D_R_A_F_T

ERA J2EE Technology Overview

Version 0.1 (Complete Draft)

February 23, 2001

Contract No:
Task Order No:

Document ID:

Prepared For:

Office of Policy for Extramural Research Administration
Office of Extramural Research
National Institutes of Health
Bethesda, MD

Prepared By:

Logicon Federal Data/ROW Sciences and Silicon Spirit Consulting Group

Silicon Spirit Consulting Group, Inc.
1801 Robert Fulton Drive, Reston, VA 20191
www.siliconspirit.com   Phone: 703-453-6849

Logicon Federal Data/R.O.W. Sciences
1700 Research Blvd, Suite 400, Rockville, Maryland 20850-3142.

Table of Contents

List of Figures

1 Background

One of the primary objectives of the National Institutes of Health (NIH) Electronic Research Administration (ERA) initiative is to provide full electronic grants administration between biomedical research applicants/institutions and the NIH Institutes and Centers. A key component of this initiative is the NIH/ERA Commons System (Commons), whose goal is to provide an internet interface for applicants and institutions to participate in an electronic grant administration process. This process includes a coupling with the Federal Commons initiative, which is currently being coordinated by Paul Markovitz of NIH's Office of Extramural Research (OER).

Historically, the original Commons effort was initiated around the 1994/1995 timeframe, resulting in a current architecture and operational state that has encountered various technology-related problems since its onset to the present time. Since late Fiscal Year 2000, NIH ERA Management (ERA Management) has determined that a new architecture is required to support the Commons in the near future and beyond. Note that ERA Management is currently headed by Dr. John-J McGowan, the NIH ERA Project Manager, and Jim Cain (OER), the ERA Implementation Manager.

The current architecture of the Commons is based on the limited technology that was the only alternative a few years ago. Since then, the internet has progressed to another level of maturity with regard to new, emerging technology and standards. Many of these newer technologies have been embraced by the IT industry today as practical platforms upon which to build reliable systems on the internet. As is the case with both IMPAC II and the Commons (i.e., the ERA System), this includes systems that are enterprise-wide and mission-critical. As a result, ERA Management now believes that a new solution for the Commons should be implemented. The high level requirements associated with this approach are based on those that are typical of today's mission-critical, internet applications. In this context, high level requirements pertain to various factors such as security, scalability, performance, maintainability, portability, and other similar parameters. Based on these and other related factors, ERA Management—in conjunction with supporting contractors, including Logicon Federal Data/ROW Sciences (ROW Sciences)—initiated two parallel, yet independent, analysis activities. The purpose of these two efforts was to evaluate potential technology solution alternatives for the Commons. This evaluation has led to the creation of two separate analysis documents, referenced below. Note that both these efforts were evaluated based on similar criteria.

The first activity was performed by RNSolutions, Inc., who created a document, entitled, "ERA Commons Application Development Technology Analysis" (*include URL here*?). This document was evaluated and ultimately distributed by NIH to a selected audience in early December 2000. The analysis for this document was performed in the context of programming languages that support web development, and included discussions on Oracle PL/SQL, ASP, PHP, Perl, and Java. The final recommendation in this regard resulted in a Java implementation utilizing a multi-tiered application architecture. It also specifically recommends Sun Solaris as the preferred platform for hosting the application architecture.

The second analysis activity was performed by Turner Consulting Group (TCG) who created a document, entitled "ERA Commons Application Server Analysis" (*include URL here*?). Note the subtle difference in titles between both documents. As with RNSolutions, TCG's document was distributed for review by NIH in early December 2000. TCG's analysis was performed in the context of current internet-supported application server products, including associated programming languages and related tools. TCG's primary recommendation is WebLogic, which is an industry leading application server that supports multi-tiered architectures with Enterprise JavaBeans (EJB). EJB is the widely-adopted server-side component architecture for Java that enables rapid development of mission-critical applications that are versatile, reusable, and portable across middleware, while protecting IT investment and preventing vendor lock-in.

One general note of clarification concerning the term *Java* is in order here. Java has been used to represent various derivations of and associations with the Java language (Java applets, Java servlets, Enterprise JavaBeans, Java Server Pages, JavaScript). To provide a more precise description of the technology, this document will use the term *Java 2, Enterprise Edition (J2EE).* J2EE is the industry term currently used to represent the Java-based platform and technologies that support multi-tiered enterprise applications.

