate directories. This simplifies the integration of EJB and CORBA components. The Naming service provided with the current IF release is that provided with the IBM WebSphere (Enterprise Edition - Component Broker) product.

4.3.1.3 Trader Service

Note Future Capability: The Trader Service provides a “Yellow Pages” for objects (and components). Service providers can register “advertisements” of service availability in the Trader Service. The Trader provides a registry of all publicly known services and may be queried to allow clients to identify candidate implementations. Each advertisement identifies various service characteristics and qualities of service. Given the general single source of service “type” within the GCSS-AF Enterprise, it is not clear at this time the need for a Trader service within the GCSS-AF IF. Requirement for a Trader service should be submitted to the TSG with supporting rationale.

4.3.1.4 Life Cycle Service

The Life Cycle service as specified by OMG provides operations for creating, copying, moving, and deleting objects in a distributed environment; an environment in which location transparency is necessary if extensibility, portability, load balancing, and fault tolerance are to be provided. The Life Cycle service supports this by providing a level of abstraction between the client creating the object and the determination of the location where the object will exist. The Life Cycle service supported by the current IF is based on the OMG CORBA services Life Cycle service specification; *it is not a complete implementation* and includes extensions. However, the IF supported CORBA implementation (as provided by *IBM WebSphere Enterprise Edition*) includes necessary life cycle capabilities through the containers in which manage the CORBA components. EJB containers provide life cycle capabilities for EJB components.

The Life Cycle Service and life cycle capabilities included in the current IF release is those provided with the IBM WebSphere (Enterprise Edition - Component Broker) product.

Note Future Capability: No plans exist to expand the Life Cycle service to include all services as specified by OMG; requirement for this expansion should be submitted to the TSG with supporting rationale.

4.3.1.5 Licensing Service

The Licensing service provides a mechanism to control use of a product (or intellectual property) in a manner determined by the specific business and customer needs. It provides operations for metering the use of components to ensure fair compensation for their use. It supports charging per session, per node, per instance creation, and per site.

Note Future Capability: The current IF does not support the Licensing service. There does not appear to be any need for a Licensing service at this time within the GCSS-AF IF. As such, any requirement for a Licensing service should be submitted to the TSG with supporting rationale.

4.3.1.6 Time Service

The Time service provides the ability to obtain the current time, determine the order in which “event” occurred, generate time-based events based on timers and alarms, and to determine the interval between events. The current IF does not support this OMG defined service. However, within a server the underlying Operating System’s time related services are available with the supported POSIX APIs.

4.3.1.7 Externalization Service

The Externalization service provides a standard way for getting data into and out of a component using a stream-like mechanism. The Externalization services supported by the current IF is a

relatively small subset of the OMG defined Externalization service. The current IF only provides the **StreamIO** and Streamable interfaces defined in the **CosStream** module for internal use only by application server product supported by the IF. As the **CosExternalization** module is not provided, the current IF does not make any Externalization services for client use. The Externalization service provided with the current IF release is that provided with the IBM WebSphere (Enterprise Edition - Component Broker) product.

Note Future Capability: No plans exist to expand the Externalization service to include all services as specified by OMG; requirement for this expansion should be submitted to the TSG with supporting rationale.

4.3.1.8 Property Service

The Property service requires applications objects to inherit the interfaces in the (OMG) standard and provide additional functionality supported through those interfaces. The purpose of properties is similar to that of attributes; however, properties are a dynamic capability that allows the addition of properties-on-the-fly. Properties are useful for applications such as desktop managers, debugging tools, browsers, and other kinds of system-management tools. The Property service is not available in the current IF release. No plans exist to provide the Property service; requirement for this expansion should be submitted to the TSG with supporting rationale.

4.3.1.9 Persistence Service

Persistence services are a set of interfaces used by a persistent object to store its persistent state. The original OMG *Persistent Object Store* (POS) Service was deemed by industry and OMG to be inadequate and is being replaced by a Persistence State Service (PSS).

Note Future Capability: PSS, which is still in the ratification process, was not available for inclusion in the current IF. As such, the PSS as specified by OMG, is not available in the current IF but the persistence service provided in the current IF release is very similar to the specification in ratification.

The current IF does provide persistence for both CORBA and EJB objects-components through containers that encapsulate them. For EJBs, container managed persistence is provided using the same managed framework mechanism provided for non-EJB objects. This persistence mechanism is integrated with the Transaction service to allow persistent objects to participate implicitly within a transaction.

Objects can utilize the IF supported persistence only through use of the managed framework. The Persistence service provided with the current IF release is that provided with the IBM WebSphere (Enterprise Edition - Component Broker) product.

Note Future Capability: The progress of the OMG PSS will be monitored and a decision made in the future as to the requirement to update the persistence service provided in the IF.

4.3.1.10 Transaction Service

Transaction service allows an application to group updates required for a single “task” into a transaction. The Transaction service ensures that either all these updates occur or none occur. If the application has correctly grouped then updates within the transaction, then the data is always updated consistently. If the application uses the Transaction service in conjunction with the Concurrency service, these updates are also not affected by updates being performed for other tasks. If persistent objects or a database is used to store the data, these updates will be permanent even if the system crashes. The IF supported Transaction service provides for both two-phase and one-phase commit processes. The CORBA *Object Transaction Service* (OTS) specification requires both flat and nested transactions.

Note Future Capability: The current IF does not support nested transactions.

In addition to the OTS, the IF also supports the *Java Transaction Service* (JTS) as well as EJB container managed persistence. The IF supported Transaction Service allows both EJB and CORBA components to participate in the same transaction. *The Transaction Service provided with the current IF release is that provided with the IBM WebSphere (Enterprise Edition - Component Broker) product.*

4.3.1.11 Query Service

The Query service, as specified by OMG, provides query operations on collection of objects. The queries are predicate-based and may return collection of objects. Queries can be specified using the SQL3 specification and the Object Database Management Group’s (ODMGs) Object Query Language (OQL). The term “query” is used here to denote general manipulation operations including selection, insertion, updating, and deletion on collection of objects.

Note Future Capability: Insertion, updating, and deletion are not yet available on the IF.

The Query service exists to allow arbitrary users and components invoke queries on arbitrary collection of objects. The current IF Query service supports a query language *Object Oriented SQL* (OOSQL); OOSQL is a subset of the SQL99 and SQL4 standards that are defined by the ANSI/ISO SQL committees. The current IF Query service provides only query capability; thus the insertion, updating, and deletion operations are not supported. The Transaction service provided with the current IF release is that provided with the IBM WebSphere (Enterprise Edition - Component Broker) product.

4.3.1.12 Concurrency Service

The Concurrency service is a set of interfaces that allow an application to coordinate access to a shared resource from multiple transactions or threads. Coordinating access to a resource requires reconciliation of any conflicting actions resulting from simultaneous, multiple transaction or thread access of a single resource. This conflict resolution shall result in the resource remaining in a consistent state. To support this, the Concurrency service supports locking using lock set. While this capability is provided, the IF supported CORBA and EJB framework provides all the necessary resource level locking and caching to coordinate resource access by multiple transactions or threads. The Concurrency service would explicitly be used by developers to provide concurrent access to datastores not supported by the IF CORBA and EJB framework. The Concurrency service provided with the current IF release is that provided with the IBM WebSphere (Enterprise Edition - Component Broker) product.

4.3.1.13 Relationship Service

The Relationship service provides a way to create dynamic associations (or links) between components that have no prior knowledge of each other. It also provides mechanisms for traversing the links that group these components. This service may be used to enforce referential integrity constraints, track containment relationships, and for any type of linkage among components. Included in the OMG Relationship service is the **CosObjectIdentity** module that provides the means to uniquely identify an object from other objects within a distributed system. This is the only service specified in the OMG Relationship service specification supported in the current IF release. The Relationship service provided with the current IF release is that provided with the IBM WebSphere (Enterprise Edition - Component Broker) product.

4.3.1.14 Collection Service

Note Future Capability: The Collection service, as defined by OMG, provides a uniform way to create and manipulate the most common collections generically. Example collections are sets, queues, stacks, lists, and bags. The Collection service defines three categories of interfaces to serve this purpose: collection interfaces and collection factories, iterator interfaces, and function interfaces. No plans exist to provide the Collection service as specified by OMG; requirement for this service should be submitted to the TSG with supporting rationale.

4.3.1.15 Security Services

The IF Security services, in conjunction with the DII COE, processing center boundary protection, DISN, and USAF/DISA operations and support provides end to end security. This encompasses user and application authentication and access control, non-repudiation of user actions, confidentiality of communicated and managed information, integrity of information and systems, and auditing and monitoring of activities of security interest. The IF Security capability provides the USAF Enterprise the ability to provide a cohesive, comprehensive, and integrated security implementation provided the Security services are employed as specified. Note that the IF Security solution includes configuration guides for computational equipment and software that, in conjunction with the IF Security services, are integral to meeting security requirements.

Authentication

This package provides services needed for Identification and Authentication activities. This includes user authentication, component authentication, handshaking involved with establishing communications between software or hardware components, and user session maintenance. The user authentication currently provided by the IF is a standard *userID/password* mechanism that includes an enterprise logon webpage.

Note Future Capability: As DoD PKI plans and implementation mature, a migration of user authentication to certificate/smart card authentication mechanism is anticipated to be supported by the IF.

Component/server authentication services are provided requiring server-side PKI certificates. Enterprise session management is provided such that a user need only log in once. The IF requires the use of IBM Policy Director and MessageSeal product to provide these services.

Access Control

This category encompasses services for controlling access to GCSS-AF objects such as components, databases, individual data objects, etc. It does **not** include objects controlled at the operating system level. GCSS-AF relies upon properly configured Operating System controls at that level. (However, Access Control allows files to be defined as controlled objects in addition to protecting them using OS controls.) Access Control provides services to establish Access control policies and data for the enterprise, to implement those policies during run-time actions of the system, and to administer the user and object data relating to access control policies and decisions. For the current release, implicit access control is provided for GCSS-AF objects on the web servers and Servlet engines, and MQSeries queues and objects.

Note Future Capability: Implicit authorization services for CORBA objects or EJBs are anticipated to be provided in a future release.

The developer with the current release can implement explicit authorization if required prior to availability of the implicit services. IF authorization services are capable of providing protection at several levels of granularity including application invoked access control down to the attribute level. Application engineers shall determine the appropriate level for their application objects, and to set up the required configuration and policy data accordingly. The IF requires the use of IBM Policy Director and MessageSeal products to provide these services.

Non-Repudiation

This category encompasses services that ensure a user cannot deny an action taken by him/her or on his/her behalf, within the system. This includes providing capabilities for creating, verifying, and properly storing digital signatures. The current IF release provides a capability for application components to implicitly sign and verify Business Object Documents (BODs) or other data and messages.

Note Future Capability: User-level digital signatures are anticipated to be available in a future release of the IF. Full non-repudiation archiving is not provided, and is not

currently planned due to lack of COTS capabilities and the immaturity of relevant standards in this area, but may be provided later.

The IF requires the use of IBM Policy Director product to provide these services.

Confidentiality

This category encompasses services that ensure unauthorized individuals cannot view, modify, or delete data without appropriate authorization, and that proper protection is applied to data and code during both storage and transmission. This package provides encryption and communications security for implicitly securing all GCSS-AF-originated messages beginning with the enterprise logon. These provisions include the use of HTTPS, SSL, Secure RPCs, and LDAPS, as provided through Netscape browsers, components of IBM Policy Director, IBM HTTP Server, IBM WebSphere, and Oracle ASO.

Note Future Capability: Multi-Level Security (MLS) is not supported but is expected to be a future requirement.

Applications that require MLS in the meantime will have to supplement the IF to satisfy these requirements.

Integrity

Integrity services are those that ensure system elements and data cannot be compromised or modified by illicit actions. The majority of the services and measures, such as boundary protection, required for Integrity is provided by the processing centers and base networks. System oversight and administration services and measures are provided by ESM package and by USAF operations and support. The IF Integrity capability provides only that information needed to tie into these external elements and with the IF supported security product base, the IBM Policy Director. In addition, the IF requires the proper application of DISA STIGs to the host servers and software that is delineated in the IF provided NT and Unix configuration.

Audit and Alarms

This category encompasses services that record information about activities taken inside the system, whether by human users or software components, and the storage and analysis of those records. This includes creation of audit records, storage of audit records, creation of audit reports, generation of dynamic on-line alarms, and analysis of events and records (whether at runtime or post-mortem). It also provides services to define and administer audit policies, as well as the technical features needed to implement the policies. It also includes intrusion detection, although most of the technical solution for intrusion detection is specified as part of the DII COE and CITS/BIP. The current IF audit provisions are limited to the facilities provided by the supported security product, IBM Policy Director and IF Log Services to record system actions.

Note Future Capability: The current IF does **not** provide audit reduction or alarm posting. Alarm and alert posting is to be provided at the RSA processing centers using the Tivoli Management System capabilities employed by DISA at an RSA. Processing centers that do not utilize this Tivoli capability should plan to implement the same

capability as DISA at an RSA. A future IF release should subsume Tivoli under the Enterprise Systems Management capability. Audit reduction and reporting capabilities are anticipated to be provided in a future IF release.

PKI and Key Management

This package provides services relating to the use of keys, especially those relating to PKI certificates and keys. As USAF is responsible for the issuance and management of certificates, this package only addresses those services dealing with certificate and key retrieval, utilization, and protection within GCSS-AF. The current IF release includes only server-side certificates, using the key protection provided by supported security product (IBM Policy Director).

Note Future Capability: As DoD PKI plans and implementation mature, user authentication certificates and key support is anticipated to be included in the IF.

4.3.2 Facilities

Like Object Service interfaces, these interfaces are also horizontally oriented, but unlike Object Services these interfaces are oriented towards end-user applications. Common facilities provide a set of generic application functions that can be configured to the specific requirements of a particular configuration. Examples of these facilities include printing, document management, database, and electronic mail facilities. Standardization leads to uniformity in generic operations and to better options for end users for configuring their working environments. These also include facilities for use over the Internet.

The high-level packages included under facilities are illustrated in Figure 8: High-Level Packages Under Facilities of the Integration Framework.

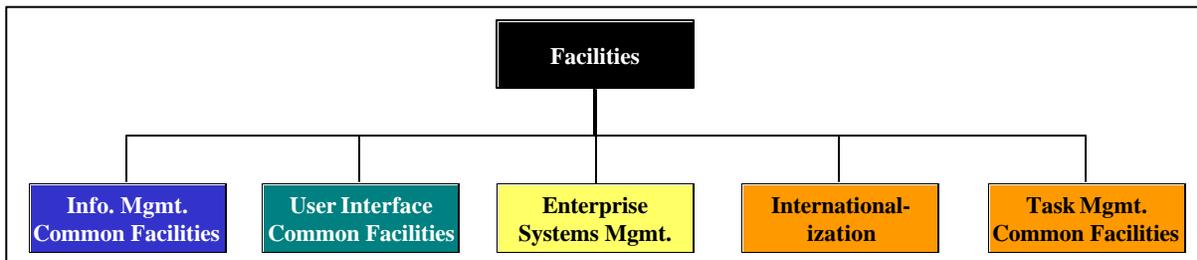


Figure 8: High-Level Packages Under Facilities of the Integration Framework

4.3.2.1 User Interface Common Facilities

The packages included under User Interface Common Facilities are illustrated in Figure 9: Packages of the User Interface Common Facilities.

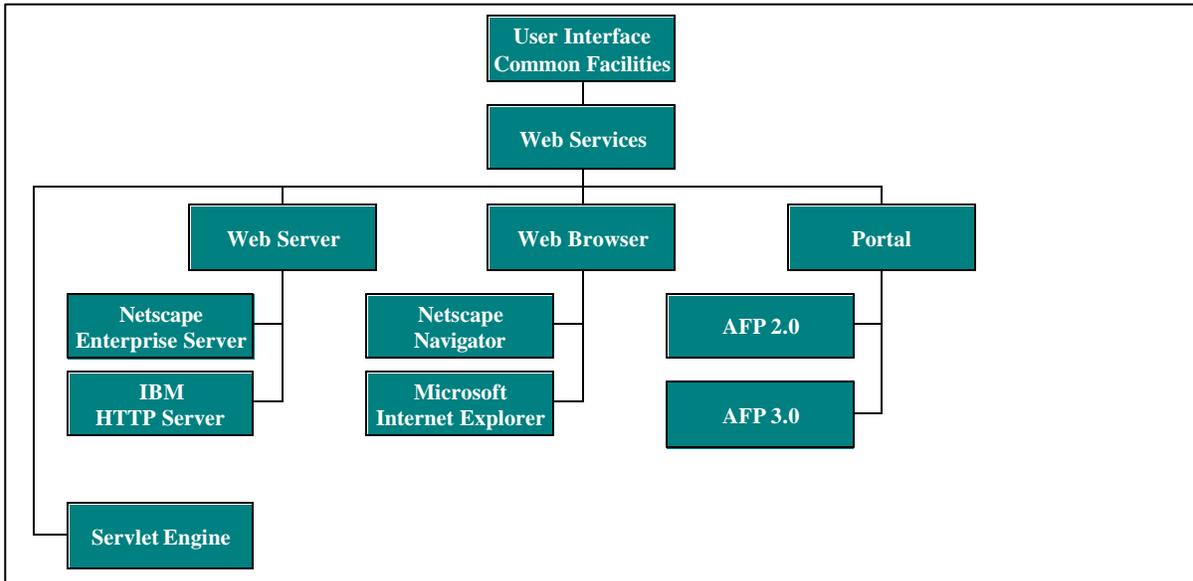


Figure 9: Packages of the User Interface Common Facilities

At the time of this writing, it was anticipated that the Air Force Portal products and services would be allocated to the Reusable Business Component Layer – Web GUI Support layer, see Section 4.4.2 Web GUI Support for more information.

Note that those services and attributes of a generic web browser and generic web server identified in the UML model are solely for the purposes of understanding how the web browser and web server interact with other components of the architecture.

4.3.2.1.1 Web Services

The use of Web-based technology for user interface and presentation is identified as the preferred method in the GCSS-AF System Specification. To this end, COTS Web Browsers provide the client-side capability while the server-side capability is provided by COTS Web Enterprise Servers. It is assumed that any reader of this document would have a general understanding of Web Browsers and Web Servers.

4.3.2.1.1.1 Web Browsers

For the current IF release, a Web Browser (as utilized for the display of information for and interaction with GCSS-AF applications) is utilized to support HyperText Markup Language (HTML) pages only (exclusive of Applets).

An Applet is a program written in the Java programming language that can be included in an HTML page, much in the same way an image is included. When the developer uses a Java technology-enabled browser to view a page that contains an Applet, the Applet's code is transferred to their system and executed by the browser's Java Virtual Machine. Applets require signing if they are employed as part of a GCSS-AF application. Validation of support for signed Applets to meet DoD requirements will be included in a future IF release.

Note that these browsers do in fact support Applets both unsigned and signed. As such they do not preclude interaction with Web Sites and URLs that utilize Applets.

Currently validated Web Browsers are the:

Netscape Communicator (or Navigator)

For detailed information on the Netscape Communicator product, refer to Netscape on-line documentation available through the [Netscape Products](#) web site at <http://www.netscape.com>.

Microsoft Internet Explorer

For detailed information on the Microsoft Internet Explorer product, refer to the Microsoft on-line documentation available through the [Microsoft](#) web site at <http://www.microsoft.com>.

Note that these browsers are validated relative to “pure” HTML pages. Further validation is required when signed Applets are supported as part of the IF.

4.3.2.1.1.2 Web Servers

A Web server is the computer program that serves requested HTML pages or files, generally as requested from a Web browser. Web Servers, for GCSS-AF, shall also support the ability to extend the Web Server functionality with (Java) Servlets. Java Servlets are the Java platform technology of choice for extending and enhancing Web servers.

Currently validated Web Servers are the:

IBM Internet HTTP Server (IHS)

Included with the IBM WebSphere Application Server product line. For more detail on IBM Internet HTTP Server, refer to IBMs on-line documentation available through the [IBM](#) web site found at <http://www.ibm.com>. In addition, IBM Redbooks on the subject can be obtained from the [IBM Redbooks](#) web site at <http://www.redbooks.ibm.com>.

Future supported Web Servers are expected to include:

iPlanet Web Server

The iPlanet Web Server (Superceding Netscape Enterprise Server) is a likely future addition to the validated IF Web Server products. For more detail on iPlanet Web Server, refer to iPlanet on-line documentation available through the [iPlanet](#) web site, <http://www.iplanet.com>.

Microsoft Internet Information Server (IIS)

The Microsoft IIS is a likely future addition to the validated IF Web Server products. It is expected that IIS would be deployed only on Windows 2000 (or above) given its full integration at the operating system level.

4.3.2.1.1.3 Servlet Engines

Servlet engines are effectively “plug-ins” to a Web Server to which a Web Server passes control when a Servlet is requested. The Servlet engine is responsible for executing the requested Servlet. The Servlet engine is also responsible for the processing of Java Server Pages.

Servlets are the Java platform technology of choice for extending and enhancing Web servers. Servlets provide a component-based, platform independent method for building Web-based applications, without the performance limitations of CGI programs. Servlets are server and platform independent leaving one free to select a "best of breed" strategy for servers, platforms, and tools.

Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. Servlets can also access a library of HTTP-specific calls and receive all the benefits of the Java language, including portability, performance, reusability, and crash protection.

JSP is a presentation layer technology that sits on top of a Java Servlets model and makes working with HTML easier. It allows the user to mix static HTML content with server-side scripting to produce dynamic output. By default, JSP uses Java as its scripting language; however, the specification allows other languages to be used (such as JavaScript and VBScript).

Currently validated Web Servers are the:

IBM WebSphere Application Server

The IBM WebSphere Application Server includes the Servlet engine. For more detail on IBM WebSphere Application Server Servlet engine, refer to IBM's on-line documentation available through the [IBM](http://www.ibm.com) web site found at <http://www.ibm.com>. In addition, IBM Redbooks on the subject can be obtained from the [IBM Redbooks](http://www.redbooks.ibm.com) web site at <http://www.redbooks.ibm.com>.

Future supported Servlet engines will be a function of any specific Web Servers and/or Application Servers validated for IF incorporation.

4.3.2.1.1.4 Portal

Note Future Capability: This section will be added with a future release to identify the AF Portal capability which at the time of this writing is defined and selected.

4.3.2.2 Enterprise Systems Management

The GCSS-AF Enterprise Systems Management (ESM) Facility manages the resources comprising the enterprise in which the IF resides. Management includes configuring, monitoring, and controlling these resources. ESM includes a collection of integrated tools, processes for operating the tools, and the people for effecting the processes.

GCSS-AF ESM is partitioned into three domains:

- System Management
- Application Management
- Network Management.

Each ESM domain may be divided into one or more management disciplines:

- Configuration Management
- Fault Management
- Integration Framework Management
- Performance Management
- Security Management
- Accounting Management

The ESM System Management domain includes Fault Management, Configuration Management, Performance Management, Security Management, and Integration Framework Administration. Systems Management encompasses the functions required to configure and monitor computer system level resources (i.e. clients, servers, and attached peripheral devices). Agents that track specific system resource (e.g. CPU, memory, disk space, and I/O) thresholds and continually gauge the overall performance of the systems perform the monitoring process.

The ESM Application Management domain includes Fault Management, Configuration Management, Performance Management, Security Management, and Integration Framework Administration. Application Management encompasses all activities required to manage key applications. This includes the identification, monitoring, tracking and fielding of software components across the enterprise. Agents that track specific application-related thresholds and continually gauge the overall performance of the systems perform the monitoring process. The primary goal of Application Management is to increase application availability.

The ESM Network Management domain includes Fault Management, Configuration Management, Accounting Management, Performance Management, and Security Management of all network resources (e.g. routers, hubs, switches, gateways, and bridges).

Note: GCSS-AF ESM capabilities have been segmented into two categories: those that will be implemented as part of the Integration Framework and those that are to be provided by various support organizations, e.g. DISA/DMC, NCC, SSG/FAB.

4.3.2.2.1 Configuration Management

Configuration Management is planning, identifying, documenting, tracking, and controlling changes to GCSS-AF required resources. Configuration Management of GCSS-AF required resources include automated discovery of system, application and network resources. Configuration Management also includes distribution and installation of software and management of software licenses. Services within the Configuration Management discipline

include Inventory (Asset) Management, Electronic Software Distribution, and License Management.

4.3.2.2.1.1 Inventory (Asset) Management

Inventory Management Services electronically capture an inventory of all network-addressable GCSS-AF managed server and desktop computers. The Inventory system supports the Desktop Management Interface (DMI) specification. The Inventory system is used to track the following types of hardware information: manufacturer, type, model, CPU type, CPU speed, amount of memory, number and size of disk drives, and warranty information.

The Inventory system is used to track the following types of software information per computer: product name, vendor, license type, executable filename, size, version, path, creation/update date and time of installation.

Inventory Management Services captures the enterprise inventory through an auto-discovery mechanism. The enterprise inventory is retained in a database that can be queried to generate reports reflecting inventory status. The information in the inventory database may be manually created, updated and deleted.

4.3.2.2.1.2 Electronic Software Distribution

Electronic Software Distribution (ESD) Services distributes software components (e.g. new software, upgrades, and patches) over the network to one or more GCSS-AF managed servers and desktops within the GCSS-AF enterprise. Software distributed includes both Commercial Off The Shelf (COTS) software and Government Off The Shelf (GOTS) software.

ESD Services synchronizes software installations across the enterprise and operates in conjunction with Inventory Services to retrieve a list of systems in need of an update. During the installation process all files being changed are backed up and automatically restored if the installation fails.

End node users may alternatively pull software from a distribution server for installation at a GCSS-AF managed server or desktop. A notification is provided the user that a software component is available for installation.

4.3.2.2.1.3 License Management

The ESM License Management Service tracks and controls software licenses across the enterprise. Software licenses can be managed per seat, per user, per managed server, per site, or per enterprise. Concurrent software licenses are also managed. License metrics are collected and tracked for:

- The number of users concurrently licensed to a product.
- The number of times a user has been denied a concurrent license.

- The number of times a user has been denied installation of an application due to the lack of a license.
- The number of installations of a product are collected and tracked.

4.3.2.2.2 Fault Management

Fault Management includes the correction of faults reported by system administrators, users and automated fault detection services. Fault Management includes the functionality for establishing a Help Desk to collect information about faults via a "trouble ticket" and tracking the fault through to resolution. Services within the Fault Management discipline include Event Management, Disaster Planning and Recovery, High-Availability Management, Backup/Recovery, Help Desk, and Remote Management (i.e. Remote Diagnostics).

4.3.2.2.2.1 Event Management

The ESM Event Management Service collects and processes events generated by enterprise components. The events are manifested as unsolicited messages written to log files and sent to an enterprise level "event console" for display to the system administrator. A message may have several different meanings. A message may be purely informational in nature, such as an application reporting that a given action has occurred. A message may also be a report from a system performance monitor complaining that a threshold has been exceeded.

Serious errors such as a disk drive error, a memory error or a power failure are also reported as a message. System parameters monitored include Central Processing Unit (CPU) utilization, disk utilization, swap space utilization, memory utilization, disk drive speeds, and power utilization. Unsolicited messages are received from operating systems, IF products, application software, and performance monitoring software. These messages are automatically processed at the "event console" and the system administrator is notified when a failure occurs or when performance has dropped below a predefined threshold level. Methods of notification supported include an alert at a system console, dialing a pager, or sending an e-mail message.

An enterprise level correlation engine categorizes the severity of a timed stamped sequence of messages. Automatic recovery from selected faults and events is supported. Event messages can be filtered to manage the alarm rate being displayed to the system administrator. Summary reports of performance, utilization, and health status of each managed resource within the GCSS-AF domain can be generated at the "event console".

4.3.2.2.2.2 High-Availability Management

The ESM High-Availability Management Service incorporates the fault tolerant capabilities of the IF COTS product set, such as fail-over recovery mechanisms and redundant processing techniques, to maximize IF availability.

4.3.2.2.2.3 Backup-Recovery

The ESM Backup-Recovery Service performs backup and recovery of GCSS-AF managed servers either centrally or locally. Backups include both file systems and databases. Backup services are capable of backing up databases while the database remains online.

4.3.2.2.2.4 Help Desk Interface

The ESM Help Desk Interface utilizes an interface to the Help Desk / Trouble Ticketing system supporting GCSS-AF (e.g. DISA/DMC, base NCC) for the automatic generation of trouble tickets. The centralized Help Desk deployed by the authorized system administrator supports the GCSS-AF system and Mission Applications. The Help Desk should provide interactive user support 24 hours a day, 7 days a week for both in-garrison and deployed locations, and report status back to the problem originator:

4.3.2.2.2.5 Remote Management

The ESM Remote Management Service supports the interface to remote control capabilities utilized by an authorized system administrator supporting GCSS-AF (e.g. DISA/DMC, base NCC) to allow remote management of a user's desktop.

4.3.2.2.2.6 Log Facilities

The ESM Log Facilities Service allows IF and Mission Applications to write messages to log files. The messages can be prioritized to reflect the severity of its contents. Each message subsequently becomes an event and is handled as outlined in Section 4.3.2.2.2.1.

4.3.2.2.3 Integration Framework Management

Integration Framework Administration includes managing Integration Framework resources such as web servers, database servers and Middleware components.

4.3.2.2.3.1 Web Management

The ESM Web Management Service manages Web server content and performs Web content configuration management. This service establishes thresholds for Web performance parameters and subsequently monitors these parameters. Web Management Service generates events when predefined thresholds are exceeded or errors are detected and passes these events to the Event Management Services.

4.3.2.2.3.2 Enterprise Database Management

The ESM Enterprise Database Management Service is comprised of the capabilities provided by the IF COTS product set (i.e. Oracle and DB2).

4.3.2.2.3 Middleware Management

The ESM Middleware Management Service is comprised of the capabilities provided by the IF COTS product set (i.e. Component Broker and MQSeries).

4.3.2.2.4 Performance Management

Performance Management is the process of maintaining performance of GCSS-AF required resources in accordance with established baselines, goals, and policies. Performance focuses on maintaining or improving the quality of service of the resources. Performance management has a real-time component and longer-term functions. The longer-term aspects of performance management include capacity planning, and modeling and simulation.

4.3.2.2.5 Security Management

Security Management is the regulation and administration of access to GCSS-AF required resources in accordance with established guidance, directives, regulations, and policies. Security management includes the distribution and maintenance of all security-related information such as authentication keys, access control codes, and user accounts. Security management also includes capabilities that help make the resources more secure such as encryption, intrusion detection, and detection of security violations. Services within the Security Management discipline include; Access Control, Identification, Authentication, Authorization, Intrusion Detection, Single Sign-On, Encryption, Virtual Private Networks, Virus Detection, Audits, and Non-Repudiation and Digital Signatures.

4.3.2.2.6 Accounting Management

Accounting Management enables charges to be established for the use of GCSS-AF resources and for costs to be identified for the use of those resources. Accounting Management includes functions to inform users of costs incurred or resources consumed; enable accounting limits to be set and tariff schedules to be associated with the use of resources; and enable costs to be combined where multiple resources are involved to achieve a given communications objective.

4.3.2.3 Information Management Common Facilities

Information Management Common Facilities provide facilities that enable information to be modeled, stored, retrieved, moved, and interchanged within an information system. Information includes both codified information held in structured databases and documentary information held in text, image, or some other form. The packages included under Information Management Common Facilities are illustrated in Figure 10: Packages of the Information Management Common Facilities.

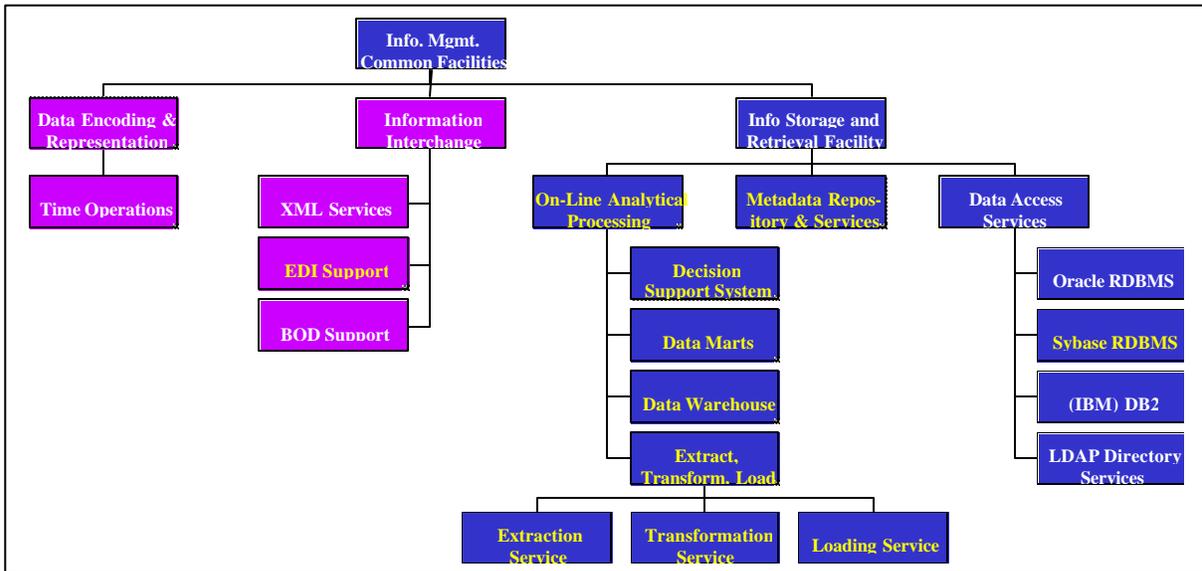


Figure 10: Packages of the Information Management Common Facilities

A good portion of the facilities identified under Information Management Common Facilities is to be provided in later releases of the Integration Framework.

4.3.2.3.1 Information Management Storage and Retrieval Facility

Information Management Storage and Retrieval facilities comprise the higher-level storage and retrieval specifications for distributed applications. These facilities encompass all database products. This includes relational databases, object oriented databases and interfaces, on-line Analytical Processing databases and tools, and directory services used in locating objects, databases, storage systems, etc. The IF currently provides only data access services encompassing the Oracle and IBM DB2 relational database products and LDAP Directory Services using the IBM SecureWay directory server.

4.3.2.3.1.1 Metadata Repository and Services

Note Future Capability: A metadata repository is central storage place to collect, integrate and deploy information about enterprise data. Repository services include numerous interfaces and scanners for various programming languages, DBMS and third party tools (i.e. CASE tools, data movement tools, etc.). The interfaces and scanners extract the appropriate metadata from the source and populate the repository data store. A metadata repository is to be provided in later releases of the Integration Framework.

4.3.2.3.1.2 On-Line Analytical Processing

Note Future Capability: OLAP (online analytical processing) enables a user too easily and selectively extract and view data from different points-of-view. To facilitate this kind of analysis, OLAP data is stored in a "multidimensional" database. A multidimensional database considers each data attribute (such as product, geographic sales region, and time period) as a separate "dimension." OLAP software can locate the intersection of

dimensions and display them. Attributes such as time periods can be broken down into sub-attributes.

OLAP can be used for data mining or the discovery of previously undiscerned relationships between data items. An OLAP database does not need to be as large as a data warehouse, since not all transactional data is needed for trend analysis. OLAP is further decomposed into the following facilities:

Decision Support Systems

A decision support system can be described as an interactive, computer-based system designed to help decision-makers solve poorly structured problems. Using a combination of models, analytical techniques, and information retrieval, such systems help develop and evaluate appropriate alternatives. DSS can be thought of as a data analysis system that makes it easy to manipulate data using computerized analytical tools like statistics packages, data mining, etc. The more sophisticated enterprise-wide analysis systems provide access to a series of decision-oriented databases or datamarts, predefined models and charts, and triggers and alerts linked to events or variables in the “corporate” data warehouse.

Data Marts

A data mart is a repository of data gathered from operational data and other sources that is designed to serve a particular community of knowledge workers. In scope, the data may derive from an enterprise-wide database or data warehouse or be more specialized. The emphasis of a data mart is on meeting the specific demands of a particular group of knowledge users in terms of analysis, content, presentation, and ease-of-use. Users of a data mart can expect to have data presented in terms that are familiar.

Data Warehouses

A data warehouse is a central repository for all or significant parts of the data that an enterprise's various business systems collect. Data from various online transaction processing (OLTP) applications and other sources is selectively extracted and organized on the data warehouse database for use by analytical applications and user queries. Data warehousing emphasizes the capture of data from diverse sources for useful analysis and access, but does not generally start from the point-of-view of the end user or knowledge worker who may need access to specialized, sometimes local databases.

Extract, Transform, and Load Tools

In managing databases, extract, transform, load (ETL) refers to three separate functions combined into a single programming tool. First, the extract function reads data from a specified source database and extracts a desired subset of data. Next, the transform function works with the acquired data - using rules or lookup tables, or creating combinations with other data - to convert it to the desired state. Finally, the load function is used to write the resulting data (either all of the subset or just the changes) to a target database, which may or may not previously exist.

ETL can be used to acquire a temporary subset of data for reports or other purposes, or a more permanent data set may be acquired for other purposes such as: the population of a data mart or data warehouse; conversion from one database type to another; and the migration of data from one database or platform to another.

On-Line Analytical Processing facilities are a future Integration Framework addition.

4.3.2.3.1.3 Data Access Services

Data Access Services are made available to applications to access data storage. This includes the client software utilized to access relational databases and directory services. The IF provides these client services for the Oracle ([Oracle Home Page](#)) and IBM DB2 ([IBM Home Page](#) or [IBM Redbooks Home](#)) relational databases and LDAP services ([LDAP Specification](#)) to the IBM SecureWay directory server ([IBM Home Page](#) or [IBM Redbooks Home](#)).

4.3.2.3.2 Data Encoding and Representation

Note Future Capability: Data Encoding and Representation provides support for data format and translations. Only Time operations have been identified to date. Data Encoding and Representation facilities are a future Integration Framework addition.

4.3.2.3.2.1 Time Operations

Note Future Capability: Time Operations provides support for the manipulation of calendar and time data. Time Operations facilities are a future Integration Framework addition.

4.3.2.3.3 Information Interchange

Information interchange facilities support the interchange of information between different users and different software components. The information interchange facilities provided by the Integration Framework include support for creating and processing BODs, XML Services for processing XML based messages, and EDI Support for creating and processing EDI messages.

Note Future Capability: EDI Support facilities are a future Integration Framework addition.

4.3.2.3.3.1 Business Object Document Support

Business Object Document (BOD) Support facilities are those that aid in the creation and processing of BODs. This includes a standard class for the BOD control area and a “template” for the BOD (business) data area. These facilities also include DTD (segments) from OAG for the BODs utilized for the IF test components (refer to Section 5.1.7 Test Component Descriptions).

4.3.2.3.2 Electronic Data Interchange Support

Note Future Capability: Electronic Data Interchange (EDI) support facilities are those that aid in the creation and processing of BODs. EDI support facilities are a future Integration Framework addition.

4.3.2.3.3 XML Services

XML Services are those that aid in the creation and processing (parsing) of XML documents. This includes an XML parser and an approach to eliminate the need to send the DTD with a message but still have the DTD located locally.

4.3.2.4 Internationalization

Internationalization involves all tasks related to making applications run in multiple countries while adhering to local standards. Currently, only the following package has been identified as illustrated in Figure 11: Internationalization Package. The current release of the IF includes **only** those facilities provided by the Java internationalization framework available in JDK™ 1.1 and later.



Figure 11: Internationalization Package

The Java internationalization framework (based on the internationalization framework developed by Taligent™) introduces various terms and specific classes to support internationalization. For Java, locale identifies a certain geographic or political region for which spoken language and format conventions are specific. The locale object does not contain any locale specific data; rather it serves as an identifier for a certain geographic or political region. Therefore, each class within the application that has locale sensitivity shall provide a method that returns the locales supported by the class.

Resource bundling is a means of separating the program code from all locale-specific data. An example of resource bundling is separating the text on a button from the button itself. Java provides a **ResourceBundle** class specifically designed to aggregate the resources needed for a specific locale. **ResourceBundle** is an abstract class that shall either be sub classed or one of its subclasses. **ListResourceBundle** (an abstract class) is used to store the localized data in an array of type Object. When sub classing **ListResourceBundle**, the developer shall override the **getContents** method and provide an array, where each item in the array consists of a String pair. The first element in the pair is the key and the second is the value associated with that key. For each locale supported a subclass of **ListResourceBundle** will be provided.

PropertyResourceBundle is an abstract subclass of **ResourceBundle** that manages resources for a particular locale by using a set of static strings from a property file. Property files contain text lines made up with keys and their corresponding values and are held in a .property file type.

To obtain a text line, use the *ResourceBundle.getString* method passing the associated key. Note that the **PropertyResourceBundle** can only be used to store strings, not other objects.

Another aspect of dealing with internationalize is the formatting of data which includes messages, numbers, percentage, currency, date, and time. Java provides various classes to support formatting of these data types.

These and other capabilities within Java provide the basis for creating internationalized GUIs. These capabilities can be used (by the GUI developer) in conjunction with the business-related components to extend their reusability.

4.3.2.5 Task Management Common Facilities

Note Future Capability: Task Management facilities provide an infrastructure for applications and desktops that model and support the processing of user tasks. The infrastructure contains facilities for managing user and project workflows, rules, and communications. Task Management facilities are a future Integration Framework addition.

The packages of Task Management Common Facilities are illustrated in Figure 12: Packages of Task Management Common Facilities.



Figure 12: Packages of Task Management Common Facilities

4.3.2.5.1 Workflow Facility

Note Future Capability: The Workflow Facility provides management and coordination of objects that are part of a work process. The Workflow Facility should provide support for:

Production-based Workflow

A structured, pre-defined process that is governed by policy and procedure. Areas in which this type of workflow is applicable include configuration management, service requests, document routing and review, purchase orders, etc.

Ad Hoc Coordination-based Workflow

An evolving workflow defined by one or more people to support coordination of knowledge workers. This type of workflow is not predefined, and is intended to support knowledge workers in their daily activities. Areas in which this type of workflow is

applicable include strategic planning, quick reaction requests, ad hoc tasking and request, etc.

Workflow Facility is a future Integration Framework addition.

4.4 Reusable Business Component Support

Note Future Capability: Relative to applications, the contents of this area include templates for building business objects/components and applications. It is envisioned that the IF will eventually provide templates and/or extensible classes for component initialization, transaction support across geographically separated processing centers, Web Interface structure, Legacy and external system interface, etc. These would be provided as aids to and/or examples for the developer. The full scope of items cataloged under Reusable Business Component Support will in many cases be derived from actual application developments and abstracting out components, templates, base and/or abstract classes that can be used to support the development of multiple applications and/or components.

The packages encompassed by the Reusable Business Component Layer are illustrated in Figure 13: Packages of the Reusable Business Component Support.

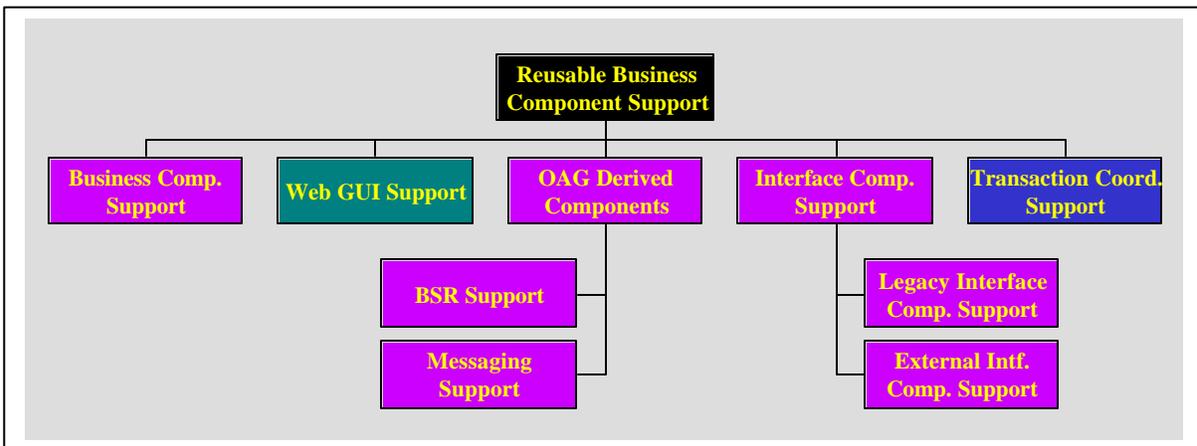


Figure 13: Packages of the Reusable Business Component Support

Reusable Business Component Support items are a future Integration Framework addition.

4.4.1 Business Component Support

Note Future Capability: Business Component Support items include those items such as application or component initialization templates, application controller templates, etc. The full scope of this area will be dictated by evolving Business Component design needs. Business Component Support items are a future Integration Framework addition.

4.4.2 Web GUI Support

Note Future Capability: Web GUI Support items include those items such as view controller templates, Portlet templates, etc. The full scope of this area will be dictated by evolving Web Interface design needs. Web GUI Support items are a future Integration Framework addition.

4.4.3 OAG Derived Components

Note Future Capability: OAG Derived Components items include those items that support the development of components dictated by the application of the Open Application Group Integration Specification in GCSS-AF application design. The full scope of this area will be dictated by evolving Business Component design needs.

Categories expected to be included under OAG Derived Components include:

BSR Support

BSR Support for the development of components to provide the (OAG) Business Service Request implementations for the BODs that an application or component will “service”. This may include event handlers, dispatchers to route a BOD to the appropriate handler, etc.

Messaging Support

Messaging Support for the handling and routing of OAG specified (BOD) messages. An example might be a Message Listener.

OAG Derived Components items are a future Integration Framework addition.

4.4.4 Legacy Interface Component Support

Note Future Capability: Legacy Interface Component Support items include those items such as templates for the GCSS-AF front-end (e.g. BSRs), conversion utilities to convert between GCSS-AF message formats and Legacy system formats, templates or base classes for aiding in Legacy system login, screen scraper based interfaces, etc. The full scope of this area will be dictated by evolving Business Component design needs.

Legacy Interface Component Support items are a future Integration Framework addition.

4.4.5 External Interface Component Support

Note Future Capability: External Interface Component Support items include those items such as templates for the GCSS-AF front-end (e.g. BSRs), conversion utilities to convert between GCSS-AF message formats and external system formats, templates or base classes for aiding in any external system login, etc. The full scope of this area will be dictated by evolving Business Component design needs.

External Interface Component Support items are a future Integration Framework addition.

4.4.6 Transaction Coordination Support

Note Future Capability: Transaction Coordination Support items include those items that might provide a higher level of transaction coordination. For example a template might be provided that can act as a transaction controller too not only initiate the transaction but also to handle transaction completion including transaction exceptions and subsequent rollback. The full scope of this area will be dictated by evolving Business Component design needs.

Transaction Coordination Support items are a future Integration Framework addition.

4.5 Service to Product Mapping

The following tables provide a mapping of the services and capabilities provided by the IF to the products that implement the service or capability. The specific versions of products relative to an IF release can be found in the Integration Framework Version Description Document.

Table 1: Service to Product Mapping: Infrastructure

Service/Capability Provided	Supporting Product(s)
Platforms	
Sun Microsystems	Sun
HP	Hewlett-Packard
Intel	Many
Network Devices	Cisco Local Director
	Other – DISA Provided
Operating Systems and Services	
Solaris	Sun
HP/UX	Hewlett-Packard
Windows NT	Microsoft
Windows 95/98	Microsoft
Windows 2000	Microsoft
Database Engines	IBM DB2
Java Virtual Machine	IBM (as provided by WebSphere)
	SUN
Network Services	

Table 2: Service to Product Mapping: Integration Services

Service/Capability Provided	Supporting Product(s)
Distributed Control	
CORBA	IBM WebSphere Enterprise Edition
Enterprise Java Platform	IBM WebSphere
COM+	Microsoft – future
Distributed Computing Environment (DCE)	
Application Server	IBM WebSphere Advanced Edition
	IBM WebSphere Enterprise Edition
Messaging	
Message Oriented Middleware	IBM MQSeries

Service/Capability Provided	Supporting Product(s)
Open Application Middleware API Specification (OAMAS)	IBM Application Message Interface (AMI)
Java Messaging Service (JMS)	Future
CORBA 3.0 Messaging	Future
Distributed Data Access	
Open Database Connectivity (ODBC)	
Java Database Connectivity (JDBC)	
Net8 / SQLNet	Oracle Net8
Distributed Communications Protocols	<u>Operating Systems</u> Solaris, HP/UX, Microsoft Windows
	IBM WebSphere
	<u>Web Browsers</u> Netscape, Microsoft IE
Network Time Protocol	As provided by DISA

Table 3: Service to Product Mapping: Technical Services/Services

Service/Capability Provided	Supporting Product(s)
Event (and Notification) Service	IBM WebSphere Enterprise Edition
Naming Service	IBM WebSphere Enterprise Edition
Trader Service	Future
Life Cycle Service	IBM WebSphere Enterprise Edition
Licensing Service	Future
Time Service	<u>Operating Systems</u> Solaris, HP/UX, Microsoft Windows
Persistence Service	IBM WebSphere Enterprise Edition
Transaction Service	IBM WebSphere Enterprise Edition
Query Service	IBM WebSphere Enterprise Edition
Concurrency Service	IBM WebSphere Enterprise Edition
Relationship Service	IBM WebSphere Enterprise Edition
Collection Service	Future
Security Services	IBM/Tivoli Policy Director
	IBM WebSphere Enterprise Edition

Table 4: Service to Product Mapping: Technical Services/Facilities

Service/Capability Provided	Supporting Product(s)
User Interface Common Facilities	
<i>Web Services</i>	
Web Browsers	Netscape Communicator
	Microsoft Internet Explorer
Web Servers	IBM WebSphere
Servlet Engine	IBM WebSphere Advanced Edition
Enterprise Systems Management	
Accounting Management	Future
Configuration Management	
Inventory (Asset) Management	Future
Electronic Software Distribution	Future

Service/Capability Provided	Supporting Product(s)
License Management	Future
Fault Management	
Event Management	Tivoli as supplied by DISA
High-Availability Management	Future
Backup-Recovery	<u>Operating Systems</u> Solaris, HP/UX, Microsoft Windows
Help Desk Interface	Future
Remote Management	Future
Log Facilities	Log4j (as supplied through LMSI-O)
Integration Framework Management	
Web Management	IBM WebSphere Advanced Edition
Enterprise Database Management	<u>Database Engines (inc. utilities)</u> IBM DB2, Oracle RDBMS
Middleware Management	IBM WebSphere
	IBM MQSeries (utilities)
Performance Management	Future
Security Management	IBM/Tivoli Policy Director
Information Management Common Facilities	
Information Management Storage and Retrieval Facility	
Metadata Repository and Services	Future
On-Line Analytical Processing	Future
Data Access Services	IBM DB2
	Oracle RDBMS
	IBM/Tivoli SecureWay Directory Server (LDAP)
Data Encoding and Representation	
Time Operations	Future
Information Interchange	
BOD Support	LMSI-O provided BOD classes
EDI Support	Future
XML Services	
Internationalization	Future
Task Management Common Facilities	
Workflow Facility	Future

Table 5: Service to Product Mapping: Reusable Business Component

Service/Capability Provided	COTS Product
Business Component Support	LMSI-O provided templates/examples
Web GUI Support	LMSI-O provided Menu Servlet
OAG Derived Components	Future
Legacy Interface Component Support	Future
External Interface Component Support	Future
Transaction Coordination Support	Future

4.6 How The IF Provides Services and Facilities

The IF provides the following to *aid* in the development of a MA implementation model and for the implemented MA to access IF services and facilities:

UML Model

A UML model (the GCSS-AF Systems Solutions UML Model, also know as “the model”) is provided which describes the services and facilities that are provided by the GCSS-AF IF. This model also contains examples of how applications might use these services.

COTS Products

All COTS products are provided as delivered by the vendor on CD. Vendor products need be installed and configured as required by the configuration on which the product(s) are to be installed. These may be development configurations, test configurations, and/or deployed configurations. Refer to the GCSS-AF IF Installation Procedures for specific installation guidance.

Class Libraries

There are a set of **jar** files and class libraries that are provided for the use of applications that require the services or capabilities provided by the IF. For descriptions of what is available, how to get them, and how to use them, reference either the model or Section 5 Design Guidance for GCSS-AF Applications Using the IF.

Code Templates, Base Classes, and Helper Classes

The IF provides a set of code templates and base classes that may be extended for the specific use of the application. Based on the design of the application the developer will choose a set of products to be used for implementation. These products will include the products specified by the IF as well as any additional product for unique requirements of the application. The selected products specified by the IF may have a set of code templates and/or helper classes associated with them that would be used to guide the development of the components of the application. An example of this would be code templates and helper classes for integrating the application with the IBM MQSeries product. The templates in this case provide a level of separation for the application business logic from the infrastructure support. The templates are in skeleton form and would be modified for specific use by the application. The helper classes constitute the interface that application developers will use to make the task of the application developer simpler and remove the specifics of the product from the interface.

Table 6: Sample Identification of Code Template, Base Class, and Helper Class, provides an identification of an item for each of these categories. The complete list of available items, along with details for their usage can be found in Appendix A of this document.

Table 6: Sample Identification of Code Template, Base Class, and Helper Class

Item	Type	Use
Application Initialization	Code Template	Provides a template that can be used by application developers as a starting point for the code required to initialize an application. Essentially a tailor-able example.
GCSS-AF Servlet	Base Class	Adds connections to IF services to complete interaction with security services. Intended to be used as base class for Servlets developed for GCSS-AF applications.
AMI Helper Classes	Helper Class	The AMI Helper classes provide additional services that an application can call to "help" in using the AMI to send and receive messages.

5. Design Guidance for GCSS-AF Applications Using the IF

This section provides the application developer basic design guidance to utilize the Integration Framework capabilities. It contains specific information about the interfaces provided, their intended use by applications, architectural recommendations for the application structure, and IF COTS product configuration expectations and considerations. This section (or guide) is *not* intended to identify how to configure or set-up IF services and/or components; it only describes how to utilize and interact with those capabilities.

The intended audience for this section is architects and developers who are familiar with object-oriented design and development, the [Common Object Request Broker Architecture \(CORBA\)](#), and the [Java 2 Enterprise Edition \(J2EE\)](#) technology. In addition, the user should also be familiar with the concept of Component Based Development as discussed in the [Application Framework Developer's Guide](#).

As the [Open Application Group Interface Specification \(OAGIS\)](#) is employed to define the coarse-level business components, the user should also be familiar with that specification. Business components identified within this specification such as Invoicing, Accounts Receivable, and General Ledger are examples of components referred to as OAGIS coarse-level components within this developer's guide. The guidance that follows assumes that components have been defined to this level, and associated Business Object Documents (BODs) and interfaces have been defined for each component. This guidance also assumes that all pertinent requirements and use cases have been allocated to these components.

Section 5.1 Mission Application Architecture Considerations provides the bridge between the component specification guidance of the [Application Framework Developer's Guide](#) and the technical component development guidance in the sections following 5.1. Distinctions between new designs, migrated designs, wrapped (Legacy) applications, and interface components are primarily limited to 5.1 Mission Application Architecture Considerations and its subsections. Throughout Section 5 references are made to the Integration Framework test components as well as citing as examples these same test components. These test components were developed both

as the means to test and exercise the Integration Framework and to provide examples to developers of how to use the IF and its services.

A web site is provided by SSG to obtain all pertinent software and guidance associated with the Integration Framework. Included at this web site are:

- Guidance for obtaining required vendor software.
- All required Integration Framework (non-vendor) software.
- The Integration Framework test components.
- Integration Framework documentation.

5.1 Mission Application Architecture Considerations

This section provides a generalized description of the structure and composition of a mission application. After the mission application architecture has been defined in terms of OAGIS coarse-grain components, the architect/designer shall choose the major design components that will be required to satisfy the requirements allocated to these OAGIS-coarse grain components. This structure and associated elements provides the context in which the description and use of IF capabilities are described.

This section provides an overview of the architectural considerations that need to be made when specifying, developing, migrating, adapting, or interfacing an application within the GCSS-AF enterprise. It is intended to provide a road map for the actual development of an application or application interface.

Section 5.1.1 Reference Application Model describes the two major models used as the reference for specifying an application.

Section 5.1.2 Reference Application Development Process identifies the various analysis, design, and implementation components and the reference process in which these components would be specified, developed, and implemented. It also defines the relationships (mappings) between the components identified in each of the analysis, design, and implementation phases.

Section 5.1.3 Application Integration provides a brief description of the possible ways of integrating applications. This section also provides a brief identification of the name space and naming conventions specified by GCSS-AF..

Section 5.1.4 Application Security Responsibilities – provides a very brief introduction to security activities required in developing an application.

Section 5.1.5 Application and Integration Framework Integration Points identifies / summarizes the integration points between an application and the Integration Framework.

5.1.1 Reference Application Model

The reference model for GCSS-AF applications is centered on two complimentary and well-proven models for distributed, client/server applications. These are the n-tier model for decomposing an applications presentation, business, and data processing into separate manageable layers and the Model-View-Controller (MVC) design pattern that allocates processing responsibilities to these components. This section will provide a brief orientation to these models. For detail description of these models, numerous articles and books are commercially available or accessible over the Web and should be consulted.

5.1.1.1 N-Tier Model

N-Tier is the model specified for GCSS-AF application development. This model involves dividing the application into 3 or more independent layers or tiers, each of which can run on the same machine or all running on different machines. This multi-tier architecture allows the application to scale across multiple processors on different machines. Figure 14: N-Tier (Mission Application) Architecture illustrates the different tiers (layers) in which components of an application will reside.

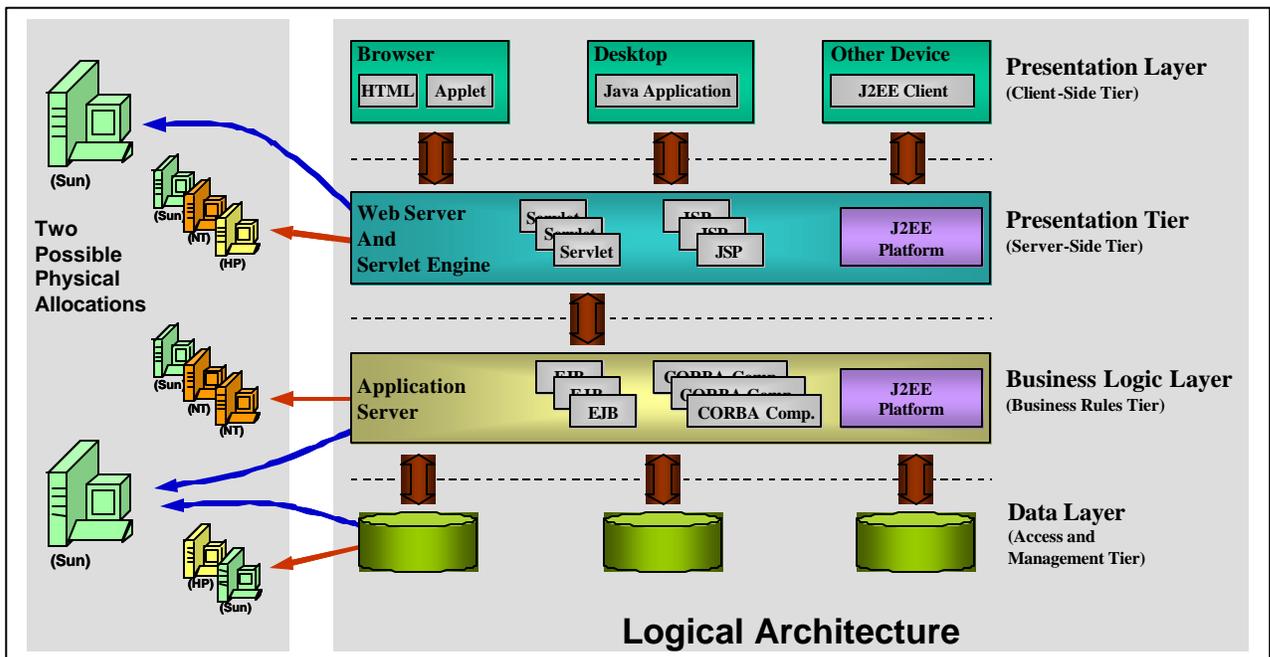


Figure 14: N-Tier (Mission Application) Architecture

Presentation Layer

Components in this layer define how the data is rendered and manipulated by the user. This can be as simple as any web browser, or may be very complicated using customized controls, forms, Servlets, JSPs, and distributed object models. Note that current Web application designs, the presentation layer is spread across both client (browser) and server machines.

Business Logic Layer

Components in this layer contain and execute the logic behind the application and do not contain any user interface. It can only be accessed by another program or component. For example, this may be the logic for an inventory control application.

Data Layer

Components in this layer provide and maintain the actual data (and/or) services required by the application. Its sole responsibility is to manage data and services. It may contain some of the business logic, such as SQL stored procedures.

For GCSS-AF, this general approach is applicable to all the following types of applications:

New and/or Migrated Application – is an application that has been developed as either a new application to GCSS-AF requirements or has been “converted” from a Legacy application meeting GCSS-AF requirements. Note that the migration of a Legacy application may very well be accomplished incrementally while maintaining application operation. During such a migration, the application would (allowably) be a mix of *migrated* and *wrapped* application types.

Wrapped (Legacy) Application – is a Legacy (or COTS) application for which a wrapper(s) (i.e. interface components) have been developed. Interface components provide GCSS-AF specified application interface styles to Legacy systems or COTS products that do not make available GCSS-AF style interfaces.

Interfaced Application – is a Legacy (or COTS) application whose data/database can be accessed directly or indirectly from an application meeting GCSS-AF requirements.

Based on the N-Tier model, all applications, regardless of type, should generally exhibit the design structure illustrated in Figure 15: Top-Level Mission Application Reference Design. The identified component categories (Presentation, Business, and Data) map directly to the layers of the n-tier model. Note that depending upon the application, not all of the categories may be required. For example, some type of service (application) may be required to provide an interface to a Legacy system and represent that Legacy system to other GCSS-AF applications and require no user interface (i.e. presentation components); this is the rationale for the intermediate (and arbitrary) *Application* category. Or an application may be required to simply display information from an existing application and effectively only require presentation components.

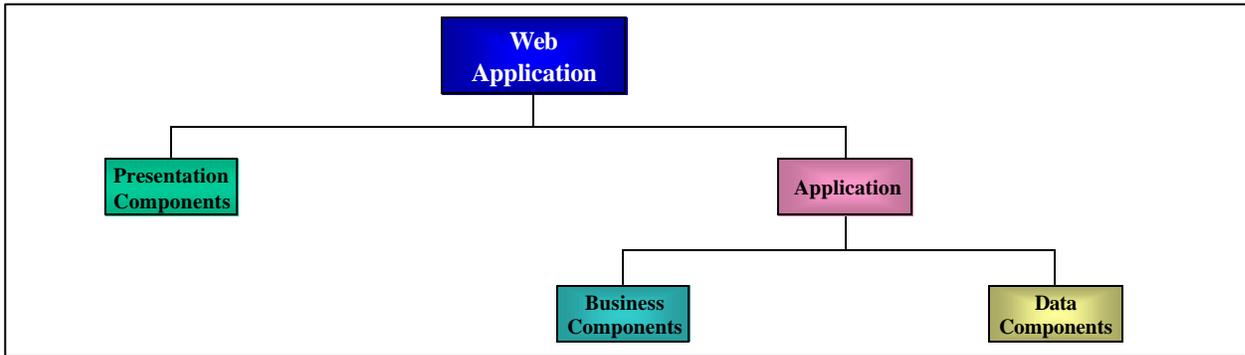


Figure 15: Top-Level Mission Application Reference Design

These component categories should first be populated with analysis level logical components that will then be refined to design/implementation (technology) components such as Servlets, JSPs, CORBA components, EJBs, databases, etc. This is nothing more than specifying a design, developing a design to meet application specification (i.e. design-level components) and then implementing them (implementation components). Note that this maps directly to the Analysis, Design, and Implementation categories of the GCSS-AF Rose Model.

5.1.1.2 Model-View-Controller Design Pattern

The Model-View-Controller design pattern is the reference structure for a GCSS-AF Mission Application. This does not mean that all applications need be built on this model; only that it should be viewed as a reference point for establishing the actual design structure for an application. It also serves to provide a context for discussion of the components of an application. One source of reference, from which the much of this MVC section was obtained, is the IBM Redbook, *Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition*.

User-to-business applications encompass interactions that have a common set of processing requirements that need to be considered on the server side of the application. These interactions are easily mapped to the classical Model-View-Controller design pattern as illustrated in Figure 16: Model-View-Controller Design Pattern.

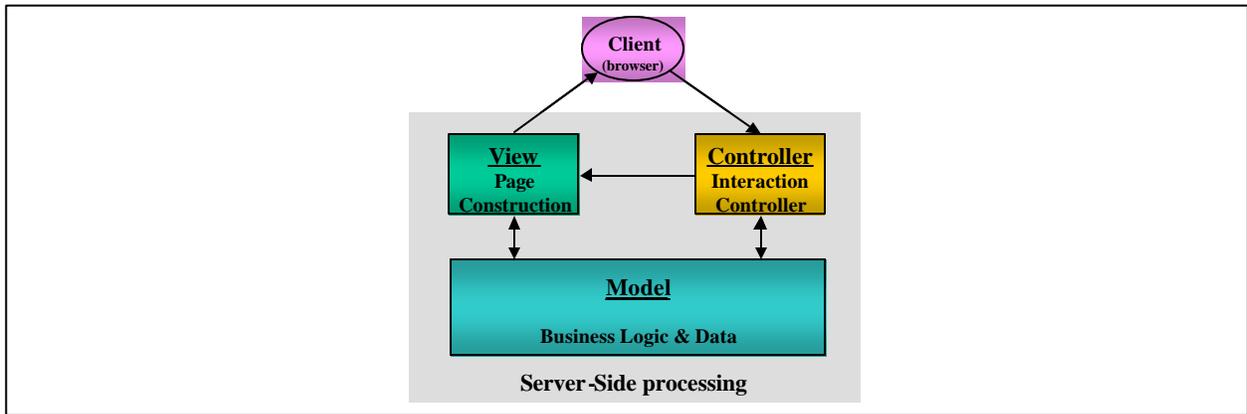


Figure 16: Model-View-Controller Design Pattern

In the Model-View-Controller (MVC) paradigm the user input, the modeling of the external world, and the visual feedback to the user are explicitly separated and handled by three types of components, each specialized for its task. Model represents the application object that implements the application data and business logic. The Controller is responsible for receiving the client request, invoking the appropriate business logic, and based on results, selecting the appropriate view to be presented to the user. The following descriptions are from the perspective of a web application.

(Interaction) Controller

The controller's primary responsibility is mapping HTTP protocol-specific input into the input required by the protocol-independent business logic, scripting the business logic elements (of business objects/components) together and then delegating to a page construction component (view) that will create the response page to be returned to the client.

Typically functions performed by the controller include:

- Validate the request and session parameters used by the interaction. (*Page Controller*)
- Verify that the client has the necessary privileges to access the requested business task. (*Page Controller*)
- Transaction demarcation. (*Application Controller*)
- Invoke business logic components to perform the required tasks. This includes mapping request and session parameters to the business logic component's input properties, using the output of these components to control logic flow and correctly chain the business logic. (*Page Controller and Application Controller*)
- Call the appropriate page constructor component based on the output of one or more of the business logic components. (*Page Controller*)

View (Page Constructor)

The page constructor is responsible for generating the HTML page that will be returned to the client; it is the view component of the application. In many cases the controller may pass the dynamic data as objects (or JavaBeans) for formatting. In other cases, the display page may invoke business logic directly to obtain dynamic data. The latter should be used with great caution as it increases both the complexity and processing requirements of the display page. Once the page constructor has the dynamic data it will typically format the data.

Model (Business Logic and Data)

The business logic, which also encapsulates the business data, is responsible for satisfying client requests. (Note that clients are not restricted to just human users but may be other applications as well.) As such, business logic shall address a wide range of potential requirements that include ensuring transactional integrity of application components, managing and accessing application

data, supporting the coordination of workflow processes, and integrating application components and applications including Legacy applications.

Mapping to N-Tier Model

The MVC model components have a specific mapping to the N-Tier model layers. The Model maps directly to the Business Logic and Data layers and the View maps directly to the Presentation layer. The Controller mapping is not so direct for GCSS-AF. This is due to the fact that the controller is “controlling” both presentation and business components. As such, the controller can be considered a composite of two controllers; a page controller managing the presentation processing and an application controller that manages the work or process flow of the business logic. Refer to Figure 17: Model-View-Controller Extended Design Pattern. Typically (but not necessarily) the page controller would map to the Presentation layer and the application controller to the Business Logic layer.

