US Department of Education

Federal Student Aid

Integration Leadership Support Contractor

July 30, 2007

# Task 23 – Architectural Models Phase II

## 23.1 Portal Architectural Model v1

Final

# Table of Contents

# Table of Figures

# Table of Tables

# Section 1: Introduction

This document presents an architectural model that documents and communicates Federal Student Aid's overall architectural vision. There are number of architectural models, each covering a particular solution domain (i.e., architectural area). This document specifically addresses portal architecture, which assist in providing insight into Federal Student Aid's vision for developing and deploying portal applications. Future architectural model documentation will cover architectural areas such as Enterprise Service Bus (ESB) and security architecture. This document also provides a background on Federal Student Aid's enterprise initiatives to achieve the Target State Vision (TSV).

## 1.1 Purpose

Federal Student Aid plans to initiate numerous acquisitions to implement its Target State Vision (TSV). An architectural model is a vehicle for Federal Student Aid to clearly communicate expectations of how solutions offered by vendors should meet the department's overall architectural requirements and vision. This architectural model provides a framework within which solutions will operate.

Federal Student Aid's current technical environment is documented by a number of documents developed by Federal Student Aid Chief Information Officer (CIO), Integration Team, and their contractors over time. A significant challenge for Federal Student Aid is to specify which of these documents is required to support the definition of a specific technical solution and which documents should be included in an acquisition package. Therefore, Federal Student Aid has decided to adopt a standard template, an architectural model that documents Federal Student Aid's architectural expectations. Federal Student Aid will require vendors to abide by this architectural model to better understand Federal Student Aid's technical environment, available resources, and the standards to which they must adhere. The objective of the standardization of all technical architecture documentation is to obtain proposed solutions from vendors that are of higher quality, in compliance with Federal Student Aid standards, and compatible with the Federal Student Aid architectural vision.

## 1.2 Intended Audience and Usage

This document is intended to be used by a technical audience, such as the Federal Student Aid CIO organization or a vendor that is preparing to respond to an Information Technology (IT) solicitation or responsible for building solutions. This document defines the standards, guidelines, and constraints of Federal Student Aid's portal architecture. Vendors will be required to take into account this architectural vision when proposing technology solutions.

## 1.3 Scope

Each architectural model is intended to cover relevant architectural elements for specific architectural areas as part of an enterprise-wide solution. Separate documents will be produced for each architectural area and together the series of documents form a complete reference architecture. Federal Student Aid is currently working on standardizing its programming model and development platform. This document describes the relevant architectural elements of Federal Student Aid's proposed programming model and development platform for building its portal architecture.

## 1.4 Document Organization

The remainder of this document is organized into the following sections:

- **Section 2: Enterprise Vision -** Provides background information on Federal Student Aid's TSV, establishes the enterprise-level architectural principles followed in the development of the model and introduces the architectural model at a high level.
- **Section 3: Portal Architecture -** Presents the portal architectural model along with detailed descriptions of the architectural tiers and key technologies leveraged within each tier.
- **Section 4: Architectural Decisions -** Documents key architectural decisions made by the Federal Student Aid's portal architecture team. These decisions are presented with associated rationale, alternatives considered, and implications of each decision.
- **Section 5: Glossary and Standards -** Identifies and defines key standards and technologies described within the portal architecture model.
- **Section 6: Use Cases -** Presents a dynamic view of the portal architecture model by walking through a generic task that the portal architecture will support.
- **Appendix A: Acronyms -** Defines all acronyms used throughout the portal architectural model.

# Section 2: Architecture Vision

Architectural models are intended to facilitate the successful development of target state solutions. Architectural models accomplish this by providing a reference that both Federal Student Aid and the solution provider can use as a baseline for communicating and architecting a solution. This section begins with a discussion about the Federal Student Aid TSV to provide context for the architectural vision. The remainder of this section describes this architectural vision in detail.

## 2.1 Target State Vision

Federal Student Aid, an office of the U.S. Department of Education administers Federal student financial assistance programs authorized under Title IV of the Higher Education Act (HEA) of 1965, as amended. Its mission is to ensure that all eligible individuals can benefit from federally funded or federally guaranteed financial assistance for education beyond high school. Federal Student Aid accomplished this mission by offering numerous assistance programs that are funded by public and private sources. There are approximately 13 million students that apply for financial aid each year.

Federal Student Aid is in the midst of a major business and systems reengineering effort to create an integrated suite of solutions under the Performance Based Organization (PBO) legislation of 1998. Incorporated within this effort, a TSV has been defined to describe how Federal Student Aid should operate and administer Title IV programs. To achieve the TSV, Federal Student Aid intends to hire system integrators to help build the target technical environment. A combination of several architectural models will facilitate the understanding of Federal Student Aid's overall architectural vision.

The TSV is based upon the following business goals: 1) delivering student aid in an efficient and cost-effective manner; 2) providing the best access to customers; and 3) maintaining appropriate levels of oversight. Federal Student Aid has identified the following four objectives in order to achieve these goals:

- **Integrate and reengineer business processes to improve operational efficiency and effectiveness, and provide a better experience for those who interact with Federal Student Aid.**

  Federal Student Aid will enhance customers' experience by improving the efficiency and effectiveness of business processes. Target state business processes are logically grouped so that they can leverage common process steps and underlying data. Complex business processes, such as those involving the origination and servicing of aid, will be streamlined, and customers will have the ability to view and manage their aid portfolio from a single access point.

- **Improve data quality and integrity to provide enhanced analytics and reporting capabilities.**

  Federal Student Aid has established both the Enterprise Data Management initiative and the Information Framework (IF) initiative to provide the data services needed to support the target state business processes. The Enterprise Data Management initiative will establish the policies, processes and procedures to ensure that Federal Student Aid focuses on data as an asset. The IF will coordinate the cleansing, movement, and integrity of data to provide consistent enterprise-wide data to all customers

and business stakeholders, while minimizing data redundancy and improving data integrity. The IF integrates organizational data (schools, lenders, servicers, and guaranty agencies), person data (applicants, students, and borrowers) and aid data (loans and grants), and provides a host of functional, reporting, and analytical capabilities. Federal Student Aid will have a central point to view and utilize integrated data from multiple systems.

- **Integrate and reengineer information systems to enable target business processes.**

  Federal Student Aid will reengineer and integrate systems to enable the target state business processes. For example, the Integrated Partner Management (IPM) initiative will enable Federal Student Aid to consistently and efficiently manage compliance and oversight for all trading partners. Federal Student Aid will continue reengineering systems to implement other target business processes, such as application processing and servicing, to improve the efficiency and effectiveness of those processes.

- **Implement an integrated, standards-based technical infrastructure that emphasizes enterprise reusable shared assets.**

  Federal Student Aid has adopted a Service-Oriented Architecture (SOA) approach to support implementation of the TSV. SOA enables a collection of standard reusable services that will be shared across all of Federal Student Aid. The TSV technical infrastructure includes the following components:

  - *Portal:* A single access point for online information and services required by Federal Student Aid customers, partners, and the general public.
  - *Enterprise Service Bus:* Provides messaging and integration services to facilitate the use of shared functions and access to shared data through SOA.
  - *Security Architecture:* Provides enterprise security services including access and identity management tools.
  - *Gateway:* Provides a single access point for external partner systems to interact with Federal Student Aid systems and services.

## 2.2 Architectural Principles

Federal Student Aid must collaborate with numerous financial institutions, schools, and government agencies in order to meet the needs of students who qualify for financial aid. This collaboration requires Federal Student Aid to exchange and process large amounts of data each day. Most of Federal Student Aid's applications were developed using proprietary technologies based on legacy platforms and are becoming, or already have become, technologically obsolete. In addition, many of the Federal Student Aid systems are stove-piped, which has resulted in effort, functionality, and data redundancy. Consequently, it is becoming increasingly difficult to keep these systems up to date. The overall talent pool for these obsolete technologies is dwindling and the systems do not conform to modern software development standards and architectures.

Therefore, to guide the development of future technological environments and to craft an architectural vision that mitigates some of these environmental deficiencies, Federal Student Aid is adopting the following key architectural principles:

**Principle 1:** The architecture should facilitate reuse of existing IT assets through creation of modular, component-based systems that conform to a Service-Oriented Architecture.

**Principle 2:** The architecture should facilitate interoperability of systems through the use of SOA enabling standards based technologies, including Simple Object Access Protocol (SOAP), Web Services Definition Language (WSDL) and Business Process Execution Language (BPEL).

**Principle 3:** The architecture should facilitate both vertical and horizontal scalability.

**Principle 4:** The architecture should facilitate high availability through the leveraging of enterprise-class hardware and software.

