Department of Health and Human Services

The Centers for Medicare and
Medicaid Services IT
Modernization Program

J2EE Application Development Guidelines

Version 1.0

November 5, 2004

Executive Summary

The Centers for Medicare and Medicaid Services (CMS) has embarked on a significant information technology (IT) modernization program to take better advantage of technology innovation and improve the quality, efficiency, and effectiveness of its services to its users, beneficiaries, and external business partners. As part of these modernization initiatives, the Office of Information Services is developing a set of IT technical guidelines that will assist in ensuring consistency and interoperability among the software applications, components, and services that comprise the CMS Infrastructure. These technical guidelines focus on providing a secure, public-facing infrastructure platform that ensures common infrastructure services for next-generation systems and applications within the CMS Internet Architecture.

CMS has adopted Java 2 Enterprise Edition (J2EE) as the standard application development environment for the CMS Infrastructure. J2EE uses a multi-tiered distributed application model in which application logic is divided into components according to its function. Successful design and integration of applications into the CMS Internet Architecture will require an understanding of the CMS enterprise application and infrastructure standards.

This document describes the CMS three-zone target architecture (Presentation, Application, and Data Zones), including the hardware and software products, and maps the components that make up a J2EE application into this architecture. The focus of this document is on application architecture, such as distributing J2EE application functionality across zones and choosing design options within each zone. The *J2EE Application Development Guidelines* inform in-house and contractor software developers who design, build, and implement J2EE solutions for the CMS Infrastructure and Internet Architecture. The identification of the standards and conventions that CMS has adopted will ensure uniform and consistent implementations of J2EE platform-based solutions.

Toward that end, these guidelines describe common infrastructure and security services, best practices, and preferred design patterns that are applicable to the CMS J2EE environment. This document identifies CMS' selected set of products as the standard components of the target J2EE environment, J2EE development tools, and supporting commercial-off-the-shelf products. Software developers can use these appropriate resources to create well-designed J2EE solutions that are flexible, scalable, and extensible.

These guidelines are based on industry best practices and are intended to be prescriptive for design and development of J2EE-based enterprise applications compliant with the CMS Internet Architecture.

This document is designed to reflect current CMS guidance for J2EE-based application development within the CMS Infrastructure. CMS will incorporate additional items within the *J2EE Application Development Guidelines* throughout the implementation of the CMS three-zone Internet architecture.

Table of Contents

List of Figures

List of Tables

1. Introduction

The Centers for Medicare and Medicaid Services (CMS) has adopted this *J2EE Application Development Guidelines* document (hereinafter the "J2EE Guidelines") as part of the Information Technology (IT) Modernization Program. The *J2EE Guidelines* delineate those standards that CMS has adopted to ensure uniform and consistent implementations of Java 2 Enterprise Edition (J2EE) Platform-based solutions throughout the CMS Enterprise.

1.1 Purpose

The purpose of the *J2EE Guidelines* is to provide clear guidance to software developers on CMS standards while software projects are initiated and developed. The information contained in this document is a summary of key lessons learned, industry best practices, and extensive research that applies to designing and developing J2EE-based enterprise applications that comply with the CMS three-zone Internet Architecture.

1.2 Scope

The guidelines in this document provide developers with resources to create well-designed J2EE solutions that are flexible, scalable, and extensible. The focus of the *J2EE Guidelines* is on application architecture, such as distributing J2EE application functionality across zones and choosing design options within each zone.

The document is a living document that will need periodic updating to take advantage of technology advances and changes in CMS standards.

1.3 Audience

This document is intended to guide CMS developers and contractors who design, build, and implement J2EE solutions for the CMS environment and Internet Architecture.

1.4 Document Organization

The *J2EE Guidelines* are organized as follows:

- Section 2 describes the CMS three-zone Internet architecture and identifies the current set of commercial products and versions supported in the CMS Infrastructure
- Section 3 describes the CMS J2EE Reference Architecture
- Section 4 describes J2EE best practices and patterns that pertain to the CMS environment
- Section 5 describes CMS' choice of development tools.

2. CMS Three-Zone Internet Architecture, Applications Environment, and Infrastructure

CMS believes in architecture that is "good enough" to accomplish CMS' business goals and purpose. The design goals of "good enough" architecture are based on four principles:

- Be flexible
- Concentrate on the most-important pieces of the architecture
- Create an architecture capable of rapid iteration
- Provide accurate documentation so others may follow or learn from previous work.

This section delineates the overall CMS three-zone Internet Architecture that is designed to ensure application efficiency and security, and provides clear guidance on the CMS three-zone Internet Architecture, environment, and infrastructure from the J2EE enterprise application perspective. This section also describes the CMS facility as it would appear to the developer of a new application during its development and deployment. From this description, an application developer can easily determine the physical and logical environment within which a new application must interact and fit. The also section includes a description of the target CMS Infrastructure. Finally, this section provides certain restrictions that may be placed on new applications (e.g., platform and operating systems selection and database usage) to ensure that their addition to CMS will continue the robustness of the CMS Infrastructure. CMS requires that developers and contractors adhere to this specific guidance when designing new CMS IT systems and applications.

2.1 CMS Three-Zone Internet Architecture

In striving to meet its mission and business goals, CMS established a standard Web-enabled architecture for new Web-based applications. This standard is represented in the three-zone Internet architecture shown in Figure 1. The outermost or Presentation Zone consists of web servers that provide presentation services to users who interface to the CMS environment via a web browser. The middle or Application Zone supports business logic for the applications and services. The innermost or Data Zone consists of the database servers used by the applications. The architecture will also support such additional network services as Public Key Infrastructure (PKI) and Domain Naming Services (DNS).