As all analysis activity to date has recommended J2EE-based solutions, NIH ERA Management has proceeded with investigating the specifics of how such a solution would be implemented in the ERA environment. To support this effort NIH has engaged an industry expert, Mr. Ani Dutta of Silicon Spirit Consulting Group, Inc., who has successfully implemented J2EE-based internet applications. Mr. Dutta is currently collaborating with ROW Sciences to develop three planning documents, targeted for publication release in early March 2001, to provide a roadmap for how the technology can be applied to meet desired ERA objectives.

This document, the "ERA J2EE Technology Overview", serves as an overview of the recommended architecture. It explains the concepts and technologies behind a standard J2EE-based implementation as well as how these will map to specifications associated with the ERA environment and requirements.

In a closely related second document, "ERA J2EE Platform and Tool Recommendations," a specific list of platforms (e.g., hardware, operating system, web servers, application servers, and other platform considerations) and tools (e.g., design tools, interactive development environments, mapping tools, and other related tools) will be recommended to implement the desired solution. Although some specific recommendations have already been made in earlier documents (e.g., WebLogic and Sun Solaris) these issues and other similar ones will be revisited to ensure they are compliant with the requirements of the architecture and the current ERA environment.

A third related document, the "ERA J2EE Management Plan," will provide a high level plan for how the effort will be managed and executed. Note that a more comprehensive Project Management Plan is currently under development in conjunction with the entire ERA Project. The "ERA J2EE Management Plan" will focus specifically on management plans and issues relating to the implementation of J2EE technology in the ERA environment. The plan will address general management issues (e.g., organization structure, risks and mitigation, staff training) and will provide a phased approach for how the Commons systems will be implemented and/or migrated. In addition, this segment will provide an outline of a proposed methodology—in concert with the current ERA/IMPAC II System Methodology—that will be used for the ERA implementation. This ensuing (i.e., adopted) methodology will describe specific tasks, milestones, level of effort, and other resource needs; and will document the full implementation process. This methodology will be based on “best practices” as defined in industry-accepted standards for J2EE development such as the Unified Modeling Language (UML) and the Rational Unified Process (RUP).

2 N-Tier Architecture Fundamentals

Since the benefits of N-Tier Architecture are well known and have been discussed elsewhere, we will just address the technical fundamentals of the N-Tier Architecture in this section. Figure 1, below, illustrates the fundamental pieces on an N-Tier architecture regardless of specifics such as types of languages (object oriented or procedural) or technology (CORBA, COM, J2EE, CICS).

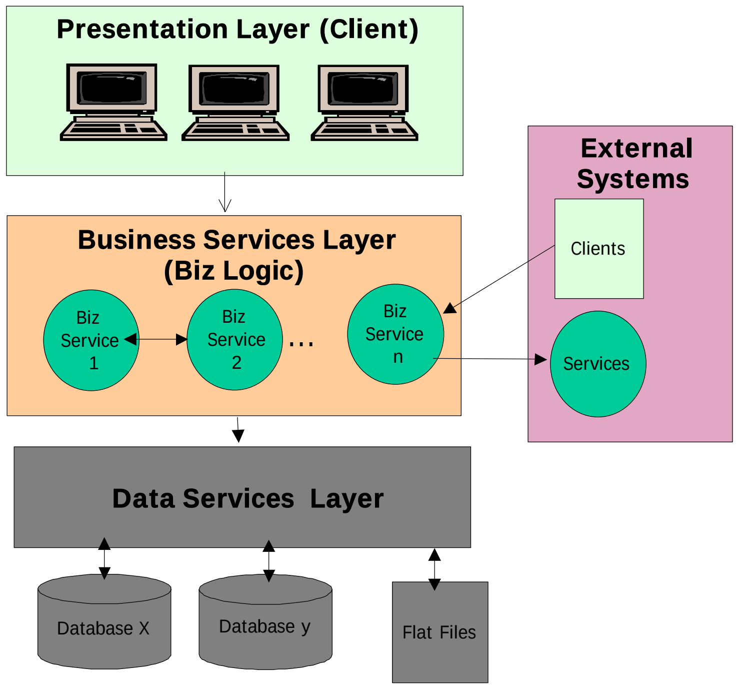

Figure 1. N-Tier Architecture (Logical View)

An N-tier architecture consists of three primary layers: Presentation Layer, Business Services Layer, and Data Services Layer. These Layers are described below.

Presentation Layer.

The Presentation Layer is responsible for displaying the system functionality to the user via the user interface. The user interface can be a desktop application, a web browser, a wireless phone display, a palmtop display, or any other presentation medium. For the purpose of this paper it is assumed that the Presentation Layer is displaying data on a web based user interface.