## 2.3 Architectural Areas

In order to realize the TSV, Federal Student Aid must solve technical challenges in several areas, ranging from the delivery of timely, accurate, and relevant business information in a user-friendly manner to the loading and processing of data in large batches. To this end, Federal Student Aid has created a reference architecture that identifies technical capabilities to provide solutions in these various problem domains.

For each solution domain, or architectural area, an architectural model is documented to communicate Federal Student Aid's vision for each area. These models provide clarity and insight into how a technical problem can be solved within each of these domains. The Federal Student Aid reference architecture is presented in Figure 2-1 below.

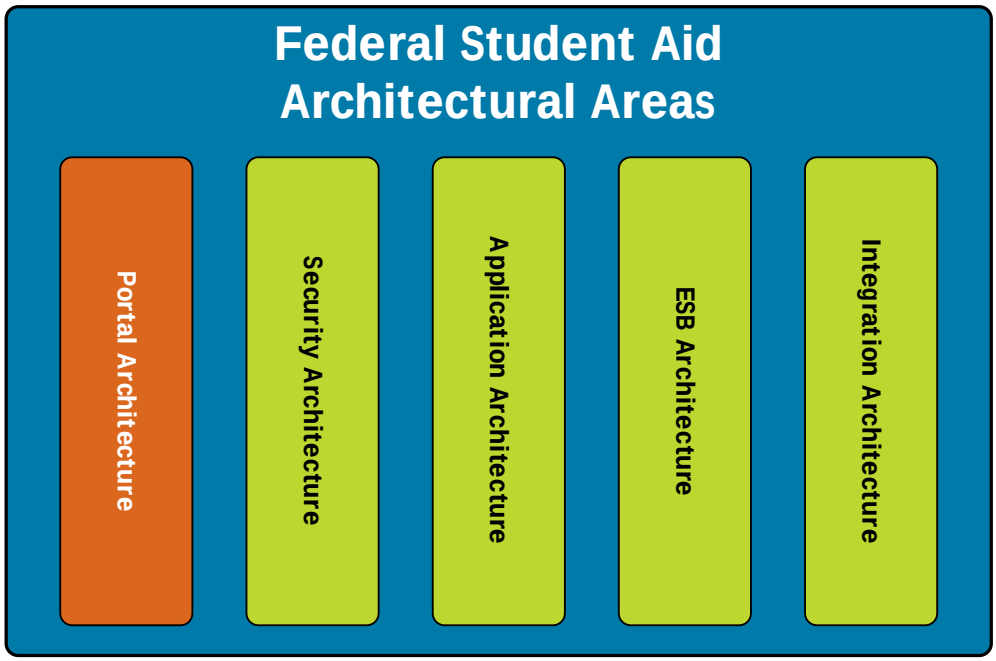


**Figure 2-1: Architectural Areas**

The following bullets describe each architectural area to be addressed by the reference architecture:

- **Portal Architecture** presents Federal Student Aid's architectural vision for crafting solutions that deliver information through enterprise portal technologies.
- **Security Architecture** presents Federal Student Aid's vision to enable secure access to all information technology assets.
- **Application Architecture** presents Federal Student Aid's vision to enable the capture of business logic within a set of enterprise services, business components and data stores to perform transactional and batch data processing operations.
- **ESB Architecture** presents Federal Student Aid's architectural vision for service integration capabilities to coordinate the flow of information between software services and applications.
- **Integration Architecture** presents Federal Student Aid's architectural vision for enabling legacy and Commercial Off The Shelf (COTS) capabilities, such as Siebel Customer Relationship Management (CRM) software, to coordinate flow of information between new and existing systems.

## 2.4 Key Concepts

Federal Student Aid's overall approach to portal architecture is governed by the following key concepts that will be incorporated into future technology solutions:

- **Service-Oriented Architecture:** SOA is a standards-based architectural philosophy supported by numerous technological standards, vendors, and products. Key to SOA is the notion of a software service, which is a set of business and technical capabilities, packaged as a service, and made available for use outside application, division, and even organizational boundaries. SOA is an important element of the overall architecture at Federal Student Aid. Federal Student Aid's architectural aim is to package currently embedded business functionality, repeated across numerous applications, into software services, and to leverage these services in future development efforts. This will allow Federal Student Aid to build an enterprise that is responsive to changing needs, facilitate systems maintenance, and facilitate higher reuse of existing IT assets.

- **Enterprise Service Bus:** An ESB is an integration architecture implemented by technologies found in a category of middleware infrastructure products usually based on web-services standards. The ESB technology provides foundational services for SOA via an event-driven and Extensible Markup Language (XML)-based messaging engine ("the bus").

- **Distributed (Tiered) Architecture:** A distributed architecture divides a solution into logical parts (tiers) based on the separation of concerns principle. For example, user interface features are contained within one tier, business functionality and rule enforcement is contained in a separate tier, and data access functionality is contained in yet another separate tier. This style of architecture has been proven time and again in solutions of varying size and scope to promote greater flexibility, simplified maintenance, and increased reusability of components. This is consistent with Federal Student Aid's architectural principles in Section 2.2.

In addition to the concepts outlined above, Federal Student Aid is interested in establishing a common service oriented programming model for all Java applications. Solution providers will be expected to design their solutions using SOA design principles. It is expected that every portal solution will retrieve data via shared services. Federal Student Aid is considering the adoption of the Service Component Architecture (SCA) as the SOA programming model. SCA brings many benefits to the project team, which includes:

- Simplify the design and deployment of portal applications.

- Establish a "transportable" set of engineering skills for service-oriented architecture (SOA) design.

- Enable many of the static analysis features that developers have come to expect in programming environments, but that have been absent in services (such as dependency analyses and type checking).

## 2.5 Architectural Model Overview

The architecture model is designed to reflect Federal Student Aid's concept of a distributed, multi-tiered approach to developing technology solutions. Each tier in the model provides a unique set of technical capabilities that when combined will meet Federal Student Aid's objectives. The model is depicted in Figure 2-2 below:

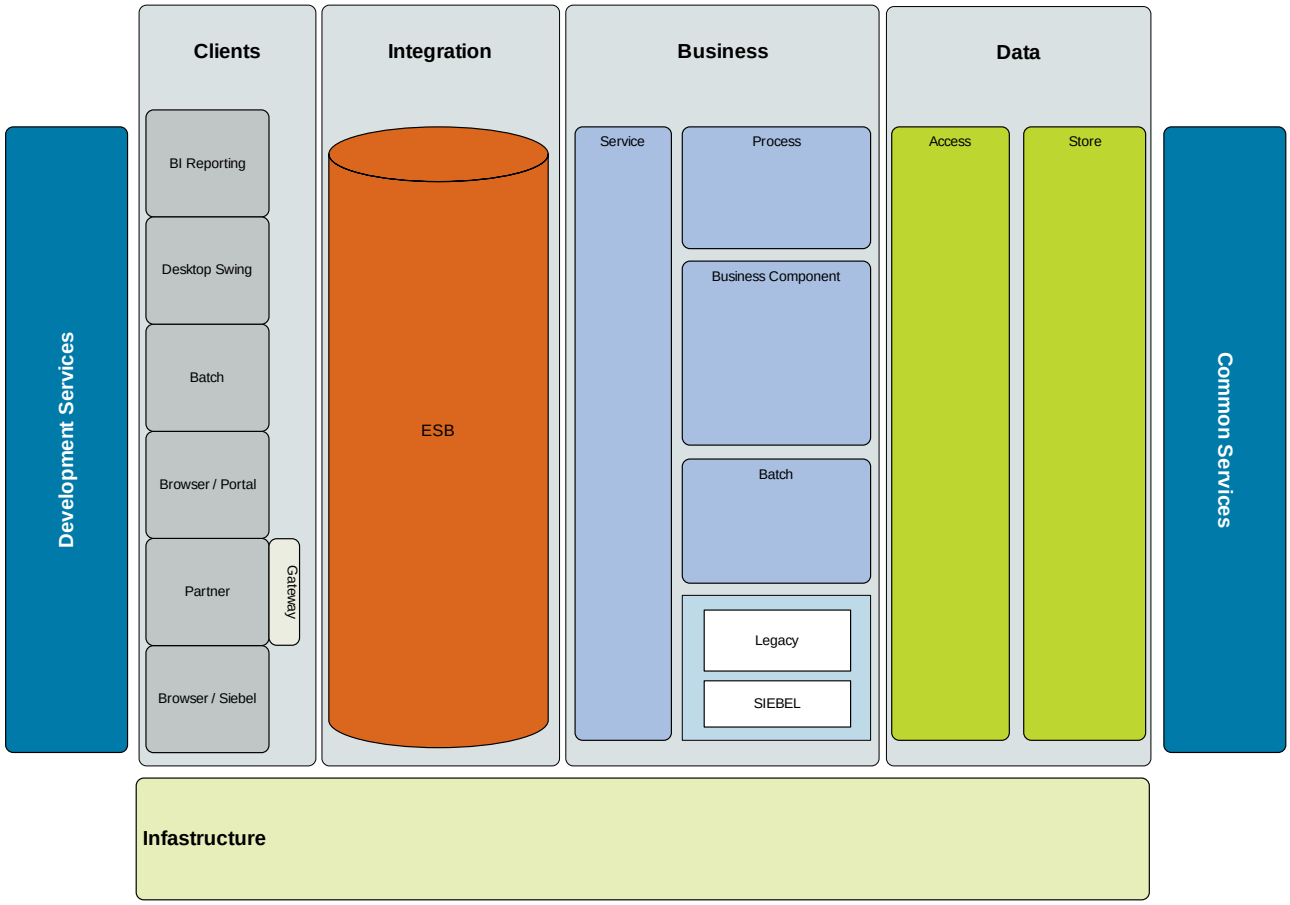


**Figure 2-2: Federal Student Aid Architectural Model**

## 2.6 Architectural Model Tiers

There are four major tiers in the architectural model. Each tier contains several technical elements that enable the capabilities within that tier. In addition to the four tiers, there are three additional areas that do not play a direct role in delivery of business functionality, but are necessary to implement each tier. These areas include development services, infrastructure, and common services. Their descriptions are outside the scope of this document. The relevant architectural tiers are presented below:

### 2.6.1 Client Tier

The Client tier of an application contains the types of users that will be interfacing with the applications. These clients can be also be classified as service consumers, who will access Federal Student Aid services through the ESB. As identified in the Figure 2-2, the architectural model identifies six types of clients that can function as service consumers, as appropriate:

- **Business Intelligence (BI) Reporting**: Enterprise reporting, querying, and analytical solutions which can operate as standalone applications or be embedded within other applications.
- **Desktop Swing**: A Swing software application that is installed on an individual desktop computer.
- **Batch**: An application which initiates batch processing of business data.
- **Web Tier - Portal**: An application that is accessible from a web browser and that may or may not be deployed within an enterprise portal infrastructure.

- **Partner**: An application which interacts through a service, which may be the product of an external business partner or another government agency.

- **Web Tier - Siebel**: The user interface of a COTS software package, such as Siebel.

With the exception of the Web Tier – Portal client, detailed descriptions of these clients and their underlying technologies are outside the scope of this document as they are not considered part of this architecture. These clients are listed to describe the types of clients that will exist in the Federal Student Aid technology environment.

### 2.6.2 Integration Tier

The Integration Tier consists of the ESB and will provide the integration services required to implement Federal Student Aid's SOA. Accordingly, the ESB will provide the reliable and manageable integration services required to facilitate implementation and use of shared functions, and to guarantee timely access to accurate shared data. Moreover, the ESB will bind islands of automation within the Federal Student Aid enterprise into a federated target state business solution that will allow processes and data to be shared and coordinated with minimal restriction.

### 2.6.3 Business Tier

The Business Tier implements the core business functionality and business logic of a solution and facilitates communication between the front end and back end tiers. At Federal Student Aid, the business tier will consist of transactional business components constructed to facilitate data processing. This tier also includes legacy and COTS products that are leveraged to provide application functionality, which may either be exposed as services or called directly through external application programmatic interfaces (API).

For the purposes of the application architecture, all outward-facing components will be exposed as services via the ESB. Outward facing components are those components whose services will be leveraged by external applications. Business components not exposed as external services (i.e., those only used within the context of other components) will be designed to adhere to the service oriented programming model.

### 2.6.4 Data Tier

The purpose of the Data Tier is to facilitate communication between business components and back-end data stores in a controlled manner. The Data Tier is subdivided into two levels as follows:

- **Access**: Consists of all application components that communicate with a database.

- **Data Stores**: Consists of the database schemas with which a given application must interact.

In most situations the business component performs the following three tasks:

- Creates an instance of a related data access component.

- Uses the data access component's interface methods in order to communicate with the database.

- Destroys the instance of the data access component once the necessary output has been received, the necessary operation(s) performed, or a thrown exception has been caught.

# Section 3: Portal Architecture

The portal architectural model provides a set of best practices to guide the development and deployment of portal applications in Federal Student Aid's enterprise portal architecture. The key enablers of the portal architecture are the development, user interface, integration, and security components. The Federal Student Aid Architectural Model is shown below in Figure 3-1. The focus of this document is the Web Tier – Portal section which has been enlarged in the model below.

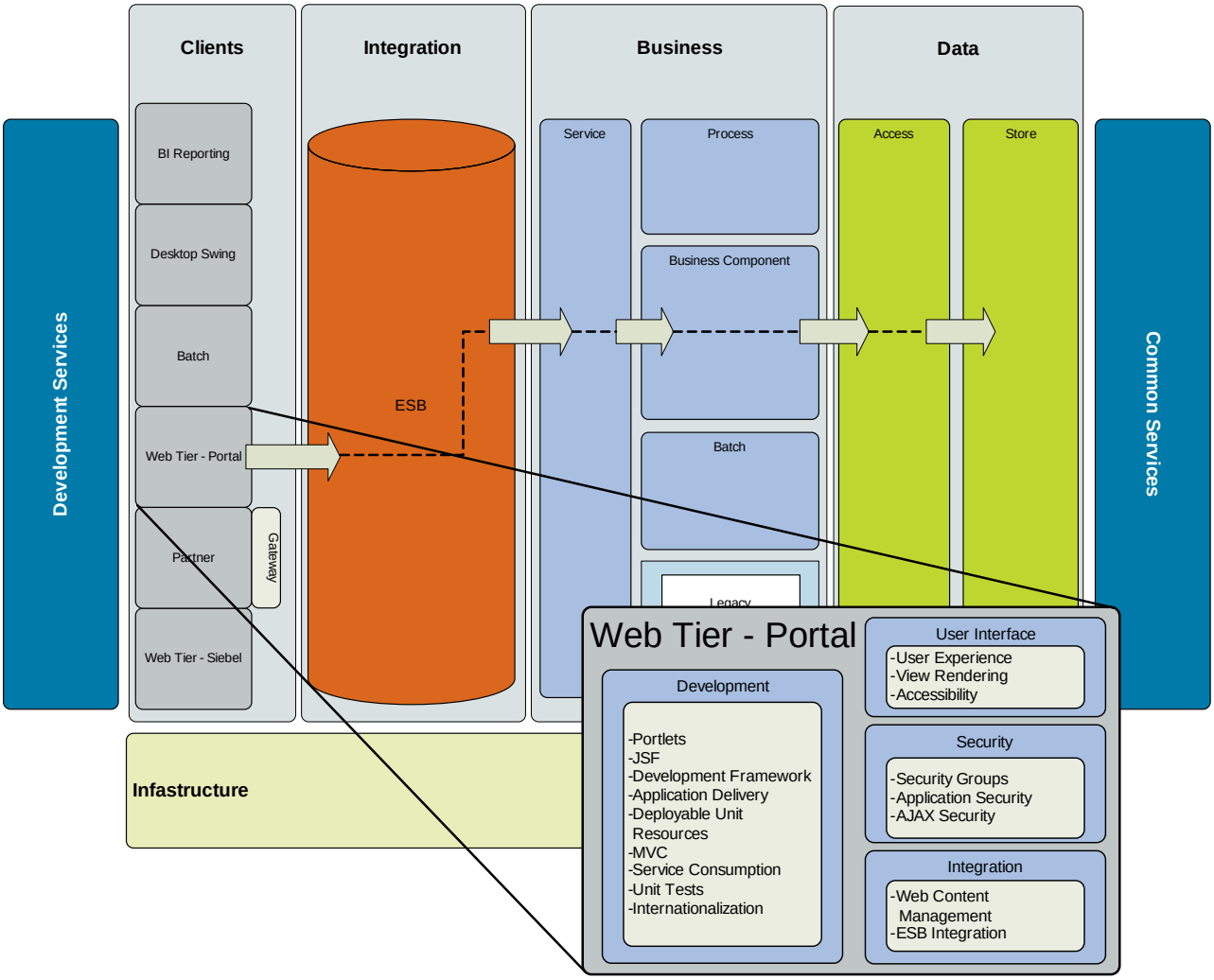


**Figure 3-1: Federal Student Aid Architectural Model (static view)**

The remaining subsections present a static view of the model that identifies technologies and recommendations for building a portal application. Section 6 presents a dynamic view of the portal model through the illustration of various use cases that would apply to the portal model.