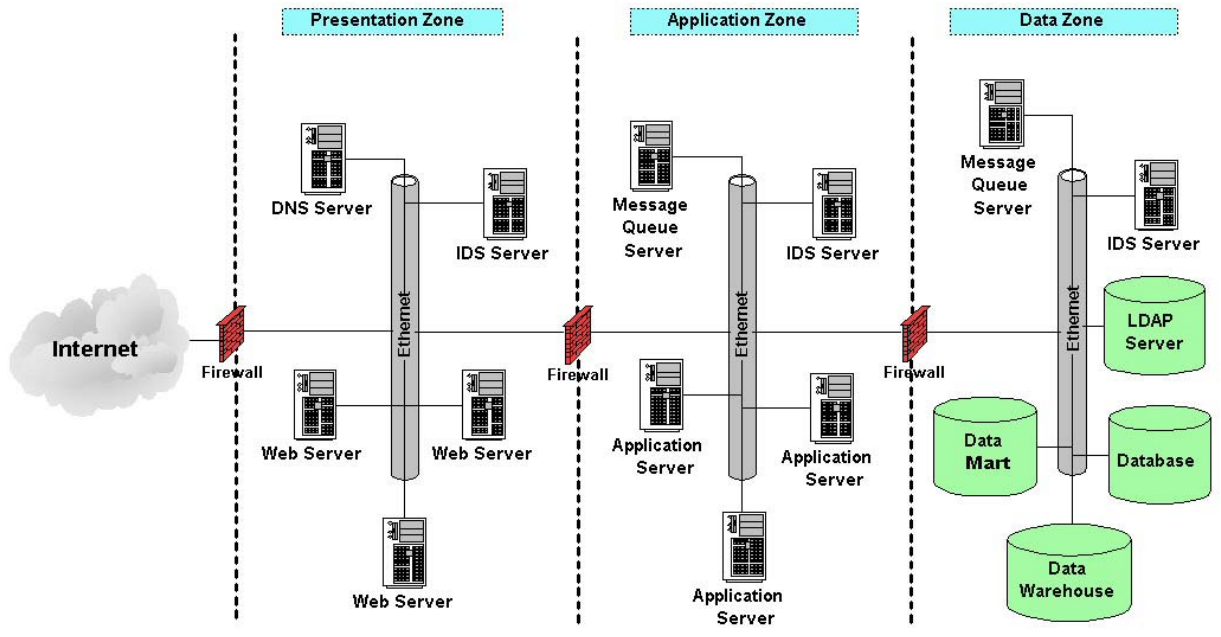

Figure 1. CMS Three-Zone Internet Architecture

An application that is designed for the CMS three-zone Internet architecture should be divided into three layers, each providing a specific application function: presentation logic layer, business logic layer, and data access logic layer. The layers cooperate with each other through well-defined interfaces that allow sharing of critical data and state information.

The CMS three-zone Internet Architecture encompasses the following:

- A thin client at the end user's interface, coupled with a layered approach to system services
- A high-level and seamless service base for internal and external enterprise users and application-processing platforms
- The thin client at the workstation is limited to a browser. A multi-layer Technical Infrastructure supports the layered approach that complies with the specification and design of the application systems and services.
- The Technical Infrastructure describes the technologies that will be assembled into a system to support CMS Enterprise application-processing requirements. The Technical Infrastructure consists of a number of segments, each providing a distinct service.

The *CMS Internet Architecture* provides additional detail of the CMS three-zone Internet Architecture.

2.2 CMS Applications Environment

CMS' objectives for its target applications infrastructure are as follows:

- Move to a storage area network (SAN) storage environment
- Create an integration and test environment (I&TE) to support pre-production verification and validation (V&V) of systems before production. This I&TE environment will allow CMS to perform load and performance testing prior to installing the system in the production environment.
- Create standards for infrastructure operations in conjunction with applications deployments.

Currently, CMS provides a testing environment directly accessible by application developers. Application developers may load and test the integration of their software with the CMS common software utilities in this multi-tier environment.

The following subsections describe the CMS Test and Production Environments, which operate under a formal change control process**.**

2.2.1 Test Environment

The CMS Test Environment is accessible to application developers for testing the functional capabilities of application software within the CMS environment. Using a formal change control process, CMS promotes applications from the Development Environment. Functional testing includes clustering, fail-over, and load balancing. All hardware and software must be tested first in the prescribed Test Environment before production. Regression testing is also required before updates are implemented on the production system.

2.2.2 Production Environment

The CMS Production Environment offers 24x7x365 support and follows Standard Operating Procedures (SOP) and regular maintenance. Application teams do not have access to the production system. CMS Change Management policies apply to both the Test and Production Environments.

2.3 CMS Infrastructure

The CMS Infrastructure, which is currently under development, provides a flexible and scalable environment to support the CMS mission and business processes. It allows no single point of failure and features distributed load balancing. The CMS Infrastructure must:

- Enable secure access, irrespective of the specific network topology
- Adjust to a continually changing business environment and provide the flexibility to adapt to changes in regulatory constraints
- Permit technology refreshment and upgrade of CMS technical capabilities without impacting ongoing operations.

The following subsections describe the standard products that CMS has selected to support J2EE enterprise applications in the CMS Infrastructure. The *CMS Target Architecture* specifies the approved commercial off-the-shelf (COTS) products (hardware and software) that otherwise support the CMS environment.

2.3.1 Operating Systems

CMS has selected the following operating system (OS) standards for mainframe and mid-level environments:

- **Mainframe:** IBM z /OS is the standard OS for the mainframe platform used for high-end corporate data servers.
- **Mid-Level:** Sun Solaris is the standard OS for all new server-class machines.

2.3.2 Relational Database Management System

CMS has selected the following relational database management systems (RDBMS) for the CMS Environment:

- **Mainframe:** The RDBMS for mainframes is IBM DB2. The mainframe servers are used for large applications and legacy applications. These servers will be configured for both high performance and maximum practical availability. They function primarily as large-scale, back-end data servers.
- **Mid-Level:** Oracle is the RDBMS for other applications.

2.3.3 Web Server and Web Application Server

The CMS standards for web server and web application servers are:

- **Sun or IBM HTTP Server:** Sun or IBM HTTP Servers serve static hypertext markup language (HTML) based on hypertext transport protocol (HTTP) requests. In the case where the data and/or the programming logic are not local to the Web Server or where the HTML is dynamically composed and may require programming logic and/or data, these services are requested from the Web Application Server in the Application Zone. HTML files that are requested by the Web Servers are passed to them from the file servers in the Application Zone. The multiple elements incorporated in the Web Server provide detailed audit logging capability.
- **WebSphere Application Server:** WebSphere Application Servers serve dynamic Web content through the Web Servers to the users. Web Application Servers accept business information from the Business Logic Layer and provide page formatting and content integration on a single page. All dynamic page content is generated at this level, but no business logic or business data is stored at this level.

2.3.4 Messaging Services

The CMS standard for messaging services is:

- **WebSphere MQ:** WebSphere MQ is the enterprise standard and transport for all messaging services. WebSphere MQ enables application integration by helping business applications exchange information across different platforms by sending and receiving data as messages.

All custom applications need to communicate through an Enterprise Messaging Broker built on top of WebSphere MQ. This Enterprise Messaging Broker performs messaging delivery services through a secure messaging interface to Enterprise data and provides appropriate audit trails. The Enterprise Messaging Broker should support messaging, workflow, and data transformation between loosely coupled application components. All COTS components should adhere to either Java Messaging Services (JMS) or MQ-standard application programming interfaces (API) to enable them to fit into the CMS J2EE messaging infrastructure.

2.3.5 COTS Products

CMS encourages the use of COTS in lieu of custom application development. COTS products must be compatible with and support the target CMS 3-zone Internet architecture.

A useful reference on selecting COTS products is the set of lessons learned described in "Assessing the Risks of Commercial-Off-The-Shelf Applications" from the Information Technology Resources Board, Revised Version, December 1999 (www.itrb.gov).

SAS and Cognos are appropriate examples of COTS products. SAS is a standard product used for statistical analysis, and Cognos is well-established business intelligence software.

