INTEGRA
CONSORZIO INFORMATICO

Contratto per *"l'acquisizione di servizi di Assistenza specialistica per la gestione e l'evoluzione del patrimonio software della Regione Basilicata".*

Repertorio n. 11016 del 25/09/2009

Progettazione: Tecnologie e ambienti di sviluppo

"ANAGRAFICA UNICA DEL PERSONALE: IPA REGIONALE, ATTRIBUTE AUTHORITY E GESTIONE ITER ACCREDITAMENTO AI SERVIZI"

CONTROLLO DEL DOCUMENTO

APPROVAZIONI			
	Data	**Autore**	
Redatto da:	26/01/2011	Dott. Maurizio Argoneto	
Approvato da:		Dott. Nicola Petrizzi	
VARIAZIONI			
Versione prec.	**Data**	**Autore**	**Paragrafi modificati**
DISTRIBUZIONE			
	Copia n°	**Destinatario**	**Locazione**
		Dott. Nicola Petrizzi	Regione Basilicata

Indice

Introduzione

L'architettura predominante nello sviluppo di **applicazioni server-side** è quella a livelli, in tale architettura i componenti sono raggruppati in livelli separati, ciascuno dei quali svolge un compito ben definito:

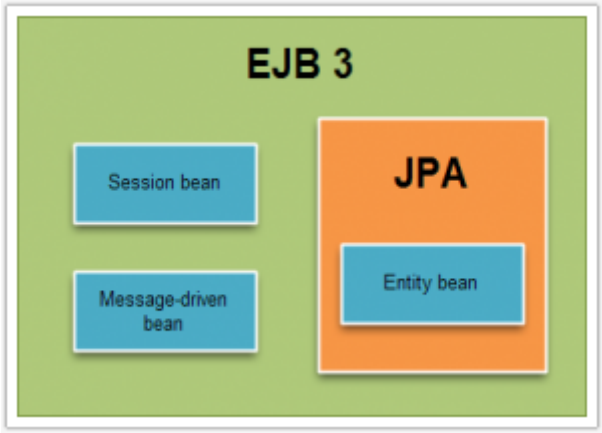
- presentation layer: è responsabile della visualizzazione della Graphical User Interface (GUI) e della gestione dell'input utente (passando le richieste al business logic layer)
- business logic layer: contiene tutta la logica dell'applicazione ovvero i processi che l'applicazione può eseguire, recupera e salva dati interagendo con il persistence layer
- persistence layer: fornisce un'astrazione ad alto livello e object-oriented del database layer
- database layer: consiste di un relational database management system.

Enterprise JavaBeans è una piattaforma per la creazione di applicazioni business portabili, riusabili e scalabili mediante il linguaggio java. Un'applicazione è costituita da componenti che vivono all'interno di un **EJB container** che fornisce a tali componenti un certo numero di servizi quali la gestione della sicurezza, delle transazioni, i supporto per i web-services… I componenti EJB sono simili a un qualsiasi altro **POJO (Plain Old Java Object)** e sono suddivisi in tre tipi: session bean, message-driven bean e entity bean. I primi due sono utilizzati per implementare la logica di business mentre gli entity bean sono utilizzati per la persistenza. In particolare, un **session bean** viene invocato da un client allo scopo di effettuare una specifica elaborazione: il

termine "session" fa riferimento al fatto che l'istanza di un bean di questo tipo non sopravvive a crash o shutdown del server. Esistono due tipi di session bean:

- **stateful**: sono quei bean il cui stato viene salvato fra una richiesta e l'altra del client
- **stateless**: sono quei bean che non mantengono alcun stato e vengono utilizzati per modellare quelle elaborazioni che si completano con una sola richiesta

I session bean possono essere invocati o localmente o da remoto mediante Java RMI.



I **Message-driven Bean** si differenziano dai session bean per il fatto che non vengono mai invocati direttamente dai client. Piuttosto i MDB sono attivati da messaggi inviati a un **messaging server**. Infine gli **entity bean** sono oggetti java che costituiscono la rappresentazione dei dati dell'applicazione e che vengono memorizzati in maniera persistente in un database. In EJB 3 al persistenza è gestita da **Java Persistence API** mediante **object-relational mapping (ORM)**: questo termine essenzialmente denota il processo di mapping dei dati fra gli oggetti e le tabelle di un database. Mentre l'esecuzione di un session bean o di un message-driven bean richiede un EJB container, per assicurare la persistenza degli entity bean è necessario ricorrere ad un **persistence provider**.