## 3.1 Web Tier - Portal

The portal tier can be broken down into four major components as shown in Figure 3-1 above:

- **Development:** The development component describes the best practices that should be followed during general portal application development.
- **User Interface:** The user interface component describes the best practices that should be followed when developing the user interface of a portal application.
- **Integration:** The integration component outlines the best practices for web content management and ESB integration.
- **Security:** The security component identifies the best practices relating to personalization management, security groups, Asynchronous JavaScript and XML (AJAX), and Web 2.0 security considerations. For additional information on security refer to Federal Student Aid Security Architecture Model document.

In the following subsections, each of the above mentioned component models are broken down into a set of best practices. Each best practice contains five key pieces of information:

- **Best Practice:** One sentence description of Federal Student Aid's best practice for developing and implementing a particular aspect of a portal application.
- **Description:** A more detailed description of the best practice and the implications of the best practice.
- **Rationale:** A short description of the industry's and Federal Student Aid's rationale for adopting the best practice. Each rationale highlights one or more key benefits of the best practice in a table such as the one shown in Table 3-1.

| Key Benefit | Architectural Fit |
|---|---|
| Industry Standards Compliance | Leveraging external components can be cost-effective. Ensures that they will work properly in the Federal Student Aid environment |

**Table 3-1 Key Benefits Example**

- **Adoption Status:** Federal Student Aid has chosen an adoption framework to inform developers about Federal Student Aid's level of experience and expertise with working with particular technologies. For further discussion about Federal Student Aid's adoption framework please see sub-section 3.3.
- **Related Standards and Technologies**: A list of any standards and technologies that relate to the best practice.

### 3.1.1 Development Component

This sub-section describes the best practices that should be followed during general portal application development. These best practices will discuss the various components of Portlet Development, Unit Testing and Internationalization.

#### *3.1.1.1 Third Party Portlet Usage*

- **Best Practice:** Third-Party (Open Source or Commercial) portlets conform to Java Specification Request (JSR) 168 Java Portlet Specification

- **Description:** JSR-168 (Portlet API 1.0) is a Java Community Process (JCP) developed specification for the creation of java portlets and is the industry standard for Java portlet development. The specification includes an API, and definitions of portlet lifecycles, portlet display modes, and integration with other Java 2 Enterprise Edition (J2EE) web technologies (Servlet / Java Server Page [JSP]).

  One of the advantages of portal development is that in many cases there are open source or commercial portlets available that satisfy requirements for some aspects of a portal's implementation. Implementers are encouraged to evaluate these portlets prior to implementing their own.

  If open source or commercial portlets are leveraged, they must conform to the JSR-168 specification.

- **Rationale:** JSR-168 is the industry standard for Java portal development.

| Key Benefit | Architectural Fit |
|---|---|
| Industry Standards Compliance | Leveraging external components can be cost-effective. Ensures that they will work properly in the Federal Student Aid environment |

**Table 3-2: Portlet Development Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: JSR-168

#### *3.1.1.2 WebSphere Portlet Factory*

- **Best Practice:** Custom portlet implementations are created with IBM WebSphere Portlet Factory (WPF).

- **Description:** WPF is a Rapid Application Development (RAD) environment that supports WebSphere Portal Server and the WebSphere stack. WPF provides standard mechanisms for creating portal views, leveraging WebSphere Portal features and integration with enterprise services via Graphical User Interface (GUI) builders.

  As WPF generally operates at a higher level of abstraction than hand coding portlets, some compromises exist. For instance, WPF currently only supports JSP and Hyper Text Markup Language (HTML) views. The internals of a portlet implementation are entirely created by the tool. FSA expects that upcoming releases of WPF will support newer standards (e.g., Java Server Faces [JSF] views).

- **Rationale:** WPF is part of the Federal Student Aid development toolset. In most situations WPF can increase productivity by providing a standard development model and the plumbing code required to implement portlets in the Federal Student Aid SOA environment.

| Key Benefit | Architectural Fit |
|---|---|
| Federal Student Aid Standards Compliance | Leveraging the Federal Student Aid toolset can increase productivity and facilitates reuse of assets and skills across the organization |

**Table 3-3: Portlet Development Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid.

- **Related Standards and Technologies**: JSR-168, WebSphere Portal, WebSphere Portlet Factory

#### *3.1.1.3 Federal Student Aid Default Portlet implementation*

- **Best Practice:** Custom portlet implementations extend the Federal Student Aid GenericPortlet implementation.

- **Description:** Portlets developed as part of Federal Student Aid portal implementations must extend the GenericPortlet implementation provided by the Federal Student Aid core installable unit. This only applies to portlets developed without WebSphere Portlet Factory.

- **Rationale:** The abstract GenericPortlet implementation provided by the core installable unit implements standard patterns for leveraging JSR-168 features as well as IBM WebSphere Portal extensions and enhancements. Use of this portlet superclass promotes consistency across Federal Student Aid portal implementations.

| Key Benefit | Architectural Fit |
|---|---|
| Federal Student Aid Standards Compliance | Leveraging the Federal Student Aid toolset can increase productivity and facilitates reuse of assets and skills across the organization |

**Table 3-4: Portlet Development Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: JSR-168, WebSphere Portal

#### *3.1.1.4 Java Server Faces*

- **Best Practice:** Custom portlet implementations use Java Server Faces (JSF) views.

- **Description:** JSF is the Federal Student Aid standard view implementation. This practice only applies to portlets developed without the aid of WebSphere Portlet Factory

- **Rationale:** JSF is a component-based view technology that is the successor to the Java Server Page technology. JSF enjoys support from the tooling provided by the Federal Student Aid development suite

| Key Benefit | Architectural Fit |
|---|---|
| Industry Standards Compliance | JSF is a J2EE standard and fully supported by the Federal Student Aid Toolset |
| Federal Student Aid Standards Compliance | Leveraging the Federal Student Aid toolset can increase productivity and facilitates reuse of assets and skills across the organization |

**Table 3-5: Portlet Development Framework Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: JSF.

#### *3.1.1.5 Portlet Component Frameworks*

- **Best Practice:** JSF components use the Apache Trinidad JSF component framework.

- **Description:** Federal Student Aid prefers that JSF components use the Apache Trinidad JSF component framework. Trinidad provides a rich set of components as well as AJAX support. Implementers should consult the Trinidad component catalog prior to implementing custom components. If custom components are still required, FSA expects implementers to extend Trinidad components, to the extent possible.

- **Rationale:** Component frameworks provide libraries of User Interface (UI) components that relieve the implementer of the need to develop common UI elements. Additionally, in many cases, a pre-existing component can be extended to provide required functionality not present in the library.

| Key Benefit | Architectural Fit |
|---|---|
| Federal Student Aid Standards Compliance | Leveraging the Federal Student Aid toolset can increase productivity and facilitates reuse of assets and skills across the organization |

**Table 3-6: Installable Units Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: Portlet API, Java Servlet API, Apache Trinidad

#### *3.1.1.6 Application Delivery*

- **Best Practice:** Portal applications leverage the Federal Student Aid Build and Deployment Framework (BDF) to develop and deliver applications.

- **Description:** The Federal Student Aid's BDF provides standard build scripts to support the development of portal applications. Federal Student Aid will build a core deployable unit using the BDF framework.

  BDF provides the following features:

  - Build of portal deployable units
  - Deployment of portal deployable units
  - Environment management (environment configuration repository)
  - Developer desktop configuration (e.g., portal server install, fixpacks)
  - Build scheduling and continuous builds
  - Logging and reporting
  - Repository of reusable components or units

- **Rationale:** Use of the framework makes certain that delivered applications are compatible with the Federal Student Aid build and deployment infrastructure. New portal applications may reuse and customize the installable unit build scripts thereby reducing time to go live.

| Key Benefit | Architectural Fit |
|---|---|
| Federal Student Aid Standards Compliance | Leveraging the Federal Student Aid toolset can increase productivity and facilitates reuse of assets and skills across the organization |

**Table 3-7: Installable Core Unit Resources Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: BDF, deployable units.

#### *3.1.1.7 Core Deployable Unit Resources*

- **Best Practice:** Portal applications leverage the resources provided in the Federal Student Aid Core Deployable Unit

- **Description:** The Federal Student Aid Core Deployable Unit provides a standard set of portlets (e.g., News and Highlights, Notification, Contact Us), UI Resources (e.g., themes, skins, images, style sheets) and shared libraries (e.g., helper classes) that leverage Federal Student Aid portal applications. Implementers are encouraged to leverage components and services in the Core Deployable Unit prior to developing their own.

- **Rationale:** The Core Deployable Unit provides a base set of services for reuse across all portal applications, allowing portal developers to focus on functionality unique to a given portal implementation. The use of the themes and skins provided in the core deployable unit will provide standard look and feel for all the portal pages.

| Key Benefit | Architectural Fit |
|---|---|
| Federal Student Aid Standards Compliance | Leveraging the Federal Student Aid toolset can increase productivity and facilitates reuse of assets and skills across the organization |

**Table 3-8: WS-I Compliance Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: Portal API, JSR 168.

#### 3.1.1.8 Model View Controller (MVC) Implementation

- **Best Practice:** Portal applications leverage standard design patterns for access to services.