3. CMS J2EE Application Infrastructure

CMS has adopted J2EE as the standard for application development in the CMS Infrastructure. J2EE uses a multi-tiered distributed application model, where application logic is divided into components, according to its function. The various components that make up a J2EE application are installed on different machines and mapped into the three-zone architecture. The successful design and integration of applications into the CMS three-zone Internet Architecture will require an understanding of the CMS enterprise application and infrastructure standards.

In addition, the CMS Infrastructure provides functionality of the J2EE specification through incorporation of the WebSphere application server. Typically, this deployment encompasses all of the functionality that WebSphere provides, including:

- Java Messaging Service (JMS)
- WebSphere MQ
- Java Database Connectivity (JDBC)[1]
- Java Transaction API (JTA)
- Remote Method Invocation (RMI)
- Internet Inter-Orb Protocol (IIOP)
- Common Object Request Broker Architecture (CORBA).

3.1 J2EE Layers

J2EE applications are made up of components in three loosely coupled layers. A J2EE component is a self-contained functional software unit that is assembled into a J2EE application with its related classes and files and that communicates with other components. The J2EE specification defines the following J2EE components:

- **Application clients:** mainly a browser that runs on the client machines
- **Java Servlet and JavaServer Pages (JSP) technology components:** web components that run on the Web Application Server
- **Enterprise JavaBeans (EJB) components (enterprise beans):** business components that run on the Web Application Server.

J2EE components are compiled and assembled into a J2EE application, verified to be well formed and in compliance with the J2EE specification, and deployed to production, where they are run and managed by the J2EE server.

Figure 2 shows the three layers of a J2EE application within the three-zone architecture. The following subsections discuss the mapping of the logical application layers into the physical three-zone architecture.

[1] The use of JDBC is potentially limited due to the inclusion of MQ and WebSphere Business Integration Message Broker (WBI Message Broker).

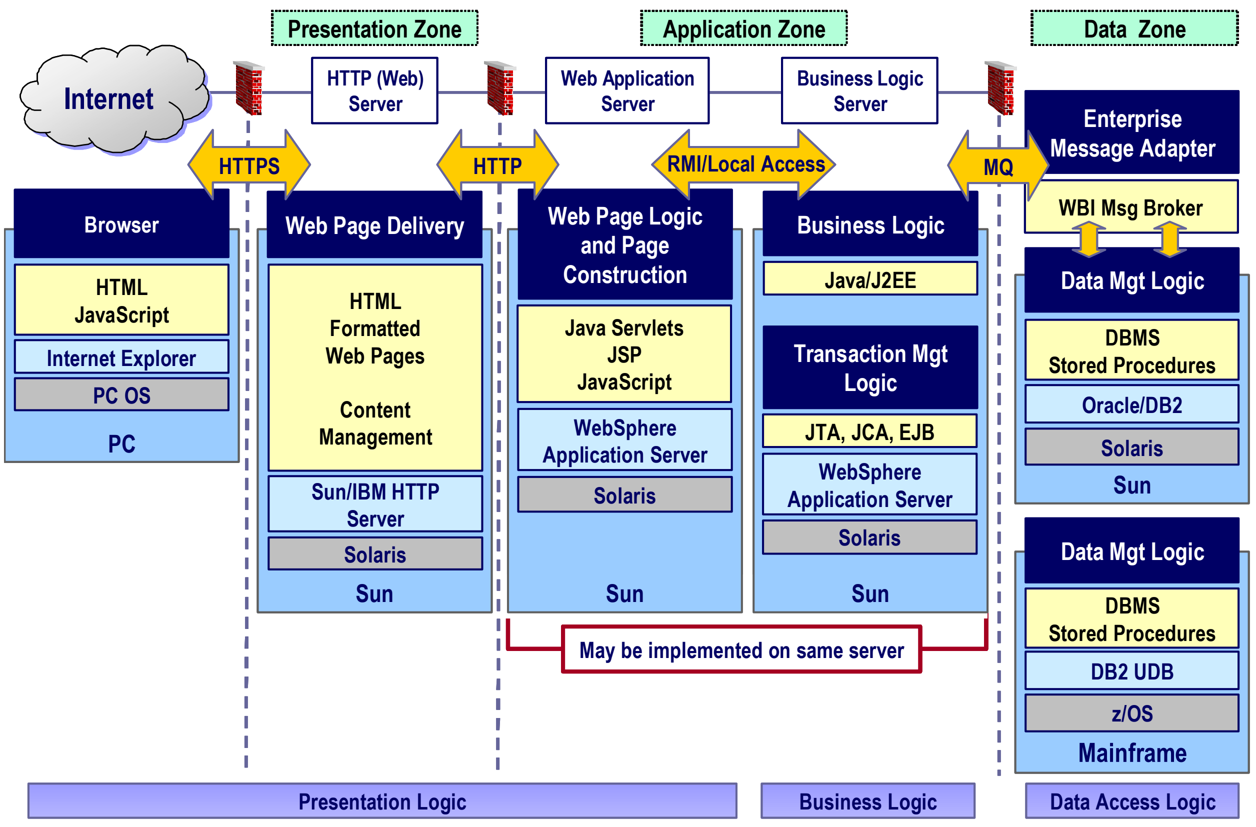

Figure 2. CMS J2EE Application Infrastructure

3.2 Presentation Layer

WebSphere-based Java Servlets and JSP will be used in the presentation layer development, but will not be used for any business logic layer development. The only approved product for Java-based applications is the WebSphere Applications Server. This layer is mapped into the Presentation Zone shown in Figure 1.

3.2.1 Web Clients

A web client consists of a web browser, which renders the pages received from the server. Developers should use best practices, and J2EE standards where applicable, for the graphical user interface (GUI) Elements. Use of applets is not allowed. Because of their security implications, applets typically violate CMS security principles. Developers should use servlet-based applications when dealing with dialogs, forms data, and information.

3.2.2 Web (HTTP) Server

The Web Server serves static HTML based on URL requests. In the case where the data and/or the programming logic are not local to the Web server or where the HTML is dynamically composed and may require programming logic and/or data, these services are requested from the Web application layer. HTML files that are requested by the Web servers are passed to them from the file servers in the application layer. The multiple elements incorporated in the Web server provide detailed audit logging capability and collection of output. Web Servers store and manage static content for user-facing applications.

Web servers present static content (such as Web pages). The common look and feel of the CMS' Web pages is enforced through the IBM Content Manager. Customization of Web page presentation is available, as appropriate to suit the application and an individual's roles, permissions, and personal preferences. Individual home pages can be created for users and personalized subject to the requirements of Section 508 of the Rehabilitation Act of 1973. These requirements are available at: http://www.usdoj.gov/crt/508/508home.html. Application- or project-specific Web servers or content managers are prohibited. The Web servers will also house several plug-ins/interfaces that will provide required services for identification, authentication and access control, incident detection, system management, and connections to various COTS servers.