The Presentation Layer contains minimal functionality (code) for displaying on the presentation medium. It is unaware of any business logic. To access any business logic, the Presentation Layer will acquire the services of the Business Services Layer. Thus the Presentation Layer is called the *client* of the Business Services Layer. Specifically, it is a very *thin client* due to it's limited functionality.

Business Services Layer.

The Business Services Layer processes all business logic of the system. This layer is typically composed of several business services or subsystems. Each business service component provides its clients with very specific services in an anonymous fashion (i.e., without knowing or caring who the client is). A business service is only aware of the business rules or business logic of the system to be built. If it needs any data to perform its job, it is unaware of the source of that data (e.g., flat file, database, email). It will rely upon the Data Services Layer to access any data.

Data Services Layer.

The Data Services Layer is responsible for accessing data on behalf of the Business Services Layer. It is very aware of the format and type of the data but unaware of the semantics of the data. The Data Services Layer is composed of different data services dependent on the types of data source (e.g., database, flat file) and the vendor of the data source (e.g., Oracle, Sybase, Informix).

3-Tier To N-Tier

It is obvious why this architecture should be called 3-tier. However, if you look closely at Figure 1 you will see that unlike classic 3-tier, the business service components in the Business Services Layer can act as both a client and a server depending on the service. Consider that if business service component bsc1 provides service x, and business service bsc2 provides service y, it is very possible that bsc1 acts as a client to bsc2 to receive service y and, alternatively, bsc2 acts as a client to bsc1 to receive service x. Since the business service components can form a hierarchy within themselves depending on the calling sequence, they can form many levels/tiers in this hierarchy, thus forming the N-Tier architecture.

Interoperability with external systems.

Expanding on the concept that a business service component is unaware of its client, we can state that an external system component can access our business service component seamlessly, if that external system component complies to the standard interface of our business service component. Conversely, our business service component can act as a client to seamlessly access the external system's component if the interface is a well-defined standard.

3 J2EE Architecture And Programming Environment

J2EE is a standard architecture specifically oriented to the development and deployment of enterprise Web-oriented applications using the Java programming language. Independent Software Vendors (ISVs) and enterprises can use the J2EE architecture both for the:

- development and deployment of intranet applications, thus effectively replacing the two-tier and traditional 3-tier models, and
- development of internet applications, effectively replacing the cgi-based approach.

Figure 2 illustrates the logical view of a J2EE n-tier architecture model.

