



Global Combat Support System - Air Force

Application Framework Developer's Guide

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

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

Document Number: **PROJ-2000-GCSSAF-0371**

Version: **2.2**
Date: **01/09/01**



Lockheed Martin Federal Systems

*1801 State Route 17C
Owego, NY 13827-3998*

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 Draft	IPT	07/20/00			INITIAL SEPARATION FROM FIRST DEVELOPER'S GUIDE
1.1 Draft		7/28/00			GENERAL CLEAN-UP: SPELLING OUT ACRONYMS, CLARIFYING TEXT, ETC.
2.0	IPT	10/31/00			UPDATE RESULTS BASED ON GENERAL REVIEW
2.1		11/07/00			UPDATE FOR NON-DEVELOPMENTAL ITEM INTEGRATION AND CLARIFYING TEXT
2.2	PORTAL TEAM	1/09/01			ATTACH AIR FORCE PORTAL INFORMATION
2.3	PORTAL UPDATE	2/9/01			REVISE AIR FORCE PORTAL GUIDANCE INFORMATION

TABLE OF CONTENTS

- 1. INTRODUCTION 6**
 - 1.1 PURPOSE..... 6
 - 1.1.1 Overview 6
 - 1.1.2 Purpose of this Document 8
 - 1.2 OBJECTIVES 8
 - 1.3 SCOPE..... 8

- 2. REFERENCE DOCUMENTS 10**
 - 2.1 GOVERNMENT DOCUMENTS 10
 - 2.2 APPLICABLE STANDARDS 10
 - 2.3 CONTRACTOR DOCUMENTS..... 10
 - 2.4 PRODUCT DOCUMENTS 11
 - 2.5 GENERAL INFORMATION TECHNOLOGY 11

- 3. GCSS-AF ARCHITECTURE 12**
 - 3.1 REFERENCE ARCHITECTURE OVERVIEW..... 12
 - 3.1.1 Integration Framework (IF)..... 13
 - 3.1.2 Application Framework 14

- 4. GCSS-AF SPIRAL LIFECYCLE PHASES 15**
 - 4.1 BUSINESS MODEL 17
 - 4.2 ANALYSIS MODEL 18
 - 4.3 DESIGN MODEL 18
 - 4.3.1 Introduction 19
 - 4.3.2 GCSS-AF Design Requirements..... 19
 - 4.3.2.1 Design Modeling of Business Components 19
 - 4.3.2.2 GCSS-AF Architecture Requirements..... 20
 - 4.4 IMPLEMENTATION MODEL 20
 - 4.4.1 Introduction 21
 - 4.4.2 What is Provided by the Integration Framework 21

- 4.5 DEPLOYMENT MODEL 22
 - 4.5.1 Introduction 22
 - 4.5.2 Deployment patterns 22
- 5. APPLICATION FRAMEWORK STANDARDS 23**
- 5.1 THE OPEN APPLICATIONS GROUP (OAG) 23
 - 5.1.1 The Open Applications Group Integration Specification (OAGIS) 23
- 6. COMPONENTIZATION 26**
- 6.1 COMPONENT DEFINITION 26
- 6.2 COMPONENT PROPERTIES/ATTRIBUTES 28
- 6.3 COMPONENT BASED DEVELOPMENT PROCESS 30
- 7. USE OF THE INTEGRATION FRAMEWORK 32**
- 7.1 APPLICATION FRAMEWORK DEVELOPMENTAL COMPONENTS 32
- 7.2 APPLICATION FRAMEWORK NON-DEVELOPMENTAL COMPONENTS (COTS, GOTS, AND LEGACY SYSTEMS) 32
- 8. APPLICATION VALIDATION AND INTEGRATION 35**
- 8.1 GCSS-AF COMPLIANCE 35
- 9. RECOMMENDED APPLICATION DEVELOPER TOOLS 36**
- 9.1 INTEGRATED DEVELOPMENT ENVIRONMENTS 36
 - 9.1.1 Visual Age for Java 36
 - 9.1.2 Visual Cafe 36
- 9.2 MODELING TOOLS 36
 - 9.2.1 Unified Modeling Language 36
 - 9.2.1.1 Rational Rose 37
 - 9.2.1.2 GCSS-AF Systems Solution UML Model 37
 - 9.2.2 Data Modeling 37
 - 9.2.2.1 IDEF1X 37
- 9.3 CLASS LIBRARIES / JAR FILES 37
- 9.4 CODE TEMPLATES BASE CLASSES AND HELPER CLASSES 38
- 10. FUTURE DIRECTION 39**
- 10.1 CONCEPT 39
- 10.2 IMPLEMENTATION 39

LIST OF FIGURES

FIGURE 1 GCSS-AF DOCUMENT AND MODEL INTER-RELATIONSHIPS..... 7

FIGURE 2 - GCSS-AF REFERENCE ARCHITECTURE 12

FIGURE 3 - APPLICATION DEVELOPER AND INTEGRATOR ROLES 16

FIGURE 4 OAG COMPONENTIZED BUSINESS PROCESS 24

FIGURE 5 OAGIS INTEGRATION SCENARIO..... 25

FIGURE 6 - COMPONENT DEVELOPMENT SIMPLIFIED FLOW 31

FIGURE 7 - LEGACY INTERFACE AND WRAPPING..... 34

LIST OF TABLES

TABLE 1 - WRAPPING ENHANCEMENTS 33

This Page Intentionally Left Blank

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 2.

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.

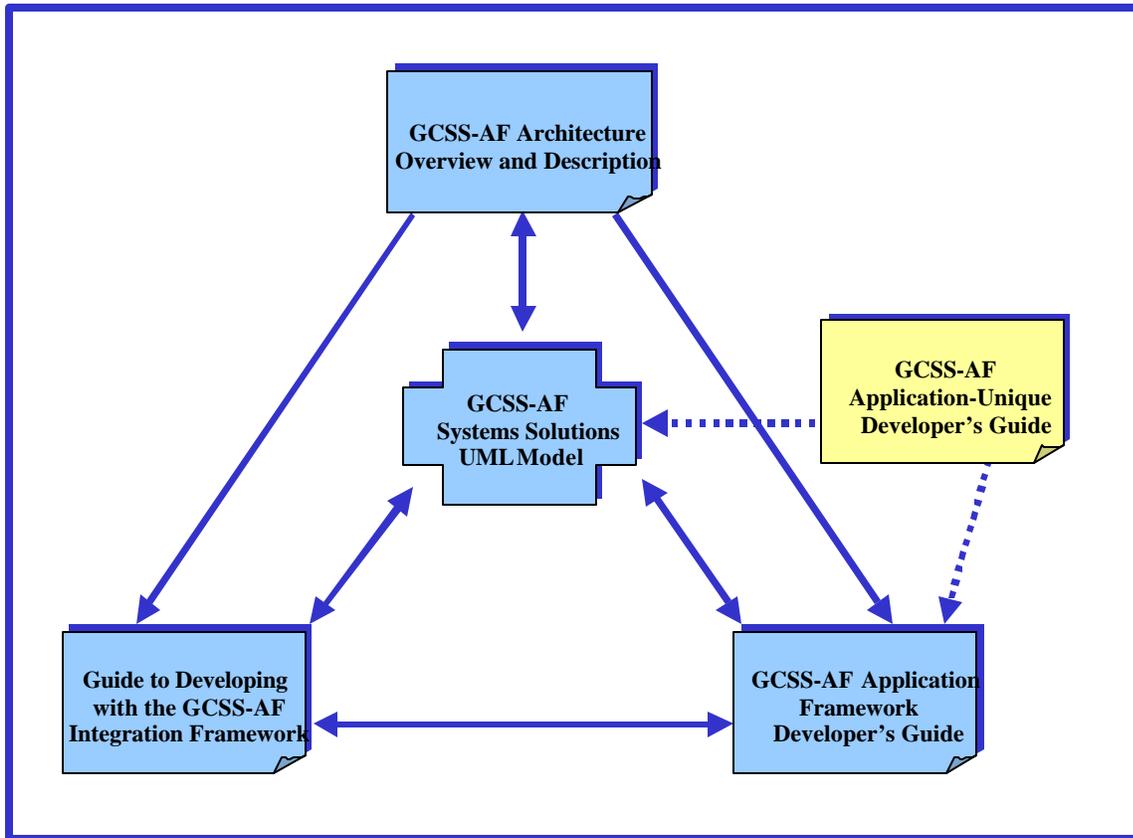


Figure 1 GCSS-AF Document and Model Inter-relationships

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 intent of the GCSS-AF Application Framework Developer's Guide is to provide a prospective application developer with the information that is necessary to implement an application that is consistent with the concepts upon which GCSS-AF is based and is GCSS-AF compliant. As depicted in Figure 1 GCSS-AF Document and Model Inter-relationships, this document is intended to be supplemented by Application Unique Developer's Guides that provide program-specific guidance.