3.2.3 Web Application Server

The WebSphere Application Server is the enterprise standard Java- and CORBA-compliant application server that integrates enterprise data, applications, and transactions while utilizing open technologies and APIs.

The WebSphere Application Server supports the complete set of J2EE standards and supports Business Logic and Presentation Logic for the CMS Enterprise Architecture (EA). The CMS EA supports Business Logic and Presentation Logic on physically separate tiers with different J2EE capabilities per tier. The description in this subsection concerns the subset of J2EE and other Java capabilities permitted in the Presentation Logic Tier resident on the Web Application Server Tier.

The CMS Application Security Services will be provided within the CMS enterprise by IBM Tivoli PolicyDirector and Tivoli Identity Management.

Usage Guidance and Limitations

The application server is a Java-based, J2EE engine, which supports a full range of enterprise data transactions on UNIX platforms. As the foundation of the WebSphere software platform, the WebSphere Application Server manages e-business applications from dynamic Web presentation to sophisticated transaction processing. This product is the standard application server to be used enterprise wide.

For the Web Application Server, WebSphere Application Server supports the following J2EE/J2SE standard capabilities:

- **J2EE**
  - JSPs
  - Servlets
  - Java Naming and Directory Interface (JNDI) – Discovery
  - Java Messaging Service (JMS) – over WebSphere MQ
  - Java Architecture for XML (eXtensible Markup Language) Binding (JAXB)
  - Java API for XML Processing (JAXP)

Java API for XM (JAX)-RPC
SAAJ
J2EE Connector Architecture (JCA)
JMX – Java Management interfaces implemented over Tivoli
Javax.transform – XML transforms
J2EE Application Deployment

- **J2SE**

XML
Logging
Beans
Locale Support
Preferences
Collections
Lang
Util
New Input/Ouput (I/O)
Networking
Server Java Virtual Machine (JVM) on Solaris.

The permissible versions of each of the foregoing specifications are determined by the version of WebSphere Application Server.

Table 1 describes each service that is applicable to this layer and the guideline for its usage.

Table 1. J2EE Standard Services for Presentation Layer

Service	Usage
Naming	
Because J2EE applications are distributed, they need a way to look up and access remote objects and resources, such as Enterprise Java Beans (EJB). This is supported via the Java Naming and Directory Interface (JNDI).	The JNDI API is the standard API for naming and directory access. The JNDI API has two parts: an application-level interface used by the application components to access naming and directory services and a service provider interface to attach a provider of a naming and directory service.
Messaging	
J2EE provides Java Messaging Service (JMS) to asynchronously send and receive messages. JMS is for program-to-program messages.	Messaging is used for all inter-application communications. It provides a common framework for integrating disparate applications and systems, including legacy systems. Messaging shall be used for the following tiers of application-to-application communication: • For communications among individual applications • For communicating among server processes and other applications, even if the processes were on the same

Service	Usage
	machine • For any inter-process communication in which an immediate response is not needed, or no response is needed, e.g., sending event messages to an auditing application • When communicating with legacy systems in situations that require more interaction than bulk file transfers
Security	
J2EE supports both declarative and programmatic security. It provides the Java Authentication and Authorization Service (JAAS) to authenticate and enforce access controls upon users.	The Java Authentication and Authorization Service (enables services to authenticate and enforce access controls upon users. It implements a Java technology version of the standard Plugable Authentication Module (PAM) framework, and extends the access control architecture of the Java 2 Platform in a compatible fashion to support user-based authorization. The Java Authorization Service Provider Contract for Containers (JACC) defines a contract between a J2EE application server and an authorization service provider, allowing custom authorization service providers to be plugged into any J2EE product.
Communication	
J2EE supports the following protocols: • Internet protocols—These include TCP/IP, HTTP 1.0, and Secure Socket Layer (SSL) 3.0 • RMI (Remote Method Invocation) protocols—RMI is a set of APIs used by Java-distributed applications, including EJBs.	The HTTP client-side API is defined by the java.net package. The HTTP server-side API is defined by the servlet and Java Server Page (interfaces. HTTPS: Use of the HTTP protocol over the SSL protocol is supported by the same client and server APIs as HTTP.
eXtensible Markup Language (XML) Transforms	JAXP provides support for the industry standard Simple API for XML (SAX) and Document Object Model (DOM) APIs for parsing XML documents, as well as support for eXtensible Stylesheet Language Transformation (XSLT) engines.
Java Management Extensions (JMX) Java Management interfaces	The Java 2 Platform, Enterprise Edition Management Specification defines APIs for managing J2EE servers using a special management enterprise bean. The Java Management Extensions (JME) API is also used to provide some management support.

3.3 Business Logic Layer

Java Beans and EJB shall be used for development of business and transaction logic in the Business Logic Layer. WebSphere MQ messaging shall be used for all intersystem communication between the Application and Data Zones.

Interface between the Business Logic and Presentation Logic Layers within the Application Zone shall be through the use of RMI or Local Interface. The restriction on the use of Java Database

Connectivity (JDBC) will prevent the use of Entity Beans in the Business Logic Layer. The use of WebSphere MQ instead of JDBC will promote loose coupling between the Business Logic and Data Access Logic.

The Business Logic Layer is mapped into the Application Zone shown in Figure 1.

3.3.1 WebSphere Application Server

WebSphere Application Server is the enterprise standard Java- and CORBA-compliant Web and application server that integrates enterprise data, application, and transactions while utilizing open technologies and APIs. It supports the complete set of J2EE standards and supports Business Logic for the CMS EA. This description concerns the subset of J2EE and other Java capabilities permitted in the Business Logic Layer.

For the Business Logic Server Layer, WebSphere Application Server supports the following J2EE/J2SE standard capabilities:

- **J2EE**
  EJBs
  RMI – EJBs interact over RMI
  JNDI – Discovery
  JMS – over WebSphere MQ
  JDBC[2]
  JAX-RPC (but over MQ transport, not HTTP)
  SOAP (Simple Object Access Protocol) with Attachments API for Java (SAAJ)
  JCA
  JMX – Java Management interfaces implemented over Tivoli
  Javax.transform – XML transforms
  JTA
- **J2SE**
  XML
  Logging
  Beans
  Locale Support
  Preferences
  Collections
  JNI
  Lang
  Util

[2] The use of JDBC is potentially limited due to the inclusion of MQ and WebSphere Business Integration Message Broker (WBI Message Broker).

New I/O

Networking

Server JVMs on both Solaris and Windows.

Usage Guidance and Limitations

As the foundation of the WebSphere software platform, the WebSphere Application Server manages e-business applications from dynamic Web presentation to sophisticated transaction processing. This product is the standard application server to be used enterprise wide.

Table 2 describes each service that is applicable to this layer and the guideline for its usage.