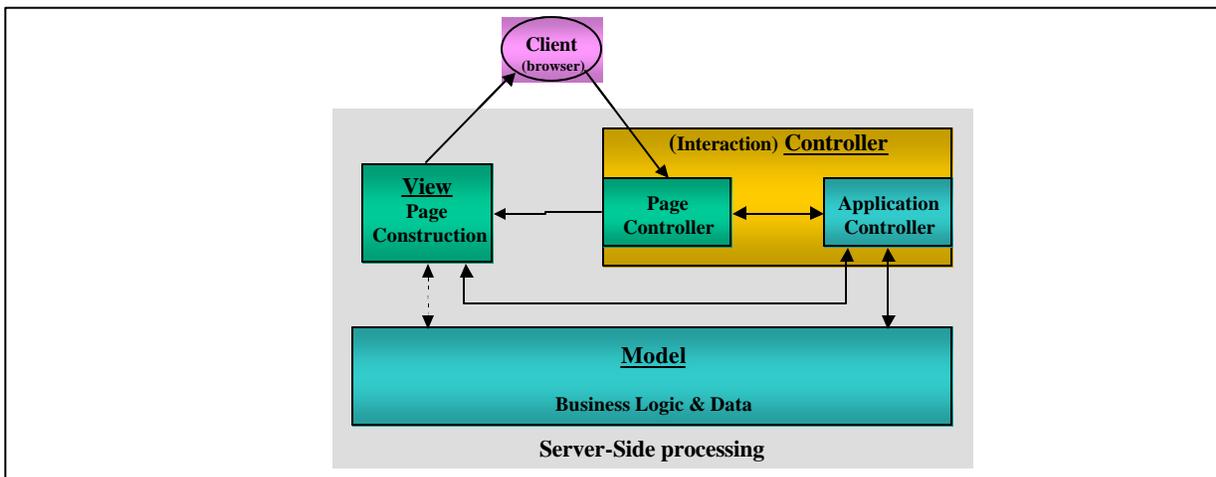


Figure 17: Model-View-Controller Extended Design Pattern

(This controller distinction is especially important for application-to-application requests and associated processing; in such cases presentation related processing on the part of the serving application is required. All business logic control would be provided through the application controller.)

5.1.2 Reference Application Development Process

This section describes (in brief) the reference process from which the various components that will comprise an application are defined and developed. While this reference process is illustrated as a sequential (waterfall) flow, it is for reference only; the actual process employed is left to the application development team or organization. However, for sake of consistency, communication, and reuse within the GCSS-AF Enterprise community, development teams or

organizations are expected to define the components identified herein. In addition, this information should be captured in the GCSS-AF Rose Model.

The reference process is illustrated in Figure 18: Reference Mission Application Development Process. It assumes that application requirements have been defined and allocated to this application and that the Business Model as identified in the Figure 18: Reference Mission Application Development Process has been developed. Refer to the *GCSS-AF Application Developer's Guide* for a description of a Business Model.

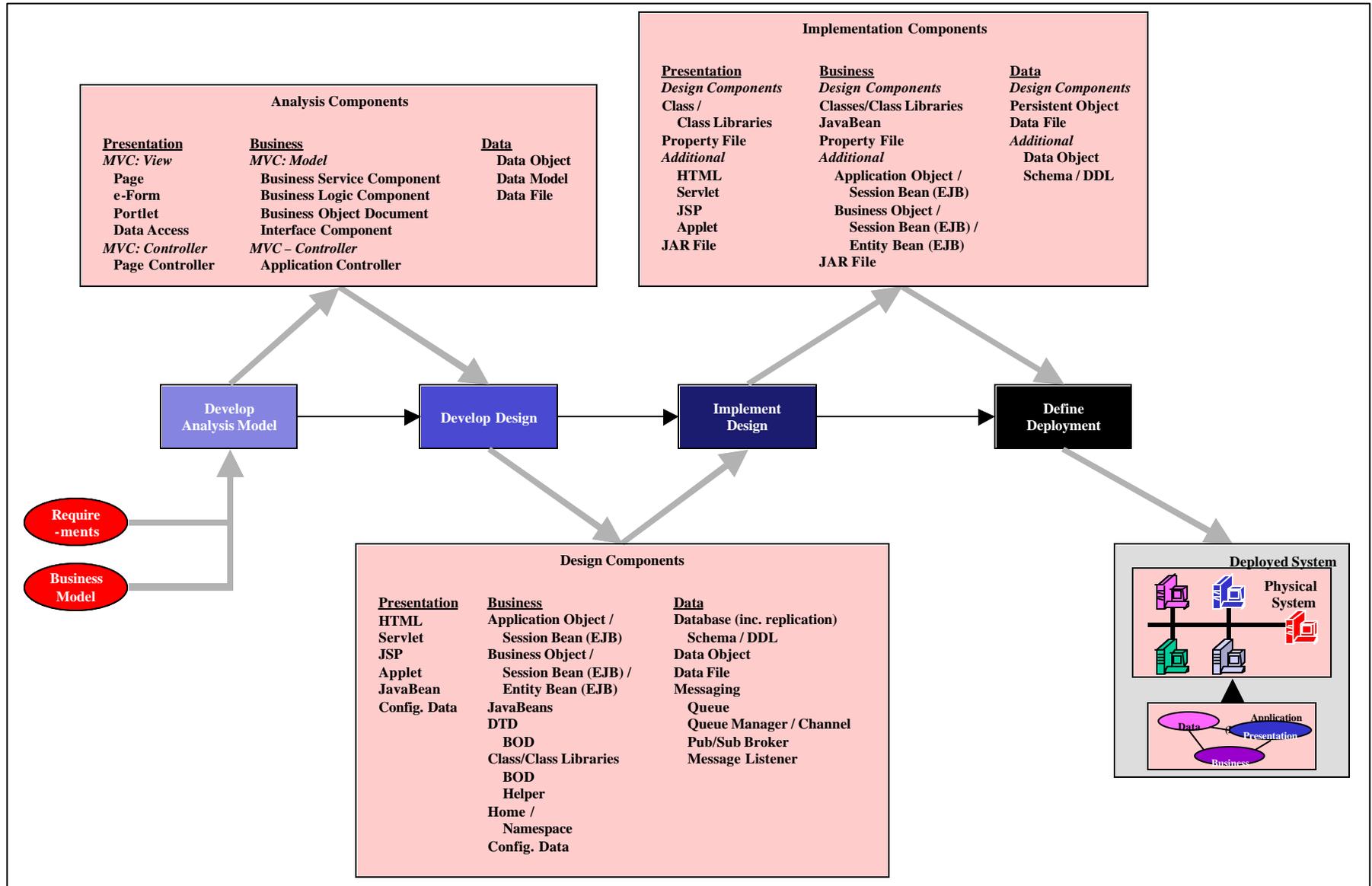


Figure 18: Reference Mission Application Development Process

5.1.2.1 Analysis

Note: This section does not contain guidance on how to develop an analysis model. This is a software engineering discipline and skill that should be held by the user of this guide. In specifying an application, the architect and/or designer needs to define the specification level components he/she needs to meet the requirements levied on the application.

For GCSS-AF, reference analysis level component types have been identified for consideration in specifying the application. These component types, by all means not the only ones allowed, are categorized in Figure 19: Mission Application - Analysis Model View relative to both the N-Tier model and the MVC pattern.

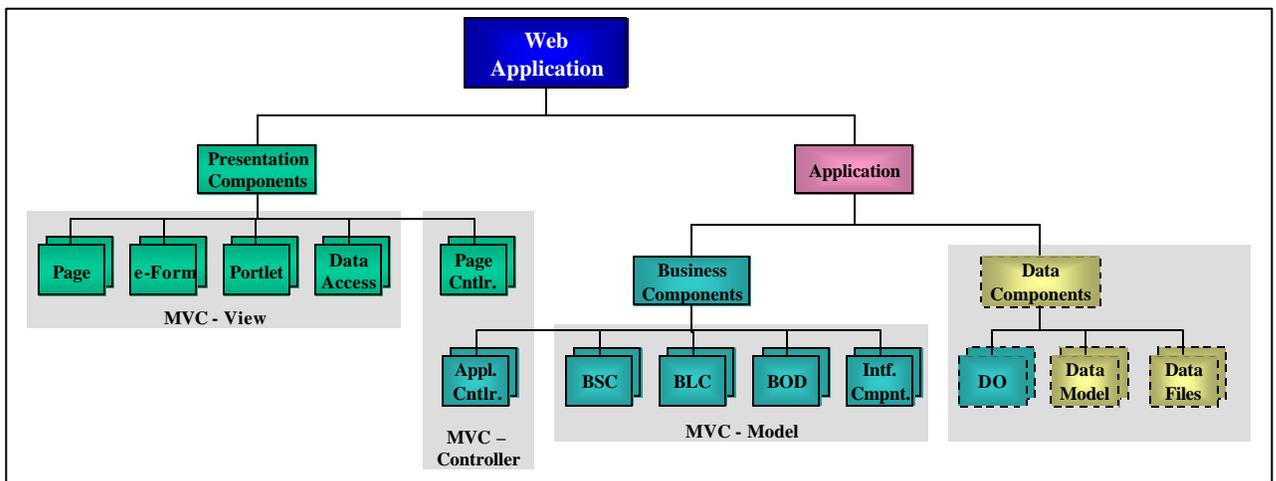


Figure 19: Mission Application - Analysis Model View

During analysis the mission application analyst(s) / systems engineer(s) would identify and specify the abstracted components for which a design will be developed. At minimum, it is expected that components would be specified to address:

- Presentation requirements for user interface in terms of display and interaction “pages.”
- Business requirements in the terms of components encapsulating business rules and logic (Business Logic Components), BODs that the application produces, and Business Service Components (BSC) that encapsulate one or more Business Service Requests (BSR). The BSR is the action that the sender application wants the receiver application to perform. BSRs are the OAGIS list of common actions for business applications to perform to accomplish application integration.

Note that the BSC may or may not also be specified as the “application” controller.

- During analysis, business data is generally *not* specified per se as business objects encapsulate business data. However, if queries will be made against the business data, a logical data model at the entity level is created.

- Additionally files, including log files, while not illustrated in Figure 19: Mission Application - Analysis Model View should be identified during analysis to include identification of the information being accessed or logged.
- During analysis, specification of interactions and/or use of the Integration Framework (IF) with respect to the application should generally be limited to identifying which (top-level) services of the IF will be used. That is, does it require security services, transaction services, naming services, etc?

Analysis modeling should be focused on the application, interactions between components of the application, and interactions with other applications. Interactions with and through the IF do not generally appear until design.

Security Considerations

During analysis, the level and type of security (if any) should be identified for each specified component.

This should include the following relative to each component:

- What information exchanges between components need to be encrypted?
- Does the requestor (or sender) of information need to be authenticated?
- Does the requestor (or sender) of information need to be authorized?
 - What are the roles that can access the component and with what privileges?
 - Who needs to be authorized, the initiating user or the requesting component?
 - What user presentation information needs to be restricted based on user role?
- What, if any, audit information needs to be captured?

Refer to Section 6 Securing the Application for security guidance details.

Application-to-Application Interactions

Business interfaces (primarily BODs) external to the application need to be identified to include:

- Associated interaction scenarios identifying types of communication (point-to-point, publish/subscribe, multi-cast),
- Integration scenarios (required or expected sequence of BOD/message exchanges), and
- Information exchanges and/or business object updates for which transactional integrity is required.

5.1.2.1.1 Analysis Component Type Descriptions

This section provides a brief identification of the reference component types that might be specified for an application. Note that these should *not* be viewed as an all-inclusive list of types.

Presentation Component Types

Page – An individual graphical user interface screen presented to a user. On the World Wide Web, a page is a file notated with the Hypertext Markup Language (HTML). Note that in the analysis stage a Web page consisting of multiple frames could be identified as either a single page or a main page including other pages. How this is specified could be a function of whether individual frame content can be used elsewhere, the level of security placed on the frame content, etc.

e-Form - (electronic form) A computer program version of a paper form. E-forms can include associated programming to automatically format, calculate, look up, and validate information for the user. E-forms allow more focus on the business process or underlying problem for which they are designed (for example, expense reporting, purchasing, or time reporting). They can understand the roles and responsibilities of the different participants of the process and, in turn, automate routing and much of the decision making necessary to process the form.

Portlet – A small portal application designed to display specific content or a user interface to a Web application. A Portlet displays the content blocks on a portal Web page. Each Portlet has a predefined role. For example, retrieve the latest news bulletin, run a specific search engine, view stock quotes, serve HTML files, display a calendar, etc.

Page Controller – Describes the page navigation’s specified for the application’s user interface. It can be thought of as providing the state transition diagrams for the application’s user interface screens (pages / forms). Note that it is left to the designer as to how to implement this navigation. It can be purely through static HTML hard links or through JSP/Servlet generated HTML.

Data Access – This is included under presentation components only for those cases where a user interface might directly access an existing database for simple display of information. This would generally be accessed via a portal. Note that access to data managed by modernized and wrapped applications would be through the business components of those applications, not a data access component in the presentation layer.

Business Component Types

Application Controller (AC) – Provides control, routing, and sequencing of requests for services of the application, generally as requested by an end-user. It would typically invoke Business Service Components and/or Business Objects as required. For processing of messages/BODs, it is left to the designer as to whether the application controller will actually “receive” a BOD itself and then dispatch the BOD to the appropriate BSC or whether the BSC would directly “receive” the BOD without any application controller involvement. Also note that for less complex applications the BSC and Application Controller may be combined into one component.

Business Service Component (BSC) - Typically a coarse-grained component that provides the public interfaces for a business process. A Business Service Component

describes the business process, and is comprised of (invokes) business objects. Examples of Business Service Components include ordering, in-processing/out-processing, inventory, etc. The OAGIS BSR is defined in the context of a BSC; that is a BSC needs to be defined to provide the processing associated with one or more BSRs to receive and process a related BOD. For analysis, it is up to the analyst to decide whether a single BSC is specified to “handle” all BSRs or to specify multiple BSCs with each BSC handling one or more BSRs.

Business Logic Component (BLC) - Relative to analysis, a finer-grained component than the BSC. It implements the business logic for a major process step, a business entity, or business event. Examples include customer, bill of material, end of fiscal year, etc. Business logic components also encapsulate business data and provide the access mechanism to the business data.

In general, a business logic component should represent an independent business object that has independent identity and lifecycle, and is referenced by multiple business objects and/or clients. Dependent objects are objects that only have meaning within the context of another business object. They typically represent fairly fine-grained business concepts, like an address, phone number or order item. For example, an address has little meaning when it is not associated with a business object like Person or Organization. It depends on the context of the business object to give it meaning. Such an object can be thought of as a wrapper for related data. Dependent objects are typically not modeled during analysis.

Business Object Document (BOD) – The model used to communicate a request from the originating mission application to the destination mission application. Each Business Object Document includes supporting details to enable the destination mission application to accomplish the action. As defined by the Open Applications Group (OAG), a BOD is the application interoperability model for specific Business Area message definitions. A BOD provides a standard message format, independent of mechanism for sending and receiving the message. The BODs Control Area includes one of the OAG defined Business Service Request (BSR) that defines the action the sender wants the receiver to perform. The BODs Business Data Area includes a definition of the data, making it a self-describing message format, and one or more occurrences of the data values.

Interface Component (IC) - Provides (GCSS-AF style) services to access Legacy systems, external systems, or COTS/GOTS/NDI products that do not provide GCSS-AF style APIs. The interface component acts as a front-end to these systems to provide required interface or service to the system. The interface component’s services are implemented using GCSS-AF style APIs. Essentially they are a specialized case of a Business Service Component or an Application Controller.

Data Component Types

In an object-oriented analysis based development, data components typically are not specified. Pertinent (analysis level) data is specified as attributes of business logic components. However, there may be cases where specific data objects or schemas are important enough during the

analysis to be specified. This might be the case for interface components or “wrapped” Legacy applications where the existing database (schema) is the “contractual” interface with the Legacy application. If queries will be required against the data, a logical data model at the entity level is created.

Data Object (DO) – Manages the essential state of the business logic component including the persistent storage of this data. While generally not specified during analysis, a data object could be specified when it is critical that the mapping of a business logic component’s data to a database schema be understood early on in the development.

Data Model - A model of the data entities that are used in a business or other context and the identification of the relationships among these data entities. Typically an entity-relationship model is developed to capture this information. While a data model is generally not specified during analysis for data associated with business objects, a data model would be captured or developed when queries or interactions with or encapsulation of a Legacy database are required.

Data Files – (primarily flat files) Typically used to maintain log information. There also may be cases (most likely Legacy application interface) where an external interface might require data to be sent to or received from another application in a flat file. For analysis, the file specification should at minimum identify the data items to be maintained in the file.

5.1.2.2 Design

Note: This section does not contain guidance on how to transform an analysis model into a design. This is a software engineering discipline and skill that should be held by the user of this guide.

In designing an application, the designer/developer needs to define the design level (technical) components he/she needs to implement the analysis model (components) of the application. For GCSS-AF, reference design level component types have been identified for consideration in designing the application. These component types are categorized in Figure 20: Mission Application - Design Model View relative to the N-Tier model.

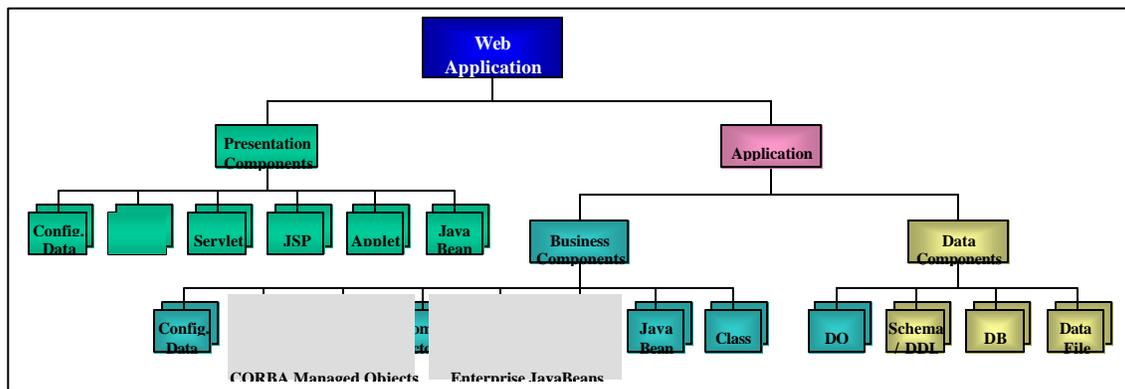


Figure 20: Mission Application - Design Model View

During design the mission application designer(s) would identify and define the technical components that are to be implemented to satisfy the analysis model and associated requirements. The components identified at this stage should identify design level components that are in fact separately installable, deployable components. They meet the GCSS-AF definition of a component as stated in the **GCSS-AF Application Developers Guide**:

*“A **software component** is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently, is self-contained, and is sufficiently specified to be useable by third parties.”*

It is intended that the components defined at the design level go down only to a level to define real component interactions and to which all functionality and required data can be allocated. Details below this are generally the province of implementation. At minimum, it is expected that components would be designed for:

- Analysis model presentation components.
- Analysis model business logic components. As such this would include:
 - Application controller(s),
 - BSCs for handling all the BSRs (BODs) provided by the application. In design, how the BSRs will be packaged shall be defined. That is, if a single BSC is used to implement all BSRs, than only one should be designed. If the BSRs are to be allocated out to separate BSCs, the design shall specifically represent this as separate BSCs. Where an *appropriate* BSC (BSR) has previously been developed and available for reuse, the design should clearly reference the component.
 - BOD classes should be designed if new or referenced if previously developed and available for reuse.
 - Business objects that provide the design for the business logic component. This could range from a single design level business object to decomposition into many business objects. This is a design decision left to the developer. However, the intent during design is to keep the design at the level of the component definition.
- During design, data objects for each business object should be developed to include allocation to specific backend data stores (RDBMS, messages, etc.).
 - As such, the databases required for business data persistence should also be defined along with the schema for these databases to support the business data.
 - The designer also needs to determine whether container managed persistence will be used (strongly recommended) or whether the business object will handle its own persistence (discouraged).
 - Files, including log files, should be identified during design to include data format of the information being accessed or logged. The method of access should also be identified.

- Components needed to support messaging requirements to send and receive BODs (or other messages). This includes:
 - Message listeners for the application.
 - Message containers for sending messages. (E.g. in WebSphere, a MQSeries Application Adapter based component for a specific message/BOD.)
- In design, the interactions with the Integration Framework and the use of it are initially identified. This includes the specific methods of a service, how and where a service is used, etc. This should not be interpreted to the extreme. For example, a designer should not try to show how an EJB container implements a transaction if the component required the EJB to be transactional.
- Information that needs to be defined to support configuration and reuse of the application components such as URLs, application/component name, information specific to the domain/locale/use that would map the application to a specific database instance, namespace, etc. Hard coding of such information should never be used.

Security Considerations

During design, the level and type of security (if any) should be identified for each designed component. Essentially what follows is a flow down from analysis components to design components.

The following should be included relative to each component:

- What information exchanges between components need to be encrypted?
- Does the requestor (or sender) of information need to be authenticated?
- Does the requestor (or sender) of information need to be authorized?
 - What are the roles that can access the component methods and with what privileges?
 - Who needs to be authorized, the initiating user or the requesting component?
 - What user presentation information needs to be restricted based on user role?
- To what files will audit information be logged and how will it be processed?

Refer to Section 6 Securing the Application for security guidance details.

5.1.2.2.1 Design Component Type Descriptions

This section provides a brief identification of the reference component types that might be part of and application design. It is these components that provide the top level of implementation of the specified application and should be directly derived from the analysis components.

Presentation Component Types

Hypertext Markup Language (HTML) – A markup language for hypertext documents on the Internet. HTML enables the embedding of images, sounds, video streams, form fields, references to other objects with URLs and basic text formatting.

Servlet - The Java platform technology of choice for extending and enhancing Web servers. Servlets provide a component-based, platform-independent method for building Web-based applications. And unlike proprietary server extension mechanisms, Servlets are server- and platform-independent. Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. Servlets can also access a library of HTTP-specific calls.

JavaServer Pages (JSP) – Technology uses XML-like tags and Scriptlets written in the Java programming language to encapsulate the logic that generates the content for the page. The application logic can reside in server-based resources (such as JavaBeans™) that the page accesses with these tags and Scriptlets. Any and all formatting (HTML or XML) tags are passed directly back to the response page. JavaServer Pages technology separates the user interface from content generation enabling designers to change the overall page layout without altering the underlying dynamic content.

Applet – A small program that can be sent along with a Web page to a user. Java Applets can perform interactive animations, immediate calculations, or other simple tasks without having to send a user request back to the server. Due to security reasons, Applet use is discouraged. If used, the use of signed Applets is required. Another reason for discouraging the use of Applets is that browser capabilities to support Applets are not consistent across vendors and may result in execution problems on the client.

JavaBean – A component (bean) specialized Java class that can be added to an application development project and then manipulated by the Java IDE. A bean provides special hooks that allow a visual Java development tool to examine and customize the contents and behavior of the bean without requiring access to the source code. Multiple beans can be combined and interrelated to build Java Applets or applications or to create new, more comprehensive, or specialized JavaBeans components. JavaBeans can also be passed between presentation components or between presentation components and business components.

Configuration Data – Information that needs to be defined to support configuration and re-use of the application components such as URLs, application/component name, information specific to the domain/locale/use that would map the application to a specific database instance, namespace, etc.

Business Component Types

CORBA Managed Object (CMO)– Loosely identified here as a CORBA business object that operates within the context of a container of an application server that provides automatic access to needed object services such as life cycle, persistence, transaction service, etc. All interactions with clients or CORBA managed objects are performed through the CORBA managed object. *(Note that while the term Managed Object is borrowed from IBM WebSphere Enterprise Edition terminology, it can be used generically to identify any non-EJB CORBA components running in any vendor's*

application server.) A CORBA managed object can be the specialization of any of the following component types.

Business Object (BO) – A CORBA object that implements the business logic for a major process step, a business entity, or business event. Examples include customer, bill of material, end of fiscal year, etc. Business objects also encapsulate business data (via data objects) and provide the access mechanism to the business data. In general, a business object should represent an independent business object that has independent identity and lifecycle, and is referenced by multiple business objects and/or clients. *(Has similarity to an EJB entity bean.)*

Application Object (AO) – A CORBA object that (might) implements the control, sequence, and transaction logic for access to business objects. As such it would provide the client access to business objects and provide any required business object access and update sequencing and transaction scope. *(Has similarity to an EJB session bean.)*

Home– Acts as both a factory and a collection for CORBA managed objects. A home provides methods to create, remove, and obtain a reference to a CORBA managed object instance. In addition, the home includes methods for finding existing CORBA managed object instances within the home. *(Has similarity to an EJB home.)*

JavaBean – Described in the Presentation Components Section above. JavaBeans can also be passed between business components or between business components and presentation components.

Enterprise JavaBeans (EJB) - The Enterprise JavaBeans component model logically extends the JavaBeans component model to support server components. Server components are reusable, prepackaged pieces of application functionality that are designed to run in an application server. They can be combined with other components to create customized application systems. Server components are similar to development components, but they are generally larger grained and more complete than development components. Enterprise JavaBeans components (enterprise beans) **cannot** be manipulated by a visual Java IDE in the same way that JavaBeans components can. Instead, they can be assembled and customized at deployment time using tools provided by an EJB-compliant Java application server. Two types of EJBs are specified.

Session Bean (SB) - Used to implement a business object that holds client-specific business logic. The state of such a business object reflects its interaction with a particular client and is not intended for general access. Therefore, a session bean typically executes on behalf of a single client and cannot be shared among multiple clients. A session bean is a logical extension of the client program that runs on the server and contains information specific to the client. In contrast to entity beans, session beans do not directly represent shared data in the database, although they can access and update such data. The state of a session object is non-persistent and need not be written to the database. A session bean is intended to be Stateful. However, the Enterprise JavaBeans specification

allows stateless session beans as a way to provide server-side behavior that doesn't maintain any specific state.

Entity Bean (EB)- Represents an object view of business data stored in persistent storage or an existing application. The bean provides an object wrapper around the data to simplify the task of accessing and manipulating it. This object interface lends itself to software reuse. For example, an entity bean representing user account information can be used by order management, user personalization, and marketing in a uniform way. An entity bean allows shared access from multiple clients and can live past the duration of client's session with the server.

Home – Provides methods to create, remove, or obtain a handle to an enterprise bean instance. In addition, the home interface of an *entity* bean provides methods for finding existing entity bean instances within the home.

Class - A template definition of the methods and variables of a particular kind of object. Most design elements, other than helper classes identified during design, can be expected to be other than “pure” classes (of a class library). One major exception to this is the BOD class.

Configuration Data – See description above under Presentation Component Types.

Data Component Types

Data Object (DO)– Manages the essential state of the business object including the persistent storage of this data. Generally at least one data object would be specified to map a business object's data to database entities and/or attributes. Data objects are only applicable to managed objects such as the EJB entity beans or CORBA managed object.

Schema / DDL – An outline or a plan that describes the tables, records, and the relationships existing in the view. The overall design of the database is called the database schema. A database schema includes such information as:

- Characteristics of data items such as entities and attributes.
- Logical structure and relationship among those data items.
- Format for storage representation.
- Integrity parameters such as authorization and backup policies.

At the design level of database abstraction all the database entities, attributes, and the relationships among them are included. One conceptual view represents the entire database. It describes all the records and relationships included in the conceptual view and, therefore, in the database. This schema also contains the method of deriving the objects in the conceptual view from the objects in the internal view.

The description of data at this level is in a format independent of its physical representation. It also includes features that specify the checks to retain data consistency

and integrity. Data Definition Language is generally used to capture the schema; although if a data model is employed it can be used to generate the DDL.

Database – The physical database in which data objects are stored or persisted. It is a design decision as to how many databases would be required to maintain the data for an application. Where an application may access an existing database, that database should be included by reference in the design.

Data Files – (primarily flat files) Typically used to maintain log information. There also may be cases (most likely Legacy application interface) where designs might require data to be sent to or received from another application in a flat file. Data structure and formats for flat file content is defined during design.

5.1.2.2.2 Analysis to Design Component Type Mapping

This section provides a mapping of analysis component types to design component types. That is, the type(s) of design components are generally used to provide a design for an analysis component type is identified. These mappings are provided as guidelines; the specific design component type(s) used is a function of the analysis component requirements, the software language used, the distributed component model selected (e.g. J2EE, CORBA), the skill and preferences of the designer, etc. Table 7: Presentation - Analysis Component Type to Design Component Type Mapping, Table 8: Business Logic - Analysis Component Type to Design Component Type Mapping, and Table 9: Data - Analysis Component Type to Design Component Type Mapping identify the possible design component types that might be used to provide a design for an analysis component type. Note that *italicized* design components are those that are the least likely to be defined during design.

Table 7: Presentation - Analysis Component Type to Design Component Type Mapping

Analysis Component	Design Component	Use Comments / Notes
Page	HTML	Generally for static content
	Servlet	Primarily for dynamic content
	JSP	For dynamic content
	Applet	For interactive animations, immediate calculations, or other simple client local tasks. Discouraged due to security issues.
	JavaBean	Reusable component processing, information passing
Form	HTML	Generally for static content
	Servlet	Primarily for dynamic content
	JSP	For dynamic content
	Applet	For interactive animations, immediate calculations, or other simple client local tasks. Discouraged due to security issues.
	JavaBean	Reusable component processing, information passing
Portlet	HTML	Dependent upon portal product support; static gadget display.

	Servlet	Dependent upon portal product support; dynamic gadget display and access to other services and applications.
	JSP	Dependent upon portal product support; dynamic gadget display.
	JavaBean	Reusable component processing, information passing
Data Access	Servlet	Provide access to external databases.
	JavaBean	Data Access Bean
Page Controller	Servlet	Typical and preferred use to implement page controller; provides most flexibility and capabilities.
	JSP	Can be used for more simple control requirements.
	JavaBean	Reusable component processing, information passing

Table 8: Business Logic - Analysis Component Type to Design Component Type Mapping

Analysis Component	Design Component	Use Comments / Notes
Business Service Component (BSC)	Application Object	For CORBA, non-J2EE implementations
	Session Bean	For J2EE implementations
	Home	Namespace location importance
	Message Listener	Asynchronous trigger of BSC on message arrival.
	Queue	To receive incoming BODs for BSR
Business Logic Component (BLC)	Queue Manager	To manage defined queue
	Business Object	For CORBA, non-J2EE implementations
	Session Bean	For J2EE implementations
	Entity Bean	For J2EE implementations
	<i>JavaBean</i>	<i>Reusable component processing, information passing</i>
	<i>Class</i>	<i>Generally not expected during design except for helper classes identified during design.</i>
	Home	Namespace location importance
	Data Object	For CORBA, non-J2EE implementations
Business Object Document (BOD)	Schema / DDL	For defined Data Objects
	Database	For defined schemas
Business Object Document (BOD)	Class	Classes to build and parse BODs and to populate and extract BOD information.
	DTD	Document Type Definition for the XML BOD.
Interface Component (IC)	See BSC (above)	An IC is essentially a specialized BSC or Application Controller. However, JavaBeans may be very apropos as either an Access Bean or for translation between the GCSS-AF "front-end" and the Legacy "back-end".
Application Controller	Application Object	For CORBA, non-J2EE implementations
	Session Bean	For J2EE implementations
	<i>JavaBean</i>	<i>Reusable component processing, information passing</i>

Table 9: Data - Analysis Component Type to Design Component Type Mapping

Analysis Component	Design Component	Use Comments / Notes
Data Object	Data Object	For defined data objects and expanded as required
		For defined data objects
Data Model	Schema / DDL	The data model from analysis would be of an existing database or for query support.
Data File	Data File	Extends the identification of the file content to include structure and format.

Note: Data component in analysis are generally not required or produced.

5.1.2.3 Implementation

Note: This section does not contain guidance on how to expand a design into an implementation. This is a software engineering discipline and skill that should be held by the user of this guide.

In implementing an application, the designer/developer expands the design level. Essentially implementation results in decomposition of the design components into additional components of various types. For example, a session bean from design could result in an implementation consisting of the session bean itself, plus a JavaBean, multiple classes (in a class library), and a property file. No guide to what may result can be provided; the specific implementation is a function of the design component itself, the software language used, the distributed component model selected (e.g. J2EE, CORBA), the skill and preferences of the designer, etc.

For GCSS-AF, reference implementation level component types have been identified for consideration in designing the application. As implementation is an extension of design, the component types are in fact the same component types identified for design and categorized in Figure 20: Mission Application - Design Model View plus some minor additions.

5.1.2.3.1 Implementation Component Type Descriptions

As previously stated, the component types are the same component types identified for design and plus some minor additions.

These *additional* component types are:

Presentation Component Types

Property File – A file that would typically contain configuration data such as that described for the configuration data component in design. Data such as URLs, application/component name, database name, information specific to the domain/locale/use, etc. could be retrieved from the property file(s) to allow an application/component to automatically configure itself or its connections.

JAR File – (**J**ava **A**rchive) is a platform-independent file format that aggregates many files into one. A JAR file is a convenient way of packaging together a set of class files to

simplify **configuration management and distribution of software**. Multiple Java **components** (.class files, images and sounds) can be bundled in a JAR file. The JAR format also supports compression, which reduces the file size. In addition, individual entries in a JAR file can be digitally sign to authenticate their origin.

JAR files are especially valuable for presentation purposes; they were originally developed in support of Applets. Multiple Java Applets and their requisite components (.class files, images and sounds) can be bundled in a JAR file and subsequently downloaded to a browser in a single HTTP transaction, greatly improving the download speed.

Business Component Types

Property File – See description above under Presentation Component Types.

JAR File – See description above under Presentation Component Types. With respect to Business Component Types, JAR files are used only for maintaining classes.

Data Component Types

Persistent Object (PO) - encapsulates the embedded SQL statements needed to insert, update, delete, and retrieve the essential state to and from the data store. The Persistent Object contains the same attributes as the Data Object; so the mapping from Data Object to Persistent Object is very straightforward. By delegating the (SQL) communication with the data store to Persistent Object, the Data Object 's own code can be kept understandable and clean. It also allows the same Data Object to be used with different back-end data stores, as a different Persistent Object can be provided for different data stores.

Note: While this Persistent Object definition is borrowed from IBM WebSphere Enterprise Edition terminology, it provides a desirable mechanism to isolate the actual data store from the Data Object independent of the actual application server or execution environment employed.

5.1.2.3.2 Design Component to Implementation Component Mapping

This section provides a mapping of design component types to implementation component types. Unlike the analysis to design mapping, implementation is essentially an expansion of the design components into additional components of various type; the actual types are a function of the design component requirements, its complexity, the software language used, the distributed component model selected (e.g. J2EE, CORBA), the skill and preferences of the developer, etc.

Table 10: Design Component Type to Unique Implementation Component Type Mapping identifies **only** the design to implementation component type mappings for those implementation component types **not already identified for design**.

Table 10: Design Component Type to Unique Implementation Component Type Mapping

Design Component	Implementation Component	Use Comments / Notes
Presentation Component Type Mappings		
Configuration Data	Property File	Configuration data generally held in one or more property files for the application or component
Classes, JavaBeans, Applets	Jar File	May be packaged in JAR file(s)
Business Component Type Mappings		
Configuration Data	Property File	Configuration data generally held in one or more property files for the application or component
Classes, JavaBeans	Jar File	May be packaged in JAR file(s)
Data Component Type Mappings		
Data Object	Data Object	In implementation, more than one data object may result from a design data object.
	Persistent Object	One per implemented data object

5.1.2.4 Component Integration Within An Application

The application developer needs to keep in mind that the ultimate goal is that applications can be created by integrating multiple (OAGIS-type coarse grain) business components. As such, components at the level of the OAGIS-coarse grain business components (i.e. the GCSS-AF Business Service Component) that comprise an application should follow the same guidelines as described for Application Integration. Refer to 5.1.3 Application Integration.

For integration all other identified components, GCSS-AF places little restrictions. The integration employed will be a function of the nature of the components, the software language used, the distributed component model selected (e.g. J2EE, CORBA), the skill and preferences of the designer, etc.

It is suggested that the integration methods chosen maximize the use of the application server capabilities provided by the Integration Framework. Some examples of this include:

- Using containers to manage persistence instead of integration “by hand” using SQL/ODBC/JDBC directly.
- Configuring containers to manage transactions instead of “coding in” begins, commits, and rollbacks, using container managed.
- Using container managed messaging instead of “coding to” the messaging interfaces themselves.

5.1.3 Application Integration

Application integration is one that results in an interaction or information exchange between two or more applications. GCSS-AF has specified that the Open Applications Group (OAG) approach be employed for this. Refer to <http://www.openapplications.org/>. This basically means that application-to-application interface is accomplished through Business Object Documents (BOD) typically sent using the messaging paradigm.

Application integration also requires that applications be able to locate the application (for either delivery of messages or application method invocation) appropriate to the domain for which the application is instantiated. To support this application namespace and naming conventions are provided.

Note: GCSS-AF has specified that the shared data not be employed for the integration of applications. That is, a database should not be employed as an information exchange mechanism between applications. There are numerous reasons for this not the least of, which is the tight coupling this imposes between applications. Shared data poses a significant maintenance issue when one of the applications needs to change its database design or even replace it as all other interfacing applications also need to be changed.

5.1.3.1 Business Service Requests

A Business Service Requests (BSR) shall be implemented by the application to process a BOD. As previously identified, The BSR is provided in a Business Service Component. Essentially the BSR accomplishes the necessary BOD processing. Depending upon the processing required, this might be accomplished by the BSR itself or by invoking other business components.

5.1.3.2 Business Object Document Delivery Mechanisms

There are two supported mechanisms for delivery of messages/BODs. The prime, recommended approach is through asynchronous messaging where the BOD is sent to a queue and then delivered to the target application(s). The other mechanism is by direct method invocation. The latter should only be used when it is absolutely necessary that the target application update its data inside the same transaction as the sender. These are briefly described in the following sections.

It should be noted that in the GCSS-AF enterprise there is no assurance as to where applications will be hosted or that they will remain on the same host or even within the same processing center throughout the applications lifetime. This is especially true given the Air Force Expeditionary Forces concept and associated deployment of applications. This is one of the drivers to specifying messaging as the prime mechanism for application integration.

5.1.3.2.1 Messaging (Asynchronous)

This is the specified mechanism for delivery of BODs for application integration. The Integration Framework supports point-to-point, multicast, and publish/subscribe means of message delivery. The current IF provided *publish/subscribe* capability is based on topics identified when publishing a message or subscribing to a message. It does not encompass content

based routing. While messaging is asynchronous communications, it supports assured delivery of a message. This means that once queued, the developer can be assured that the message will ultimately be delivered to the receiving application(s).

As messaging is asynchronous with applications de-coupled from one another through message queues, support for two-phase commit of data updates across applications is limited to inclusion of message queuing or de-queuing within a transaction context. While this may seem inadequate to developers steeped in earlier technology and approaches based on the two-phase commit paradigm, the availability of assured delivery with messaging has been proven to address all but the most critical data updates.

The design and implementation of the messaging associated with an application require the definition, design, and implementation of:

- Message listeners to trigger the Business Service Components
- Queues for the messages
- Queue managers and channels to manage movement of messages in and out of the queues
- *Publish/subscribe* brokers for published messages.

This might include use of existing ones or creation of new ones. **Note that this requires an understanding of the environment in which the application will be deployed.** Refer to Section 4.2.2 Messaging for details on messaging within the Integration Framework.

5.1.3.2.2 Method Invocation (Synchronous)

In order to support the situation when it is absolutely necessary that the target application of a message (BOD) update its data inside the same transaction as the sender, the Integration Framework also allows a message to be passed as a parameter in a method call. As such this provides the ability to encompass the data updates of both sender and receiver to be within the scope of the same transaction. While this is allowed, it should only be employed when absolutely necessary as its use:

- Results in a tight coupling of applications that in a distributed component environment is undesirable
- Requires that all underlying transaction services (CORBA OTS) either be branded or from the same vendor to assure interoperability of transaction services
- Can be unreliable over wide-area networks.

Refer to Section 5.4

5.1.3.3 Application Namespace and Naming Conventions

The Air Force enterprise is extremely large and diverse with many different site/locales or organizations requiring (application) services and information that shall be either individually or separately managed. This means that for some applications, the same application may need to be deployed (or instantiated) for each site/locale or organization if the information managed by the application needs to be separately managed. It may also be required that, depending upon the application and managed information, the applications be deployed at different processing centers (or even sites/locales). As such, it is essential that a means be identified to allow:

- Users to connect to the (deployed) application appropriate for the user's role, organization, or (site/locale) assignment
- Applications to send messages to the (deployed) application appropriate for the domain, organization, or site/locale for which it is processing information
- Applications invoke methods of the (deployed) application appropriate for the domain, organization, or site/locale for which it is processing information.

To accomplish this it is important that naming services, directory services, and messaging items (queues, queue managers) employ conventions to allow routing of user requests, messages, and invocations to the application appropriate to application and requested service. Figure 21: IF Naming Conventions Relative to Application Interaction illustrates where these conventions might be used.

The conventions of note and illustrated in Figure 21: IF Naming Conventions Relative to Application Interaction include:

Application Namespace

The namespace implemented within the Naming Services for CORBA and EJB components/objects provides for deployment of the same application type in support of different sites/locales or organization of the Air Force Enterprise. This is critical to forming properly scoped searches within the naming service(s). The IF has defined the following convention to be employed by the IF.

-- *Product Standard Structure*
----- *Server Group or Server* to represents site/locale or organization
----- *Applications*

The key level (above) relative to naming conventions is the Server Group or Server. Its name, where applicable, shall represent either the site/locale or organization and the application. The convention selected is to preface the key word Servers (for Server Group) or Server with LocationApplication. Refer to Section 5.5.2.8 Naming Conventions and Services for details. Note that this convention also takes into account the need to be able to provide workload management and fault tolerance and/or recovery as well.

A location name/value pair, initially passed in the HTTP Header, is provided to allow dynamic determination of the appropriate Server Group or Server.

Note: While this convention was defined with the WebSphere Application Server in mind, it should be, in some form, usable across different vendor application servers. While server/server groups may not be the appropriate entity for other application servers, a distinction in the name service that can be employed to implement site/locale or organization should be available.

Security Services Application Object Namespace

Maintains the information required to make (explicit) access control checks to application (modules, interfaces, and methods). As with the Application Namespace, this namespace also provides for deployment of the same application type in support of different sites/locales or organization of the Air Force Enterprise. The IF has laid out this namespace as identified in Section 5.3.2.2.6 Access Control.

A qualifier name/value pair, initially passed in the HTTP Header, is provided to allow dynamic determination of the appropriate node in the Application Object Namespace. As the current IF requires these access checks to be explicit, it is critical that the application making the access check, ensure that the Application Object Namespace node for which the access check is made corresponds to the (Server Group or Server) node in the Application Namespace in which the application is “running”.

Messaging Namespace (Conventions)

The naming convention employed for messaging provides for unique message queue names to route to the correct deployed application for cases where the same application type is multiply deployed in support of different sites/locales or organization of the Air Force Enterprise.

The name, where applicable, shall represent either the site/locale or organization and the application. The convention selected is to preface the type of queue with ***Application.Location*** or in the case of publish/subscribe to provide unique streams, ***Location.Default.Stream*** (for each required location or organization) to which applications can publish or subscribe to messages. Refer to Section 5.5.2.8 Naming Conventions and Services for details of the naming conventions.

A location name/value pair, initially passed in the HTTP Header, is provided to allow dynamic determination of the appropriate message queue or stream.

HTTP Header Location and Qualifier Name/Value Pairs

The IF Menu System was specifically designed to ensure that the appropriate location and qualifiers were passed in the HTTP Header where the user selected a link to an application supporting a specific location or organization. This menu system, along with the supporting Directory Service maintained information is detailed in Section 6.3.1 Planning for Access Control

Note: When other than the IF Menu System is utilized to access applications requiring location (and qualifier) distinction, it is up to the user interface (approach) to provide the location and qualifier information in the HTTP Header. As such, the AF Portal approach shall also comply with this requirement; otherwise the issues addressed by the IF Namespace and Naming Conventions shall be re-addressed. Hard coding of information and “creation” of separate applications for each unique location or organization should **not** be considered an option.

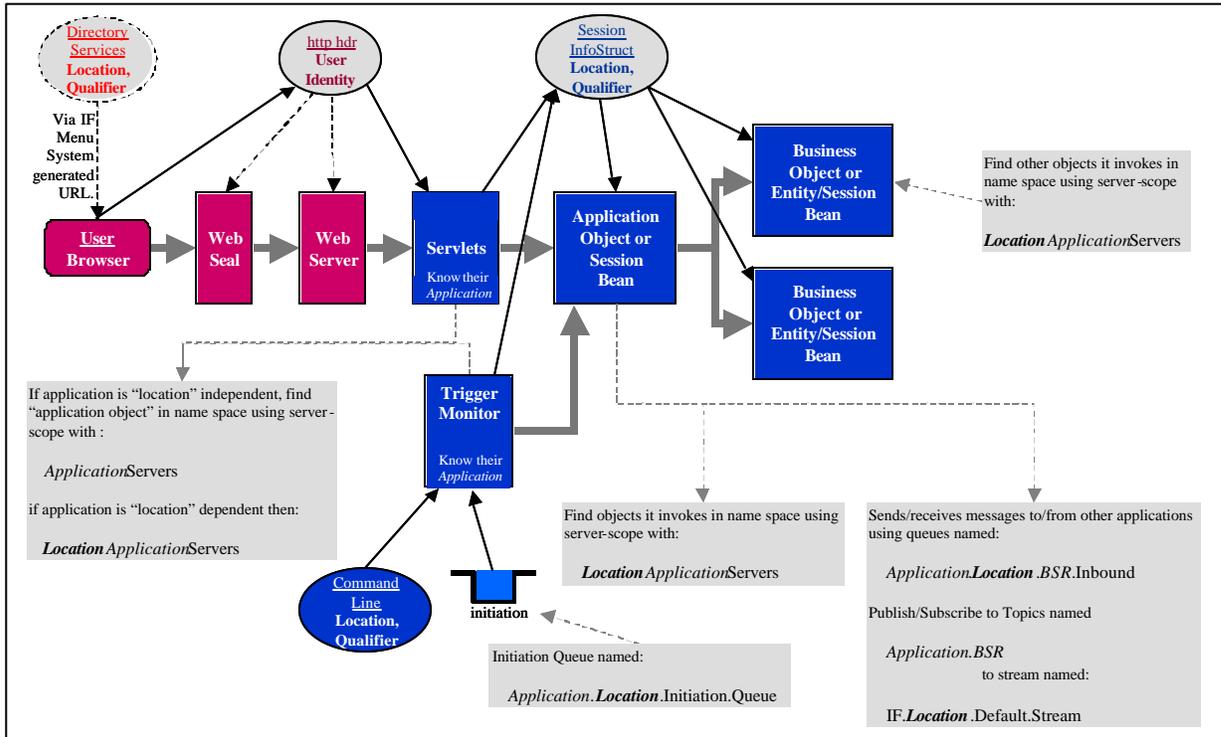


Figure 21: IF Naming Conventions Relative to Application Interaction

The correlation’s between these various namespaces and conventions is illustrated by example in Figure 22: Namespace and Naming Conventions Mapping Example. In the example, the locations configured are *enterprise* and *base-x* and the qualifiers are */GCSS-AFAPPS/ILS /enterprise* and *GCSS-AFAPPS/ILS /base-x*.

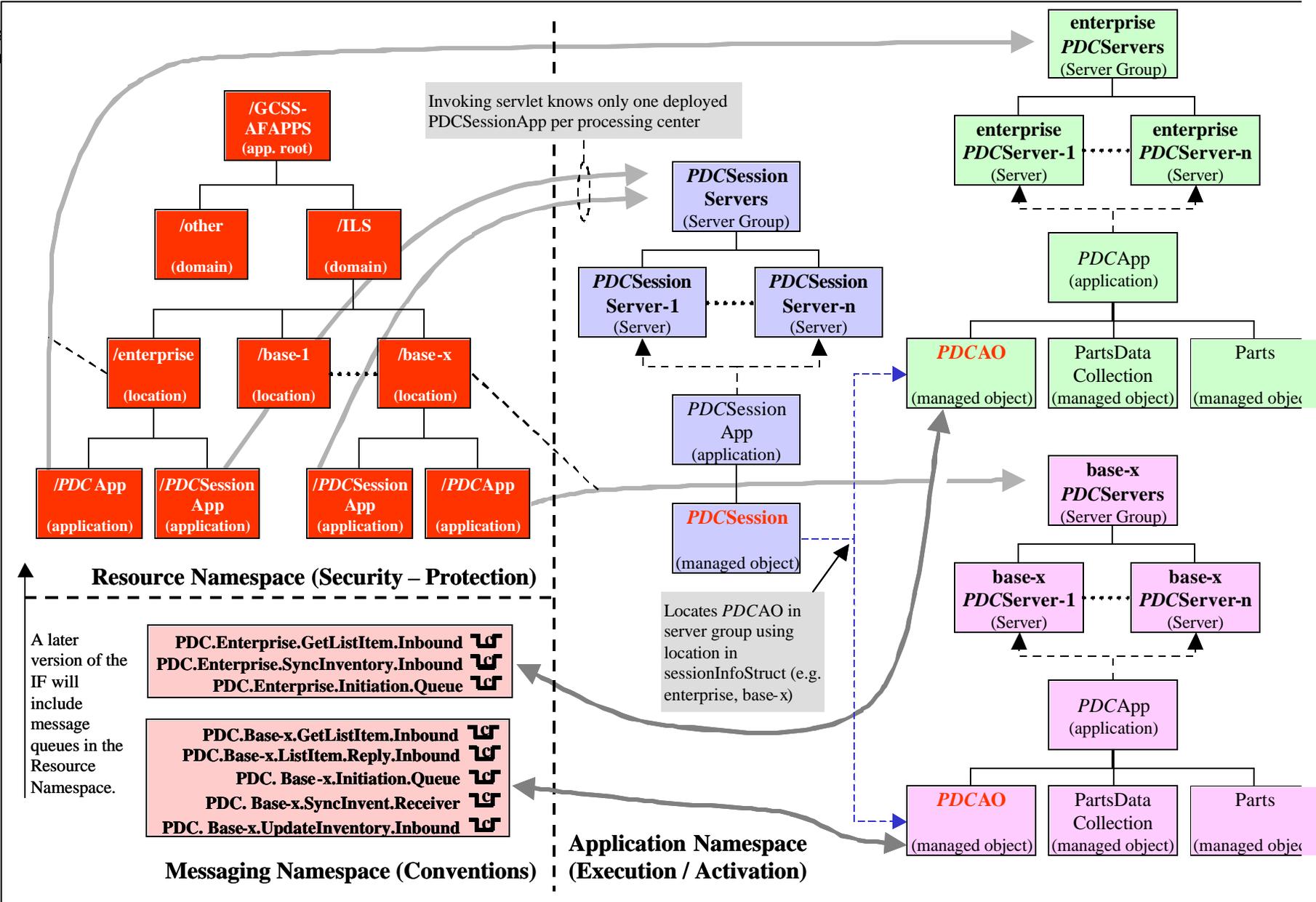


Figure 22: Namespace and Naming Conventions Mapping Example

5.1.3.3.1 “Routing” Requirements Example

The following example (*implemented by the IF test components and corresponding lab configuration*) is provided to illustrate the need and use of the “routing” capabilities provided by the IF. This example, illustrated in Figure 23: IF Routing Relative to Application Interaction, consists of the following elements:

- An *imaginary* enterprise consisting of **three** sites: Base-1, Base-3, and a Headquarters. There are also **two** geographic regions defined. Region-1 includes Base -1 and Base -3 and Region-2 includes the Headquarters.
- **Seven** applications of **three** service categories; the service categories represent an enterprise wide service, a regional service, and a site-specific service. The applications are:
 - Parts Data Collection (PDC) that maintains total identity of all parts and available quantities at a site. There are **four** deployed PDC applications (each with its own database); one for each site of the *imaginary* enterprise and **one** maintaining the total identity of all parts in the *imaginary* enterprise inventory.
 - A regional pseudo-Supply application through which parts can be ordered. There are **two** deployed pseudo-Supply applications (each with its own database); **one** for each region of the *imaginary* enterprise.
 - An enterprise-wide Requisitioning application through which all parts can be ordered.
- A user at a specific base requests a service of an application (*Requisitioning*) serving the whole enterprise. This request is made to the deployed application for that base. Refer to User Role Routing: Base Level in Figure 23: IF Routing Relative to Application Interaction.

The GCSS-AF IF provides a menu system that returns a menu to the user at login with URLs that point to applications specific to the users role and site/locale. Refer to Section 6.3.1 Planning for Access Control for details.

- The *Requisitioning* application in this example needs to interact with the deployed *PDC* application associated with the user’s base and the deployed *pseudo-Supply* application for the region in which the base is located. Refer to Message Routing: Region Level and Message Routing: Base Level in Figure 23: IF Routing Relative to Application Interaction.

Note: The GCSS-AF IF provides services for applications to determine the user’s role and site/locale. Refer to Section 5.3.2 Service Use for details. It also provides conventions by which message queues and topics/streams are named for sending or publishing of messages (BODs). Refer to Section 4.2.2 Messaging for details. Figure 21: IF Naming Conventions Relative to Application Interaction provides a quick view of these conventions and where they might be used.

- The deployed *PDC* application and the deployed *pseudo-Supply* application need to send a response to the enterprise *Requisitioning* application that sent them the original message.

Note: The GCSS-AF IF provides conventions for routing this response; it simply specifies that the originating sender identify the appropriate queue name as the response queue. Refer to Section 4.2.2 Messaging for details.

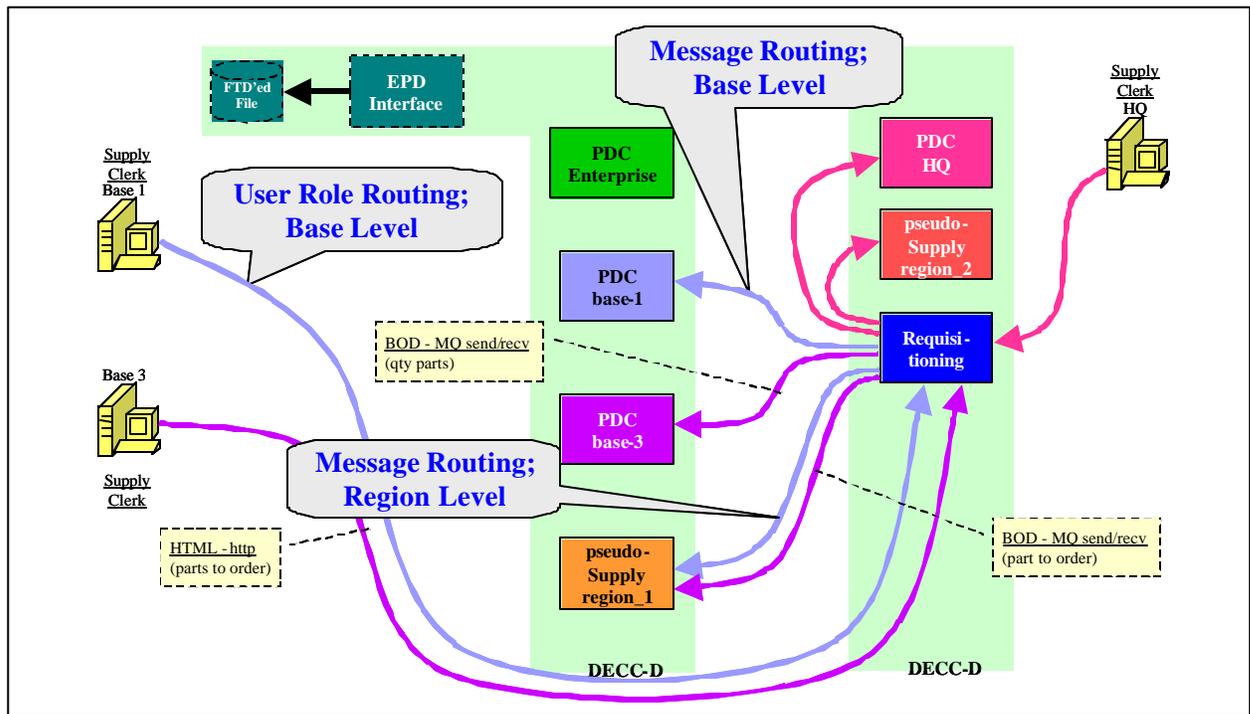


Figure 23: IF Routing Relative to Application Interaction

Not illustrated by the example is the use of namespace in the allocation and deployment of applications in the directory servers. The GCSS-AF IF also identifies a convention for the structure and use of namespace in the naming service(s) employed for placing EJB and CORBA components.

5.1.4 Application Security Responsibilities

The Integration Framework is required to provide end-to-end protection of GCSS-AF user-to-application, user-to-data, application-to-application, and application-to-data interactions. And it is required to do so in a consistent manner providing the ability to selectively configure (or use) only the level needed in order to minimize the performance impacts that result from the use of security services and features.

The following, as annotated in Figure 24: Application Security Integration Points and Considerations, describe at a top-level where application of security is specified along with the

nature of the security application. For these descriptions IF Security Services are generically identified and **not** the underlying COTS products and LMSI-O developed services. For detailed descriptions of security relative to application development using the Integration Framework, including the HTTP Header and **SessionInfoStruct** mentioned below, refer to Section 6 Securing the Application.

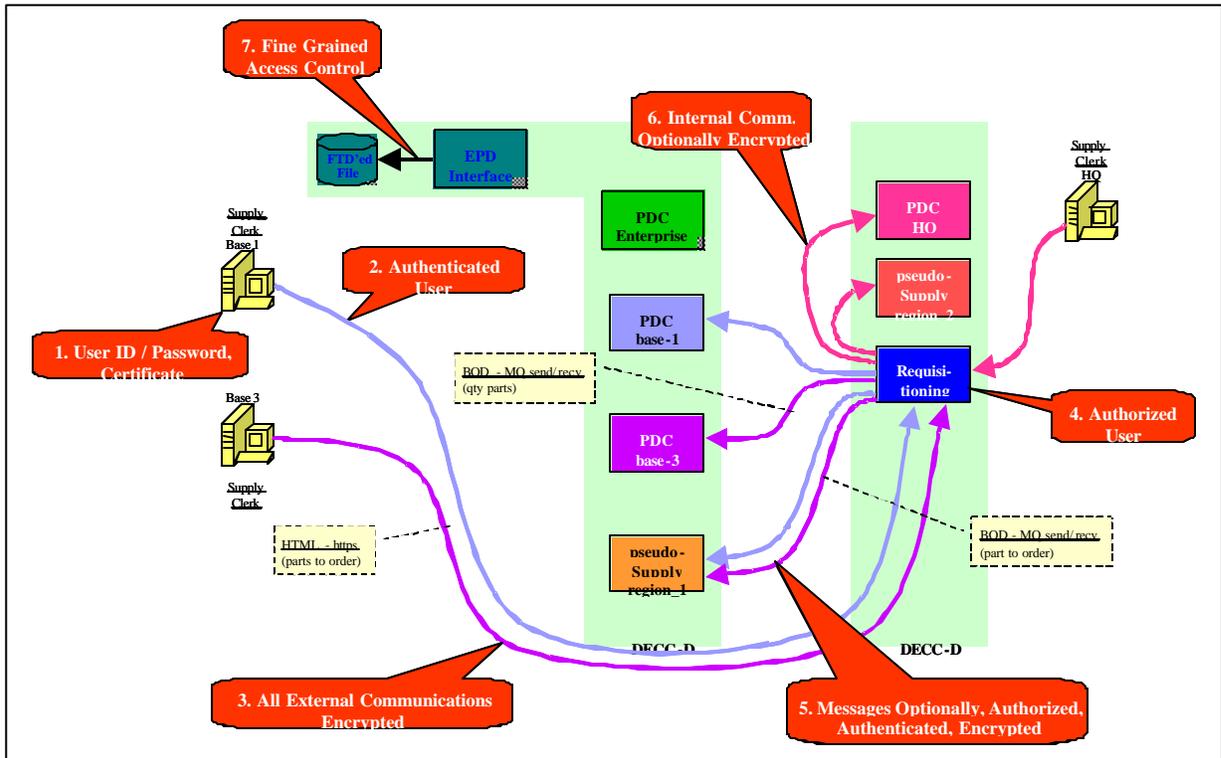


Figure 24: Application Security Integration Points and Considerations

1. **User ID / Password, Certificate** – For GCSS-AF, all users are required to login either using a *userID/password* or using a **DoD PKI certificate**. The login to GCSS-AF is provided through IF Security Services. **Note that through the current Integration Framework release, only *userID/password* is supported.** To support a single-sign-on, application user interfaces (including Portlets) and application components should, for all access checks (authorizations), make use of the security information passed in the HTTP Header or the IF provided **SessionInfoStruct**.
2. **Authenticated User** – The IF Security Services provides for authentication of the user, verifying the user is identified in the IF Security Services Directory. Note that this is essentially a configuration effort and not a programming effort.
3. **All External Communications Encrypted** – all communications are required to be encrypted outside of the protected enclaves provided by the processing centers. This includes not only Web (HTTPS) traffic but also messaging traffic and access to databases external to a requesting processing center. Encryption is accomplished through the use of

- HTTPS, VPNs for messaging, and ORACLE ASO for database access. Note that these are essentially configuration efforts and not programming efforts.
4. **Authorized User** – While only one access check is illustrated, in reality each component that makes up the application may make a separate access check. As an example, IF Security Services may first make an Access check to a user interface component. Then a Business Component invoked by the user interface component may make its own access check. In this example, the IF Security Services makes the initial access check based entirely on Security configuration data (e.g. Access Control Lists) without any programming requirement on the application developer. The second check, for the current IF, requires a programmatic check on the part of the Business Component using the information passed to it in the IF provided **SessionInfoStruct**. Note the need for this latter check as a programmatic check has been identified for eventual elimination by a more complete implementation of the CORBA Security Services.
 5. **Messages Optionally Authorized, Authenticated, Encrypted** – The current IF release employs the same security mechanisms as used by DISA for their (MQSeries) messaging security. Authorization is not provided by these mechanisms. The available encryption and authentication entails the configuration of a VPN (as required) and a DISA developed MQSeries channel security exit to provide encryption and authentication for messages sent between processing centers. Accomplishing this requires a configuration effort and not a programming effort.

Note Future Capability: A future release of the IF plans the incorporation of enhanced messaging security that provides for access control to message queues, the ability to digitally sign messages, and the ability to individually encrypt messages. This capability will be tightly integrated with the current IF Security Services, utilizing the same underlying directory services and ACL repositories.
 6. **Internal Communications Optionally Encrypted** – GCSS-AF Security Policy does not require encryption within a protected enclave. As such, the application of encryption within a protected enclave should only be used when absolutely necessary as encryption imposes significant performance impacts. Establishing encryption (SSL) is a configuration effort and not a programming effort.
 7. **Fine Grained Access Control** – Fine Grained Access Control (FGAC) as used here is the ability to individually protect individual attributes or objects managed by an application. Currently the IF does not provide unique services to implement. These services have been identified as a need for a later IF release. In the interim, capabilities of the database management systems provided, programmatic checks, and coarser-grained policy director checks are available for programmatic implementation.

As can be seen, the IF approach to security is to minimize the application of security through programming and maximize it through configuration.

5.1.5 Application and Integration Framework Integration Points

This section is intended to provide a combination overview and summary of the integration points between an application and the Integration Framework. Some of these points have previously been discussed as they pertain specifically to an application developer's responsibility while others have not been discussed, as they are more relevant to configuration, deployment, and/or administration of either the application or the Integration Framework itself. Figure 25: Integration Framework Interface and Configuration Integration Points illustrates these integration points. The components/blocks depicted in the figure are described in the following paragraphs.

Local Director(s)

Local Director represents the external network interface into the GCSS-AF enterprise. From a developer standpoint, this is nothing more than the HTTP/HTTPS input from or output to a client.

WebSeal (Proxies)

Provide user login, user authentication into GCSS-AF, establishes and maintains the (web) users session, and provides (user) access control to web resources. Just as important is that it establishes the user credential with which all "downstream" user authorization and access checks will be made.

Security Servers

Represent all the other security servers required to provide authorization, authentication, and security administration. It also maintains associated repository information such as user profiles, application profiles, GCSS-AF (application) menu information, and access control lists.

Web Servers

Provide the processing of http requests and serves HTML. It also includes plug-ins to (or for) Servlet engines to process JSPs and Servlets.

Servlet Engines

Provide for the processing of server-side presentation layer services through execution of JSPs and Servlets.

Application Servers

Provide for the processing of business layer services through execution of business logic components and associated messaging.

(Federated) Naming Service

Provides the directory services in which all GCSS-AF component "locations" can be found. The application server products provide this directory service(s). However if required, it is a development or deployment responsibility for any naming services federation or "registering" a component in multiple naming services.

Messaging Backbone

The underlying MQSeries messaging product through which all business components interchange Business Object Documents.

These descriptions are provided to better place in context the Integration Point identification and descriptions that follow. This information can also serve to identify the integration points that shall be addressed when considering replacing IF COTS products or adding alternative COTS products for specific functionality such as an Application Server.

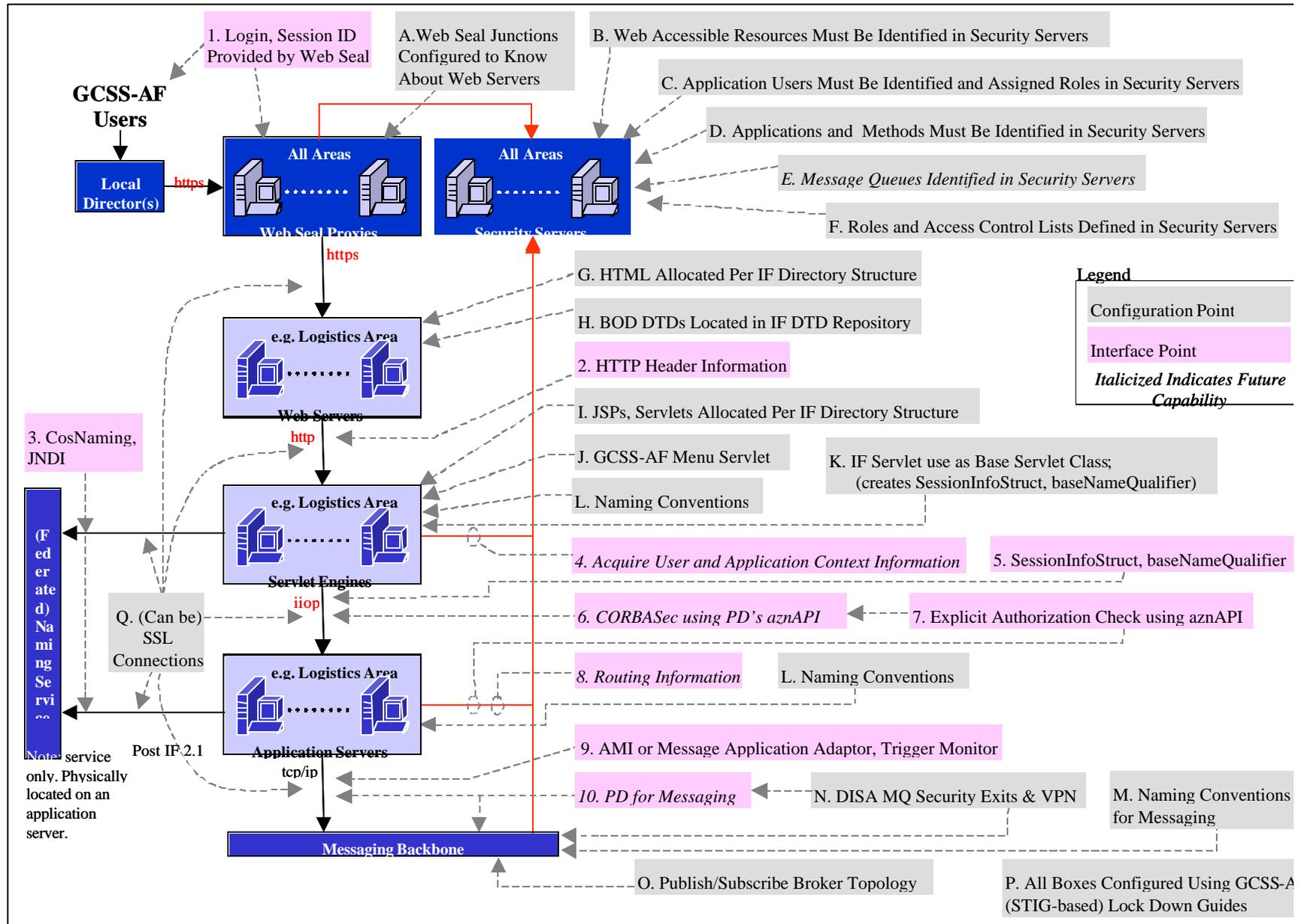


Figure 25: Integration Framework Interface and Configuration Integration Points

5.1.5.1 Interface Points

An Interface Point is defined as an integration point that actually implement an interface between the application and Integration Framework components and/or services.