The User Interface facility, or presentation layer, of GCSS-AF as of 1/3/2001 has been designated to be the Air Force Portal (AFP). The current AFP is based on a Corona demonstration portal, and is designated AFP 1.1. To learn how to add content to this portal, visit the web site <http://factory.plumtree.com/>. In the second half of 2001 Air Force Portal Version 3.0 will be deployed based on a competitively selected commercial product. At that time there will be embedded updates to the GCSS-AF Developer's Guides completed to incorporate the User Interface Facility, Air Force Portal into the GCSS-AF Developer's Guides.

This document is not intended to provide step by step guidance on how applications are to be designed and developed. An Application Developer shall be responsible for utilizing their development processes and tools for the work that they do. However, there are several factors related to GCSS-AF that will impact the developer's work. In addition, the Guide to Developing with the GCSS-AF Integration Framework provides guidance to an application developer concerning the use of the Integration Framework.

There are a number of items that are accessible on the GCSS-AF Program Web page at Gunter Air Force Base (AFB) that provide insight into the overall goal of the GCSS-AF Architecture. These items include released documents and scheduled meeting information.

1.2 Objectives

The GCSS-AF Application Framework Developer's Guide provides the developers of the business components, which are elements of an application, with the pertinent information, or pointers to the pertinent information, that is necessary to develop and integrate business components that are compliant with the GCSS-AF Reference Architecture. This information includes a description of how newly developed components are validated and integrated into the architecture, as well as how GCSS-AF impacts application development.

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 the 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 Spiral Lifecycle Phases as depicted in the UML Model and maps these phases to the various Developer and Integrator Roles.
- Section 5: This section describes standards applicable to application development. It describes the OAG and its standards from an integration perspective.
- Section 6: This section describes componentization and introduces the concept of component based development.
- Section 7: This section provides a brief description of the Integration Framework and its applicability to GCSS-AF application development. It also describes approaches for integrating non-developmental items with the Integration Framework.
- Section 8: This section provides a brief description of Application Validation and Integration into GCSS-AF. This section also provides a brief description of compliance.
- Section 9: This section describes the recommended development tools for application development within GCSS-AF.
- Section 10: This section depicts the future direction of GCSS-AF and the lifecycle for model and software artifacts.

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

- 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

2.2 Applicable Standards

- Open Applications Group Common Middleware API Specification (OAMAS), Release 1.0
- Open Applications Group Integration Specification (OAGIS), Release 6.2

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) 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.

2.4 Product Documents

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

2.5 General Information Technology

- Component Software : Beyond Object-Oriented Programming by Clemens Szyperski (ISBN: 0201178885)
- The Unified Software Development Process by Ivar Jacobson, Grady Booch, and James Rumbaugh (ISBN: 0201571692)
-

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.

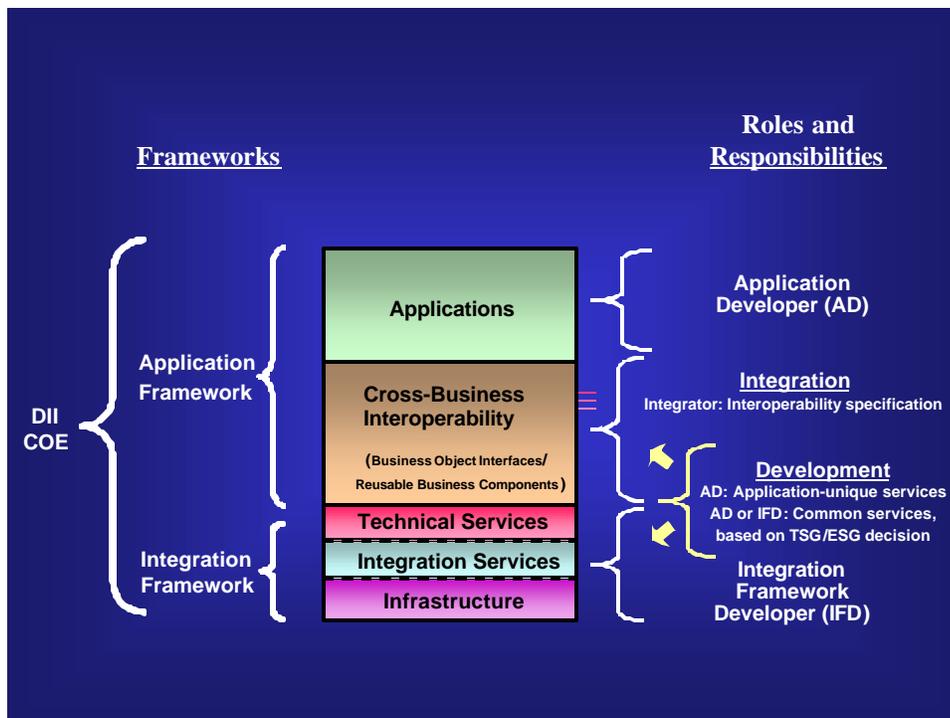


Figure 2 - GCSS-AF Reference Architecture

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.

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 the Guide to Developing with the GCSS-AF Integration Framework. A summary of these layers of the architecture is provided below.

The lowest layer, **Infrastructure**, provides the Operating System (OS) and major system level Commercial Off The Shelf (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 Common Object Request Broker Architecture (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 must also be DII COE compliant. This layer also encompasses the development tools required to construct and assemble components.

4. GCSS-AF Spiral Lifecycle Phases

Models are complete abstractions of systems or contexts. Models are blueprints of systems used for system construction and renovation. They are used to understand and manage complexity within systems and are used for communication and assurance of architectural soundness. A UML model is a representation of the problem domain and system software. Each model contains views, diagrams, and specifications to visualize and manipulate the elements in them.

To represent GCSS-AF in model form, a lifecycle has been identified for each spiral development within GCSS-AF. The five phases of that lifecycle, which are named after the UML models produced in each phase, and *their associated developers* are:

- **Business Model** (*GCSS-AF Integrator*) - the model of the Application's Business Activities
- **Analysis Model** (*GCSS-AF Integrator*) - the model of the Application's Enterprise Architecture at the Business Object level.
- **Design Model** (*Application Developer*) - the model of the Application's Internal Architecture and components.
- **Implementation Model** (*Application Developer*) - the model of the actual executable components that compose the application.
- **Deployment Model** (*GCSS-AF Integrator and Application Developer*) - the model of the application components laid down on the hardware that processes the components.

The GCSS-AF Integrator is responsible for the **Business Model**, the **Analysis Model** and a portion of the **Deployment Model**. The Application Developer is wholly responsible for the **Design and Implementation Models** and associated phase activities for his Application spiral. The Application Developer also shares responsibilities with the Integrator for the **Deployment Model** and phase activity. Figure 3 - Application Developer and Integrator Roles depicts this division of responsibilities.