Table 2. J2EE Standard Services for Business Layer

Service	Usage
Naming	
Because J2EE applications are distributed, they need a way to look up and access remote objects and resources, such as EJBs and data sources. This is supported via the Java Naming and Directory Interface (JNDI).	The JNDI API is the standard API for naming and directory access. The JNDI API has two parts: an application-level interface used by the application components to access naming and directory services and a service provider interface to attach a provider of a naming and directory service.
Data Access	
J2EE supports both declarative and programmatic data access. It provides the JDBC API for connectivity with relational database systems. It provides the Connector architecture (resource adapters) to give applications uniform access to various kinds of enterprise information systems.	Data interchange is managed as an Enterprise common service because all application access to CMS data stores must be through these services. This access provides data abstraction to application developers and isolates them from the underlying details of database structures and the data manipulation languages. Data management and data optimization activities are contained within the data interchange service layer, allowing database specialists to independently construct and maintain these services without impact to the applications that use them. These services also include wrapper services such as security and privacy control. logging of data access requests, and error checking. The use of Java Database Connectivity (JDBC) is potentially limited due to the inclusion of MQ and WebSphere Business Integration Message Broker (WBI Message Broker).
Security	
J2EE supports both declarative and programmatic security. It provides the Java authorization to enforce access controls upon users.	The Java Authentication and Authorization Service (JAAS) enables services to authenticate and enforce access controls upon users. It implements a Java technology version of the standard Plugable Authentication Module framework, and extends the access control architecture of the Java 2 Platform in a compatible

Service	Usage
	fashion to support user-based authorization. The Java Authorization Service Provider Contract for Containers (JACC) defines a contract between a J2EE application server and an authorization service provider, allowing custom authorization service providers to be plugged into any J2EE product.
Messaging	
J2EE provides JavaMail and the Java Messaging Service (JMS) to asynchronously send and receive messages. JavaMail is for e-mail messages. JMS is for program-to-program messages.	Mail messages should be accomplished via the WBI Message Broker and SMTP adapter.
Communication	
Remote Method Invocation (RMI) protocols—RMI is a set of APIs used by Java distributed applications, including EJBs. Object Management Group (OMG) protocols—allow J2EE applications to communication with remote CORBA objects.	J2EE applications can use RMI-Internet Inter-Orb Protocol (IIOP), with IIOP protocol support, to access CORBA services that are compatible with the RMI programming restrictions. Such CORBA services would typically be defined by components that live outside of a J2EE product, usually in a legacy system. Only J2EE application clients are required to be able to define their own CORBA services directly, using the RMI-IIOP APIs. Typically, such CORBA objects would be used for callbacks when accessing other CORBA objects.

3.4 Data Access Logic Layer

The Data Access APIs provide a layer of abstraction between the applications and the data in the Data Zone as well as the location and structure of the data stores. The Data Access APIs abstract the users of high-level business logic partitions from the low-level mechanics of data access—such as Structure Query Language (SQL)—and transaction mechanisms, such as commit, rollback, or cursor control. The business logic partitions of the applications do not perform direct SQL procedures on the databases. Instead, they invoke the Data Access APIs to access data (Create, Read, Update and Delete). The Data Access APIs will support data retrieval for read-only use and creating/updating/deleting data.

Data interchange is managed as an Enterprise common service because all application access to CMS data stores must be accomplished through these services. This access provides data abstraction to application developers and isolates them from the underlying details of database structures and the data manipulation languages. Data management and data optimization activities are contained within the data interchange service layer, allowing database specialists to independently construct and maintain these services without impact to the applications that use them. These services also include wrapper services such as security and privacy control, logging of data access requests, and error checking.

This layer is mapped into the Data Zone shown in Figure 1. Table 3 describes each service that is applicable to this layer and the guideline for its usage.

Table 3. J2EE Standard Services for Data Access Logic Layer

Service	Usage
Naming	
Because J2EE applications are distributed, they need a way to look up and access remote objects and resources, such as EJBs and data sources. This is supported via the Java Naming and Directory Interface (JNDI).	The JNDI API is the standard API for naming and directory access. The JNDI API has two parts: an application-level interface used by the application components to access naming and directory services and a service provider interface to attach a provider of a naming and directory service.
Data Access	
J2EE supports both declarative and programmatic data access. It provides the Java Database Connectivity (JDBC) API for connectivity with relational database systems. It provides the Connector architecture (resource adapters) to give applications uniform access to various kinds of enterprise information systems.	Data interchange services are provided in three layers: (1) Messaging and data access services (2) Enterprise application integration (EAI) messaging (3) Data access services (DAS) The functions of the Messaging and DAS are: • Logging • Error checking and handling • Security and privacy The use of JDBC is potentially limited due to the inclusion of MQ and WBI Message Broker.
Transaction	
J2EE supports both declarative and programmatic transactions. It provides the Java Transaction API (JTA) to handle transaction processing.	Online transactional data processing must be to authoritative data stores through the DAS.
Messaging	
J2EE provides JavaMail and the Java Messaging Service (JMS) to asynchronously send and receive messages. JavaMail is for e-mail messages. JMS is for program-to-program messages.	Mail messages should be accomplished via the WBI Message Broker and SMTP adapter.

4. Common Infrastructure Services and Preferred Design Patterns

This section describes common infrastructure services and preferred design patterns that are applicable to the CMS J2EE environment. Projects shall design applications using the design patterns and implement their applications using the available infrastructure services. This section concludes with a description of best practices and applicable recommendations for CMS in-house developers and contractors.

4.1 Infrastructure Services

At present, CMS offers application security services in support of application development. The following subsection addresses the J2EE Application Security Services.

4.1.1 J2EE Application Security Services

Application Security Services are the security services that are available to COTS applications, developers, and eventually, to the end user.

Security services should allow applications to abstract away security functions from business logic. This helps to minimize the complexity that application developers and integrators have to deal with when developing and deploying secure applications. In some cases, applications have to enforce their own security, such as fine-trained access control that is tied to complex business logic. Typically, the CMS Application Security Services would intercept client requests and enforce security policy in a manner that is transparent to the applications.

Table 4 lists the CMS Application Security Services, the applicable technology, and sensitivity.

Table 4. Technology and Sensitivity of CMS Application Security Services

CMS Application Security Services	Security Technology	Sensitivity		
		Low	Med	High
Identification & Authentication	UserID / Password		X	X
	UserID / Password Plus Hardware or Software Token			X
	Single Sign-On		X	X
	Registration		X	X
Authorization/Access Control	Role Based		X	X
Cryptography	Data Encryption		X	X
	Digital Signature		X	X
	Cryptography Infrastructure		X	X
Audit	Audit Logs	X	X	X

Whenever possible, application developers should leverage the J2EE authentication model and J2EE roles in conjunction with their specific extended rules. If a complex business rule is needed to make a security decision, developers may write the code to implement it.