These Interface Points include:

1. **Login, Session ID Provided by Web Seal** - Web Seal being the IF Security Services Web (http) entry into the GCSS-AF Enterprise provides the login and authentication of a user to establish a “session” and user identification intended to be employed throughout the user’s session.

There is no application development responsibility specifically associated with this. (However, security information passed in the http header as a result of this login is application development pertinent information that is discussed in a later item.)

Note that the IF currently provides for *userID/password* login; the use of certificates/smart cards has been identified by USAF for a future IF capability. Refer to Section 6 Securing the Application for further information.

2. **HTTP Header Information** - Includes the user credential created by the IF Security Services (WebSeal) and the appropriate location and qualifiers (as available using the IF Menu System). These are available for use by applications as described in section 5.1.3.3 Application Namespace and Naming Conventions. Servlets derived from the IF provided IFServlet base class automatically obtains this information. Other Servlets can access this information through the standard Java Servlet class specified services provided to access http header information.
3. **CosNaming, JNDI – CosNaming** is to be accessed by application components to create and/or locate components with which the component needs to interact. It is up to the application/component to form the proper namespace node for the desired component. The IF provides the location and qualifiers to assist in this as described in section 5.1.3.3 Application Namespace and Naming Conventions. Applications / components access these naming services using CORBA **CosNaming** or the standard J2EE JNDI services.
4. **Note Future Capability: Acquire User and Application Context Information – Maintained in repositories for user profile information and application information including information that identifies application associations. The later provides a means by which configuration information can be employed to allow an application to dynamically determine the identity of applications with which it needs to interface. Applications / components access these repositories through LDAP.**
5. **SessionInfoStruct, baseNameQualifier** – Available to applications to obtain user information that can be used to tag data in a database, against which to make access checks, and to form appropriate namespace searches or names. This assumes that the user

interface (controller) Servlet was derived from the IF provided IFServlet base class to automatically build these structures. Refer to 5.1.3.3 Application Namespace and Naming Conventions and 5.1.4 Application Security Responsibilities for further information including pointers to more detailed information.

6. **Note Future Capability: CORBASec using PD's aznAPI** – The means by which implicit authorization can be configured for access to a component (methods). This is a future capability; for the current IF, the explicit access check described below is required.
7. **Explicit Authorization Check using aznAPI** – is provided by the IF as both an interim means of providing access checks for component methods and as a means for providing application / components to make (programmatic) access checks on resources more fine grained than a method. Applications / components can make these checks using the aznAPI either using an IF provided or using the aznAPI directly. Configuration of appropriate information in Security Service repositories is also required. Refer to Section 6 Securing the Application for further information.
8. **Note Future Capability: Routing Information** – Maintained in a repository for application information including information that identifies application associations. The later provides a means by which configuration information can be employed to allow an application to dynamically determine the identity of applications outside of the same location and/or organization with which it needs to interface. Note that this is in addition to the information provided by the SessionInfoStruct, and baseNameQualifier that provide the means to map to applications within the same location and/or organization. Applications / components access this repository through LDAP.
9. **AMI or Message Application Adapter, Trigger Monitor** – provide the mechanisms by which applications / components interface with IF Messaging Services. Note that this includes the use of a Trigger Monitor also. Applications / components outside of the application server use the AMI and access it through the AMI APIs. Applications / components running in the application server build message components using the Message Application Adapter to interact with IF Messaging Services. In addition, the IF provides classes to allow applications / components to configure a message to be sent using the *publish/subscribe* paradigm. A Trigger Monitor to activate a component of the arrival of a message is provided for tailoring for specific use. Refer to Section 4.2.2 Messaging for further information.
10. **Note Future Capability: PD for Messaging** – Essentially an interceptor for MQSeries APIs that provides the ability to individually encrypt messages, and provide access checks to message queues, digitally sign messages. Applications / components access this capability the same as if they were invoking MQSeries APIs. Since the IF provides APIs and application adapters that perform the actual MQSeries API invocations, this should be hidden from the application. As there may be other specific security APIs required to be invoked prior to sending or receiving a message this is identified as an Interface Point and not a Configuration Point. Use of PD for Messaging also requires information to be configured in support of the capability.

5.1.5.2 Configuration Point

A Configuration Point is defined as an integration point that requires information to be “configured” in an Integration framework repository or requires configuration of IF employed products to be configured/deployed in a particular fashion.

These Configuration Points include:

- A. Web Seal Junctions Configured to Know About Web Servers** - All Web Servers “secured” by IF Security Services shall have junctions configured between the Web Seals and the Web Servers. While not an application development concern per se, the application analyst and/or “deployer” shall decide through which Web Servers the application will be accessible. Refer to Section 6 Securing the Application for further information.
- B. Web Accessible Resources Shall Be Identified in Security Servers** - All Web accessible resources, for which security at the URL level is desired, needs to have that URL configured in the IF Security repositories along with associated ACLs. Refer to section Section 6 Securing the Application for further information.
- C. Application Users Shall Be Identified and Assigned Roles in Security Servers** - All GCSS-AF users and the ir assigned roles shall be configured in the IF Security repositories. Refer to Section 6 Securing the Application for further information.
- D. Applications and Methods Shall Be Identified in Security Servers** - All components and their protected methods for which security is desired, needs to have that component and its methods configured in the IF Security repositories along with associated ACLs. Refer to Section 6 Securing the Application for further information.
- E. Note Future Capability: Message Queues Identified in Security Servers** - All message queues, for which security is desired, needs to have that queue configured in the IF Security repositories along with associated ACLs.
- F. Roles and Access Control Lists Defined in Security Servers** - All roles to which users (or components) will be assigned need to be configured in the IF Security repositories. All ACLs that will be placed on protected resources need to be configured in the IF Security repositories. Refer to Section 6 Securing the Application for further information.
- G. HTML Allocated Per IF Directory Structure** - HTML files should be allocated on Web Servers following the directory structure identified by the IF. Refer to Section 4.3.2.1 User Interface Common Facilities for further information.
- H. BOD DTDs Located in IF DTD Repository** - All DTDs shall be allocated in the IF DTD Repository provided by the IF in order for XML parsing services, as configured

for the IF, can find the DTD at the processing center where the parsing service is invoked. Refer to Section 5.3.2.3.3 DTD Repository for further information.

- I. JSPs, Servlets Allocated Per IF Directory Structure** - JSPs and Servlets should be allocated on Servlet Engines following the directory structure identified by the IF. Refer to section Section 4.3.2.1 User Interface Common Facilities for further information.
- J. GCSS-AF Menu Servlet** - Application information needs to be configured in the IF Security repositories in order for the IF Menu System to be able to present the application to an authorized user. Note that it is this configuration information that is used to ensure that the Web Browser provides the correct location information when an application is invoked. Refer to Section 6.3.1 Planning for Access Control for further information.
- K. IFServlet** - This IF base Servlet class is provided to application developers to provide the means to inherit the creation of the SessionInfoStruct and baseNameQualifier. In addition, it also implements the currently required use of the **ServletLoginHelper** to automatically log (WAS-AE) Servlets into the (WAS-EE) application server. *The latter will not be required in a future release of the IF.* Refer to Section 4.3.2.1.1.3 Servlet Engines for further information.
- L. Naming Conventions** - The IF specifies naming conventions in order to locate the correct (deployed) application based on location, site, and/or organization. The naming convention covers application/component names, message queue names, naming service namespace structure, and security directory namespace structure. Refer to section 5.1.3.3 Application Namespace and Naming Conventions for further information.
- M. Naming Conventions for Messaging** - The IF specifies naming conventions in order to locate the appropriate queue to send information to the correct application based on location, site, and/or organization. The naming convention covers message queue names, queue managers, and publish/subscribe topics and message stream. Refer to section 5.1.3.3 Application Namespace and Naming Conventions for further information.
- N. DISA MQ Security Exits & VPN (IF 2.1)** - Until the future incorporation of the Policy Director for Messaging product (described in Interface Point 9 above), messaging security is that currently implemented by DISA for their messaging activities. This essential consists of configuring in the MQSeries channel security exit developed by DISA, configuring in the assigned *userID* and *password* to the operating system. In addition, if encryption of external message traffic is required, it is required that a VPN be configured for the required message traffic. Refer to Section 6 Securing the Application for further information.

- O. Publish/Subscribe Broker Topology** - The IF has established a publish/subscribe broker topology and related naming conventions for the GCSS-AF Enterprise. This topology provides an enterprise broker and processing center brokers. In addition, publish/subscribe streams are established on a location, site, and/or organization basis. Applications employing publish/subscribe are expected to utilize this topology and conventions. Refer to Sections 4.2.2 Messaging and 5.1.3.3 Application Namespace and Naming Conventions for further information.
- P. All Boxes Configured Using GCSS-AF (STIG-based) Lock Down Guides** - As the IF security solution encompasses not only the IF Security Services but also computer and operating system configurations using the IF developed GCSS-AF Lock Down Guides that are based on the DoD STIGs. It is required that these guides be applied to all servers in order to meet security requirements. Refer to Section 6 Securing the Application for further information.
- Q. (Can be) SSL Connections** - The IF provides the ability to protect most inter-box connections using SSL. This is accomplished through configuration of the connections for which protection is desired. (Note that USAF and DISA have currently identified that connections within (firewall) protected enclaves do not require protection. Refer to Section 6 Securing the Application for further information.

5.1.6 Development Environment Notes

This section is intended to provide some general guidance relative to the development environment an application developer shall put in place. This will be discussed in the context of the reference development process discussed in section 5.1.2 Reference Application Development Process and the security requirements and integration points discussed in sections 5.1.4 Application Security Responsibilities and 5.1.5 Application and Integration Framework Integration Points.

Note: It is important to understand that GCSS-AF currently does not specify a specific development methodology, process, or development toolset. Information that follows should be viewed as reference information and is not mandated. While what follows may have the appearance of a waterfall approach, it is assumed that a spiral development process will most likely be employed with the development phases overlapping and repeated.

5.1.6.1 Analysis Phase Development Environment Notes

Table 11: Analysis Phase Development Environment Items identifies the recommended development environment items that an application developer should have in place for the analysis phase of development. Note that later phases will include many of the same development items and will reference this table for descriptions. Also, the prototype

development item description and details is deferred to the implementation discussion as tool employed for prototype development tools can be expected to be some of the same tools employed for actual development.

Table 11: Analysis Phase Development Environment Items

Development Environment Item	Current IF Tested Product(s)	Notes
		As applicable: <i>Development Process Item</i> (from Figure 18: Reference Mission Application Development Process)
Requirements Management	DOORS	<p>A requirement management tool is strongly recommended for maintaining, tracking, status of requirements and how they are met. This needs to include allocation to analysis components. <i>While not required, use of DOORS would allow better integration and merging of requirements across the enterprise.</i></p> <p>Requirements allocated to each specified analysis Presentation, Business, and Data component. Note that allocation can be at a higher level of abstraction than the specific component; i.e. allocation can be to a (UML) package that contains multiple components.</p>
UML Modeling Tool	Rational Rose, SourceSafe	<p>UML is the specified modeling language for GCSS-AF. GCSS-AF maintains a Rational Rose UML model of the GCSS-AF enterprise. It is expected that application developers will provide the required information and model to allow new application developments to be merged into this enterprise model. In addition, a configuration management capability needs to be provided for the model. While not required, use of Rational Rose and SourceSafe would allow better integration and merging of the UML models across the enterprise.</p> <p>All analysis Presentation, Business, and Data components are specified in the UML model. GCSS-AF has specified a partitioning of business models, analysis models, design models, and implementation models. The analysis model is developed during this phase.</p>
Data Modeling	ERWIN	For those developments where significant queries directly to backend databases are required, a data model should be developed. To a certain degree, this model can be expressed using UML. However, for complex data models, traditional data models should be developed.
	Rational Rose	For any required analysis Data component, an associated entity should be found in the data model. This entity should also be represented in the UML model.
Performance Modeling	OPNET	Depending upon the particular application being developed, a degree of performance modeling may be required during analysis. USAF has standardized on the use of OPNET for performance modeling. In addition, various performance models will be available for incorporation into an application's model to represent parts of the system not specifically part of the application being developed.
Prototype Development (<i>Integrated Development Environment Environment</i>)	Refer to Table 14: Integrated Development Environment Items	During analysis, it may be necessary to develop prototypes to validate concepts or approaches. At minimum, an IDE should be available for this. However, depending upon the scope of the prototype a full up IF environment could be required. (For details refer to the Integrated Development Environment notes in Table 14: Integrated Development Environment Items .)

Development Environment Item	Current IF Tested Product(s)	Notes
		As applicable: <i>Development Process Item</i> (from Figure 18: Reference Mission Application Development Process)
Performance Testing	WinRunner, LoadRunner, Sniffers	For prototypes for which performance data is required, test tools may be required to exercise the prototype and to capture pertinent performance data. USAF currently employs the WinRunner and LoadRunner test tools.
Documentation	Microsoft Office	Documentation of all types is produced during analysis. This includes document, spreadsheets, diagrams/graphics, etc. GCSS-AF has standardized on the Microsoft Office product suite for this purpose.
Software Development Folders (SDF)	Windows File System Structured By System Tree	GCSS-AF recommends that a system tree for the development be established along with a directory structure based on the system tree. Development artifacts are then to be allocated to nodes of the directory structure. GCSS_AF has "standardized" on Microsoft Windows as the environment in which development artifacts would be maintained.

5.1.6.2 Design Phase Development Environment Notes

Table 12: Design Phase Development Environment Items identifies the recommended development environment items that an application developer should have in place for the design phase of development. Note that for the most part, the development environment items are the same as those required for the analysis phase.

Table 12: Design Phase Development Environment Items

Development Environment Item	IF Tested Product(s)	Notes
		As applicable: <i>Development Process Items</i> (from Figure 18: Reference Mission Application Development Process)
Requirements Management	DOORS	During design, most of requirement management will consist of updates, corrections, and allocations. <i>Also refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items.</i> As a trace-ability between analysis components and design components is recommended in the UML model, it is not required to provide additional allocations to design components within DOORS.
UML Modeling Tool	Rational Rose, SourceSafe	During design, not only are the design model developed but the initial identification of the components (captured in the UML component model is developed. An early view of the deployment model may also be developed if sufficient information is available. Also refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items . All design Presentation, Business, and Data components are specified in the UML model. GCSS-AF has specified a partitioning of business models, analysis models, design models, and implementation models. The

Development Environment Item	IF Tested Product(s)	Notes
		As applicable: <i>Development Process Items</i> (from Figure 18: Reference Mission Application Development Process)
		design model is developed during this phase.
Data Modeling	ERWIN	Refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items .
	Rational Rose	For any required design Data component, an associated entity should be found in the data model. This entity should also be represented in the UML model.
Performance Modeling	OPNET	Depending upon the particular application being developed, a degree of performance modeling may be required during design to make trade offs in approach. Also refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items .
Prototype Development (<i>Integrated Development Environment</i>)	Refer to Table 14: Integrated Development Environment Items	During design, it is expected that prototypes will be developed to validate approaches. At minimum, an IDE should be available for this. However, depending upon the scope of the prototype a full up IF environment could be required. (For details refer to the Integrated Development Environment notes in Table 14: Integrated Development Environment Items .)
Performance Testing	WinRunner, LoadRunner, Sniffers	Refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items .
Documentation	Microsoft Office	Refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items .
Software Development Folders (SDF)	Windows File System Structured By System Tree	Refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items .

5.1.6.3 Implementation Phase Development Environment Notes

Table 13: Implementation Phase Development Environment Items identifies the recommended development environment items that an application developer should have in place for the implementation phase of development. During this phase an Integrated Development Environment (IDE) is the most critical to the successful implementation of an application. It also poses the most significant impact to a developer that chooses to utilize a development environment other than that tested and utilized by the Integration Framework. It is therefore strongly recommended that if a different IDE is employed, the developer very early in the development determine the approach for utilizing this (different) IDE relative integration, test, and deployment into GCSS-AF (and the IF).

Table 13: Implementation Phase Development Environment Items

Development Environment Item	IF Tested Product(s)	Notes
		As applicable: <i>Development Process Items</i> (from Figure 18: Reference Mission Application Development Process)
Requirements Management	DOORS	During implementation, very little requirement activity is expected. Most of requirement management will consist of updates, corrections, and allocations. <i>Also refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items.</i>
UML Modeling Tool	Rational Rose, SourceSafe	If an implementation UML model is required the following applies. Note Future Capability: For implementation, it is left to the developer to decide on the course of implementing the design specified in the design model portion of the UML model. Two main paths are available. 1. (If using Rose) develop the initial implementation in the UML model and derive the initial implementation (code) from the model and then use round trip engineering for updating the implementation model. 2. Develop the implementation using the IDE and extract the implementation model from the code. Also refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items .
Data Modeling	ERWIN	If a data model is required the following applies. A data model would be developed which identifies all entities, their attributes, and relationships between entities. The DDL would then be derived from the data model. Note that DDL can be extracted either from an ERWIN data model or from the classes identified in Rose.
	Rational Rose	Also refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items .
Performance Modeling	OPNET	During implementation or at minimum immediately following integration and testing, an updated model of the application should be completed to validate deployment requirements for the application. Also refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items .
Integrated Development Environment (IDE)	Refer to Table 14: Integrated Development Environment Items	During implementation a full up IF environment would be required. However, as implementation itself has various phases not all of the IDE may be required until later in the implementation (such as final implementation) when integration of components is underway. Table 14: Integrated Development Environment Items presents this in more detail.
Software (CM) Repositories	PVCS	During implementation it is critical that software is developed under configuration management. It is recommended that the software CM repository be structured (including node names) in such a fashion as to allow ease of correlation between the CM repository nodes and the SDF nodes. While the IF currently uses PVCS, use of PVCS is not a recommendation, the IF itself is considering a change to a different CM tool.
Testing		During implementation, testing at various levels will be necessary. This encompasses unit tests, component tests, application tests, and system tests, security tests, and performance tests. As with the IDE, not all of the IDE may be required until later in the implementation (such as final implementation) when integration of components is underway.
Documentation	Microsoft Office	For Java based components, it is recommended that the code be annotated to allow extraction of JavaDocs. Also refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items .

Development Environment Item	IF Tested Product(s)	Notes
		As applicable: <i>Development Process Items</i> (from Figure 18: Reference Mission Application Development Process)
Software Development Folders (SDF)	Windows File System Structured By System Tree	Note Future Capability: Also refer to Analysis Phase notes in Table 11: Analysis Phase Development Environment Items .

Relative to application **final** implementation, it will typically be necessary to have a full-up development and test environment in place (to include the IBM WAS EE). **Final** implementation, as used here, means the developer’s integration and testing in a (near) full-up functional system. *Use of USAF development and staging labs for the developer’s final integration and testing is an item that needs to be negotiated on an application-by-application basis.* The reason other venues were not explored is that there are just too many possible development environments for which a process / approach could be defined.

Currently GCSS-AF does not address the use of a development environment other than the one employed by the Integration Framework development. Should a contractor and/or development organization decide on a different development environment, it currently is their responsibility to determine how it will be employed and it’s use it in conjunction with the application deployment into the GCSS-AF Enterprise.

Table 14: Integrated Development Environment Items identifies the recommended Integrated Development Environment items that an application developer should have in place for the implementation phase of development.

Table 14: Integrated Development Environment Items

Integrated Development Environment Item	IF Tested Product(s)	Notes
		As applicable: <i>Development Process Implementation Items</i> (from Figure 18: Reference Mission Application Development Process)
Web Authoring		A Web authoring tool provides for ease of development of Web pages and minimizes the level of programming skills the Web page developer requires. Prototype Note: Should be expected to be required for prototype requiring Web based user interfaces. HTML, JSPs, Servlets, Applets, Image Files
Java Development Environment	Visual Age for Java	Should supply a complete Java development environment including testing of units, JavaBeans, EJBs , and applications. Ideally supports deployment into the application server environment. Prototype Note: Should be expected to be required for prototypes developed in Java. EJBs, Homes, JavaBeans, Servlets, JSPs, Applets,

Integrated Development Environment Item	IF Tested Product(s)	Notes
C++ Development Environment	Visual Age Libraries	<p>As applicable: <i>Development Process Implementation Items</i> (from Figure 18: Reference Mission Application Development Process)</p> <p>Should supply a complete C++ development environment including testing of units, CORBA components, and applications. Ideally supports deployment into the application server environment.</p> <p>Prototype Note: Should be expected to be required for prototypes developed in C++ or if a complex Java prototype requires the IF supported application server. (The IBM WebSphere EE requires the use of a C++ compiler).</p> <p>C++/CORBA Components,</p>
Libraries, .JAR files	Not Applicable	<p>Contain Integration Framework service and utility APIs for invocation by application software.</p> <p>Prototype Note: Depending upon the scope of the prototype may or may not be provided.</p> <p>C++/CORBA Components, EJBs, JavaBeans, Servlets, JSPs, Applets,</p>
Application Server Component Development		<p>Used to create CORBA Managed Components, configure EJBs for deployment into the application server, map component data to databases, provide services for container managed messaging, etc.</p> <p>Prototype Note: Depending upon the scope of the prototype may or may not be provided.</p> <p>C++/CORBA Managed Components, EJBs, Homes, DOs, POs,</p>
Text (based) Files	Word, Notepad, WordPad, ...	<p>Used for text files that are not automatically generated by other IDE tools.</p> <p>Prototype Note: Most likely will be required.</p> <p>Property Files, DTD files, DDL files,</p>
XML Development		<p>Used for developing XML objects including DTD files. Provides tools to parse, build, and analyze XML code.</p> <p>Prototype Note: Only required if developing a prototype requiring XML (BODs).</p> <p>BODs</p>
UML to Components		<p>Used to extract designs from UML and generate the “code templates” for the modeled components. This includes the generation of IDL.</p> <p>Prototype Note: Generally not required, but if a design is available for extraction, it has utility.</p> <p>EJBs, CORBA Managed Components,</p>
Reverse Engineering to UML	Rational Rose	<p>Used to extract a UML model from developed code.</p> <p>Prototype Note: Would not be used for prototypes.</p> <p>UML Implementation Model</p>
UML to DDL	Rational Rose	<p>Used to extract DDL from UML defined classes.</p> <p>Prototype Note: Generally not required, but if a design is available for</p>

Integrated Development Environment Item	IF Tested Product(s)	Notes
		As applicable: <i>Development Process Implementation Items</i> (from Figure 18: Reference Mission Application Development Process) extraction, it has utility. Schema/DDDL for Data Objects (in RDBMS)
Database Development and Deployment	Oracle, DB2	Used to create, configure, and administer the relational databases. Includes setting up replications, distribution, etc. Prototype Note: Generally required assuming prototypes will include the need to persist data. Relational database
Messaging Utilities		Used to create required queues, queue managers, channels, pub/sub brokers and streams, etc. Also used for monitoring and administering the messaging “system”. Prototype Note: Only required for those prototypes utilizing messaging.
Application (Component) Deployment	WAS AE/EE Systems Management	Used to deploy the actual applications into the run time environment. Prototype Note: Generally required assuming prototypes would be more extensive than just developing inside of the Java or C++ development environment. Currently the IF has tested only using WAS EE although for some prototypes, AE can suffice. However, if (two-phase commit) transactions are included, EE is required. Executable (Deployed) Applications
Run Time Environment	WAS AE, WAS EE, Oracle, DB2, PD, DNS,	This is the actual run-time environment into which applications are deployed. Prototype Note: Generally required assuming prototypes would be more extensive than just developing inside of the Java or C++ development environment. Currently the IF has tested only using WAS EE although for some prototypes, AE can suffice. However, if (two-phase commit) transactions are included, EE is required. Executable (Deployed) Applications

Table 15: Test Tool Categories identifies the recommended test tool categories for which an application developer should have tools in place for the implementation phase of development.

Table 15: Test Tool Categories

Test Tool Categories	IF Tested Product(s)	Notes
		As applicable: Development Process Implementation Items Tested (from TBA)
Unit Testing	Visual Age for Java, Visual Age for C++	Java and C++ classes can be tested as individual units using these tools. Prototype Note: Should be expected to be required for prototypes developed in Java. Java Classes, C++ Classes
Component Testing	Visual Age for Java, Visual Age for C++, Oracle, DB2	Individual components can be tested using these tools. Multiple cooperating components can be tested to a certain level as limitations such as 2-phase commits exist. Prototype Note: Should be expected to be required for prototypes. C++/CORBA Components, EJBs, Homes, JavaBeans, Servlets, JSPs, Applets,
Application / Integration Testing	WAS AE, WAS EE, Oracle, DB2, PD, DNS, WinRunner	Multiple cooperating components up to the complete application can be tested. Multiple cooperating applications can also be tested. The WinRunner tool allows test cases to be scripted for automated functional testing. Prototype Note: Generally required assuming prototypes would be more extensive than just developing inside of the Java or C++ development environment. Currently the IF has tested only using WAS EE although for some prototypes, AE can suffice. However, if (two-phase commit) transactions are included, EE is required. Executable (Deployed) Applications, C++/CORBA Components, EJBs, Homes, JavaBeans, Servlets, JSPs, Applets,
Performance and Stress Testing	LoadRunner	Used to simulate multiple and simultaneous users. Allows scripting of scenarios to support automation of performance/stress tests. Prototype Note: Only required if prototype is being developed with performance concern. Executable (Deployed) Applications, C++/CORBA Components, EJBs, Homes, JavaBeans, Servlets, JSPs, Applets, ...
Regression Testing	WinRunner, LoadRunner	WinRunner provides for scripted functional tests while LoadRunner provides for scripted stress testing. Prototype Note: Only required if prototype will be repetitively exercised. Executable (Deployed) Applications, C++/CORBA Components, EJBs, Homes, JavaBeans, Servlets, JSPs, Applets, ...

5.1.7 Test Component Descriptions

Throughout section 5 references are made to the Integration Framework test components as well as citing as examples these same test components. These test components were developed both as the means to test and exercise the Integration Framework and to provide examples to developers of how to use the IF and its services. Figure 26: Test Components UML Analysis Class Diagram identifies the test components specified and provided by the Integration Framework. **Note that this analysis model will be updated to identify all the applicable components as defined in section 5.1.2.1 Analysis.**

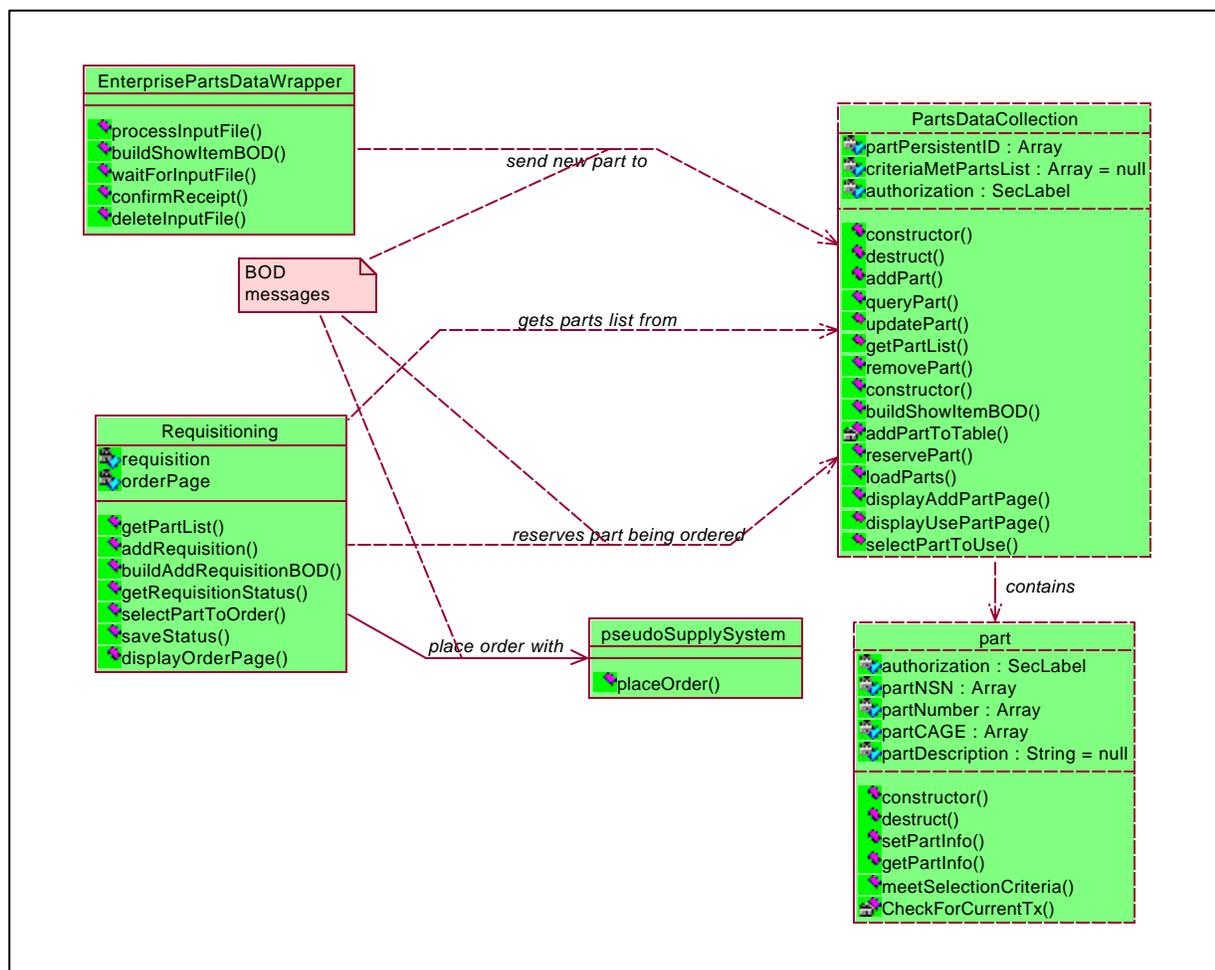


Figure 26: Test Components UML Analysis Class Diagram

The components identified in the diagram along with a brief description of the component follow:

PartsDataCollection (PDC)

This component maintains a list of parts and available quantities. It provides a user interface by which a user can add parts, delete parts, view parts information, and modify parts information. In addition it provides Business Service Requests for adding parts (for bulk load) and for updating inventory quantities as would be used for supplying parts for a part order.

The PDC also can be configured as either an Enterprise level component or a Base level component. As an enterprise level component it publishes all part additions, modifications, or deletes to components that have subscribed for enterprise PDC changes. As a Base level component it subscribes to enterprise PDC changes and synchronizes its information with the enterprise PDC using the published information.

Part

This component contains the information about the part and provides the methods to set and get the part information. This is what the PDC contains.

Enterprise Parts Data Wrapper (EPD)

This component provides an example of an interface (wrapper) to a Legacy system that utilizes a flat file as its interface mechanism. The EPD monitors a file system directory for a file addition (representative of a file FTP to this directory from a Legacy system). Files “sent” to this interface component can contain many, many parts (information) that are sent to the enterprise PDC for addition. BODs are sent to the PDC BSR for adding parts effectively performing a bulk load. The EPD has no user interface.

Requisitioning Component

This component provides the means by which a user can order parts. A user interface provides a means for a user to view a list of parts matching the users selection criteria. The user can then select one of the displayed parts and place an order for the part along with a desired quantity. The requisitioning component maintains a history of ordered parts, sends a message to the PDC to “reserve” the desired quantity of parts for this order, and sends a message to the (pseudo) Supply System to process the order.

Pseudo-Supply System

This component represents a Legacy supply system that will actually process part orders. This component simply creates an order in a database.

Figure 27: Test Component UML Design Packages identifies the test component (UML) design packages implemented and provided by the Integration Framework.. The test component UML design models are located in the Integration Framework package of the top-level Test package of the System Solutions Rose Model. Owing to the size of the test components design model, it is **not** included in this developer’s guide.

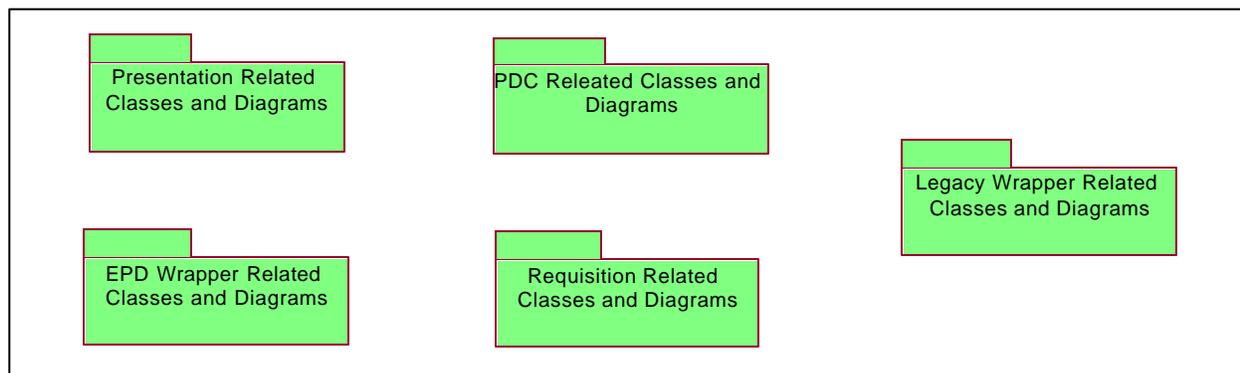


Figure 27: Test Component UML Design Packages

5.1.8 Additional IBM Reference Material

In addition to this guide, there are several very useful Guides and Redbooks produced by IBM that provide descriptions of the development of enterprise applications in conjunction with IBM WebSphere Enterprise Edition. These can be found through the IBM Websites at www.ibm.com (Guides) and www.redbooks.ibm.com (Redbooks). Of note to implementations for the “Data Layer” are the documents identified in the following list. While specific chapters and sections relative to data layer components are relatively obvious in these documents, it is strongly suggested that the user of this guide be familiar with the contents of these documents.

- WebSphere Application Server, Introduction to WebSphere Application Server, Version 3.5, SC09-4430-01, Second Edition (June, 2000).
- IBM Component Broker Connector Overview, SG24-2022-02, Published on June-19-1998 (*Redbook*)
- WebSphere Application Server Enterprise Edition Component Broker, Programming Guide, Version 3.5, SC09-4442-01, Second Edition (August, 2000).
- WebSphere Application Server Enterprise Edition Component Broker, Advanced Programming Guide, Version 3.5, SC09-4443-01, Second Edition (June, 2000).
- WebSphere Application Server, Writing Enterprise Java Beans in WebSphere, Version 3.5, SC09-4431-02, Second Edition (June, 2000).
- WebSphere Application Server Enterprise Edition Component Broker, Procedural Application Adaptor Development Guide, Version 3.5, SC09-4572-00, First Edition (August, 2000).
- IBM WebSphere Application Server Enterprise Edition Component Broker 3.0: First Steps, SG24-2033-03, Published on August-31-2000 (*Redbook*).

- WebSphere Application Server Enterprise Edition Component Broker, Application Development Tools Guide, Version 3.5, SC09-4445-01, Second Edition (July, 2000).
- WebSphere Application Server Enterprise Edition Component Broker, System Administration Guide, Version 3.5, SC09-4448-01, Second Edition (July, 2000).
- User-to-Business Patterns Using WebSphere Enterprise Edition: Patterns for e-business Series, SG24-6151-00, Published on September-15-2000 (*Redbook*).
- WebSphere Application Server Enterprise Edition Component Broker, Getting Started with Component Broker for Windows NT (or Solaris), Version 3.5, SC09-4443-01, Second Edition (August, 2000).
- WebSphere Application Server Enterprise Edition Component Broker, Programming Reference, Version 3.5, SC09-4446-01, Second Edition (July, 2000).

5.2 Presentation

5.2.1 Overview

The presentation layer components are responsible for providing the end user interface. The preferred approach for the presentation to the user is to be Web Browser-based. While other approaches are not precluded, there is no specific support provided by the Integration Framework.

The presentation layer components are typically divided into two different elements. The part that runs on the client workstation (the client-side tier) and the other that supports the requests from the user that runs on the application/web server (the server-side tier). A Web browser provides the client-side functionality, displaying HTML returned from a Web Server. The server-side capabilities are provided through Servlets, Java Server Pages (JSPs), and HTML.

The topics covered in this section are:

- The use of Servlets - In particular the use of the Integration Framework provided **IFServlet** base class will be discussed including where to get it, what services are provided, and how to extend it for application use.
- The use of JSPs in providing a Graphical User Interface for the user using a web browser.
- The use of HTML in providing a browser independent interface for the user.
- The potential use of Applets in a web application solution.

- Configuration items that are required in order to enable and use the Integration Framework provided security services from within the presentation layer.

Throughout this section code snippets are used as examples that are taken from the test components that have been developed to drive the Integration Framework for testing purposes. For a complete understanding of the recommended use of these technologies, refer to the **PDCClientClasses** and **ReqClientClasses** from the test components. These are the Servlets that provide the server-side tier of the presentation layer for the "Parts Data Collection" and "Requisitioning" test components respectively.

In addition to this guide, there are several very useful Redbooks produced by IBM that provide descriptions of the development and use of Servlets and JSPs in conjunction with IBM WebSphere Advanced Edition. These Redbooks can be found on the IBM Web site at <http://www.redbooks.ibm.com>.

5.2.2 Service Use

5.2.2.1 Servlets

Servlets are a server-side alternative to the use of Java Applets. Java Applets are programs that are downloaded and run in a Client JVM and referenced directly in Web pages. When a browser loads a Web page that contains a reference to an Applet, the Applet is downloaded to the client box and executed in the browser. As the Applets grow in size, the download times become unacceptable. Another issue faced by Applets is compatibility. In order to run an Applet the developer must have a compatible browser. Applets are also faced with the restrictions in terms of what server-side resources are accessible. Additionally, Applets provide a security risk and, based on the client location security policy, may not be passed through the security firewall thus precluding their use.

These problems have identified the need for server-side Java. Servlets are an option for server-side Java development that helps to solve the problems that are faced when using Applets. Servlets are generic extensions to Java-enabled servers that are most commonly used to extend Web Servers. They are dynamically loaded to handle requests from the Web Server. Servlets are run inside a Java Virtual Machine(JVM) running on the server side and therefore do not depend on browser compatibility.

Servlets can be used for any number of Web-related applications. The primary use for a Servlet is to produce HTML that is to be returned to a client workstation containing the results of some request from that client. The Servlet receives the request, uses the functions of the components in the business logic layer to gather, process and return the results of the request. The data returned is then formatted into an HTML document that is returned to the client browser.

The Servlet is the controller in the model-view-controller design pattern. Refer to Section 5.1.1.2 Model-View-Controller Design Pattern for the details of this design pattern. The Servlet controls the workflow in terms of the order in which the business logic is invoked in order to carry out the user's request. It may be that only one Business Component is required to be called

in order to obtain the desired result but, it may also be the case that multiple calls to multiple business components are required. In any event, the result obtained from any business logic invocation is then used to populate the HTML request that is returned to the client's browser.

HTTP Servlets extend the Web server capabilities by creating a framework for providing request and response services over the WEB. When a Client sends a request to the WEB server that references a Servlet, the WEB server forwards the request information to an application server and has the Servlet construct the client response.

Servlets help control the execution of the backend business logic since they have access to backend components and construct the client response largely based on the capabilities and information provided by these components. The number of Servlets that are required to support an application would depend on the business components that makeup the application. It is possible that there may be one Servlet for each business component. The Business Component Servlet would receive parameters from the request that determine which operation(s) to invoke. An alternative would be to provide a different Servlet for each operation of the Business Component.

In the Integration Framework environment Servlets are deployed on an IBM WebSphere Advanced Edition Application Server that may or may not reside on the same machine as the Web Server. Placing the Application Server on a separate machine allows a single Web Server to use the services of multiple Application Servers thus, providing a level of scalability, load balancing, and fault tolerance. The physical location of the Servlets is determined by configuration settings applied when installing an application to the Application Server.

Servlets provide two key roles in the Integration Framework. They must pass on user authentication and authorization information and they must locate and call methods of distributed components. The following sections describe how an application developer uses the IFServlet and the Integration Framework naming service to implement these roles.

5.2.2.1.1 The IFServlet and Its Use

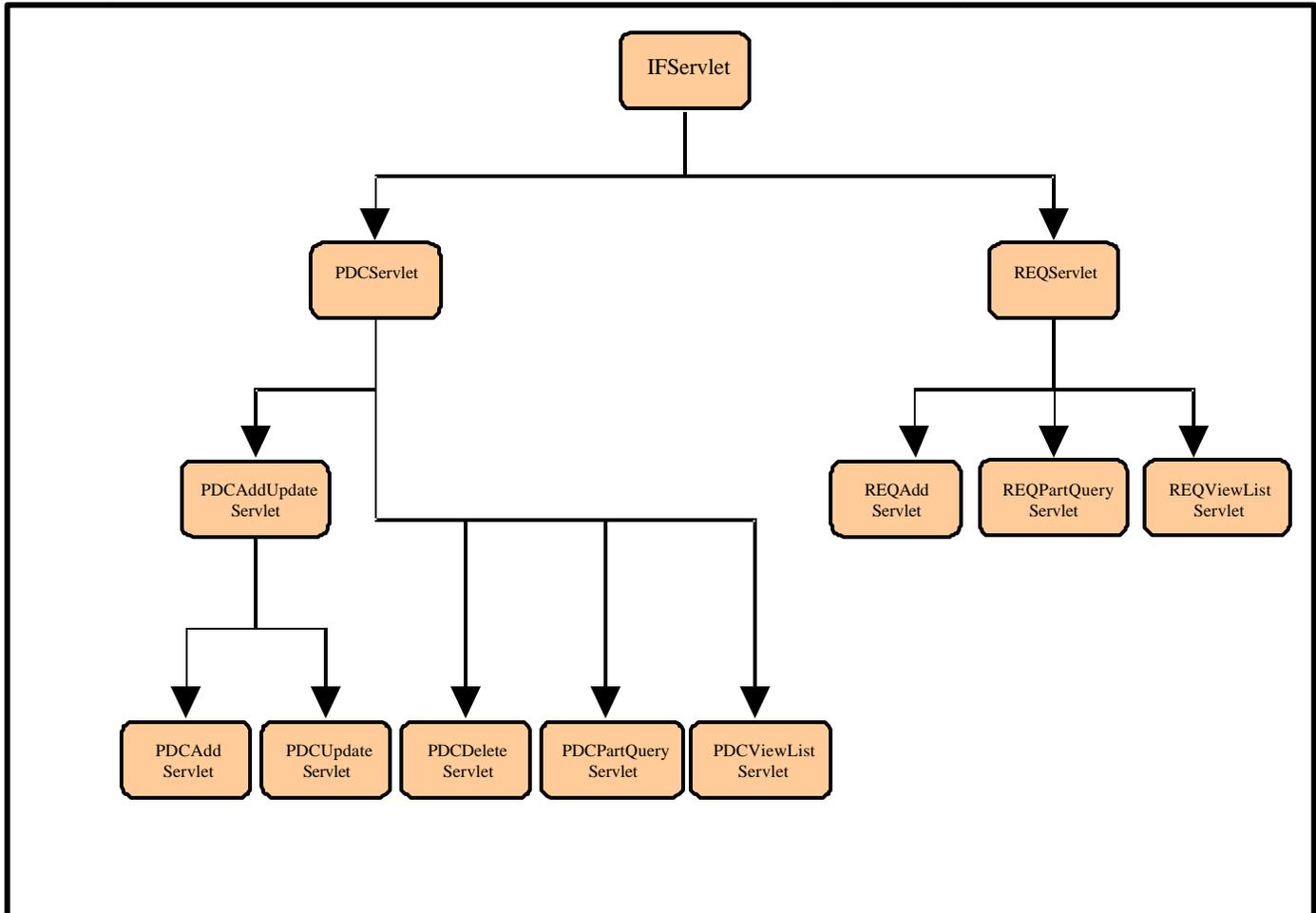
The Integration Framework provides a common base class from which to develop secure Servlets that interface with EJB and/or CORBA components. The **IFServlet** is the base class derived from the **HTTPServlet**, and is intended to be the only Servlet in the class hierarchy that implements the abstract *doPost* and *doGet* methods of the **HTTPServlet**. This is done to ensure that control will pass through the **IFServlet**, allowing common Integration Framework functions to be performed. All application specific Servlets should extend the **IFServlet** base class.

Java Server Pages (JSPs) can interact with servlets at the lower levels in the hierarchy and as long as the IFServlet methods are not overridden, they will be available throughout the hierarchy. Adding additional function to the servlets lower in the hierarchy is accomplished through the use of the abstract **doPostService** and **doGetService** methods. Figure 28: Example Servlet Hierarchy provides an example of servlet hierarchies. Notice that the Servlets to the left of the diagram support one application while the ones at the right support an entirely separate

application. Each makes use of the **IFServlet** and the capabilities that it provides for the common functions required by all Servlets built on the Integration Framework.

Figure 28: Example Servlet Hierarchy

Figure 29: IFServlet Source View shows the methods that are provided by the **IFServlet**. The



IFServlet includes four abstract methods (indicated with a large capital “A” to the right of the method) that must be implemented by Servlets that extend the **IFServlet**. These are:

- **doGetService**
- **doPostService**
- **initializePropertyManager**
- **initializeServerConnection**

Each of these abstract methods is discussed in detail in the following sections. All methods are described in Appendix A.

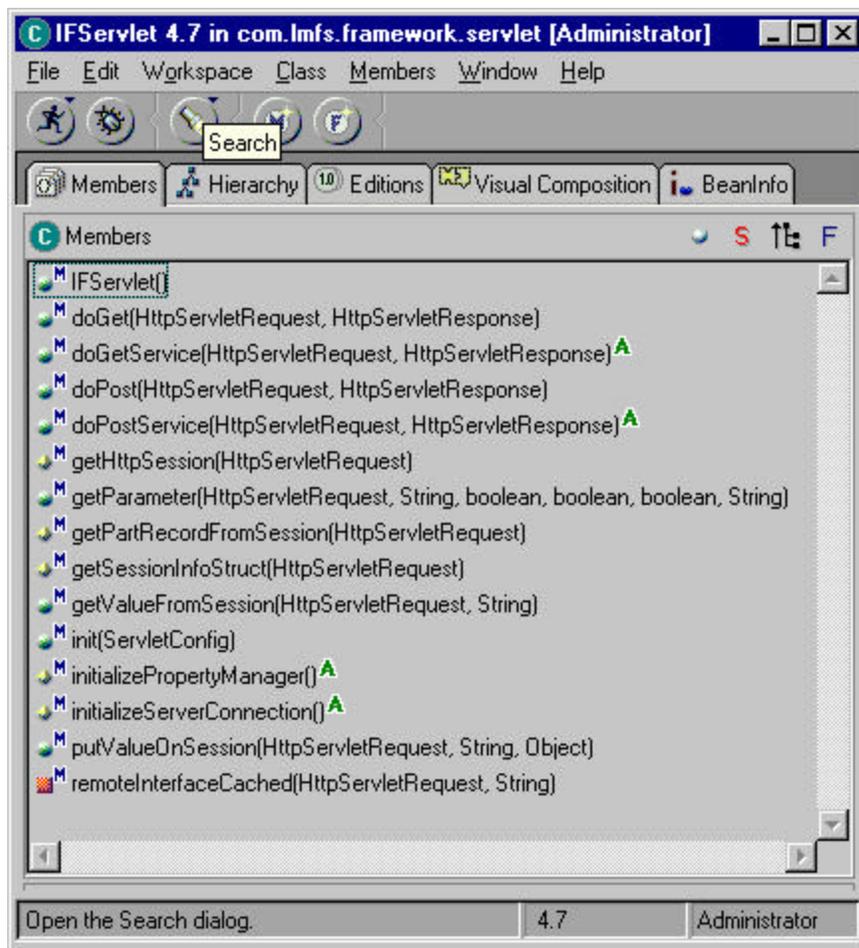


Figure 29: IFServlet Source View

5.2.2.1.1.1 Use of doGetService - doPostService - abstract methods

The IFServlet processes security code common to all Servlets in the *doGet* and *doPost* methods. This includes programmatically logging in to the WebSphere Enterprise Edition application server and reading and processing the property file. These then call **doGetService** and **doPostService** respectively. *doGetService* and *doPostService* are abstract methods that the developer implements to provide any additional functionality and/or application specific processing

The best way to illustrate the pattern of the intended use of the IFServlet abstract methods *doGetService* and *doPostService* is through an example.

Figure 30: Add Part Control Flow Scenario shows the flow of control when a user from the client browser adds a part in the Parts Data Collection test application provided with the Integration Framework.

The flow of control assumes that all the initial security actions have taken place. This means that Local Director has established a connection with a WebSeal Proxy, and the WebSeal proxy has authenticated the user.

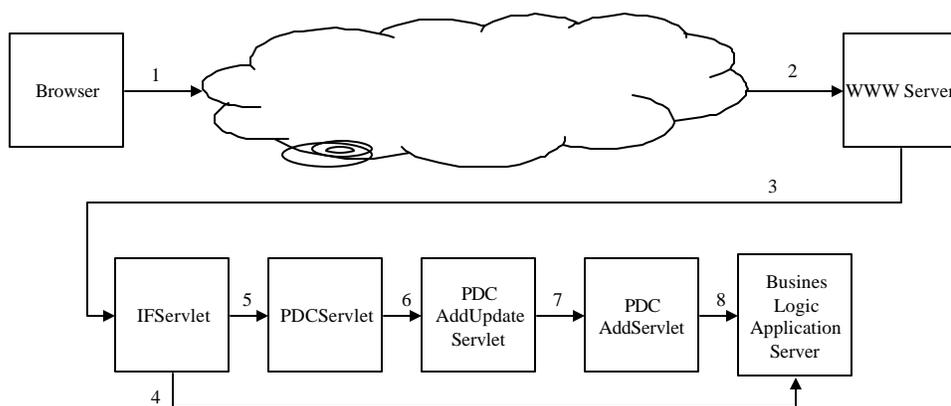


Figure 30: Add Part Control Flow Scenario

The following is a breakdown of the example above, Figure 30: Add Part Control Flow Scenario:

1. The Web Browser submits a request to add a new part to the parts data collection. The browser sends the request to their local Web Proxy Server.
2. The request travels to the backend Web Server.
3. The Web Server passes the request to the *doPost* method of the requested Servlet. In this case the target Servlet is the **PDCAddServlet**. Since that Servlet does not contain an implementation for the *doPost* method control is passed up the hierarchy until an implementation is found. This is how the **IFServlet** intercepts control so that the required security operations can be performed.
4. The **IFServlet** programmatically logs in (authenticates) to the WebSphere Application Server Enterprise Edition (WAS-EE), a.k.a. Component Broker (CB), via a **ServletLoginHelper**. After this connection has been checked/established the *doPostService* method is called to handle the users request. In this case the request is passed to the **PDCServlet** for handling.

5. The **doPostService** of the **PDCServlet** is invoked to handle the **addpart** request from the user. Again, there is no implementation provided for this abstract method in the **IFServlet**. This allows the handling of the *doPost* requests to be handled at a lower level where actions specific to the request can be performed.
6. The **PDCServlet** does not provide an implementation of the *doPostService* method so the request is passed down the hierarchy until an implementation is found. In this case, **PDCAddUpdateServlet** provides one in which the common functions required for both adding and updating part information are performed. The *doPostService* method retrieves the part information from the request and then invokes the *Persist* method of the lower level Servlet, in this case the **PDCAddServlet**, to complete the action.
7. The *Persist* method of the **PDCAddServlet** is invoked. This method gets the remote interface of the Parts Data Collection component that is running in the application server and invokes the *addPart* method of that interface.
8. The *addPart* method of the Parts Data Collection component is invoked which causes the information passed in with the user request to be added to the backend datastore.

5.2.2.1.1.2 Use of initializePropertyManager abstract method

The **IFServlet** uses an application specific property file that is based on the **IFPropertyManager** class. Each Servlet that extends the **IFServlet** must implement the *initializePropertyManager* abstract method. This is implemented to name the property file that will be used for a given application Servlet or group of Servlets. The **IFPropertiesManager** wrappers the **PropertyResourceBundle** class in Java. The **IFServlet** specific properties that must be included in the referenced property file are shown in Table 16: Servlet Property File Contents.

As far as a developer is concerned, the entire implementation of this abstract method will look like the following example:

```
protected void initializePropertyManager()
{
    propertyManager = new IFFilePropertyManager("PDCSession");
}
```

Figure 31: Example of IFPropertyManager Abstract Method

The parameter in quotes, **PDCSession**, names the application specific property file.

Once initialized, the values contained in the property file can be extracted and used to alter the execution of the application. This method allows the way the application behaves to be modified without requiring it to be re-compiled.

An example of how to retrieve and store the values contained in the property file is contained in the *init* method of the **IFServlet**. Please refer to the source code of the **IFServlet** for a complete understanding of how this is accomplished.

Table 16: Servlet Property File Contents

File MUST contain Properties	Possible/Example Values	Description
Security_on=	True, False	Set to 'true' when the business logic application server security is turned on (<i>AE to EE security</i>), set to "false" when security is not activated.
Logging_config=	C:\WebSphere\AppServer\hosts\default_host*\Log4J.properties (* <i>Your Specific application server name. i.e. IPMS</i>)	Specifies the property file that is used to set the initial logging configuration attributes. If NOT using logging services, leave the value of this parameter blank.
Initial_context_factory=	com.ibm.ejb.cb.runtime.CBCtxFactoryHostWidenedDefault	Context factory used to locate Components in the naming service.
Provider_url=	iiop://rsalapp4.gcss.af.mil:900 <i>Example only</i>	This contains the DNS entry of the business logic application server host and port.
Host=	rsalapp4 <i>Example only</i>	The machine name of the business logic application server host machine.
Gcss_userid=	cell_admin <i>Example only</i>	Set to a DCE Cell administrator ID. Used only if the business logic application server security is turned on
Gcss_password=	core1234 <i>Example only</i>	Set to a DCE Cell administrator <i>password</i> . Used only if the business logic application server security is turned on
Client_style_url=	ClientStyles.properties	Properties passed to the CBGlobalSeries initialization. This file is obtained from the Component Broker deployment tools. Only the following two properties need to be in this file: <ul style="list-style-type: none"> • com.ibm.CORBA.nameServerHost= and • com.ibm.CORBA.nameServerPort=

An application developer may add additional properties to this file in order to handle application specific needs. Since the Servlet that extends the **IFServlet** must provide an implementation of the *IFPropertyManager* abstract method, the initialization of application specific properties can be handled there. They will be accessed in the same manor as the properties obtained in the **IFServlet**. Again, refer the *init* method of the **IFServlet** for the details on how this is done.

5.2.2.1.1.3 Use of initializeServerConnection abstract method

The *inititalizeServerConnection* method is used to initialize a connection between the Servlet and the IBM WebSphere Application Server Enterprise Edition - Component Broker application server (WAS-EE). This enables the Servlet to have access to business logic components that run in that environment and to use the Component Broker (CB) naming service. An IFServlet based Servlet performs all the same actions that any other client executing outside WAS-EE would be required to do. Much of this complexity is hidden from the IF developer but still requires some understanding on how to implement the abstract methods. More information about the naming service is provided in Section 5.2.2.2.

The method **CBSeriesGlobal** is used to get an initialize the ORB. The following is the abstract implementation of the *initializeServerConnection* method . This code shows how the Servlet would initialize the **CBSeriesGlobal** object for a connection to the WAS-EE environment.

```
protected void initializeServerConnection()
{
    try
    {
        String[] args = {"", ""};
        com.ibm.CBCUtil.CBSeriesGlobal.Initialize(args, ORBProperties);
    }
    catch (java.lang.Exception e)
    {
        System.err.println("exception " + e + " caught in init...");
        e.printStackTrace();
    }
}
```

Figure 32: Example of Abstract Method Implementation of initializeServerConnection

If the client program uses either a copy helper or primary key class with an attribute that is an object reference, then initializing **CBSeriesGlobal** is a requirement. This is because the implementation of the copy helper and primary key depend on **CBSeriesGlobal:orb()** when using the ORB **object_to_string()** operation. It is also a requirement to use **CBSeriesGlobal** if security or object handles will be used in the Component Broker Client. Using **CBSeriesGlobal** in all Component Broker client programs is generally recommended.

The **CBSeriesGlobal** interface provides a number of different *Initialize* methods depending on the needs as a Java Servlet, Applet or application. All of these *Initialize* methods encapsulate the calls to initialize the ORB and get an initial reference to the Naming Service. Refer to the CB Documentation [“Java Programming model ->The Java Client->Initializing the Component Broker client environment”](#) for more information.

5.2.2.1.1.4 Security - Authorization

The Integration Framework proxy server (WebSeal) puts authentication and authorization information in the **HTTPRequest** header for common use by Integration Framework application components. For common use and to provide a level of abstraction from the WebSeal implementation, this information is captured in a security object named **SessionInfoStruct**.

The **SessionInfoStruct** object is defined as follows:

```
Class SessionInfoStruct
    UserSessionInfoStruct userSessionInfoData;
    AppSessionInfoStruct appSessionInfoData;
    string[] dynInfo;
Class UserSessionInfoStruct
    UserSecInfoStruct userSecInfoData;
    string[] dynInfo;
Class UserSecInfoStruct // User parameters required to make authorization decisions
    String username;
    String groups;
    String creds;
    string[] dynInfo;
Class AppSessionInfoStruct // parameters to aid the app in identifying its location in
//the Policy Director Object Space
    String baseNameCTX;
    String baseNameQualifier;
    string[] dynInfo;
```

Figure 33 Example of SessionInfoStruct Code

A business logic component that is utilizing component level security checking will require this object as an input parameter. The **IFServlet** base class provides the *getSessionInfoStruct* method that is used to extract the user information from the **HTTPRequest** and build the **SessionInfoStruct**. Once the information has been retrieved this structure needs to be passed as a parameter in the method call.

To illustrate, the following line of code is a call to get a list of parts from the CORBA **PDCSessionAO** Interface:

```
// A list of parts is returned
PartRecordArray = getRemoteInterface(request).getParts(1,getSessionInfoStruct(request));
```

Figure 34: Example of PDCSessionAO Interface Code

There are three method calls in this single line of code. The first gets the remote interface of the CORBA component executing in WAS-EE. The second is a call to the method that is provided by the **IFServlet** base class that gets the security information that is contained in the request object. Finally, the *getParts* method of the remote interface is invoked.

5.2.2.1.1.5 Obtaining the IFServlet and other required "jar" and class files

For application/Servlet development the following jars need to be imported into the application developers development environment. A brief description of what is used in these jar files is contained in Table 17: Servlet Development Dependencies.

Table 17: Servlet Development Dependencies

IF jar File	IF VAJ Repository Project	Description
Com_lmfs_framework_servlet.jar	FrameworkTestComponents	Contains the implementation of the IFServlet base class.
Com_lmfs_framework.jar	FrameworkTestComponents	Contains the classes used by the IFServlet for managing property files and the ServletLoginHelper used to gain access to the Component Broker environment.
IBM CB Prereqs.jar	IBM CB Prerequisites	A subset of classes from the somajor.zip that must be imported into the development environment. This also contains the CBGlobalSeries class that provides the naming services to the IFServlet.
CORBAFWAuth.jar	CORBA Framework Authorization	Contains class definitions specific to the Security between AE and the CB environment.
GCSSAFLog4j.jar	GCSS_AF_IF_ESM_log4j	Provides the APIs for the logging services.
FrameworkComponents_Security.jar	FrameworkComponents_Security	Contains IF security classes. The IFServlet calls a helper class, contained in this jar file, which is used to get security information pertaining to the user making the request. This information is used to populate a security structure, sessionInfoStruct . This structure is passed as a parameter in component method calls and then used to perform authorization checks.

The Integration Framework provides the jar files and a VAJ repository that contains the projects listed above. These classes must be loaded into the development environment workspace before beginning development.

The file **somajor.zip** must be imported into the development environment in order to support the development of application Servlets; this is represented by the project "IBM CB Prerequisites" found in Table 17: Servlet Development Dependencies.

For application use at runtime, two IF Jar files are provided by export from VAJ that allow a developer to use the **IFServlet**. These are:

- **com_lmfs_framework_servlet.jar**
- **com_lmfs_framework.jar**.

These Jar files are located in the directory **/h/IFSServices/lib** on any machine where the WebSphere AE application server is installed.

The **IFSSERVICES.jar**, provided by the Integration Framework, is used to combine framework specific classes and jar files together into one jar file. Only Integration Framework specific jars are contained in the **IFSSERVICE.jar** file. Since this jar file also contains class files that the **IFServlet** uses, the **IFSSERVICES.jar** is placed in the **/h/IFSServices/lib** folder and then added to the Application Servers CLASSPATH environment variable on a runtime system.

The file **somajor.zip**, obtained from IBM WebSphere Enterprise Edition, must be included at the end of the base CLASSPATH environment variable that is found in the **admin.config** file in **/Websphere/appserver/bin**. The runtime location of this zip file is **/h/IFSServices/lib** on the AE hosts.

5.2.2.2 Using the Naming Service

The Naming Service provided by the WebSphere Enterprise Edition (WAS-EE) is used by the Integration Framework to provide a federated CORBA and EJB namespace. WebSphere Advanced Edition (WAS-AE) uses the same ORB as WAS-EE, but implements a different interface to the naming service. This is an important distinction to understand when developing in an environment other than WAS-EE. More information on this topic can be found in the Business Logic section.

Servlets use the Naming Service to locate application components executing in the WAS-EE environment.

5.2.2.2.1 Using the Naming Service – Corba

Once **CBSeriesGlobal** is initialized, by the abstract method *initializeServerConnection*, the Servlet must get a HOME Interface of the CORBA component using the **CBSeriesGlobal NameService** to resolve a factory finder. The factory finder in turn can be used to resolve a home interface. The *getHome* method is referenced from the **findByPrimaryKey** (listed below) which gets a CORBA Object reference. With the Object reference, calls to any of the components remote interface methods is possible.

```
private PDCSessionAO findByPrimaryKey(HttpServletRequest req, String location)
{
    // Generic CORBA Object
    org.omg.CORBA.Object obj = null;

    // Create the Key
    byte theKeyString[] = null;
    PDCSessionAOKey pdcSessKey = PDCSessionAOKeyHelper._create();

    pdcSessKey.location(location);
    theKeyString = pdcSessKey._toString();

    // Get the Home, use the PrimaryKey to locate the object, or else create it
    com.ibm.IManagedClient.IHome theHome = getHome();
    try
    {
        logger.info("location = " + location);

        obj = theHome.findByPrimaryKeyString(theKeyString);

        if (logger.isDebugEnabled())
        {
            logger.debug("pdcSessKey = " + pdcSessKey);
            logger.debug("theKeyString = " + theKeyString);
            logger.debug("getHome() returned: " + theHome);
            logger.debug("Found obj: obj = " + obj);
        }
    }
    catch (com.ibm.IManagedClient.INoObjectWKey nowk)
    {
        // Create the PDCSession Object
        try
        {
            obj = getHome().createFromPrimaryKeyString(theKeyString);
            if (logger.isDebugEnabled())
            {
                logger.debug("Caught " + nowk);
                logger.debug("Object successfully created: obj = " + obj);
            }
        }
        catch (Exception ex)
        {
            logger.info("Caught General exception: " + ex);
            logger.info("ERROR: createFromPrimayKeyString() failed, Exception:
"
                + ex);
        }
    }
    catch (Exception e)
    {
        try
        {
```

```
        obj = theHome.createFromPrimaryKeyString(theKeyString);
            if (logger.isDebugEnabled())
                logger.debug("Object successfully created: obj = " + obj);
        }
        catch (Exception ex)

        {
            logger.info("Caught General exception: " + ex);
        }

        System.out.println("Printing stack trace");
        e.printStackTrace(System.out);
    }

    if (logger.isDebugEnabled())
        logger.debug("Calling PDCSessionAOHelper.narrow(obj)...");

    PDCSessionAO pDCSessionAO = PDCSessionAOHelper.narrow(obj);

    if (logger.isDebugEnabled())
        logger.debug("Called PDCSessionAOHelper.narrow(obj).");
return pDCSessionAO;
}

// Get the home of the PDCSessionAO
private static com.ibm.IManagedClient.IHome getHome()
{
    logger.debug("*** Called GetHome() v1");
    org.omg.CORBA.Object obj = null;
    FactoryFinder factoryFinder = null;
    String pdcSessionName = new String("PDCSessionModule::PDCSessionAO.object interface");
    com.ibm.IManagedClient.IHome pdcSessHome = null;

    System.out.println("***** Getting the Home interface from the Name Service *****");

    // Locate the factory finder
    try
    {
        if (logger.isDebugEnabled())

            logger.debug("CBSeriesGlobal.nameService().resolve_with_string(\"host/resources/factory-
finders/PDCSessionServers-server-scope\)");
            obj = CBSeriesGlobal.nameService().resolve_with_string("host/resources/factory-
finders/PDCSessionServers-server-scope");
            if (logger.isDebugEnabled())
                logger.debug("FactoryFinderHelper.narrow(obj)");
            factoryFinder = FactoryFinderHelper.narrow(obj);
        }
        catch (Exception e)
        {
```

```
        System.out.println("ERROR: resolve_to_factory_finder() failed, Exception: " + e);
        System.exit(1);
    }

    // Resolve the Home for PDCSessionAO
    try
    {
        System.out.println("factoryFinder.find_factory_from_string(\"" + pdcSessionName +
"\");");
        obj = factoryFinder.find_factory_from_string(pdcSessionName);
        System.out.println("IHomeHelper.narrow(obj)");
        pdcSessHome = com.ibm.IManagedClient.IHomeHelper.narrow(obj);
    }
    catch (Exception e)
    {
        System.out.println("ERROR: resolve_to_PDCSession_home_via_find_factory failed,
Exception: " + e);
        System.exit(1);
    }
    return pdcSessHome;
}
```

Figure 35: Example of findByPrimaryKey code

5.2.2.2.2 Using the Naming Service - EJB

For EJBs, once **CBSeriesGlobal** is initialized by the abstract **initializeServerConnection**, as discussed in Section 5.2.2.1.1.3 Use of initializeServerConnection abstract method, **getInitialContext** is used to resolve an EJB Home that is used to resolve the remote EJB interface.

The following example shows how this is done:

```
protected Requisition getServiceBean(HttpServletRequest request) throws Exception
{
    if (logger.isDebugEnabled())
        logger.debug("Getting the 'Requisition' EJB Home...");

    //Locate the EJB using the CB naming service.
    Object homeObject =
        getInitialContext(request).lookup("Requisition");

    if (logger.isDebugEnabled())
        logger.debug("Got the 'Requisition' EJB Home.");

    RequisitionHome beanHome = (RequisitionHome)
        javax.rmi.PortableRemoteObject.narrow((org.omg.CORBA.Object) homeObject,
        RequisitionHome.class);
}
```

```
        if (logger.isDebugEnabled())
            logger.debug("Narrowed the General EJB Home Object to the specific
                Requisition Home Object.");

            return beanHome.create();
        }

protected InitialContext getInitialContext(HttpServletRequest request)
{
    initialContext = (InitialContext) getValueFromSession(request, "initialContext");
    if (initialContext == null)
    {
try
        {
            System.out.println("try to get the initial context");
            initialContext = new InitialContext(EJBProperties);
        }

        catch (javax.naming.NamingException e)

        {
            logger.error("Error getting the EJB InitialContext: " + e);
        }
        catch (Exception e)
        { System.out.println("Caught exception in getInitialContext : " + e);
        }

    }
    System.out.println("initialContext= " + initialContext);
    return initialContext;
}
```

Figure 36: Example of `getInitialContext` code

5.2.2.3 JavaServer Pages (JSPs)

Java Server Pages (JSPs) are a simple but powerful technology used to generate dynamic to be presented to a client on the server-side. It provides a way to separate content generation from content presentation. JSPs are often coupled with other Servlets to allow flexible use of built-in Servlet capability and the power of scripting Java directly in a stateless document. When a client requests a JSP page, it is sent to the application server. The JSP processor Servlet (located on the Application server) dynamically builds a Servlet from the JSP source page. This Servlet is then executed with resulting output being written to the client response (HTTP) and sent to the clients' browser. The JSP (like HTML) can submit form data to an application server just by referencing a Servlet. If the base class of that Servlet is **IFServlet**, (this class handles the *doGet* and *doPost*

processing at this level) it calls **doGetService/doPostService** for processing at lower levels of the Servlet hierarchy. This is the way to pass input and start the application Servlets that extend the **IFServlet** thereby extending the services of this base class to the application.