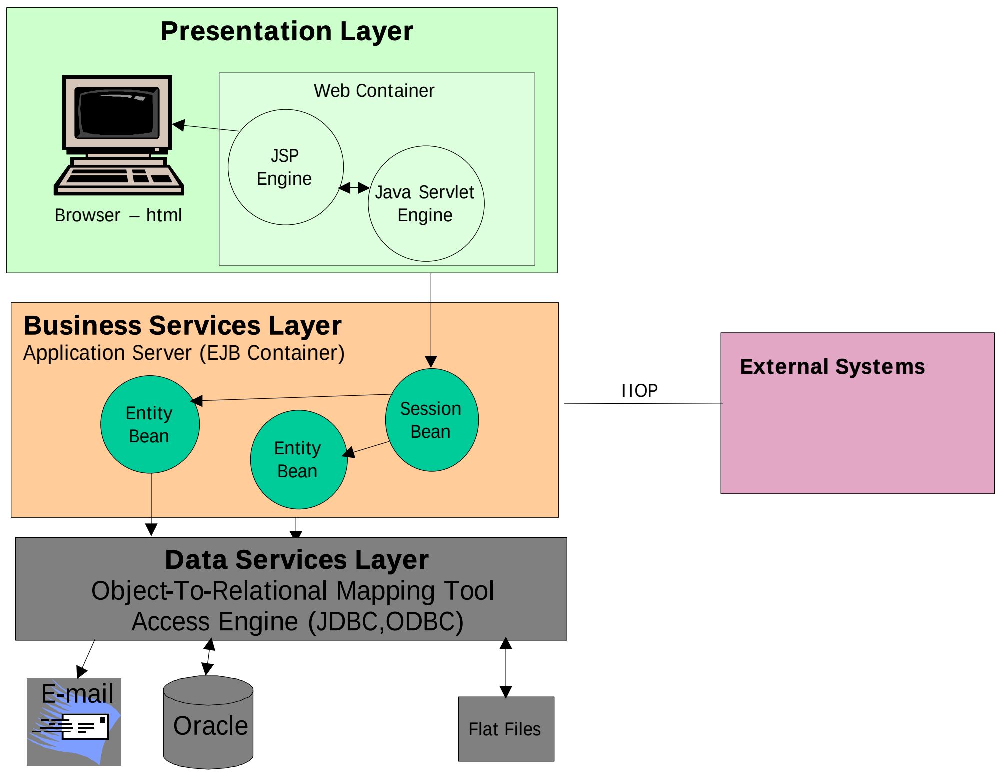

Figure 2. J2EE N-Tier Architecture

J2EE architecture is the platform-neutral architecture for the Java application environment. J2EE architecture extends "Write Once, Run Anywhere^{TM}" capability to reusable component development.

The "J2EE Platform Specification" (currently version 1.2) defines a set of standard component software Application Program Interfaces (APIs) for the Java platform. This specification was developed by Sun with a number of leading industry partners and was then refined based on broad general input from developers, customers, and end-users during a public review period.

To write J2EE application components, an application programmer needs to obtain a J2EE product from a product provider. Product providers are typically operating system, database system, application server, or web server vendors who implement the J2EE platform according to the "Java 2 Enterprise Edition Platform Specification" (available publicly at http://java.sun.com/j2ee/download.html#platformspec). A J2EE product contains the J2EE APIs, J2EE application server, a web server, and deployment tools. In short, everything needed to write and assemble application components, and to deploy complete J2EE applications.

The following sections will describe the various pieces of the J2EE N-tier architecture. Please refer to Figure 2 for the discussion in this section.

3.1 J2EE Presentation Layer

Web Containers and Web Components

A *Web container* is a system-level entity that provides life-cycle management and runtime support for JSP pages and servlet components. A *web component* is either a *servlet* or a *JSP page*, which can use the services of its container. The following paragraphs describe the JSP and servlet technology in detail.

JavaServer Pages (JSP) technology allows web developers and designers to rapidly develop and easily maintain, information-rich, dynamic web pages that leverage existing business systems. As part of the Java family, JSP technology enables rapid development of web-based applications that are platform independent. JavaServer Pages technology separates the user interface from content generation, enabling designers to change the overall page layout without altering the underlying dynamic content.

JavaServer Pages technology uses eXtensible Markup Language (XML)-like tags and scriptlets written in the Java programming language to encapsulate the logic that generates the content for the page. Additionally, the application logic can reside in server-based resources (e.g., Enterprise JavaBeans) that the page accesses with these tags and scriptlets. Any and all formatting (HTML or XML) tags are passed directly back to the response page. By separating the page logic from its design and display and supporting a reusable component-based design, JSP technology makes it faster and easier than ever to build web-based applications.

JavaServer Pages technology is an extension of the Java Servlet technology. Servlets are platform-independent, 100% pure Java server-side modules that fit seamlessly into a web server framework and can be used to extend the capabilities of a web server with minimal overhead, maintenance, and support. Unlike other scripting languages, servlets involve no platform-specific consideration or modifications; they are Java application components that are downloaded, on demand, to the part of the system that needs them. Together, JSP technology and servlets provide an attractive alternative to other types of dynamic web scripting/programming that offers platform independence, enhanced performance, separation of logic from display, ease of administration, extensibility into the enterprise, and, most importantly, ease of use.

For a standard J2EE implementation, the following processing steps illustrate how the various components of the presentation layer interact with each other in a typical run time scenario:

1. The web server receives a request from the user browser and transmits it to a Java servlet. The servlet will interpret the request and then invoke an operation on an EJB on the Application Server.

2. Based on the resultant data from step 1, the servlet will put the data in the servlet session data structure of the next JSP page to display and pass this to the JSP engine.

3. The JSP engine will generate the next HTML page based on the JSP pages' static html content and the dynamic data content from the session data structure passed in by the servlet.

4. The JSP engine will pass the HTML page to the web server process.

5. The web server process will forward the HTML via the HTTP protocol to the user browser.

3.2 J2EE Business Services Layer

3.2.1 Enterprise Beans

Enterprise beans are server components written in the Java programming language. They contain the business logic for the application. For example, a checkbook client might invoke the debit and credit methods of an account enterprise bean.

There are two types of enterprise beans: session beans and entity beans.

Session Beans

A session bean represents a client in the J2EE server. A client communicates with the J2EE server by invoking the methods that belong to a session bean. For example, an online shopping client might invoke the enterOrder method of its session bean to create an order. A session bean converses with the client, and can be thought of as an extension of the client. Each session bean can have only one client. When the client terminates, its corresponding session bean also terminates. Therefore, a session bean is transient, or non-persistent.