- **Description:** Portlet controller classes (JSR-168 GenericPortlet implementations or their Plain Old Java Object (POJO) analogues) should not implement business logic. Rather, they are expected to handle requests, invoke the appropriate services, and return the data that will be rendered by the view. This approach avoids the Fat Controller antipattern that has a tendency to develop in both portal and traditional web applications that poorly implement the MVC pattern.

  This can be addressed by using Remote Facade pattern. Remote Façade provides a coarse grained façade to a web of fine grained objects, allowing the controller to remain lightweight. The Façade could be accessed using SOAP, EJBs or messaging. In all cases, the design must clearly separate the façade's responsibilities from the portal or view implementation.

- **Rationale:** These patterns represent industry best practice for the development of Web applications.

| Key Benefit | Architectural Fit |
|---|---|
| Industry Best Practice Compliance | Proper use of design patterns ensures that implementations do not repeat common mistakes |

**Table 3-9: MVC Implementation Compliance Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: Patten application is a matter of applying heuristics and is technology and language independent. Standard references for patterns are Design Patterns (Gamma et al.), Core J2EE Design Patterns (Deepak et al.) and Patterns of Enterprise Application Architecture (Fowler). See: http://www.titu.jyu.fi/modpa/Patterns/pattern-RemoteFacade.html. http://www.titu.jyu.fi/modpa/Patterns/pattern-mvc.html

#### 3.1.1.9 Service Consumption

- **Best Practice:** Service interfaces are statically typed.

- **Description:** Federal Student Aid prefers that service interfaces be statically typed. Overloading primitive types (e.g. using a string parameter to pass an XML document) are to be avoided. Interface

parameters and return values are to be atomic, primitive types, or domain abstractions. The Dynamic Invocation Interface (DII) are not be used in implementations.

Static typing may be achieved using:

- Interfaces and stubs generated during compile time (e.g., EJB interfaces)
- Predefined static interfaces (e.g., POJO interfaces).
- XML Document Schemas (e.g., XML Schema Definition [XSD])

Services exposed by ESB, which work in a loosely coupled architecture, can be utilized by using interfaces to the Channel Adapter as described by the Channel Adapter integration design pattern Static typing gives only partial safety, therefore a test driven approach is strongly recommended.

- **Rationale:** Dynamic interfaces provide lot of flexibility, but are likely to create runtime issues. Statically typed interfaces support compile time checking and help catch a majority of the errors upfront. Design patterns such as the Channel Adapter could be used to achieve static typing in a loosely couple architecture.

| Key Benefit | Architectural Fit |
|---|---|
| Industry Best Practice Compliance | Proper use of design patterns ensures that implementations do not repeat common mistakes |

**Table 3-10: Service Consumption Compliance Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.
- **Related Standards and Technologies**: EJB, XML, XSD, DII, ESB. See: http://www.enterpriseintegrationpatterns.com/ChannelAdapter.html

#### *3.1.1.10 Unit Tests*

- **Best Practice:** Portal implementations include automated unit tests.
- **Description:** Implementers must furnish a suite of executable tests based on the base JUnit framework and JUnit extensions (e.g., HttpUnit). Tests must be executable as part of the build process provided by the Federal Student Aid BDF. Unit level (e.g., individual class) as well as integration level (e.g., UI level tests with HttpUnit) must be provided. The test suite must provide at least 90% code coverage of the delivered code and integration tests should cover all identified use cases.
- **Rationale:** Comprehensive, automated unit testing can provide an objective benchmark of code quality. Test suites that are utilized throughout the development lifecycle provide immediate feedback to a variety of stakeholders.

| Key Benefit | Architectural Fit |
|---|---|
| Industry Best Practice Compliance | Proper use of automated unit testing ensures implementation quality |

**Table 3-11: Unit Test Compliance Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: JUnit, Agile Methods

#### *3.1.1.11 Internationalization*

- **Best Practice:** Portal applications that require multilingual support are i18n enabled.

- **Description:** Federal Student Aid Portal applications must implement support for internationalization (i18n). I18n support involves coding practices and the use of various Java Software Developers Kit (SDK) features, as opposed to the creation of an alternate website or portal in the target language. The following activities should be considered:

    - Identify Culturally Dependent Data

        - Text - Messages, UI labels and button text, online help, etc.

        - Non-Text UI Elements - Colors, graphics, sound and icons may require localization

        - Measures and Quantities - Dates, times, numbers, measurements, etc.

        - Address Data - Addresses, phone numbers, etc.

        - Personal Data - Titles and honorifics

    - Store text in locale specific Java Resource Bundles

    - Use the Java locale support to format dates, times, and numbers

    - Use Java locale support for string comparison and character classification

    - Convert strings processed by the system to Unicode format

- **Rationale:** Federal Student Aid portal applications target various user communities. The Java SDK provides support for i18n, but these practices should be followed to effectively leverage this support.

| Key Benefit | Architectural Fit |
|---|---|
| Industry Best Practice Compliance | Proper use of internationalization can simplify multilingual application implementation |

**Table 3-12: Internationalization Compliance Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: Java Software Development Kit

### 3.1.2 User Interface Component

- This sub-section describes the best practices that should be followed when developing the user interface of a portal application. These best practices cover User Experience, View Rendering and Accessibility.

#### *3.1.2.1 Enhanced User Experience*

- **Best Practice:** Portal applications leverage Web 2.0 and AJAX Federal Student Aid standard frameworks to enhance the user experience.

- **Description:** AJAX is a standard mechanism underlying the development of rich web applications. Implementers will leverage AJAX when it is determined that a rich user interface will enhance the portlet user experience. The dojo AJAX toolkit will be used in AJAX implementations.

  If a custom portlet cannot be developed efficiently with WPF and its AJAX support, AJAX development in JSR-168 containers must utilize rendering servlets to address the lack of support for asynchronous calls. (This issue will be fixed in JSR-286, Portlet API 2.0.)

- **Rationale:** End users are becoming accustomed to interactive web applications. Judicious use of AJAX can enrich the user experience, particularly in dashboard-style applications. A common framework facilitates reuse of talent and resources across the organization

| Key Benefit | Architectural Fit |
|---|---|
| Industry Best Practice Compliance | Proper use of AJAX can enrich the user experience |

**Table 3-13: Enhanced User Experience Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: AJAX

#### *3.1.2.2 View Rendering*

- **Best Practice:** Custom portlets separate view rendering from processing.

- **Description:** In accordance with basic MVC design pattern principles, portlet implementations must neither generate view markup in the controller or model, nor implement business logic in the view. Even more so than traditional Web applications, portlets may be subject to skinning (e.g., the application of new HTML theme) or be required to support radically different view technologies (e.g., Wireless Markup Language (WML)). Lightweight view templates (e.g., JSP's) ensure that portlets do not impose undue limitations upon cosmetic changes to existing views or the application of new view technology.

- **Rationale:** In order for Federal Student Aid to fully exploit the advantages of the portal development model, implementers must adhere to view development standards.

| Key Benefit | Architectural Fit |
|---|---|
| Industry Best Practice Compliance | Proper use of design patterns ensures that implementations do not repeat common mistakes |

**Table 3-14: View Rendering Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.
- **Related Standards and Technologies:** MVC Design Pattern. See: http://www.titu.jyu.fi/modpa/Patterns/pattern-mvc.html

#### *3.1.2.3 Portlet UI*

- **Best Practice:** Portal applications provide a path to Section 508(C) compliant accessibility.
- **Description:** Web based applications must adhere to Section 508(C) accessibility standards. However, Federal Student Aid also encourages vendors to develop rich user interfaces where appropriate. To address these conflicting goals, Federal Student Aid portal implementers should implement 508(C) compliant paths through a portal alongside the default rich UI implementation. This approach permits the delivery of a 508(C) compliant application without sacrificing rich UI benefits.
- **Rationale:** 508(C) compliance is required for applications. Federal Student Aid desires Rich UI applications.

| Key Benefit | Architectural Fit |
|---|---|
| Industry Standard Compliance | 508(c) compliance is mandatory for applications |
| Federal Student Aid Standards Compliance | Dual path approach ensures that Federal Student Aid portal users may benefit from rich UI implementations |

**Table 3-15: Portlet UI Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.
- **Related Standards and Technologies**: AJAX, Section 508(C)

### 3.1.3 Integration Component

This sub-section outlines the best practices for web content management, ESB integration, and search appliances.