In most cases, CMS applications will depend entirely on the Security Services of the CMS Infrastructure for their security needs. The application Risk Assessment, System Security Plan, and Security Certification and Accreditation documentation are to be prepared under the assumption that all required capabilities of the CMS Infrastructure Security Services are in place, fully operational, and functioning correctly.

It is expected that applications will rarely require custom security measures beyond available Security Services. Any custom security measures must be approved by CMS management on a case-by-case basis.

CMS has chosen Tivoli Access Manager as the standard product for the application security services. The Tivoli Access Manager is a policy-based access control solution for enterprise applications. Tivoli Access Manager has a Java run-time component, which uses the WebSphere Application Server version of the Java run time.

Special Considerations

- Tivoli Access Manager requires the use of IBM LDAP Directory Server as the user registry
- The login polices set in Tivoli Access Manager are honored only when the user logs in with a password.

Detailed Description

By providing authentication and authorization APIs and integration with application platforms such as J2EE, Tivoli Access Manager helps to secure access to business-critical applications and data spread across the extended enterprise.

Tivoli Access Manager helps provide a self-protecting environment through:

- Prevention of unauthorized access by using a single security policy server to enforce security across multiple file types, application providers, devices, and protocols
- Web-based Single Sign-on
- Robust auditing capabilities.

4.1.2 Future Infrastructure Services

It is anticipated that CMS will provide other infrastructure services in support of application development.

4.2 Preferred Design Patterns

This section describes the most important patterns for the design of CMS J2EE applications: Model-View-Controller, Messaging Gateway, and Messaging Mapper. As CMS infrastructure evolves, additional design patterns may be added.

4.2.1 Model-View-Controller

The Model-View-Controller design pattern is central to the successful adoption of J2EE and is fundamental to the design of good J2EE applications. It is simply the division of the application into the following parts:

- Those parts responsible for business logic (the ***Model***—often implemented using EJBs or simple Java objects)
- Those parts responsible for presentation of the user interface (the ***View***—usually implemented with JSP and tag libraries, but sometimes with XML and XSLT).
- Those parts responsible for application navigation (the ***Controller***—usually implemented with Java Servlets or associated classes like Struts controllers).

The **Model** is the application's business logic and data. This is any of the functional logic that CMS business applications use. This code and data is conceptually independent of the View and the Controller; it is possible to have one Model supporting both desktop and web applications (i.e., multiple presentation layers). Alternatively, multiple applications may access the same business logic.

The **View** is that part of the application that displays information to the user. In a web application, the display is on the screen of the remote user. Thus, the view is the web page returned to the user following the request.

The **Controller** is that part of the application that manages the user input. In a desktop application, input can be keystrokes or mouse clicks. In a web application, input is most likely an HTTP request.

Problem

An application design must have a strategy for serving current and future client types. An architecture pattern can be used to structure the application into isolated parts to provide greater flexibility, reuse, and ease of maintenance.

Solution

A major consideration for using MVC design patterns is that the user interface is the element most likely to change in any relatively mature application. An application may be required to display an additional table field, or may need the application to be accessible in multiple languages**.**

If the user interface is kept separate from the application's business logic, then changes to the application interface do not run the risk of breaking that logic. At the same time, the separation of the user interface from the back-end logic allows changes in logic without affecting the front-end interface. CMS recommends that applications for the CMS Infrastructure environment be designed with this principle in mind.

Figure 3 illustrates the MVC design pattern as applied to Java-Based Web Development.

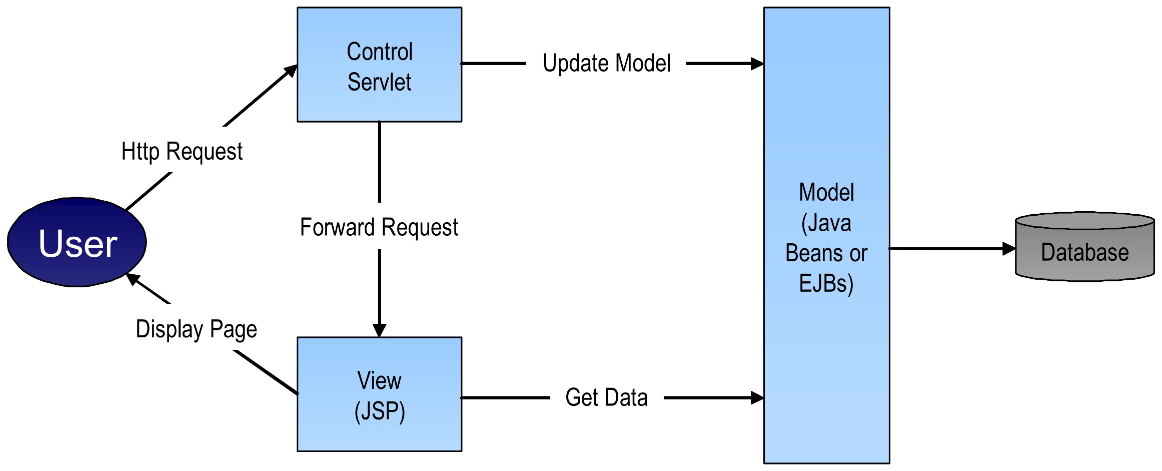

Figure 3. Model-View-Controller Design Pattern

In the CMS environment, Common Gateway Interface (CGI) on the Web server is not allowed because of security considerations. Thus, only a Servlet is allowed over CGI.

Typical uses for an HTTP Servlet include:

- Processing and/or storing data submitted by an HTML form
- Managing state information on top of the stateless HTTP (e.g., for an online shopping cart system that manages shopping carts for many concurrent customers)
- Implementing Web Services on a transactional basis.

In order to function as a controller in an MVC configuration, a Servlet must be able to trigger the display of a JSP. This can be done in two ways: the first way is by sending an HTTP redirect request back to the user, indicating which page the user's browser must now request. Redirecting is not recommended in the CMS Infrastructure. The redirect requires an additional request by the user's browser, which takes extra time. Redirecting from the first to the second request may require passing data through the URL, which may cause some security risk.

The second, and far superior, method is to use the Request Dispatcher. There is ample guidance available for using this tool in the literature on implementing Servlets and JSPs.

Developers should separate Business Logic (Java beans and EJB components) from Controller Logic (Servlets/Struts actions) and from Presentation (JSP, XML/XSLT). The use of the entity beans is potentially limited due to the inclusion of MQ and WBI Message Broker.

4.2.2 Messaging Gateway

The Messaging Gateway encapsulates messaging-specific code (e.g., the code required to send or receive a message) and separates it from the rest of the application code. Only the Messaging Gateway code knows about the messaging system; the rest of the application code does not. The Messaging Gateway shields the application developer from vendor-specific APIs. As an example, a Messaging Gateway exposes domain-specific methods such as GetMedicareID that accept strongly typed parameters.