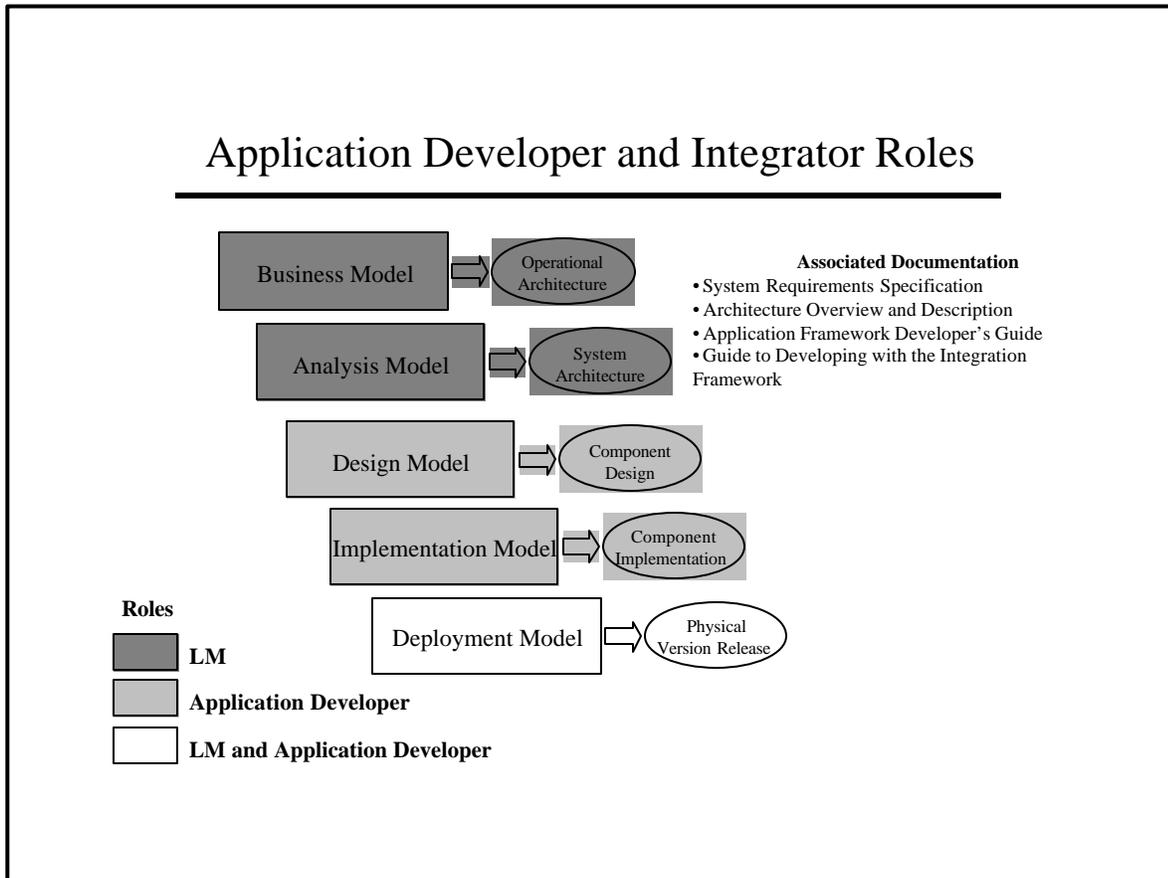


Figure 3 - Application Developer and Integrator Roles

Currently the tool used to develop and maintain the GCSS-AF UML imposes some structural restrictions on how the UML is built. There are four views that are mandated. These four views are the Use Case View, the Logical View, the Component View and the Deployment View. The five lifecycle models are composed within these views.

- 1) The **Business Model** is composed in the Use Case View and the Logical View. The Use Case View contains Business Activities captured as Mission Scenarios (overriding missions) and Use Cases (steps) that make up a Business Activity. The Logical View contains a UML Package called Business Model where Activity Diagrams that graphically represent the Business Activities from the Use Case View are captured.
- 2) The **Analysis Model** is composed in the Logical View. A UML Package called Analysis Model in the Logical View captures the Collaboration Diagrams that realize the Business Activities previously captured in the Business Model.
- 3) The **Design Model** is composed in the Logical View and the Component View. A UML Package called Design Model in the Logical View captures the Sequence Flow

Diagrams that realize the Collaboration Diagrams previously captured in the Analysis Model. In the Component View, the Design Objects used in the Logical View Design Model package are grouped into Components in anticipation of Deployment.

- 4) The **Implementation Model** is composed in the Logical View and the Component View. A UML Package called Implementation Model in the Logical View reverse engineers the code that realizes the Sequence Flow Diagrams of the Design Model. During the reverse engineering the Component View is populated with the actual Component Packaging of the developmental and non-developmental items.
- 5) Finally **Deployment Model** is composed in a single Deployment View for the whole of GCSS-AF (restriction of the tool). Here Components are allocated to the processors they run on.

Within each lifecycle phase GCSS-AF software is structured following the Layer System View and is divided into two framework layers: Application Framework and Integration Framework. The Application Framework Layer consists of the Application Business Components following primarily the OAG component Structure. These Application Business Components can be found in the Application Package. The Integration Framework Layer forms the underlying foundation and is primarily found in the Integration Framework Package. The Application Framework invokes the Integration Layer through defined Application Program Interfaces (APIs) that follow industry standards, where possible. Within the Integration Layer, the full set of Open System Interconnection (OSI) layers (below the Application Layer) is represented; but those layers are encapsulated within the Integration Layer.

Throughout the Rational Rose model, color coding of artifacts is used to identify the Lifecycle phase in which the artifacts are resident. The output diagram contains this color coding. For example, artifacts take the color of the atomic artifact furthest along in the Lifecycle. Atomic artifacts change color when they enter a phase.

The following sections will describe each of the five phases and detail the activities of an Application Developer as they employ the capabilities of the GCSS-AF IF during the Design, Implementation and Deployment phases.

4.1 Business Model

The **Business Model** identifies key data, roles, and behavioral requirements to be considered in the Analysis phase.

During the Business Model phase, the Integrator will either create all of the required high level Business Model artifacts or update the existing Business Model artifacts, depending upon whether the spiral under development is an early spiral or a later one. This is accomplished through the Integrator working with the customer in composing use cases, creating User Requirements, and creating Activity Diagrams. In addition, the Customer defined Business Metrics are mapped by the Integrator to the GCSS-AF

Business Metrics. The Business Metrics are used to prioritize spirals and commit to improvements through spiral implementations.

4.2 Analysis Model

In the Analysis phase, the necessary activities and resources are planned, the features are specified, and the architecture is designed in order to carry out the behavior captured in the **Business Model**.

The GCSS-AF Integrator will identify that portion of the Analysis Model that pertains to the current or next spiral. This Analysis Model shall include the identification of GCSS-AF artifacts required for the spiral under consideration. As the application unique business components are analyzed, the interfaces needed by the business components shall be specified in either Interface Definition Language (IDL) or via the OAGIS Application Messaging Interface (AMI) (e.g., Extensible Markup Language (XML), Document Type Definition (DTD)). In addition, all semantics and performance characteristics of the interface shall be captured as well. This interface information is captured in the GCSS-AF Interface Repository. User Requirements from the Business Model are normalized. The normalized requirements are allocated to the Business Components (see Application Framework Standards Section 5 for discussion of Business Components) and Integration Framework Components. The allocated requirements are collected into the Program System Requirements Specification (SRS). Use Cases and associated System Requirements are allocated to the next increment to be designed. All these work products are provided to the Application Developer as the application unique Architecture Description.

4.3 Design Model

In the Design Model phase, how the system will be realized in the implementation phase is described.

The Application Developer shall design each spiral to meet the Program System Requirements Specification (SRS) requirements. The Architecture Description shall be the basis from which the Application Developer shall design the application.

Deliveries from the Design Model phase are dependent on the processes the Application Developer follows and the needs of the Government. At a minimum, the deliveries should include a UML model with Sequence Flow Diagrams realizing the allocated Use Cases and System Requirements.

At the end of the Design Model phase, the Application Developer shall deliver the UML design model.

4.3.1 Introduction

The *Unified Software Development Process* by Ivar Jacobson, Grady Booch, and James Rumbaugh defines the design model as:

“an object model that describes the physical realization of use cases by focusing on how functional and nonfunctional requirements, together with other constraints related to the implementation environment, impact the system under consideration. In addition, the design model serves as an abstraction of the system's implementation and is thereby used as an essential input to activities in implementation.”

The design model, when complete, is used as a blueprint for the implementation of the system. That is, the design model takes the generic analysis model and describes the specific implementation.

The design model contains the definition and description of design classes that will ultimately be included in the implementation of the system. The design classes specify the attributes, operations, and parameters that will be included as a part of the implementation.

In the design model the flow-of-events is depicted with the use of sequence diagrams.