#### *3.1.3.1 Web Content Management*

- **Best Practice:** Portal applications use IBM Workplace Web Content Management for content management

- **Description:** A web content management system is a Content Management System (CMS) for web site or portal management. Web content management systems are often used for storing, controlling, versioning, and publishing industry-specific documentation such as news articles, operator manuals, technical manuals, sales guides, and marketing brochures. IBM Workplace Web Content Management software offers end-to-end web content management. Content can be created (using a What You See Is What You Get (WYSIWYG) rich text editor), managed, and subsequently published to multiple Web sites. The following bullets present a list of web content management features:
    - End-to-end Web content management so content can be created (using WYSIWYG rich text editor), managed, and subsequently published to multiple Web sites.
    - A collaborative environment that makes it easy for users of all levels to work together, and complete review and approval of workflow processes.
    - Content creation through a WYSIWYG rich text editor or by importing text from another application (such as a word processor).
    - Built-in content lifecycle workflow that provides review and approval capabilities, and automatic expiration of old content.
    - Versioning and rollback to previous content versions.
    - Personalization that delivers content relevant to the users and their roles.
    - Better-managed business content that is immediately available in the portal. Users create, manage, and publish information straight into the portal for high availability of accurate, relevant, and up-to-date information across all organizational boundaries.
- **Rationale:**
    - Streamlines the creation, publication, and lifecycle management of content for internal and external Web sites.
    - Publishes information on demand in minutes, not days, for improved responsiveness to customers, partners, suppliers, and employees.
    - Personalizes Web sites to deliver targeted information and services based on a user's profile or actions.
    - Ensures consistent, professional look-and-feel across multiple sites.

| Key Benefit | Architectural Fit |
|---|---|
| Federal Student Aid Standards Compliance | Leveraging the Federal Student Aid toolset can increase productivity and facilitates reuse of assets and skills across the organization |

**Table 3-16: Web Content Management Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.
- **Related Standards and Technologies**: IBM Workplace Web Content Management

#### 3.1.3.2 ESB Integration

- **Best Practice:** Portal applications leverage the ESB to invoke enterprise services.

- **Description:** Federal Student Aid prefers that Portal applications use WebSphere ESB and WebSphere DataPower Integration Appliance XI50 for portal backend services communication. ESB provides the transport mechanism that developers can use to publish their WSDLs, rather than to the outside world. The usage of ESB does not increase development efforts.

- **Rationale:** The use of ESB by portal applications for backend communication allows Federal Student Aid to track usage and report on reliability of services without burdening the developers.

| Key Benefit | Architectural Fit |
|---|---|
| Federal Student Aid Standards Compliance | Leveraging the Federal Student Aid toolset can increase productivity and facilitates reuse of assets and skills across the organization |
| Industry Standards Compliance | Leveraging the ESB via industry standard interfaces improves consistency of SOA architecture |

**Table 3-17: ESB Integration Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: ESB, Portlet API, DataPower X150

#### 3.1.3.3 Search Appliance

- **Best Practice:** Portal applications use Federal Student Aid's Google Search infrastructure.

- **Description:** Federal Student Aid prefers that Portal applications use the Federal Student Aid Google Search infrastructure.

- **Rationale:** Federal Student Aid has selected Google Search to provide search services across the organization. This practice ensures consistency of search implementations across Federal Student Aid.

| Key Benefit | Architectural Fit |
|---|---|
| Federal Student Aid Standards Compliance | Leveraging the Federal Student Aid toolset can increase productivity and facilitates reuse of assets and skills across the organization |

**Table 3-18: Google Search Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: Google Search

### 3.1.4 Security

The Security component identifies the best practices relating to security groups, application security, and AJAX and Web 2.0 specific security considerations. Security issues outside these components will be address in the Security Model Document.

#### *3.1.4.1 Security Groups*

- **Best Practice:** Portal applications leverage enterprise and site level groups prior to creating custom security groups.

- **Description:** Federal Student Aid enterprise and site directories established by the Enterprise Portal Group contain user groups. These enterprise groups may adequately describe user groups for a particular portal implementation. Analyze these directories' group structures for suitability prior to implementing custom groups for an individual portal effort.

- **Rationale:** This practice will mitigate the proliferation of redundant user groups within Federal Student Aid.

| Key Benefit | Architectural Fit |
|---|---|
| Federal Student Aid Standards Compliance | Leveraging the Federal Student Aid organizational structure simplifies group management |

**Table 3-19: Security Groups Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies:** LDAP

#### *3.1.4.2 Application Security*

- **Best Practice:** Portal applications and portlets are developed to mitigate the risk of common web application security vulnerabilities

  **Description:** Web applications are susceptible to a variety of security vulnerabilities if secure development practices are not followed. The following are some of the most common security vulnerabilities:

  - Cross Site Scripting (XSS) – Applications send user supplied data back to a browser without validation or encoding.

  - Injection Flaws – User supplied data is directly executed as a command or part of a command by the web application (i.e. Structured Query Language [SQL] injection).

  - Malicious File Execution – User uploaded files are executed directly by the web application.

  - Insecure Direct Object Reference – Direct references to application objects are exposed directly to the user (e.g., exposing a database record id in a Uniform Resource Locator [URL]).

- Cross Site Request Forgery (CSRF) – Forces logged on victim's browser to send a request to a vulnerable website that then performs an action on behalf of the victim.
- Information Leakage and Improper Error Handling – Unintentionally revealing configuration or application data (e.g., displaying stack traces to the end user).
- Broken Authentication and Session Management – Failing to protect credentials and session tokens throughout their life cycle.
- Insecure Cryptographic Storage – Failing to encrypt sensitive data or employing weak encryption schemes.
- Insecure Communications – Not using Secure Sockets Layer (SSL) for authenticated communication, or communications that transmit sensitive data.
- Failure to Restrict URL Access – Allowing direct, unauthenticated access to any sensitive URL.

- **Rationale:** Federal Student Aid portal development vendors shall implement security measures to mitigate known web application exploits

| Key Benefit | Architectural Fit |
|---|---|
| Security | Mitigates the possibility of security exploits |

**Table 3-20: Application Security Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.
- **Related Standards and Technologies**: Open Web Application Security Project (OWASP) Top Ten

#### *3.1.4.3 AJAX Security*

- **Best Practice:** Ajax (Web 2.0) Portlets decline malicious requests and prevent direct execution of JavaScript responses to defeat JavaScript Hijacking attacks.
- **Description:** The advent of the AJAX development model permits the creation of a new class of rich Web-based applications. AJAX is built upon the ability to send Hyper Text Transfer Protocol (HTTP) requests from the browser without refreshing an entire web page. These requests are typically comprised of data in XML or JavaScript Object Notation (JSON) formats. JSON allows one to serialize JavaScript objects and arrays that are dynamically evaluated by parsers on the client or server. In the case of server to client communication, it is possible to construct an attack that allows a malicious web site to read the contents of a JSON response to the client from the original website.

  This vulnerability requires the following:

  - A user is tricked into accessing a malicious website via a spoofing attack
  - The original website transfers JSON objects and/or arrays without implementing security measures

The user accessing the spoofed website can trigger the transmission of sensitive data from the original website to the malicious website.

In order to prevent these attacks an AJAX enabled application must do the following:

- Decline malicious requests - The application must ensure that information that only the server and the client application share is kept private during every AJAX request and response. In the J2EE domain, this is implemented with the Java Servlet API's session ID. Only the original server and the client share knowledge of the session ID. Requests from a page served by the malicious site would contain a session ID unknown to the server, allowing it to deny the request

- Prevent direct execution of the JavaScript response in the browser - This defense requires that the server send invalid or unexecutable JavaScript in AJAX responses to the client. For example, commenting out the entire response. In this case, only a page served by the trusted server knows how to fix up the JavaScript received by the server (e.g., removing the comments before parsing).

The portal implementation vendor must implement both of these techniques to minimize the risk of this exploit. Currently, most AJAX toolkits are beginning to implement native support for these techniques. However, it is the responsibility of the implementer to validate that a delivered solution properly implements these defenses.

- **Rationale:** JavaScript Hijacking is an AJAX specific variation of the Cross Site Scripting Attack. All Web applications must implement security measures to mitigate known Web application exploits

| Key Benefit | Architectural Fit |
|---|---|
| Security | Mitigates the possibility of AJAX security exploits |

**Table 3-21: AJAX Security Key Benefits**

- **Adoption Status:** Emerging - No institutional experience at Federal Student Aid. This is a mature industry standard.