The following code is from the Servlet test components provided with the Integration Framework and its associated JSP used to dynamically construct a table containing the result of the client request.

The first section of code shows how to use the Servlet *doPostService* method to call the *getPartRecordList* method, which makes the component level call to get a list of parts. Once the list is received, it is put into the list on the session. Then the call is made to the JSP page to process the list.

```
public void doPostService(javax.servlet.http.HttpServletRequest request,
    javax.servlet.http.HttpServletResponse response) throws Exception
{
    .....

    // Forward to the appropriate JSP
    if (state.equalsIgnoreCase("view") || state.equalsIgnoreCase("delete") ||
state.equalsIgnoreCase("update"))
    {
        // invoke the remote method of the PDC that produces a list of parts that
meet the
        // criteria provided by the user.
        partRecordList = getPartRecordList(request, response, stockNumber);
        .....

        // Put the partRecordList on the session
        getHttpSession(request).putValue("partRecordList", partRecordList);
        context.getRequestDispatcher("PDCViewList.jsp").forward(request,
response);
    }
    .....
}
```

Figure 37: Example of Servlet doPostMethod Usage

Note Future Capability: Below are snippets from the JSP page, It looks very much like an HTML document but it has Java scripting in it.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN"><!-- Sample JSP file -->

<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Page Designer V3.0.2 for Windows">
<META http-equiv="Content-Style-Type" content="text/css">
<TITLE>
View Part List
</TITLE>
</HEAD>
```

```
<BODY BGCOLOR="#B7C1F1">
```

Notice that the next tag sets up the jsp for using the list object that was stored in the session.

```
<jsp:useBean class="java.lang.Object" id="partRecordList" scope="session" />
```

```
.....
<%
try
{
  PDCHelperModule.PartRecord[] _p0 = (PDCHelperModule.PartRecord[])partRecordList; // throws an
exception if empty. %>
.....
```

This loop is used to construct the Dynamic table. Notice _p0 is the partlist object, iterate though the list and fill in the dynamic content from the partlist on the session. Also note that the first item of each row of the table is a radio button and that this button is assigned a "stocknumber" from our session object.

```
for (int _i0 = 0; ; ) {
  try {
    String stockNumber = _p0[_i0].stockNumber; %>
    <TR>
      <TD><INPUT name="partRecordGroup" type="radio" value="<%= stockNumber %>"></TD>
      <TD><%= _p0[_i0].stockNumber %></TD>
      <TD><%= _p0[_i0].manufacturer %></TD>
      <TD><%= _p0[_i0].partNumber %></TD>
      <TD><%= _p0[_i0].quantity %></TD>
      <TD><%= _p0[_i0].itemDescription %></TD>
    </TR><%
    _i0++;
  }
  .....
```

Below values are placed in hidden Inputs to maintain state between JPS form submission and the servlets that may be called on the submit.

```
<INPUT type="hidden" name="state" value="<%= request.getParameterValues("state")[0] %>" >  
<INPUT type="hidden" name="location" value="<%= request.getParameterValues("location")[0] %>" >  
<INPUT type="hidden" name="basenamequalifier" value="<%=  
request.getParameterValues("basenamequalifier")[0] %>" >  
.....  
</CENTER>  
</BODY>  
</HTML>
```

Figure 38: Example Code taken from JSP Page

5.2.2.4 Applets

Applets are a good choice when infrequent or no access to server side resources is required. Some types of client side programs are very well suited to Applets. Applets tend to have a richer more robust GUI set. Development using Applets is not prohibited by the IF, but is not a recommended approach and as such, no specific support is provided to aid in their development. If Applets are used as part of a solution, it is recommended that they be signed for security purposes.

5.2.2.5 Hypertext Markup Language (HTML)

HTML is the mechanism that is used to present static content information to the end user. This information might be in the form of tables, images, links to other resources, forms, etc. In any event, the Web browser interprets the HTML document and the results are displayed. HTML is used for two reasons:

Platform Independence

HTML is not specific to a given platform, and therefore, as long as the platform of interest has a Web Browser, the desired information can be displayed. One caveat is that care must be taken when using some of the advanced features or browser specific non-standard extensions of HTML, as this may cause these applications to become browser specific.

No Active Content

Because of security concerns there is a desire to reduce or eliminate the use of active content in the Web browser. The primary reason for the concern is the possibility that malicious software may be loaded into the browser. Also, Air Force policy prohibits the use of dynamic content on the client. The use of static HTML and HTML produced from JSPs for presentation eliminates this concern.

5.2.3 Communications Between Layers

5.2.3.1 Vertical

1. Extend the **IFServlet** as the base class for all Servlets. This provide a capability to programmatically log in (authenticates) to the WebSphere Application Server Enterprise Edition (WAS-EE) via a **ServletLoginHelper**. Refer to Section 5.2.2.1.1 The IFServlet and Its Use for additional details.
2. Ensure the correct properties are placed in the Application specific file extended from **IFPropertyManager**. Refer to Section 5.2.2.1.1.2 Use of initializePropertyManager abstract method for additional details.
3. Implement the **doGetService** and the **doPostService** with application specific processing as described in Section 5.2.2.1.1 The IFServlet and Its Use for additional details.
4. Implement the appropriate **initializeServerConnection** abstract for EJB or CORBA as detailed in Section 5.2.2.1.3 Use of initializeServerConnection abstract method or 5.2.2.2.2 Using the Naming Service - EJB.

5.2.3.2 Horizontal

Servlets are able to perform a wide range of functions. For example, a Servlet can:

- Create and return an entire HTML page containing dynamic content based on the nature of the client request.
- It can call a JSP page to handle the formatting and even gather dynamic data from component calls made in other Servlets.
- Communicate with other server resources, including CORBA components, EJBs databases, and Java-based applications.
- Handle connections with multiple clients, accepting input from and broadcasting results to multiple clients.

5.3 Business Logic

5.3.1 Overview

The Business Logic layer consists of the components that are intended to contain the application processing logic. This logic implements the identified business rules for the given application. The components that execute in the business logic layer do not have any user interface components, as they have no responsibility to interact with the user. Problems sensed with the data should be communicated to the user interface layer through return values from methods that the components in the presentation layer used to display messages to the user.

The terms business logic and business rules are often used interchangeably in reference to code that occupies the functional region between the presentation layer and the data access layer. The Business Logic layer component is stateless in that it does not hold data between method calls. It encapsulates the details of the underlying data structure so that the client doesn't have to be aware of table structures, relational structures, or even the underlying column names. The Business Logic Layer also encapsulates transactions and performs indivisible business transactions with single method calls.

The topics covered in this section are:

Guidelines for the use of EJBs

- Session Beans
- Entity Beans
- Initialization recommendations
- Development issues

Guidelines for the use of CORBA components

- Application Objects (AO)
- Business Objects (BO)
- Initialization recommendations
- Development Issues

Business layer messaging support

- Business Object Document (BOD) use
- Supporting classes and guidance provided by the Integration Framework
- Supporting software templates provided by the Integration Framework

Security Considerations

- Enabling security
- Method level authorization – What is required by the developer

- Obtaining access to the Integration Framework provided security service
- Additional parameter required
- Performing the authorization check
- What to do if the authorization check fails

Design issues related to deployment

- Designing for workload management
- Designing for scalability

Throughout this section code snippets are used as examples that are taken from the test components that have been developed to drive the Integration Framework for testing purposes. For a complete understanding of the recommended use of these technologies, refer to the "Parts Data Collection" CORBA application test components and the "Requisitioning" EnterpriseJavaBean test components. These business logic components execute in the application server and can be used to help the reader understand the details of the topics discussed throughout this section.

In addition to this guide, there are several very useful Redbooks produced by IBM that provide descriptions of the development of enterprise applications in conjunction with IBM WebSphere Enterprise Edition. These Redbooks can be found on the IBM Website at <http://www.redbooks.ibm.com>.

5.3.2 Service Use

The services provided by the Integration Framework to the Business Logic component include:

Table 18: IF Business Logic Component Services

Services	Functionality
IFSServices.jar	Framework level client classes and framework classes provide for use by the Business Logic Components
TextMessageApp	MQ communication support from within a business application deployed on the application server
Logging Facility	Framework classes that provide runtime configurable debug and logging of information for system management support
Info Structure Level Security	Container managed configured security with the Application Server
IFCBSecurity Service	Framework service that provides application level authorization

5.3.2.1 Enterprise JavaBean Components

This section is provided to aid in the development of Enterprise JavaBean (EJB) Components. It is comprised of four subsections: overview, initialization, development issues, and deployment issues. The overview subsection gives some fundamental framework concepts. It contains a high level overview of EJB Components and associated terminology. The intent is not to

introduce the developer to EJB designs but rather to put into perspective the remaining subsection, as they specifically apply to developing and deploying EJBs using the services of the Integration framework.

These subsections will not flow like an EJB design cookbook; they will highlight and provide guidance on issues relating to the Integration Framework's product selection, services requirements, constraints, or techniques. They will include specific code samples on any rules or design constraints imposed on an EJB developer.

5.3.2.1.1 Enterprise JavaBean Component Overview

An Enterprise JavaBean (EJB) is a non-visual component of a distributed, transactional, and possibly persistent enterprise application. EJBs run within the context of a software system called an **EJB Server**. The Component Broker or CB portion of the IBM WebSphere Application Server Enterprise Edition (WAS-EE) product is the Integration Framework's EJB Application Server. An EJB Server may contain one or more **EJB Containers** that are responsible for handling the creation of new instances of the bean, details of persistence, thread safety, transaction support and so on. It is important to stress that within the Integration Framework there is one container for each bean that is in the EJB component.

The container is the environment in which Enterprise JavaBeans execute. Its role is to provide the bean with services, so that the bean can remain blissfully unaware of the underlying mechanisms used in implementing these services. The deployment descriptors are the main way in which information is communicated from the EJB developer to the container in which the bean will be deployed. For example, when the bean is deployed, the container reads the deployment descriptor associated with that particular EJB bean and automatically provides the necessary transactional support. The Enterprise JavaBean Development Issues subsection will address any constraints on container managed services (i.e. naming services and transactional support).

Enterprise JavaBeans come in two fundamentally different types: session beans and entity beans. An EJB component is a set of session beans and/or entity beans.

Session beans are an extension of the client application and are responsible for managing processes or task. They are function-oriented components. They represent a set of behaviors that reside on a server and can be invoked by clients either for a length of a method (in the stateless session bean case) or for the life of the bean (in the stateful case). Since these beans bear a lot of resemblance to standard distributed objects built using CORBA or RMI, most of the same "design patterns" and strategies that apply to these technologies also apply to session bean design.

Entity beans model data entities and provide shared distributed transactional access to persistent data. An individual entity bean represents a single persistent entity – for instance a row in a relational database.

The Enterprise JavaBeans is composed of:

- A Home Interface (used to *create* and/or *find* beans)
- Key Implementation (identity of entity beans)
- The Remote Interface (lists business methods which client can invoke)
- Bean Implementation (the Business Logic)

The Deployment descriptor (setting used during deployment and describes runtime policies)Figure **39: Pictorial Representation of an Enterprise JavaBean Component** graphically describes the enterprise JavaBean in the context of its deployment environment.

Although EJBs are straightforward to write, they can take advantage of advanced services such as concurrency, persistence, and transactional support. EJB design shifts the burden of implementing these services from the shoulders of the application developer to those of the container provider.

Enterprise JavaBeans provide several benefits for application developers:

- Allow the developer to build distributed applications by combining components developed using tools from different vendors.
- Make it easy to write applications. The developer does not need to deal with the low-level details of transaction and state management, multithreading, resource pooling, and other complex low-level APIs. However, if necessary, expert programmers can still gain direct access to the low-level APIs such as user transactions.
- The EJB specification that governs the use of enterprise beans is compatible with other Java APIs and CORBA. It also provides for interoperability between enterprise beans and non-Java applications.

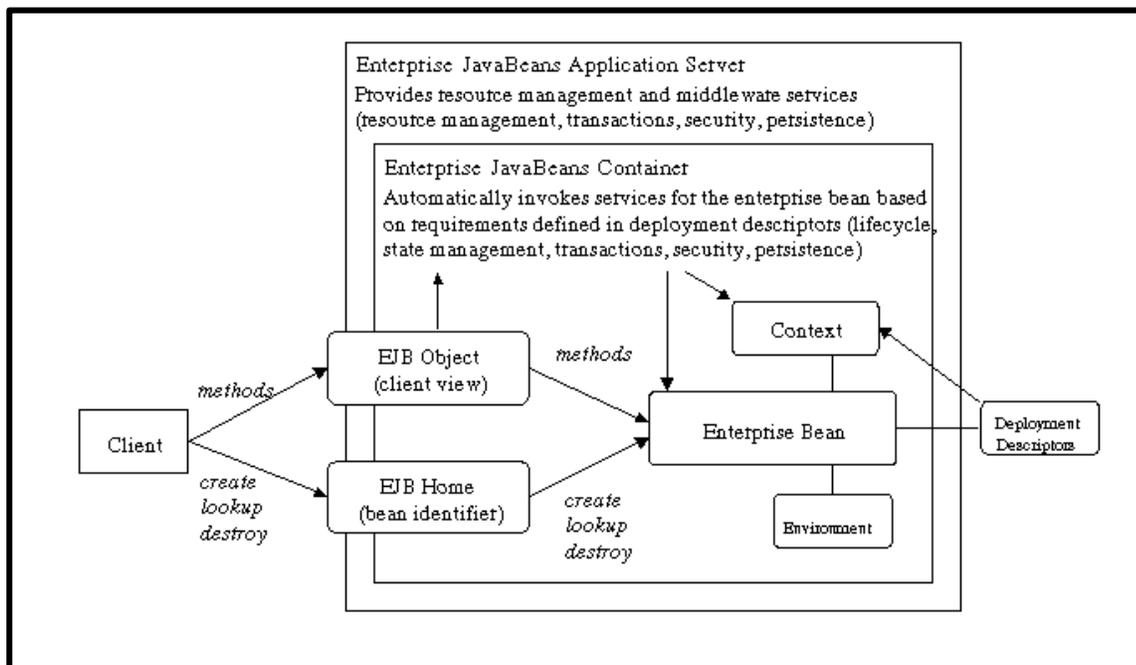


Figure 39: Pictorial Representation of an Enterprise JavaBean Component

EJB development is done using an enterprise JavaBean development tool such as the recommended IBM VisualAge for Java (VAJ) Integrated Development Environment (IDE). The beans are then deployed into the IBM WebSphere Application Server Enterprise Editions development environment, Object Builder. Here the mapping of entity beans to a database and the production of run-time code takes place. Development and deployment issues associated with EJB components are described in detail within the corresponding subsection.

5.3.2.1.1.1 Session Beans

As mentioned earlier, session beans act as controllers for other backend business logic. Usually one or more session beans drive multiple entity beans to do the real work. As their name implies, **Session Beans** are generally tied to the lifetime of a given client session. They are relatively short-lived. Session beans are divided into two basis types: **Stateless** or **Stateful**.

A Stateless session bean is a collection of related services; each represented by a method. The bean maintains no state from one method invocation to the next. When the user invokes a method on a stateless session bean, it executes the method and returns the results without knowing or caring what other requests have gone before or might follow. Think of a stateless session bean as a set of procedures or batch programs that execute a request based on some parameters and return a result. Stateless session beans are used when there is no client-specific state variables. Contrary to popular belief stateless session beans can in fact, possess instance variables. These variables must be shared among multiple potential clients (e.g. they should be read-only).

This means that any two instances of a stateless bean are equivalent. The stateless session bean does not require any arguments on the create method. Any bean instance can be used for any client.

A Stateful session bean is an extension of the client application. It performs tasks on behalf of the client and maintains state related to the client. Stateful session objects are created in response to a single client's request, communicate exclusively with a single client, and are destroyed when the client no longer needs them. They have a preset timeout period. The timeout period is specified in the deployment descriptor. Each time a business method is invoked the timeout clock is reset. If the client fails to use the Stateful bean before it times out, the bean instance is destroyed and the remote reference is invalidated. This prevents the Stateful session bean from lingering long after a client has shut down or otherwise finished using it. Since Stateful beans maintain conversation between methods, they can implement any of several *ejbCreate()* methods that take different sets of arguments. Note the *ejbCreate()* method is essentially analogous to a constructor for Enterprise JavaBeans; it initializes an instance's internal state variable(s).

A session bean does not require a primary key class or find method because the developer does not need to search for specific instances of session beans. They are created to provide a service for a particular client and are not persistent; each client will create its own instance of the session bean.

5.3.2.1.1.2 Entity Beans

Entity beans are essentially a transactional object persistence mechanism. They handle persistence and data access. Entity beans are long-lived; they exist across client sessions, are shared by multiple clients, and remain alive even after a restart of the server or other failure. Typically, an Entity bean maps directly to a row in an underlying database. Within the framework both DB2 and Oracle persistent storage is supported using the product supplied application adapters.

Application developers may develop their own adapters for other persistent stores. Within the world of Entity beans, there are two main types of beans: Entity beans with **Bean-Managed Persistence (BMP)** and Entity beans with **Container-Managed Persistence (CMP)**. An Entity bean with bean-managed persistence contains code provided by the application developer that controls the updates to the underlying database. In contrast, an Entity bean with container-managed persistence contains no such code; instead it relies on the container to update the database as necessary.

In general, Entity beans with container-managed persistence are simpler to implement (because they do not require any imbedded database code). CMP entities are beans whose persistence (for example, the storing and retrieving of their data from a backing store like a Relational Database) is managed by the EJB container. This means that the container would, for instance, manage both generating and executing SQL code to read and write to the RDB.

Bean-Managed persistence EJBs leave the management of such details as what SQL is executed to the developer of the EJB. Each BMP EJB is responsible for storing and retrieving its own state from a backing store in response to specific “hook” messages (like **ejbLoad()** and **ejbStore()**) that are sent to it at appropriate times during its lifecycle.

Whether BMP or CMP, each Entity bean instance has an associated **Primary Key**. Logically, a Primary Key is a value or combination of values that allows the user to uniquely specify a row of data. A client may *create* a new Entity Bean or *find* an existing bean with its associated primary key. The *ejbCreate()* method and *ejbFindByPrimaryKey()* are the corresponding methods in the bean implementation that support creating and finding beans.

While session beans require the *ejbCreate()* method it is up to the application developer of the entity bean whether a *create* method is required. An entity bean implementer may choose to have the entity beans represent objects already in a particular database and to not allow the creation of new ones. In this case the developer would not need a *create* method.

The *ejbFindByPrimaryKey()* method will be generated by the container for CMP beans and must be implemented by the developer in BMP designs.

5.3.2.1.1.3 Enterprise JavaBean Test Component

The **IFTC4ReqComponent** is an EJB Component developed and deployed as a test component for the Integration Framework. It contains a controlling stateless session bean (Requisition) that provides the high-level application interface to the client and coordinates the work of three other beans. The other beans are the **OrderHistory** entity bean, the Inventory stateless session bean, and the Ordering stateless session bean. This Integration Framework test component contains coding samples for various bean design constraints and requirements. The beans:

- Perform application to application level communication using MQ container managed messaging through the MQSeries Application Adapters.
- Utilize the Integration Framework’s Security service for application level authentication.
- Are integrated with the Integration Framework’s Logging service in support of Enterprise System Management.
- Contain runtime configurable debug code.
- Provide both container level and bean managed transactional control.
- Use an oracle database for persistence storage.
- Implement the *create by copy helper* design pattern.
- Has a supporting utility Java class to perform common functions across beans.

They can be referenced to additional details on the snippets of code provided in the initialization and development subsections.

5.3.2.1.1.4 Enterprise JavaBean Initialization

The EJB container is responsible for creating objects that implement the remote interface; these objects act as proxies, forwarding the business method calls on to the object. When a client

invokes a *create* method on the bean's home interface, the container creates an object of the appropriate type and calls a method in the EJB class called **ejbCreate()**. The role of the *ejbCreate()* method is roughly analogous to that of a constructor in an ordinary Java program. It initializes the state of a class with any necessary argument variables.

For the CMP Entity Bean developed in VAJ, the *ejbcreate()* method will only use key fields as parameters. If desired, the **create from a copy helper** design pattern can be emulated by following these steps:

- Remove the default **ejbcreate** from the remote interface.
- Add a new method called **ejbcreate** with parameters for all the CMP fields or a structure containing all fields.
- Set the CMP fields by using the *setter* methods (tool-generated) for the associated field, passing in the corresponding parameter.
- Call the original *ejbcreate* method for the key fields.
- Add the new overloaded *ejbcreate()* method to the remote interface.

By not modifying the tool-generated methods, but extending the functionality by repackaging, the developer ensures the integrity of the default code and logic.

The Framework imposes a requirement on the business logic design to initialize the Logging Service prior to using it. Initialization occurs by calling the following API in the Logging Service: **"PropertyConfigurator.configure (filename);"**

For EJB components there are two recommend design strategies to implement this. One, a component may choose to have the session bean, supporting the control of the application, perform this at EJB creation time. The other style is to have a service or utility Java class that is shared amongst the beans within the component, perform this operation at elaboration time (in the construction). This latter option is the way the requisition test component (**IFTC4ReqComponent**) was implemented. All beans within the requisition test component have an instance variable of the **RequisitionUtilites** class. The call to initialize the logging facility was gated within the constructor of the **RequisitionUtilites** class so as to perform this call once for the EJB component.

Which strategy used depends on factors such as: whether the developer will need an *ejbcreate* method, the number of *ejbcreate* methods, or whether there is a need for a utility class. It is up to the EJB developer to determine an approach that ensures that the Logging Services is initialized once and only once for the EJB component and prior to any calls to log information.

5.3.2.1.1.5 Enterprise JavaBean Development Issues

As mentioned earlier, Visual Age for Java Enterprise Edition is the recommended development environment to support Enterprise JavaBean development and deployment. It provides a full Java development environment and also provides the necessary links and mechanisms via WebSphere's Object Builder (OB) to facilitate the bean deployment to the WebSphere

Application Server Enterprise Edition. Deployment of all EJB within the VAJ – CB environment follow a normal procedural flow. This process is well integrated and repeatable within the two environments. The VAJ online documentation takes the user through the VAJ to OB process in a step by step approach.

In the normal process for the development and deployment of EJBs to the Component Broker environment:

1. Beans developed and the business logic tested within the VAJ environment. Note that beans using the Framework Business Object Documents (BODs) will not be able to test in the VAJ WebSphere tool test environment. There is a conflict between the tools **org.omg.w**.
2. Jar files are built and exported via VAJ-EE to the Component Broker's Object Builder where the container and associated deployment descriptors are mapped from the VAJ specific environment to the necessary Corba specific library environment required for CB deployment. Additional files are created that are required by the EJB server, CB, to manage the EJB. The level of automation in this process varies and will be described in more detail in the deployment subsection.
3. The EJB application family is then deployed to Component Broker via the Component Broker's Make process.

There are several development areas of concerns imposed on the EJB developer because of this development and deployment paradigm. The intention of this subsection is to provide guidance and techniques for developing within this environment. These constraints include preparing the VAJ development environment, finding object homes, and transactional control is contained within this subsection.

Preparing the development environment

The EJB development environment with VAJ will need to have the following steps performed in order to build Integration Framework deployable enterprise bean components:

- Add Component Broker Client classes to the workspace. The batch file **createCBjar.bat** is an NT based solution for creating a CB.jar file containing all the packages and classes needed to deploy an EJB in the WAS-EE. This executable can be found on the SSG web site. This batch process uses the somojor.zip file provided in the <drive>:Cbroker/lib directory of the application server. It extracts all classes associated with a set of twenty-two packages and creates a CB.jar containing all these classes. This jar file can then be imported into the VAJ workspace; note since it modifies two standard VAJ packages it must be imported into each developers' VAJ workspaces.
- Add Integration Framework classes to the workspace. The VAJ projects associated with any service (i.e. BODs, MQ messaging) will need to be imported into the workspace.

- Add any client classes the developer will be communicating with component to component.

How EJBs find beans within their component

The Java Naming and Directory Interface (JNDI) is used to find the name of an EJB home object. By EJB specification, the *InitialContext* class serves as the client's interface to the JNDI. It contains bindings to a variety of naming services (JNDI, CORBA **CosNaming**, DNS, and so on); providing a single interface that can be used to link to any naming services in the client's environment that supports JNDI. This is especially important for developers of Integration Framework EJB components since they will be developing and preliminary testing in the VAJ WebSphere Test Environment with the JNDI naming service and deploying within WAS-EE using the CORBA common object service naming service.

What this means for an EJB developer is that bean homes will be found using the JNDI independent of the naming services and let the JNDI property values dictate which naming services to use. The developer will need to externalize these properties (**providerUrl** and **initialContextFactory**) in environment variables, property files, or resource bundles rather than hard code them into the enterprise bean code. Thereby, allowing the coding logic to remain the same; it is the value of the properties that initialize the JNDI that will dictate which service to use. The requisition test component contains several examples of locating a beans home.

This sample component used the resource *bundle* class to make runtime configuration changes. The **InitialContext** value was dependent on the naming service used. As mentioned previously, the VAJ WebSphere Test Environment uses the JNDI naming service and the WAS-EE uses the CORBA Common Object Services for its naming service. What this translates to the bean developer is different values in the bundle property file for the **InitialContextFactory** key. For example when the requisition component was deployed in ObjectBuilder the property file contained an entry of **initial_context_factory = com.ibm.ejb.cb.runtime.CBCtxFactoryHostWidenedDefault**. While the entry of **initial_context_factory = com.ibm.ejs.ns.jndi.CNInitialContextFactory** was used when testing in the development environment. Whether the developer is testing in VAJ WebSphere Test Environment with the JNDI naming service or at deployment time within WAS-EE using the CORBA COS naming service the value of the initial context factory is what will be adjusted to specify which naming service to use. A sample of code utilizing a resource bundle to initialize the JNDI is shown in Figure 40: Locating and creating an EJB Home.

When performing a JNDI lookup on an enterprise bean deployed in the Component Broker application server the JNDI home name passed to the *lookup* method is the JNDI name specified in the enterprise bean's deployment descriptor abstractor from the beans property page in VAJ.

When using Session Beans, it is a good idea to make the reference to the EJB object as a class-level variable rather than a variable that is local to a method. This allows the EJB client to repeatedly invoke methods on the same EJB object rather than having to create a new object each

time the client invokes a *Session Bean* method. This approach is not recommended for Servlets, which must be designed to handle multiple threads.

```
...  
  
java.util.PropertyResourceBundle bundle =  
    (PropertyResourceBundle) PropertyResourceBundle.getBundle("/bundleFilename");  
java.lang.String initialContextFactory = bundle.getString("initial_context_factory");  
java.lang.String providerUrl providerUrl = bundle.getString("provider_url");  
  
java.util.Properties p = new java.util.Properties();  
p.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY, initialContextFactory);  
p.put(javax.naming.Context.PROVIDER_URL, providerUrl);  
  
// Create the initialContext Object  
initialContext = new javax.naming.InitialContext(p);  
...  
// Using the initialContext object create the EJB home  
YourBeanNameHome BeanHomeVar = null;  
Object homeObject = initialContext.lookup(bundle.getString("Your_Bean_home_name"));  
BeanHomeVar = (YourBeanNameHome)  
    Javax.rmi.PortableRemoteObject.narrow((org.omg.CORBA.Object) homeObject,  
    YourBeanNameHome.class);  
// Using the EJB home create an instance of the bean  
YourBeanName BeanNameInstance = BeanHomeVar.create();  
...  
...
```

Figure 40: Locating and creating an EJB Home

How EJBs find other components within the EJB Server

An EJB component should locate all services or components deployed within WAS-EE with the CORBA COS naming services. The basic steps are as follows:

- Get the factory finder, specifying the searching scope
- Resolve the string name to a CORBA object
- Narrow the object to get the home

Figure 41: Locating a WAS-EE deployed component provides a snippet of code for locating a deployed object from within an Enterprise JavaBean component. This design philosophy removes any dependency of “*type of the component*” (i.e. EJB or CORBA) from style of communication. All components deployed within the WAS-EE will locate other components using the CORBA COS naming service.

```
....  
// get the factory finder  
java.lang.String ffPath = "host/resources/factory-finders/" +  
    com.ibm.CBCUtil.CBSeriesGlobal.serverName() +
```

```
        "-server-scope-widened");
org.omg.CORBA.Object iObj =
    com.ibm.CBCUtil.CBSeriesGlobal.nameService().resolve_with_string(ffPath);
theFactoryFinder = com.ibm.IextendedLifeCycle.FactoryFinderHelper.narrow(iObj);

// resolve the String name to a CORBA object
java.lang.String TMOutboundClassName = bundle.getString("TMOutboundHomeName")
java.lang.String ffpPath = TMOutboundClassName + ".object interface";
org.omg.CORBA.Object obj = theFactoryFinder .find_factory_from_string(ffpPath);

// narrow the Object to get the Home
outboundMsgHome =com.ibm.IMessageHome.OutboundMessageQueueHelper.narrow(obj);
....
```

Figure 41: Locating a WAS-EE deployed component

Transactional support

Applications use transactions to group related updates to data such that all of the updates occur or none do. Typically, an application: Starts a transaction, makes the updates and associates them with the transaction, and terminates the transaction. When an application terminates a transaction, it can request that the transaction is either rolled back or committed. If the application requests rollback, all of the updates it has made are undone. If the application requests that the transaction is committed, the Transaction Service checks that each object involved in the transaction is able to make its updates permanent. If all objects indicate that they can, the transaction is committed. Otherwise, the updates are undone just as if the application requested a rollback. The result of a transaction (that is, whether it is committed or rolled back) is referred to as its outcome. The transaction attribute, associated with the bean, is set as a deployment descriptor in the VAJ development environment.

When writing the code required by an enterprise bean to support transactions, remember the following basic rules:

- For transactions, a session bean can either use container-managed transactions or implement bean-managed transactions; entity beans must use container-managed transactions.
- An instance of a stateless session bean cannot reuse the same transaction context across multiple methods called by an EJB client; Stateless Session Bean's methods are independent of each other. Therefore, it is recommended that the transaction context be a local variable to each method that requires a transaction context.
- An instance of a Stateful session bean can reuse the same transaction context across multiple methods called by an EJB client. Therefore, make the transaction context an instance variable or a local method variable at the developers discretion. (When a transaction spans multiple methods, the developer can use the **javax.ejb.SessionSynchronization** interface to synchronize the conversational state with the transaction).

- If an enterprise bean begins a transaction, it must also complete that transaction either by invoking the *commit* method or the *rollback* method. Also, an EJB Bean can not begin a second transaction until the first transaction has been committed; this will cause an exception to be thrown indicating nested transactions.
- In the EJB server (CB) environment, a Stateful session bean that implements the **TX_BEAN_MANAGED** attribute must begin and complete a transaction within the scope of a single client method call.
- The EJB server (CB) utilized in the Integration Framework does not support the setting of the transaction attribute for individual enterprise bean methods; the transaction attribute can be set only for the entire bean.
- The MQ Application Adapter interface, which is used by EJB for application to application level communication, requires all messaging to be encapsulated within a transaction. The user can explicitly control this in a session bean by setting the transaction attribute to **TX_BEAN_MANAGED** or let the container manage the transaction by setting the transaction attribute to **TX_REQUIRES**. The developer must also design with the constraint that synchronous communication with MQ cannot be done within a single transaction. That is an application can't be designed to send a message and then wait for a reply within the same transaction. The reason for this is within the transaction scope the message will not be placed on the queue until the transaction is committed. However the developer would not commit the transaction until the message is received and processed successfully. If the developer designed the request reply communication within a transaction scope the transaction would always take the rollback path.

The transaction attributes that have restrictions in WAS-EE are:

TX_BEAN_MANAGED -It notifies the container that the bean class directly handles the transaction. The developer must write the code that explicitly demarcates the boundaries of a transaction. . This attribute can be specified only for session beans and it cannot be specified for individual bean methods.

TX_SUPPORTS and **TX_NOT_SUPPORTED**- Not supported in the EJB Server (CB).

There are three basic steps involved in a bean managing a transaction (see Figure 42: Bean managed transaction for an example):

1. The enterprise bean class must set the value of the **javax.ejb.SessionContext** object reference in the *setSessionContext* method. The setting of this value is done on the developers behalf when the client calls the *create* method on the home interface in the EJB Server environment (CB).
2. Create a **javax.transaction.UserTransaction** object by calling the *getUserTransaction* method on the **SessionContext** object reference to obtain the current transaction reference.

3. The **UserTransaction** object is used to participate in the transaction by calling transaction methods, such as *begin* and *commit*, as needed.

```
import javax.transaction.*;
...
public class MyStatelessSessionBean implements SessionBean {
    private SessionContext mySessionCtx = null;
    ...
    public void setSessionContext(.SessionContext ctx) throws RemoteException {
        mySessionCtx = ctx; // Note: This attribute is set by the session bean's
                            // container at creation time
    }
    ...
    public float doSomething(long arg1) throws RemoteException {

        // Use userTran object to call transaction methods
        UserTransaction userTran = mySessionCtx.getUserTransaction();
        ...
        userTran.begin();

        // Do transactional work
        ...
        userTran.commit();
        ...
    }
    ...
}
```

Figure 42: Bean managed transaction

5.3.2.1.1.6 Enterprise JavaBean Deployment Issues

Once the developer has completed developing and preliminary unit testing in the VAJ environment they will deploy the beans in WAS-EE.

If the developer plans to deploy the enterprise beans to WebSphere Application Server, Enterprise Edition, they need to be exported to an EJB JAR file composed specifically for the Component Broker (CB) deployment tools. This is an option on the export menu in the VAJ bean development environment.

Component Broker contains an EJB Deployment tool for deploying beans from one development environment into the Component Broker development environment. The EJB Deployment Tool works with Object Builder to create and compile the files required by the EJB server (CB) to manage an enterprise bean. The EJB Deployment Tool introspects the EJB JAR file, paying attention to the EJB home, the EJB object classes and the deployment descriptors. The EJB Deployment Tool generates a model that Object Builder uses to create the necessary deployment library files. The output of this process is a set of server side and client side JAR and library files.

Visual Age for Java provides an option, which automatically launches component broker at the time the developer exports the bean component to the EJB Jar file. Component Broker's EJB Deployment Tool will then take the developer through a set of screens allowing them to map the VAJ developed beans for deployment within the WAS-EE. If the workstation does not have the WebSphere Application Server, Enterprise Edition installed, the developer can launch the CB deployment tools separately after exporting the EJB jar file on another machine containing Object Builder. This entire process is explicitly defined in the VAJ online help in deploying EJBs. Depending on implementation, some data types may have to be remapped. If the developer wishes to deploy the CMP beans to Component Broker, be aware that VisualAge for Java converters other than **VapTrimStringConverter** are not directly supported there. However, the Object Builder tool can automatically handle the mapping of most basic Java data types in preparation for deployment to Component Broker. The safest course of action is to check the schema created in Object Builder.

At this point in the deployment process all business logic components utilize the same process for deploying in the WAS-EE. The EJBs and the CORBA components are identical. This generic process is documented in Component Brokers Online help. It takes the developer through the steps running the generated make files and deploying using the system manager user interface.

If the EJB component utilizes any of the Integration Framework services the developer will need to have **IFSSERVICES.jar** in classpath for code generation and at runtime.

5.3.2.2 Corba Components

5.3.2.2.1 Corba Components Overview

CORBA Component Based applications are specifications for interfaces. The CORBA Interface definition language (IDL) is one of the important aspects of the CORBA specification. IDL is an object-oriented language that identifies object classes, class hierarchies, and methods to provide particular services. An interface between components is written independent of operating systems and programming languages and thus provides great flexibility to developers working in environments with different platforms and development tools. The IDL file is compiled using the CORBA **IDL** compiler, which generates the stubs and skeletons required for a distributed application. These stubs and skeletons are a set of interfaces and classes from which object implementations and complete applications are built.

CORBA's Object Request Broker (ORB) locates objects, manages connections and communications between stubs and skeletons, and invokes object methods on behalf of the client.

CORBA Server Test Components provided with the Integration Framework were developed and deployed using IBM's WebSphere Application Server Enterprise Edition (WAS-EE). One of the major product components of WAS-EE is Component Broker (CB). CB is one of the most complete and integrated implementations of the Common Object Request Broker Architecture

(CORBA) initiative.¹ CB provides an environment for distributed computing at the enterprise level. CB is an Object Server that consists of both a runtime package called the Component Broker Connector (CBConnector) and a development environment called VisualAge Component Development for WebSphere Application Server Enterprise Edition (CBTools).

The runtime package provides a server in which business object components run and is managed through a set of management tools. These components run in a robust multi-threaded server, which provide easy access to a wide variety of services and capabilities. For more detailed information on the Object Services refer to *WebSphere EE Programming Guide Advanced*, SC09-4443-00. These object services are available in an integrated fashion based on the OMG model and the Component Broker Managed Object Framework (MOFW). Behind these high-level Component Broker services are the CORBA Object Services used by the server frameworks. Server frameworks shield application developers from low-level interfaces to system services and CORBA services and help to make application objects portable across platforms.

5.3.2.2.2 Corba Component Initialization

Java Corba components can make use of the **log4j** logging facility provided with the Integration Framework. The logging facility must be initialized before use, that is, before any method of the component logs a message using the logging facility. The Component Broker framework provides a Corba component method called *initForCreation()*. It is recommended that component-specific initializations occur in this method. The use of the logging facility typically requires initialization of a static variable, which can and does occur in this method for the PDC test application component provided with the Integration Framework.

When the *initForCreation()* method is overridden, the Object Builder tool will produce warning messages to this effect at the time it checks the model for errors. To the extent that these messages are produced by the overriding of **initForCreation()** for initialization of logging, these messages are expected and can be safely ignored.

5.3.2.2.3 Corba Component Development Issues

5.3.2.2.3.1 No framework logging from C++ components

In the realm of Component Broker application components, it should be noted that logging facilities are provided by the Integration Framework only for Java components, not for C++ components. This is because the version of the C++ compiler that ships with Component Broker does not support a language feature (specifically, the *namespace* keyword) used by the C++ port of the logging facility. Consequently the logging facility cannot be used for logging in C++ application components that get deployed in Component Broker.

¹ Component Broker Information Library

5.3.2.2.3.2 Use of threads in request/response messaging

When using the Request/Response messaging style from within a Component Broker component and using the MQAA, one plausible approach might be to send a message and immediately issue a blocking read of the response message. It turns out that this approach does not work using Component Broker 3.5 and the accompanying MQAA because the thread in which the message was sent and in which the response is being read never relinquishes control. As a result, although the message does get sent, the response never arrives because the responder program, which runs in a different thread in the application server, does not get control while the first thread is waiting for the response. This appears to be an artifact of how the application server containers work. The workaround is to code the sender so that it explicitly relinquishes control, say via **Thread.sleep()**, as it is waiting for the response message. This allows the responder thread to execute and the response to arrive, ready to be read by the first thread.

5.3.2.2.4 Corba Component Deployment Issues

Note Future Capability: The System Management User Interface (SMUI) is the tool used to create, configure and deploy Applications and Servers. It is important to know what to name the ServerGroups and Servers since the Application code and Clients are dependent on it to locate the Homes.

In the PDC application set, several elements of the application have influenced the way the servers are configured and named and how the factory-finders are used to locate the needed homes. We have the concept of 'location' which determines what data is accessed. The **PartsDataCollection** and **Part** objects are not dependent on the actual database they access. That database is specified only when the servers supporting these classes are defined in the System Management EUI. Since a single installation can support servers accessing more than one 'location' (ENTERPRISE, BASE1, BASE3 and so on) specific database, the name assigned to the servers reflects the 'location' that the server supports. Depending on whether the Homes are workload managed or not, we have used different factory-finders.

- Non-workload managed

The **PDCSessionAO**, which receives *location* as an argument, can locate appropriate objects, by using 'server-scope' factory finders, constructing the full name from the location value. Thus the Server that supports the ENTERPRISE location is named **ENTERPRISEPDCServers** and the Server Group that includes all these servers is named **ENTERPRISEPDCSvrGrp**.

E.g. "**host/resources/factory-finders/<location>PDCServers-server-scope**"

Note Future Capability: Where location = ENTERPRISE, BASE1, BASE3 and so on

Thus, the Server Names are **ENTERPRISEPDCServers**, **BASE1PDCServers** and so on.

Other objects used by the **PDCSessionAO** (e.g. the **IFCBSecurityInfo**, **TMinbound** and **TMOutbound** classes) do not need to come from the same server, though, if they are available there access to remote objects may be avoided. Note that the Messaging application is configured with our Server so it would be found at the server-scope level but the Security Server is configured as a separate Server, so to locate both objects using the same factory-finder, the option is to widen the search. Thus we used the following factory-finder `<ServerName>-server-scope-widened`.

Note Future Capability: E.g. "host/resources/factory-finders/<ServerName>-server-scope-widened"

where `<ServerName >` is the current server.

We can substitute `<ServerName>` with the function "CBSeriesGlobal.serverName()" which returns the name of the Server in which the code is running..

For Clients to locate the home for the **PDCSessionAO**, the server-scope factory-finder is used.

-Workload managed

Figure 43: Example of `host/resources/factory-finders/<ServerName>-server-scope-widened` Code

To support Workload Management, the modifications were made to **PDCSessionAO** and **PartsDataCollection/Part** objects. **PDCSessionAO** is defined to have an UUID key such that it is not restricted to one instance per 'location'. Each user will get a separate instance of the **PDCSessionAO**, which controls the transactions. An additional AO, **PDCAO** is also defined to have an UUID and is created to support **PartsDataCollection/Part** objects.

The **PDCAO** contains all the Business logic and supports all the query and collection management across the **PartsDataCollection** and **Part** objects. The relationship between **PartsDataCollection** and **Part** objects is still maintained. Also the **PDCSessionAO** and **PDCAO** object classes have been defined with WorkLoad Managed homes but the instances of the classes are NOT workload managed. Thus the homes for these classes are registered in the namespace under the `<ServerGroupName>-server-scope'` as described above. The ServerGroups and Servers are named such that the Client can easily locate the proper home for these object. Therefore the Server that supports the ENTERPRISE location is named **ENTERPRISEPDCSvr<nn>** and the ServerGroup that includes all these servers is named **ENTERPRISEPDCServers** where **nn = 01, 02** and so on. Similarly the location BASE1 results in **BASE1PDCSvr<nn>**, and **BASE1PDCServers**, etc. Thus the **PDCSessionAO** can locate appropriate objects by using 'server-scope' factory finders, constructing the full name from the location value.

Note Future Capability: Since the **PCDSessionAO** object needs to access **PDCAO** objects, which are configured to support a specific 'location', this class uses the

<**ServerGroupName**>-server-scope factory-finder to locate the home for the properly configured class.

E.g. “**workgroup/resources/factory-finders/<location>PDCServers-server-scope**”

where location = ENTERPRISE, BASE1, BASE3 and so on.

Thus, the **ServerGroupNames** are **ENTERPRISEPDCServers, BASE1PDCServers.**

The PDCAO also needs to locate the home for the **PartsDataCollection** and Part objects that are configured to the 'current location', i.e. the location that was used to locate the PDCAO object's home. Since all three of these classes are packaged in the same 'application' they will all be supported by any server that includes the PDCAO class. The strategy used to locate the home uses the function **CBSeriesGlobal.serverName()** (which returns the name of the server in which the code is running) and constructs the name for the proper factory-finder from that value.

E.g. “**workgroup/resources/factory-finders/<ServerName>-server-scope**”

where <**ServerName**> is the current server.

We can substitute <**servername**> with the function **CBSeriesGlobal.serverName()** which returns the name of the Server in which the code is running.

Clients of the PDC Application set need only locate the home for the **PDCSessionAO class**, and that home can be located using any of several factory-finders,

E.g. “**workgroup/resources/factory-finders/PDCSessionServers-server scope**”

where servergroupname = PDCSessionServers

This factory-finder is the most restrictive, though, because this class is 'visible' in the workgroup and cell, the 'workgroup-scope' and 'cell-scope' factory finders will also find the needed home.

5.3.2.3 Business Object Document Support

The Open Application Group's (OAG) Business Object Document (BOD) is the architecture used to communicate messages or business documents between software applications or components. Each Business Object Document includes supporting details to enable the destination business application to accomplish the action.

Architecturally, the Business Object Document consists of two areas, shown in the following figure:

- Control Area
- Business Data Area

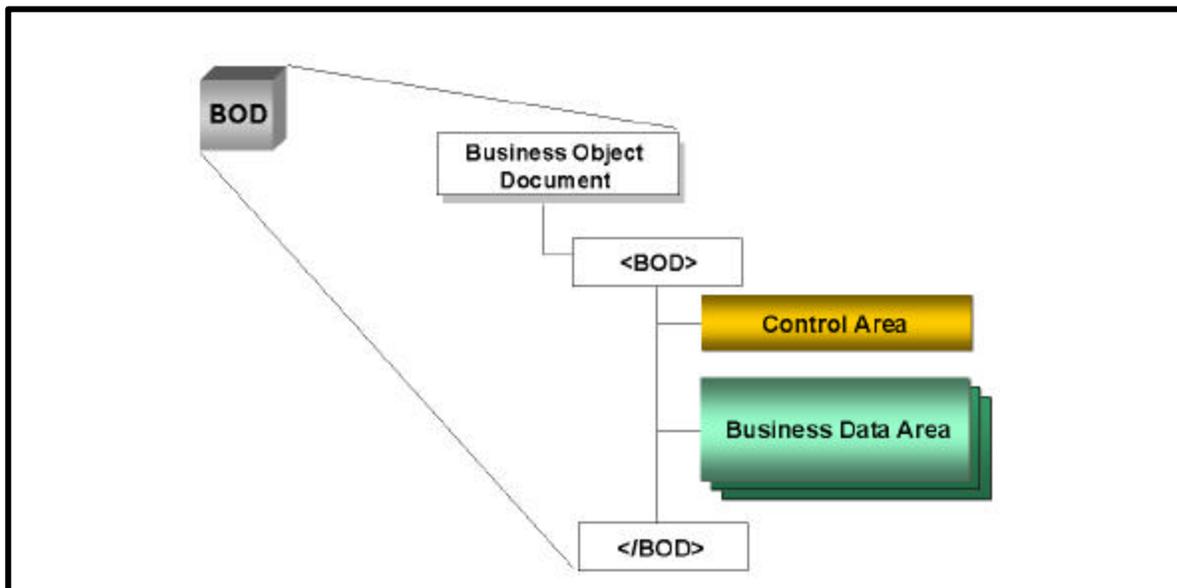


Figure 44: BOD Structure

Business Object Documents are represented as Extensible Markup Language (XML) documents. The OAG provides the definition of the content of each of the BODs defined for application to application communication. Guidelines are also provided that allow the USERAREA contained within the Business Data Area to be extended, if required, to include additional BOD data while still maintaining BOD commonality.

Complete sets of Document Type Definition (DTD) files are available from the OAG web site at <http://www.openapplications.org/>. The DTD is the formal definition of the BOD and is used by the parser to validate the overall structure and content of a message. Simply put, the DTD acts as a template used to define the message structure and relationships between the data elements. In some cases, the DTD information is embedded inside the XML message. For the purposes of this OAGIS XML implementation, DTDs will always be deployed as a set of separate files.

This OAGIS XML solution is implemented as a set of three resource DTDs, and an additional DTD for each business service request. The resource DTDs include information that is common across all BODs. These resource DTDs are used to define Data Types, Fields, and Segments that are used in the BODs. A small number of complex data types, or Super Segments are also defined. The service request support files define individual request unique definition and structure, as well as additional attributes or element restrictions that are not defined in the resource files. Changes in the definition of a BOD to support the development of a particular application must be reflected by the developer in the appropriate DTD(s).

The Integration Framework provides a Java base class that presents an implementation of the basic structure of a BOD. This class allows the control area and a generic data area to be created. *Setter* and *getter* methods are also provided for the data elements that make up the

control area. It is the responsibility of the application developers to create any extension classes that are required for building and parsing the data area of the any additional BODs required by the application.

Refer to the Open Applications Group Integration Specification (OAGIS) for additional details on the structure and content of the Business Object Documents defined by that group as well as rules governing their use. To obtain a copy of the OAGIS document, or any additional information about the Open Applications Group please refer to <http://www.openapplication.org>.

5.3.2.3.1 Business Service Requests

A Business Service Request (BSR) is the component of a Business Object Document (BOD) that a sender uses to describe a requested action by the recipient. A BSR is made up of the noun, the verb, and the revision. Every BOD contains one, and only one, BSR. . Each of the nouns and verbs, as defined by the OAG, are also described in the OAGIS document.

Verb

The Verb is the actual service to be performed. The Verb can be thought of as the action verb of the Business Service Request.

Examples of a verbs used in BSRs are:

- **SYNC,**
- **SHOW,**
- **GET,**
- **LIST,**
- **Etc.**

Noun

The Noun indicates the object the service is to be performed on, such as G/L Journal Entry. The Noun can be thought of as the action item of the Business Service Request.

Examples of nouns used in BSRs are:

- **INVENTORY**
- **ITEM**
- **PO**
- **SALESORDER**
- **Etc.**

Revision

Revision is used to identify the version of the Business Service Request. The version is a three-digit field, beginning with **001** and incremented each time the Business Object Document

specification is changed. Each BOD has its own revision number to specifically identify the level of that BOD, not just the release version of the OAGIS specification.

Example verb-noun-revision combinations used by the test components supplied with the Integration Framework are:

- **SyncInventory003**
- **UpdateInventory003**
- **AddRequisitn002**
- **GetlistIem001**
- **ListItem001**

The Integration Framework provides a base class to be used for the creation of BODs. This base class implements the *getter* and *setter* interface methods necessary to build the control area of the BOD. Part of that control area is the Business Service Request.

5.3.2.3.2 XML Usage

As mentioned above BODs are represented as XML documents. XML documents consist of elements delimited by tags, attributes of those elements, and data, in a well-formed order. Elements in the document can also be nested. The element tags, nesting, attributes, and order are defined in a DTD. An XML document can be classified as well-formed, or valid. Well-formed documents respect the syntax – proper begin and end tags, proper nesting, etc. Valid documents not only are well-formed, but they also follow the structure detailed in the DTD. The BODs used in the IF must be valid. An XML Parser is used to check an XML document to determine if it is just well-formed or valid. The IF uses the IBM Apache XML parser for this purpose.

The Document Object Model (DOM) is an API for XML and HTML documents. The W3C specification of DOM level 1 provides a low-level set of fundamental interfaces that can be used to access and manipulate any part of a structured document. In brief, XML documents are considered as data and DOM is used to access that data. In the IF, the DOM interfaces are used in the BOD classes to get, set, add, and delete elements in the XML BOD. The parser will break the BOD into a DOM tree, and then the programmer uses these relations from DOM to get to the specific elements. If the BOD was left as a string instead of being parsed in to a DOM, the programmer would have to repetitively scan the string looking for tags, process the data within the tags, then do it again for the next tag.

The recommendations for the specification for XML as well as the Document Object Model are available from the World Wide Web Consortium (W3C) at their web site: <http://www.w3c.org>.

5.3.2.3.3 DTD Repository

The Document Type Definitions (DTDs) for the BODs are located in a repository. Each BOD has a unique DTD. The repository also contains the three resource DTDs provided by the Open Applications Group.

It's recommended that each processing center, at the sending and the receiving ends of the communication, have its own copy of the repository that is stored at a given URL on the local Web Servers. This URL is "**gcss-af_dtd_repository**". This URL has to be the same at each site, and requires an entry in the Domain Name Server (DNS) for that site. This allows the DTD to be resolved and located locally and eliminates the need for an external repository.

A few things to keep in mind when dealing with a DTD repository are:

- If the URL is changed, the change must be reflected in the BOD DTDs, as they contain a hard-coded reference to the location of the BOD DTD in the DTD repository.
- If the structure of a BOD gets modified, for example a new element is added to the customer defined **USERAREA**, the DTD for that BOD must be distributed and deployed to all the repositories.
- If a new BOD is added, the corresponding DTD must also be created and added to the repositories at all ends of the communication.
- If DTDs on each end of the communication do not match, the receiving end will not be able to parse the BOD that was created on the sending end and will then be unable to access the data contained in it.

5.3.2.3.4 Supporting Classes and Templates

Provided with the IF is the BOD base class (**BOD.java**) as well as the application specific BODs which are extensions of that class used by the various test components provided with the IF. Several extended classes to the basic BOD class are provided as examples for application developer use. These are **ConfirmBOD**, **GetListItemBOD**, **ListItemBOD**, **SyncInventoryBOD**, and **UpdateInventoryBOD**.

The base BOD class consists of the methods to create, parse and access the basic elements of the XML document. The *getter* and *setter* methods for elements in the Control Area of a BOD are in this class. The extension classes contain the *getter* and *setter* methods for each element in each of the specific BODs. These methods use the access methods in the base class to manipulate the unique elements of that particular extended BOD class. For example, the methods in the base class would allow manipulation of XML BOD elements in the DOM tree structure relative to a particular branch in that tree, such as **get2deep()**. This will start at the current node and traverse both horizontally and vertically through the child/levels, 2 layers deep. The methods in the extension classes would employ those base class methods.

As an example, a *getter* method such as, **getItem()**, would position the node pointer using the XML parser DOM methods to the correct document node as specified by the BODs DTD, then pass the child/levels to the **get2deep()** base class method and return the value of the BOD element "Item". The structure of each of the *getter* methods, for each of the elements of that particular BOD, would be similar, first position the node pointer, then pass along the child/level information to the appropriate base class method. There are also methods specific to each

extended BOD class that enable the developer to build the tree structure document for that specific BOD and also methods to traverse that particular structure. An example of such a method is *findElement()* in the **ListItemBOD** class. The BOD programmer must pay very close attention to the definition of the structure of each BOD while coding the creation and traversal methods in order to ensure the BODs will parse correctly.

The method for parsing and validating a BOD is in the base BOD class string constructor. But each of the extended classes will use that base class parser method to verify the XML BOD. Then the extension class will set the node pointers for the class specific data area nodes in that BOD in order for the *setter* and *getter* methods to start at the correct node position.

5.3.2.3.4.1 Extension of BOD base class

The BOD base class provides an implementation of the control area of a BOD. The control area is common across BODs of different types. However the data areas of different BODs differ. The extension of the BOD base class provides the BOD-specific implementation of the data area.

An application developer would extend the base class to create a new BOD class by first subclassing the BOD base class defined in **BOD.java**. Then *setter* and *getter* methods would be added to the new BOD which navigate the DOM hierarchy of the BOD as it is defined in the BODs formal specification. These methods allow applications to set or get the values of the attributes of the BOD.

5.3.2.4 Security

Refer to Section 6 Securing the Application for overall guidance on security. The following section addresses aspects of security that are the responsibility of the MAs using EJB Components and CORBA Server Components.

Table 19: EJB & CORBA Server Components Security Activities an extraction from the full table in Section 6 Securing the Application, identifies the activities that are the responsibility of the MA engineering team. Except for the actual programming aspects, all of the activities identified herein need to be coordinated with the GCSS-AF Operations and Support Organization and the GCSS-AF Security Organization.

Table 19: EJB & CORBA Server Components Security Activities

Security Area	Activity
Authentication	Define component identities <ul style="list-style-type: none"> ◆ Components that are not invoked directly or indirectly by an end user may need to authenticate to the system as a component identity. ◆ Components that need to act on behalf of a user to invoke methods that the user would not normally be given carte blanche access to may need to authenticate to the system as a component identity.
Access Control	Define groups/roles and object access control rules (ACLs) <ul style="list-style-type: none"> ◆ Refer to 6.3 Access Control for additional details on the tasks involved.

	Using API(s) as needed for access control ♦ Refer to 6.3.5 Access Control and Web Objects for details.
Audit and Alarms	Generation of non-COTS security audit records ♦ This step is required only if MA requires audit beyond what IF provides. Refer to 6.7 Audit and Alarms and in particular 6.7.1 Logging Framework Security Events for additional details.

5.3.2.2.5 Authentication

The GCSS-AF Integration Framework relieves the MA from providing a separate authentication mechanism. The MA has the responsibility for trusting that WebSEAL has properly authenticated a user and accepts the identity that is passed to it as the originating identity for all actions. The MA will use this identity for Access Control and Audit requirements.

The MA has the responsibility for defining component identities in the scenarios identified in Table 20: MA Authentication Scenarios. This is not meant to be an exhaustive list, but merely to stimulate the MA to think about how access control and the authentication mechanisms as part of its design task.

Table 20: MA Authentication Scenarios

Situation	Example
Components that are not invoked directly by an end user may need to authenticate to the system as a component identity.	For example a trigger application that reads and processes messages off of a queue may not have access to the originating user information in a form that would allow it to make specific authorization decisions based on that originating user. The methods that the application is invoking require an identity to make its access control decisions. Another example is a wrapped Legacy application where a file is FTP into a directory. An MA can process the file and invoke secured methods using the component's identity.
Components that need to act on behalf of a user to invoke methods that the user would not normally be given carte blanche access to may need to authenticate to the system as a component identity.	A Finance MA may provide a method to checkForSufficientFunds on an account based on an account number and a dollar amount. The finance MA doesn't want to open this method up to all end users as this may allow unauthorized users to gather information about other accounts. The finance MA creates another method called <i>userCheckForSufficientFunds</i> that verifies that the user has access to that account before providing feedback. Users are allowed access to this method and a component identity is created that is allowed access to the original <i>checkForSufficientFunds</i> method.

The IF provides the *gcssafAuthenticateUser* method that is part of the **IFCBSecurityInfoAuthz Object** in Component Broker to perform this function for the application. The **sessionInfoStruct**

created by this function can be passed as a parameter to CORBA and EJB methods just as the **sessionInfoStruct** created for the user and passed in by the Servlet or EJB. To make applications work properly, it may be necessary to add additional parameters, as required by the invoked application. For example, the *location* and *basenamequalifier* parameters of the **sessionInfoStruct** are required by the test components.

```
com.ibm.IBOIMExtLocal._IUUIDPrimaryKeyImpl UUIDKey = new
com.ibm.IBOIMExtLocal._IUUIDPrimaryKeyImpl();
UUIDKey.generate();
static private org.omg.CORBA.Object obj=null;
obj = theHome.createFromPrimaryKeyString(UUIDKey.getUuid());
x = IFCBSecurityInfoAuthzHelper.narrow(obj);
IFCBSecurityInfoStructObjectHelper usi = new IFCBSecurityInfoStructObjectHelper();
UserSessionInfoStructHolder h = new UserSessionInfoStructHolder();
h.value = usi.sessionInfoStructData.userSessionInfoData;
booleanResult = x.gcssafAuthenticateUser("cn= ILS -BASE1-
PDC.TRIGMON,ou=USAF,ou=PKI,ou=DoD,o=U.S. Government,c=us",
"AppsPassword", h );
```

Figure 45: IFCBSecurityInfoStructFigure 8: gcssafAuthenticateUser Code Example

The process for planning for access control for application identities is the same as for end users. See 6.3.1 Planning for Access Control to see the steps involved.

Refer to 6.2 Authentication for recommendations on standards and conventions for application *userIDs*.

5.3.2.2.6 Access Control

The primary responsibility from a security perspective of the MA is to create method level authorizations using the Policy Director² infrastructure.

In order to provide method level authorizations, the MA needs to perform the steps below that are defined in Section 6.3.1 Planning for Access Control:

1. Determine Resources Requiring Protection
2. Derive Group Data
3. Derive User Data
4. Determine Level and Type of Protection
5. Set Up Data in Policy Director
6. Develop Access Control Checks in MA Software
7. Coordinate Fielding in an Operation Environment

² A background in the Policy Director product is essential for the planning phases of these tasks and for inputs specifically into the product. Sufficient details are provided in this section for MA developers to write access control checks for EJB and CORBA Server Components.

This section will attempt to identify specifics related to EJB and CORBA Server Components in this process. When there is no distinction, the reader will be referred to Section 6 Securing the Application

Step1 - Determine Resources Requiring Protection

The methods of the CORBA and EJB Server Components are the potential objects requiring protection. During the design phase for **Step - 1**, it is sufficient for planning purposes merely to define the methods within the module, interface, and method hierarchy. **Step - 4** will require the actual IDL that was generated. For CORBA Server Components this may very well be the same, but the EJB Server Components methods get wrapped within WebSphere. The generated IDL uses the wrapped name of the method.

It is not necessary to perform access checks on every method.

Step 2 - 2. Derive Group Data

Please refer to 6.3.1 Planning for Access Control. There is nothing unique about EJB or CORBA Server Components for identifying user roles.

Step 3 - Derive User Data

Please refer to Section 6.3.1 Planning for Access Control. There is nothing unique about EJB or CORBA Server Components for identifying the rules by which users are added to each role.

Step 4 - Determine Level and Type of Protection

6.3.1 Planning for Access Control provides guidance and philosophy on how and where to place Access Control Lists. This section will cover the output of this step as it pertains to EJB and CORBA server Components.

Three steps were identified for this process:

- A. Derive the hierarchy of objects and functions.
- B. Determine the business rules to apply in deciding access to each item in the hierarchy.
- C. Derive the ACLs corresponding to each rule or set of rules at each point in the hierarchy.

Step A - Derive the hierarchy of objects and functions

Policy Director Object Space for Applications -The **Object Namespace** is used by the MA when making explicit access control checks using the **aznAPI**. The MA needs to coordinate with the GCSS-AF Enterprise Authority and its Operations and Support arm to identify where the MAs application falls in the Object Space.

The IF has standardized on **/GCSS-AFAPPS** as the application root for all MAs. The GCSS-AF Enterprise Authority and its Operations and Support arm are responsible for defining and maintaining the standards and conventions for the rest of the object namespace. One recommendation is to use the convention detailed in Table 21: IF Policy Director Application Object Namespace (Recommendation). The recommended standard is to use uppercase, so as not to have to remember mixed case rules, for the upper levels of the object space. When defining the object space for modules, interfaces, and methods, the industry standard conventions for these should be used.

Table 21: IF Policy Director Application Object Namespace (Recommendation)

Level	Description
/	Policy Director root object.
/GCSS-AFAPPS	Application root for all GCSS-AF MAs. This level is fixed.
/GCSS-AFAPPS/<domain>	<domain> is the functional domain that is responsible for providing services. E.g. ILS. It is recommended that each domain have its own namespace file on the Master Security Management Server. ³ This recommendation helps an administrator manage the file as it grows and allows different administrators to support different domains at the concurrently.
/GCSS-AFAPPS/<domain>/<location>	<location> is the base via a georef code or other standard, MAJCOM, enterprise, etc that the applications defined below it support. The GCSS-AF Enterprise Authority should tightly control the list of locations. The significance of location is based on the data. If the backend data is retrieved from a single database for all bases, then a location of enterprise may be appropriate. If a different database and therefore application instance is required for each base, then a location that identifies the base is appropriate.
/GCSS-AFAPPS/<domain>/<location>/<application>	<application> is an arbitrary name that uniquely identifies an application

After the application, the CORBA/EJB Server Component's module, interface, and then method follow it in the namespace. The MA should use the generated IDL to extract the modules, interfaces, and methods. EJB developers should particularly take note as the EJB methods are wrapped when deployed in Component Broker. The method will be prepended with **passthru_**. The safest mechanism, however, is to verify what gets generated in the IDL. It is necessary for the MA to use the module, interface, and method in addition to the front piece of the application object namespace shown in Table 21: IF Policy Director Application Object Namespace (Recommendation) when making access control checks. An example of a Policy Director Namespace for a CORBA Server Object is shown in Figure 46: Policy Director Object Space for PDC Test Component. An example of a Policy Director Namespace for an EJB Server Component is shown in

³ Refer to the Tivoli SecureWay Policy Director Administration Guide for details on adding 3rd party namespaces.

Figure 47: Policy Director Object Space for PDC Test Component (Part A) and Figure 48: Policy Director Object Space for PDC Test Component (Part B).

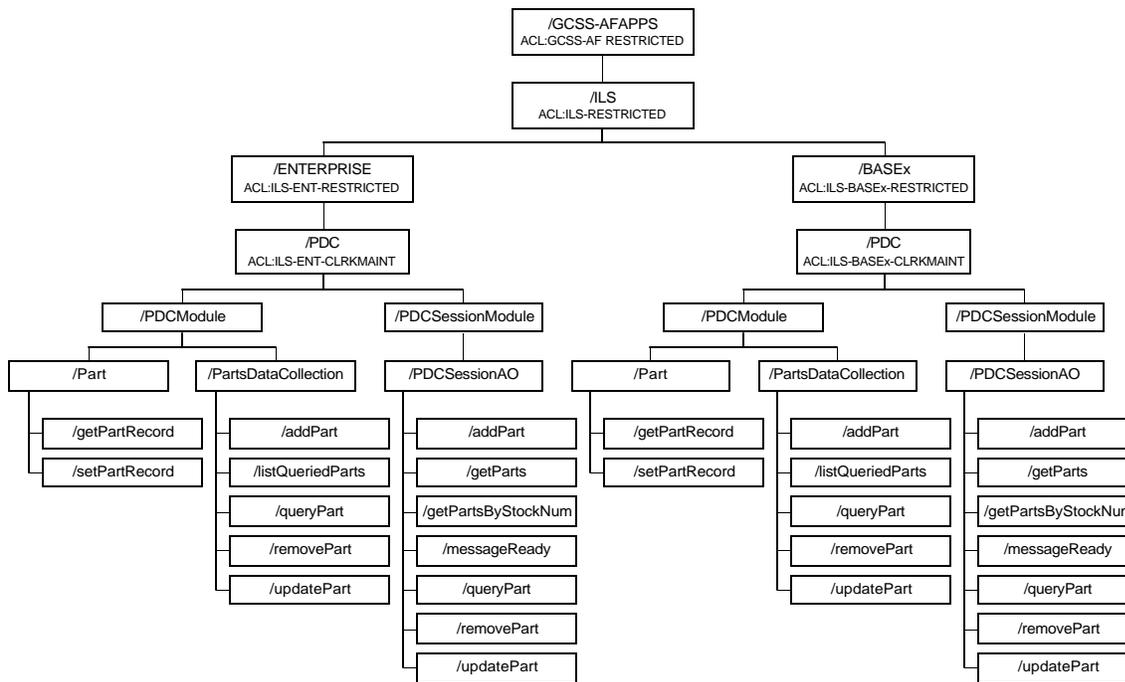


Figure 46: Policy Director Object Space for PDC Test Component

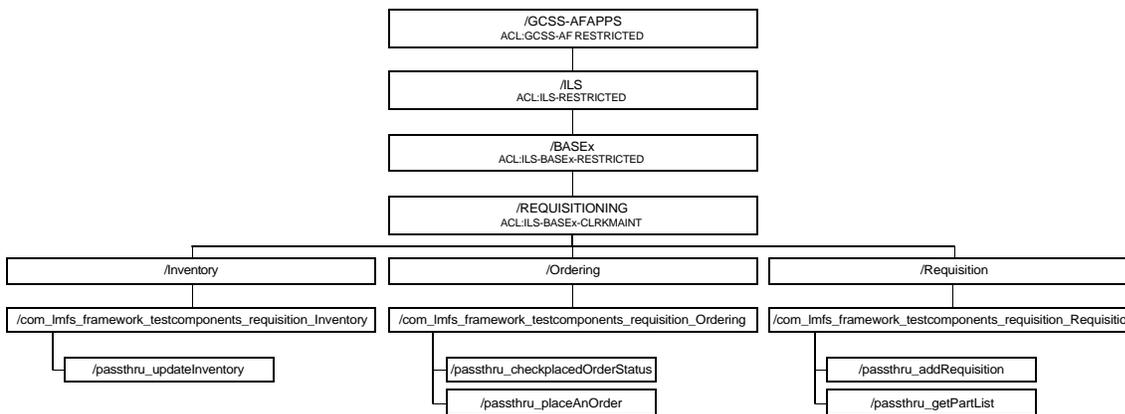


Figure 47: Policy Director Object Space for PDC Test Component (Part A)

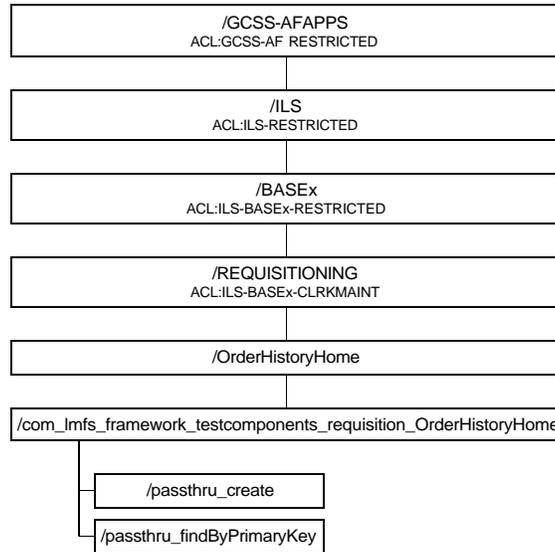


Figure 48: Policy Director Object Space for PDC Test Component (Part B)

Step B - Determine the business rules to apply in deciding access to each item in the hierarchy.

Step C - Derive the ACLs corresponding to each rule or set of rules at each point in the hierarchy.

Steps B and C are performed simultaneously. The natural mechanism for defining ACLs is for the MA to walk the namespace and determine who (which user role/group) needs access to what and the type of access required by that role. Refer to Table 22: Example Policy Director Access Control Lists (ACL) for a list of ACLs that were defined for the PDC Test Component.

Each rule should be set as high in the hierarchy as possible. For example, if there is only one business rule governing who can read any item the MA must protect, then apply this rule to the root level of the hierarchy, not to each item individually. Figure 46: Policy Director Object Space for PDC Test Component delineates assignments of ACLs. Methods without an ACL explicitly defined traverse up the tree and use the next defined ACL in the namespace.

Table 22: Example Policy Director Access Control Lists (ACL)

ACL Name	Type (user/group) & Name	Permissions ⁴
GCSS-AF Restricted	Group GCSS-AF-Admins	Full Access
	Authenticated	Traverse
	Unauthenticated	No Access
ILS-ENT-RESTRICTED	Group GCSS-AF-Admins	Full Access
	Group ILS -Admins	Full Access
	Group ILS -ENT-Admins	Full Access
	Authenticated	Traverse
	Unauthenticated	No Access
ILS-ENT-CLRKMANT	Group GCSS-AF-Admins	Full Access
	Group ILS -Admins	Full Access
	Group ILS -Supply Clerk	Traverse, Create, Read, Update, Delete, Execute
	Group ILS -Maintenance	Traverse, Create, Read, Update, Delete, Execute
	Authenticated	Traverse
	Unauthenticated	No Access
ILS-BASEx-RESTRICTED	Group GCSS-AF-Admins	Full Access
	Group ILS -BASEx-Admins	Full Access
	Group ILS -BASEx-Users	Read
	Authenticated	Traverse
	Unauthenticated	No Access
ILS-BASEx-CLRKMANT	Group GCSS-AF-Admins	Full Access
	Group ILS -Admins	Full Access
	Group ILS -Supply Clerk	Traverse, Create, Read, Update, Delete, Execute
	Group ILS -Maintenance	Traverse, Create, Read, Update, Delete, Execute
	Authenticated	Traverse
	Unauthenticated	No Access

Step 5 - Set Up Data in Policy Director

This activity is the responsibility of the Operations and Support Organization with input from the steps above from the MA. Refer to 6.3 Access Control for additional information on setting up data in Policy Director.

As indicated above, the EJB and CORBA Server Components use a 3rd Party Namespace in Policy Director. It is necessary to update the appropriate ASCII text files and policies in Policy Director when EJB and CORBA Server Components are added, updated, or deleted to maintain this 3rd Party Namespace.⁵ The recommendation is to, at a minimum, maintain a separate namespace file for each domain for ease of administration and file locking.

Step 6 - Develop Access Control Checks in MA Software

The tasks required to perform this Step:

⁴ Table 31: Pseudo-Supply Test ACLs in 6.3.1 Planning for Access Control uses the actual permission labels used by Policy Director. They are written here to provide a higher level understanding for a developer and not for actual implementation in Policy Director. Please refer to the Tivoli SecureWay Policy Director Administration Guide and 6.3.2 Setting up ACLs for the actual format of the permissions.

⁵ Please refer to the Tivoli SecureWay Policy Director Administration Guide for instructions on updating 3rd Party Namespaces.

- A. Accept the **sessionInfoStruct** type as a parameter to the method
- B. Build the parameters required by the *gcssafAccessDecisionAllowed* method
- C. Find/Create a **CBSecurityInfo** model instance (scope host-scope-widened)
- D. Invoke the *gcssafAccessDecisionAllowed* method
- E. Turning security on in SMUI

Step A -Accept the sessionInfoStruct type as a parameter to the method

Each method that needs to be protected with access control checks shall accept a **sessionInfoStruct** structure as a parameter. If the method does not directly make access control checks, but invokes other methods that do require access control checks, then the method would still need to accept a **sessionInfoStruct** structure to be able to pass it to the invoked methods. The sections below describe how to perform this for CORBA and EJB Server Components.

CORBA Server Components -See Figure 49: Example IDL from PDCSessionModule of the PDC Test Component provides an example IDL for a CORBA Server Component. CORBA Server Components need to create a dependency on the **IFCBSecurityInfo** model in the Component Broker Object Builder tool when any of the methods use the **sessionInfoStruct** type. This dependency is reflected in the IDL by having the **IFCBSecurityInfoFile.idl** include file.