Entity Beans

An entity bean represents a business object in a persistent storage mechanism such as a database. For example, an entity bean could represent a customer, which might be stored as a row in the customer table of a relational database. (An entity bean's information does not have to be stored in a relational database. It could be stored in an object database, a legacy application, a file, or some other storage mechanism. The type of storage mechanism depends on the particular implementation of EJB technology.)

Comparing Session and Entity Beans

Although both session and entity beans run in an EJB container, they are quite different. The following table contrasts session and entity beans:

	Session Bean	**Entity Bean**
Purpose	Performs a task for a client.	Represents a business entity object that exists in persistent storage.
Shared Access	May have one client.	May be shared by multiple clients.
Persistence	Not persistent. When the client terminates its session bean is no longer available.	Persistent. Even when the client session of EJB container terminates, the entity state remains in a data source.

The flexibility of the EJB architecture allows you to build applications in a variety of ways. Consider the role of session and entity beans in a real life implementation such as an online shopping application. In this application the JSP/Servlet within the Web Container would be the client of a shopping session bean. When the application needs to find a product or enter an order, it instructs the servlet to call the appropriate business methods in the session bean. The session bean would then be the client of entity beans which managed the relevant objects, such as an order, product, or customer. Because entity beans are persistent, their state is stored in the database.

3.2.2 EJB Container

Enterprise bean instances run within an EJB container (*container* for short). The container is a runtime environment that controls the enterprise beans and provides them with important system-level services. Since you don't have to develop these services yourself, you are free to concentrate on the business methods in the enterprise beans. The container provides the following services to enterprise beans:

- Persistence
- Transaction Management
- Security
- Remote-Client Connectivity.
- Life-Cycle-Management.
- Database Connection Pooling.

Persistence (CMP and BMP)

Persistence means that the state of an object needs to be saved to some sort of persistent storage (typically a file or a database), allowing the object to be read in and used at some later time, instead of being recreated each time. In non J2EE implementations, an object's persistence had to be implemented manually by the programmer, typically by implementing SQL-based database calls to save and restore its state. With a J2EE implementation, the fields that should be persistent are simply specified, and the container then takes the responsibility for saving and restoring these fields. This reduces the programming effort by the developer significantly. This feature is called *container managed persistence (CMP)*. In some cases, using CMP is not possible. For example, if you are integrating J2EE into an existing 2-Tier system where even though the business logic is tightly intertwined with data access, you may wish to retain your investment in that codebase. In this situation the developer would have to trade off the benefits of the container and manually develop code to integrate with the current code (typically a stored procedure) within the bean. This feature is called *bean managed persistence (BMP)*.

Transaction Management

When a client invokes a method in an enterprise bean, the container intervenes in order to manage the transaction. Because the container manages the transaction, code transaction boundaries do not have to be coded in the enterprise bean. The code required to control distributed transactions can be quite complex. Instead of a programmer writing and debugging complex code, the enterprise bean's transactional properties are simply declared in the deployment descriptor file. The container itself reads the file and handles the enterprise bean's transactions.

Security

The container permits only authorized clients to invoke an enterprise bean's methods. Each client belongs to a particular role, and each role is permitted to invoke certain methods. The roles and the methods they may invoke are declared in the enterprise bean's deployment descriptor. Because of this declarative approach, code routines that enforce security are not needed.

Remote Client Connectivity

The container manages the low-level communications between clients and enterprise beans. After an enterprise bean has been created, a client invokes methods on it as if it were in the same virtual machine.

Life Cycle Management

An enterprise bean passes through several states during its lifetime. The container creates the enterprise bean, moves it between a pool of available instances and the active state, and, finally, removes it. Although the client calls methods to create and remove an enterprise bean, the container performs these tasks behind the scenes.

Database Connection Pooling

A database connection is a costly resource. Obtaining a database connection is time-consuming and the number of connections may be limited. To alleviate these problems, the container manages a pool of database connections. An enterprise bean can quickly obtain a connection from the pool. After the bean releases the connection, it may be re-used by another bean.

3.3 J2EE Data Services Layer

The Enterprise JavaBeans specification does not require an implementation to support a particular type of database. Therefore, the databases supported by different J2EE implementations may vary.

The Data Services Layer is primarily supported by the EJB container in conjunction with an object to relational mapping tool. Session beans typically will not access data directly. Instead, the session bean will call the appropriate set of entity beans to complete a client request. The entity beans in turn will access its object attributes. The object attributes will be populated with data from the data source via the container at runtime.