- **Related Standards and Technologies**: AJAX, JSON

## 3.2 Development Tools

This section presents, in Table 3-22 below, the tools that should be leveraged when developing and deploying the technology components that are part of the portal architecture.

| Key Development Tool | Technology | Notes |
|---|---|---|
| Apache Trinidad JSF component framework | Portal Application Development | Provides a rich set of components as well as AJAX support. |

| Key Development Tool | Technology | Notes |
|---|---|---|
| IBM Workplace Web Content Management | Portal Application Integration | Provides end-to-end Web content management. |
| Rational Application Developer | Portal Application Development | Helps developers rapidly design, develop, assemble, test, profile and deploy Portal, Web, Web services and SOA applications. |
| WebSphere DataPower Integration Appliance XI50 | Portal Application Integration | Provides transport-independent transformations between binary, flat text files and XML message formats. |
| WebSphere ESB | Portal Application Integration | Provides Web services connectivity and JMS messaging. |
| WebSphere Portal Server | JSR-168 Portlet support plus WebSphere extensions | Provides support for standard portlet implementations as well as a variety of required services not covered by the portlet specification. |
| WebSphere Portlet Factory | Portal Development | RAD environment for Portlet development. |

**Table 3-22: Key Development Tools**

## 3.3 Adoption

Federal Student Aid has developed an adoption framework to help describe standard technologies, as well as emerging ones, that are under consideration for future use. These technologies are listed in Table 3-23 below along with their associated adoption status:

- **Emerging**- No institutional experience at Federal Student Aid
- **Candidate**- No production experience but some experimental experience at Federal Student Aid
- **Growing**- Limited production experience at Federal Student Aid
- **Mature**- Proven production experience at Federal Student Aid

| Technology | Adoption Status | Explanation |
|---|---|---|
| EJB | Emerging | No institutional experience at Federal Student Aid. This is a mature industry technology. |

| Technology | Adoption Status | Explanation |
|---|---|---|
| SOAP | Growing | No institutional experience at Federal Student Aid. This is a mature industry standard. |
| ESB | Candidate | No production experience, but some experimental experience at Federal Student Aid. This is a mature industry standard |
| JSF | Candidate | No production experience, but some experimental experience at Federal Student Aid. This is a mature industry standard |
| WebSphere Portal Server | Candidate | No production experience, but some experimental experience at Federal Student Aid. This is a mature industry standard |

**Table 3-23: Adoption Examples**

## 3.4 Constraints

Federal Student Aid has standardized several key technologies required to support Federal Student Aid's next generation portal architecture. The following standards are core foundation technologies to this architecture:

- WebSphere Application Server (6.x +)
- WebSphere Process Server (6.x +)
- WebSphere Portal Server (6.x +)
- Data Power X150 ESB Appliance

The conceptual architecture assumes that these base technologies are available for use at Federal Student Aid and focuses on implementation options and patterns. Unavailability of the above technologies will affect some of the best practices that are suggested in the architectural model.

# Section 4: Architectural Decisions

This section presents key architectural decisions that were made as part of the Portal architecture definition efforts. These decisions were derived from numerous discussions among key stakeholders. These decisions reflect current thinking of Federal Student Aid's for defining Portal architecture.

| Architecture Decision | | | |
|---|---|---|---|
| 1. Presentation tier does not contain business logic | | | |
| With the exception of simple client-side validation, business logic should be encapsulated in the services exposed in the business tier. | | | |
| Architectural Area: | Web Tier - Portal | Implications: | Applications should implement business logic in the service layer |
| Rationale: | Industry best practice holds that failure to properly separate concerns in tiered applications leads to implementations that are difficult to maintain. | Decision Data: | MVC frameworks, patterns and best practices coupled with Service and Component oriented architectures obviate the need to code business logic in the presentation tier. |

| Architecture Decision | | | |
|---|---|---|---|
| 2. Use of supplemental MVC frameworks must be approved by Federal Student Aid | | | |
| JSF provides a component-based MVC framework for Java web application development. However, a few development scenarios exist that may be addressed by supplemental JSF frameworks such as Seam and Shale. Implementer should receive approval from Federal Student Aid before employing these frameworks. | | | |
| Architectural Area: | Web Tier - Portal | Implications: | Applications should use JSF only in the vast majority of cases. |
| Rationale: | JSF, in combination with the portal framework provides most of the functionality required by portal applications. | Decision Data: | JSF and the Java portal frameworks satisfy the "80-20 rule" for portal application development. |

# Section 5: Glossary and Standards

This section presents and defines key standards and technologies that are part of Federal Student Aid's Portal architecture model. The purpose of this section is to clearly reflect standards and technologies chosen for inclusion in the model. Readers should reference Appendix A for a complete list of acronyms used in this document.

The portal architecture model is based on key concepts, standards and technologies that are part of the best practices for building modern business portal applications. However, some of these concepts, standards, and technologies are still evolving; therefore, they are presented here to avoid confusion and ambiguity on the interpretation of these concepts, standards, and technologies and their use.

Table 5-1 below presents a glossary of technologies and standards employed in this document.

| Term | Definition |
|---|---|
| Agile Methods | Agile processes satisfy the customer through early and continuous delivery of valuable software, while supporting changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage and promote sustainable development. |
| COTS | Commercial off-the-shelf: A term for software or hardware products that are ready-made and available for sale, lease, or license to the general public. They are often used as alternatives to in-house developments or one-off government-funded developments. |
| CRM | Customer relationship management software |
| DAO | Data Access Object: A software component that provides a common interface between the application and one or more data storage devices, such as a database or file. Also, frequently used to refer to the Data Access Object design pattern. |
| DataPower X150 Integration Appliance | A 1U (1.75" thick) rack-mountable network device capable of transforming between disparate message formats, including binary, legacy, and XML, and providing message routing and security. XI50 can be used for cost-effective XML enablement of mainframes, wirespeed enterprise message buses, and enterprise application integration. |
| EJB | Enterprise Java Beans: A server-side component architecture used to develop transaction-based distributed systems on the J2EE platform |
| ESB | Enterprise Service Bus: An ESB is an integration architecture implemented by technologies found in a category of middleware infrastructure products usually based on web-services standards. The ESB technology provides foundational services for a service-oriented architecture (SOA) via an event-driven and XML-based messaging engine ("the bus"). |
| JSON | JSON (JavaScript Object Notation) is a lightweight data-interchange format that is easy for humans to read and write, and is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, |

| Term | Definition |
|---|---|
| | Python, and many others. These properties make JSON an ideal data-interchange language. |
| JSR -168 | The Java Portlet Specification defines a contract between the portlet container and portlets and provides a convenient programming model for portlet developers. The Java Portlet Specification V1.0 was developed under the Java Community Process as JSR 168 |
| JUnit | JUnit is an open source regression testing framework that is used to write repeatable unit tests in java. It is an instance of the xUnit architecture for unit testing frameworks. |
| LDAP | Lightweight Directory Access Protocol: An application protocol for querying and modifying directory services running over TCP/IP. |
| Legacy | Describes a system that is in current production use at the time a new development effort is undertaken |
| Model View Controller (MVC) Pattern | A standard design pattern for the implementation of user interface interaction. The Model contains all data and behavior other than that used for the User Interface. The View represents the display of the model. The Controller accepts user input, invokes the Model, and passes the results to the View for display to the User. |
| OLTP | Online Transaction Processing: A class of programs that facilitate and manage transaction-oriented applications, typically for data entry and retrieval transaction processing. |
| ORM | Object-Relational Mapping: A programming technique for converting data between incompatible type systems in databases and Object-oriented programming languages. |
| OWASP Top Ten | The OWASP Top Ten provides a powerful awareness document for web application security. The OWASP Top Ten represents a broad consensus about what the most critical web application security flaws are. The U.S. Defense Information Systems Agency has listed the OWASP Top Ten as key best practices that should be used as part of the DOD Information Technology Security Certification and Accreditation (C&A) Process |
| Portlet | Portlets are pluggable user interface components that are managed and displayed in a web portal. Portlets produce fragments of markup code that are aggregated into a portal page. Typically, following the desktop metaphor, a portal page is displayed as a collection of non-overlapping portlet windows, where each portlet window displays a portlet. Hence a portlet (or collection of portlets) resembles a web-based application that is hosted in a portal. Some examples of portlet applications include email, weather reports, discussion forums, and news. |
| POJO | Plain Old Java Object: A Java object which does not follow a specific object model, convention or framework and which, absent the use of any cross-platform integration technologies, can only be utilized by Java-based systems |
| Portal | A Web portal is a single point of access to information that is linked to various logically related internet-based applications and of interest to various types of users.<br><br>Portals present information from diverse sources in a unified way. They provide an excellent way for enterprises to provide a consistent look and feel with access control and procedures for multiple applications, which otherwise would have been different entities altogether. |
| Query | A database query, the standard way information is extracted from databases. |
| RDBMS | Relational Database Management System: A database management system in which data is stored in tables and the relationship among the data is also stored in tables. |
| Section 508(C) | Section 508 (C) requires the federal government to make all goods and services including Web pages fully accessible. It identifies specific standards for Internet and Web accessibility, which are often used as a basis for evaluating whether or not Web sites meet accessibility requirements. |
| SCA | Service Component Architecture: A set of specifications that describe a model for |