The interaction of the application being developed with the IF is shown in both the class and sequence diagrams in this model.

4.3.2 GCSS-AF Design Requirements

The following sections discuss the various design elements that the application developer must take into consideration when developing the design model.

4.3.2.1 Design Modeling of Business Components

The design model provides the details of the application specific business logic. This effort will produce a description, at a minimum, of the following:

- **Exposed Application APIs:** The application must define what APIs are provided for use by other components. The service or function provided is defined during the use case and scenario development and characterized in the class modeling. The resulting interfaces are documented in Interface Definition Language (IDL).

- **Definition of the Business Service Requests (BSRs):** The application must define what BSRs are produced and consumed by the application. The decomposition of the Business Object Document (BOD) associated with a BSR is presented in the class diagrams. The use of the BSR is depicted in the sequence diagrams.
- **Use of IF Provided APIs:** The APIs for the services and capabilities provided by the IF that are used by the application are specified. This would typically be accomplished through the use of sequence diagrams in the modeling tool.

The data to be managed by the application must also be modeled. If the application is not very data intensive this may be done in the UML modeling tool. If the application is data intensive this should be performed with the aid of a data-modeling tool, such as ERWin. A data-modeling tool might also be used if the application is generating objects from an existing database. Currently the only instance in the IF where XML is used is in the representation of the BODs. In the future this may be extended to represent application-managed data as well.

4.3.2.2 GCSS-AF Architecture Requirements

There are a set of guidelines defined for use by the application developer that discuss when and how various types of components should be used (i.e. servlets, Enterprise JavaBeans, and CORBA components.) The details of these guidelines can be found in the Guide to Developing with the GCSS-AF Integration Framework.

In addition to following these guidelines, there are several other issues that must be taken into consideration. These issues include expandability, portability, and reuse.

Depending on the nature of the application being developed, it may be necessary to expand the system in the future to accommodate a higher number of users. If this is a possibility, the application developer must address how this might be accomplished.

4.4 Implementation Model

In Construction and Assembly, new components are constructed and existing components are assembled into the delivered application. The delivered application is then reverse engineered populating the Construction and Assembly package and the Component View with the actual components of the application. The Construction and Assembly package and actual Component View are then reconciled with the Design package and the previously specified Component View from the Design Model.

In the test phase, the entire system is verified.

Deliveries from the Implementation Model phase are dependent on the processes the Application Developer follows and the needs of the Government. At a minimum, the

deliveries should include a UML model with Components realizing the Design Model allocated to processors in the preliminary Deployment Model.

The Application Developer shall implement each spiral to meet the Program SRS requirements. The Application Developer shall work with the Integrator to ensure the necessary Integration Framework components are made available for reuse. The Integrator shall provide these components and ensure that these components are compliant with the DII COE. At the end of the Implementation phase, the Application Developer will deliver:

1. UML Implementation Model
2. UML Deployment Model (preliminary)
3. Validated and Verified GCSS-AF Compliant application

4.4.1 Introduction

The primary purpose of the implementation model is to describe how the elements in the design model are to be implemented as components. The implementation model also describes how the components will be organized based on the implementation environment, the programming language to be used, how the components depend on each other, and what services provided by the IF will be utilized by this application. The model is represented as a hierarchy of implementation subsystems containing components, interfaces, and lower-level subsystems.

The implementation model contains the definition of additional classes and components not previously defined in the design model. An example of this type of class or component would be one that may be required in order to tie the application being developed to some service provided by the IF. This might be something as simple as a wrapper used to do the translation necessary when integrating components written in two different languages.

The implementation model may also contain the definition and description of stubs that can be used to develop or test components in the application. The stubs may represent an interface to another system, or any other type of external contributor to the application.

4.4.2 What is Provided by the Integration Framework

The artifacts that are provided by the Integration Framework (IF) that aid in the development of the implementation model are described in the [Guide to Developing with the GCSS-AF Integration Framework](#). That document describes the IF content by giving a brief description of the services and facilities available in the IF, as well as, provides design guidance for GCSS-AF Applications using the IF. For more detailed information regarding the IF, the Application developer must access the [GCSS-AF Systems Solutions UML Model](#).

4.5 Deployment Model

Mission Applications have various requirements associated with their deployment. The GCSS-AF IF has been designed to be scalable to support these various requirements. This scalability allows for patterns of Mission Application deployment.

4.5.1 Introduction

The Application Developer shall work with the Integrator, the Customer and the System Program Office (SPO) in the actual implementation of the capability for the Customer (i.e., the Deployment). Compliance Test shall be conducted and Compliance Status shall be verified. In addition, all Stewarded and Received interfaces shall be incorporated into the Interface Repository, such that they are available to personnel associated with GCSS-AF. Finally, the Models shall be validated. If there are problems identified, the problem source shall be identified by the Integrator. Based on the problem source, the problem will be assigned the appropriate owner to work the problem to closure.

At the end of the Deployment Model phase, the final UML Deployment Model will be complete.

4.5.2 Deployment patterns

Deployment patterns are common approaches to categorize deployments. They cover such things as deploying a ported multi-echelon Mission Application or a new theater only Mission Application. The purpose of the IF Deployment Patterns is to provide the Application Developer with a concept of the various possible ways in which to deploy their Mission Application in conjunction with the IF capabilities. During the Analysis and Design Phases of the Application Development, the Integrator and Application Developer need to decide how the Mission Application is to be deployed. To assist in this decision process, the Integrator and Developer will reference the pre-existing Deployment Patterns contained in the UML Model.

These Deployment Patterns will depict the distribution of functionality across the enterprise (e.g. Base, MajCom, Headquarters, Air Force Wide).

Because the development of the IF is ongoing, these Deployment Patterns will not be available until after the release of the GCSS-AF UML Model for the IF Version 2.0. Additionally, many Operations and Support issues need to be considered in conjunction with the Deployment. Currently deployment and Operations and Support of the Integration Framework still need to be addressed.

5. Application Framework Standards

In terms of standards that apply to GCSS-AF, the Business Component Framework is based upon the Open Applications Group (OAG) standards. In addition, the technical components of the Integration Framework utilize CORBA, OAMAS and other standards as documented in the Guide to Developing with the GCSS-AF Integration Framework. The Referenced Documents section of this document lists the government documents, applicable standards and contractor documents that are pertinent to Business Object development.

5.1 The Open Applications Group (OAG)

The tenants of the Application Framework are based upon the Open Applications Group (OAG) integration concepts. The OAG is a non-profit, vendor centric, consortium comprised of enterprise application software developers. The purpose of the OAG is to create Open Applications Integration by establishing and publishing specifications to enable business object integration across the enterprise. Their focus is on the concept of a Business Object Document (BOD) and its associated business processes used to exchange data between business components. Currently, there are two specifications within the OAG:

- Business – Open Applications Group Integration Specification (OAGIS)
- Technical – Open Applications Group Common Middleware API Specification (OAMAS)

The OAGIS is focused on the analysis of business processes via the concept of integration scenarios. This specification provides the concepts associated with categorizing business components, the associated integration scenarios between business components, and the grouping and format of the business data, which is transferred between the components (the Business Object Document).

To address the need for a common mechanism to transmit the data between business components, independent of the format of the data, the OAG specified the Open Applications Group Common Middleware API Specification (OAMAS). This specification is an attempt to have Middleware manufacturers develop their APIs in a consistent format.

Additional information about the OAG, as well as the OAGIS and OAMAS may be obtained from the OAG Worldwide Web site at www.openapplications.org.

5.1.1 The Open Applications Group Integration Specification (OAGIS)

Componentization enables integration of "best of breed" practices from commercial products by enabling standardized interfaces to new products and a wrapping of existing products. It essentially specifies a framework for "plug-and-play" Business Logic. This concept is exemplified in the OAG's OAGIS where a consortium of

enterprise application software developers has formed to develop a standard way of interfacing business applications. Componentization is an integration-based philosophy and OAGIS is a key mechanism for standardization and interoperability.