Setting up the data source for the container is a fairly easy task. During development, XML based deployment descriptor files are populated with pointers to the data source(s) of the data to be accessed. This information includes references to the object relational mapping tool as well (in the case where the database is a relational database). Also during entity bean development time, the object relational mapping tool is run to generate data access (SQL) code to access the database tables for each entity bean.

During application runtime, when an entity bean is called, the data required for the entity bean is loaded by the container in memory based on the access code generated by the object relational mapper tool. Again, this type of persistence support by the container is called *Container Managed Persistence (CMP).*

If for some reason, the container needs to be bypassed (typically while integrating with existing systems which were developed using non-EJB technology), the bean developer can insert the data access code manually. Again, this mechanism of direct data access by the bean is called *Bean Managed Persistence (BMP).*

3.4 Interoperability with External Systems.

3.4.1 XML

XML is becoming a de facto standard for data exchange in the industry. The Commons be using XML as the primary mechanism for exchanging information with external systems. XML is a text-based markup language for defining what information is, as opposed to how it looks.

For example, in an XML document you can define the words *Zelda Zee* to be a customer name and *ZZ* to be a customer ID. This way, the information can be handled according to what it is. In a HyperText Markup Language (HTML) document, you can make Zelda Zee and ZZ appear in bold or italics, but you cannot define them to be a customer name and customer ID.

Document Type Definition (DTD) is part of the XML specification. It specifies the kinds of tags that can be included in an XML document, and the valid arrangements of those tags. A DTD is used to validate XML structures to make sure you create only valid XML structures and that XML structures you send or receive are valid.

Programs to access XML documents use industry standard specifications such as Simple API for XML (SAX) or Document Object Model (DOM) APIs. Typically a utility component in the J2EE application will parse data stored in XML format and load it into an object format based on the DOM or SAX APIs. Then, the utility would act as a client and invoke the business services layer to run validation of the DOM /SAX based XML data before populating the data into the corresponding Enterprise Java Beans. Third party tools providing the DOM/SAX APIs may be used to reduce the development interval.

3.4.2 RMI and CORBA/IIOP

In terms of implementation, J2EE solutions can only be implemented with the Java language. However, enterprise systems must be able to converse or interoperate with other systems (of partners, customers, etc.) which are built with other languages (e.g., COBOL, C, C++). The Common Object Request Broker Architecture (CORBA) was a specification developed by the Object Management Group (OMG) before the advent of J2EE to address seamless communication between disparate systems. The transport mechanism that was used for this purpose was IIOP (Internet Inter Orb Protocol). The J2EE specification initially used a java based Remote Method Invocation (RMI) mechanism but later combined this with IIOP to achieve the interoperability needed to integrate with CORBA based systems.

4 Applying J2EE To ERA Commons

In this section we will discuss the application of the J2EE architectural reference model to the NIH ERA Commons application. Note that this a technical approach only at this point. We will finalize the Commons architecture and design through application of the development methodology as described in the "ERA J2EE Management Plan".

Figure 3, below, shows the standard J2EE n-tier architecture within the Commons environment. We will discuss the Commons specific implementations in each of the three layers.