Un persistence provider è essenzialmente un framework ORM (come Hibernate o TopLink) che supporta le Java Persistence API.

1.1 Scopo del Documento

Il presente documento viene redatto al fine di illustrare la soluzione progettuale e tecnologica del DB Unico del Personale regionale che verrà inglobato in un contesto più generale d'inquadramento e cioè la mappatura di tali dati in una struttura che si riconduce all'IPA Nazionale e che garantirà nel medio periodo la centralizzazione di tutti i dati relativi agli enti, la loro struttura e il personale/servizi che essi espongono.

1.2 Definizioni ed Acronimi

Lista e descrizione delle definizioni e degli acronimi.

Acronimo	Significato
AA	Attribute Authority
IMS	Identity Management System

1.3 Riferimenti

Documento d'integrazione degli applicativi nell'IMS.

Documento dell'IPA nazionale

La scelta tecnologica

Nell'ambito di questo progetto la scelta tecnologica ricadrà su una duplice possibilità. Verranno infatti creati dell EJB per poter utilizzare strumenti più veloci sicuri e affidabili da applicazioni java e poi utilizzeremo gli EJB come dei Web Services. Per implementare tale scenario sono possibili diverse soluzioni. Grazie alla combinazione di **AXIS e J2EE** è possibile ottenere l'implementazione di un sistema distribuito che può basare:

- la robustezza, la sicurezza, l'affidabilità, la gestione delle transazioni e la persistenza delle informazioni, la gestione delle risorse in generale, basandosi esclusivamente sui servizi offerti dalla piattaforma J2EE.
- l'interoperabilità tra l'applicazione lato server e una specifica applicazione lato client (realizzata in qualsiasi linguaggio di programmazione) grazie all'utilizzo dei Web Services.

Le soluzioni principali per esportare gli ejb tramite web services sono sostanzialmente le seguenti:

Soluzioni proprietarie degli EJB container

Quasi tutti i container permettono di esportare i propri EJB come web services. All'atto del deployment si specifica di voler esportare l'ejb come web services. Il tool chiederà di inserire il file WSDL che descrive il servizio ed un file XML per ulteriori informazioni che serviranno al container all'atto del deployment.

Utilizzare un motore SOAP esterno all'EJB container: wrapper

Utilizzando un motore SOAP esterno all'EJB container, posso implementare un web services che fa da **wrapper** tra i miei EJB ed il client SOAP. Il web services implementerà al suo interno tutta la logica per l'accesso all'EJB. La presente soluzione è sempre applicabile, con qualsiasi sistema si utilizza. In realtà è molto

più utile, quando non si vuole esportare esattamente la logica implementata negli EJB, mascherandone parte di essa.

Soluzione offerta da AXIS

Axis permette d'interagire con gli EJB di tipo Session Stateless in modo semiautomatico, tramite l'utilizzo di uno dei suoi tanti provider. Precisamente il provider "Java:EJB". Non serve altro che scrivere un file WSDD per il deployment su axis dei servizi che si vogliono esportare inserendo come parametri al servizio i seguenti attributi:

- *beanJndiName:* il nome registrato nell'architettura JNDI del Session Bean Stateless con cui si vuole interagire;
- *homeInterfaceName:* il nome dell'interfaccia home remota;
- *remoteInterfaceName:* il nome dell'interfaccia remota del servizio;
- *jndiContextClass:* il nome della classe che implementa l'interfaccia JNDI;
- *jndiURL:* l'url del server di naming.

Sicurezza di accesso a risorse EJB

Enterprise JavaBeans ha un modello di sicurezza elegante, flessibile e portabile su sistemi eterogenei. La sicurezza delle applicazioni coinvolge essenzialmente due funzioni: l'autenticazione e l'autorizzazione.

L'**autenticazione** è il processo attraverso il quale si verifica l'identità dell'utente e si traduce tipicamente nel controllo di un username e una password. L'**autorizzazione** invece è il processo mediante il quale si determina quale utente ha accesso a una data risorsa una volta che è stato autenticato: in un sistema aperto un utente autenticato può accedere a qualsiasi risorsa, in molti

sistemi però occorre restringere l'accesso alle risorse agli utenti autorizzati. Per rendere più efficiente il mantenimento delle autorizzazioni può essere utile organizzare gli utenti in **gruppi** e quindi garantire l'accesso alle risorse agli utenti che appartengono ad un dato gruppo. Il concetto di **ruolo** è strettamente connesso al concetto di gruppo ed ha a che vedere con le "responsabilità" ricoperte da un dato utente dell'applicazione all'interno del gruppo di appartenenza.