Componentization (as defined by OAG) - the process of breaking down business applications into functional components that have tangible points of integration to other components. (see Figure 4 - OAG Componentized Business Process)

Note: The OAG Component is equivalent to a GCSS-AF Business Object

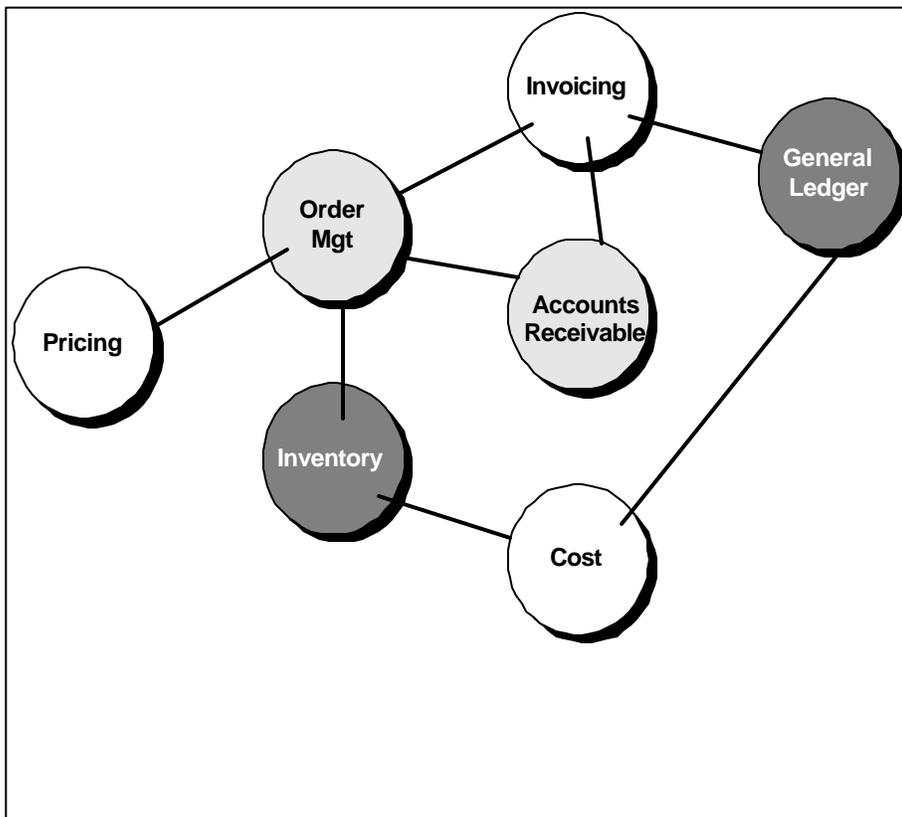


Figure 4 OAG Componentized Business Process

The focus of OAGIS is the real need for interoperability at the business process level and the potential benefits for customers of being able to mix and match all OAGIS - compliant solutions to address their specific requirements. When users want to purchase business applications from multiple vendors, they have the difficult task of getting the business applications to work together without the benefit of controlling the integration of those business applications. In addition, customers are struggling with the even larger task of integrating all of their systems into a coherent information technology

infrastructure to support their business. This approach enforces the discipline required to logically componentize enterprise applications into standardized function and data, thus promoting integration.

Through OAGIS, application integration can be optimized for all, instead of having different costly solutions for each user. The benefits are realized from both the customer's and the software vendor's perspective.

The OAGIS facilitates the interoperability of business components by defining the concept of a Business Service Request (BSR) between two or more business components. The message passed as a result of the BSR is the Business Object Document (BOD). The Business Object Document is the model used to communicate a request from the originating business application to the destination business application. Each Business Object Document includes supporting details to enable the destination business application to accomplish the action. The BOD has a specified format for each of the BSRs. (see Figure 5 OAGIS Integration Scenario)

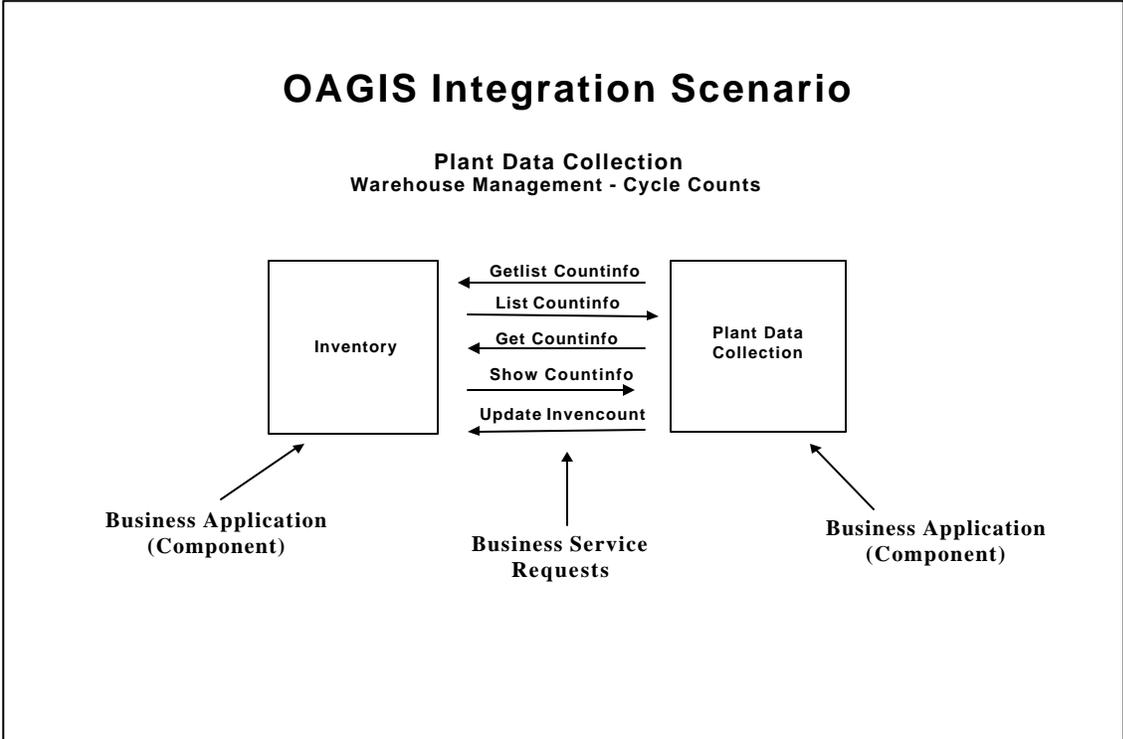


Figure 5 OAGIS Integration Scenario

6. Componentization

It is important that the reader have a clear understanding of what is meant by componentization and what the characteristics of a component are.

When reading the available literature today and asking prominent authors in the field, it becomes apparent that many definitions for components exist today. Therefore, it is important to define what a component is within the GCSS-AF paradigm and to delineate the properties of components. The component definition used is an adaptation of the definition provided in *Component Software Beyond Object-Oriented Programming*. Within this document the terms component and software component are used interchangeably.

6.1 Component Definition

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.

The unit of composition for a component is defined by the content of a component and the results of analysis of the factors that contribute to the definition of the component. The contents of a component may include class libraries (C++, Java, 4GL); encapsulated software modules (ActiveX controls, JavaBeans, Enterprise JavaBeans, CORBA services); framework environments (OAG's OAMAS, IBM's San Francisco); CASE models; and pre-built (COTS, Legacy) applications. The following major factors need to be considered in determining the size or granularity of a component:

- **Abstraction** – The level of abstraction desired for the functionality within a component contributes to the sizing of the component. If the methods used to implement an interface are not of interest to developers, they should be hidden internally within the component. However, if the methods are of interest to developers, they may be candidates as separate components, or they may be exposed as additional interfaces of the component. In the object-oriented environment, a class would define the minimum component size.
- **Accounting** – Since components are the unit of deployment in a system, they become the unit of accounting for the system. In systems where monitoring and tracking of resource utilization is required, considerations for component size are important. If there are many fine-grained components exposed to the Middleware versus bound within coarse-grained components, the overhead of tracking can be prohibitive. If the components are too coarse-grained, accurate tracking can not be accommodated. Also, the resources required by the component may exceed the capacity of the system.
- **Coupling** – The degree of coupling desired between exposed interfaces is another determinant of object granularity. For loosely coupled components within a system,

the context dependencies need to be minimized to the extent feasible. For large complex systems, the system should be partitioned into separately bounded units for analysis. This bounding defines the granularity of components that can be loosely coupled.