```
...
#include <IFCBSecurityInfoFile.idl>
...
module PDCSessionModule {
  interface PDCSessionAO;

  interface PDCSessionAO : IManagedClient::IManageable
  {
    ...
    ...
    void addPart(in long pdcID,in PDCHelperModule::PartRecord pRecord,in
IFCBSecurityInfo::SessionInfoStruct sessInfoStruct ) raises
(IManagedClient::IDuplicateKey,PDCHelperModule::PDCException);
    ...
  }; // end interface PDCSessionAO
}; // end of module PDCSessionModule
```

Figure 49: Example IDL from PDCSessionModule of the PDC Test Component⁶

EJB Server Components -See Figure 50: Example IDL from com_lmfs_framework_testcomponents_requisition_OrderingTie.java of the Requisition Component Test Component provides an example IDL for an EJB

⁶ Snippet taken from PDCSessionClasses.idl

Server Component. EJB Server Components need to have the **IFSServices.jar** file in their classpath in order to use the **sessionInfoStruct** type.

```

...
public interface com_lmfs_framework_testcomponents_requisition_OrderingTie extends java.rmi.Remote
{
...
...
// Bean-specific business methods
    public java.lang.String placeAnOrder_object_ (IFCBSecurityInfo.SessionInfoStruct arg0,
    java.lang.String arg1, java.lang.String arg2, java.lang.String arg3, int arg4, java.lang.String arg5) throws
    java.rmi.RemoteException, java.lang.Exception;
...
}
    
```

Figure 50: Example IDL from com_lmfs_framework_testcomponents_requisition_OrderingTie.java of the Requisition Component Test Component

Step B -Build the parameters required by the gcssafAccessDecisionAllowed method

To build the parameters required by the Access Control method, the MA Developer has to use information available from the code itself and information obtained and coordinated with the Operations and Support Organization.

The parameters of the *gcssafAccessDecisionAllowed* method are:

Table 23: Parameters of the GCSSAF/AccessDecisionAllowed Method

Parameter	How Obtained
UserSessionInfoData	Obtained from sessionInfoStruct parameter. This structure is created and passed in from the invoking application (Servlet, JSP, other CORBA or EJB Server Component) as part of the sessionInfoStruct parameter.
SecLabelString	The Policy Director Object Space (relative or fully qualified context). If this field is formatted as a relative context, then it is pre-pended with the AppPdLabel field to define the Policy Director Object space to check the permission against. Additional information for determining the contents of this field is provided below
Permission	Action that the method is trying to perform. Refer to Section 6.3.2 Setting up ACLs for a definition of the permissions.
AppPdLabel	Initial context. Built with information from the sessionInfoStruct parameter and MA specified information. Additional information for determining the contents of this field is provided below.
CacheTimeout	Should be set to 0 for now. The design was anticipating a caching mechanism in the future.

UserSessionInfoData -The **UserSessionInfoData** parameter is contained in the **sessionInfoStruct** that is passed as a parameter to applications that require security access control checks. The data for the **userSessionInfoData** piece of

the **sessionInfoStruct** is created by Policy Director as a part of the authentication process. Servlets and

SecLabelString -The purpose of the **SecLabelString** and the **AppPdLabel** in the *gcssafAccessDecisionAllowed* method is to identify the object as it is represented in Policy Director.

To create **SecLabelString**, the MA Developer can determine the names of the module, interface, and method of the method they are trying to protect. In the code example, the format that the Access Control method is expecting this information is as follows: **String secLabelStr =**
"[/PDCSessionModule/PDCSessionAO/addPart/]" ; Because it was pre-pended with a “.” it is a relative Policy Director Namespace Context and the **AppPdLabel** parameter will be used to fully qualify it.

AppPdLabel -To create the **AppPdLabel**, the MA Developer obtains certain information from the **sessionInfoStruct** parameter. The **sessionInfoStruct** parameter contains the **baseNameQualifier** and **baseName** context. The **baseNameQualifier** parameter identifies the initial levels of the Policy Director Object Namespace where this application resides. It is composed of the fixed **/GCSS-AFAPPS** root context for GCSS-AF applications and the functional domain. For our pseudo-supply test application, the **baseNameQualifier** is **/GCSS-AFAPPS/ILS**.

The **baseNameCtx** is the location that the data represents. Refer to Table 21: IF Policy Director Application Object Namespace (Recommendation) for additional information. Examples from the IF Test Components for this field are; ENTERPRISE, BASE1, BASE2, and BASE3.

It is incumbent on the invoking app to determine this information and supply appended to the end of the **AppPdLabel** is the Application Identity. As identified in **Step 4 - Determine Level and Type of Protection** of this process, this label is created by the MA and coordinated with the Operations and Support Organization.

The design of the application may determine an alternate approach of obtaining the **AppPdLabel** parameter. The recommendation is to limit the amount of hard coding, except for static content.

```
public void addPart( int pdcID, PDCHelperModule.PartRecord pRecord,  
IFCBSecurityInfo.SessionInfoStruct sessInfoStruct)  
throws com.ibm.IManagedClient.IDuplicateKey,  
PDCHelperModule.PDCException  
{  
    ...  
    // local variables
```

```
// security Label String
String secLabelStr = ["/PDCSessionModule/PDCSessionAO/addPart/"];
// appPDLabel String
String appPDLabelStr = sessInfoStruct.appSessionInfoData.baseNameQualifier +
"/" + sessInfoStruct.appSessionInfoData.baseNameCtx +
"/PDC" ;
...
}
```

Figure 51: Example of Obtaining Parameters Required by the gcssAccessDecisionAllowed Method

Step C -Find/Create an IFCBSecurityInfo model instance (scope server-scope-widened)

Figure 52: Code Example of Finding IFCBSecInfo Home and Figure 53: Code Example of Creating IFCBSecurityInfo Instance provide an example of the code necessary to create and locate an **IFCBSecurityInfo** model instance.

```
private com.ibm.IManagedClient.IHome getIFCBSecInfoHome()
{
/**
 * This method will locate the Home for the IFCBSecInfo class
 * It will use a server-scope-widened server based on the current
 * serverName to locate that home.
 */
...
// local variables
// temp CORBA Object
org.omg.CORBA.Object obj = null ;

try {
if (iIFCBSecInfoHome == null ) {
// getSSWFactFinder() returns a name service with the following parameters
// "host/resources/factory-finders/" +
// CBSeriesGlobal.serverName() + "-server-scope-widened
obj = getSSWFactFinder().
find_factory_from_string("IFCBSecurityInfo::IFCBSecurityInfoAuthz.object interface") ;
...
iIFCBSecInfoHome = com.ibm.IManagedClient.IHomeHelper.narrow(obj) ;
...
} // end if
}
catch (Exception exc) {
logCat.error("PDCSessionAO-" + location() + "::getIFCBSecInfoHome::Exception caught: ", exc) ;
}
...
return iIFCBSecInfoHome ;
...
}
```

Figure 52: Code Example of Finding IFCBSecInfo Home

```
private IFCBSecurityInfo.IFCBSecurityInfoAuthz createIFCBSecInfoAuthz()
{
    ...
    /**
     * This method creates the IFCBSecurityInfoAuthz object using the home
     * @return IFCBSecurityInfo.IFCBSecurityInfoAuthz
     */
    if ( iFCbSecInfAthz == null ) {
        ...
        try {
            // create and generate the uuid key
            com.ibm.IBOIMExtLocal._IUUIDPrimaryKeyImpl iFCbSecKey = new
com.ibm.IBOIMExtLocal._IUUIDPrimaryKeyImpl() ;
            iFCbSecKey.generate() ;

            org.omg.CORBA.Object obj = getIFCBSecInfoHome().createFromPrimaryKeyString(
iFCbSecKey.getUuid() ) ;
            iFCbSecInfAthz = IFCBSecurityInfo.IFCBSecurityInfoAuthzHelper.narrow( obj ) ;
        }
        catch (Exception exc) {
            logCat.error("PDCSessionAO-" + location() + "::createIFCBSecInfoAuthz::Exception caught: ", exc) ;
        }
        ...
    } // end if

    return iFCbSecInfAthz ;
    ...
}
```

Figure 53: Code Example of Creating IFCBSecurityInfo Instance

Step D -Invoke the gcsafAccessDecisionAllowed method

Refer to Figure 54: Code Example of Invoking the gcsafAccessDecisionAllowed Method as an example for invoking the **gcsafAccessDecisionAllowed** method.

```
public void addPart( int pdcID, PDCHelperModule.PartRecord pRecord,
IFCBSecurityInfo.SessionInfoStruct sessInfoStruct)
    throws com.ibm.IManagedClient.IDuplicateKey,
    PDCHelperModule.PDCException
{
    ...
    // check if user is allowed to access the method
    boolean accessAllowed = createIFCBSecInfoAuthz().gcssafDecisionAccessAllowed
    ( sessInfoStruct.userSessionInfoData,
    secLabelStr,
    "K",
    appPDLLabelStr,
    0 );
    ...
    if ( accessAllowed ) {
        try {
            // Perform the action
            ...
        }
        catch (com.ibm.IManagedClient.INoObjectWKey nowk) {
            // used as an example catch for the code snippet, other exceptions are handled as well
            ...
        }
    }
    // end if accessAllowed
    else {
        // User not authorized. Perform error handling
        logCat.warn("PDCSessionAO-" + location() + "::addPart::ACCESS DENIED!!" );
        throw new PDCHelperModule.PDCException( "PDCSessionAO-" + location() + "::addPart::ACCESS
DENIED!!" );
    }
    ...
}
```

Figure 54: Code Example of Invoking the gcsafAccessDecisionAllowed Method

Step E -Turning security on in SMUI

The MA needs to establish a secure environment when fielding their application in WebSphere. The list assumes that the WAS EE server was installed and configured with security turned on and specifically on the name server application.⁷ The MA should refer to the IF Software Installation Procedures document for current recommendations on configuring the MA when fielding the application in the WebSphere environment.

⁷ Refer to the IF Software Installation Procedures document for installation and configuration of the WAS EE server.

Step F -Coordinate Fielding in an Operational Environment

From a security perspective, at a minimum the MA needs to supply to the Operations and Support Organization the following information that was generated via the previous steps:

1. Group Names and Membership Rules
2. All of the IDL generated for the application (Relates to Object Space)
3. The functional domain that this application resides in
4. The locations (a.k.a. **baseName** context field) that the data represented by the application will support. This is a joint activity with the Operations and Support Organization.
5. ACLS (Roles and Permissions)
6. ACL allocation to the PD Object Namespace

Additional information will be required from the MA based on the Operations and Support Procedures.

5.3.2.2.7 Audit and Alarms

The primary responsibility from a security perspective of the MA is to create method level authorizations using the Policy Director⁸ infrastructure.

Refer to 6.7 Audit and Alarms for an overview of the audit and alarm mechanisms provided by the IF.

In general, applications should not need to log security related events separately from the IF. However, in the event that MA requirements analysis indicates the need to log (for security purposes) some event that is not already logged by the IF security services, the IF provides a mechanism for doing so. Section 6.7.1 Logging Framework Security Events indicates how the logging service can be initialized. The logging process is explained in Section 6 Securing the Application.

(Note that **log4j** is primarily intended to log non-security events for debug or other purposes.)

5.3.3 Communications Between Layers

5.3.3.1 Vertical

An application may want to communicate with or use services in the Presentation or Data later. This section will describe any special steps the developer needs to take in order to do this.

⁸ A background in the Policy Director product is essential for the planning phases of these tasks and for inputs specifically into the product. Sufficient details are provided in this section for MA developers to write access control checks for EJB and CORBA Server Components.

5.3.3.1.1 Presentation

No services or recommendations are provided for communicating to this layer.

5.3.3.1.2 Data layer

If an application wants to use ORACLE services from inside a Component Broker application, it will be necessary to use the ORACLE Application Adapter, which functions as an interface between CB and ORACLE. Use of the ORACLE application adapter is described in the online help that accompanies it.

Similarly, an Application Adapter is provided for the DB2 database. If an application wants to use DB2 services from inside a Component Broker application, it will be necessary to use the DB2 Application Adapter, which functions as an interface between CB and ORACLE. Use of the DB2 application adapter is described in the online help that accompanies it.

It is also possible for applications developers to write their own application adapters to link CB components and the back-end of their choice. This path requires extensive knowledge of Component Broker internals and those of the back-end chosen and is at least very difficult. This option is not recommended nor for the faint of heart.

5.3.3.2 Horizontal

The approach taken by the Integration Framework is to have Business Service Components communicate using the Business Object Document mechanism specified by the Open Applications Group. It is intended that applications or application components will pass BODs between each other in order to communicate desired requests using MQSeries.

Refer to the Open Applications Group Integration Specification (OAGIS) for a complete description of the Business Service Request and the Business Object Document that the OAG have defined. That specification can be obtained from the Open Applications Group web site at www.openapplicationsgroup.org.

Messaging services are provided between applications by MQSeries. Component Broker application components interface to MQSeries using the MQSeries Application Adapter (MQAA).

The MQSeries application adapter provided by Component Broker is used primarily to provide a semi-transparent integration between Component Broker business applications and non-Component Broker applications that are based directly on MQSeries. The MQSeries application adapter supports communication between Component Broker servers on Windows NT and Solaris and an MQSeries queue manager on the same local host.

The use of the Component Broker MQSeries Application Adapter is shown in the following figure:

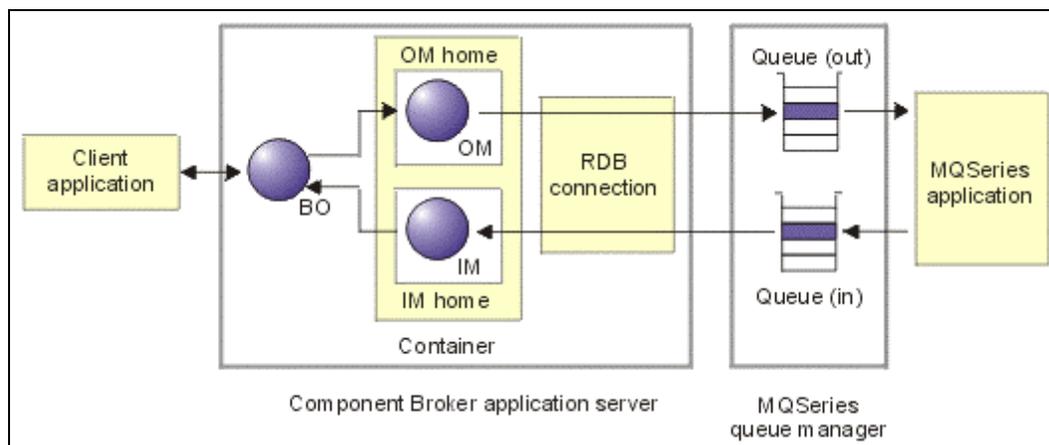


Figure 55: Use of the MQSeries application adaptor by a Component Broker application

From a client perspective, an MQSeries-backed application looks like any other Component Broker-based client application that uses Component Broker transaction services.

Note: The MQSeries application adapter currently only supports a transactions container policy; to throw an exception and abandon the call when used outside the scope of a transaction. (Atomic transaction method calls are not supported.)

The managed objects representing messages behave like any other managed object that uses the transaction services. Because such an object represents a message in a queue manager, its life cycle is controlled by the standard messaging data access operations, insert, retrieve, update, and delete (IRUD). Component Broker, and its MQSeries application adapter framework, drive these OM and IM object instances and call the IRUD methods at appropriate times. For example, if a client application is trying to get an inbound message that is not currently instanced in the application server, it creates a new IM object and issues the *retrieve* method on it to get the message from its queue. If the client commits the transaction, the message is removed from its queue.

For more information about how the Component Broker programming model uses the MQSeries application adapter, see the “The Component Broker message processing model” topic in the Component Broker documentation.

The Integration Framework provides extensions to the out-of-the-box IBM inbound and outbound message components. The extensions are called **TMOutbound** and **TMinbound** (components in the **TextMessage** application) and allow the application using them to decide at runtime which queue to use for putting or getting a message. They also provide access to many attributes of the incoming and outgoing message objects such as priority, expiry, feedback and encoding. The extension of the specialized home for incoming message objects is called **IExtendedMessage**. The extension of the incoming message object is called **TMinbound**, and

the extension for outbound message objects is called **TMOutbound**. The extensions are used in a manner comparable to those provided by IBM. To use the **TMOutbound** object, a copy helper is instantiated and its attributes set to the values desired of the outgoing message. The copy helper is then passed to the *put()* method of the outbound message object home.

Here is an example of sending a message using the **TMOutbound** message object:

```
// set the queueName for GetListItem BOD from Base PDCs to Enterprise PDC
// set to PDC. ENTERPRISE.GETITEM.INBOUND
String destQueue = "PDC." + nameStr + ".GETLISTITEM" + ".INBOUND" ;
// copy for TMOutbound
TextMessageCopy.TMOutboundCopy outCopy =
TextMessageCopy.TMOutboundCopyHelper._create() ;

// set the destination queueName
outCopy.queueName(destQueue) ;

// set replyToQueue name for ListItem BOD from Enterprise PDC to Base PDCs
// set to PDC.<loc>.LISTITEM.REPLY.INBOUND where loc = BASE1, BASE2, or
BASE3
String replyToQueue = "PDC." + nameContext + ".LISTITEM" + ".REPLY" +
".INBOUND" ;

// set ReplyToQ name
outCopy.replyToQ(replyToQueue) ;

// set correlator to an empty string
outCopy.correlator("");

// set reply message as the message data
outCopy.messageData( getLstltnStr.getBytes() ) ;

// convert the copy helper to a byte array
theCopyOrKeyStr = outCopy._toString() ;

// start a transaction

// obtain access to a transaction control object.
obj = CBSeriesGlobal.orb().resolve_initial_references("TransactionCurrent") ;
currentTransaction = org.omg.CosTransactions.CurrentHelper.narrow(obj) ;

currentTransaction.set_timeout(180) ;

// begin transaction
currentTransaction.begin() ;

// put the message to the output queue - returns the correlator
String outboundCorrel = getOutboundMsgHome().put(theCopyOrKeyStr) ;

// commit the transaction      currentTransaction.commit(true) ;
```

Figure 56: Sending a message using TMOutbound

The code for **getOutboundMessageHome()** in the previous code sample is found in the following code sample. This code follows the standard recipe for locating a home. See the PDC source code for the definition of the method *getSSWFactFinder()* used in the following code sample.

```
private com.ibm.IMessageHome.OutboundMessageQueue getOutboundMsgHome()
{
/**
 * This method will locate the Home for the TMOutbound class
 * It will use a server-scope-widened server based on the current
 * serverName to locate that home.
 * @return com.ibm.IMessageHome.OutboundMessageQueue
 */

// local variables
// temp CORBA object
org.omg.CORBA.Object obj = null ;

try {
    if (iOutboundMsgHome == null ) {
        obj = getSSWFactFinder().
            find_factory_from_string("TextMessage::TMOutbound.object interface") ;

        iOutboundMsgHome =
com.ibm.IMessageHome.OutboundMessageQueueHelper.narrow(obj) ;

    }
}
catch (Exception exc) {
}

return iOutboundMsgHome ;
}
```

Figure 57: Getting the Outbound Message home

To use the **TMinbound** message object, a key helper is created and initialized with data describing the message to be retrieved, such as what queue to look on and what message correlator to look for, if any. The key helper is stringified and passed to the *getUsingKeyString()* method on the inbound message home. The code for locating the inbound message home, **getInboundMessageHome()**, can be found in the **PDCSession** application test component business object provided with the Integration Framework.

Here is an example of how to receive a response message that is correlated to the message sent in the preceding code snippet, all using the **TMinbound** message object. Additionally this code waits up to a maximum time limit for the response message and throws an exception if no message arrives in the allotted time.

```
// wait for the response – ListItemBOD
// key for TMIInbound
TextMessageKey.TMIInboundKey inKey =
TextMessageKey.TMIInboundKeyHelper._create() ;

// set queue name to access for message
inKey.queueName(replyToQueue) ;

// ensure correlator key is set from the outbound message
inKey.correlatorKey(outboundCorrel) ;

// get the Key in byte[]
theCopyOrKeyStr = inKey._toString() ;

// get the ExtendedInboundMessageQueue
IextendedMessagingModule.ExtendedInboundMessageQueue inHome =
IextendedMessagingModule.ExtendedInboundMessageQueueHelper.narrow(getInbo
undMsgHome()) ;

// Start while loop
// If message is received, break out of the while loop
final int maxWait = 120000 ; // upper limit for cumulative wait time
final int waitIncr = 500 ; // wait increment
int cumulativeWait = 0 ; // track wait time so far

while (true) {

// wait and check for time limit
try { Thread.sleep(waitIncr) ; } catch ( InterruptedException ie ) { } ;
cumulativeWait += waitIncr ;
if ( cumulativeWait > maxWait ) {
throw (new com.ibm.IMessageHome.ImessageNotFound()) ;
}

try {

// begin transaction
currentTransaction.begin();

// get the message
obj = inHome.getUsingKeyString(theCopyOrKeyStr) ;
inMsg = TextMessage.TMIInboundHelper.narrow(obj) ;

// convert message byte array to a string
msgString = new String(inMsg.messageData()) ;

// commit the transaction
currentTransaction.commit(true) ;

} // end try

catch (com.ibm.IMessageHome.ImessageNotFound nm) {
// No message in the queue
currentTransaction.commit(true) ;
logCat.warn("PDCSessionAO-" + location() + "::getEnterprisePartsList::No more
```

```
messages to process" );  
    } // end catch lmessageNotFound  
    catch (Exception exc) {  
        // commit the transaction  
        currentTransaction.commit(true) ;  
    } // end catch Exception  
  
} // end while
```

Figure 58: Receiving a correlated message using TMOutbound

When the **TMMMessage** application is deployed as part of another application, it will be necessary to configure in SMUI two containers relating to MQSeries for the application. Using the **PDCSession** component as an example, the containers are located in SMUI under:

Management Zones -> PDCSession Zone -> Configurations -> PDC Session Configuration -> RDB Connections.

The containers are called **MQInboundContainer** and **MQOutboundContainer** and correspond to the **TMInbound** and **TMOutbound** objects respectively. To configure the containers for the application, right-click on the container name and select Properties. Then for the database name and open string, fill-in the name of the queue manager that the **TMInbound** and **TMOutbound** objects will use in the application.

Note: The application component is restricted to using just one queue manager for incoming messages and one for outgoing messages (they can be the same queue manager).

Also, when the **TMMMessage** application component is deployed as part of another application such as the **PDCSession** component, it will be necessary include two additional application components, **IMQAAServices** and **iDXAAAServices**, when configuring the server for the application in SMUI.

5.3.3.2.1 Message Listener

The message listener is a program that monitors an initiation queue for messages and takes action when a message arrives. The action taken could be to send another message somewhere, to execute a program, to invoke a method on a CORBA object, or all of these and more. Any action that can be programmed is possible. Generally, a message listener is associated with an application; that is, they are usually application specific. The trigger monitor provided with the IF application test components is associated with the PDC component. When a message arrives on the initiation queue monitored by the trigger monitor, the monitor invokes the *messageReady()* method on the **PDCSession** object, which is part of the PDC application.

To use a trigger monitor, one defines the initiation queue to be monitored and configures MQSeries to put a trigger message into the initiation queue when conditions the developer cares about are satisfied, e.g., when a message arrives on one of several queues and needs to be processed. Once the initiation queue is defined, any other applications that the trigger monitor communicates with need to be started, and finally the trigger monitor itself needs to be started. Then it will wait for a trigger message, and take the defined action(s) when one appears on the initiation queue.

The trigger monitor provided as part of the Integration Framework test components can be used as a template or starting point for writing a trigger monitor. IBM also provides a sample trigger monitor in the MQSeries distribution, which will be extendable for the users needs. Since MQSeries can be configured so that multiple queues are triggered and send a trigger message to a given initiation queue, a single trigger monitor application can “handle” trigger messages for multiple input queues.

In addition to monitoring an initiation queue for trigger messages, the trigger monitor supplied as part of the Integration Framework also is responsible for registering publisher and subscriber Component Broker application components with the MQSeries message broker. To do this, the trigger monitor reads an initialization file that contains instructions on which applications should be registered as *publishers* and/or *subscribers*. The registration occurs only at startup of the trigger monitor; then the trigger monitor falls into a loop that monitors the initiation queue for trigger messages.

The C++ trigger monitor provided as part of the Integration Framework test components uses the pub/sub helper routines defined in the header file **pubsub.h** provided as part of the Integration Framework. These routines simplify the generation of the specially formatted control messages to be sent to the message broker to register and deregister publishers and subscribers.

The C++ trigger monitor provided as part of the Integration Framework test components uses an initialization file to determine what registrations or de-registrations of publishers or subscribers to make and where to send them. A sample initialization file is show below. Lines beginning with ‘#’ are comments and ignored by the trigger monitor. The file is divided into stanzas by tokens enclosed in square brackets. The tokens provide the action that the trigger monitor is to perform and the subsequent six lines that are not a comment provide information about the topic to register and the queue and queue manager to register and where to send the control message.

```
#-----  
[registerPublisher]  
# EXCEPTIONS. If the next non-comment line matches the location code we are given at startup, skip  
this stanza.  
  
# TOPIC. The next non-comment line is the topic to register.  
SYNCINVENTORY  
# STREAM. The next non-comment line is the stream to register. <location> is substituted with the  
location code  
# we are given at startup.  
IF.<location>.DEFAULT.STREAM  
# QMGR. The next non-comment line is the queue manager to register.  
  
# QUEUE. The next non-comment line is the queue to register  
  
# BROKER QUEUE. The next non-comment line is the queue to which we put the formatted control  
message.  
SYSTEM.BROKER.CONTROL.QUEUE  
# BROKER QMGR. The next non-comment line is the queue manager to which we put the formatted  
control message.  
QMR1C1  
#-----  
[registerSubscriber]  
# EXCEPTIONS.  
ENTERPRISE  
# TOPIC  
SYNCINVENTORY  
# STREAM  
IF.ENTERPRISE.DEFAULT.STREAM  
# QMGR  
QMR1A1  
# QUEUE. <location> is substituted with the location code we are given at startup.  
PDC.<location>.SYNCINVENTORY.RECEIVER  
# BROKER QUEUE  
SYSTEM.BROKER.CONTROL.QUEUE  
# BROKER QMGR  
QMR1C1  
#-----
```

Figure 59: Example of Trigger Monitor Application Code

5.3.4 Design Issues Concerning Deployment

5.3.4.1 Application Design to Support Workload Management

Applications expecting high-volume use should consider using workload management. Workload management (WLM) is the discipline of defining, monitoring and actively managing work in the distributed network. In Component Broker, work is the dispatch, routing, and receipt of requests between objects in the distributed network and their eventual execution within a Component Broker application server. When more clients use an application, the amount of work increases, and the load on the servers increases.

Clients normally locate resources at a pre-configured server location, perhaps a named server or the server running on their local host. Fixing the relationship between the client and server is a problem for scalability in large enterprises. When multiple servers exist, which could potentially service a client's request, it is not desirable to force the client to go to one fixed server. The workload distribution mechanism, part of WLM, allows the ORB to dynamically allocate an application server to process a request. The goal is to minimize client request response times and maximize server throughput by reducing load imbalance. This can be achieved by locating resources at a workgroup scope and allowing the choice of an appropriate, active server to be determined by the ORB.

A home is an excellent example of a workload manageable object because it is often one of the first types of objects to be accessed remotely. Many client applications will locate a home for a required business object, perhaps using a factory finder, and then use that home to find or create that business object on the home's server.

It is just as desirable to distribute the work involved in finding, creating, and processing queries for business objects as it is for the work involved in subsequent operational requests. For objects that only need a default home implementation, the developer can select one of the supplied workload manageable homes. Specialized home can also be configured as a workload manageable home by using a workload managing container in the same way a non-workload managed container would be used.

For more information on using the workload management features of Component Broker, see the Component Broker online help for 'workload management'.

Application designers should be careful not to create "bottlenecks" in workload managed applications. Just because there is a workload-managed home does not mean that the developer has done all they can. For example, if an application uses a WLM home and non-UUID keyed objects and routes control of client requests through one of several non-UUID keyed objects based on the value of the non-UUID key, it is possible under high load to have a bottleneck in each of the non-UUID keyed objects, and it is possible that the non-UUID keyed object will force serialization of the incoming requests.

One way around the bottleneck is to define and have clients create WLM objects, which have a UUID as their primary key. In this case, when a client looks up and/or creates a WLM object with a UUID key, it is assured that the client will get its own WLM object in the application server and hence its own thread of execution rather than being serialized through one instance of a non-UUID keyed application object. So storage space for the extra UUID keyed objects is traded for throughput speed. For high performance, it is recommended that use of WLM UUID-keyed objects extend as far back toward the data store as possible so that each client has its own thread of execution in a chain of UUID keyed objects reaching nearly to the data store. This approach is at the other end of the speed vs. space tradeoff spectrum than is the use of a non-UUID keyed object as a serializing gateway to back end services.

Use of workload management in the application server may have implications for any inter-component or inter-application messaging that is being performed. Workload management clusters several application servers into one large virtual server from the point of view of the client application. In order for all instances of the server group comprising the large virtual server to have the same view of MQSeries queue configurations, MQSeries clustering of queue managers co-residing on the application servers may be required.

5.3.2.2.8 Example of workload management deployment

The following diagram illustrates a workload-managed configuration from end to end. Moving from the bottom of the diagram to the top, we will describe each layer.

Web Seal Junctions

Web Seal Junctions are set-up for Stateless (Servlet) applications; the initial request by a user for an application will be routed to the Web Server with the best response. As a stateless junction, subsequent requests by that user for that application will be routed to the Web Server with the best response. Junctions for Stateful (Servlet) applications can also be configured resulting in use of cookies to identify web server affinity. As such, once a Web Server has been established for a user of an application, subsequent requests by that user for that application will be routed to the same Web Server.

Stateless Web Seal Junctions

Using stateless Web Seal junctions to work load manage web server and Servlet processing requires that the WebSphere Servlet Engine be configured to *persist* session objects. This allows continuation of a session on a different Servlet engine.

Servlets

Servlets are multi-threaded, with each new user of a Servlet getting a new thread. Each Servlet finds an associated session application object/session bean in the appropriate server group in CBs namespace.

Application Requests

For requests to an application in a work load managed server group, requests will be to an application on one of the servers in the server group as determined by the server group work load management algorithm in effect.

Computational-intensive Business Objects

“Computational-intensive” business objects (that would significantly impact performance if the same instance were used by all clients of this object) are (transient) configured such that each new client of an application object /session bean gets a new instance.

Back End Databases

Different database instances (e.g. one supporting Base_1 and another supporting Base n,) of an application type may have separate back-end databases. However, Instances of different application types can have the same or different back-end databases

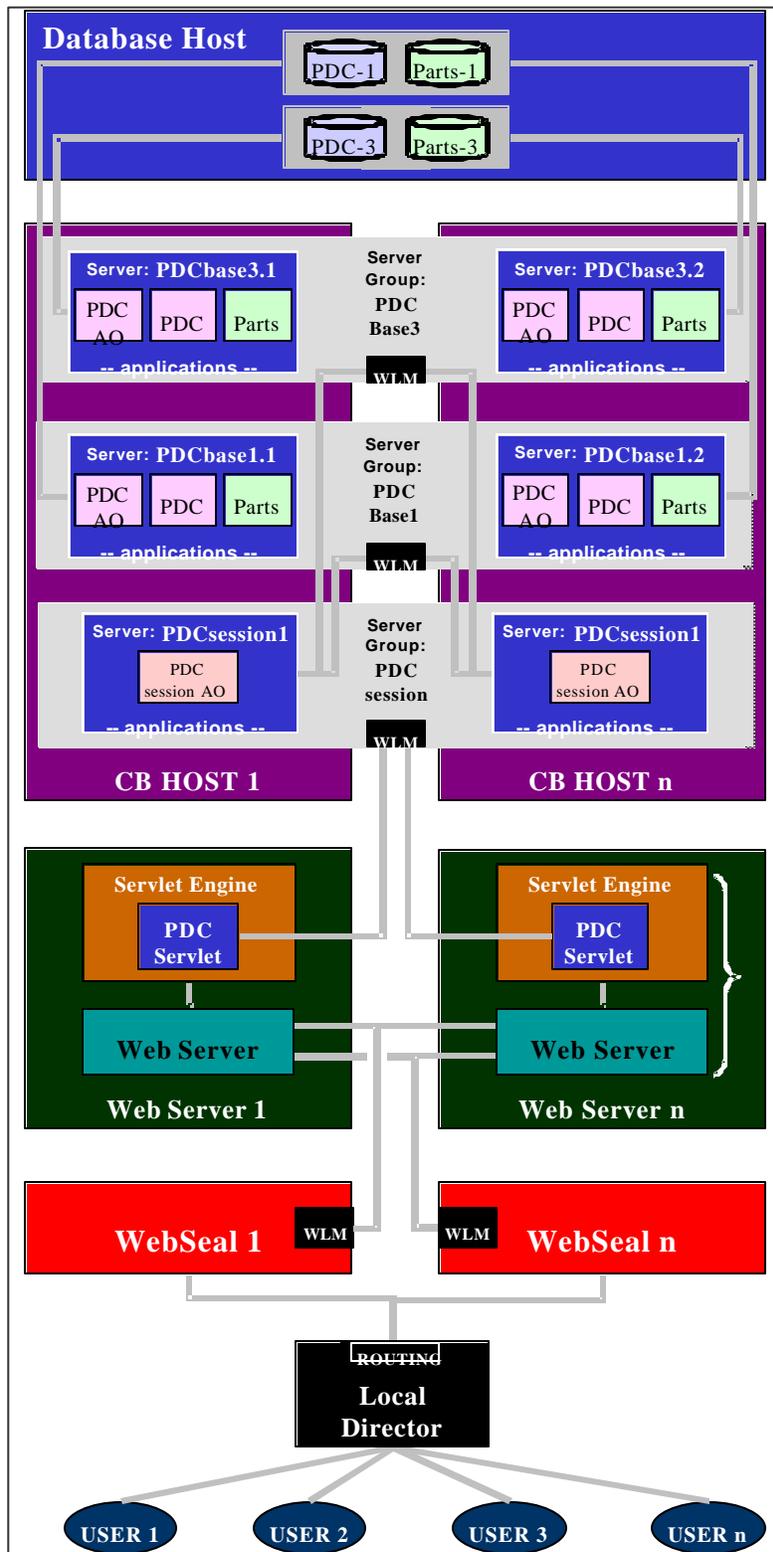


Figure 60: Illustration of Workload Managed Configuration

5.4 Data

5.4.1 Overview

The Data layer consists of the components that are directly related to providing persistence for business data. But note that some data layer components, depending upon their function, may in fact **not** persist their data. The data layer also provides services for implementing relationships between objects and/or components and for performing queries on objects. The components that execute in the data layer do not have any user interface components, as they have no responsibility to interact with the user.

While the data layer is depicted as separate from the Business Logic layer, this depiction is from a logical standpoint. In practice it will be seen that some of the data components of an application are physically co-located with their associated business layer components. While Business Logic components will include data items, they will generally have an associated data object that manages the data, including any persistence of the data. It is the data object that provides the interface to back-end data stores.

5.4.2 Service Use

The services provided by the Integration Framework to a data layer component include:

Table 24: IF Data Layer Component Services

Services	Functionality
Application Server Containers (IBM WebSphere for CORBA, EJB components)	From a data perspective, the containers provide for data persistence and data transactions. Reference: <i>WebSphere Application Server, Introduction to WebSphere Application Server</i> <i>WebSphere Application Server Enterprise Edition Component Broker, Programming Guide</i> ImanagedCollections chapter in the Managed Object Framework of: <i>WebSphere Application Server Enterprise Edition Component Broker, Programming Reference</i>
Object Transaction Service (OTS)	Provides the overall transaction service for all component transactions. Reference: Transaction Service chapter of: <i>WebSphere Application Server Enterprise Edition Component Broker, Advanced Programming Guide</i> CosTransactions chapter in the Managed Object Framework of: <i>WebSphere Application Server Enterprise Edition Component Broker, Programming Reference</i>
Java Transaction Service (JTS)	Provides <u>explicit</u> transaction service to Java components. Reference: <i>Java Transaction Service Specification, Version: 1.0, December 8, 1999</i> <i>Java Transaction API Specification, Version: 1.0.1, April</i>

Services	Functionality
	29, 1999
Concurrency Service	<p>Provides the ability to explicitly “lock” resources to prevent corruption of resource updates resulting from simultaneous updates of the same resource by multiple components.</p> <p>Reference: Concurrency Service chapter of: <i>WebSphere Application Server Enterprise Edition Component Broker, Advanced Programming Guide</i> CosConcurrencyControl chapter in the Managed Object Framework of: <i>WebSphere Application Server Enterprise Edition Component Broker, Programming Reference</i></p>
Query Service	<p>Provides the ability to perform structured queries on objects.</p> <p>Reference: Query Service chapter of: <i>WebSphere Application Server Enterprise Edition Component Broker, Advanced Programming Guide</i> CosQuery, CosQueryCollection, Iextended Query in the Query Service chapters and Object-Oriented SQL chapter of: <i>WebSphere Application Server Enterprise Edition Component Broker, Programming Reference</i></p>
JDBC, ODBC Drivers (For Oracle 8i, IBM DB2)	<p>Provides the ability for a Java or C/C⁺⁺ object / component to manage its own persistence.</p> <p>Reference: <i>Oracle8i JDBC Developer's Guide and Reference, Release 8.1.5, Part No. A64685-01</i> <i>IBM DB2 Universal Database Enterprise – Extended Edition, Quick Beginnings, Version 6, SC09-2832-00</i></p>
Relational Databases (Oracle 8i, IBM DB2)	<p>Provides the back-end data store for object / component persistence. Can also be accessed using SQL and native product client services.</p> <p>Reference: <i>Oracle8i Concepts, Release 2 (8.1.6), December 1999, Part No. A76965-01</i> <i>IBM DB2 Universal Database Administration Guide, Design and Implementation, Version 6, SC09-2839-00</i></p>

For details of these services refer to the manuals and documents identified as references.

5.4.2.1 Data Objects and Persistence

Relative to application development in the area of data, two primary objects are key. They are the data object and the persistent object. These objects are not accessible from clients and are specifically associated with a particular business object.

Data Object (DO) - that manages the essential state of the business object including the persistent storage of this data. Only one data object can be used to map a business

object's data to database entities and/or attributes. The data object isolates the business object from having to:

- Know which of many data stores to use to *persist* its state.
- Know how to access the data store.
- Manage the data store access.

Data objects are only applicable to managed objects such as the EJB entity beans or CORBA managed object. The data object uses the associated persistent object to persist the data in the data store mechanism (relational database for the current Integration Framework).

The data object, if required, can implement data specific rules. These rules are actually implemented by the DO Implementation object that provides the actual method implementations.

Persistent Object (PO) - Assists the data object in storing the business object's essential state to a specific data store. It encapsulates the embedded SQL statements needed to insert, update, delete, and retrieve the essential state to and from the data store. It provides the mapping of data from code format to data store format and maintains a key that is used to locate a corresponding entry in the data store.

The persistent object contains the same attributes as the data object; so the mapping from data object to persistent object is very straightforward. By delegating the (SQL) communication with the data store to persistent object, the data object code can be kept understandable and clean. It also allows the same data object to be used with different back-end data stores, as a different persistent object can be provided for different data stores.

The relationship of these objects to the business object they support is shown by example in Figure 61: Business, data, and Persistent Object Relationships. The objects illustrated are from the Integration Framework Parts Data Collection Test Component. Also refer to the *WebSphere Application Server Enterprise Edition Component Broker, Programming Guide* for details of developing these objects using the Integration Framework supported WebSphere application server.

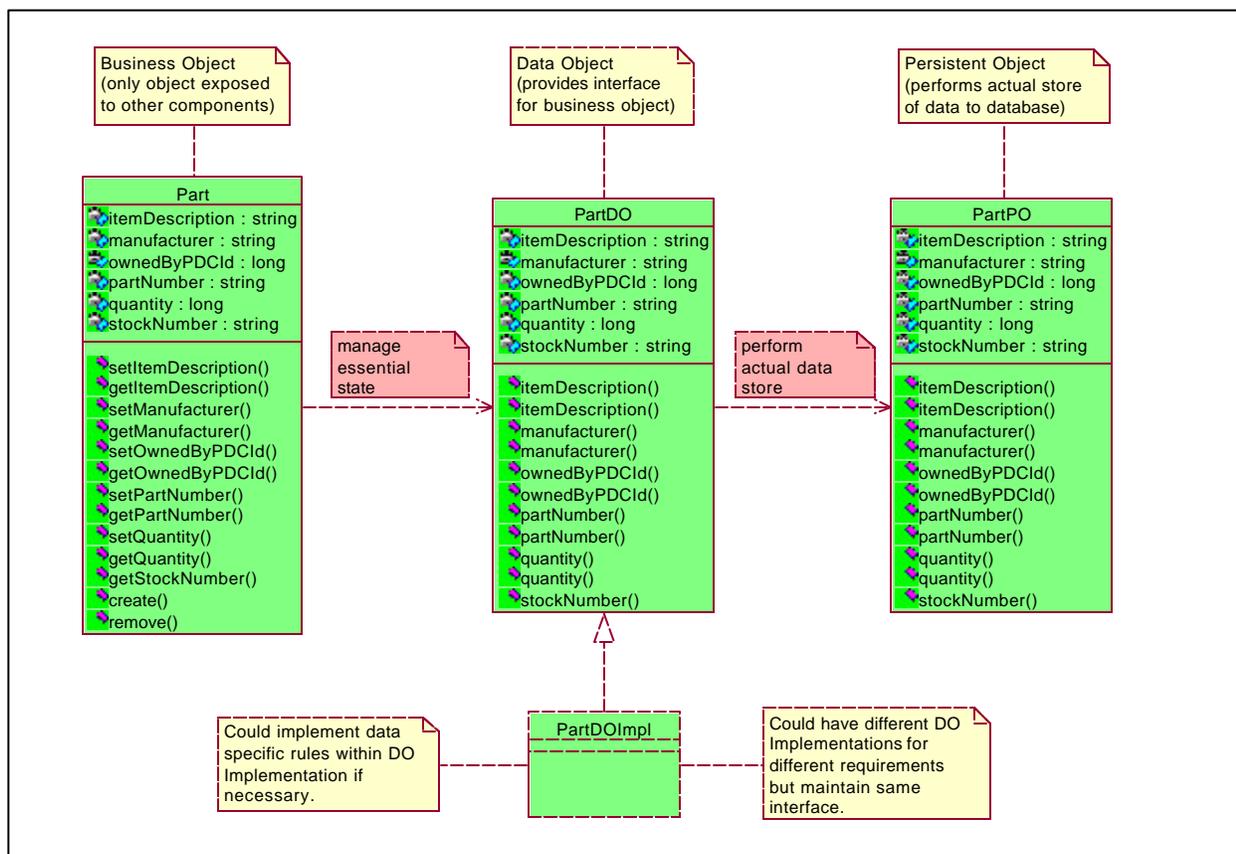


Figure 61: Business, data, and Persistent Object Relationships

Developers employing the current Integration Framework Application Server (IBM WebSphere Enterprise Edition) can utilize the associated development tools (Object Builder) to create these objects. Refer to the *WebSphere Application Server Enterprise Edition Component Broker, Application Development Tools Guide* for details. For developments not employing these tools, the objects would need to be created “by hand”.

In an enterprise as large and varies as GCSS-AF, there will be cases where a business object that provides a single interface to a group of closely related objects. Some of these objects may include object utilizing Legacy databases, data residing in different databases or even database types (e.g. Oracle, DB2). But as previously identified, a business object can only have a single associated data object the implications of which include only a single database and a single database type. To address this situation the concept of a composite business object is defined.

5.4.2.1.1 Composite Business Object

A Composite Business Object is a business object that provides a single interface to a closely related group of objects. In addition to providing its own methods and attributes, the composite business object can also expose any or all methods of the member objects. In this fashion the composite business object can include constituent objects of different implementation languages or backed by different databases or database types.

In employing a composite business object, the following advantages and disadvantages should be considered.

Advantages:

- Simplifies client interface to the Business Logic.
- Reduces the number of remote method calls.
- Provides the means for one business object to access more than one data store (database).

Disadvantages:

- Requires extra coding in order to integrate and properly initialize the constituent objects.
- Activating a composite business object may needlessly activate constituent objects.

Refer to the *Components Working Together* chapter of the *WebSphere Application Server Enterprise Edition Component Broker, Application Development Tools Guide* for details of implementing a composite business object using the Integration Framework supported WebSphere application server(s) and associated development tools.

5.4.2.1.2 Object Relationships

This section provides a summary for implementation of object relationships. Generally this will be implemented through the data object implementation. The data object implementation for a component that contains a reference to another component requires getters and setters that are more complex than those that implement attributes with simple mappings to backend resource managers such as SQL tables.

Refer to the *Assembling and Deploying Components* chapter of the *WebSphere Application Server Enterprise Edition Component Broker, Programming Guide* for details.

Top-down Versus Bottom-up Relations

The component's business object interface defines references, but does not know how the relationship is implemented. This is a key part of the encapsulation provided by the MOFW. In this section, implementation strategies and how they appear on the data object implementation are introduced.

There are two basic strategies that are applied to implementing object references that appear in the business object:

Top-down -This approach allows alteration or definition of a database schema that underlies a component that has references to other components.

Bottom-up -This approach implies preservation of an existing schema. While there are times when the object resolution approach characterized as bottom-up can be used for new top-down applications, the inverse is not true. The two strategies apply equally well to cardinality-1 and cardinality-n types of relationships.

Top-down Customizations

The top-down approach is characterized by the presence of a string in the database table of the containing object. This string contains information that the data object can use to produce the object reference required by the business object interface.

There are many different combinations of CORBA and Component Broker abstractions that could be used to map object references. The following useful patterns have been identified by the programming model:

Stringified Object Reference -Stores the stringified object reference as a variable length string. It is the simplest form in structure and the largest in size. Multiple references to the same object or other objects in the same application adaptor environment store duplicate prefix data.

Object Name -Stores a Naming Service name of an object as a variable length string. This is much shorter than the SOR. Only objects that are named in the Naming Service can be referenced using this pattern.

Home Name and Key-Used for objects that are not named in the Naming Service. Instead of the object name, it stores the name of the object home and its primary key within the home. The stored representation for this pattern is a pair of variable length strings: the home name and the stringified primary key.

Queryable Collection Name and Query String -Used for objects that are contained in a queryable home or named collection. It stores a pair of variable length strings: the collection name and a query string that uniquely returns the object.

Bottom-up Customizations

In the bottom-up case, in addition to knowing that the business object interface has a getter and a setter, there is also a known value in an existing database or other resource to which the data object is mapped. It is most probably a foreign key, a value that can be used, along with other Component Broker system function, to render the object reference which is being requested in the upper level (business object) interface.

FindByPrimaryKeyString -Using this pattern, a value (or values) from the underlying resource manager is used to formulate a key. The home for the kind of object being found is determined. The key is then used to do a `findByPrimaryKeyString` on the object.

Query -This pattern leverages the Query Service. In this case, the foreign-key is used as the basis to formulate a query into the table that contains the data for the referred objects. The result set can then be used to return values to the business object interface. This pattern works for both 1-to-1 and 1-to-n relationships with various exception handling required.

1-1 Relationships

For the top-down case, using the handle-pattern that is supported by Object Builder is recommended. The handle concept of Component Broker encapsulates the actual pattern that is

used to store the object reference. From this perspective, each 1-to-1 relationship in the top-down case stores a handle.

For the bottom-up case, Object Builder supports the foreign key pattern suggested and described previously. This is the recommended pattern.

1-n Relationships

For the top-down case, two patterns are supported by Object Builder for 1-n (one-to-many) relationships. Conceptually, they involve either keeping a reference collection of the object relationships or resolving the object relationships using the Query Service.

For the bottom-up case, the solution for 1-n relationships is to use the Query Service.

Summarizing Relationship Implementations

The patterns previously described are summarized in the following table.

Table 25: Cardinality Example

	Top-Down	Bottom-Up
Cardinality-1	Store the reference Uses handles support to store a stringified version of the object reference	Foreign key Uses findByPrimaryKeyString to locate referenced object
Cardinality-n	Reference collection Uses handles support to store a reference to a collection that contains the "object relationship"	Foreign key Uses query service to return the "object relationships" Requires a 1-to-1 reference to implement the other way

Additional Customizations

Additional data object customizations are possible. If the customization options that are optimal for a given relationship are not supported directly, it is possible to alter the mapping patterns that are used in such a way that Object Builder round tripping is still preserved. This is done through the use of a mapping helper.

Mapping Helpers

A mapping helper is a class that contains mapping methods. Mapping methods provide the conversion between the attribute types of the two objects. You can either use the mapping helpers provided by Object Builder, or you can define your own.

5.4.2.1.3 Container Managed Persistence

Just as the Integration Framework (provides and) recommends the use of application servers for hosting GCSS-AF applications and components, it also recommends delegation of the persistence business (object) data to the containers provided by the application server. For both Enterprise Java Beans and CORBA Managed Objects, the containers in which they “execute” can be configured to manage the persistence to the back-end data stores (databases). By doing so,

much of the tedious implementation “work” to make and manage requests to the back-end data store is eliminated.

Refer to the *WebSphere Application Server Enterprise Edition Component Broker, System Administration Guide* for details of configuring container-managed persistence using the Integration Framework supported WebSphere application server.

5.4.2.1.4 Object / Bean Managed Persistence

While the Integration Framework supports object or bean managed persistence, it does **not** recommend its use. Object or bean managed persistence can significantly reduce the (configurable) flexibility of using a component with different transaction policies, caching policies, etc. In addition, it places a greater burden on the developer to manage the persistence of the object and can have an impact on component portability. Object or bean managed persistence requires the use of an ODBC or JDBC driver in order to access the back end data store (database).

5.4.2.1.5 Caching Considerations

Caching of data relative to a distributed architecture is highly dependent upon the application server that is employed. As such, caching will be discussed in relative to the specific application servers supported.

5.4.2.1.5.1 IBM WebSphere Enterprise Edition Cache Capabilities

The Cache Service enhances concurrency and performance by supporting optimistic and pessimistic caching of data. In optimistic caching, frequently used data is cached in the memory of the Component Broker server and not reread from the database on each transaction. Cached data is invalidated based on a time-out value. Pessimistic caching is used when the application must be guaranteed current data and uses a higher degree of isolation to guarantee that transactions can be serialized. The caching mode is established through use of the WebSphere Enterprise Edition System Management User Interface on each object type.

The Integration Framework employing the IBM WebSphere application server provides significant flexibility relative to caching data. The Integration Framework allows caching to be configured for the business object, the data object, and/or the underlying database services themselves. It is up to the application developer to determine the applicability of caching to the specific application and its components as well as where and how caching is best employed. Refer to the *Cache Service* chapter of the *WebSphere Application Server Enterprise Edition Component Broker, Advanced Programming Guide* for details of utilizing caching services using the Integration Framework WebSphere EE supported application server(s). Refer to the *Cache Service* chapter of the *WebSphere Application Server Enterprise Edition Component Broker, System Administration Guide* for details of configuring caching services using the Integration Framework supported WebSphere application server.

5.4.2.2 Concurrency Service

The Concurrency Service is intended primarily for use in a transactional environment. It consists of a set of interfaces that allow an application to coordinate access by multiple transactions or threads to a shared resource. When multiple transactions or threads try to access a single resource at the same time, any conflicting actions are reconciled so that the resource remains in a consistent state.

Note: However the use of Concurrency Service by user application code within the application server-programming model is limited. This is due to the application server framework already providing the necessary resource-level locking and caching to coordinate resource access by multiple transactions or threads.

Refer to the *Concurrency Services* chapter of the *WebSphere Application Server Enterprise Edition Component Broker, Advanced Programming Guide* for details of configuring and utilizing concurrency services using the Integration Framework supported WebSphere application server.

5.4.2.3 Query Service

The Query Service enables you to find objects in an application server collection based on a set of conditions described with an object-oriented structure query language (OOSQL). The OOSQL enables you to describe complex search criteria. It is an extension of SQL with features for handling object collections, object attributes, and methods in query statements. The Query Service can return a list of object references, or it can return a list of object attribute values. The Query Service takes advantage of search capabilities and indexes in the underlying database to make searching for objects efficient.

Refer to the *Query Services* chapter of the *WebSphere Application Server Enterprise Edition Component Broker, Advanced Programming Guide* for details of configuring and utilizing query services using the Integration Framework supported WebSphere application server.

5.4.2.4 Transaction Service

The Transaction Service enables programmers to implement transactions by using standard object-oriented interfaces in a distributed environment. Application servers use the Transaction Service to ensure that each application has correctly grouped the updates in the transaction so that the data is always updated consistently. If the application uses the Transaction Service in conjunction with the Concurrency Service, these updates are not affected by updates being performed for other tasks.

Refer to the *Transaction Service* chapters of the *WebSphere Application Server Enterprise Edition Component Broker, Programming Guide* and *WebSphere Application Server Enterprise Edition Component Broker, Advanced Programming Guide* for details of configuring and utilizing the Transaction Service using the Integration Framework supported WebSphere application server(s).

The following paragraphs are provided to highlight key areas relative to transaction processing. They are not intended to eliminate the need to utilize the documentation identified above.

Top-level and Flat Transactions

The most common type of transaction used by applications is the top-level transaction. The first transaction created by an application is always a top-level transaction.

The CORBA specification also describes another type of transaction, called a *sub-transaction*, often referred to as a *nested transaction*. The Integration Framework provided Transaction Service does **not** currently support sub-transactions.

Lifetime of a Transaction

Applications use transactions to group related updates to data so that all or none of the updates occur. Typically, an application:

- Starts a transaction.
- Makes the updates and associates them with the transaction.
- Ends the transaction.

When an application ends a transaction, it can request that the transaction is either rolled back or committed. If the application requests a rollback, all of the updates are undone.

If the application requests that the transaction is committed, the Transaction Service checks that each object involved in the transaction is able to make its updates permanent. If all objects indicate that they can, the transaction is committed. Otherwise, the updates are undone just as if the application requested a rollback.

Transaction Scope and Context

The Transaction Service allows multiple objects to participate in a transaction. These objects can be distributed across multiple operating system processes and threads and each object can be working with more than one transaction at once.

To control which transaction an object is working on at a particular point in the code, the Transaction Service provides a *transaction context*. This is a collection of Transaction Service objects that represents the transaction.

The *scope* of a transaction is made up of all the locations within your application where the transaction context is in use. In general, the scope of the transaction increases over the lifetime of the transaction as the transaction context is passed from object to object.

The Transaction Service provides two mechanisms for passing transaction context:

- The most common method is *implicit propagation*, where the transaction context is associated with a thread and is available to each method called within this thread that understands transactions. If a remote method is called, the Transaction Service

automatically passes the transaction context to the thread in the remote server process where the method is run.

- An alternative method is for the application to pass the `CosTransactions::Control` reference as an explicit parameter on a method call. The recipient method could then resume the context on its thread of execution. This is called *explicit propagation*. The Component Broker Transaction Service implements checked behavior that disallows resumption of an explicitly propagated Control if that Control has been propagated between execution environments. Passing a Control between threads of the same execution environment is allowed.

Transaction Retry Limits

The Transaction Service can be configured to limit the number of times it attempts to contact a server during the two-phase or one-phase commit. If this limit is reached, the Transaction Service uses a pre-configured value as the action taken by the unavailable objects.

Transaction Time Limits

An application can set a time limit for its transactions. This time limit applies in all operating system processes and threads that are part of the transaction's scope. It is specified as the transaction is started and runs until the call is made to stop the transaction.

When the time limit is reached, the transaction is said to have *timed out* and if the call to stop the transaction has not been made, the Transaction Service rolls back the transaction.

Container Managed Transactions, non-EJB Components

To enable the container in which a component is deployed to manage the transactions for the component, it is necessary to configure the transaction policy for the container appropriately.

The Integration Framework supported WebSphere application server provides the following configurations if a method of the component is invoked without a transaction being active:

- Throw exception
- Begin a new transaction
- Ignore condition

Container Managed Transactions, EJB Components

A transaction attribute is configurable per the EJB specification that defines the transactional manner in which the container invokes enterprise beans. The current Integration Framework supported WebSphere application server supports this for the bean as a whole; individual transactional attributes for each method is not supported. The EJB specification identifies the following configuration parameters:

- TX_REQUIRED
- TX_REQUIRED_NEW (WebSphere does not support)

- TX_SUPPORTS (WebSphere does not support)
- TX_NOT_SUPPORTED (WebSphere does not support if used with container managed persistence)

5.4.2.4.1 Object Transaction Services

As can be seen from the above description, the Object Transaction Services (OTS) provided by the Integration Framework is in fact the CORBA Services OTS. Limitations and extensions provided by the current IF supported application server are described in the *Transaction Service* chapter of the *WebSphere Application Server Enterprise Edition Component Broker, Advanced Programming Guide*.

However, for applications / components implemented under the Integration Framework supported application server, the containers in which the components are deployed should be configured to manage the transactions..

5.4.2.4.2 Java Transaction Services

While the Java Transaction Services and the Java Transaction API are supported, they should generally not be employed directly by applications / components implemented under the Integration Framework supported application server. Rather the containers in which the components are deployed should be configured to manage the transactions. For details of the Java Transaction Services and the Java Transaction API refer to the *Java Transaction Service Specification and the Java Transaction API Specification*.

5.4.2.4.3 Transactions With Legacy Systems

While the Integration Framework provides the services to implement many Legacy system interfaces requiring transactions, this has not been a capability with which the Integration Framework development has been tasked. The Integration Framework supported WebSphere application server provides procedural application adaptors to interface to existing systems and COTS packaged systems. The Procedural Application Adaptor (PAA) of Component Broker enables Component Broker applications to access procedural resources such as CICS, IMS, or SAP.

Component Broker provides the CICS / IMS / SAP Application Adapter that is based on its Procedural Application Adaptor infrastructure. It consists of both a development environment within Component Broker, as well as a run-time environment. The run-time environment integrates the Component Broker services such as transaction capability with the various technologies to communicate with these procedural systems like Communications Server, CICS Universal Client, SAP Connector, and Host On-Demand.

The actual interface with the Legacy system that would utilize the Procedural Application Adaptor should be incorporated in an Interface Component that provides the GCSS-AF component view (and interface) to other GCSS-AF components.

For details of the Procedural Application Adaptor *WebSphere Application Server Enterprise Edition Component Broker, Procedural Application Adaptor Development Guide*.

5.4.2.4.4 Messaging in the Transaction

Transaction scope relative to the Integration Framework Message services is the commit of a message to a queue (for a put) or the commit of the removal of a message from a queue (for a get). What this means is that when a message is “sent” to another component inside of a transaction, the action that is contained within the transaction is the actual queuing of the message on the specified queue; until the commit the message is essentially staged. If a rollback is requested than the message is never queued. Conversely, when a message is retrieved from a queue inside of a transaction, the action that is contained within the transaction is the retrieval of the message from the specified queue but the message is not removed from the queue until the commit occurs. If a rollback is requested than the message is never removed from the queue. Note that the actual delivery of the message from one component to another is not included in the transaction scope.

Message sends to and gets from components implemented in the Integration Framework supported WebSphere application server are accomplished through a Message Adaptor (container) which manages the transaction with the underlying messaging capability (MQSeries).

5.4.2.4.5 Inter- Application Transactions

GCSS-AF has specified that the primary mechanism of inter-application data interchanges be accomplished using messaging to send BODs. As such, the applicability of a two-phase commit protocol is not provided. However, modern application design paradigms emphasizes loose coupling between applications and even components of an application and typically do not require this type of protocol. Where necessary, design patterns exist that can provide a similar end result. Figure 62: Example Transaction Emulation Across Components illustrates one possible way of achieving this.

For those cases where a two-phase commit is absolutely necessary, the Integration Framework supports method level invocation passing a BOD as a parameter. However, if employed it should be restricted to transactions only within a local-area network and not a wide-area network.

5.4.2.5 Database Access Services

Container Managed Persistence

The containers, provided by the Integration Framework supported WebSphere application server, provide database access for the components which it contains.

JDBC

JDBC drivers are available for components not employing container managed persistence to access data store.

SQL

SQL can be used by components operating outside of the Integration Framework supported WebSphere application server.

5.4.2.6 Security

Data Object Access

As Data Objects (and Persistent Objects) are co-located within the same component as the business object for which it manages data persistence and provides no exportable interface, there is no requirement for securing access to the data object.

Database Access

For containers that access a specific database, a userid/password is configurable when the component is deployed to provide authorization checks by the database.

Fine-Grained Access Control

While the Integration Framework does not currently provide validated fine-grained access control patterns and approaches, mechanisms are available by which this can be implemented where the need arises. This includes the ability to provide a userid or role attribute with each data object and have data object implementation test retrieved objects userid or role attribute against the requesting user or user role.

5.4.3 Communication Between Layers

5.4.3.1 Vertical

5.4.3.1.1 Presentation

Servlet and Data Object

For presentation components that need to access data (as objects) directly, a data object can be developed as either a JavaBean or simply as a class and have the access to the back-end data store accomplished by the data object using either JDBC or SQL. The communications between the servlet and data object would simply be method invocation using parameters as required.

Servlet and Relational Database

This should be accomplished as identified above for Servlet and Data Object.

5.4.3.1.2 Business Logic

Business Object and Data Object

Business Object access data objects as a “local object”. For cases where caching is configured for a business object, the business object must provide `syncToDataObject` and `syncFromDataObject` methods to be invoked by the data object as needed. The data object (through the persistent object) provides the actual access to the back-end data store.

Business Object and Relational Database

This should be accomplished as identified above for Business Object and Data Object.

5.4.3.2 Horizontal

5.4.3.2.1 Database

Data Object and Relational Database

The Data Object accesses the relational database through the persistent object that employs SQL. For data objects outside of the Integration Framework supported WebSphere application server, JDBC drivers, ODBC drivers, or SQL can be employed to access relational databases.

Database Replication

The Integration Framework supported relational databases provide the ability to replicate information to synchronize database (copies).

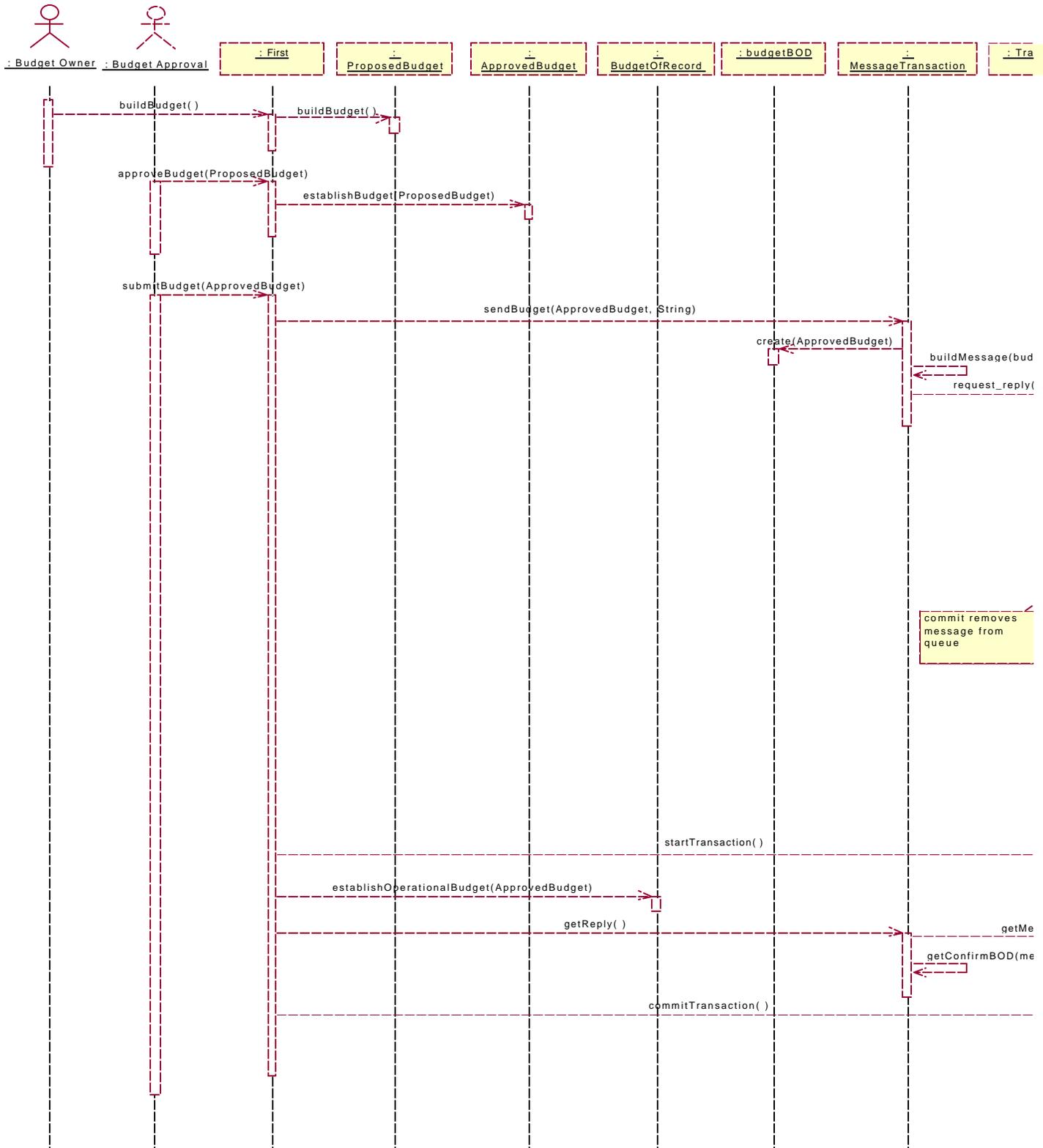
5.4.3.2.2 Transaction Managers

Object Transaction Service objects communicate using IIOP (CORBA) to coordinate transactions. This is a standard part of the OTS.

5.4.3.2.3 Resource Managers

The Object Transaction Service provided by the Integrated Framework can interoperate with XA-compliant resource managers.

Figure 62: Example Transaction Emulation Across Components



5.5 Messaging Guidance

5.5.1 Message Transmission and Reception Types

Messaging systems support multiple styles of communication between applications that include *send and forget / receive*, *request/reply* and *publish/subscribe*.

An application using the *send and forget* style sends a message to another application or list of applications but does not expect any reply. An example is a data replication application.

An application using the receive style receives a message from another application and sends no reply. An example is a data display terminal which accepts data and displays it on a screen.

Applications using *request/reply* are like client server applications where a client is making a request from a server and expecting a reply.

Publish/subscribe is rather like *send and forget* except that the sending application does not know who the recipients are. Instead the sender sends the message to a broker who manages the subscriptions of applications that requested to receive messages. The broker decides which subscribing applications should receive a published message by matching the subscriptions with either the message topic information or the content of the message.⁹

5.5.1.1 Send and Forget / Receive

The *send and forget* style of messaging has a sender application which sends an outgoing message and does not expect to receive a response message back. An example of a *send and forget* messaging style is a stock update service which sends updates to a display terminal for display to stock traders. The service sends messages and does not expect a reply back from the display terminals.

The Integration Framework test components use the send and forget messaging style when the EPD Wrapper sends a **SyncInventory** BOD to the Enterprise PDC without requesting a confirm BOD reply.

The *receive* style of messaging has a receiver application which receives incoming messages and does not send a response message. An example of a receiver message style is a data terminal that receives updates from a master source and displays them on a monitor. The data terminal receives incoming messages and acts on them, but does not produce a response message.

5.5.1.2 Request / Response

The *request/response* style of messaging has a requester application which sends a *request* message and expects to receive a *response* message back. The responder application receives the

⁹ Open Applications Group Common Middleware API Specification

request message and produces the response message(s) which is sent back to the requester application. The responder application uses information in the request message to know how to send the response message back to the requester.¹⁰

The Integration Framework test components use the request/response messaging style when the EPD Wrapper sends a **SyncInventory** BOD to the Enterprise PDC requesting a confirm BOD reply. The Enterprise PDC processes the **SyncInventory** BOD and returns a confirm BOD to the EPD wrapper.

5.5.1.3 Publish / Subscribe

With *publish/subscribe* messaging a publisher application publishes messages to subscriber applications via a broker. The message published contains application data and one or more topics strings that usually describe the data. A subscribing application subscribes to topics informing the broker which topics it is interested in. When the broker receives a message from a publisher it compares the topics in the message to the topics in the subscription from subscribing applications. When they match the broker forwards the message to the subscribing applications.¹¹

The Integration Framework test components use the *publish/subscribe* messaging style when the Enterprise PDC publishes part update information to the subscribing Base PDCs.

5.5.1.4 Synchronous Communications Emulation

With synchronous communications emulation messaging an application sends a message to a receiving application and waits for the response message to be received. This is really a kind of *request / response* messaging in which the sender simply waits for an answer, whereas in *request/response* messaging, the sender may not wait for a response; instead responses may be processed asynchronously.

For this case, when two or more applications wish to communicate synchronously but still wish to use the MQSeries messaging infrastructure, it is recommended that each application define an inbound queue and that the application code for the conversation be written using blocking reads on the input queues. This will approximate synchronous communication when the messaging infrastructure is operating nominally.

The Integration Framework test components use synchronous communications emulation messaging between the EPD Wrapper component and the Enterprise PDC component when the EPD Wrapper sends the **SyncInventory** BOD and requests a confirm BOD back from the Enterprise PDC.

¹⁰ Open Applications Group Common Middleware API Specification

¹¹ Open Applications Group Common Middleware API Specification

5.5.1.5 Quality of Services Options

5.5.2 Message Queue Configuration Considerations

The heart of MQSeries is the message queue manager (MQM), MQSeries' run-time program. Its job is to manage queues and messages for applications.

A program may send messages to another program that runs in the same machine as the queue manager, or to a program that runs in a remote system, such as a server or a host. The remote system has its own queue manager with its own queues. The queue manager transfers messages to other queue managers via *channels* using existing network facilities, such as TCP/IP, SNA or SPX. Multiple queue managers can reside in the same machine. They also need channels to communicate.

Application programmers do not need to know where the program to which they are sending messages runs. They put their messages on a queue and let the queue manager worry about the destination machine and how to get the messages there. MQSeries knows what to do when the remote system is not available or the target program is not running or busy.¹²

Except for *publish/subscribe*, it is recommended that applications define one queue for each incoming BSR message flow. Applications may choose to further define queues to separate messages based on some application-specific discriminator, e.g. message size or message priority.

Where possible, queues should be configured to trigger execution of the programs that will process the messages on the queue. The MQSeries option "trigger on first" is recommended. Triggered programs should be written to process messages until no more messages are on the queue.

About Message Queues

Queues are defined as objects belonging to a queue manager. MQSeries knows a number of different queue types, each with a specific purpose. The queues the developer uses are located either in the machine and belong to the queue manager to which it is connected, or in the server (Client side). Figure 10 lists different queue types and their purposes. More detailed information is below.

Queue Types

Local queue	A real queue
Remote queue	Structure describing a queue
Transmission queue (xmitq)	Local queue with special purpose

¹² MQSeries Primer, IBM

Initiation queue	Local queue with special purpose
Dynamic queue	Local queue created "on the fly"
Alias queue	Alternate name
Dead-letter queue	One for each queue manager
Reply-to-queue	Specified in request message
Model queue	Model for local queues
Repository queue	Holds cluster information ¹³

The Integration Framework test components use all of the preceding queue types except the dynamic and repository queues.

5.5.2.1 Static Queues

A static queue is defined to be just a queue that is not dynamic. Static local queues compose the majority of the queues used by the Integration Framework test components, and will likely compose the majority of the queues used by a new application. Static queues are predefined by an administrator; they are not created "on the fly" at runtime. Every Integration Framework test component uses at least one static local queue.

5.5.2.2 Dynamic Queues

A dynamic queue is defined "on the fly" when the application needs it. Dynamic queues may be retained by the queue manager or automatically deleted when the application program ends.

Dynamic queues are local queues. They are often used in conversational applications, to store intermediate results.

Dynamic queues can be:

- Temporary queues that do not survive queue manager restarts
- Permanent queues that do survive queue manager restarts¹⁴

The developer will want to use a dynamic queue if they want MQSeries to create a queue "on the fly" for the application at runtime. One case where a dynamic queue is useful is for generating reply-to-queues for client applications that send messages to a server's input queue. The client opens a model queue and gets the name of a dynamic queue that MQSeries creates for the client. Then the client sets the reply-to-queue information (in the message it is to send to the server) to be the name of the dynamic queue. The client then sends the message to the server and listens on the dynamic queue for the response from the server.

¹³ MQSeries Primer, IBM

¹⁴ MQSeries Primer, IBM

Use of a dynamic queue is recommended for those cases where the developer does not need the queue after the application ends. For example, the developer may want to use a dynamic queue for the “reply-to” queue. The name of the reply-to queue in the **ReplyToQ** field of the MQMD structure is specified when a message is put on a queue.

To create a dynamic queue, use a template known as a model queue, together with the MQOPEN call. The developer creates a model queue using the MQSeries commands or the operations and control panels. The dynamic queue created takes the attributes of the model queue.

When the developer calls **MQOPEN**, specify the name of the model queue in the *ObjectName* field of the MQOD structure. When the call completes, the **ObjectName** field is set to the name of the dynamic queue that is created. Also, the **ObjectQMgrName** field is set to the name of the local queue manager.¹⁵

The Integration Framework test components do not use any dynamic queues.

5.5.2.3 Local and Remote Queues

Local Queue

A queue is local if it is owned by the queue manager to which the application program is connected. It is used to store messages for programs that use the same queue manager. For example, program A and program B each has a queue for incoming messages and another queue for outgoing messages. Since the queue manager serves both programs, all four queues are local.

Remote Queue

A queue is “remote” if it is owned by a different queue manager. A remote queue definition is the local definition of a remote queue. A remote queue is not a real queue. It is a structure that contains some of the characteristics of a queue hosted by a different queue manager. The application programmer can use the name of a remote queue just as he or she can use the name of a local queue. The MQSeries administrator defines where the queue actually is. Remote queues are associated with a transmission queue.

Notes:

- A program cannot read messages from a remote queue.
- The developer does not need a remote queue definition for a cluster queue.¹⁶

A remote queue will need to be used when the developer wants queues on a remote queue manager to be visible on another queue manager. An alternative to use of remote queues is

¹⁵ Pg 104, IBM “MQSeries Application Programming Guide” manual

¹⁶ MQSeries Primer, IBM

clustering, which is commonly used for load balancing and workload management as well as for making queues visible across queue managers.

The Integration Framework test components use remote queues on one queue manager, called **QMR1A1**, to communicate with the publish/subscribe broker running on a different queue manager called **QMR1C1**. As mentioned above, though, the majority of the queues used by the test components are static local queues.

5.5.2.4 Queue Aliases

Alias queues are not true queues but definitions. They are used to assign different names to the same physical queue. This allows multiple programs to work with the same queue, accessing it under different names and with different attributes.¹⁷

The Integration Framework test components use alias queues to allow the requisitioning component to talk with the pseudo-supply Legacy application. The requisitioning component constructs the name of the queue it will use at runtime, which is defined on a per-base basis, and aliases are defined for all the possible names that could be constructed. These aliases point to the input queue for the pseudo-supply Legacy application input queue, which is defined on a regional basis. More concretely, the requisitioning component can generate the queue names

PSEU.BASE1.ADDREQUISITN.INBOUND

PSEU.BASE3.ADDREQUISITN.INBOUND

both of which are defined as queue aliases pointing to the queue

PSEU.REGION1.ADDREQUISITN.INBOUND

which is where the pseudo-Legacy supply component looks for input.

5.5.2.5 Cluster queues

A cluster queue is a local queue that is known throughout a cluster of queue managers, that is, any queue manager that belongs to the cluster can send messages to it without the need of a remote definition or defining channels to the queue manager that owns it.

There are two quite different reasons for using clusters: to reduce system administration and to improve availability and workload balancing.

In a traditional MQSeries network using distributed queuing, every queue manager is independent. If one queue manager needs to send messages to another it shall have defined a transmission queue, a channel to the remote queue manager, and a remote queue definition for every queue to which it wants to send messages. If the developer groups queue managers in a

¹⁷ MQSeries Primer, IBM

cluster, the queue managers can make the queues that they host available to every other queue manager in the cluster. Then, assuming the developer has the necessary network infrastructure in place, any queue manager can send a message to any other queue manager in the same cluster without the need for explicit channel definitions, remote queue definitions, or transmission queues.

As soon as the developer establishes even the smallest cluster they will benefit from simplified system administration. Queue managers that are part of a cluster need fewer definitions and so the risk of making an error in the definitions is reduced.¹⁸

5.5.2.5.1 Fault Tolerance and Load Balancing

The developer can set up a cluster of queue managers that has more than one definition for the same queue (for example, the queue managers in the cluster could be clones of each other). Messages for a particular queue can be handled by any queue manager, which hosts an instance of the queue. A workload-management algorithm decides which queue manager handles the message and so spreads the workload between the queue managers.

The developer may organize the cluster such that the queue managers in it are clones of each other. This means they are able to run the same applications and have local definitions of the same queues. Because the developer can have more than one instance of an application, each receiving messages and running independently of each other, the workload can be spread between their queue managers.

The advantages of using clusters in this way are:

- Increased availability of queues and applications
- Faster throughput of messages
- More even distribution of workload in the network

Messages destined for a particular queue can be handled by any one of the queue managers that host an instance of that queue. This means that applications need not explicitly name the queue manager when sending messages. A workload management algorithm determines which queue manager should handle the message.

Because more than one queue manager is able to handle the same message, the risk of delayed delivery when a queue manager or communications link is unavailable is greatly reduced. The workload management algorithm tries one queue manager after another, if an initial attempt to deliver a message should fail.¹⁹

¹⁸ Pg. 41, IBM “MQSeries Queue Manager Clusters” manual

¹⁹ Pg. 41, IBM “MQSeries Queue Manager Clusters” manual

Use of MQSeries clustering is recommended for applications with high-throughput or high-availability requirements. Clustering should not be performed across a wide-area network. For this reason, clustering is recommended primarily for use within an RSA.

The Integration Framework test components do not use queue clustering.

5.5.2.6 Message Policies

A policy is an administrator created definition, external to the application, that controls how the commands operate. A program selects a policy by passing a policy name as a parameter on calls.

Policies are defined in the repository and can be used to control, for example:

1. The attributes of the message, such as priority.
2. How the call operates, such as whether the call is part of a transaction.
3. Whether added value functions are to be invoked as part of the call, such as auditing and exception handling.

An application could choose to use a different policy on each call and only specify those parameters in the policy that are relevant to the particular call. It would then be more possible to have policies shared between applications such as **Transactional_Persistent_Put**. Another approach would be to have application specific policies that specified all the parameters for all the calls made in an application. For example **Payroll_Client policy**. Both uses are possible using AMI.²⁰

The AMI will automatically retry when temporary errors are encountered on sending a message, if requested by the policy. (Examples of temporary errors are queue full, queue disabled, and queue in use).²¹

It is recommended that a systems administrator define use the AMI System Administration tool to define policies and to store them in a “master” repository, and that the master repository be used to regularly refresh “slave,” or subordinate, repositories deployed in other locations.

The default settings for AMI policies were adequate for use in the Owego SIL. Policy definitions will depend directly on the application characteristics. It is recommended that a system administrator become familiar with the MQSeries AMI policy options and work with application developers to define policies suitable for each deployed application. Use of the default IBM policies is recommended unless there is an application-specific reason to deviate from them.

²⁰ Open Applications Group Common Middleware API Specification

²¹ Pg. 6, IBM “Application Messaging Interface” manual

The Integration Framework test components use AMI, and therefore AMI policies, in two places: the EPD Wrapper component and the pseudo-Legacy supply component. The EPD Wrapper component uses the Lockheed-developed AMI Helper classes, a Java interface to the generic AMI Java services, to communicate with MQSeries. The pseudo-Legacy supply component uses the C++ bindings for AMI to communicate with MQSeries. For each component, an administrator pre-defined the AMI service points and policies, which the applications use.

5.5.2.7 Message Logging

Logging for Persistent Messages

Circular logging-Use circular logging if all that is wanted is restart recovery, using the log to roll back transactions that were in progress when the system stopped. Circular logging keeps all restart data in a ring of log files. Logging fills the first file in the ring, then moves on to the next, and so on, until all the files are filled. It then goes back to the first file in the ring and starts again. This continues as long as the product is in use, and has the advantage that the developer never runs out of log files.

Linear logging-Use linear logging if both restart recovery and media or forward recovery is wanted (recreating lost or damaged data by replaying the contents of the log). Linear logging keeps the log data in a continuous sequence of files. Space is not reused, so the developer can always retrieve any record logged from the time that the queue manager was created.

As disk space is finite, the developer may have to think about some form of archiving. It is an administrative task to manage disk space for the log, reusing or extending the existing space as necessary.²²

5.5.2.7.1 Recommendation

While linear logging offers recoverability and auditability superior to that of circular logging, linear logging is also much more difficult to administer, particularly across a large enterprise. Therefore it is recommended that circular logging be used. The Integration Framework test components use MQSeries configured with circular logging (the default logging style).

5.5.2.8 Naming Conventions and Services

It is recommended that MQSeries installations conform to the DISA guidelines except as noted in Section 5.5.2.8.1 below. Topics should be named as follows:

Location of sender + "/" + "application id" + "application message type", e.g.,
"Base1/Payroll/EmployeeSummaryList".

²² Pg. 215, IBM "MQSeries System Administration" manual

5.5.2.8.1 Deviations from and uses of the DISA naming convention standards

Paragraph numbers refer to the DISA MQSeries naming convention standards document.

GCSS will deviate in most cases from paragraph 3.8.3.7.2²³ by omitting the name of the application sending the message and the associated application details.

5.5.2.8.2 Usage of the DISA naming convention standards

- GCSS will specify the location of an application with an identifier immediately following the application name. And GCSS will specify the Business Object document type in an identifier immediately following the location identifier. Example:
PDC.BASE1.SYNCINVENTORY.INBOUND (inbound **SyncInventory** BOD for Parts Data Collection component at Base 1).
- The application component of the queue name will be at most 8 characters in width. The location name component of the queue name will be at most 9 characters in width. The Business Object Document type component of the queue name will be at most 20 characters in width. The GCSS Enterprise Authority will make sure that location, Business Object document, and application abbreviations are unique and consistently applied across the enterprise.

0								1									2										3										4							
1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8							
Application Name (8-char max)								▪	Location Name (9-char max)									▪	BOD Name (20-char max)										▪	INBOUND, OUTBOUND or RECEIVER (optional)														

Figure 63: DISA Naming Conventions Model

Examples:

- **ACPS.EGLIN.SYNCINVENTORY.INBOUND**
- **CPARS.LUKE.GETLISTITEM.OUTBOUND**
- **DMLSS.WRGHTPAT.UPDATEINVENTORY.INBOUND**
- **ARMS.DOVER.CONFIRMBOD.INBOUND**

In these examples, the location Wright-Patterson AFB was abbreviated as **WRGHTPAT**. The GCSS Enterprise Authority will create and manage such abbreviations.

²³ DISA MQSeries naming convention standards document

3. The application name “**IF**” is reserved for use by the Integration Framework.
4. Stream queues used for publish/subscribe will use a BOD name of “**DEFAULT.STREAM**” and will omit the **INBOUND/OUTBOUND** optional suffix.
5. Queues used to receive publications will use the **RECEIVER** suffix.

5.5.2.9 Tips for MQSeries developers and administrators

- **Applications should set message priority and persistence directly rather than relying on the queue defaults.**

This is important because after an application is deployed, it is possible that a system administrator could change the default message priority and/or persistence attributes on the queue(s) the application is using, which change might inadvertently break the application.

- **All MQSeries logging should be done to fast devices, e.g., SCSI disk drives.**

This is because MQSeries writes to the log for each persistent message. Thus the speed of log writes can put an upper bound on queue manager throughput.

- **Each major application should have its own queue manager.**

If two applications share a queue manager and one is high-volume, it is possible that the high-volume application will saturate the queue manager to the detriment of the low-volume application.

- **Different channels should be defined only when there is something physically different about the transports, e.g., they use different hardware or communications protocols.**

If two channels ultimately send data across the same piece of physical wire, the *throughput* is limited by the wire, not by the channels. Because of this, it is claimed that one channel is as good as two.

Separation of channels by communications protocol is recommended to ease administration of the MQSeries installation.

- **Applications developers should design their applications such that errors in triggered applications (1) send notification to an operator, and (2) turn off triggering.**

If no notification is sent to an operator, then it is possible the error will not be quickly noticed or corrected. If triggering is left on, the error can occur multiple times as messages are added to the queue.

- **Applications should be designed to write error reports to a central "error queue" which triggers pages or otherwise solicits human intervention.**

A central location for error monitoring and reporting is an important piece of an overall systems management strategy. This recommendation comes from IBM's experience of managing the Chicago Mercantile Exchange, a large MQSeries installation.

- **Most triggered applications should be TRIGGER_ON_FIRST.**

Most of the time we want a triggered application to “wake up” when there are messages on the queue, to process all the messages on the queue, and to exit when there are no more messages on the queue. This corresponds to trigger type FIRST.

If a triggered application is trigger type EVERY it is possible in a high-volume situation for the queue manager to exhaust system resources by starting too many instances of the triggered application.

If a triggered application is trigger type DEPTH, it is possible that the application will not be triggered in a timely fashion – if fewer than *TriggerDepth* messages are placed on the queue.

See the MQSeries Application Programming Guide, chapter 14 for a thorough discussion of triggering.

- **Clustering is not recommended if the queue manager is not saturated, unless fault failover is desired.**

Clustering incurs overhead that is best avoided unless the accompanying benefits are needed.

- **Applications should set the expiration date on non-persistent messages with a short lifetime.**

This is to avoid the need to “clean-up” old non-persistent messages later. If the expiration date of non-persistent messages is not set, the messages will remain on the queue until the next queue manager restart.

5.5.3 Publish/Subscribe approach

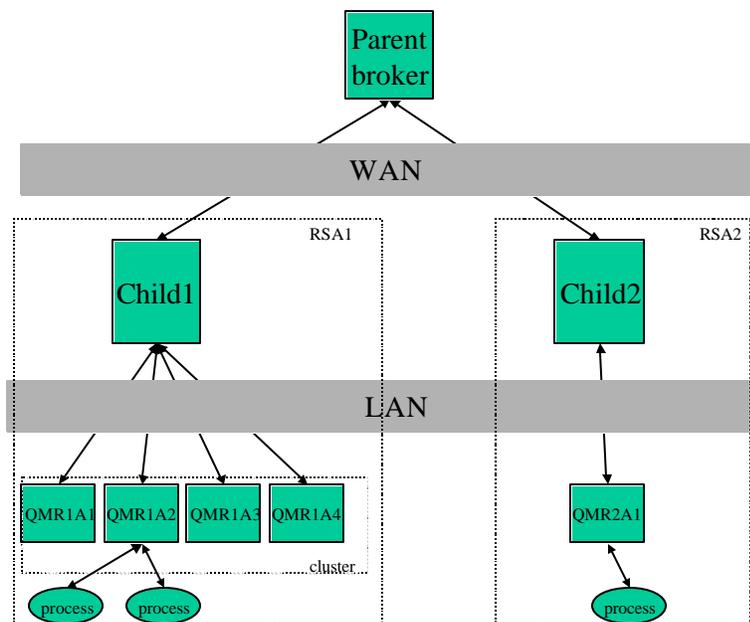


Figure 64: Parent/Child Hierarchy

It is desired for a process to be able to publish to its local queue manager and to have the publication be received by processes running on remote queue managers while minimizing WAN traffic. To this end it is recommended that queue managers be organized as depicted above in order to establish a *publish/subscribe* broker “skeleton” throughout the enterprise. A *publish/subscribe* broker process is run on the parent and on child queue managers. The child queue managers are connected to application queue managers via standard MQSeries distributed queuing (channels, remote queues).

A process, say on queue manager **QMR1A1**, publishes to a locally defined remote queue pointing to a stream on child 1. The broker on child 1 sees the publication and publishes it to the other brokers in the broker network, including child 2. A remote queue is defined on child 2 that resolves to a local queue on **QMR2A1**. The process on QMR2A1 specified this remote queue when it subscribed to the publication’s topic. The broker on child 2 puts the publication to this remote queue on child 2 and the message winds up in a local queue on **QMR2A1**, which is visible to the subscribing process.

This approach can also be used to *publish* to a clustered queue. A process in **RSA1** can specify as its subscription queue the name of a clustered queue. When a publication is received, the messages are distributed among the clustered queues by a cluster workload exit.

In the deployment of queue managers in support of the test components, the parent and child 1-queue managers will co-reside on the same physical machine as the queue manager **QMR1A1**.

When a message is published, it is published under a given topic on a given stream. To simplify administration of the *publish/subscribe* network, it is recommended that separate streams be defined for each major location or application that will be using the message broker network. This will allow configuration of the various streams, for example, for performance and security.

5.5.3.1.1 Utility Components for MQAA

Lockheed has developed two Component Broker utility components which extend the basic functionality provided by the MQAA. These components are in a module called **TMMessage** and the components are called **TMOutbound** and **TMInbound**, used respectively for sending outbound and receiving inbound messages. The requisitioning component and the Base and Enterprise PDC components use these utility components to communicate with MQSeries.

5.5.4 Messaging from within the Application Server

The following approach is used to communicate between MQSeries applications and Component Broker CORBA components.

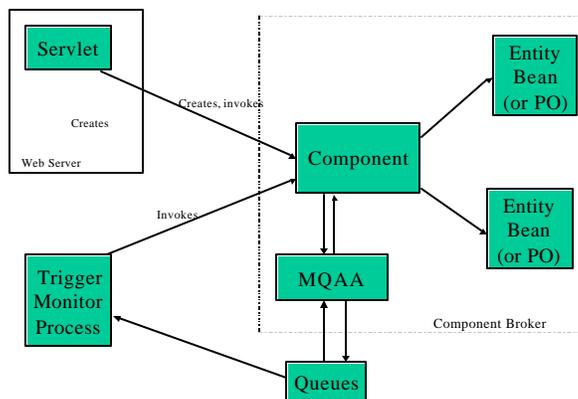


Figure 65: Model of Communication Between MQSeries Applications and Component Broker CORBA Components

See also Section 5.3, “Communications between layers – Horizontal,” of this document for more information on messaging from within the application server.

5.5.4.1.1 Initialization

Trigger monitor process is started and it initializes any *publish/subscribe* subscriptions for the application (optional). Then it blocks on a read on an initiation queue. When a message arrives the trigger monitor will look up or create a Session Bean and call the *messageReady()* method on the application component.

5.5.4.1.2 Messages coming from an input queue

Asynchronous case

In the asynchronous case there is a separate process, the trigger monitor, listening on the initiation queue of the (triggered) input queue. When a message appears on the input queue (and hence on the initiation queue), the trigger monitor calls the *messageReady()* method of the application component, passing as an argument the name of the input queue. The component uses a **TMInbound** object to pull the message off the input queue. The component then processes and optionally replies to the message.

Synchronous case

In the synchronous case, the component effectively blocks an MQAA get (using the **TMInbound** object) on the input queue. When a message arrives on the queue, the get succeeds and the component processes and optionally replies to the message.

Note that any synchronous use of MQSeries shall take into account the possibility that the synchronous call, e.g. a CORBA call, will time out before a response to the message is received via MQSeries. This could occur, for example, if the responding MQSeries application is not running. In this case the request messages would accumulate on the responder program's input queue; they would not be answered in a timely fashion.

The Integration Framework test components use a trigger monitor, PDC_TM.exe, to monitor several input queues for the Enterprise and Base PDCs. In addition to the basic trigger monitor functionality (read a message, invoke **messageReady()**), this trigger monitor contains initialization logic specific to the PDC applications. In particular, it registers the Enterprise PDC with the *publish/subscribe* broker as a publisher of information, and it subscribes the Base PDCs to the topic on which the Enterprise PDC will publish.

5.5.4.1.3 Outgoing messages

The component uses a **TMOutbound** object to put the message on to the desired destination queue, or to publish the message under the desired topic.

5.5.5 Messaging from an Application outside the Application Server

The approach for messaging outside the application server is to have the application use the AMI interface to MQSeries either directly or through the **AMIHelper** Java classes provided as part of the Integration Framework. The AMI interface is complete and relatively easy to use. IBM provides several sample programs with the AMI distribution. The AMI Helper classes provided by Lockheed further encapsulate the AMI interface, reducing even more the development effort required for an application outside the application server to send a message. The File Wrapper test application component provided with the Infrastructure Framework gives an example of how to use some of the **AMIHelper** classes, and **Javadoc** documentation for the AMI Helper classes is provided with the Infrastructure Framework.

5.6 USING THE INTEGRATION FRAMEWORK LOG FACILITY

The Integration Framework (IF) 2.0 Log Facility is implemented using the **log4j** package. The **log4j** package is a message logging utility written in Java and distributed under the **IBM Public License Version 1.0**. The **log4j** package is delivered with the IF and it is maintained at <http://jakarta.apache.org/log4j/doc/index.html>. IF 2.0 Internal components and IF 2.0 Test applications use the **log4j** package to write prioritized messages to the operator's console and a log file. Any application installed on the IF 2.0 which is written in Java can use the **log4j** package to log messages. For applications written in C/C++, the IF provides a C/C++ interface to **log4j** incorporating the Java Native Interface (JNI).

5.6.1 Log4j Version 0.8.5b Package Overview

The **log4j** package provides a simple yet flexible mechanism to output log statements to a file, a **java.io.Writer**, a **java.io.OutputStream** or a remote Unix **syslog** daemon. Table 26: Log4j Packages on page 217 depicts the packages comprising **log4j**.

Table 26: Log4j Packages

org.log4j	The main package of log4j .
org.log4j.examples	Example usage of log4j including source code.
org.log4j.helpers	This package is used internally.
org.log4j.net	Package for remote logging.
org.log4j.nt	Package for NT event logging.
org.log4j.performance	Package to measure the performance of the different log4j components.
org.log4j.xml	XML based configurators.
org.log4j.xml.examples	Example usage of log4j with XML (including source code).

Inserting log statements into code is a low-tech method for debugging it. It may also be the only way because debuggers are not always available or applicable. This is usually the case for multi-threaded applications and distributed applications at large. As Brian W. Kernigan and Rob Pike put it in their book, **The Practice of Programming**:

As personal choice, we tend not to use debuggers beyond getting a stack trace or the value of a variable or two. One reason is that it is easy to get lost in details of complicated data structures and control flow; we find stepping through a program less productive than thinking harder and adding output statements and self-checking code at critical places. Clicking over statements takes longer than scanning the output of judiciously-placed displays. It takes less time to decide where to put print statements than to single-step to the critical section of code, even assuming we know where that

*is. More important, debugging statements stay with the program;
debugging sessions are transient.*

In case of problems with an application, it is helpful to enable logging so that the problem can be located. With **log4j** it is possible to enable logging at runtime without modifying the application binary. The **log4j** package is designed so that log statements can remain in *shipped* code without incurring a high performance cost. It follows that the speed of logging (or rather not logging) is capital. When logging is turned off, log statements of this package have a computational cost ranging from 30 nanoseconds to a few microseconds. Optimization is done by avoiding argument construction by calling *Category.isDebugEnabled()* method before making the logging request.

5.6.1.1 Prerequisites

- **Log4j** is JDK 1.1.x and 1.2 compatible.
- The **TextPaneAppender** requires Swing.
- The **org.log4j.xml** package requires an XML parser.

5.6.1.2 Using log4j

Using the **log4j** package the developer can determine at *runtime* which log statements will be printed on a file or an **OutputStream** of their choice. The decision is based on the *priority* and *category* of the log statement. **Log4j** accommodates all 8 priority levels of UNIX **syslog**. However, four priority levels are preferred. These priorities are **ERROR**, **WARN**, **INFO** and **DEBUG** listed in decreasing order of severity. These are all defined in the **Priority** class. The **Category** class defines a set of methods **debug**, **error**, which specify the priority of a log statement. For example, for some **Category** instance **cat** the log statement **cat.warn(...)** has the **WARN** priority. These methods are used to selectively print log statements. They come in two flavors, the first admitting a message **String** and the second admitting a **Throwable** parameter in addition to the message string. As the name indicates, the **message** parameter is the message to be printed. The category settings are used to determine *at runtime* whether to print a log statement or not.

5.6.1.3 Category Hierarchies

One of the distinctive features of **log4j** its hierarchical categories and their evaluation. Categories are defined by their names. A category may also be assigned a priority but not necessarily. We say that category named "X" is higher ranking than a category named "X.Y". However, the category named "X" bears no relation to categories "x" or "XY" or "z". More formally, a category *C* is said to be a parent of *D*, if *C*'s name followed by a dot, is a prefix of *D*'s name (using case-sensitive comparison). At the base of the category hierarchy is the "root" category, which always exists and is assigned a default priority.

Rule 1

The *chained priority* for a given category *C*, is equal to the first non-null priority in the category hierarchy, starting at *C*.

According to Rule 1, the chained priority of a category named "X.Y.Z", is the priority assigned to category named "X.Y.Z". If however "X.Y.Z" is not assigned a priority, then the chained priority will be the priority of the category named "X.Y". If category named "X.Y" is undefined or is not assigned a priority, then the chained priority of "X.Y.Z" will be the priority of the category named "X". If "X" is undefined or not assigned a priority, "X.Y.Z" will assume the priority of root, which is always defined.

Table 27: Chained Priorities Examples below uses four examples to depict various assigned priority values and their resulting chained priorities according to Rule 1.

Table 27: Chained Priorities Examples

Example 1			Example 2		
Category name	Assigned priority	Chained priority	Category name	Assigned priority	Chained priority
Root	Proot	Proot	Root	Proot	Proot
X	none	Proot	X	Px	Px
X.Y	none	Proot	X.Y	Pxy	Pxy
X.Y.Z	none	Proot	X.Y.Z	Pxyz	Pxyz
Example 3			Example 4		
Category name	Assigned priority	Chained priority	Category name	Assigned priority	Chained priority
Root	Proot	Proot	Root	Proot	Proot
X	Px	Px	X	Px	Px
X.Y	none	Px	X.Y	none	Px
X.Y.Z	none	Proot	X.Y.Z	Pxyz	Pxyz

The root category can be retrieved with the *getRoot* method. The *Category.getInstance* method is used to retrieve a given *Category* instance. This method takes the name of the desired category as a parameter. The user does not have to worry about the order of definition of categories. When instantiated a child category will link itself to its nearest parent. Similarly, when instantiated a parent category will link itself to all its existing children categories. By the way, the linking is fast, even for hundreds of categories.

Rule 2

A log statement of priority *p* in a category with chained priority *q*, will be printed if $p \geq q$.

Rule 2 uses the ordering relation: **ERROR, WARN, INFO, DEBUG**.

In summary, the category hierarchy allows the programmer to enable (or disable) logging at a high level and disable (or enable) logging at a lower level. One could imagine more elaborate

mechanisms of selection but hierarchical categories (with some thinking) are usually expressive enough.

The **log4j** package is designed so that categories are instantiated and used (but not configured) at compile-time. They are usually assigned priorities at run-time by reading a configuration file.

5.6.1.4 Log Output

The log output can be customized in many ways. Moreover, one can completely override the output format by implementing one's own `Layout`. Here is an example output using `PatternLayout` with the conversion pattern `"%r [%t] %-5p %c{2} %x - %m\n"`