La sicurezza in Java EE si basa su **Java Authentication and Authorization Service API (JAAS)**. JAAS è progettato in maniera tale che l'autenticazione e l'autorizzazione possano essere effettuate a qualsiasi livello Java EE (web, EJB, …) di fatto però molte applicazioni Java EE sono accessibili dal web e condividono le informazioni di autenticazione fra i livelli. JAAS facilita questo approccio perché una volta che un utente si autentica ad un dato livello, l'autenticazione si propaga agli altri livelli evitando in tal modo di dover ripetere le procedure di autenticazione: l'oggetto **Principal** rappresenta l'authentication context condivisibile tra i livelli.

Una volta che un utente si autentica ad un dato livello fornendo le sue credenziali, JAAS fornisce un Principal che è associato ad uno o più ruoli, l'application server quindi controlla se l'utente è autorizzato ad accedere alle risorse richieste.

Se necessario il Principal viene propagato fra i livelli in maniera del tutto trasparente allo sviluppatore. Le specifiche servlet nascondono un grande numero di dettagli ad alto livello per l'autenticazione e l'autorizzazione. Lo sviluppatore deve semplicemente indicare al container quali risorse vuole proteggere, come proteggerle , come gestire le credenziali e quali ruoli hanno accesso alle risorse: il container fa il resto. La sicurezza a livello web è configurata usando gli elementi **login-config** e **security-constraint** del file web.xml.

Notifica delle variazioni sull'AA con JMS

Di solito le comunicazioni fra i componenti di un'applicazione sono sincrone il che significa che l'oggetto che effettua la chiamata e l'oggetto che viene invocato devono essere entrambi presenti affinchè la comunicazione possa avvenire. La **comunicazione sincrona** comporta anche il fatto che il client deve attendere che l'oggetto invocato completi l'elaborazione della richiesta prima di poter procedere oltre. I **Message-Oriented Middleware (MOM)** permettono a due oggetti di comunicare in maniera asincrona facendo da ponte fra l'uno e l'altro: in questo caso non è più necessaria la presenza contemporanea dei due oggetti in quanto il messaggio verrà trattenuto dal MOM e consegnato solo quando il destinatario potrà riceverlo.

Un message-oriented middleware è un software che consente a due componenti di **comunicare in maniera asincrona**. Quando un componente invia un messaggio, il MOM lo memorizza nella locazione specificata dal mittente e ne comunica l'avvenuta ricezione. Il componente che invia il messaggio è detto **producer**, la locazione nella quale il messaggio viene memorizzato è detta **destinazione**. Successivamente il componente interessato al messaggio, detto **consumer**, può recuperarlo dalla locazione nella quale è stato memorizzato. Un **modello di messaging** definisce il modo in cui il processo di messaging coinvolge mittenti e destinatari. I due modelli più popolari, standardizzati in Java EE, sono il **point-to-point** e il **pubish-subscribe**. Nello schema point to point, un messaggio viaggia da un produttore a un singolo consumatore e le destinazioni del messaggio sono dette code; questo modello non garantisce che i messaggi siano consegnati in secondo un particolare ordine. Nello schema publish-subscribe un singolo produttore invia un messaggio ad una destinazione che prende il nome di topic e tutti i consumatori che sono registrati presso quel topic possono riceverlo. Questo modello lavora si adatta

particolarmente bene nel caso in sui si voglia effettuare il **broadcasting di informazioni** verso diversi sistemi.

Java Messaging Service (JMS) fornisce uno standard di accesso ai MOM che costituisce un'alternativa all'utilizzo di API proprietarie; ad eccezione di Microsoft Message Queuing, la maggior parte dei prodotti MOM infatti supporta JMS. Supponiamo di voler lanciare dei processi attraverso una piccola web application senza volerne attendere il completamento.
In questo caso sfruttando il modello point-to-point potremmo inviare un messaggio contenente il nome del comando che deve essere eseguito ad una coda, e fare in modo che un consumer prelevi i messaggi dalla coda e ne esegua i comandi.

Questo modello può essere attuato direttamente con degli EJB oppure dotandosi di un software di terze parti che provveda a notificare e a gestire le code.