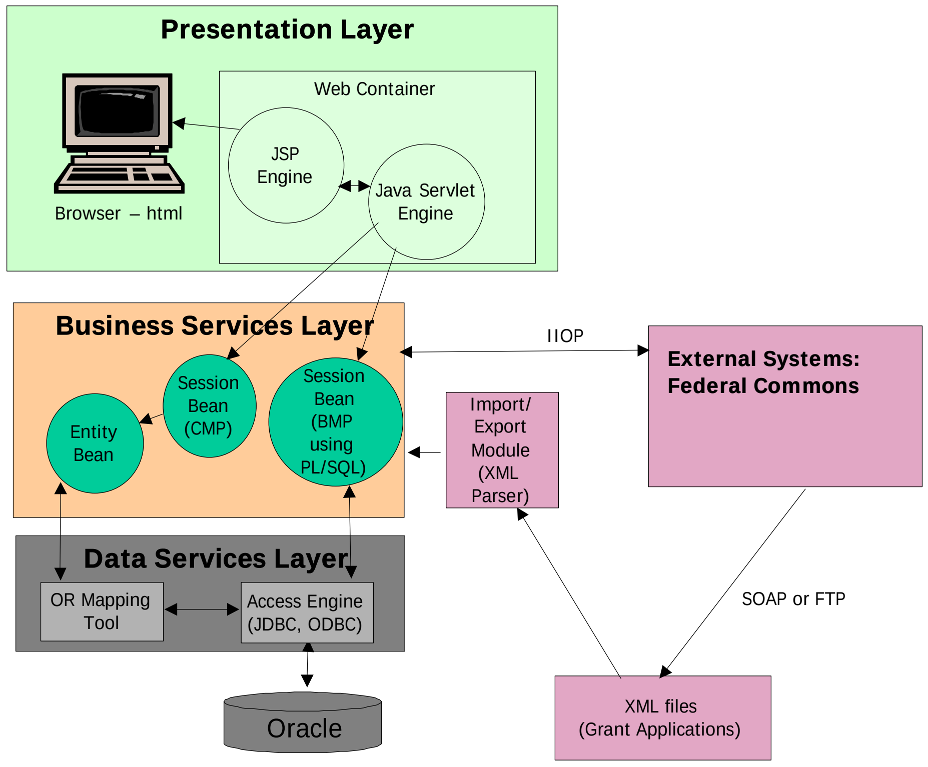

Figure 3. J2EE Applied to Commons

4.1 Presentation Layer

The Commons implementation of this layer will be identical to the J2EE reference model. There will be a very thin client implemented by JSP 1.2 pages for display, and a controller Java Servlet to process commands invoked by the user on the JSP form. The Servlet will then invoke the Enterprise JavaBeans in the Business Services Layer. If desired, the actual HTML pages (screens) currently used by the Commons can be reused in the J2EE architecture. However, it may be desirable to take this opportunity to redesign certain screens based on user requirements.

In addition to the J2EE architectural standard, several design standards called *design patterns* have emerged in the industry as a standard way of building reusable frameworks. Details of the design pattern to be used will be determined in accordance with the development methodology. The *Model View Controller (MVC)* pattern is currently being considered to build a reusable framework for the Presentation Layer to gather data from the Business Services Layer. For more information, refer to http://www.javaworld.com/jw-12-1999/jw-12-ssj-jspmvc.html for a thorough explanation and an example of the MVC.

4.2 Business Services Layer

The Business Services Layer will follow the J2EE 1.2 (EJB 1.1) specification. Some vendors are providing implementations for the newest EJB 2.0 specification. We will consider those features depending on their maturity level at Commons development time.

Special consideration needs to be given to retain the current investment in the PL/SQL based business logic (both in the existing Commons and IMPACII systems). PL/SQL based business logic can be easily reused within the Business Services Layer by using *Bean Managed Persistence (BMP)*. This will allow Enterprise Java Beans to directly call PL/SQL procedures and packages stored in the Oracle RDBMS. As new business logic is developed, implementation of this in either a CMP (EJB only) or BMP (EJB and PL/SQL) approach will be determined as a detailed design decision based on the specific business requirements.

One important note on this topic is that the reuse of PL/SQL in this case is focused on business logic. Certain Commons systems (X-Train and CRISP) also use Oracle PL/SQL extensions to support the Presentation Layer (e.g., handling HTTP requests and generating HTML pages). This specialized use of PL/SQL cannot be reused in a J2EE environment; however, the HTML pages embedded within the PL/SQL code can be reused if the same user interfaces are desired.

4.3 Data Services Layer

Wherever possible and practical, the Data Services Layer will be fully separated from the Business Services Layer to be compliant with the J2EE standard. In these instances an object to relational mapping tool will be used to map the Bean attributes to the underlying Oracle database housing the Commons data. In cases where existing PL/SQL is used, the Data Services layer will only consist of the access engine (JDBC or ODBC) that will support calls to the Oracle RDBMS from an EJB. A good survey of object-data integration issues can be found online in a two part article entitled "Crossing the Object-Data Divide" (http://www.sdmagazine.com/articles/2000/0003/0003j/0003j.htm and http://www.sdmagazine.com/articles/2000/0004/0004k/0004k.htm).

Another significant data services issue to consider is the physical database architecture of the new Commons system. Currently, the Commons uses a separate physical database from IMPAC II with a vastly different physical data model. Based on a database analysis that occurred during 2000, it was determined that the best course of action is to modify the Commons data model to more closely mimic the IMPAC II model. However, it is unclear at this point whether or not the two databases should be physically combined.

Physically combining the databases would greatly simplify the solution by removing any issues concerning replication or reconciliation between the databases (which has been a major problem to date). However, there are a host of security concerns to be considered if the internet-based Commons applications were to directly access the IMPAC II enterprise database. The advanced security capabilities of the J2EE architecture can be used to help alleviate some of these concerns, but any decision to physically merge the databases will need to be made based on a careful consideration of all the technical and business issues during the full design process.