A Messaging Gateway sits between the application and the messaging system and provides a domain-specific API to the application (see Figure 4). Because the application does not know that it is using a messaging system, the gateway can be transparently replaced with a different implementation that uses another integration technology, such as Web services.

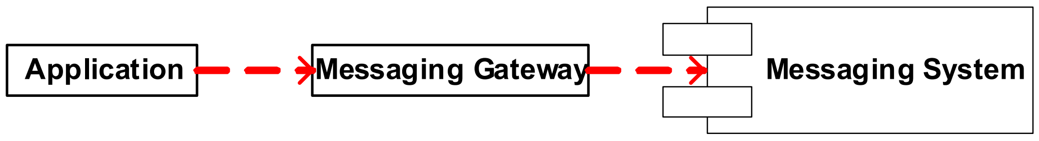

Figure 4. Messaging Gateway

Problem

The CMS Internet Architecture guide states that no direct database interaction may be initiated from the Application Zone. All queries must be handled via a messaging system between the Application and Data Zones. The problem is encapsulating access to the messaging subsystem from the rest of the application.

Solution

The preferred solution is to use a Messaging Gateway, a class that wraps messaging-specific method calls and exposes domain-specific methods to the application. In general, an application should not be aware that it is using a messaging system for enterprise data integration or integration to external systems or resources. Most of the application's code should be written without messaging in mind. At the points where the application integrates with the messaging subsystem, there should be a thin layer of code that performs the application's part of the integration. When the integration is implemented with messaging, the Messaging Gateway is that thin layer of code that attaches the application to the messaging system.

The other advantages of the Messaging Gateway pattern are as follows:

- **Encapsulate Asynchronous Nature of Messaging System:** Messaging systems are inherently asynchronous. This can complicate the code to access an external function over messaging. The Messaging Gateway can expose a simpler function with synchronous semantics, which encapsulates the asynchronous nature of the request and reply messages.
- **Application-Specific Error Handling:** In addition to simplifying the coding of the application, the Messaging Gateway also eliminates dependencies of the application code on specific messaging technologies. The Messaging Gateway can catch all message-specific exceptions and throw application-specific (or generic) exceptions instead. This is very helpful if it is ever necessary to switch the underlying implementations, e.g., from JMS to Web services.

- **Improved Unit Testing:** Messaging Gateways make excellent testing vehicles. By separating the interface from the implementation, two implementations are available: one "real" implementation that accesses the messaging system and a "fake" implementation for testing purposes.

4.2.3 Messaging Mapper

The Messaging Mapper, as shown in Figure 5, accesses one or more domain objects and converts them into a message as required by the messaging channel. It also performs the opposite function, creating or updating domain objects based on incoming messages. Since the Messaging Mapper is implemented as a separate class that references the domain object(s) and the messaging layer, neither layer is aware of the other. The layers also are unaware of the Messaging Mapper. The Messaging Mapper can be invoked by the Messaging Gateway described in Subsection 4.2.2.

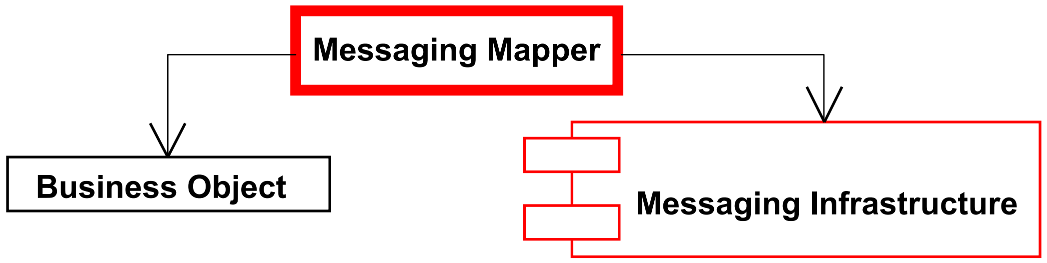

Figure 5. Messaging Mapper

Problem

How can data be moved between domain objects and the messaging infrastructure while keeping the two independent?

Solution

The preferred solution is to create a separate Messaging Mapper that contains the mapping logic between the messaging infrastructure and the domain objects. Neither the objects nor the infrastructure have knowledge of the Messaging Mapper's existence.

When applications use messaging, the messages' data are often derived from the applications' domain objects. There are some distinct differences between messages and objects. For example, most objects rely on associations in the form of object references and inheritance relationships. Many messaging infrastructures do not support these concepts because they have to be able to communicate with a range of applications, some of which may not be object-oriented at all.

4.3 Best Practices and Recommendations

CMS has developed these *J2EE Guidelines* based on a set of lessons learned and best practices. CMS in-house developers and contractors should consider these best practices and recommendations in the development and implementation of J2EE-based applications in the CMS Infrastructure.

4.3.1 Static Web Applications

The CMS architecture maintains a heavy focus on security.[3] The greater the depth of the content within the application, the better. Placing images and static content on Web servers (Presentation Zone) is acceptable; however, where content is sensitive (not necessarily secure), it must reside on the Application Server (Application Zone). With this approach, access to such content will only be available through the Web server.

Server-sided applications can utilize client-sided complements of JavaScript for presentation and URL management, but no business rules should be stored on the client. J2EE applications should have all state management on the server, since there may be security implications with storing sensitive information in the Document Object Model of the browser.

4.3.2 Simple Dynamic Web Applications

When delivering a dynamic web site, the developer must take a slightly different perspective on how to separate the standard, most unchanging parts of a web page from the parts that do change. By limiting the number of pages or templates, the web page will be easier to maintain.

No matter how simple or complex the web site, users make requests to the server and the server returns pages to the user. When a database enters the mix, the content of those pages inevitably becomes dynamic. Sometimes even the choice of pages becomes dynamic.

Developers should consider presenting this kind of data to the user through JSP, Java Servlets, or Java Beans. Regardless of the mechanism chosen, developers should be aware that the Servlet engine is the main operator of the Application Server and the central driver for all Servlets, JHTML, and JSP. Java Beans deployment provides encapsulation to the dynamic implementation of their instances and is desirable from an Object-Oriented Design (OOD) perspective.

Developers should carefully consider the implications of concurrent threading and state when developing applications. To ensure that the business pipeline does not put other processes—such as XML parsing—in a wait state while processing, WebSphere threading should be considered carefully during the design phase.

[3] CGI shall neither be designed, developed, nor installed on any of the CMS Web servers.

4.3.3 Files and Directory Structure

Along the lines of the CMS Infrastructure directory structure, application teams should advise on extended and other directory structures to be used. This aids construction of fault-tolerant designs for items like shared volumes, RAID configurations, and consistency.

A cardinal rule of CMS policy is the proscription against overwriting an existing file when making changes to an environment.

Application developers must use a convention to temporarily backup the file or directory. This ensures an immediate rollback and represents an error-reduced practice for changing the environment during a deployment.