| Term | Definition |
|---|---|
| | building applications and systems using a Service-Oriented Architecture. SCA extends and complements prior approaches to implementing services, and SCA builds on open standards such as Web services. |
| Siebel | Siebel is principally engaged in the design, development, marketing and support of CRM applications. |
| SOA | Service-Oriented Architecture: SOA is an architectural style. Applications built using an SOA style deliver coarse grained functionality as services that can be shared when building applications or when integrating within the enterprise or with trading partners. |
| SOAP | A technique for serializing object values to XML and reconstructing them as objects. SOAP handles the round trip between object and XML. May also be referred to as Service-Oriented Architecture Protocol |
| SQL | Structured Query Language: A fourth-generation (4GL) programming language used specifically to query a relational database and perform basic record operations such as reads, inserts, updates and deletes |
| Swing | Swing is a graphical user interface (GUI) toolkit for Java. It is one part of the Java Foundation Classes (JFC). Swing includes GUI objects such as text boxes, buttons, split-panes, and tables. |
| WAS | IBM WebSphere Application Server: A software application server. WAS is built using open standards such as J2EE, XML, and Web Services. |
| Web Service | A software system designed to support interoperable Machine to Machine interaction over a network. Web services are frequently just Web APIs that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. |
| WebSphere Portal Server | IBM's portal software which runs on top of WAS. |
| Workflow | A description of the movement of information through a work process |
| WS-BPEL | Web Services Business Process Execution Language: An XML-based programming language used to describe a business process |
| WSDL | Web Services Definition Language: An XML format used to define the external interface to a web service as a series of endpoints |
| XML | Extensible Markup Language: A platform-neutral markup language used to describe structured information in document form |

**Table 5-1: Glossary**

# Section 6: Use Case

This section presents the portal architectural model in action through a dynamic representation of the portal architecture. This is accomplished by depicting a use case and sequence diagram.

## 6.1 Overview

This subsection provides an overview of use case example applied to the portal architecture model. The following use case will be used to describe this architecture and its components.

**Financial Data View:** The financial data view use case describes the steps involved to retrieve the financial data from a database when a student wants to view his/her current loan data. This is a key service that Federal Student Aid provides to its customers.

## 6.2 Financial Data View

The service consumer transmits two key pieces of information when a student wants to view his/her financial data:

1. Borrower ID
2. Aid ID

The Borrower and Aid ID are queried against the person database to return the person and loan information. The sequence diagram for this use case is described below.

### Use Case and Sequence Diagram Description

The Service Consumer initiates the process by accessing the portal, which authenticates the person. The Portal presents a portal view page on which the person enters the Borrower ID and Aid ID into Financial Data Portlet. The request on the portlet initiates a call to the model. The model accesses the backend data through ESB, exposed via Web Services, while using Remote Façade pattern.

The following describes the interchange for the financial data view. Process steps affecting each of the concerned entities – Person, Portal, Portlet, and ESB – are provided:

Step 1: Portal authorizes the student and displays request page

- The student tries to access the portal page for financial data.

- The person is authenticated by an authentication mechanism (see Security Architecture model) and authorized by the portal.
- Portal presents financial data request page to the student.

Step 2: Student submits Person and Loan data to the Portal

- The student enters the Person and Loan data into the financial data request page.
- Portal sends the request to the Financial Data Portlet.
- Financial Data portlet creates and instance of Financial Data portlet Model (see MVC pattern) and sends request to get the data.
- Financial Data portlet Model gets a reference to the Financial Data web service.

Step 3: Get data from the database

- Financial Data portlet Model uses the Web service reference to send data request onto DataPower ESB.
- DataPower X150 ESB performs security checks and sends the message onto WebSphere ESB.
- Financial Data web service, that is implemented using Remote Façade pattern, picks up the Web Service request from the WebSphere ESB.
- Remote Façade uses collection of fine grained objects to get coarse grained view of data.
- Financial data web service sends the data back.

Step 4: Portal sends a render request

- Financial data portlet gets a render request from the portal.
- The portlet uses the model and redirects to a view that uses JSF.

Figure 6-1 illustrates the use case described above through the use a sequence diagram.

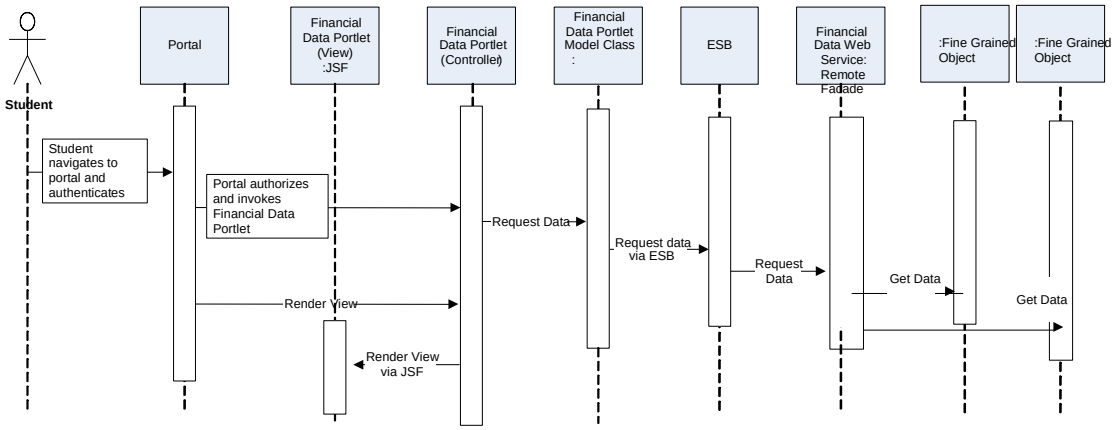


**Figure 6-1: Financial Data View Sequence Diagram**

# Appendix A: Acronyms

| Acronym | Definition |
|---|---|
| AJAX | Asynchronous JavaScript and XML |
| API | Application Programmatic Interfaces |
| ASN.1 | Abstract Syntax Notation One |
| BDF | Build and Deployment Framework |
| BI | Business Intelligence |
| BPEL | Business Process Execution Language |
| CIO | Chief Information Officer |
| CMS | Content Management System |
| COBOL | Common Business Oriented Language |
| COTS | Commercial Off the Shelf |
| CRM | Customer Relationship Management |
| CSRF | Cross Site Request Forgery |
| CSV | Comma Separated Value |
| DAO | Data Access Object |
| DBMS | Database Management System |
| DII | Dynamic Invocation Interface |
| EDI | Electronic Data Interchange |
| EJB | Enterprise Java Bean |
| ESB | Enterprise Service Bus |
| GUI | Graphical User Interface |
| HEA | Higher Education Act |
| HTML | Hyper Text Markup Language |
| HTTP | Hyper Text Transfer Protocol |
| IF | Information Framework |
| IP | Internet Protocol |
| IPM | Integrated Partner Management |
| ISO | International Organization for Standardization |
| IT | Information Technology |
| J2EE | Java 2 Enterprise Edition |
| JCP | Java Community Process |
| JFC | Java Foundation Classes |
| JSF | Java Server Faces |
| JSON | JavaScript Object Notation |

| Acronym | Definition |
|---|---|
| JSP | Java Server Page |
| JSR | Java Specification Request |
| LDAP | Lightweight Directory Access Protocol |
| MVC | Model View Controller |
| OLTP | Online Transaction Processing |
| ORM | Object Relational Mapping |
| OWASP | Open Web Application Security Project |
| PBO | Performance Based Organization |
| POJO | Plain Old Java Object |
| RAD | Rapid Application Development |
| RDBMS | Relational Database Management System |
| SAML | Security Assertion Markup Language |
| SCA | Service Component Architecture |
| SDK | Software Developer's Kit |
| SOA | Service-Oriented Architecture |
| SOAP | Simple Object Access Protocol |
| SQL | Structured Query Language |
| SSL | Sockets Layer |
| TCP | Terminal Connection Point |
| TSV | Target State Vision |
| UI | User Interface |
| URL | Uniform Resource Locator |
| WAS | WebSphere Application Server |
| WML | Wireless Markup Language |
| WPF | WebSphere Portlet Factory |
| WS-BPEL | Web Services Business Process Execution Language |
| WSDL | Web Services Definition Language |
| WSIL | Web Interface Specification Language |
| WYSIWYG | What You See Is What You Get |
| XML | Extended Markup Language |
| XSD | XML Schema Definition |
| XSS | Cross Site Scripting |

**Table A-1: Acronym List**