- **Compilation** – Components are the upper limit of a compilation unit since a component has the property of independent deployment. Setting the compilation limit to the component provides the best global optimization opportunities. However, a trade off needs to be made at compilation time before picking the component as the level of compilation. It may be better to choose classes or modules within a component as the compilation unit.
- **Delivery** – This refers to the unit of delivery for deployment. The complexity of documenting and the cost of administering components for delivery need to be considered when determining the granularity of components. Training efforts need to be considered in this area as well.
- **Exception Handling** – Ideally, a component handles all of the exceptions within its scope. This requirement affects the size of the component since it may need additional services to handle the exceptions. Also, if there are exceptions not handled within the component, they must be part of the interface definition and the post conditions that are specified for the component.
- **Fault Containment** – Another consideration for component sizing is the need to support fault tolerance and the need to contain the impacts of faults. This typically requires that physical (e.g., redundant systems with voting, alternate network paths) or temporal (e.g., rollback and recovery, guaranteed delivery) fault tolerance must be provided. The boundaries of fault tolerance may provide insight to the granularity required for components.
- **Loading** – The context dependencies of a component determine the loading implications for a component. An effect of component loading with context dependencies is the need for dynamic linking facilities and version checking capabilities. If the context dependencies are not met, the component cannot be loaded. Therefore, if a component is truly independent, the loading issues are simplified and the risk of failure at load time is reduced.
- **Locality** – The distribution requirements of a system composed from components is a major factor to be considered in determining the granularity. Since a component must be loaded as a complete unit, the services provided within the component will reside on a single resource. If there is tight coupling with other components that can not reside on the same resource in a networked environment, significant performance issues may result. Also, it may be desirable to have fine-grained components for services that are provided for many components so that they may be replicated without causing the consumption of large resources.
- **Maintenance** – The stability of the services provided by a component are another consideration for sizing. It is desirable to minimize the impact on a system from

changes due to errors, changing implementations or extending capabilities. The ability to localize these effects to a minimum number of components of small size is important for supporting electronic distribution of updates. Also, the number of components to be maintained impacts the configuration management and administration effort involved for the system.

A component is also defined by contractually specified interfaces. These interfaces, or signature, define the methods or procedures, data variables and invariance, preconditions and post conditions associated with a component's access points. The contract between a component's provider and the user may also include non-functional commitments (e.g., performance, size, and environment). The interface is defined using IDL (Interface Definition Language). The semantics of the IDL are converted for the technology (i.e., CORBA, ActiveX or COM) used for implementation. Preconditions define the conditions that must be met prior to calling the provider's interface. Preconditions also define the configurable attributes that must be defined as part of the instantiation of the component. Post conditions define the implementation that the provider must meet prior to returning control to the caller of the component. Additional features of a component that may be discovered during use can not be utilized without a modification of the contract since changes outside of the contract may occur when addressing non-functional enhancements.

Explicit context dependencies define the resources and environment that the component relies on being available. These context dependencies may include other components, operating systems, platforms, or Middleware technologies (e.g., CORBA, MOM, or COM+).

6.2 Component Properties/Attributes

There are many properties and attributes that can be associated with components. The major properties required of components are:

- **Independent Deployment** – The component must be well separated from its environment and from other components. The component encapsulates its constituent parts. A component can not be partially deployed.
- **Self-contained** – A component needs to encapsulate its implementation and interact with its environment only through well-defined interfaces.
- **Useable by Third Parties** – A component cannot require a third party to have knowledge of its implementation to utilize it in the composition of a system. The interfaces and context dependencies, available to the third party for system assembly or composition, provide sufficient and complete information to support the composition of a system.
- **Persistent State** - Prior to loading, a component cannot have a persistent state. A component may have configurable parameters that are specified as part of the

loading process to give it persistence. The loading or instantiation is accomplished through the CORBA factory, Java constructor, or CORBA servant. The resulting instance of a component within an executing system has a persistent state and is identified as an 'object'. An example would be a Budget Component which when instantiated could have a state designating it the Budget for a Base or the Budget for a MajCom. Prior to instantiation there would be no indication of this.

- **Introspection** - Introspection is the capability to customize and configure a component during assembly and the capability to dynamically discover and invoke component capabilities at runtime. Static information about the component is provided through the IDL definition of the interfaces and context dependencies. Although component technology specifications support the dynamic invocation of components at run time based on key parameters, there are currently no practical implementations of this capability that scale for large complex systems. Therefore, for the near future, the specification of components to execute will be static. The selection of the component will be performed as part of the component assembly process for a system by examining the component interfaces defined through the IDL. However, the location of the actual component may be dynamic and managed through the ORB.
- **Immutable Interfaces** - Once interfaces are contracted and a baseline established, they shall not change. If existing interfaces to a component require change, a new component and new component name must be defined. This addresses the need to protect the fielded components in a large complex system from failure due to interface changes. A decision needs to be made at fielding whether or not the old component is to be maintained. When a component is extended (e.g., adding a new interface) without changing the existing interfaces, only the version (see Versioning) needs to change.
- **Versioning** - Component versions shall be part of their interfaces. As part of the loading process for components, the version number is made available for determining if a particular component is to be loaded. Version numbers for a component change whenever a component's interface is extended or its implementation changes. Since implementation changes may impact performance, it may be necessary to specify a specific version of a component when loading.
- **Extensibility** – Extensions to components at any layer of the architecture are possible as long as they utilize components from the next lower layer of the architecture. For example, during business component construction (see The Open Applications Group (OAG) Section 5.1) in the Application Framework layer, existing components defined in the Integration Framework layer must be utilized to provide services for new business components. If a necessary capability is not available in the integration framework layer, it must be created before the new business component can be constructed. For example, if a business component requires a new presentation service that is not available in the integration framework layer, that capability must be added to the integration framework layer presentation services

prior to being used in the business component. The interfaces of the internal components will not be available to users of the new component unless their interfaces are defined as part of the new component's external interfaces.

- **Externalization (Serialization)** – The business component interfaces only contain serial streams of information – no objects. This requires that objects within the business component must serialize and de-serialize their interfaces that are also external interfaces to the business component.

Note: The OAMAS standard requires that all business interfaces be serialized. This is required since the Middleware technologies may not support non-serialized interfaces and non Object-Oriented (O-O) interfaces only support serialized interfaces.

Additional attributes that further define a component within GCSS-AF are:

- **Object-Orientation.** The previous properties defined for a component identify the need that components, at a minimum, perform encapsulation. However, the remaining properties of O-O, namely polymorphism and inheritance, are not mandatory. Since the target architecture is object-oriented, it is highly desirable that all of the O-O properties be utilized. However, during the transition and as long as legacy systems are included, all components will not be true O-O implementations. Since components are not necessarily executable (e.g., class library), this attribute may not be applicable.
- **Granularity.** Within the component environment, it is common to discuss the granularity, or size, of components. Fine-grained components are small in size and have applicability across a wide range of application types. Coarse-grained components are typically large in size and more limited in their applicability. These coarse-grained components are more closely related to a server-centric implementation of an organizational business model.
- **Reuse Level.** Components may be reusable at the source code, binary code, or specification level to create new components or new versions of existing components.

6.3 Component Based Development Process

The Component-Based Development Process (CBD) describes the creation and deployment of software-intensive systems assembled from components. These components may be newly developed components, reusable components available from previous development efforts, and/or Commercial-off-the-Shelf (COTS) / Government-off-the-Shelf (GOTS) products that provide the required functionality and are compliant with the GCSS-AF System Architecture Framework. CBD partitions software development into three phases: Modeling, Construction and Assembly.