The code migration and deployment processes assume that the Java application code will be included in an Enterprise Application Resource (EAR) File. The EAR file is used to bundle all of the components that a Java web application will require. The bundled components include static HTML, images, Servlets, JSP, and EJBs, as well as the deployment descriptors that instruct the container on how to run the application. The EAR file is a standard J2EE application package, and is supported by WebSphere.

The Web Application Resource (WAR) file is used to bundle web application resources that the application requires. WAR files bundle all of the same components that an EAR file bundles, with the exception of the EJBs. WAR files are typically used by smaller applications that only require Servlets and JSP. The WAR file is a standard J2EE application package, and is supported by WebSphere.

The Java Archive file typically contains only Java-class files. In most cases, JAR files are wrapped in an EAR file or WAR file.

4.3.4 Separate Data Management From User Interface

CMS requires that separate procedures be written to manage data and call them from the user interface (UI) code. This allows the developer to redesign the interface (a more common change) without redesigning the data management code.

The business logic is collected in Java beans representing OOD and equivalent functionality. The glue and transaction model is represented in the JSP implementation. JSP covers the transaction model, fault tolerance, and load balancing (scalability) aspect. Meanwhile, the browser behavior and presentation layers can be managed by JSP presentation templates or passed through Java script for rendering on the respective DOM.

4.3.5 Principles for the Data Architecture

The following principles guide the CMS data architecture:

- Data is an enterprise resource that is not owned by any application or by any organization other than CMS
- Data needed by CMS applications must be obtained and stored within CMS Data Zone
- Data stores are to be loosely coupled with applications

- A single copy of data must be designated as the authoritative source for data—any other copy or replication of data must be designated as non-authoritative
- Data must be kept secure
- Data must be easily and quickly available for authorized applications and users
- Access to the data must be independent of configuration and location
- Normal data management levels of service must be provided
- Database management systems, as well as tools and COTS products that use a database management system, must be both scalable and able to handle the extremely large volumes of data inherent to CMS
- Data architecture must be able to transition across succeeding generations of technology and organization.

5. CMS J2EE Standard Development Tools and Products

CMS has selected a set of products as the standard components of the target J2EE infrastructure. Table 2 presents the list of products applicable to CMS utilities and services.

Table 5. J2EE Standard Development Tools and Products

Utilities and Services	Mid-Tier Products/Standards
Operating System	Sun Solaris
Database Management System (DBMS)	Oracle, UDB
Messaging	IBM WebSphere MQ
Application Zone Server	IBM WebSphere for Sun
Presentation Zone Web Server	IBM WebSphere for Sun
Java Developer Tools	J2EE SDK, Developer Pack, WebSphere Studio, Eclipse
Programming Language	Java
Content Management/Unstructured Data	IBM Content Manager
Web Content Management	Stellent Content Server
Data Interoperability	XML and XML Schema
Network Authentication/Access	Java Enterprise LDAP (Sun ONE) or LDAP Proxy
LDAP Authentication	Java Enterprise LDAP (Sun ONE) or LDAP Proxy
Identification and Authentication	Tivoli Identity Management
Authorization/Logical Access Control	Tivoli PolicyDirector and Tivoli Identity Management

Other COTS products that are proposed for use at CMS must be compatible with these products and standards. The *CMS Target Architecture* document (published in September 2004) provides more information on other standard tools and software.

Acronyms

API	Application Programming Interface
CGI	Common Gateway Interface
CMS	Centers for Medicare and Medicaid Services
CORBA	Common Object Request Broker Architecture
COTS	Commercial-Off-The-Shelf
DAS	Data Access Services
DB2	Database 2 (IBM product)
DNS	Domain Naming Services
DOM	Document Object Model
EA	Enterprise Architecture
EAI	Enterprise Application Integration
EAR	Enterprise Application Resource
EJB	Enterprise JavaBeans
GUI	Graphical User Interface
HHS	Health and Human Services
HTML	HyperText Markup Language
IIOP	Internet Inter-Orb Protocol
IT	Information Technology
J2EE	Java 2 Enterprise Edition
JAAS	Java Authentication and Authorization Service
JACC	Java Authorization Service Provider Contract for Containers
JAS	Java Agent Service
JAX	Java API for XML
JAXB	Java Architecture for XML Binding
JAXP	Java API for XML Processing
JCA	J2EE Connector Architecture
JDBC	Java Database Connectivity
JMS	Java Messaginge Service
JMX	Java Management Extensions
JNDI	Java Naming and Directory Interface

JSP	JavaServer Page
JTA	Java Transaction API
JVM	Java Virtual Machine
LDAP	Lightweight Directory Access Protocol
MQ	Message Queue
MVC	Model-View-Controller
OOD	Object-Oriented Design
OS	Operating System
PAM	Plugable Authentication Module
PKI	Public Key Infrastructure
RAID	Redundant Array of Inexpensive Disks
RMI	Remote Method Invocation
SAAJ	SOAP with Attachments API for Java
SAS	Formerly the “Statistical Analysis System,” now the name of the company
SAX	Simple API for XML
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
UDB	Unified Database
UI	User Interface
V&V	Verification and Validation
WAR	Web Application Resource
WBI	WebSphere Broker Integration

List of References

The key Department of Health and Human Services (HHS)/CMS documents and Java/J2EE standards and references relied on in the preparation of the *J2EE Application Development Guidelines* include:

1. *CMS Internet Architecture,* Centers for Medicare and Medicaid Services, July 2003.
2. *CMS Enterprise Messaging Infrastructure*, Centers for Medicare and Medicaid Services, December 2003.
3. CMS Web-Enabled Application Architecture (in preparation), Centers for Medicare and Medicaid Services.
4. CMS Intrusion Detection System Internet Architecture and Design (in preparation), Centers for Medicare and Medicaid Services.
5. CMS Data and Database Architecture (in preparation), Centers for Medicare and Medicaid Services.
6. *CMS Target Architecture*, Centers for Medicare and Medicaid Services, September 2004.
7. *CMS Policy for Software Quality Assurance*, Document Number CMS-CIO-POL-QA001.1, Centers for Medicare and Medicaid Services, July 2002.
8. IEEE/EIA/ISO/IEC 12207 Standard for Information Technology—Software Life Cycle Processes.
9. *Java 2 Platform Enterprise Edition Specification, v1.3*, Bill Shannon, Sun Microsystems Inc., 2001.
10. *Java 2 Platform, Standard Edition, v 1.4.2 API Specification*, http://java.sun.com/j2se/1.4.2/docs/api/index.html
11. Redbook: WebSphere Application Server and WebSphere MQ Family Integration, SG24-6878, http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246878.html
12. *Core J2EE Patterns: Best Practices and Design Strategies* (2nd edition), Deepak Alur, John Crupi, & Dan Malks. Prentice Hall PTR 2003, ISBN 0131422464.
13. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Gregor Hohpe and Bobby Woolf. Pearson Education Inc., ISBN 0321200683.