```
176 [main] INFO  examples.Sort - Populating an array of 2 elements in reverse order.
225 [main] INFO  examples.SortAlgo - Entered the sort method.
262 [main] DEBUG SortAlgo. OUTER i=1 - Outer loop.
276 [main] DEBUG SortAlgo.SWAP i=1 j=0 - Swapping intArray[0] = 1 and intArray[1] = 0
290 [main] DEBUG SortAlgo. OUTER i=0 - Outer loop.
304 [main] INFO  SortAlgo.DUMP - Dump of interger array:
317 [main] INFO  SortAlgo.DUMP - Element [0] = 0
331 [main] INFO  SortAlgo.DUMP - Element [1] = 1
343 [main] INFO  examples.Sort - The next log statement should be an error message.
346 [main] ERROR SortAlgo.DUMP - Tried to dump an uninitialized array.
      at org.log4j.examples.SortAlgo.dump(SortAlgo.java:58)
      at org.log4j.examples.Sort.main(Sort.java:64)
467 [main] INFO  examples.Sort - Exiting main method.
```

Figure 66: Example of Log Output Code Using `PatternLayout`

The first field is the number of milliseconds elapsed since the start of the program. The second field is the thread outputting the log statement. The third field is the priority of the log statement. The fourth field is the rightmost two components of the category making the log request. The fifth field (just before the '-') is the *nested diagnostic context* (NDC). Note the nested diagnostic context may be empty as in the first two statements. The text after the '-' is the message of the statement.

5.6.1.5 Usage Example

Figure 67: Compile-Time Directive Usage Example Figure 72 uses compile-time directives to assign priorities to categories. The preferred way is reading these directives from a configuration file during application initialization. See **PropertyConfigurator.configure** and **DOMConfigurator** for possible configuration file formats.

```
Public static void main(String[] args) {  
    // Assigning priorities to categories. This is typically done at  
    // initialization time  
    Category a = Category.getInstance("a");  
    a.setPriority(Priority.DEBUG);  
    Category x = Category.getInstance("x");  
    x.setPriority(Priority.WARN);  
    ... other code  
}
```

Figure 67: Compile-Time Directive Usage Example

Figure 68 is code that would normally be executed after the code in Figure 67: Compile-Time Directive Usage Example. The accompanying comments explain the function of the code. Before using **log4j**, the reader should make sure to understand this example.

```
{  
    // Variable a will automatically refer to instance defined above.  
    Category a = Category.getInstance("a");  
  
    // Variable x will automatically refer to instance defined above.  
    Category x = Category.getInstance("x");  
  
    // A new category instance will be created. This instance will not  
    // be assigned a priority.  
    Category x_y = Category.getInstance("x.y");  
  
    // This statement will print, because WARN = a = DEBUG.  
    a.warn("RFC 724 defines the IP protocol.");  
  
    // This statement will print, because DEBUG = a = DEBUG.  
    a.debug("The value of variable I is " + I);  
  
    // This statement will not print, because DEBUG not print,  
    // because INFO = x = WARN.  
    x.warn("Looking at a screen for too long may hurt your eyes.");  
  
    // This statement will print, because ERROR = x = WARN.  
    x.error("Can't have the cake and it it too!");  
  
    // This statement will not print, because x_y is not assigned a  
    // priority x_y will assume the priority of x which is WARN.  
    X_y.debug("x is the parent of x_y");  
}
```

Figure 68: Example Code of Compile-Time Follow up

5.6.1.6 Initialization

Log behavior is usually configured during program initialization. Use the configuration methods in **BasicConfigurator**, **PropertyConfigurator** or **DOMConfigurator** to configure **log4j** parameters from a configuration file. For shipped code, call *BasicConfigurator.disableInfo* method during program initialization to disable logging messages that are of priority **INFO** or **DEBUG**.

5.6.1.7 Format of Log Output

The format of log output depends on the layout instance assigned to the appender in use. At present time, **log4j** comes with a powerful layout called the **PatternLayout** that can be configured using a conversion pattern.

5.6.1.8 Multiple Appenders

Log4j categories can have multiple appenders. The *addAppender* method adds an appender to a given category. **The appenders follow the category hierarchy**. In other words, the appenders of the child are added to the appenders of the parents. One can override this behavior by **setting the additivity flag to false**.

Rule 3

The output of a log statement of category *C* will go to all the appenders in *C* and its parents. This is the meaning of the term "appender additivity".

However, if a parent category of *C*, say *P*, has the additivity flag set to *false*, then *C*'s output will be directed to all the appenders in *C* and its parents up to and including *P* but not the appenders in of any of the parents above *P*.

Categories have their additivity flag set to *true* by default.

Table 28 shows an example illustrating the function of multiple appenders.

Table 28: Multiple Appender Example

Category Name	Added Appendors	Additivity Flag	Output Targets	Comment
Root	A1	not applicable	A1	The root category is anonymous but can be accessed with the Category.getRoot() method.
x	A-x1, A-x2	true	A1, A-x1, A-x2	Appendors in root are added to appenders in "x".
x.y	none	true	A1, A-x1, A-x2	Appendors of "x" and root.
x.y.z	A-xyz1	true	A1, A-x1, A-x2, A-xyz1	Appendors in "x.y.z", "x" and root.
Security	A-sec	false	A-sec	Only appendors of "security" due to false setting of the additivity flag.
Security.access	none	true	A-sec	Only appendors of "security" due to false setting of the additivity flag in "security".

5.6.2 Integration Framework 2.0 Implementation Guidelines

5.6.2.1 Installation

The Integration Framework Version 2.0 is using **log4j** Version 0.8.5b. This version of the **log4j** package is delivered as **log4j-v0.8.5b.zip**. To install, unzip this file into the `INSTALL_DIR` directory creating `INSTALL_DIR\log4j-v0.8.5b`.

The `INSTALL_DIR` directory can be any directory the developer chooses. For example, during IF development, `INSTALL_DIR` was `c:\h\IFSServices\lib\log4j` and the **log4j** directory was `c:\h\IFSServices\lib\log4j\log4j-v08.5b`.

To use the **log4j** package, add the **log4j.jar** file, located in the `log4j-v0.8.5b` directory, to the Java application's classpath. Refer to the `INSTALL_DIR\log4j-v0.8.5b\javadoc` directory for complete documentation on how to implement the **log4j** package. Browsing `INSTALL_DIR\log4j-v0.8.5b\javadoc\overview-summary.html` is a good place to start

5.6.2.2 Category Names

There will be a minimum of one **log4j** message category per class. The name of these message categories will be set to the class name. The fully qualified name of a class in a static block for class X can be retrieved with the statement `X.class.getName()`. Note that X is the class name and

not an instance. The **X.class** statement does *not* create a new instance of class X. Here is the suggested usage template assigning the name of the **Foo** class to the **cat** category.

```
package a.b.c;

public class Foo {
    static Category cat = Category.getInstance(Foo.class.getName());
    ... other code
}
```

Figure 69: Example Code for Assigning Foo Class to the cat Category

Message sub-categories may be created. To create a sub-category name, simply append a unique identifier string to the parent category name. An example would be to create a sub-category for the method M1 in the class **Foo**.

```
static Category cat = Category.getInstance(Foo.class.getName()+"_M1");
```

Figure 70: Example Code to Create a Sub-Category in the Class Foo

5.6.2.3 Message Priorities

Four message priorities are allowed: **ERROR**, **WARNING**, **INFO**, **DEBUG**.

Table 29: Message Priorities

ERROR	A resource has failed and is currently not operational or a resource is near failure
WARNING	A resource is in a cautious condition.
INFO	A resource has performed normal activity.
DEBUG	Used by the application programmer to isolate programming errors.

ERROR and **WARNING** priority messages will be enabled when IF Version 2.0 is delivered, while **INFO** and **DEBUG** priority messages will be disabled.

5.6.2.4 Configuration File

All IF 2.0 internal components and test applications will use a single Configuration File per application. Key entries in the Configuration File are used by the application to configure where the messages are logged to and which priority levels are enabled for logging.

The default **log4j** configuration file used by the IF Version 2.0 is described in Section 5.6.2.5.

5.6.2.5 Default log4j Configuration File

The following is the default **log4j** Configuration File used by the IF 2.0. The comments in the file describe how each of the various keys is used to configure the **log4j** logging environment.

5.6.2.6 Log Statement Format

As discussed in Section 5.6.2.5, the default IF 2.0 log output will be to the console and to a log file. Prior to delivery of the IF to the customer, the Console Appender A1 will be removed. A complete log entry shall not exceed 256 characters.

Below are examples of the log output for the Appender A2 and A1 format strings. Highlighting is used to identify each field of the conversion pattern and the log entry.

```
Console Appender: A1

ConversionPattern: %-6r [%12.12t] %-5p %30.30c (%x) - %m\n

Log Entry:

15 [main] DEBUG PDCSessionModuleBO.PDCSessionAOBOBase (HostName=svr1) - Entering
PDCSessionAO-ENTERPRISE::messageReady
```

Figure 71: Example Code for Log Output Appender A1

The first field, **2000-09-07 15:23:51,291**, is the **ISO8601** date and time stamp. The second field, **[main]**, is the thread outputting the log statement enclosed in brackets. The third field, **DEBUG**, is the priority of the log statement. The fourth field, **PDCSessionModuleBO.PDCSessionAOBOBase**, is the rightmost thirty-five characters of the category making the log request. The text after the “-” is the user specified message of the statement. The last format field in the conversion pattern, the “/n”, generates a new line.

```
File Appender: A2

ConversionPattern: %-6d{ISO8601} [%12.12t] %-5p %35.35c - %m\n

Log Entry:

2000-09-07 15:23:51,291 [main] DEBUG PDCSessionModuleBO.PDCSessionAOBOBase - Entering
PDCSessionAO: ENTERPRISE::messageReady
```

Figure 72: Example Code for Log Output Appender A2

The first field, '15', is the number of milliseconds elapsed since the start of the program. The second field, '[main]', is the thread outputting the log statement enclosed in brackets. The third

field, **DEBUG** , is the priority of the log statement. The fourth field, **PDCSessionModuleBO.PDCSessionAOBOBase**, is the rightmost two components of the category making the log request. The fifth field, (**HostName=svr1**), (just before the “-“) and enclosed in parentheses is the *nested diagnostic context* (NDC). Note the nested diagnostic context may be empty. The text after the “-“ is the user specified message of the statement. The last format field in the conversion pattern, the “/n”, generates a new line.

5.6.2.7 Example log4j Program

The following is an example of an implementation of the **log4j** Package. Comments in the example describe the actions being performed.

5.6.3 A Note Regarding Tivoli Compatibility

All log files created using the **log4j** package are intended to be processed by a Tivoli Log File Adapter (LFA) and presented as an event to an operator on a Tivoli Enterprise Console (TEC). It is expected that these LFAs will be implemented by the Defense Information Systems Agency (DISA). Since a Tivoli LFA can only process log files which contain single line log messages, the application developer is cautioned to ensure that the length of all log file messages are less than 256 characters. Included in the 256 characters are the log message fields automatically included by **log4j** and the application developer's message string.

Single messages processed by the TEC can be assigned severity levels. The most severe messages can represent alerts and alarms that once displayed on the TEC monitor, will cause either a manual response from an operator or an automated response from the TEC. Message priority assignment will be implemented by DISA with proper guidance from the application developers.

Some alerts and alarms may be represented by a sequence of related events, with each event generating a log message to the TEC. An example of this might be a sequence of invalid login attempts within a specified time period. Rules can be generated which allow message sequences to be processed by the TEC using its powerful correlation engine and handled as if they were a single event.

History logs maintained by the TEC can later be processed into user friendly reports highlighting any anomalous behavior that may have occurred during the time period encompassing the report. Please refer to the Tivoli Documentation Suite for a further discussion of Tivoli Log File Adapters and the Tivoli Enterprise Console.

6. Securing the Application

6.1 Overview of IF Security

The IF provides technical security components that should satisfy most requirements for security, for MAs that are GCSS-AF-compliant. This sub-section provides an overview of these components and assistance in determining their applicability to a given Mission Application's needs.

The MA Security Engineer

Important: MA teams DO need to have a Security Engineer. Although the IF supplies widely useful components and the GCSS-AF Integrator can assist in understanding how these components are used, each MA team still needs someone who can:

1. Determine and analyze security requirements specific to the MAs domain (this may include a formal or informal vulnerability assessment).
2. Decide on behalf of the MA which of those requirements the IF components satisfy (in concert with the GCSS-AF Integrator).
3. Determine an approach for satisfying any MA security requirements the IF components cannot satisfy.
4. Derive the data required to support the MAs use of IF security components.
5. Lead the implementation of the MA security design, including use of APIs and interfaces to IF security components, and any non-IF security implementation (if this includes any additional COTS components, additional firewall considerations may need to be worked with USAF and DISA).
6. Coordinate security testing with the GCSS-AF Integrator and the USAF.

The MA security engineer need not necessarily be an information security specialist. The person chosen for this position should, however, have a background that includes sufficient familiarity with information security technical topics to allow the individual to understand security requirements and to coordinate with the GCSS-AF Integrator and the USAF in discussing both requirements and technical solutions. They should have good communication skills, and the ability to self-educate themselves to fill in their security knowledge as required.

Security From Browser to EJB/CORBA Component

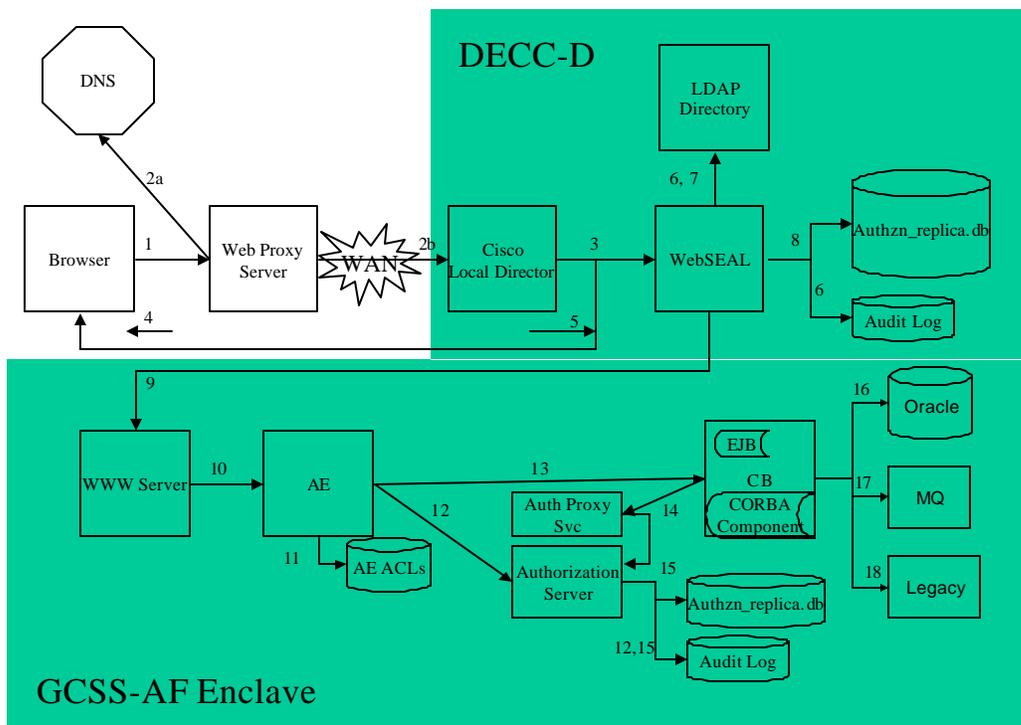


Figure 73: Operational Diagram: Authentication & Authorization

The interaction described in the following series of scenarios is shown in Figure 73: Operational Diagram: Authentication & Authorization:

(Note: The boxes in the diagram should be interpreted as functional rather than physical entities. The colored areas indicate nominal physical locations. For IF v2.0, all servers are presumed to be located in a single DISA DECC-D, and thus are protected by the DECC-Ds' perimeter defenses (firewall, etc.), although these elements are not shown on the diagram. The GCSS-AF Enclave includes all shaded areas in the diagram above, and is protected by a separate security "policy" at the firewall and routers.)

1. The Web Browser requests a URL (e.g. <https://www.gcssaf.rsa1.disa.mil/global/apps/menu>) The browser sends the request to their local Web Proxy Server (assuming one is in use, otherwise Proxy's activity is performed by Web Browser).
2. The Web Proxy Server performs a DNS lookup to www.gcssaf.rsa1.disa.mil and gets an IP address of the Cisco Local Director. The request travels encrypted (HTTPS) over NIPRNET through the DECC-Ds external firewall to a Cisco Local Director in the Demilitarized Zone (DMZ) at the DECC-D.

3. The Cisco Local Director forwards (routes) the request to one of N identically configured WebSEAL servers. Local Director also maintains state information based on the SSL session-id to forward additional requests from the same browser to the same WebSEAL server and vice versa. (It is necessary to always have traffic flowing between the Browser and the same WebSEAL server once a session has been established. Otherwise the user would be forced to login again.) Note: All future traffic between WebSEAL and the Browser will be routed through the Cisco Local Director and the user's proxy server.
4. The WebSEAL server will determine whether a session has already been established with this Browser. If not, then during SSL negotiations, the Browser will prompt the user to send a client certificate. The SSL negotiation process will validate the client's certificate. If the SSL negotiation is successful and the user's certificate is valid, WebSEAL does an LDAP search of the Directory to find an object that matches the Distinguished Name of the client certificate. If it finds it in the LDAP Directory, it looks for a Distinguished Name attribute that identifies the Policy Director user that the certificate maps to. If the Distinguished Name in the attribute is a valid Policy Director user, then the user is considered authenticated. Skip to **Step 7** acquiring the credential.
5. If the user does not submit a client certificate, WebSEAL will send a login form (login.html or pkmslogin.html) to the Browser instead of the user's initial URL request. The user will enter the user's userID (a.k.a. principal) and password in the form on the Browser and click the Login button. The Browser will post this information, encrypted over the network via HTTPS (SSL) to WebSEAL.
6. WebSEAL will perform an LDAP search for the *userID* (principal) as a Distinguished Name (dn) in the LDAP directory. Principal is an attribute of an **InetOrgPerson** in the LDAP directory. When it finds a match, it compares the *password* supplied in the form with the *password* attribute of the found **InetOrgPerson** object. (Technically, it performs an LDAP Bind to accomplish this match – if the bind operation is successful, it indicates the *userID* was found and the *password* matched.)
7. After a successful authentication, WebSEAL builds the credentials for the user (including the **uuid** of the user, **uuids** of groups to which the user belongs, and other material) and other material (string versions of principal and group names) and caches this material. Caching keeps WebSEAL from the time consuming task of building the credential with each request. WebSEAL maps the SSL **sessionid** with this credential, and some timestamp information. For future Browser gets and posts, WebSEAL uses the SSL **sessionid** as proof of prior authentication if it finds a match in its cache table. WebSEAL checks the time stamp information against timeout parameters, including a session timeout and a user inactivity timeout. These timeouts are configurable by the administrator. (IF defaults for these are 4 hours, and 30 minutes, respectively). It forces a new authentication (i.e. a new **pkmslogin** form) if any one of them is expired. With *PKI Certificate* based authentication, it will prompt the user to re-select a client certificate. The capabilities and configuration of the browser will determine whether the browser will prompt the user for a password to the certificate database again or whether the user will see a prompt at all if they only have one certificate.

8. After a successful authentication, WebSEAL checks the user's authority to access the original URL request using the user's credential against the replicated authorization policy database.

If the Audit permission on the ACL for this object was turned on, then the success or failure of the access check is logged on the WebSEAL server.

If the user fails the authorization check, a configurable HTML file indicating the user was not authorized to access the URL.

9. If the user passes the authorization check, WebSEAL forwards the user's original URL request to the backend system. WebSEAL authenticates to the backend Web Server using mutual SSL authentication. The certificate used in the SSL communication to the backend Web Server is the WebSEAL server certificate. The Web Server uses the certificate used in the SSL communication to authenticate the user. No *password* is required.
10. WAS-AE is passed a WebSEAL *userID* and *password* in the BA Header and uses it to establish a WAS-AE session; this is a separate "session" from the user session to WebSEAL.²⁴ (The use of a *userID* and *password* for this session is necessitated by WAS AE 3.02s lack of support for certificate-based mutual authentication.²⁵) This occurs for the initial connection with every WebSEAL box.

Subsequent requests, even if they come from a different User/Browser, benefit from this by reusing this already authenticated session. (Note: The connection has configurable timeouts, which force re-authentication if one of them is exceeded, as explained in **Step - 7.**)

Limiting the number of times authentication takes place increases performance. The GCSS-AF integration framework will limit management of Air Force *userIDs* and *passwords* to the front-end Policy Director WebSEAL sessions and backend Legacy systems.

Managing separate *userIDs* and *passwords* for individual GCSS-AF users on the backend systems (WebSphere Application Server Advanced Edition and WebSphere Application Server Enterprise Edition), is not scalable and would potentially create additional security holes by never changing backend *passwords*. In addition, the Servlets will check the individual user's authorization, not WebSEAL's authorization, by making explicit

²⁴ (In a future release, the IF will support the use of Servlet Redirectors or a similar mechanism to facilitate the distribution of WAS-AE function separately from Web Server function. The details will be dependent on the specific COTS capabilities, especially at an enterprise level. Until then, the IF will require the Web Server and AE to be co-hosted.)

²⁵ WAS 3.5 is supposed to provide this feature, which will free up the BA Header to support single-sign-on (SSO), see step 17 and 18.

authorization checks to Policy Director using the user's credential passed in the HTTP header.

The Web Server passes the requested URL back to the browser. This includes the GCSS-AF Main Menu, filtered to present only the subset of items that this user is authorized to access.

The user can select an action from the menu. This request is processed according to **Steps 1-4**, keeping in mind that the user is already authenticated, and that Local Director and WebSEAL already have established a connection. Each menu choice is associated with a Servlet, so the menu choice is passed to the Servlet engine (WAS AE) for processing. (Actually, the menu choices can be associated with a Servlet, JSP, or HTML file; for purposes of this discussion, we use a Servlet, because this provides the easiest way of showing the full range of security paths.)

11. The Servlet engine checks to see whether WebSEAL has authorization to the Servlet. If so, then it invokes the Servlet. This check uses a WebSEAL *user ID* and *password* that are passed in the BA Header of the HTTP request.
12. The Servlet extracts the credential information supplied by WebSEAL in the HTTP header from the **HTTP_IV_CREDS** parameter. The Servlet can check²⁶ the permissions of the credential to perform a specific action by performing an authorization check to a Policy Director (PD) Authorization Server using encrypted Distributed Computing Environment (DCE) Remote Procedure Calls (RPC). First it checks whether the user can invoke the Servlet itself, and then it checks against any specific methods that the Servlet invokes that require separate authorization decisions.

The PD Authorization Server checks the credential and the action requested against the Access Control List (ACL) of the object attempting to be accessed and the PD Authorization Server sends a decision back to the Servlet.

If the user is not authorized, the Servlet could return an application specific error message or a generic page that would be created for unauthorized access.

If the user has access to invoke the Servlet and the specific method, in this case assume an Enterprise JavaBean or CORBA Object and method, the Servlet programmatically logs in²⁷ (authenticates) to the WebSphere Application Server Enterprise Edition (WAS-EE), a.k.a. Component Broker (CB), via a Servlet Login Helper. To do this, the Servlet

²⁶ The Servlet is not required to perform an access check, since the menu filtering presents the user with only those choices they are authorized for. However, it is good security practice to repeat the check at this point, because the traffic from browser to servers could have been tampered with after the menu filtering was performed.

²⁷ Actually, it checks to see if there is already a user session in place for this user. If so, it verifies that the credentials for that session are still valid. If either check fails, then it logs in; for this narrative, we are presuming this is the first time through, so a session is not already established, thus a login is needed.

first establishes an SSL session to Component Broker (using a client keyring on the AE side and a server keyring on the CB side). Component Broker takes the *userID* and *password* supplied by the Servlet Login Helper and logs in to the Component Broker's DCE Cell.

If the user does not successfully authenticate, the Servlet sends a page to the user indicating a system error and to retry their request. If the **Servlet Login Helper** successfully authenticates to CB (DCE), then the Servlet invokes the session AO of the requested object. The Servlet passes the credential supplied in the HTTP Header by WebSEAL as a parameter to the methods invoked in the object (using the **SecInfoStruct** data structure, assisted by the **IFSServices** helper classes; this includes passing the **BaseNameQualifier** and **BaseNameContext** derived from the menu system).

13. The EJB/CORBA object checks the permissions in the credential to access the EJB/CORBA object's method by performing an authorization check to a Policy Director (PD) Authorization Server Proxy using SSL RMI over IIOP.
14. The PD Authorization Server Proxy makes the calls to the PD Authorization Server on behalf of the EJB/CORBA object using DCE RPC. It is necessary to use the PD Authorization Server Proxy because the PD servers are in a different DCE cell than CB servers. A hierarchical DCE cell allowing intercell communication could potentially be created, but PD and CB are moving away from DCE in their future releases, so the effort to manage hierarchical cells would be thrown away.

The PD Authorization Server checks the credential and the action requested against the Access Control List (ACL) of the object attempting to be accessed and the PD Authorization Server sends a decision back to the PD Authorization Server Proxy and on to the requesting EJB/CORBA object.

If the user is not authorized, the EJB could return an error code to the Servlet indicating that the user was not authorized to perform the requested function. The Servlet could return an application specific error message or a generic page that would be created for unauthorized access.

If the user is authorized, the EJB performs the action.

15. The EJB/CORBA object may need to use Message-Oriented Middleware (MOM), specifically the MQSeries product, to communicate to another application hosted at a different DECC-D. This communication includes the use of a DISA security "exit" and VPN to provide nominal encryption and authentication on the messages.²⁸

²⁸ IBM/Tivoli is preparing a Policy Director module ("Policy Director for Messaging") specifically to secure MQSeries traffic. The IF will include this product when it is released and integrated, at which time the DISA exit and VPN will be phased out.

16. The EJB/CORBA object may have persistence in an Oracle database. The connection to the Oracle database is managed by the container and uses a container supplied *userID* and *password*. Individual *userIDs* and *passwords* to the backend database are not possible with the current configuration of PD and CB software.
17. The Servlet may connect to a backend database that requires a *userID* and *password*. PD WebSEAL will eventually be able to supply uniquely defined *userID* and *password* for that resource in the BA Header to be used to log in to the backend database. PD WebSEAL can only pass a single set of *userID* and *password* parameters in the BA header for supplying *userID* and *password* information to backend systems.²⁹
18. The Servlet or EJB may connect to a backend Legacy system that requires a *userID* and *password*. (See **Step - 17** for discussion.)

Design Note Regarding Legacy Systems and Databases

There are basically two options for connecting Legacy systems to the IF front-end that get around the problem mentioned in **Step - 17**, to provide for separate user authentication to those Legacy systems (each option has some possible variants for implementation) before the IF is updated to include WebSphere v3.5:

Come through WebSEAL, then go through a set of AE and CB servers dedicated to interfacing with this Legacy system. This allows for separate *userID*'s but all shall have the same *password*.

1. Come through WebSEAL but then pass control to a non-IF-based backend. By not using AE, with its current restriction on mutual authentication, this frees the BA Header to pass a **Legacy ID** and *password* via GSO. (WebSEAL can theoretically be junctioned to different backend servers, although we have not tested this.)
2. In either case, the *userID/password* would be stored in the LDAP repository and retrieved by WebSEAL when needed. The details of implementing this scheme (including any schema changes to the LDAP directory) have yet to be worked out.

6.1.1 Security Requirements

The security requirements for the IF are derived from the *GCSS-AF System Security Policy 15 December 1999*. This document “establishes security policy for the Global Combat Support System - Air Force (GCSS-AF), in accordance with AFSSI 5024. It identifies policies and procedures that will minimize the risk of operating GCSS-AF in the System High Security Mode at the *Sensitive Unclassified* and *SECRET* levels.” (Executive Summary) The derived individual

²⁹ IF 2.0 cannot provide this capability since WAS AE 3.02 did not support direct mutual authentication via certificates; see step 10. Hence the BA header had to be used to pass *ID/password* with AE. WAS 3.5 supposedly provides for mutual authentication, hence the BA header will be freed up for database signon info in a future IF release.

security requirements are documented in the GCSS-AF Security Sub-System Specification. This sub-section provides an overview of the IF security requirements to assist MA teams in determining which of their security requirements are already dealt with by the IF. For the individual security requirements that are summarized here, see the *GCSS-AF Security Sub-System Specification*.

Overview

The most basic summary of the requirements the IF security components are intended to meet is the “Orange Book” (*DoD 5200.28-STD, Trusted Computer System Evaluation Criteria* (TCSEC)) **C2 level** of security. This specifies a level of “discretionary” access control that the Government considers sufficient for securing systems that do not handle data above Secret classification. (The Orange Book is supplemented by the “Red Book”, which provides guidelines for implementing security in a networked environment.) The IF is expected to support operation at the unclassified sensitive, system high mode, as a minimum. It is also required to support systems that operate at the Unclassified level, or the Secret level, but not at both levels, nor at levels above Secret.

Note Future Capability: The DoD “Rainbow Books” are being superseded by the international Common Criteria for information security. As of the writing of this Developer’s Guide, specific guidance had not come down for the use of Common Criteria (CC) by the IF. It is expected that this guidance will be forthcoming in the near future, and the IF will convert to the CC in a future release. However, it is expected that this process will not involve major technical modifications to the IF security solution.

An overall IF requirement that affects the decisions made regarding the security solution is that COTS products shall be used wherever possible to satisfy all technical requirements. In the security area, the Tivoli Policy Director suite was selected to provide maximum COTS coverage. In addition, IBMs WebSphere, Tivoli’s ESM products, and Oracle’s ANO all provide elements of security coverage.

The IF security solution is intended to provide security coverage for applications that employ the overall IF architecture. In particular, this means that security shall apply to communications technologies supported by the IF architecture (CORBA and MOM), to database components of the IF, and shall be inter-operable with the IF ESM components.

Authentication Requirements

The IF shall provide strong authentication services enabling human users and system entities to be uniquely identified and to have actions they undertake verified and traceable to them. For human users, this will be a *userID* plus *password* mechanism (with standard limits on *password* choices and administration), with a planned migration to DoD class 3 and ultimately class 4 PKI in future releases. For system entities, this will be a server DoD class 3 PKI certificate mechanism, to be used whenever a communication path is opened within a GCSS-AF session, with a planned migration to DoD class 4 PKI (i.e. smart cards or equivalent) in a future release. For human users, the IF is required to support an enterprise-wide GCSS-AF sign-on through a

web browser. The intention is for all MA, Legacy, and sub-system sign-ons to be handled implicitly by the IF (i.e. to provide “single sign-on” or SSO). Due to COTS limitations, full SSO capability could not be provided in the IF v2.0 release. As the products mature, the IF will provide more complete SSO function.

Access Control Requirements

The IF shall provide a mechanism to control access to GCSS-AF modernized applications and their data. This is to include a role-based discretionary access control methodology, and shall allow access controls to be applied down to the object and method levels. There shall be a mechanism to administratively enable and disable user access, to be applied under various circumstances (e.g. a user is inactive for a specified period of time). The access control mechanism shall provide for distinguishing between “privileged” and non-privileged users for various purposes.

Non-Repudiation Requirements

The IF is to provide general capabilities for both end user and system component digital signatures. These features are not yet provided by the IF v2.0 release. Future IF releases will provide digital signature capabilities.

Confidentiality Requirements

The IF shall provide mechanisms allowing all data transmissions to be encrypted by government-approved means. (However, whether to actually apply encryption on any given channel is a Government decision. Basic USAF direction is that transmissions that stay within the GCSS-AF server enclave need not have encryption applied.)

Note Future Capability: An intrusion detection (IDS) capability for the IF is planned for a future release; for IF v2.0, we rely on the IDS provided by CITS/BIP and DISA, at the site boundary.

Virus checking is handled by DII COE anti-virus package(s). User sign-ons to servers shall be severely restricted, and shall require a strong authentication mechanism. The IF solution shall limit the use of mobile code (in conformance with recent DoD mobile code policy direction).

Integrity Requirements

IF and MA data residing on IF servers is to be protected from malicious or unintentional alteration. The IF is responsible for recommending configuration guidelines for Windows NT, Sun Solaris and HP-UX systems (i.e. the DII COE platforms), for thin clients (including browsers) on Windows NT workstations, and for IF database products. The MA is responsible for recommending configuration guidelines for non-NT workstations, for thick client workstations, for servers not using DII COE platforms, and for non-IF database products that the MA may select.

Audit and Alarm Requirements

The IF is required to maintain an audit trail for all security-relevant actions taken by, or on behalf of, a human user. This requirement includes provision for end-to-end user accountability (i.e. credential delegation). Audit collection and analysis is to be automated, as much as possible.

Note Future Capability: The IF v2.0 release cannot provide full credential delegation, due to COTS limitations, but will supply this capability as the products permit. The same is true of audit automation.

PKI and Key Management Requirements

The IF is expected to be able to use DoD PKI certificates and keys. Issuance and management of the certificates and keys is a Government responsibility.

Note Future Capability: User PKI services will be provided as DoD PKI plans mature; a future IF release will include these services. The IF will inter-operate with DoD PKI certificates (X.509v3), and will be capable of checking Certificate Revocation Lists (CRLs) provided by DoD PKI services.

6.1.2 The ISO Security Model

The basis for the analytical model of security in GCSS-AF is the ISO 10181 (and 7498-2) Security Standard. This standard is the basis for the IF security components, including:

Authentication

This package provides services needed for Identification and Authentication activities. This includes user authentication, component authentication, handshaking involved with establishing communications between software or hardware components, and user session maintenance. The IF supports *PKI Certificate* authentication and the standard *userID/password* mechanism that includes an enterprise logon web page.

Note Future Capability: As DoD PKI plans and implementation mature, a migration of user authentication to a smart card authentication mechanism and CRL checking of certificates is anticipated to be supported by the IF.

Component/server authentication services are provided requiring server-side PKI certificates. Enterprise session management is provided such that a user need only log in once. The IF requires the use of IBM Policy Director product to provide these services.

Access Control

This category encompasses services for controlling access to GCSS-AF objects such as components, databases, individual data objects, etc. It does **not** include objects controlled at the operating system level—GCSS-AF relies upon properly configured Operating System controls at that level. (However, Access Control allows files to be defined as controlled objects in addition to protecting them using OS controls.) Access Control provides services to establish access

control policies and data for the enterprise, to implement those policies during run-time actions of the system, and to administer the user and object data relating to access control policies and decisions. For the current release, implicit access control is provided for GCSS-AF objects on the web servers and Servlet engines, and MQSeries queues and objects.

Note Future Capability: Implicit authorization services for CORBA objects or EJBs are anticipated to be provided in a future release.

Explicit authorization can be implemented by the developer with the current release if required prior to availability of the implicit services. IF authorization services are capable of providing protection at several levels of granularity including application invoked access control down to the attribute level. Application engineers shall determine the appropriate level for their application objects, and to set up the required configuration and policy data accordingly. The IF requires the use of IBM Policy Director product to provide these services.

Non-Repudiation

This category encompasses services that ensure a user cannot deny an action taken by him/her or on his/her behalf, within the system. This includes providing capabilities for creating, verifying, and properly storing digital signatures.

Note Future Capability: User-level digital signatures, and the capability for an application to implicitly sign and verify signatures on Business Object Documents (BODs) or other messages, are anticipated to be provided in a future release of the IF. Full non-repudiation archiving is not provided, and is not currently planned due to lack of COTS capabilities and the immaturity of relevant standards in this area, but may be provided at a later date.

The IF requires the use of IBM Policy Director product to provide these services.

Confidentiality

This category encompasses services that ensure unauthorized individuals cannot view, modify, or delete data, and that proper protection is applied to data and code during both storage and transmission. This package provides encryption and communications security for securing all GCSS-AF-originated messages beginning with the enterprise logon. These provisions include the use of HTTPS, SSL, Secure RPCs, and LDAPS, as provided through Netscape browsers, components of IBM Policy Director, IBM HTTP Server, IBM WebSphere, and Oracle ASO.

Note Future Capability: Multi-Level Security (MLS) is not supported but is expected to be a future requirement. Applications that require MLS in the meantime will have to supplement the IF to satisfy these requirements.

Integrity

Integrity services are those that ensure system elements and data cannot be compromised or modified by illicit actions. The majority of the services and measures, such as boundary protection, required for Integrity is provided by the processing centers and base networks. System oversight and administration services and measures are provided by ESM package and by USAF operations and support. The IF Integrity capability provides only that information needed to tie into these external elements and the IF security product, IBM Policy Director. In addition, the IF requires the proper application of DISA STIGs to the host servers and software that is delineated in the IF guidelines for NT and Unix configuration.

Audit and Alarms

This category encompasses services that record information about activities taken inside the system, whether by human users or software components, and the storage and analysis of those records. This includes creation of audit records, storage of audit records, creation of audit reports, generation of dynamic on-line alarms, and analysis of events and records (whether at runtime or post-mortem). It also provides services to define and administer audit policies, as well as the technical features needed to implement the policies. It also includes intrusion detection, although most of the technical solution for intrusion detection is specified as part of the DII COE and CITS/BIP. The current IF audit provisions are limited to the facilities provided by the supported security product, IBM Policy Director and IF Log Services to record system actions.

Note Future Capability: The current IF does **not** provide audit reduction or alarm posting. Alarm and alert posting is to be provided at the RSA processing centers using the Tivoli Management System capabilities employed by DISA at an RSA. Processing centers that do not utilize this Tivoli capability should plan on implementing the same capability as DISA at an RSA. A future IF release should subsume Tivoli under the Enterprise Systems Management capability. Audit reduction and reporting capabilities are anticipated to be provided in a future IF release.

PKI and Key Management

This package provides services relating to the use of keys, especially those relating to PKI certificates and keys. As USAF is responsible for the issuance and management of certificates, this package only addresses those services dealing with certificate and key retrieval, utilization, and protection within GCSS-AF.

The IF supports server-side certificates, using the key protection provided by supported security product (IBM Global Security Kit). The IF also supports client-side certificates for the purposes of authentication and confidentiality services.

Note Future Capability: As DoD PKI plans and implementation mature, smart card authentication and CRL checking of certificates is anticipated to be supported by the IF.

6.1.3 Non-Technical Aspects of Security

Technical security components are only the starting point of a security “solution”. This is true of the IF, as of any distributed system. The non-technical elements of security include:

- Personnel security – includes security training, personnel screening and registration, promoting security consciousness among system users, and related activities
- Physical security – includes control of physical access, visual access, access to storage, EMSEC, system inter-connections, etc.
- Security management – includes conducting periodic risk assessments, defining and supporting a consistent security policy, providing for incident response capability, contingency plans, etc.
- Security testing and accreditation
- Procedural aspects – includes ensuring that system operations are handled in a security-conscious manner, using defined and reviewed processes
- Administrative aspects – includes system operation, maintaining system records, supporting system audits and forensic activities as needed, etc.

Provisions for these areas are a Government responsibility. Mission applications need to coordinate activities in these areas with those of overall GCSS-AF management. In general, the USAF and DISA share this responsibility, for GCSS-AF. See the GCSS-AF Ops and Support Plan for details.

6.2 Authentication

The most visible service that the IF provides is the user authentication service. The IF supports two mechanisms for user authentication: *userID/password* and *PKI Certificate based* authentication. All users will be authenticated via Policy Director WebSEAL. The significance of the IF providing an authentication service is that it centralizes user registration and relieves the MA from needing to provide another authentication service. Enterprise session management

is provided such that a user need only log in once³⁰. The IF requires the use of **IBM Policy Director** product to provide these services.

For applications that have audit and authorization requirements or other requirements needing a user's identity, the IF provides mechanisms for obtaining and passing this information amongst applications. The details of these mechanisms are explained in the subsections.

Refer to Section 6 Securing the Application for an identification of the authentication mechanisms that are used between the various components in the IF. These paths can visually be seen in Figure 73: Operational Diagram: Authentication & Authorization in Section 6.1 Overview of IF Security.

Table 30: Authentication Matrix

Path	Authentication
Browser – WebSEAL	<p><i>UserID /password</i></p> <ul style="list-style-type: none"> ◆ Individual end user's unique "name" ◆ Individual end user's unique <i>password</i> <p>User registry is in an LDAP Directory. WebSEAL performs a search of secUser for a dcePrincipal=<userid>. If success, performs an LDAP bind with resulting Distinguished Name (minus the secAuthority=Default tag) and the user supplied <i>password</i>.</p> <p><i>PKI Certificate Based</i></p> <ul style="list-style-type: none"> ◆ Client Certificate <p>User registry is in an LDAP Directory. WebSEAL performs a search of the LDAP directory for an object that matches the Distinguished Name in the certificate. If it finds one, it looks for a secCertDN attribute from the ifDNMap objectclass that maps to a Distinguished Name of a Policy Director user. If the Distinguished Name in the map matches a Policy Director user, then the user is authenticated.</p>
WebSEAL – Web Server	<p>One-way SSL authentication. WebSEAL rejects the connection if the DN in the supplied certificate from the Web Server does not match the DN configured in the junction.</p>
WebSEAL – Web Server/WAS AE Servlet Engine	<p><i>UserID /password</i></p> <ul style="list-style-type: none"> ◆ Individual WebSEAL server identity ◆ Individual WebSEAL server <i>password</i> <p>WAS AE configuration is setup to use the same user registry in the LDAP directory as Policy Director. WAS AE performs a similar process for authenticating the WebSEAL identity as the WebSEAL did for the end user. WebSEAL identity and <i>password</i> maintained in a configuration file on each WebSEAL installation.</p>
WebSEAL – IBM SecureWay Directory	<p><i>UserID /password</i></p> <ul style="list-style-type: none"> ◆ Individual WebSEAL server identity ◆ Individual WebSEAL server <i>password</i>

³⁰ The system forces the user to re-authenticate after inactivity timeouts, overall session timeouts, and if the user were connected to a different WebSEAL server whether at the same or different DECC-D IF installation.

Path	Authentication
Policy Director Servers—Policy Director Servers (includes WebSEAL – Security Management Server and Authorization Servers – Security Management Server)	<i>UserID /password</i> via DCE using kerberos 5 DCE keytab file with a <i>password</i> stash file. See Policy Director COTS documentation and the GCSS-AF installation procedures for more information. Also, refer to IBM and Transarc DCE documentation.
WAS AE Servlet Engine – IBM SecureWay Directory	<i>UserID /password</i> ◆ Common WAS AE server identity ◆ Common WAS AE server <i>password</i> Configured in the Global Security Settings
WAS AE Servlet Engine – Authorization Servers	<i>PKI Certificate</i> x.509 certificate generated using the SSLSVRCFG tool during installation/configuration. Configured in the <i>aznapi.conf</i> file.
WAS AE Servlet Engine – UDB (a.k.a. IBMs DB2)	<i>UserID /password</i> ◆ Sas.server.props and sas.client.props WAS AE configuration files
WAS AE Servlet Engine – WAS EE Component Broker	<i>UserID /password</i> ◆ Application specific <i>userid/password</i> configured in the applications property files. Procedurally this could all use the same <i>userid</i> and <i>password</i>
WAS EE Component Broker – Authorization Servers	<i>PKI Certificate</i> • x.509 certificate generated using the SSLSVRCFG tool during installation/configuration. Configured in the <i>aznapi.conf</i> file.
WAS EE Component Broker – DCE Cell Directory Service	<i>UserID /password</i> via DCE using kerberos 5 DCE keytab file with a <i>password</i> stash file. See Policy Director COTS documentation and the GCSS-AF installation procedures for more information. Also, refer to IBM and Transarc DCE documentation.
WAS EE Component Broker – Oracle	<i>UserID /password</i> ◆ <i>UserId/password</i> and <i>connect string</i> configured in CB SMUI Management Zones → <Application> Zone → Configurations <Application> Config → RDBConnections → <Application> Container
WAS EE Component Broker – UDB (a.k.a. IBMs DB2)	<i>UserID /password</i> ◆ WAS EE Component Broker configuration file
Policy Director Servers – DCE Cell Directory Service (CDS)	<i>UserID /password</i> via DCE using kerberos 5 DCE keytab file with a <i>password</i> stash file. See Policy Director COTS documentation and the GCSS-AF installation procedures for more information. Also, refer to IBM and Transarc DCE documentation.
WAS EE Component Broker – MQSeries MQM	<i>UserID /password</i> <i>UserId/password</i> and <i>connect string</i> configured in SMUI Management Zones → <Application> Zone → Configurations <Application> Config → RDBConnections → <Application> Container It uses the operating system user that the application is running as if no <i>userid/password</i> is supplied.
MQSeries MQM – MQSeries MQM	<i>UserID</i> and <i>password</i> hardcoded in a DISA supplied Security Exit

The sub-sections that follow explain the authentication paths that affect MA development, in more detail.

For developers of applications that consist of static HTML links:

- Presume users are authenticated before the URL is invoked.
- If the application needs user authentication data (specifically, the user's name/ID or the roles/groups the user belongs to), it can get this information from the HTTP header; see 6.2.1.)
- If the application needs to use the group/role information it gains from the HTTP header, the developer should coordinate with the GCSS-AF Enterprise Authority to ensure that the group/role names match between Policy Director and the application.

6.2.1 User Authentication

All users are authenticated via a Policy Director WebSEAL server (an IF component) to gain access to GCSS-AF. The two mechanisms supported by the IF are *userID/password* and *PKI certificate* based authentication. Policy Director WebSEAL is configured to request a user certificate and fallback to *userID/password* if no certificate is presented.

PKI Certificate Based Authentication

When a user enters a GCSS-AF protected URL³¹ into their web browser, an SSL handshake occurs. The Policy Director WebSEAL server sends to the browser a list of Certificate Authorities that it will recognize and requests the browser to send a user certificate. What occurs next depends on the capabilities and configuration of the user's browser.

If the user has a user certificate, the user will be prompted for a password to unlock the certificate database³². The browser verifies the server's certificate against the list of Certificate Authorities that are recognized by the browser. The browser will display a list of the user's certificates that were signed by one of the Certificate Authorities in the list that the server sent in the SSL handshake. It will prompt the user to select a certificate to use in this session.

Upon selection of the user's certificate the SSL handshake continues. The browser sends the certificate to the Policy Director WebSEAL server that verifies the user's certificate. This verification process includes checking the validity period (dates when the certificate is valid) and verifying the certificate chain is signed properly. The GCSS-AF WebSEAL product does not support CRL checking as of this time.

After verifying the certificate is valid, WebSEAL then checks to see if the Distinguished Name on the user's certificate maps to a valid user in Policy Director. WebSEAL searches for the Distinguished Name in an LDAP directory with an auxiliary objectclass of *ifDNMap*. If it finds an entry, it reads the Distinguished Name in the *secCertDN* attribute field. Lastly WebSEAL

³¹ GCSS-AF URLs use the HTTPS secure web protocol.

³² This process assumes that it was the first access to the certificate database or that the web browser was configured to always prompt for the certificate database's password.

then searches for the mapped Distinguished Name to see if it points to a valid Policy Director user.

Upon successful authentication, WebSEAL creates the user's credential and goes through the same process as with userid/password authentication for checking authorizations and providing web pages.

If the server's signer certificate, a.k.a. the Certificate Authority, is not in the browser's list, the user will be prompted if they wish to recognize the server's certificate³³. If a user has certificates but opts not to use them³⁴ or if the user does not have any user certificates, the user may select "Cancel" at any of the certificate prompts. The WebSEAL server will recognize that a user certificate was not presented and prompt the user for a userid and password. Some web browsers have the capability of being configured to not display a notification to the user if the user had no certificates that were supported by the server. This is an important configuration consideration as many users will not initially have certificates. Users will not want to have to acknowledge a dialog box indicating they don't have a certificate every time they want to log onto GCSS-AF with just a userid and password.

Some web browsers have the capability of being configured to prompt the user for the certificate database password every time a certificate is requested. Some web browsers also have the capability of being configured to not prompt the user to select a certificate if there is only one in the list.

Userid/Password Authentication

If a user does not present a certificate during SSL negotiation because the user chose not to or because the user does not have one, then WebSEAL will determine that the user is not authenticated and present the user with an enterprise logon screen prompting the user for a *userID* and *password*.³⁵ WebSEAL will use the response to determine whether the user is recognized by GCSS-AF. This mechanism provides the capability for a single enterprise logon capability.

No matter which authentication mechanism is used, WebSEAL then passes the user information, called a "credential," on to the backend systems in the HTTP header. To allow the system to be seamless, the MA has to use the information that is provided in the HTTP header to identify the user within their application and trust that WebSEAL authenticated the user properly.

WebSEAL essentially acts as a reverse HTTP proxy and after reformatting the HTTP request, makes the HTTP request to the backend Web Server. In the HTTP header, WebSEAL adds three name/value pairs: **IV_USER**, **IV_GROUPS**, and **IV_CREDS**³⁶.

³³ Typically browsers will prompt the user if they would like to accept (recognize) this server's certificate as valid for this session or forever.

³⁴ This may occur if a user other than the machine's owner is allowed to use the machine.

³⁵ Refer to the Tivoli SecureWay Policy Director Administration Guide v3.7 for details.

³⁶ Refer to the Tivoli SecureWay Policy Director Administration Guide v3.7 and the Tivoli SecureWay Policy Director Administration Guide: Additions and Corrections Version 3.7 for details.

The MA has to use the information that is provided in the HTTP header to identify the user within their application and trust that the user was authenticated properly by WebSEAL.

COTS or GOTS applications may opt to integrate with Policy Director and these name/value pairs directly. The preferred approach is to use the **SessionInfoStruct** structure for obtaining the session information. This approach isolates the MA from any changes to the mechanism by which WebSEAL provides the information. If an application has its own access checking mechanisms, it will need to pull the authenticated user's name (from IV-USER) and the user's role information (from IV-GROUPS). See Section 6.3 for details.

Example:

```
username = sessionInfoStruct.userSessionInfoStruct.userSecInfoStruct.username
```

6.2.1.1 Credential Passing

The **SessionInfoStruct** contains the user's username, groups that the user is a member of, and the user's credential. This credential identifies the user to Policy Director when making authorization decisions. The credential is an opaque structure. It contains the unique user identifier of the user and the unique group identifiers for the groups that the user is a member of, and a variety of other Policy Director specific information³⁷.

It is the responsibility of the MA to pass along the **SessionInfoStruct** from the Servlet or JSP as a parameter to the methods in the CORBA or EJB components so that access control decisions may be made from within those components. From the PDC test component, **PDCAddServlet** Servlet:

```
protected void persist(HttpServletRequest request, HttpServletResponse response, PartRecord
partRecord) throws Exception
{
    getRemoteInterface(request).addPart( 1, partRecord, getSessionInfoStruct(request));
}
```

Figure 74: Example of PDCAddServlet Code

6.2.1.2 Single Sign-On

The IF security architecture provides for a single sign-on (SSO) capability to "back end" applications, databases, Legacy systems, etc.

³⁷ Refer to the Tivoli SecureWay Policy Director Authorization API: Java Reference Version 3.7 for details.

Note Future Capability: Currently, a product limitation in WebSphere AE required the channel for communicating SSO information (the BA header) to be occupied for establishing the SSL connection for authentication and encryption between WebSeal and AE; see [“Security from Browser to EJB/CORBA Component”](#) in Section 6.1, especially **Steps - 17 and 18**, for a brief discussion of the details. In the next release of the IF, based on a new version of AE, this issue is expected to be resolved, so that a user’s back end identity information can be communicated to AE, to EE, or to non-WebSphere back end systems.

6.2.1.3 Legacy and Database Considerations

The IF security architecture provides for connections to Legacy systems and databases. It does NOT extend IF security features INTO these systems and databases, but provides for coordinating information with them. (This means that existing Legacy security mechanisms, including access checks and encryption, should be maintained until a modernized IF-based component is implemented to replace the Legacy system.)

When connecting to a back end Legacy system, there are a number of options, depending on where the connection is made. The simplest case is where WebSeal does the initial enterprise authentication, then passes control to a Legacy Interface Component (LIC) or wrapper, which can map the user’s enterprise identity to any Legacy identity(s) based on the user’s credentials supplied by WebSeal. The MA team can write this LIC or wrapper based on the details of the Legacy system in question. (Presuming the LIC is a first step in phasing out the Legacy system, its design should also reflect interfaces needed throughout the planned modernization, and provide for user identity to be passed as appropriate. But, in any case, the LIC should accept the credentials passed by WebSeal in the header and should be able to map between that user identity and *userID*(s) needed by the Legacy system being interfaced to.)

Note Future Capability: See [“Security from Browser to EJB/CORBA Component”](#) in Section 6.1, especially the [“Design Note Regarding Legacy Systems and Databases”](#), for a brief discussion of the details necessitated by the current product set. Currently, the same limitation described in Section 6.2.1.2 will prevent IF components behind WebSeal (i.e. AE and EE) from being able to pass authentication information to Legacy systems.

It is recommended that when modernizing Legacy systems, every attempt be made NOT to have multiple identities for individual users inside the Legacy system, or modernized Legacy components. The maintenance of multiple identities is expensive, and the mapping of identities inside the IF is computationally expensive and therefore impacts system performance. Ideally, the modernized Legacy system should be designed to use the user’s enterprise identity, to minimize both of these impacts.

The IF presumes that individual databases have their own provisions for security, and does not attempt to replace these but to interface to them. For example, Oracle provides a security module (Advanced Security Option or ASO). The IF presumes ASO is employed whenever Oracle is employed.

However, many databases expect users to logon to them individually. Ideally, an MA will be able to define Identifications (ID) based on roles/groups, so that separate database logons for individual users do not need to be maintained. (Ultimately, the IF will support audit reduction methods that will remove the objection to this, that user actions can't be tracked through group logons – a delegated audit chain would permit the user to be tied to specific group-logon actions.)

However, in the meantime, the IF architecture does provide for passing a user identity to a database such as Oracle.

Note Future Capability: Currently, a product limitation in WebSphere AE required the channel for communicating database logon information (the BA header) to be occupied for establishing the SSL connection for authentication and encryption between WebSeal and AE; see [“Security from Browser to EJB/CORBA Component”](#) in Section 6.1, especially **Steps - 17 and 18**, for a brief discussion of the details. In the next release of the IF, based on a new version of AE, this issue is expected to be resolved, so that a user's back end identity information can be communicated to AE, to EE, or to non-WebSphere back end systems.

6.2.2 Application Authentication

There will be occasions where an MA needs to authenticate to the system with the application's own identity. An example of this includes *wrapped* Legacy systems that cannot determine who the originating user is or be able to tie it back to a Policy Director user in order to create the user's credential. Another example would be when the MA needs one application to act on behalf of a user to perform a function. For example the developer may not want to give an end user direct access to a financial MA through an ACL, but they need the Supply MA to check to see if the user has sufficient funds to procure material. In this instance, the developer needs to authorize the Procurement MA to check for sufficient funds, and not the end user. Therefore the developer needs to create Procurement MA identity(ies) and add them to groups and ACLs following the Access Control process. These identities then need to be used to authenticate between application components.

6.2.2.1 Application to Application Authentication

The IF provides the *gcssafAuthenticateUser* method that is part of the **IFCBSecurityInfoAuthz** object in Component Broker to perform this function for the application. The **sessionInfoStruct** created by this function can be passed as a parameter to CORBA and EJB methods just as the **sessionInfoStruct** created for the user and passed in by the Servlet or EJB. To make applications work properly, it may be necessary to add additional parameters, as required by the invoked application. For example, the location parameter of the **sessionInfoStruct** is required by the IF test components.

The process for planning for access control for application identities is the same as for end users. See 6.3.1 Planning for Access Control to see the steps involved.

Standards and Conventions

It is recommended that standards and conventions, particularly in the area of naming conventions, be developed and maintained by the GCSS-AF Enterprise Authority and its Operations and Support arm. As the number of applications grow, the ability to identify what items are pertinent to the developer amongst hundreds or thousands of entries becomes significant. One recommended convention is outlined below. Basically it would put the responsibility of defining the functional domain (e.g. ILS) and location (e.g. Base Name or **Georef** code, MAJCOM, Enterprise, etc) to the GCSS-AF Enterprise Authority. The definition of the application and **appID** would be the responsibility of the MA coordinating with the Operations and Support Organization to verify that the names are not currently in use. The **appID** may be specific to an instance or to a class of components. Case is significant. The recommendation is to standardize on uppercase to make it easier to implement. Mixed case rules are harder for all parties involved to remember.

Application Identity Convention:

<domain>-<location>-<application>.<appid>
(e.g. ILS-BASE1-PDC.TRIGMON)

6.2.2.2 MQSeries Message Queue Manager Channel Exits

Authentication security for IBMs MQSeries Message Queue Manager is implemented via channel exits. This task is an Operations and Support task.

The steps involved are:

1. Obtain an updated security exit from the DISA SSO with the *name/password* pairs for the channels that are involved.
2. Alter the channel definitions to use the exit

Alter the Channel Definitions to Use the Exit

The exit works in pairs, one instance on each end of a channel pair. The exit is implemented by associating it with a SENDER or RECEIVER channel.

For example:

```
ALTER CHANNEL(SENDER.CHANNEL) CHLTYPE(SDR) +  
SECEXIT('ChannelExit.dll(SecurityExit)') SCYDATA('3') REPLACE  
ALTER CHANNEL(RECEIVER.CHANNEL) CHLTYPE(RCVR) +  
SECEXIT('ChannelExit.dll(SecurityExit)') SCYDATA('3') REPLACE
```

Figure 75: Example Queue Manager Channel Exits Code

The SCYDATA parameter is a value that tells the exit what level of logging should be provided. The log file itself is hard-coded and is something that should be specified to DISA SSO when the exit is requested.

Background

A security exit was created for IBM's MQSeries Message Queue Manager that provides basic authentication via a *userID /password* pair associated with each channel pair. These are hard-coded in a channel array within the exit as no acceptable secure repository could be identified that would meet DISA Field Security Office guidelines. Accordingly, for each new system implementation the user shall request a custom exit from DISA SSO.

Note: The security exit controls authentication between Message Queue Managers. It does not control submission of a message from a client. The security exit only supports normal channels and does not support cluster channels.

Note: It is planned that a COTS product (Policy Director for Messaging) will replace this security exit in the future.

Mechanics

The exits themselves interact by exchanging what are called handshake messages. When a SENDER channel is started, the RECEIVER partner exit is initiated. In our case, the receiver exit defers control. The SENDER exit then is given control and sends a handshake message containing the *userIDs* and *password* combination from the hard-coded channel table within the exit. This handshake message is not encrypted. The RECEIVER exit is then given control and attempts to validate the *userIDs/password* combination using whatever security interface is appropriate for the operating system in question. For example, on Windows NT the Win32 **LogonUser()** function is used, on AIX the **getpwnam()** function is used, etc.. If the RECEIVER channel is able to validate the *userIDs/password* combination it returns control to MQSeries indicating that message flow may then begin. If not, it returns control indicating that the channel is to be stopped. In effect, it provides the same level of security as does an ftp daemon verifying a user.

To provide confidentiality services for MQSeries traffic over the WAN, DISA has implemented Virtual Private Networks (VPNs). The handshake including the *userIDs* and *password* would be encrypted on the WAN over channels between MQSeries Message Queue Managers. Within the GCSS-AF enclave the MQSeries traffic will be in the clear.

6.3 Access Control

Access Control (or Authorization) is the aspect of information security that limits access to computer resources, especially programs and data, to protect them against unauthorized modification, loss, or disclosure. (There is an aspect of access control involving limiting access to physical computer resources and facilities. This aspect is not taken into consideration here, because on GCSS-AF, it is under control of the USAF and is dealt with in the Ops and Support documentation.)

In this section, we will explain the general flow of access control checks, as background. Then, in Section 6.3.1 Planning for Access Control, we describe the actions needed to plan for applying access controls to the MA. In Sections 6.3.2 Setting up ACLs through 6.3.4 Setting Up User

Data, we explain how various types of access data are set up in Policy Director. In Section 6.3.5 Access Control and Web Objects, we show how access checks are programmed using the services provided with the IF.

For developers of applications, that consists of static HTML links:

- **If the application does not require access checks beyond the initial access to the main URL/link, the developer does not need to worry about this section.**
- **If the application (product) performs its own access checks, the developer will need to pick up the user's name and groups information from the HTTP header (cross-reference Section 6.2). In this case, the developer shall ensure that the group names that are placed into Policy Director for the "MA" match the ones they will be checking against in the application/product's access database.**
- **If the application wants to use the IF's access check methods, the developer needs to read this section and do the appropriate design and development.**
- **In any event, it will be useful for the developer to read Section 6.3.5 Access Control and Web Objects.**

Access Control Concepts

In the IF, the access control mechanism is based on the tools provided by Policy Director, which in turn implements concepts common to industry-standard authorization methods. The basic concepts are:

User – A "user" is a representation of an identified and authenticated human user.

Group – A "group" is a representation of multiple users who share common access rights to objects and functions. This corresponds to the term "role"; i.e. users who have the same "role" will be assigned to a "group" together. (In other words, the IF provides a "role-based access control" or RBAC mechanism, via Policy Director.)

Access Control List (ACL) – An "ACL" is a representation of the access rights applied to an object or function. It represents the rules that shall be satisfied in order for a user to be authorized to access an object or perform a function.

Policy – In this context, a "Policy" is the combination of data about users, groups, and ACLs, that embodies the business rules about who should be allowed to do what inside the application or system.

The principle that ties these concepts together is this:

To perform a requested function or access a requested data object, the policy shall show that a user is assigned to a group that has the access right corresponding to the ACL for the requested function/object.

For example, John Smith is a user who needs to be able to perform operations on data objects of the *Parts* class. We create a user entry for John Smith inside Policy Director. We create a

Group inside Policy Director called Supply Clerk, to which John Smith's *userID* is assigned. We assign an ACL to Parts, inside Policy Director, and include Supply Clerk as a group that has certain access rights according to that ACL. Now, when John Smith sits down at his workstation, logs on, and requests to perform some function on a Parts object, the request is passed to Policy Director, which sees that John is authenticated (from his logon), that he is a Supply Clerk, and that Supply Clerk is a group that satisfies an ACL entry for that function on the Parts object, so Policy Director authorizes John to perform the requested function.

(For a more detailed explanation of these concepts, and how they apply in Policy Director, see Chapter 7 "Understanding Access Control" in *Tivoli SecureWay Policy Director Administration Guide Version 3 Release 6*.)

Note that from the perspective of Policy Director, an MA is a "third party" application, so the explicit access checks in the MA will use the **aznAPI** as explained in the Administration Guide. However, in GCSS-AF, we provide a Java interface to the API, to buffer the developer from some of the complexities of the API, to avoid the need for MA code changes as the API is updated, and to handle some of the associated data structures without requiring the developer to know their details.

Access Control Flow

General flow of access control in Section 6.1 Overview; discuss 'patterns' of access control here (use Ifs per this section; do own internally using authentication data passed by WebSeal; just use WebSeal's check on initial URL.)

Access Control Administration

In the deployed environment, DISA and the GCSS-AF Ops and Support organization are responsible for maintaining the IF access information. However, during development, MA teams may need to be able to set up test data for access checking. They are referred to Appendix A of the Policy Director Administration Guide for information on the use of the **ivadmin** tool for this purpose.

6.3.1 Planning for Access Control

The MA engineering team is responsible for planning and defining the data needed to ensure proper access controls are applied to MA data, functions, and menu items. The access control data will be entered into Policy Director, which performs the actual access checks. The MA engineering team will also need to decide at what level to request explicit access checks inside MA software, and to include the necessary software calls in their design and development.

IMPORTANT – This discussion is about access control for applications, as distinct from access control to individual boxes, operating system file access controls, etc. These are also important elements of an overall security approach. However, while the IF provides recommended "lockdown" procedures for IF servers, including recommended registry settings to limit access to those servers, the ultimate responsibility for boxes and their contents rests with the USAF (for workstations at bases) and DISA (for workstations and servers at DECC-Ds). Therefore, we

presume those organizations adequately address these elements and we limit this discussion to application-specific aspects of access control, especially those relating to MA data and functions. (Do be aware that application teams may provide input to USAF and DISA regarding access settings that need to be applied to files or folders that are specific to that application. The process for determining this information is similar to that described below.)

Principles of Access Control

The essential objectives access control methodologies will achieve are:

1. Ensure users have only the access rights required to perform their duties;
2. Limit access to especially sensitive resources to a minimum number of users with special privileges;
3. Restrict users from performing unauthorized functions (through automated means, as much as possible).

The objectives summarize the purpose of the IF access control requirements. We expect that they will also summarize the access control requirements of most MAs. If an MA has additional requirements that do not fall under one of these objectives, that MA may require additional analysis or technical support for providing access control.

Note: Access checks can be either **explicit** or **implicit**. **Explicit** access checks require specific software additions or changes to application code. The IF performs **Implicit** access checks automatically, without changing application software to provide for them. The advantage of **explicit** access checks is that they allow an application to control where checks are performed, and to retain control over the enforcement of access. (I.e., the application software receives the result of an access check and decides what to do about it, whereas an **implicit** check will be configured to automatically perform a specific action when a check succeeds or fails). The disadvantage of **explicit** checks is, obviously, that they do require the writing of extra code.

In the IF, all access checks on MA data and objects are **explicit**.³⁸ It is the MAs responsibility to ensure proper access controls are established and applied for MA information (using the IF tools and guidelines outlined below).

Information Sources

Information to use in setting up access controls will come from sources such as:

1. Legal documents (e.g. Computer Security Act, Privacy Act, DoD controls, etc.)

³⁸ This is a result of COTS functionality in the current releases of Policy Director and WebSphere. In upcoming releases of these products, IBM expects to provide support for implicit as well as explicit access checks. Once these releases are integrated into the IF, MA teams will be able to choose which kind of access check to implement.

2. Existing security classifications (e.g. data that is already identified as Sensitive, Secret, etc.)
3. MA Requirements (if explicit requirements associated with access do not exist, the MA engineering team should derive such requirements)
4. Use cases
5. Functional experts

Key Steps in Providing for Access Control

The basic steps an MA team needs to perform in this area are:

1. Determine what resources, individual, or collectively, need protection.
2. Decide what job roles, or groups of users, will require what type(s) of access to each resource (or group of resources). Derive Group data from this information.
3. Decide how to determine which MA users to place in each role or group that is established in the second step. Derive user data and registration guidance from this information.
4. For each resource listed in **Step - 1**, determine what level and type of protection is required (relying heavily on the Group data derived in **Step - 2**). Derive ACL data from this information. Also, decide where MA software needs explicit access checks, based on the level of protection analysis.
5. Coordinate with the GCSS-AF Operation Authority to set up the ACL, Group and User data in Policy Director.
6. Program and test the software based on the decisions in **Steps - 2** through **4**. Adjust the ACL, Group and User data, if needed.
7. Coordinate with the GCSS-AF Operation Authority and DISA, to migrate the application, including its access control data, to the operational environment on the IF.

Guidance on performing each of these steps is available in the rest of this section and its sub-sections. (Note: Although they are presented here as discrete steps, in reality, the process should be regarded as iterative; i.e. it will be valuable to revisit earlier steps as more information is obtained.)

Step 1 - Determine Resources Requiring Protection

The first step is to determine what MA information and functionality may require protection. This assessment needs to be performed in conjunction with the owner(s) of each resource the

MA shall protect. In this step, the objective is not to define the kind or level of protection, only the objects or a function to which protection needs to be applied.

The output from this step will be a list of data objects, MA functions or components that require protection from unauthorized access.

This step occurs during the MAs Requirements Analysis and Architecture phases of development.

Resources that may need to be assessed during this step include:

- A. Databases
- B. Database partitions
- C. Individual data objects (rows)
- D. Files or folders
- E. MA software components
- F. MA functions (methods on MA objects, in the OO sense of “object”)

(Again, recall that physical objects, such as computer system units, are protected by appropriate physical and personnel controls as determined by USAF and DISA, so they should *not* be listed here. Also, any data that does not require access control, such as information that is freely available to the public, may not need to be listed here, unless it is combined with more sensitive data during storage or presentation. For example, a public web page’s contents may not need to be access controlled for *reading*, precisely because it is “public.” At the same time, the developer may need to control what actions can be taken even on public objects. So, for example, the source file for that public web page may need to be controlled for *writing*, because it could otherwise be defaced by a cracker in a way that would be embarrassing to the Air Force.)

Note that the data objects that are listed will depend in part on the nature of each MAs specific activities. For example, an MA such as Finance will work heavily with documents, i.e. files in folders. Another MA, for example Supply, may be concerned solely or primarily with more granular data stored in databases. Therefore, the list of data objects is likely to vary from MA to MA, and providing precise guidance for what to include is not possible without knowing the details of each MAs environment.

However, all MAs will provide user-callable functions, and will have one or more software components. All functions that an MA will provide to users should be listed at this stage, along with any software components that may be invoked by a user. For example, in our Pseudo-Supply test software, we provided two components (Parts Data Collection, and Requisition), and one or more functions under each (Get Part, View Parts List, Add Part, Use Part, etc.).

Step 2 - Derive Group Data

The next step is to define the Groups that will represent user roles. This activity should flow directly from the previous one, as the developer is defining resources that require protection, they should be thinking about Groups that will use each resource. In this step, the developer will be completing the list of Groups and documenting how they relate to the resources; the individual ACLs that correspond will be finalized in **Step - 4**.

The MA should also look to see what roles are currently defined in the system that they might be able to re-use. The Operations and Support Organization and the Security Organization need to be employed to aid in this analysis. Before re-using existing roles, it is necessary to ensure the rules for adding users to the role are well defined and agreed to. Otherwise when other MA using the same role have slightly different criteria for adding users to the role, undesired access may be granted to the MA.

Note: One of the major benefits of the IF is the ability to re-use existing role/group definitions. Each MA doesn't necessarily have to create its own unique role/group. If all Gunter AFB users can access the TDY MA and the Benefits MA, then a single group called "GUNTER AFB USERS" can be created; i.e. there do not need to be separate groups for TDY users at Gunter and Benefits users at Gunter. Any new MAs requiring that user set can re-use that Group. This consideration will be especially important to an MA using data shared with other MAs.

It is necessary for the MA and the Operations and Support Organization to maintain which groups an MA uses. The MA needs to be informed when a change, to the criteria for adding users to the groups, is being requested. All MAs need to agree to the change. If the decision to make the change is not unanimous, the "losing" MAs will need to select/create a different group. It is also recommended that the list of groups the MA uses be maintained so that the Operations and Support Organization can periodically poll the MAs to verify that groups are still required. Otherwise, orphaned groups that are not used by any MAs clutter the system.

In general, the developer should define as few Groups as possible while still allowing needed functionality. Note that there is a pre-defined group called "any-authenticated" that can be used to reduce the number of Groups needed – it represents all users who have valid credentials to log onto the system. For many applications, a significant number of actions (such as reading public files) may be able to be covered by this group, thus avoiding the need to define several groups to limit access to these actions.

In our Pseudo-Supply test software, we defined these groups or roles:

Supply Clerk – users who read parts data and place parts orders. These users were sub-divided by base; i.e. some could only read parts data for Base 1, some for Base 2, etc; this restriction was to simulate the "one base, one wing" concept.

Maintenance Clerk – users who enter new parts and update existing parts entries in the database. We also divided these users by base and MAJCOM.

Admin – users who administer the Parts Data Collection databases. Here we defined administrators for each database – a base-level admin for each base’s PDC, an enterprise admin for the USAF enterprise PDC, and a “super admin” with access to all levels of PDC database. (Note that this is NOT the same as a system-level “super” admin or “root” user.)

In all, there were 15 groups defined to handle this set of roles.

In the configuration of the test applications, some users were configured to support multiple bases. If the MA has a consistent grouping of users, like MAJCOM users that support that support multiple bases, a group for each of the MAJCOMs could be created. It could be included in the ACLs, to be discussed below, for the given bases. This reduces administration, because it would not be necessary to add MAJCOM users to each and every base group that they is that they need access to. The MAJCOM users can be added to a specific MAJCOM group that provides access to only the specified resources within the base. The process for defining groups and ACLs is intertwined and should be re-iterated a few times for optimization.

The output from this step should be a list of Groups (roles) to which users can be assigned. The complete list should include all groups/roles needed to perform all functions in the application in an authorized manner consistent with the MAs requirements. In other words, there should be no case where a user would need another (undefined) role in order to perform a needed function inside the MA.

This step should be performed during the MAs Design phase of development.

Step 3 - Derive User Data

The output from this step should be a list of user attributes and business rules that are needed in order to assign users to Groups. It need NOT include an explicit list of all users, only guidance to those who register users to assist them in ensuring each user is assigned to the Groups appropriate to that user’s job(s). This involves information specific to each MA, is not peculiar to the IF, and is thus difficult to provide step by step instructions for.

It may involve consideration of some or all of these considerations:

- The user’s rank
- The user’s organization or command or MAJCOM
- The user’s security clearance level

- The user's location(s) – especially, what location(s) is the user allowed to read or write data for, even though they may not be physically at those location(s)
- Any temporary assignments the user may be directed to
- Specific characteristics required by an MA (e.g. can a user handle data for nuclear weapons? For specific aircraft types? For restricted personnel files? Etc.)

It is necessary to ensure the criteria, the attributes and business rules, for adding users to the role(s) are well defined and agreed upon. Otherwise, when other MAs using the same role have slightly different criteria for adding users to the role, undesired access may be granted to the MA. As identified in the previous section, it is incumbent on the Operations and Support Organization to notify the MA of a change to the criteria of adding users to a group. This notification allows the MA time to voice any concerns about the change particularly if the change may provide unintentional unauthorized access to an MA and its data.

This step should be performed during the MAs Design phase of development.

Step 4 - Determine Level and Type of Protection

Virtually all information requires some kind of protection. However, some information is of such low criticality, or is so generally available, that its damage or deletion will cause no practical loss. Likewise, some information is of such high criticality or sensitivity that its damage or deletion could be catastrophic. Therefore, an essential action an MA team shall perform is to decide which of the resources under the MAs control fall into which levels of criticality.

The output from this step should be a set of access categories and the rules or methods of determining how any object or function is allocated to a category. The criteria for this determination should be objective, carefully defined and documented, and based on the risks. The categories and rules should be expressible in the form of ACLs that can be entered in Policy Director.

This step should be performed during the MAs Architecture phase of development.

This is primarily a risk and cost/benefit assessment activity. That is, the monetary and system performance costs of applying access controls to a particular object shall be weighed against the losses that could be incurred if the access controls are not applied. The costs of protection should not exceed the benefits of protection.

For each data object or function that was listed in **Step 1**, provide answers to some basic questions:

Does this item require access protection? (This most basic risk assessment should be performed first for all items. Only those that are judged to require explicit access checks should go through the rest of this analysis.)

If it does, what actions require protection? For example, the MA may have items that can be read by anybody, but can only be written to by users with specific roles.

Note that the Create, Read, Update, Delete, Execute actions³⁹ are provided for in Policy Director. Other actions can be defined to Policy Director so that ACLs can be applied to them. Therefore, one of the decisions the MA engineers need to make is whether other actions will be performed on this item, and to document them. Typically, in an object-oriented design, these actions will be represented as methods on an object; in GCSS-AF, this will be a CORBA object, or an Enterprise JavaBean.

Do the permissions vary by location or organization? For example, in our Pseudo-supply example, there were base supply databases and an overall AF enterprise supply database. We presumed a requirement to limit access to the overall enterprise database to a restricted set of users. We also presumed a requirement that limited access to each base database to users with certain roles pertaining specifically to that individual base. Therefore, we needed to provide for access rules based on location. Some MAs will need similar rules, some may need an additional “region” check, and some may need to limit access by MAJCOM or unit.

Do the permissions vary by security classification? I.e., does the MA handle items that are classified? If so, how are the classified items partitioned from unclassified ones, and what rights does a user need to access each classification? (Keep in mind that the IF is not designed to handle material above the Sensitive Unclassified level.)

At what level should the access checks be applied? In general, it is recommended to apply access checks at the ‘highest’ or ‘outermost’ level (of the data hierarchy, menu or folder tree, etc.) that satisfies security requirements, in order to reduce processing overhead. For example, it is better to apply access checks to an entire database than to individual items in the database. The ideal situation is to place objects being protected into a hierarchy, and then apply ACLs at the highest level(s) in the hierarchy that provide the needed protection.

Does the MA have any unique or MA-specific rules that affect access to data objects or functions? For example, in the supply domain, there are a number of special categories of parts – hazardous materials, nuclear materials, etc. – that require special permissions to order or handle.

Are there any special, temporary, or emergency conditions that may affect access rules? For example, does the MA have data that should be restricted under normal circumstances, but may have those restrictions relaxed under wartime deployment conditions?

³⁹ See 6.3.2 Setting up ACLs for further details on the definitions of the actions/permissions.

The answers to these questions will provide the basic data the MA team needs to define the ACLs on the items listed in Step 1.

In general, the process to follow should be:

- A. Determine the business rules to apply in deciding access to each item in the hierarchy. Push each rule as high in the hierarchy as possible. For example, if there is only one business rule governing who can read any item the MA shall protect, then apply this rule to the root level of the hierarchy, not to each item individually.
- B. Derive the ACLs corresponding to each rule or set of rules at each point in the hierarchy.
- C. Derive the hierarchy of objects and functions. (Note: As much as possible, this hierarchy should be mapped to the menu hierarchy discussed in Section 5.2.

See Table 31: Pseudo-Supply Test ACLs, for a list of ACLs that were created for the Pseudo-Supply test application.



Table 31: Pseudo-Supply Test ACLs

ACL Name	Description	Groups And Permissions
ACL-ILS-BASE1-RESTRICTED ⁴⁰	<p>This ACL is used strictly for filtering the menu system. The <i>audit</i> flag is turned off because the menu filtering system was checking to see if the user had access and no access was actually being attempted. The menu system has a separate object space, /IF-Menu, in Policy Director.</p> <p>This ACL was created to allow all of our Pseudo-Supply test application administrators for a given base and the more generalized administrators to view a menu item.⁴¹</p>	<p>user cell_admin abcTdmlrx group ivmgrd-servers Tl group iv-admin abcTdmlrx group GCSS-AF-Admins abcTdmlrx group ILS-Admins abcTdmlrxKRUDE group ILS-Base1-Admins abcTdmlrxKRUDE any-other⁴² T unauthenticated A</p>
ACL-ILS-BASE1-CLRKMAINT	<p>This ACL is used strictly for filtering the menu system. The <i>audit</i> flag is turned off because the menu filtering system was checking to see if the user had access and no access was actually being attempted. The menu system has a separate object space, /IF-Menu, in Policy Director.</p> <p>This ACL was created to allow all of our Pseudo-Supply test roles for a given base and the appropriate administrators to view a menu item.</p>	<p>user cell_admin aAbcTdmlrx group ivmgrd-servers ATl group iv-admin aAbcTdmlrx group GCSS-AF-Admins aAbcTdmlrx group ILS-Admins aAbcTdmlrxKRUDE group ILS-Base1-Admins aAbcTdmlrxKRUDE group ILS-Base1-Supply_Clerks ATRr group ILS-Base1-Maintenance ATKRr any-other AT unauthenticated A</p>

⁴⁰ Similar ACLs were created for the Base2, Base3, and Enterprise “locations” with the groups changed to reflect the appropriate location. This applies to all of the ACLs listed used by the IF test components.

⁴¹ The menu system checks the IF Menu Object space for the menu against the “r” lower-case r READ permission.

⁴² *any-other* is the syntax used by the ivadmin command-line tool for modifying the ACL. It is displayed as any-authenticated in ivconsole, the Policy Director Administrator GUI.



ACL Name	Description	Groups And Permissions
ACL-ILS-BASE1-PDC-MAINTENANCE	<p>This ACL is used strictly for filtering the menu system. The <i>audit</i> flag is turned off because the menu filtering system was checking to see if the user had access and no access was actually being attempted. The menu system has a separate object space, /IF-Menu, in Policy Director.</p> <p>This ACL was created to allow the appropriate base Maintenance personnel to view a menu option as well as the appropriate administrators, but not the Supply Clerks.</p>	user cell_admin abcTdmlrx group ivmgrd-servers Tl group iv-admin abcTdmlrx group GCSS-AF-Admins abcTdmlrx group ILS-Admins abcTdmlrxKRUDE group ILS-Base1-Admins abcTdmlrxKRUDE group "ILS-Base1-Supply_Clerks" T group ILS-Base1-Maintenance TKRr any-other T unauthenticated A
ACL-ILS-BASE1-REQUISITION	<p>This ACL is used strictly for filtering the menu system. The <i>audit</i> flag is turned off because the menu filtering system was checking to see if the user had access and no access was actually being attempted. The menu system has a separate object space, /IF-Menu, in Policy Director.</p> <p>This ACL was created to allow the appropriate base Maintenance personnel to view a menu option as well as the appropriate administrators, but not the Supply Clerks.</p>	user cell_admin abcTdmlrx group ivmgrd-servers Tl group iv-admin abcTdmlrx group GCSS-AF-Admins abcTdmlrx group ILS-Admins abcTdmlrxKRUDE group ILS-Base1-Admins abcTdmlrxKRUDE group ILS-Base1-Supply_Clerks TKER group ILS-Base1-Maintenance TKER any-other T unauthenticated A
AUD-ILS-BASE1-CLRKMANT	<p>This set of ACLs (Base1-3, & ENT) protects the ILS Pseudo-Supply test MA. Each role/group is given the appropriate access as defined by our design using the KRUDE permissions⁴³ defined in Section 6.3.2.</p> <p>The set of permissions in this ACL has the Audit permission set (upper case A) as this is ACL is effectively attached to the methods of the application. This ACL is physically placed at the application level and the ACL flows down the sub objects: modules, interfaces, and methods.</p>	user cell_admin aAbcTdmlrx group ivmgrd-servers ATl group iv-admin aAbcTdmlrx group GCSS-AF-Admins aAbcTdmlrx group ILS-Admins aAbcTdmlrxKRUDE group ILS-Base1-Admins aAbcTdmlrxKRUDE group ILS-Base1-Supply_Clerks ATRr group ILS-Base1-Maintenance ATKRr any-other AT unauthenticated A

⁴³ The KRUDE permissions are defined in the following section.



ACL Name	Description	Groups And Permissions
AUD-ILS-BASE1-REQUISITION	<p>This set of ACLs (Base1-3, & ENT) protects the ILS Requisition test MA. Each role/group is given the appropriate access as defined by our design using the KRUDE permissions⁴⁴ defined in Section 6.3.2.</p> <p>The set of permissions in this ACL has the Audit permission set (upper case A), as this is ACL is effectively attached to the methods of the application. This ACL is physically placed at the application level and the ACL flows down the sub objects: modules, interfaces, and methods.</p>	user cell_admin aAbcTdmlrx group ivmgrd-servers ATl group iv-admin aAbcTdmlrx group GCSS-AF-Admins aAbcTdmlrx group ILS-Admins aAbcTdmlrxKRUDE group ILS-Base1-Admins aAbcTdmlrxKRUDE group ILS-Base1-Supply_Clerks ATKer group ILS-Base1-Maintenance ATKer any-other AT unauthenticated A

⁴⁴ The KRUDE permissions are defined in the following section.

The output should also include a mapping of Groups to ACLs that were defined in step 2. Each ACL should have at least one Group that maps to it. Each Group should map to one or more ACLs. If there are any “orphan” ACLs or Groups, it indicates an error in the analysis, and steps 2, 3 and 4 should be re-visited.

Step 5 - Set Up Data in Policy Director

The output from this step should be a copy of Tivoli Policy Director with the ACLs, Groups, and User data laid out in accordance with the output from steps 1-4. (Note that this is presumed to be in a test environment. Once the use of the data and associated MA software is validated in the test environment, including any updates to the data, the required setup should be formally documented. It will then be able to be entered into the production environment.)

This step should be performed during the MAs Design and Code phases of development.

Step 6 - Develop Access Control Checks in MA Software

Based on the analysis performed above, the engineering team can now code explicit access control checks that were identified above in the MA software. See Section 6.3.5 Access Control and Web Objects and its subsections.

The output from this step will be MA software with access checks, ready to be unit and string tested.

This step should be performed during the MAs Design, Code, and Test phases of development.

6.3.2 Setting up ACLs

6.3.2.1 ACLs

A group of permissions is called an Access Control List (ACL) or policy template (See the [Policy Director Administration Guide](#), chapter 7). Users and groups and their permissions are listed within an ACL. ACLs define user and group Read, Write, Execute, and Delete access to objects, as well as numerous other restrictions and permissions. When the ACL is applied to an object, the policy, which that ACL represents, is enforced on that object. For example, if a policy were implemented to require authentication of all users on an object O before being read, the unauthenticated user would be added to the template with no access permissions:

The Policy Director Authorization Server compares user credentials to permissions in ACLs to determine what a user is allowed to do.

ACLs define which groups or users are allowed different types of access. An ACL can be applied to any protected object at any point in the Policy Director namespace. In Policy Director, the protected object namespace is a hierarchy of objects that can be protected. Installed with the product are the root containers */WebSEAL*, */NetSEAL*, and */Management*. (Users and Groups are stored and displayed separately within the Policy Director Management Console.)

All objects existing under the applied point will inherit parent access control lists.

This is a very powerful mechanism that permits applying access controls to large groups of objects with one action. Developers should use it whenever possible, by carefully defining their object hierarchies to take advantage of common access needs. This minimizes both development time and maintenance expense.

Other permissions that can be granted/restrictions that can be applied include:

- IF-Designated Method level access control attributes (K,R,U,D, & E)
- Custom Mission-Application Defined permissions (shall be approved by the GCSS-AF Enterprise Authority)
- Whether user access to an object should be audited
- Ability to attach an ACL to object
- Ability to Traverse to the next object down in the namespace
- Integrity of object access
- Privacy of object access (P permission)
- External (third-party) authorization requirements

Additionally two different ACL-specific users are:

All users (any-authenticated) permissions on an object
Unauthenticated user's permissions on an object

Each ACL has a name, and the name applied to an ACL should represent the security policy of the specific configuration. For example, the ACL applied to a protected object namespace root object should indicate that it is the master default security policy and that it is for that root object of the namespace.

A single character identifies permissions that will be checked for access decisions. The following permissions should be used for method-level access control and fine-grained access control (The methods referenced refer to methods within Component Broker):

Table 32: Access Control List

K	Create Required by any method with the prefix “create” in its operation name. For example, the createFromPrimaryKey method on IHome requires that the developer possess the “K” permission.
R	Read Required by any method, including readable attributes, with the prefix “get” in its operation name. For example, the <i>getHome</i> method on the Imanageable interface and the priority_filter attribute on the CosNotifyChannelAdmin::ConsumerAdmin interface (when used as a <i>getter</i>) requires that the developer possess the “R” permission.
U	Modify (update) Required by any method, including writeable attributes, with the prefix “set” in its operation name. For example, the set_qos method in the CosNotification:QoSAdmin interface, or the priority_filter attribute on the CosNotifyChannelAdmin::ConsumerAdmin interface (when used as a <i>setter</i>) require that the developer possess the “U” permission.
D	Delete (remove) Required by any method with the prefix “remove” in its operation name. For example, the remove method on the CosLifeCycle::LifeCycleObject interface requires that the developer possess the D permission.
E	Execute Required by any other method not satisfying the conditions indicated above.

For the Integration Framework Test applications, the *actions* are mapped as follows:

Add Part = K Query Part = R Update Part = U Remove Part = D AddRequisition=E
--

Figure 76: IF Test Applications Actions

The MA engineering team will need to determine or define the mapping between each function/method requiring an access check and the PD Action they are making the access decision for.

Policy Director has a number of built-in permissions in four categories: Base, Generic, NetSEAL, and WebSEAL. They are listed below. See the Policy Director Administration Guide page 86- for descriptions on their function.

Table 33: Policy Director Built-in Permissions

Base (a) Attach (A) Audit (b) Browse © Control (g) Delegation (I) Integrity (P) Privacy (T) Traverse	Generic (d) Delete (m) Modify (s) Server Admin (v) View
	NetSEAL © Connect (p) Proxy
	WebSEAL (l) List Directory ® Read (x) Execute

6.3.2.1 Application of ACLs

In “Guidelines for a Secure Namespace” (Administration Guide, p 92), the following guidelines are listed:

“These guidelines provide information that helps to make the namespace secure:

1. Set high-level security policy on container objects at the top of the namespace. Set exceptions to this policy with an explicit ACL on objects lower in the hierarchy.
2. Arrange the protected object space so that most objects are protected by inherited rather than explicit ACLs.
3. Inherited ACLs simplify the maintenance of the tree because they reduce the number of ACLs the developer shall maintain. This lower maintenance reduces the risk of an error that could compromise the network.
4. Position new objects in the tree where they inherit the appropriate permissions. Arrange the object tree into a set of subtrees, where each subtree is governed by a specific access policy. The developer determines the access policy for an entire subtree by setting an explicit ACL at the root of the subtree.
5. Create a core set of ACL templates and re-use these ACLs wherever necessary. Because an ACL template is a single source definition, any modifications to the template affect all objects that are associated with this ACL.

6. Control user access using groups. It is possible for an ACL to consist of only group entries. Adding users to or removing users from these groups can control access to an object by individual users efficiently.”

“Sparse ACL model for ACL inheritance” (Administration Guide, p 96) details how to create a Sparse, or inherited, ACL model.

“Any object without an explicitly attached ACL inherits the ACL of its nearest container object with an explicitly set ACL. In other words, all objects without explicitly attached ACLs inherit ACLs from container objects with explicitly attached ACLs. A particular chain of inheritance is broken when the developer attaches an explicit ACL on an object.”

This means that one can attach an ACL at the root of a namespace, and the permission settings of that ACL flow down to any child namespace entries.

6.3.3 Setting Up Groups

Groups are also referred to as “roles” because of their ability to divide users by job or responsibility. Policy Director allows the storage of its groups in an LDAP directory. This section will outline how to store groups in LDAP through Policy Director, naming conventions while doing so, and how to map roles across tiers or Policy Director namespace branches.

Policy Director Groups Stored in LDAP

Policy Director defines groups using a description within the console, and stores the groups in LDAP. The distinguished name for the group is specified when the group is created in the Policy Director Console. The DN should reflect the centrally located roles container within LDAP. For example, one could create a container object within LDAP such as “**ou=Roles,ou=GCSS-AF,ou=USAF,ou=DoD,o=U.S. Government,c=us**” This space is reserved for Policy Director groups. Then any new roles would have a DN of “**cn=GroupName,ou=Roles,ou=GCSS-AF,ou=USAF,ou=DoD,o=U.S. Government,c=US**”. The groups should not be saved in the same LDAP container as the users. I.e. the user DN should not contain “**ou=Roles**”. Groups should also not spread throughout the LDAP server because of the time, which Policy Director Console would require to retrieve all the groups.

IMPORTANT – The MA development team should strive to minimize the number of groups it defines. In other words, set up as few groups as possible while still capturing the MAs access control requirements. For example, if the MA does not require data access to be restricted depending on what base the user is assigned to, then do NOT define a separate group for each base’s users. Instead define groups at the MAJCOM, region, or even USAF levels, which will reduce the number of groups required.

Group Naming Convention

A few general rules of thumb should be applied when defining and naming groups. Group names shall be coordinated with the GCSS-AF Enterprise Authority to avoid conflicts. The case of the names should be consistent; using all uppercase is recommended, because Policy Director is case-sensitive. It is recommended that the group name be a combination of the domain, location, and role description, separated by hyphens. However, any consistent scheme could be used. A hierarchical concatenation of the name is recommended. This requires concluding that which is most general, most specific, and everything between for a given MA or domain. If the domain were most general, it would appear first. Were location to be the next appropriate level, it would appear next. For example:

```
ILS-BASE1-ADMINS  
ILS-BASE2-SUPPLY_CLE RK
```

In the example, the domain (ILS) is placed first, followed by the location, and finally the role description itself.

Mapping Roles Across Trees in Policy Director

A mission application may manage different-tiered objects to be protected. For example, web pages may be located under the WebSEAL tree, and methods to be protected may be located in a custom tree or under the WebSphere tree. Administrators of a particular portion of the Policy Director tree (Ex. WebSEAL) may also be selected to administer corresponding portions of the other (Ex. method-level access checks under a custom tree). To facilitate administration, it may be desirable to use the same roles in both portions of the Policy Director tree.

The Integration Framework contains a segment called **IFSPDSPolicyDirectorScripts**. The **bin\IFSPDSAdmin** directory of this segment contains scripts to set up example Policy Director objects on both NT and Solaris. The objects are grouped logically by object.

6.3.4 Setting Up User Data

Each user added via Policy Director's Console is stored in LDAP in a similar fashion to groups. Policy Director stores the user's name, description, LDAP DN, and other data about the user such as *password* age, login failures, and the user's last login in LDAP.

Users are usually added though the PD Console. The console allows associating a description with the user. The user data is stored in an LDAP directory. The user data should be located in one place, determined by the user DN. The base portion of PD User DNs should consist of "**ou=Users,ou=GCSS-AF, ou=USAF, ou=DoD, o=U.S. Government, c=US**". The format of the common name should be based on AF-defined user naming standards.

Making Sure the Necessary Data Resides in the Application

There may be cases where an application requires information that is not normally provided for in PD, to complete its requirements for access control. Use of the IF-provided data and methods can facilitate the design in such cases. For example, a Supply application might need to be able to identify users who have access to Hazardous Materials. “HazMat” is not a field normally maintained in an LDAP directory. One option is to extend the LDAP schema to include a “HazMat” field that can be mapped to a user identity. Then the MAs Servlet could receive the ID in the SessionInfoStruct, do an LDAP lookup to find “HazMat” for that user, and pass the result to the MAs EJB or CORBA components via the extra fields in the SessionInfoStruct. This way, the MA only has to do this lookup once, rather than once for each method the user may invoke. See Appendix A of this Developer’s Guide for details of the classes, methods, and structures supplied by the IF that may assist in this sort of application need.

6.3.5 Access Control and Web Objects

The IF supports the capability for providing access control on Web objects. The IF protects URLs, using the Tivoli SecureWay Policy Director product. All users access Web Content through the Policy Director WebSEAL component of the Policy Director product. It is necessary to implement the Key Steps in Providing for Access Control to provide Access Control to Web Objects.

Key Steps in Providing for Access Control

Note that the steps described below are very similar to those in Section 6.3.1. The intention is to focus in this section on actions within each step that are specific to web objects (URLs, HTML pages, etc.), so that applications that only interface to the IF at the Web level can concentrate on the specifics of their situation.

The basic steps an MA team needs to perform in this area are:

1. Determine what resources, individual or collectively, need to be protected.
2. Decide what job roles, or groups of users, will require what type(s) of access to each resource (or group of resources). Derive Group data from this information.
3. Decide how to determine which MA users will be placed in each role or group that is established in step 3. Derive User data and registration guidance from this information.
4. For each resource listed in step 1, determine what level and type of protection is required (relying heavily on the Group data derived in step 2). Derive ACL data from this information. Also, decide where MA software needs explicit access checks, based on the level of protection analysis.
5. Coordinate with the GCSS-AF Operation Authority to set up the ACL, Group and User data in Policy Director.
6. Program and test the software based on the decisions in steps 2-4. Adjust the ACL, Group and User data, if needed.

7. Coordinate with the GCSS-AF Operation Authority and DISA, to migrate the application, including its access control data, to the operational environment on the IF.

Guidance on performing each of these steps for Web Objects is provided in the rest of this section. (Note: Although they are presented here as discrete steps, in reality, the process should be regarded as iterative; i.e. it will be valuable to revisit earlier steps as more information is obtained.)

Step 1 - Determine Resources Requiring Protection

The first step is to determine what MA information and functionality may require protection. This assessment needs to be performed in conjunction with the owner(s) of each resource the MA shall protect. In this step, the objective is not to define the kind or level of protection, only the objects or functions to which protection need to be applied.

The output from this step will be a list of data objects, MA functions or components that requires protection from unauthorized access.

This step is an iterative process that is performed during the MAs Requirements Analysis, Architecture, Design, and Implementation phases of development.

Resources that may need to be assessed during this step include:

Web Objects, that is, anything addressable by a URL: Servlets, JSPs, static HTML pages, CGI scripts, graphics, PDF files, and other documents

It is recommended that the developer identifies all URL addressable information even identifying those that do not need access control. It may be sufficient to do this at a directory level and not a file level. This is a necessary step because it is desirable to structure the directory tree and the content contained therein such that ACLs can be applied efficiently. For example, it would not be wise to put limited access content in the same directory as publicly available content. If this is done, the developer will have to place ACLs on each item in the directory and will not be able to apply it at a directory level. The MA Developer should not carry this out to the extreme. As the application evolves over the life cycle, additional functionality and capabilities will be created as well as new roles identified for this application. The guidance is simply that the MA Developer should consider the security ramifications of the application layout from the inception of the design including the directory structure.

The WebSphere Application Server Advanced Edition (WAS-AE) creates aliases for application root directories that are accessible by a URL. The layout of the application directories in WAS-AE also needs to be taken into account.

Step 2 - Derive Group Data

The next step is to define the Groups that will represent user roles. This activity should be undertaken in light of the entire MA and not just from the Web Objects point of view. Please refer to 6.3.1 Planning for Access Control for details on this process.

Step 3 - Derive User Data

The output from this step should be a list of user attributes and business rules and criteria that are needed in order to assign users to Groups. This activity should be undertaken in light of the entire MA and not just from the Web Objects point of view. Please refer to 6.3.1 Planning for Access Control for details on this process.

This step should be performed during the MAs Design phase of development.

Step 4 - Determine Level and Type of Protection

To reiterate the information in Section 6.3.1: Virtually all information requires some kind of protection. However, some information is of such low criticality, or is so generally available, that its damage or deletion will cause no practical loss. Likewise, some information is of such high criticality or sensitivity that its damage or deletion could be catastrophic. Therefore, an essential action an MA team shall perform is to decide which of the resources under the MAs control fall into which levels of criticality.

This is repeated because the MA, working with the Operations and Support Organization and the Security Organization, needs to evaluate the criticality of the MAs web objects (Servlet, JSP, HTML page, etc) in regard to the overall application. If the application is providing explicit access control checks for the backend methods invoked by the web objects, then it may be acceptable to use a high level broad protection of the web objects. This eases administration and allows the Operations and Support Organization and Security Organization to focus on protecting the actual methods of the applications.

If however the web objects themselves contain data that is deemed critical or sensitive, then care shall be used on placing more precise and stringent access controls on the web object itself. As this increases the amount of overall critical objects to protect, it increases the amount of administration. In this instance, much more care should be taken in designing the layout of the web objects to ease the security administration on those objects.

An MA Developer needs to coordinate with the Operations and Support Organization and the Security Officer to:

- ◆ Identify what resources need to be protected

- ◆ The MA identifies the Servlets, JSPs, HTML pages, graphic pages, etc and the subdirectory structure related to their application.
- ◆ The Ops and Support Organization identifies the web server(s) that the MA will be hosted on and the parent directory for the MA. A *junction*, in WebSEAL, refers to a set of identically configured and maintained backend Web Servers that are logically grouped together under a common access point. The *junction name* associated with this *junction* is used in the URL to access the backend web server.
- ◆ The Operations and Support Organization will have to identify the parent directory, and/or an alias to the parent directory, for the MA. This is necessary, as the MA will most likely be sharing a web server with other MAs.
- ◆ The MA should design its application directories with security in mind. Grouping items with similar access control requirements under the same directory branch is preferred over mixing items with various requirements in the same directory branch. Policies and access control requirements will change over time, but planning, up front, can ease the long-term security administrative burden.

As indicated in Section 6.3.1 Planning for Access Control, the MA needs to identify what resources need to be protected and who needs to access them. This identification includes the initial access via the URL reference to the Servlets, JSPs, html pages, graphic pages, and the rest of the URL addressable objects.

The namespace that Policy Director uses to identify URL objects is the WebSeal namespace.

Table 34: Policy Director Object Space WebSEAL Branch

/	Root Policy Director Object Space Entry
/WebSeal	Beginning of all WebSeal protected resources
/WebSeal/<Server Group Label>	Each DECC installation will define at least one WebSeal server. Each group of identically configured WebSEAL servers will be identified by a <WebSEAL Server> label. By default this is the WebSEAL server name. This can be changed with the padmin” server modify <server name> baseurl <Server Group Label>” command.
/WebSeal/<Server Group Label>/<junction>	A junction is the name that will be used in the URL to reference a set of identically configured backend Web Servers. It is important to remember that there is one junction per web server set and not one per application running off the web server.
/WebSeal/<Server Group Label>/<junction>/<WebServer document roots and subdirectories, etc>	This is the start of the document root directories on the backend Web Servers. Policy Director dynamically queries this content for display purposes in its admin tool by essentially performing a directory list request.

See Figure 77: Example Policy Director Object Space WebSeal branch for an example of the hierarchy.

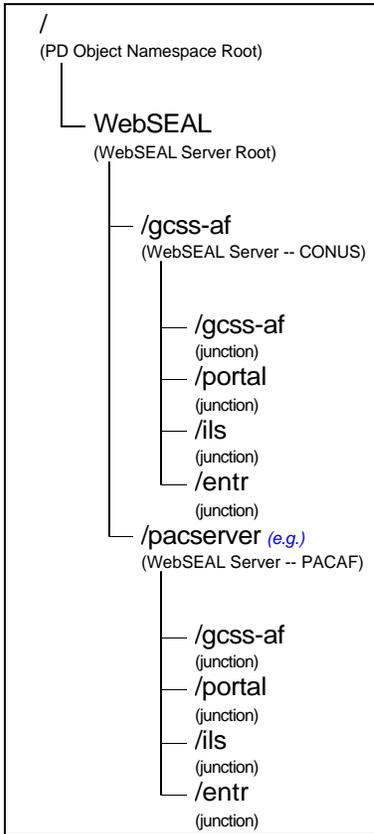


Figure 77: Example Policy Director Object Space WebSeal branch

The /WebSeal branch is fixed. The WebSEAL server name is never seen by the end user.

Create a Junction -The Operations And Support Organization will create a junction name is chosen to represent the content that the backend Web Server set that

6.4 Non-Repudiation

Non-repudiation encompasses services that ensure a user cannot deny an action taken by him/her or on his/her behalf, within the system. This includes providing capabilities for creating, verifying, and properly storing digital signatures.

Because digital signatures involve the use of pairs of signing keys (one private, one public), its use is predicated on the use of a Public Key Infrastructure (PKI). See Section 6.8 for information about PKI in the context of the IF.

Non-repudiation facilities typically include:

- Using digital signatures for (user) identification; can be used on email or individual documents.
- Providing a capability for software programs or processes to “sign” files or messages they generate on behalf of a user.
- Signing software to ensure it has not been tampered with. The IF does not police software signing. However, the *GCSS-AF System Security Policy* requires software delivered to GCSS-AF to be signed; see that document for details of the procedures that are required.
- Archiving signed items and the relevant certificates, to support later audits or legal inquiries.

Note Future Capability: Non-repudiation archiving is not provided, and is not currently planned due to lack of COTS capabilities and the immaturity of relevant standards in this area, but may be provided at a later date.

6.4.1 User Digital Signatures

Note Future Capability: User-level digital signatures are anticipated to be provided in a future release of the IF. This will be coordinated with the integration of the IF with DoD PKI in the Air Force.

Any requirement for users to sign forms or documents should be documented and tied to existing digital signature methods. Applications that have such requirements will need to select a COTS digital signature product that meets their requirements, and work with the GCSS-AF Enterprise Authority to integrate that product with the IF; or will need to write digital signature modules as part of their development effort. For example, they can use the digital signature services provided by the Java Cryptography Architecture (JCA) services included with the Java Development Kit.

Note: Digital signatures are often required on user email. However, the IF does not specify tools for the user’s desktop (beyond a web browser), so the IF does not address email signature capability. The Air Force has initiatives that extend to email signatures, and if an Application has a requirement in this area, the Application team should coordinate with the appropriate AF group(s).

6.4.2 Application Digital Signatures

Note Future Capability: Application digital signatures are anticipated to be provided in a future release of the IF. This will be coordinated with the integration of the IF with DoD PKI in the Air Force.

As described in Section 5, Business Object Documents (BODs) are the basis for inter-application communication in the IF.

A decision that should be made as BODs are being defined for an application is which BODs, if any, will require digital signatures. Basically, a digital signature should be used

when it is important to be able to verify the source of a BOD at a later point for legal, auditing, or financial accountability reasons. For example, a BOD that carries personnel information may need to be signed to satisfy Privacy Act considerations. Digital signatures should also be employed when it is important to ensure that the source of a BOD cannot later deny originating the BOD; this will generally not be an issue except where the BOD is tied to a human user as the (delegated) author. Digital signatures should NOT be used where these specific needs do not exist, because digital signature processing can be detrimental to the application's performance.

Any need for components to sign BODs or documents should be documented and tied to existing digital signature methods. Applications that have such requirements will need to select a COTS digital signature product that meets their requirements, and work with the GCSS-AF Enterprise Authority to integrate that product with the IF; or will need to write digital signature modules as part of their development effort. For example, they can use the digital signature services provided by the Java Cryptography Architecture (JCA) services included with the Java Development Kit.

6.5 Confidentiality

Confidentiality is the security characteristic that ensures that unauthorized individuals cannot view, modify, or delete data without appropriate authorization, and that proper protection is applied to data and code during both storage and transmission. Note that handling of authorization is covered in Section 6.3; this section will explain how the IF applies data protection methods.

In the IF, as in most secure systems, encryption is the primary mechanism for data protection. The IF expects infrastructure level protection to be applied to data files and static storage, and it specifies settings to ensure that data and code is protected while in use by application products; Section 6.5.1 describes this. The IF security architecture specifies at least one encryption mechanism for every off-box communication path in the architecture; see Section 6.5.2.

Because most of the encryption mechanisms specified in the IF are intrinsic to the infrastructure (i.e., they are determined by configurations or settings, rather than by software), developers seldom have to deal with encryption explicitly.

6.5.1 Protecting Static Data

For the purpose of this section, “static data” refers to the files used and created by the developer for their application. The intent of this section is to ensure that developers have the information necessary to develop software that does not compromise a secure configuration.

The requirements levied upon the developer may be found in the following DISA documents (in addition to the [GCSS-AF System Security Policy](#)):

- Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification, version 4.1, October 3 2000 [CM-38541]
- DII COE Windows NT Application and Kernel Developer's security Guidance, July 1999
- DII COE UNIX Kernel Developer's Security Guidance, June 1999
- UNIX Security Technical Implementation Guide, version 3 release 1, October 29 1999
- Database Security Technical Implementation Guide, version 3, release 1, October 30 1999

The DII COE documents are available from the DISA web sites:

http://diicoe.disa.mil/coe/coeeng/SECURITY_PAGES/securitydocs.html
<http://dod-ead.mont.disa.mil/cm/general.html>

The Security Technical Implementation Guides (STIGs) are available, for .gov and .mil sites, at <http://iase.disa.mil/>. Personnel such as DOD contractors who legitimately need access to this information but do not have a .mil or .gov address should contact the Field Security Operations Support Desk at DSN 570-9946, Commercial 717-267-9946, or e-mail to fso_spt@ritchie.disa.mil.

At a minimum, the developer should examine the appropriate STIG manual(s) that pertain to their platform (UNIX, NT, Database) as early as possible in the development phase. This will allow the developer time to work around any design issues that the STIGs may affect. For example, running a process with **setuid** set to root on UNIX is prohibited. If the application currently contains an executable with the **setuid** set to root, the developer shall either come up with another method of running the application or request a waiver from DISA to allow it.

It is recommended that the developed application be installed and verified on systems that have been locked down per the STIGs. This will provide the developer with the assurance that their application performs as expected in a secure environment and will also provide insight into problems that may occur during deployment. The actual STIG installation instructions shall be provided by the Mission Application SPO (i.e. obtained by the SPO from DISA) for the targeted DISA deployment sites (DECC-Ds). Note that the installation instructions may vary from site to site so it is recommended that all target sites be contacted, if possible.

Note: Even if the MA will be deploying servers at sites outside DISAs span of control (e.g. AF Bases), it is recommended that the MA still use the STIGs for locking down its servers. This will ensure a valid level of protection, and will also ensure consistency with other elements of the AF that use the IF at DISA facilities, thus lessening the chance of inter-operability problems (due to inconsistent port closures, for example).

In addition to the truly static data protection methods that are described above, the IF also provides for secure settings to be used in its application-support components, e.g. Component Broker (WebSphere EE). For example, the containers that EJBs are deployed in are configured to be secure in operation. These settings are described in the appropriate places in the IF Installation Guides (and in Section 5.2 and 5.3 of this Developer's Guide), as they are expected to be applied when the IF is deployed.

Developers who set up test environments (including development tools) should replicate these settings, so that development and testing is done in the same environment that the applications will run in. If an MA team discovers that one of these settings interferes with the operation of its application, it should notify the GCSS-AF Integrator, and work with the Integrator to resolve the issue. The MA team should NOT simply change such settings unilaterally!

6.5.2 Protecting Data Transmissions

Most data transmissions within GCSS-AF are encrypted via installation and configuration procedures. The diagram below (a copy of diagram 1 in Section 6.1) shows the various paths.

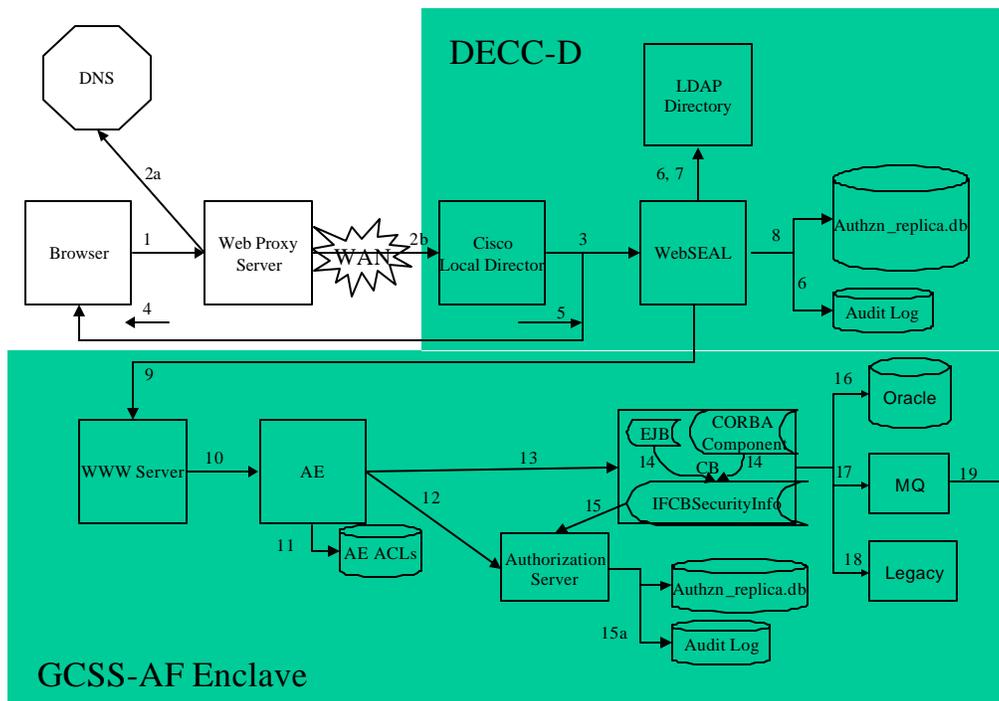


Figure 78: GCSS-AF Enclave Model I

See Table 35: Confidentiality Network Traffic Matrix below for a description of each transmission path between elements of the IF, and its encryption mechanism. Each path number in the diagram has a corresponding entry in Table 1. An exception that would fall under the responsibility of the Operations and Support team is the encryption between the client on the WAS AE platform and the WAS EE Component Broker application.

IMPORTANT: The Air Force has directed that all paths shall have an encryption mechanism available as part of the IF. However, for paths between servers that are located in the same DECC-D, the Air Force’s initial direction is that encryption shall not be turned on, due to the performance impact and the minimal risk to traffic that is totally enclosed in the DECC-Ds secure facility. Any traffic that may enter or leave the DECC-D (especially traffic to/from a user’s browser, and MQSeries traffic that is intended to handle inter-application communications) shall have encryption applied.

If this direction results in an MA requirement not being met, the MA development team needs to coordinate with the GCSS-AF Enterprise Authority to determine if the MA can use the existing IF accreditation, or if it needs a separate set of servers with encryption turned on inside the DECC-D. In addition, if the MA plans to deploy servers outside the DECC-Ds, the MA team will need to assess with Site Security Officer the risks that will result, and apply appropriate encryption to resolve those risks.

Table 1 indicates the approach to providing confidentiality for that path. Where the developer will need to take some action, the text below the table will describe what is required, in further detail.

Table 35: Confidentiality Network Traffic Matrix

Path	Encryption
Browser – WebSEAL	HTTPS WebSEAL server certificate (support for Netscape CA) Supports use of client certificates if available.
WebSEAL – Web Server	HTTPS WebSEAL server certificate Web Server server certificate
Web Server -- WAS AE Servlet Engine	No native encryption The OSE Redirector mechanism does not support encryption
WebSEAL – IBM SecureWay Directory	LDAPS WebSEAL server certificate LDAP server certificate
Policy Director Servers -- Policy Director Servers (includes WebSEAL – Security Management Server and Authorization Servers – Security Management Server)	DCE over SSL or DCE over GSS Policy Director Server Certificates

Path	Encryption
Policy Director Servers -- Policy Director Servers (includes WebSEAL – Security Management Server and Authorization Servers – Security Management Server)	SSL over TCP Policy Director Server Certificates Some communication between servers is socket based and not DCE based.
WAS AE Servlet Engine – IBM SecureWay Directory	LDAP (unencrypted) SPR written – deficiency in product for support of LDAPS
WAS AE Servlet Engine – Authorization Servers	DCE over SSL or DCE over GSS for Policy Database Updates SSL over TCP for access control checks. Both use Policy Director Server Certificates
WAS AE Servlet Engine – UDB (a.k.a. IBMs DB2)	No native encryption WAS AE configuration/metadata/ACLs are required to be in UDB.
WAS AE Servlet Engine – WAS EE Component Broker	HTTPS for Authentication Process WAS AE server certificate WAS EE server certificate GIOP for Other Traffic
WAS EE Component Broker – Authorization Servers	DCE over SSL or DCE over GSS for Policy Database Updates SSL over TCP for access control checks. Both use Policy Director Server Certificates
WAS EE Component Broker – DCE Cell Directory Service	DCE over SSL or DCE over GSS Policy Director Server Certificates
WAS EE Component Broker – DCE Oracle	SQL*Net via Oracle ANO Note: Oracle ANO doesn't use certificates for encryption. It uses an alternate encryption mechanism
WAS EE Component Broker – UDB (a.k.a. IBMs DB2)	No native encryption; see text below this table. Component Broker configuration/metadata is required to be in UDB. The GCSS-AF IF recommends that application data be stored in an Oracle server.
Policy Director Servers – DCE Cell Directory Service (CDS)	DCE over SSL or DCE over GSS Policy Director Server Certificates
WAS EE Component Broker – MQSeries MQM	N/A. Traffic isolated to same host.
MQSeries MQM – MQSeries MQM	DISA current practice is to establish VPNs between MQSeries MQMs that communicate across a WAN. The traffic is left in the clear within a LAN.

For all entries that mention “certificates”, the MA team will need to obtain and install certificates in its development test machines. The procedure for doing this is described in the IF Installation Guides. For deployment, the machine certificates will be obtained and installed by DISA in the DECC-D.

All other authentication/encryption mechanisms mentioned are enabled by applying the appropriate configuration settings as described in the IF Installation Guides, except for these paths:

Note Future Capabilities:

WAS AE Servlet Engine – IBM SecureWay Directory

As the table indicates, this path should have LDAPS applied. However, the current levels of these two products do not inter-operate via LDAPS. The IF team has submitted a Software Problem Report (SPR) to Tivoli, and Tivoli is working to resolve the issue.

WAS AE Servlet Engine – UDB (a.k.a. IBMs DB2), and WAS EE Component Broker – UDB (a.k.a. IBMs DB2)

(The AE-UDB path is not specifically shown on the diagram due to space limitations, but the mechanism would be similar to the one for CB-UDB, which is path 16 in the diagram.) There is no native encryption method that is inter-operable between these products. The IF team has determined that there are two basic methods that can potentially be used:

1. NetSeal over GSS Tunnel – a combination of NetSeal client on the AE or CB box, with NetSeal on the DB server, configured to use a GSS (or SSL) secure tunnel. NetSeal receives and decrypts the SQL request and passes it to the database engine. (This requires closing the TCP port (6720) on the DB server box, to ensure this path is the only one through which requests can reach the database.) Note that NetSeal supports a wider range of encryption algorithms, in particular Triple DES, over SSL than it does over GSS.
2. Third-party software – apply third-party software, such as that from HiT Software, that is specifically designed to protect DB2 traffic. This will also involve adding a client to the AE or CB box, and a server module to the DB box, and establishing a secure (SSL) tunnel via appropriate configuration.

Because DB2 is not the recommended application database under the IF, only limited testing of these solutions has been done. However, if an MA requires encryption for the use of DB2, we recommend setting up NetSeal using SSL – this uses existing IF components and supports stronger encryption algorithms than the GSS tunnel. (Note, though, that this recommendation is limited to NT. We expect to re-visit this issue in the future.)

MQSeries MQM – MQSeries MQM

The intent of the IF architecture is for Message-Oriented Middleware (MOM), specifically the MQSeries product, to be used for all inter-site communications. (See Section 5.) DISA has an existing Virtual Private Network (VPN) mechanism that encrypts and tunnels all MQSeries traffic to/from a DECC-D.

Refer to Section 6.2.2.2 MQSeries Message Queue Manager Channel Exits for the details regarding the authentication mechanism between two MQSeries MQM instances

6.6 Integrity

Integrity services are those that ensure system elements and data cannot be compromised or modified by illicit actions. The majority of the services and measures, such as boundary protection, required for Integrity is provided by the processing centers and base networks. System oversight and administration services and measures are provided by ESM package and by USAF and DISA operations and support. The IF Integrity capability provides only that information needed to tie into these external elements and with the IF supported security product base, the IBM Policy Director. In addition, the IF requires the proper application of DISA STIGs to the host servers and software. The sub-sections that follow provide a basic description of the integrity services the IF employs or ties to.

6.6.1 The GCSS-AF Enclave

The GCSS-AF security architecture calls for all GCSS-AF servers and supporting equipment (routers, hubs, etc.) to be guarded in a physically and logically distinct sub-net, or “enclave” at each server site. (Currently, all server sites are DISA DECC-Ds.) This allows the closest possible security policy to be applied at both the site boundary and the sub-net boundary. The policy shall, of course, provide for all communication paths and protocols required for IF traffic, but can restrict all other paths and protocols without interfering with the operation of other sub-nets that may be in operation at the same site.

The network setup for GCSS-AF is notionally depicted in Figure 79: GCSS-AF Enclave. (Other network configurations can validly be examined for fielding of the GCSS-AF Integration Framework. This one roughly corresponds to the current fielded configuration, however, except that there is currently only one DECC-D fielding the IF, DECC-D Montgomery.) Figure 79: GCSS-AF Enclave also provides an example of relevant portions of the DNS configuration (all URLs and IP addresses shown are examples, only).

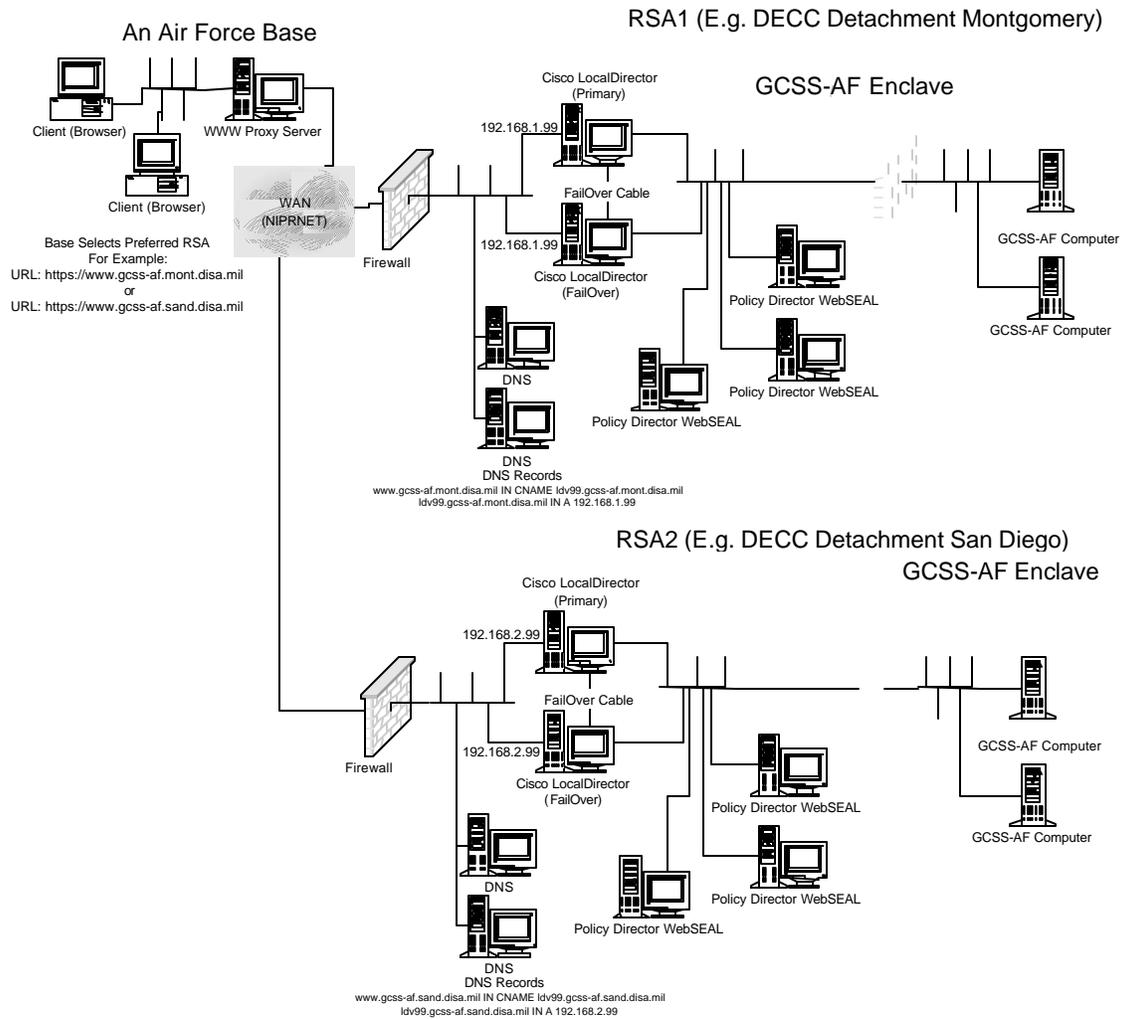


Figure 79: GCSS-AF Enclave Model II

Note that in the diagram, “Firewall” is used to depict the full perimeter network setup, including routers, hubs and switches, and intrusion detection tools, as well as the actual firewall itself. DISA currently uses PIX firewalls that are capable of supporting high-speed network lines, multiple security policies on separate sub-nets, etc. This boundary or perimeter, is set up in accordance with DISA standards that provide a level of protection similar to that specified by CITS BIP.

The DNS, Cisco Local Directors, and Policy Director WebSEALs are the IFs piece of the network infrastructure. They collectively provide for name-based lookups, re-direction and session maintenance to back-end servers, and web proxying. WebSEAL is also the point at which user authentication is performed and initial access checks to IF resources are done (see Sections 6.2 and 6.3). “GCSS-AF Servers” include both application and security servers; their integrity is assured by a combination of the perimeter and IF infrastructure, and by their own on-box configuration.

MA developers do not need to be directly concerned about details of this setup unless they need to add boxes to the enclave. The basic cases for this are as follows:

1. Add a server with non-IF COTS products, especially if the products provide function already provided by an IF product or component, or if the box uses a different operating system platform than the IF servers. Deployment of this server will have to be coordinated with the GCSS-AF Enterprise Authority, and with DISA (especially see Section 6.6.3 below, for firewall issues that need to be considered). Deployment of this server will likely require a separate Certification and Accreditation activity, which can be time-consuming and expensive.

Depending on the combination of COTS products on the server, it may represent an additional level of risk to the enclave as a whole (e.g. if it requires opening a wide range of ports that are otherwise un-needed in the enclave, or if it requires a non-secure protocol to be used). This option is therefore NOT recommended.

2. Add a server with IF COTS products (e.g. WebSphere, MQSeries) and MA software or data that relies on those products. Deployment of this server should be easier to coordinate through the GCSS-AF Enterprise Authority, and should not require a separate C&A activity (i.e. the existing accreditation should be able to be updated, with less effort than obtaining a new one).
3. Add MA software to an existing IF server. The easiest case to coordinate through the GCSS-AF Enterprise Authority (at least as far as technical issues are concerned).

6.6.1.1 Server Integrity

All IF servers deployed to a DECC-D will have the appropriate Security Technical Implementation Guide (STIG) lockdowns applied. These are instructions designed to apply the maximum level of protection to resources on that individual box, including port, folder, file, and registry protections. DISA also applies certain standard anti-tampering tools to each server at the same time.

DISA is responsible for applying the STIGs to servers as they are deployed. However, developers should be aware of what the STIGs require, as they can affect both performance and function of the server, especially of some COTS products. See Section 6.5.1 Protecting Static Data.

6.6.2 The DMZ

The GCSS-AF security architecture provides for a “demilitarized zone” or DMZ to be established at a server location, by deploying a second, internal, firewall (at the point denoted by the gap in the lines inside the GCSS-AF enclave in the diagram.) This serves to provide a second layer of protection to the servers sitting behind the inner firewall, by

locking down the inner firewall more tightly to open only the specific ports and IP addresses needed to allow boxes in the DMZ to communicate with boxes in the enclave. The enclave would then receive no traffic except that which originates in the DMZ. (The exception would be the servers that support MQSeries, but as explained in Section 6.5.2, these servers have VPNs applied to protect their traffic.)

The external firewall allows end users from virtually anywhere to access the WebSEAL hosts using the HTTPS encrypted protocol. All other protocols to the WebSEAL hosts would be blocked. The external firewall would also block any attempt to access any of the other GCSS-AF hosts using any protocol.

The existence of the internal firewall provides additional protection by only allowing the WebSEAL hosts in the Network DMZ:

1. To only access the GCSS-AF web servers using HTTPS
2. To only access the LDAP directory (Authentication hosts, typically configured on the Authorization servers) using the LDAPS protocol
3. To only access the Security Management Server (Security Master Server) using DCE

The benefit of this is that if the WebSEAL servers were to be compromised, the attacker would only have limited holes to export on the GCSS-AF servers on the internal network.

The internal firewall can be removed, and in fact has been removed from the initial fielding configuration. This is because firewalls can significantly affect performance. (It was determined, in consultation with DISA and the GCSS-IF SPO, that the PIX firewall supported by DISA for the perimeter provides sufficient protection that an internal firewall and DMZ should not be required. In particular, the PIX firewall can support applying separate security policies to separate sub-nets behind it, thus allowing all GCSS-AF servers to be protected as an enclave, as described in Section 6.6.1.)

6.6.3 Firewall Considerations

Computer network perimeter security, using a mixture of firewalls and routers, is mandated at every USAF base and DISA installation throughout the world. To avoid problems with the deployment of their applications, Application Developers working on tasks that run on the GCSS-AF Integration Framework should have a basic understanding of network perimeter security. They should also have an understanding of the procedures to use to determine if they will have any additional security requirements for deployment and what to do if there are.

6.6.3.1 Roles in Perimeter Security and Security Policy

Routers provide the capability to help secure the perimeter of a protected network. They are used to direct and control much of the data flowing across computer networks. They can be configured to control access, resist attacks, shield other network components, and even protect the integrity and confidentiality of network traffic. Firewalls help secure the protected network by filtering network traffic and dropping unwanted packets. They use filtering rules configured by the network administrator to restrict incoming packets to specific domains or specific networks. They can also filter traffic based on the destination address or port of the packet, thus limiting packets to systems in the protected enclave to ports recognized and accepted by the network security administrator. Firewalls have become a single point of network access where traffic can be analyzed and controlled according to parameters such as application, address, and user, for both incoming traffic from remote users and outgoing traffic to the Internet.

A router can also be used as part of defense-in-depth approach as shown in the diagram below. It acts as the first line of defense and is known as a screening router. It contains a static route that passes all connections intended for the protected network to the firewall. The firewall provides additional access control over the content of the connections.

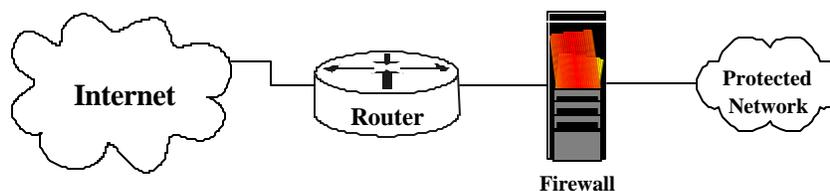


Figure 80: Typical One-router, One-firewall Internet Connection Configuration

Another approach is to position one router at the connection between the local premises and the Internet, and then another router or firewall between the existing firewall and the protected network. This configuration offers two points at which policy can be enforced. It also offers an intermediate area, often called the de-militarized zone (DMZ) between the two routers. The DMZ is often used for servers that shall be accessible from the Internet or other external network.

6.6.3.2 Obtain Application/COTS Product Port Numbers

Application Developers shall be aware of any network ports their application uses as well as any ports used by the COTS products their application requires, if any. The port(s) used by their application will be known to them, but those used by the COTS products shall be obtained from either the vendor's documentation or from their technical support help desk. The only way to truly validate the ports being used by the application is to perform a port analysis on the network while the test procedures are being run. The port analysis for IF 2.1 is available in the [IF 2.1 Software Installation Procedures](#).

6.6.3.3 Register Port Numbers

Once the application's ports numbers are all identified, they shall be registered with the DoD at:

<http://dod-ead.mont.disa.mil/qry/internet/Register/Port.htm>

The following text is from this site:

“Register Socket and ID Information

NOTE: Per I&RTS policy, ALL DoD programs (i.e., DII COE, DoDIIS, AGCCS, AFGCCS, GCCS, GCCS-T, and GCSS) shall register segment-related information as early as possible in the development cycle.

All DII COE TCP Numbers are required to be registered/verified if:

1. If the developer is using a well know port (Range 0 - 1023 Well Known Ports) verify the port number and name and be sure to use the port correctly. The developer SHALL NOT use a "well known port 0 -1023" or its name, for a Non-standard use.
2. If the developer is using a port in the Range 1024 - 49151 (Registered Ports)
3. If the developer is using a port in the Range 49152 - 65535 (Dynamic and/or Private Ports) an email shall be sent to cm@ncr.disa.mil with the port Name, Number and Range.

If the segment requires associated TCP/UDP sockets, the developer shall register the TCP/UDP Sockets. Socket registration is not required for all segments. The developer may also view as list of sockets that have already been assigned.”

6.6.3.4 Open Ports at Installation Sites

After registering the required ports with the DoD, the developer shall contact the appropriate personnel at DISA in order to obtain permission to open the required port(s) in the network perimeter security at the required installation sites. It is recommended that the developer work with their program contracts office to formalize this requirement with their customer. If the application requires access at additional DISA sites, these sites shall also be notified of the port requirements.

6.6.3.5 Validate Port Usage

The only way to truly validate the ports being used by the application is to perform a port analysis on the network while the test procedures are being run. The port analysis for IF

2.1 is available in the IF 2.1 Software Installation Procedures. If a firewall is available at the development site, it is recommended that the application be tested with only the required ports being allowed through. To ensure a successful deployment, it is also recommended that the port analysis be performed as early in the test phase as possible to allow for the customer to be formally notified and for the appropriate DISA and DoD arrangements to be made.

The Integration Framework currently requires access through the network perimeter security for the following ports:

HTTPS	443
LDAPS	636
MQSeries	1414 and 1415
DCE	49152(A currently unspecified number of dynamic ports above)

6.6.4 Multi-Level Security

Note Future Capability: Multi-level security (MLS) refers to a system requirement that the system be capable of securely interchanging data between security levels (e.g. between Secret and Unclassified areas of a network). Achieving multi-level security requires applying specific tools and techniques, and would result in a number of changes to the IF security architecture. Currently, the IF does not have a requirement for MLS, and therefore does not yet support MLS. It is possible that a future release of the IF will provide MLS capabilities. In the meantime, if an MA has MLS requirements, it will need to resolve those requirements itself. The simplest approach to this that still could gain from the IF, would be to deploy separate IF-based enclaves for Secret and Unclassified operation, with a MA-selected “Guard” product to govern traffic between them.

6.7 Audit and Alarms

This category encompasses services that record information about activities taken inside the system, whether by human users or software components, and the storage and analysis of those records. This includes creation of audit records, storage of audit records, creation of audit reports, generation of dynamic on-line alarms, and analysis of events and records (whether at runtime or post-mortem). It also provides services to define and administer audit policies, as well as the technical features needed to implement the policies. The current IF audit provisions are limited to the facilities provided by the supported security product, IBM Policy Director and IF Log Services to record system actions.

Note Future Capability: The current IF does **not** provide audit reduction or alarm posting. Alarm and alert posting is to be provided at the DECC-D

processing centers using the Tivoli Management System capabilities employed by DISA. Processing centers that do not utilize this Tivoli capability should plan on implementing the same capability as DISA at a DECC-D. A future IF release should subsume Tivoli under the Enterprise Systems Management capability. Audit reduction and reporting capabilities are anticipated to be provided in a future IF release.

This document defines the approach to raising and handling of **Security alerts** and **alarms** for GCSS-AF. It is not intended to define or limit the approach to **Application defined alerts** or **alarms**. (Some similarity is to be expected, however, because of the common use of Tivoli products.)

For developers of applications that consist of static HTML links:

- **If the application does not require access checks (other than against its main URL), OR if the developer is using the IFs access check methods as described in Section 6.3, then extra security audit/alert checks do not need to be added** (UNLESS the developer has special auditing requirements that go beyond those described in the GCSS-AF System Security Policy (i.e. standard C2-level auditing). In this case, they will not need to take any actions (log files are generated by the IF COTS tools), but should read this section for background.
- **If the application does its own access checks, the developer is responsible for logging and/or raising alerts for events surrounding those access checks.** They may use the **log4j** tool or their own, to perform the logging actions. The developer should read this section for background and coordinate with the GCSS-AF Enterprise Authority to determine log contents (severity levels, etc.) as discussed briefly below.

6.7.1 Logging Framework Security Events

6.7.1.1 Background

An integrated Enterprise System Management tool with mechanisms for handling security auditing and alarms is not currently provided by the IF. **Therefore, the IF current release, by agreement with the AF, will employ an interim approach for security audits and alarms, intended to minimize expenditures for code and integration that will be superseded when this capability becomes available. This section describes this interim approach.**

6.7.1.2 Executive Summary of the Approach

Security auditing will be based on the log files generated by Policy Director. The WebSeal and Authorization Server components of Policy Director are the primary originators of log files for security purposes. Currently, there are no modifications made to the native COTS log file messages or formats. Mission Applications (MAs) can also

write to log files (using the IF Logging Service and the **log4j** open source tool) as they detect conditions that may require attention; MA engineers can determine the message content, so long as the messages do not exceed 256 ASCII characters.

Each message written to a log file will be characterized by a severity level. We use the levels DISA employs; **FATAL, CRITICAL, MINOR, WARNING, and HARMLESS**, as our default levels. Where a COTS product produces messages that do not fit these levels, we will map the COTS messages to these levels (see below). MAs are expected to tailor their error messages to these levels as part of their engineering effort.

On the Tivoli side, we will ultimately use Log File Adapters (LFAs) to convert the log file entries into input to the Tivoli Enterprise Console (TEC) by way of Tivoli agents. The TEC will display the actual alert or alarm. Log File Adapters are tailored to the format of each type of log file, and can perform filtering of messages from that file. Log File Adapters are not expected to perform audit reduction or correlation of messages from multiple log files. Log File Adapters are rule-driven. LMSI-O and the AF will develop the rules for handling IF-COTS-generated log messages, which are to be embedded in the LFAs. The development and delivery of LFAs has been postponed to a later release of the IF.

Rules for handling MA-generated log messages are the responsibility of each MA engineering team. The rules will determine what messages the LFA should pass on to Tivoli, and what parameters (e.g. severity) should be associated with them.

Assumptions :

1. Alerts and alarms will be displayed on the Tivoli ESM console (TEC). The Tivoli ESM product(s) are being used “as is” and treated as a black box for purposes of IF design. Tivoli Management Environment (TME) supports assigning actions to messages; these can range from simply displaying the message on the Tivoli console, to generating audible alarms, email, executing a recovery procedure, etc.
2. Mission Applications (MAs) are expected to use the **log4j** open source package to generate log file entries as needed. Each MA engineering team will need to determine, in conjunction with LMSI-O and the AF, what errors or conditions to log, and what severity or priority to assign to each one.
3. Legacy systems connected to the IF will be expected to use a wrapper or interface component (IC) that is GCSS-AF compliant. The wrapper or IC can generate log files like MAs do, that can result in audit records (or ultimately, alarms) being raised through the IF as described in this document, or the Legacy system can handle audit requirements through its normal mechanism(s) outside of the IF.
4. Due to resource constraints, the IF will not attempt to use non-ASCII text logs as the source for alert/alarm generation. This will preclude the use of DCE audit trail files, in particular, as these are in binary format.

5. DISA will administer Tivoli ESM, with guidance from USAF and LMSI-O regarding GCSS-AF needs.
6. DISA will be responsible for correlating messages to isolate errors, perform audit reduction, etc., at least for now. As indicated above, Log File Adapters are not expected to do this, so it will be primarily a manual process.
7. The primary COTS-generated log files are expected to be those from Policy Director's WebSeal and Authorization Server components. Initial analysis indicates that while WebSphere AE and EE can generate log files, their content is less useful for security purposes and less readily amenable to writing rules for Log File Adapters. In addition, it was agreed that MQSeries logging would be postponed to a later IF release, to allow the use of the upcoming Policy Director for Messaging component. Therefore, the later IF integration will address log files generated by these products; this effort is expected to track to IBM's own integration of Policy Director, Tivoli, and WebSphere products.

6.7.1.3 Generating and Reading Log Files

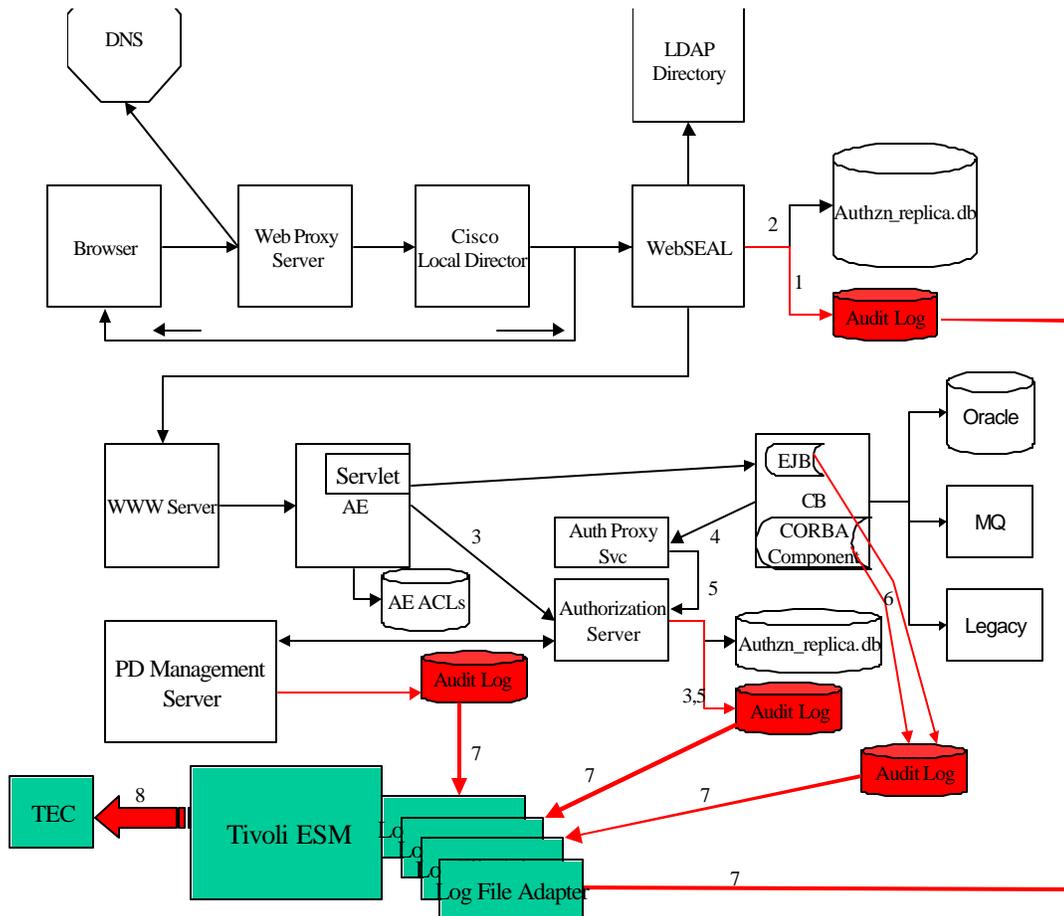


Figure 81: Model of Log File Generation

The main steps involved in generating and using log files shown in this diagram are explained below:

1. When a user initially requests a URL associated with GCSS-AF, they are asked to log in, and WebSeal performs an authentication. For IF 2.1, this involves a standard **ID+password** authentication. The success or failure of this authentication process is logged.
2. After a successful authentication, WebSeal checks the users authority to access the original URL request using the user's credential against the authorization policy database.
3. If the Audit permission on the Access Control List (ACL) for this object was turned on, then the success or failure of the access check is logged on the WebSeal server. For IF 2.1, the Audit permission **will** be turned **on** in the ACL for all URLs that is part of the GCSS-AF menu hierarchy.

4. If the user passes the authorization check, WebSeal forwards the user's original URL request to the Servlet. This forwarded request will include the user's credential information in the HTTP header.
5. When the forwarded request reaches it, the Servlet for the function requested by the user extracts the credential information supplied by WebSeal in the HTTP header from the **HTTP_IV_CREDS** parameter. The Servlet checks the permissions of the credential to perform a specific action by performing an authorization check to a Policy Director (PD) Authorization Server. First it checks whether the user can invoke the Servlet itself, and then it checks against any specific methods that the Servlet invokes that require separate authorization decisions.
6. The PD Authorization Server checks whether the credential-holder is allowed to perform the requested action (method) on the object attempting to be accessed, and sends a decision back to the Servlet. Additionally, the Authorization Server will log the access checks success or failure to the audit log file.
7. If the user has access to invoke the Servlet and the specific object and method, then the Servlet programmatically logs in (authenticates) to the WebSphere Application Server Enterprise Edition (WAS-EE), a.k.a. Component Broker (CB), via a Servlet Login Helper. The Servlet communicates to Component Broker over SSL. Component Broker takes the *userID* and *password* supplied by the Servlet Login Helper and logs in to Policy Director's DCE Cell. If the user does not successfully authenticate, the Servlet can send a page to the user indicating a system error and to retry their request. If the Servlet Login Helper successfully authenticates to PD (DCE), then the Servlet invokes the specific EJB/CORBA object. In either case, the success or failure of the authentication is logged. The Servlet passes the credential supplied in the BA Header by WebSeal as a parameter to the methods invoked in the object.
8. The EJB/CORBA object checks the permissions of the credential to access the EJB/CORBA object and the specific method by performing an authorization check to a Policy Director (PD) Authorization Server. First the EJB/CORBA object checks whether the user can invoke the EJB/CORBA object itself, and then it checks against any specific methods that the EJB/CORBA object invokes that require separate authorization decisions.
9. The PD Authorization Server checks the credential and the action requested against the Access Control List (ACL) of the object attempting to be accessed and the PD Authorization Server sends a decision back to the requesting EJB/CORBA object. The Authorization Server logs the result of this authorization check.

10. If the user is not authorized, the EJB/CORBA object should return an error code to the Servlet indicating that the user was not authorized to perform the requested function. (The details of handling this error are an MA programming decision. The Servlet can return an application specific error message or a generic page that would be created for unauthorized access.)
11. If the user is authorized, the EJB/CORBA object performs the action. If errors are encountered in the course of performing the action, the EJB/CORBA object can log them. The error log entry's content and severity is dependent on the engineering of the Mission Application of which the EJB/CORBA component is a part.
12. **Note Future Capability:** The Log File Adapter detects a change in the log file(s) it is monitoring. The adapter pulls the log file contents and applies its rules to the contents to determine what (if any) messages need to be passed to Tivoli ESM for action.
13. Tivoli ESM receives the message(s) from the Log File Adapter. It applies its configuration settings to them, to perform whatever action(s) are associated with the message(s) received. If the settings indicate that an alert or alarm is to be raised, Tivoli ESM passes the alert/alarm to the TEC for display.

Note that the primary sources for auditing are the log files generated by elements of Policy Director. The main source of information about these log files that is publicly available is the Policy Director Administration Guide (especially Chapter 12; this section includes material from that chapter, tailored to the GCSS-AF IF).

6.7.1.3.1 IF Test Components (EJB and CORBA Components)

These components are supplied as examples of how Mission Applications may be engineered to supply error information. They are **not** intended as definitive sources of error information. It is expected that each MA engineering team will work with LMSI-O and DISA to determine the appropriate types of errors to be logged by that MA, and how they are to be categorized, including whether they will result in alerts or alarms.

Where handling of errors is not constrained by specific MA requirements, it is expected that they will be handled according to IF-supplied design patterns (see the GCSS-AF Rose Model).

6.7.1.4 Prioritizing Log Entries

As indicated above, DISAs prioritization scheme will be used for the IF. This scheme is explained in this table:

Table 36: Explanation of DISA Prioritization Scheme

T/EC Event Severity	Significance	Event Seen By
FATAL	A resource has failed and is currently not operational. Administrators should respond immediately and continue working until the problem is resolved. These events will be forward to other sensory grid for other locations, such as Neurastar.	Help Desk SA\DBA
CRITICAL	A resource is near failure. SA\DBA should respond immediately to ensure no degradation of service soon.	Help Desk SA\DBA
MINOR	A resource is in highly cautious condition. SA\DBA should respond and closely monitor to ensure no degradation of service.	SA\DBA
WARNING	A resource is in cautious condition. SA\DBA should closely monitor or proactively respond to the problem.	SA\DBA
HARMLESS	A resource has returned to its normal state.	SA\DBA

The following table shows an initial recommendation for prioritizing events based on the IF use cases. Note that not all of these events can be captured directly from the Policy Director audit trail files or log files, and thus may require manual correlation. It is expected that later releases will support more automated recognition and prioritization of events.



Table 37: Recommendation of Events for IF Use Cases

TRIGGER EVENT		SUB-EVENT	RECOM MEND SEVTY.	RECOMMEND ALERT GEN'D?	RECOMMEND DATA LOGGED	RECOMMEND DATA ALERTED
#	Description					
2.5.1	Logon Normal Path	Success	Harmless (Warning if is for a privileged user account, e.g. "root")	No	Client Date/Time Server Date/Time Logger Date/Time Unique Subj. ID Type of Event Success/Fail Origin of Request & Destination of Request (IPs) Name of Program or File Introduced (<STD>)	
		Successive Failures	Minor (Critical if it is for a privileged user account, e.g. "root")	Yes – Could be a hack attempt.	" + Number of tries Number of tries from each specific Workstation, total number of retries (May be a job for reduction tools) (<Retry Attempts>)	<STD> <Retry Attempts>
2.5.2	Logon Problem w. Establishing SSL	Bad Record MAC – A record is receive with an incorrect MAC	Critical	Yes – Could be an attempt at cracking the SSL Session.	<STD> <Retry Attempts> Certificate Information: Client dn, Server dn, algorithms(encryption, compression, key exchange) (<Certificate Information>)	<STD> <Retry Attempts> <Certificate Information>
		Decompression Failure – The decompression function received improper input.	Warning	Most SSL implementations do not use compression	<STD> <Retry Attempts> <Certificate Information>	



	Handshake Failure – The sender was unable to negotiate an acceptable set of security parameters given the options available.	Critical	Note: a hacker could be trying all of the servers to see which one's will use a lower level security	<STD> <Retry Attempts> <Certificate Information>	<STD> <Retry Attempts> <Certificate Information>
	No Certificate – No appropriate certificate is available.	Minor or Critical	Note: This could occur when a server doesn't know the certificate authority of the client's certificate.	<STD> <Retry Attempts>	<STD> <Retry Attempts>
	Bad Certificate – A certificate was corrupt, contained signatures that did not verify correctly, could not authenticate chain (i.e. no common or cross-certified certificate authority)	Minor or Critical	Yes - May indicate tampering	<STD> <Retry Attempts> <Certificate Information>	<STD> <Retry Attempts> <Certificate Information>
	Unsupported Certificate – A certificate was of an unsupported type.	Warning	No-The certificate may be a back level or future level certificate	<STD> <Retry Attempts> <Certificate Information>	
	Certificate Revoked – A certificate was revoked by its signer, either the end user's certificate or on the CAs in the chain. The certificate or one of the certificate authorities in the chain is listed on a Certificate Revocation List or Authority Revocation List.	Minor	Yes - May indicate an attempt at using an invalid server certificate. More likely indicates need to update a server's certificate.	<STD> <Retry Attempts> <Certificate Information>	<STD> <Retry Attempts> <Certificate Information>
	Certificate Expired – A certificate has expired or is not currently valid.	Minor	Yes	<STD> <Retry Attempts> <Certificate Information>	<STD> <Retry Attempts> <Certificate Information>



		Certificate Unknown – Some other (unspecified) issue arose in processing the certificate, rendering it unacceptable.	Minor	Yes	<STD> <Retry Attempts> <Certificate Information>	<STD> <Retry Attempts> <Certificate Information>
		Illegal Parameter – A field in the handshake was out of range or inconsistent with other fields.	Critical	Yes - May indicate tampering, should not be a regular occurrence	<STD> <Retry Attempts> <Certificate Information>	<STD> <Retry Attempts> <Certificate Information>
2.5.3	Logon - Problem Establishing GCSS-AF Session		Minor	Yes – May indicate an attempted session spoofing attack, or may indicate a server communication path has been interrupted	<STD> <Retry Attempts> <Certificate Information> Session ID, other Session Information (<Session Information>)	<STD> <Retry Attempts> <Certificate Information> Session ID, other Session Information (<Session Information>)
3.5.1	Logoff Normal Path	Success	Harmless	No	<STD> <Session Information> <Certificate Information??>	
		Failure	Critical or Minor	May indicate an inconsistency in the system's state.	<STD> <Session Information> <Retry Attempts> <Certificate Information>	<STD> <Session Information> <Retry Attempts> <Certificate Information>
3.5.2	Logoff Invalid Certificate		Critical or Minor	Yes	<STD> <Session Information> <Retry Attempts> <Certificate Information>	<STD> <Session Information> <Retry Attempts> <Certificate Information>
3.5.3	Logoff User Profile Deleted		Warning	No-Occurs as a result of administrative actions.	<STD> <Session Information> <Retry Attempts> <Certificate Information>	
3.5.4	Logoff User Profile Altered to Disallow Logon		Warning	No-Occurs as a result of administrative actions.	<STD> <Session Information> <Retry Attempts> <Certificate Information>	
3.5.5	User Inactivity		Harmless	No	<STD>	



	Timeout				<Session Information> <Certificate Information>	
4.5.1	Configure User Profile Normal Path	Success	Harmless	No	<STD> <Session Information> <Certificate Information>	
		Failure	Critical	Yes-May indicate attempt at tampering with a user profile.	<STD> <Retry Attempts> <Session Information> <Certificate Information>	<STD> <Retry Attempts> <Session Information> <Certificate Information>
4.5.2	Configure User Profile Not Lockable for Write Access		Warning	No – Probably indicates another admin is using the profile	<STD> <Session Information>	
4.5.3	Configure User Profile Remove Profile when None Exists		Critical	Yes – May indicate another admin has deleted the profile, OR that an illicit user has tampered with the profile database	<STD> <Session Information>	<STD> <Session Information>
4.5.4	Configure User Profile Update Failed		Critical	Yes-May indicate attempt at tampering with a user profile.	<STD> <Retry Attempts> <Certificate Information> <Session Information>	<STD> <Retry Attempts> <Certificate Information> <Session Information>
5.5.1	Configure Organizational Profile Normal Path	Success	Harmless	No	<STD> <Session Information> <Certificate Information>	
		Failure	Critical	Yes-May indicate attempt at tampering with an organizational profile.	<STD> <Session Information> <Retry Attempts> <Certificate Information>	<STD> <Session Information> <Retry Attempts> <Certificate Information>
5.5.2	Couldn't lock Organizational profile for write access		Warning	No – Probably indicates another admin is using the profile	<STD> <Session Information>	
5.5.3	Remove when there		Critical	Yes – May indicate	<STD>	<STD>



	are no organizational profiles			another admin has deleted the profile, OR that an illicit user has tampered with the profile database	<Session Information>	<Session Information>
5.5.4	Organizational Profile Update Failed		Critical	Yes-May indicate attempt at tampering with an organization profile.	<STD> <Retry Attempts> <Certificate Information> <Session Information>	<STD> <Retry Attempts> <Certificate Information> <Session Information>
6.5.1	Configure Security Label Use Normal Path	Success	Harmless	No	<STD> <Session Information>	
		Failure	Critical	Yes-May indicate attempted label tampering	<STD> <Session Information> <Retry Attempts> <Certificate Information>	<STD> <Session Information> <Retry Attempts> <Certificate Information>
7.5.1	Single Sign-On Normal Path		Harmless	No	<STD> <Retry Information> <Certificate Information> <Session Information>	
7.5.2	SSO User Lacks Access to Perform Function		Minor or Critical	Yes-May indicate an attempt at hacking SSO	<STD> <Retry Information> <Certificate Information> <Session Information>	<STD> <Retry Information> <Certificate Information> <Session Information>
7.5.3	SSO System is Unable to log into Necessary Legacy Applications		Minor	Yes-for administering accounts	<STD> <Retry Information> <Certificate Information> <Session Information>	<STD> <Retry Information> <Certificate Information> <Session Information>
4.1.5.1	Initial Load of Parts, Normal Path		Harmless	No	<STD> <Certificate Information> All Part Attributes & Security Information (<Part Information>)	
4.1.5.2	Alternate Path – Message (BOD) Format Mismatch		Warning	Yes – Could indicate tampering with an application or an	<STD> <Part Information> <Retry Attempts>	



	With Standard (DTD)			attempted session spoof, or simply a coding problem	BOD Source, BOD Destination (<BOD Information>) Mismatched fields of DTD	
4.1.5.3	Alternate Path – Receiver (PDC) Inoperable		Minor	Yes – Could indicate tampering, or just a server is down	<STD> <Retry Attempts> <Part Information> <BOD Information>	
4.1.5.4	Alternate Path – Loss of Message Transport Mechanism (Queue Manager)		Critical	Yes – Could indicate several types of attack	<STD> <Retry Attempts> <Part Information> <BOD Information> Queue Manager Name	<STD> <Retry Attempts> <Part Information> <BOD Information> Queue Manager Name
4.1.5.5	Alternate Path – Loss of Queue Channel		Critical	Yes	<STD> <Retry Attempts> <Part Information> <BOD Information> Queue Name	<STD> <Retry Attempts> <Part Information> <BOD Information> Queue Name
5.1.4.1	Use Part - Normal Path		Harmless	No	<STD> <Certificate Information> <Part Information>	
5.1.4.2	Alternate Path – Port Access Denied by Firewall		Critical	Yes (If port access denial is understood by application)	<STD> <Retry Attempts> <Part Information> <BOD Information>	
5.2.4.1	Update Part - Normal Path		Harmless	No	<STD> <Certificate Information> <Part Information>	
5.2.4.2	Alternate Path – Subscriber (base PDC) Inoperable		Critical	Yes-Base Admin. Should be notified of Service Losses	<STD> <Retry Attempts> <BOD Information>	<STD> <Retry Attempts> <BOD Information>
5.2.4.3	Alternate Path – Loss of Message Transport Mechanism		Critical	Yes-Administrator should be notified of Service Losses	<STD> <Retry Attempts> <Part Information> <BOD Information> Queue Manager Name	<STD> <Retry Attempts> <Part Information> <BOD Information> Queue Manager Name



5.2.4.4	Alternate Path – Loss of Queue Channel <i>(Test Case Only)</i>		Critical	Yes-Administrator should be notified of Service Losses	<STD> <Retry Attempts> <Part Information> <BOD Information> Queue Name	<STD> <Retry Attempts> <Part Information> <BOD Information> Queue Name
5.2.4.5	Alternate Path – Unauthorized Application Publish Request		Critical	Yes-Could indicate attempt at introducing rogue data	<STD> <Retry Attempts> <Part Information> CORBA Publishing Object Name, Publishing Data, Naming Service Used, Other CORBA data (<CORBA Information>)	<STD> <Retry Attempts> <Part Information> <CORBA Information>
5.3.4.1	Delete Part - Normal Path		Harmless	No	<STD> <Certificate Information> <Part Information> <BOD Information>	
5.3.4.2	Alternate Path – Subscriber (base PDC) Dies After Message Retrieval		Minor	Yes-Base Admin. should be notified of Service Losses	<STD> <Retry Attempts> <CORBA Information>	<STD> <Retry Attempts> <CORBA Information>
5.4.4.1	Normal Path - Order Part		Harmless	No	<STD> <Certificate Information> <Part Information> <BOD Information>	
5.4.4.2	Alternate Path – Digital Signature Error		Minor	Yes-Could indicate attempt at message tampering	<STD> <Certificate Information> <BOD Information> <Retry Attempts> <Part Information>	<STD> <Certificate Information> <BOD Information> <Retry Attempts> <Part Information>
5.4.4.3	Alternate Path – Transaction Failure, Requisition Message Processing		Minor	Yes-Application Administrator should be notified of transaction failures	<STD> <BOD Information> <Retry Attempts> Requisition Information	<STD> <BOD Information> <Retry Attempts> Requisition Information



5.4.4.4	Alternate Path – Transaction Failure, Part (PDC) Persistence Update		Minor	Yes-Application Administrator should be notified of transaction failures	<STD> <BOD Information> <Retry Attempts> Persistence Information	<STD> <BOD Information> <Retry Attempts> Persistence Information
5.4.4.5	Alternate Path – Transaction Failure, Order Database Addition Failure		Minor	Yes-Application Administrator should be notified of transaction failures	<STD> <BOD Information> <Retry Attempts> Order Information Database Information (Server, Listening Port, Instance, SID, Table(s), Operation)	<STD> <BOD Information> <Retry Attempts> Order Information Database Information (Server, Listening Port, Instance, SID, Table(s), Operation)

6.8 PKI and Key Management

Public Key Infrastructure (PKI) is a method of providing authentication and encryption using paired public and private keys, and a “certificate”. DoD is in the process of designing and rolling out a PKI structure to cover all personnel and systems under DoD command. The Air Force is participating in this process, and the IF will integrate with the resulting structure over time.

This section documents services relating to the use of keys, especially those relating to PKI certificates and keys. As USAF is responsible for the issuance and management of certificates, this package only addresses those services dealing with certificate and key retrieval, utilization, and protection within GCSS-AF. Where PKI is employed on the servers, the IBM Global Security Kit provides key protection.

6.8.1 User PKI

The current implementation of the IF provides support for the use of user certificates for the purposes of authentication and confidentiality. The process of authentication and confidentiality are provided by the client’s browser and by the Policy Director WebSEAL server. It is necessary to have the Certificate Authorities of any valid clients to be in the IBM Global Security Kit key database on the Policy Director WebSEAL server for client certificate validation. This process needs to be repeated for each WebSEAL server in the configuration. If the WebSEAL server’s certificate is not in the client’s browser as a site certificate or if the Certificate Authority of the WebSEAL server is not in the client’s browser as a signer certificate, then the user will be prompted if they wish to accept the server’s certificate for the session or forever. If the user selects forever, then the server’s certificate is added to the site certificates in the browser’s key database.

The current IF implementation does not support verifying the user supplied certificate (or its certificate chain) against the list of revoked certificates on the applicable CRLs.

It is not sufficient to just to have your certificate be validated, in order to be identified by the IF. It is necessary for the Distinguished Name from the user-supplied certificate to map to a valid Policy Director user.

6.8.2 Application PKI

The current implementation of the IF uses PKI predominantly for supporting confidentiality services. The majority of the communications between the IF servers are encrypted using server certificates. The majority of services use the IBM Global Security Kit as their key repository as well as for the PKI libraries. The only exception is for

those services that do not support encryption and the Microsoft IIS Web Server that uses its own libraries.

The IBM SecureWay Director can support authentication via the “client” certificate and the IBM/Tivoli SecureWay Policy Director WebSEAL product can authenticate via the Web Server’s certificate. That is, the Distinguished Name in the Web Server’s certificate must match the Distinguished Name identified for that junction.

In the tables below, the shaded areas indicate where the IF is using PKI.

Table 39: Authentication Matrix

Path	Authentication
Browser – WebSEAL	<i>UserID /password</i> ◆ Individual end user’s unique “name” ◆ Individual end user’s unique <i>password</i> User registry is in an LDAP Directory. WebSEAL performs a search of secUser for a dcePrincipal=<userid> . If success, performs an LDAP bind with resulting Distinguished Name (minus the secAuthority=Default tag) and the user supplied <i>password</i> .
	<i>PKI Certificate Based</i> ◆ Client Certificate User registry is in an LDAP Directory. WebSEAL performs a search of the LDAP directory for an object that matches the Distinguished Name in the certificate. If it finds one, it looks for an attribute that maps to a Distinguished Name of a Policy Director user. If the Distinguished Name in the map matches a Policy Director user, then the user is authenticated.
WebSEAL – Web Server	One-way SSL authentication. WebSEAL rejects the connection if the DN in the supplied certificate from the Web Server does not match the DN configured in the junction.
WebSEAL – Web Server/WAS AE Servlet Engine	<i>UserID /password</i> ◆ Individual WebSEAL server identity ◆ Individual WebSEAL server <i>password</i> WAS AE configuration is setup to use the same user registry in the LDAP directory as Policy Director. WAS AE performs a similar process for authenticating the WebSEAL identity as the WebSEAL did for the end user. WebSEAL identity and <i>password</i> maintained in a configuration file on each WebSEAL installation.
WebSEAL – IBM SecureWay Directory	<i>UserID /password</i> ◆ Individual WebSEAL server identity ◆ Individual WebSEAL server <i>password</i>
Policy Director Servers—Policy Director Servers (includes WebSEAL – Security Management Server and Authorization Servers – Security Management Server)	<i>UserID /password</i> via DCE using kerberos 5 DCE keytab file with a <i>password</i> stash file. See Policy Director COTS documentation and the GCSS-AF installation procedures for more information. Also, refer to IBM and Transarc DCE documentation.

⁴⁵ Unless otherwise noted, all certificates support Netscape Certificate Authority generated server certificates.

Path	Authentication
WAS AE Servlet Engine – IBM SecureWay Directory	<i>UserID /password</i> <ul style="list-style-type: none"> ◆ Common WAS AE server identity ◆ Common WAS AE server <i>password</i> Configured in the Global Security Settings
WAS AE Servlet Engine – Authorization Servers	<i>PKI Certificate</i> x.509 certificate generated using the SSLSVRCFG tool during installation/configuration. Configured in the <i>aznapi.conf</i> file.
WAS AE Servlet Engine – UDB (a.k.a. IBMs DB2)	<i>UserID /password</i> <ul style="list-style-type: none"> ◆ Sas.server.props and sas.client.props WAS AE configuration files
WAS AE Servlet Engine – WAS EE Component Broker	<i>UserID /password</i> <ul style="list-style-type: none"> ◆ Application specific <i>userid/password</i> configured in the applications property files. Procedurally this could all use the same <i>userid</i> and <i>password</i>
WAS EE Component Broker – Authorization Servers	<i>PKI Certificate</i> <ul style="list-style-type: none"> • x.509 certificate generated using the SSLSVRCFG tool during installation/configuration. Configured in the <i>aznapi.conf</i> file.
WAS EE Component Broker – DCE Cell Directory Service	<i>UserID /password</i> via DCE using kerberos 5 DCE keytab file with a <i>password</i> stash file. See Policy Director COTS documentation and the GCSS-AF installation procedures for more information. Also, refer to IBM and Transarc DCE documentation.
WAS EE Component Broker – Oracle	<i>UserID /password</i> <ul style="list-style-type: none"> ◆ <i>Userid/password</i> and <i>connect string</i> configured in CB SMUI <i>Management Zones</i> → <i><Application> Zone</i> → <i>Configurations</i> <i><Application> Config</i> → <i>RDBConnections</i> → <i><Application> Container</i>
WAS EE Component Broker – UDB (a.k.a. IBMs DB2)	<i>UserID /password</i> <ul style="list-style-type: none"> ◆ WAS EE Component Broker configuration file
Policy Director Servers – DCE Cell Directory Service (CDS)	<i>UserID /password</i> via DCE using kerberos 5 DCE keytab file with a <i>password</i> stash file. See Policy Director COTS documentation and the GCSS-AF installation procedures for more information. Also, refer to IBM and Transarc DCE documentation.
WAS EE Component Broker – MQSeries MQM	<i>UserID /password</i> <i>Userid/password</i> and <i>connect string</i> configured in SMUI <i>Management Zones</i> → <i><Application> Zone</i> → <i>Configurations</i> <i><Application> Config</i> → <i>RDBConnections</i> → <i><Application> Container</i> It uses the operating system user that the application is running as if no <i>userid/password</i> is supplied.
MQSeries MQM – MQSeries MQM	<i>UserID</i> and <i>password</i> hardcoded in a DISA supplied Security Exit

Table 40: Confidentiality Network Traffic Matrix

Path	Encryption
Browser – WebSEAL	HTTPS WebSEAL server certificate (support for Netscape CA) Supports use of client certificates if available.

Path	Encryption
WebSEAL – Web Server	HTTPS WebSEAL server certificate Web Server server certificate
Web Server -- WAS AE Servlet Engine	No native encryption The OSE Redirector mechanism does not support encryption
WebSEAL – IBM SecureWay Directory	LDAPS WebSEAL server certificate LDAP server certificate
Policy Director Servers -- Policy Director Servers (includes WebSEAL – Security Management Server and Authorization Servers – Security Management Server)	DCE over SSL or DCE over GSS Policy Director Server Certificates
Policy Director Servers -- Policy Director Servers (includes WebSEAL – Security Management Server and Authorization Servers – Security Management Server)	SSL over TCP Policy Director Server Certificates Some communication between servers is socket based and not DCE based.
WAS AE Servlet Engine – IBM SecureWay Directory	LDAP (unencrypted) SPR written – deficiency in product for support of LDAPS
WAS AE Servlet Engine – Authorization Servers	DCE over SSL or DCE over GSS for Policy Database Updates SSL over TCP for access control checks. Both use Policy Director Server Certificates
WAS AE Servlet Engine – UDB (a.k.a. IBMs DB2)	No native encryption WAS AE configuration/metadata/ACLs are required to be in UDB.
WAS AE Servlet Engine – WAS EE Component Broker	HTTPS for Authentication Process WAS AE server certificate WAS EE server certificate GIOP for Other Traffic
WAS EE Component Broker – Authorization Servers	DCE over SSL or DCE over GSS for Policy Database Updates SSL over TCP for access control checks. Both use Policy Director Server Certificates
WAS EE Component Broker – DCE Cell Directory Service	DCE over SSL or DCE over GSS Policy Director Server Certificates
WAS EE Component Broker – DCE Oracle	SQL*Net via Oracle ANO Note: Oracle ANO doesn't use certificates for encryption. It uses an alternate encryption mechanism
WAS EE Component Broker – UDB (a.k.a. IBMs DB2)	No native encryption; see text below this table. Component Broker configuration/metadata is required to be in UDB. The GCSS-AF IF recommends that application data be stored in an Oracle server.
Policy Director Servers – DCE Cell Directory Service (CDS)	DCE over SSL or DCE over GSS Policy Director Server Certificates
WAS EE Component Broker – MQSeries MQM	N/A. Traffic isolated to same host.
MQSeries MQM – MQSeries MQM	DISA current practice is to establish VPNs between MQSeries MQMs that communicate across a WAN. The traffic is left in the clear within a LAN.