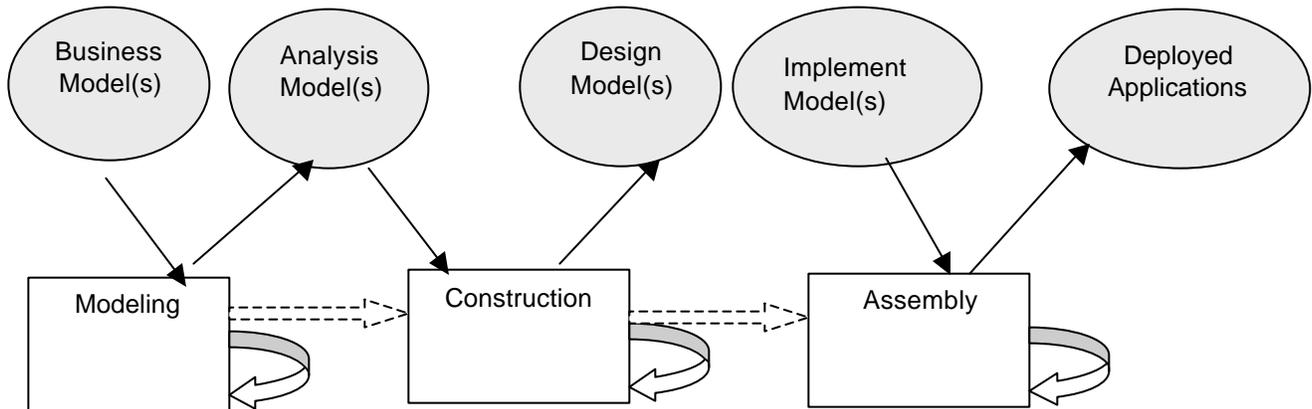


Figure 6 - Component Development Simplified Flow

While GCSS-AF does not specify how these phases are to be performed, it is expected that sound software engineering principles and practices will be followed throughout the process. GCSS-AF does, however, require certain development artifacts to be produced to support integration assessment of the Mission Application and its components. These artifacts are defined in the documentation defining the integrator's role.

Figure 6 - Component Development Simplified Flow provides a very simplified view of the relationships of the phases of component development. It is important to note that component-based development, like any modern development, is intended to be used in an incremental and spiral fashion. Once components are constructed, instances can be assembled to produce multiple and different applications.

7. Use of the Integration Framework

7.1 Application Framework Developmental Components

The Application Developer is expected to develop his Mission Application upon the GCSS-AF Integration Framework. Additional detail regarding the approach to developing with the Integration Framework and the capabilities provided by the Integration Framework can be obtained from the Guide to Developing with the GCSS-AF Integration Framework.

7.2 Application Framework Non-Developmental Components (COTS, GOTS, and Legacy Systems)

When using a non-developmental item to provide some Application Framework functionality, a wrapping of the Legacy System, GOTS, or COTS product should be applied. This wrapping makes the non-developmental item a GCSS_AF non-developmental component. The non-developmental component appears to the rest of GCSS-AF as a GCSS-AF component.

Wrapping provides a mechanism for integrating non-developmental items that are candidates for components into a component based software architecture. Wrapping techniques provide a natural way of integrating non-developmental items with each other and with new software. Wrapping provides access to non-developmental items through an encapsulation layer. The encapsulation exposes only those attributes and operations desired by the software architect. The wrapper serves as an interoperability bridge between a non-developmental item and the software architecture. On one side of the bridge, the wrapper communicates using the non-developmental items' existing communication facilities. On the other side of the bridge, the wrapper presents external components a clean interface that provides abstract services.

The purpose of component wrappers can be understood best in the context of architecture-based systems integration. At the architecture level, a wrapper needs to be more than just a simple encapsulation layer. The wrapper must implement the architecture design in all aspects. The wrapper provides interoperability between the architecture and the non-developmental items. It also should provide value-added functions and information, such as metadata, data conversions, and other architecture features.

Approaches to consider for component wrapping include:

Encapsulation - Encapsulation is the most general form of wrapping. It is the separation of interfaces from implementation. Encapsulation can be used to partition a non-developmental items into one or more components. Each component can be encapsulated separately, and then the system can be reintegrated using object-based communications.

Layering - A layer provides a mapping from one form of application program interface to another. Layering is used in the GCSS-AF Component Architecture to map between the OAGIS standard-based BSR/BOD APIs and legacy or external systems

Data Migration - Many legacy systems and GOTS revolve around large amounts of data. Such systems may be migrated or wrapped. Migration involves moving the data to another data model. Wrapping involves adding layering code to provide access to the legacy database.

Middleware - The term Middleware encompasses a wide range of commercial system integration software. Middleware software service components can extract information from business applications, databases, or legacy systems and transform that information to standard formats. Middleware can provide a mechanism for data to get from place to place; manage application logic and resources; or directly support significant application functionality.

Table 1 - Wrapping Enhancements depicts ways wrapping can change a component's interface.

Table 1 - Wrapping Enhancements

<i>Before Wrapping</i>	<i>After Wrapping</i>
Unique API	Desired API
Unique access mechanism	Uniform access mechanism
Non-exchangeable data format	Exchangeable data format
Limited metadata	Complete/uniform metadata
Inadequate interoperability	Meets Interoperability Needs System-component independence
Limited/proprietary security and Management interfaces	Uniform/comprehensive security and management interfaces

Figure 7 - Legacy Interface and Wrapping, illustrates the architectural approach for communicating between Business Components; wrapping COTS and GOTS applications to form Business Components; and implementing a Legacy Interface type

of Business Component that provides the interface between GCSS-AF Business Components and existing Legacy or External systems.

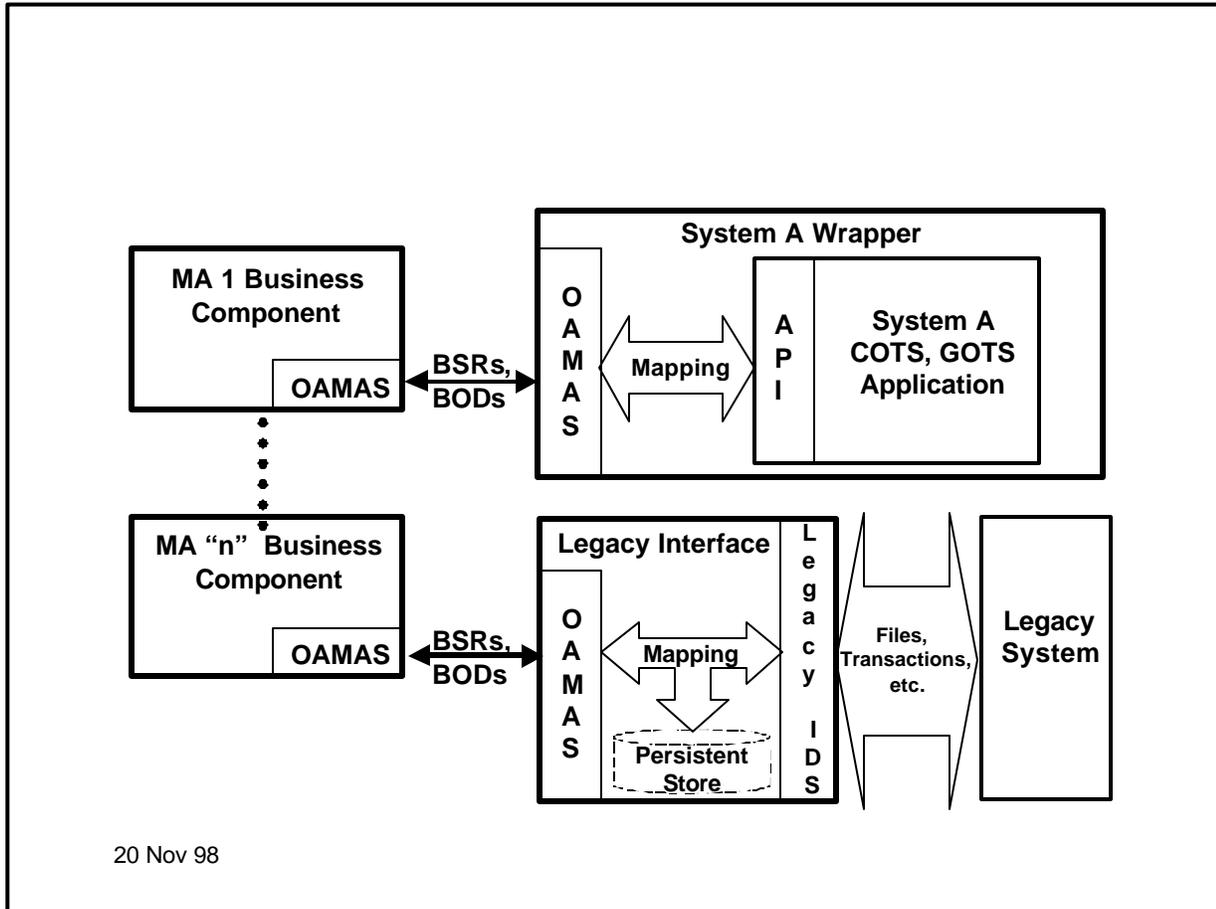


Figure 7 - Legacy Interface and Wrapping

When integrating existing COTS or GOTS applications into the GCSS-AF System Architecture, a wrapper is required unless the application directly supports OAGIS BSR defined interfaces through OAMAS. The resulting Business Component supports the standard communications between components and only exposes the methods that are required from the COTS or GOTS application to support GCSS-AF requirements. Once a Legacy application or COTS/GOTS application is wrapped to create a Business Component, all of the rules associated with interfacing to a component apply.