4.4 Interoperability with External Systems.

Any external systems (e.g., Federal Commons) can converse with Commons via the following mechanisms.

API Based. An Application Programming Interface (API) mechanism will require programs from each system to communicate with each other via a standard application level protocol. The logical choice here is the IIOP protocol. Using this mechanism we will be able to seamlessly integrate with other systems using different languages and operating systems as long as they can converse with Commons in IIOP.

File Based. Some of the Commons clients (grantee organizations) use their own tools to generate research grant applications that are well embedded in their business workflow. Rather than changing the way the clients do business, the Commons can simply ask them to transmit the application files to it in an open standard format. The logical choice here is XML, which can be processed by a J2EE application and allows the flexibility needed to define the semantics of the data. Figure 3 shows the link from the Federal Commons to the ERA Commons. The transport of the XML files can be done via FTP or via a more advanced protocol specifically targeted for XML called Simple Object Based Protocol (SOAP).

The Import/Export module can read the XML based grantee data into an XML based Domain Object Model (DOM). Next, the Import/Export model can invoke the services of the Business Services Layer to run validation and business rules checking before populating the data into the Commons data source.

Appendix A—Acronyms/ Abbreviations/Definitions

API	Application Program Interface
BMP	Bean-Managed Persistence
CMP	Container-Managed Persistence
Commons	NIH/ERA Commons System
CORBA	Common Object Request Broker Architecture
DOM	Document Object Model
DTD	Document Type Definition
EJB™	Enterprise Java Beans. A component architecture for the development and deployment of object-oriented, distributed, enterprise-level applications. Applications written using the Enterprise JavaBeans architecture are scalable, transactional, and secure.
EJB Container	A container that implements the EJB component contract of the J2EE architecture. This contract specifies a runtime environment for enterprise beans that includes security, concurrency, life cycle management, transaction, deployment, naming, and other services. An EJB container is provided by an EJB or J2EE server.
Enterprise Bean	A component that implements a business task or business entity and resides in an EJB container; either an entity bean or a session bean
Entity Bean	An enterprise bean that represents persistent data maintained in a database. An entity bean can manage its own persistence or it can delegate this function to its container.
ERA	Electronic Research Administration
HTML	Hypertext Markup Language. A markup language for hypertext documents on the Internet. HTML enables the embedding of images, sounds, video streams, form fields, references to other objects with URLs and basic text formatting.
IIOP	Internet Inter-ORB Protocol. A protocol used for communication between CORBA object request brokers.
IMPAC II	Information for Management, Planning, Analysis, and Control system, II
ISV	Independent Software Vendor
J2EE™	Java 2, Enterprise Edition
JDBC™	An API for database-independent connectivity between the J2EE platform and a wide range of data sources
JMS	Java Message Service

JSP	JavaServer Pages. An extensible Web technology that uses template data, custom elements, scripting languages, and server-side Java objects to return dynamic content to a client. Typically the template data is HTML or XML elements, and in many cases the client is a Web browser.
MVC	Model View Controller
NIH	National Institutes of Health
OER	Office of Extramural Research
OMG	Object Management Group
RDBMS	Relational Database Management System
RMI	Remote Method Invocation. A technology that allows an object running in one Java virtual machine to invoke methods on an object running in a different Java virtual machine.
RUP	Rational Unified Process
SAX	Simple API for XML. An event-driven, serial-access mechanism for accessing XML documents.
Servlet	A Java program that extends the functionality of a Web server, generating dynamic content and interacting with Web clients using a request-response paradigm
Session Bean	An enterprise bean that is created by a client and that usually exists only for the duration of a single client-server session. A session bean performs operations, such as calculations or accessing a database, for the client.
SQL	Structured Query Language
TCG	Turner Consulting Group
UML	Unified Modeling Language
Web Component	A component that provides services in response to requests; either a servlet or a JSP page
Web Container	A container that implements the Web component contract of the J2EE architecture. This contract specifies a runtime environment for Web components that includes security, concurrency, life cycle management, transaction, deployment, and other services. A Web container provides the same services as a JSP container and a federated view of the J2EE platform APIs. A Web container is provided by a Web or J2EE server.
XML	eXtensible Markup Language. A markup language that allows you to define the tags (markup) needed to identify the data and text in XML documents. J2EE deployment descriptors are expressed in XML.