8. Application Validation and Integration

An Application Architecture Description set of work products shall be created specifically for the application and is the prime delivery of the Analysis Model (see Analysis Model Section 4.2). That set of work products will be the basis from which the Application Developer shall design and implement the component(s) that they are responsible for developing.

In performing development activities as part of GCSS-AF, the application developer is involved in two levels of validation and integration. First, the developer is totally responsible for ensuring that the business components that are developed are compatible with other business components required for their application and with the GCSS-AF environment. The Application Developer shall establish procedures and processes to accomplish this first level of validation and integration.

The second level of integration is accomplished after the developer, through the application SPO, releases components for integration into the GCSS-AF architecture. At this point, the Application Developer shall provide the documentation, test procedures, and test results for the integrator to use during GCSS-AF integration and test. GCSS-AF integration and test will consist of the integrator assessing GCSS-AF compliance of the component(s), as referenced in Section 8.1 - GCSS-AF Compliance, as well as installing and running the components in the GCSS-AF system. This second level of integration and testing shall be performed to ensure the component(s) function in the architecture without interfering with other systems within the enterprise.

8.1 GCSS-AF Compliance

The GCSS-AF Compliance Survey, when complete, will describe the two aspects of GCSS-AF Compliance, Interoperability (Information Stewardship) and use of the GCSS-AF Integration Framework.

9. Recommended Application Developer Tools

9.1 Integrated Development Environments

9.1.1 Visual Age for Java

Visual Age for Java is IBM's Java Integrated Development Environment (IDE). It provides support for building and testing Java applets, servlets, and Enterprise JavaBean components. *This is the recommended IDE. Use of Visual Age for Java automates the build of developed components with IBM WebSphere as well as supports testing at the developers workstation in the WebSphere environment.*

9.1.2 Visual Cafe

Visual Café from Symantec, provides a complete Java Integrated Development Environment (IDE) for the heterogeneous enterprise. *Use of Visual Café limits the use of the complete set of capabilities provided by the IBM WebSphere product.*

9.2 Modeling Tools

Models are complete abstractions of systems or contexts. Models are blueprints of systems used for system construction and renovation. They are used to understand and manage complexity within systems and are used for communication and assurance or architectural soundness. Due to the rapid increase in the complexity of system requirements, successful modeling is an essential task.

9.2.1 Unified Modeling Language

The Unified Modeling Language (UML) is a language that applies to modeling and systems. It is used for specifying, visualizing, constructing and documenting systems and the artifacts of software intensive systems.

A UML model is a representation of the problem domain and system software. Each model contains views, diagrams, and specifications to visualize and manipulate the elements in them.

The Unified Modeling Language (UML) can be used for specifying, visualizing and constructing the elements of software systems. It allows you to visualize, understand and refine your requirements and architecture before committing them to code. The UML ensures that large and complex systems can be modeled successfully.

9.2.1.1 Rational Rose

Rational Rose is an object-oriented analysis and design and construction tool providing visual modeling, component-based development, and support for the Unified Modeling Language (UML). *This is the recommended Object-Oriented Analysis and Design (OOA&D) modeling tool.*

9.2.1.2 GCSS-AF Systems Solution UML Model

One of the primary artifacts developed for use by GCSS-AF developers is the GCSS-AF Systems Solution UML Model. This model represents the GCSS-AF IF and is used to document the services and facilities of the GCSS-AF IF. As mission applications are modeled, those models will be incorporated into the GCSS-AF Systems Solution UML Model.

The GCSS-AF Systems Solution UML Model 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.

9.2.2 Data Modeling

9.2.2.1 IDEF1X

IDEF (Integrated Definition Method) was originally developed under the U.S. Air Force's Integrated Information Support System Project under the Integrated Computer Aided Manufacturing program. IDEF1X is the data modeling component of IDEF. IDEF1X is a language for specifying and defining data structures. It helps a developer consider complex data structures and business rules without being overly concerned about the management database that will be used to implement the system.

ERwin from PLATINUM is an IDEF1X database design tool recommended to help design, generate, and maintain high-quality, high-performance database applications.

9.3 Class Libraries / Jar Files

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 this information, reference the [Guide to Developing with the GCSS-AF Integration Framework](#).

9.4 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 the implementation. These products will include the products specified by the IF as well as any additional product required for the unique solution 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 that makes the task of the application developer simpler and removes the specifics of the product from the interface. For this information, reference the [Guide to Developing with the GCSS-AF Integration Framework](#).

10. Future Direction

10.1 Concept

As GCSS-AF applications are developed, the GCSS-AF Integrator shall incorporate the GCSS-AF compliant artifacts into a Unified Modeling Language (UML) repository. The artifacts contained in this UML repository shall be available to application developers for their use in designing and implementing their applications. Initially, the repository shall contain few artifacts, but eventually there shall be a number of GCSS-AF compliant artifacts that can be used.

In order to facilitate the use of the tool of the developer's choice, the model shall be published to the Web, utilizing a capability that is available in the Rational Rose suite, such that it can be viewed without the need for a tool license. Application Developers shall provide their updates to the model in UML, so that the new Business components can be successfully integrated into the GCSS-AF Architecture Model. The developers are free to use the UML modeling tool of their choice. One license for Rational Rose shall be required by the developer, however, to capture the final version of the model and ensure it can be successfully integrated with the GCSS-AF Architecture Model.

The initial view of the model that is available on the Web page shall contain documentation that explains how to utilize the information that is being provided. This will include how to navigate through the model. Also included shall be information on recent changes that have been incorporated into the model. This shall be especially important in cases where the developers are receiving a subsequent version of the model and need to be aware of the differences from the version with which they were previously working.

The frequency with which the model will be updated will vary depending upon the level of development activity that is underway. On a monthly basis, the model shall be updated, even if only for minor changes. If there has been no change activity to the model in the last month, the developers will be notified that the model will not be updated.

The model itself will provide information for managing the GCSS-AF Architecture Model as well as information that describes and supports the model. This is the approach currently being explored.

There shall also be a test suite identified that will be used to verify the adequacy of the interfaces. The Application Developer shall develop a test suite and conduct testing to verify DII COE compliance.

10.2 Implementation

The GCSS-AF Integrator shall define the use case threads for an application, which will drive definition of the spiral. The version of the GCSS-AF Architecture Model that is published to the Web shall include the information that is pertinent to the Business

Object(s) that the developer is responsible for. The Application Developer shall agree to the Integrator's definition. The provided information shall include the identification of the elements of the existing architecture (i.e. Integration Framework) that the newly developed Business Object(s) shall interface with and/or shall utilize as services. As the components of the architecture are developed, the classes that are identified as part of the process shall have operations associated with them. For each operation, there shall be documentation captured in Interface Definition Language (IDL) UML, or XML DTD notation that addresses the functionality performed by the operation, as well as any input values needed by the operation, and the return value of the operation. This information is the Application Program Interface (API). As the components of the system are identified, the same type of information about how to interface with those components must be specified. This is important information for the developer to know from two perspectives. First, it tells the developer where to find the information that is required in order to communicate with the existing components. Second, it indicates the type of information the developer needs to provide in terms of the level of documentation for the newly developed Business Object.