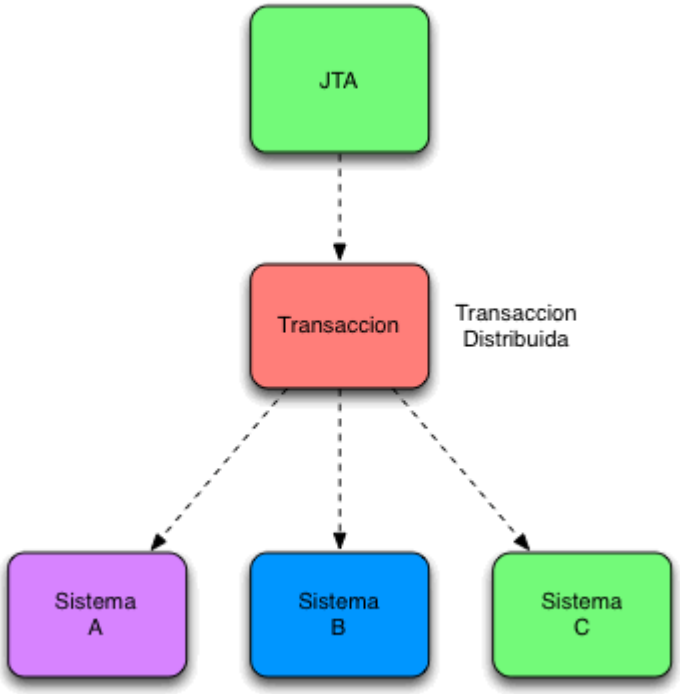
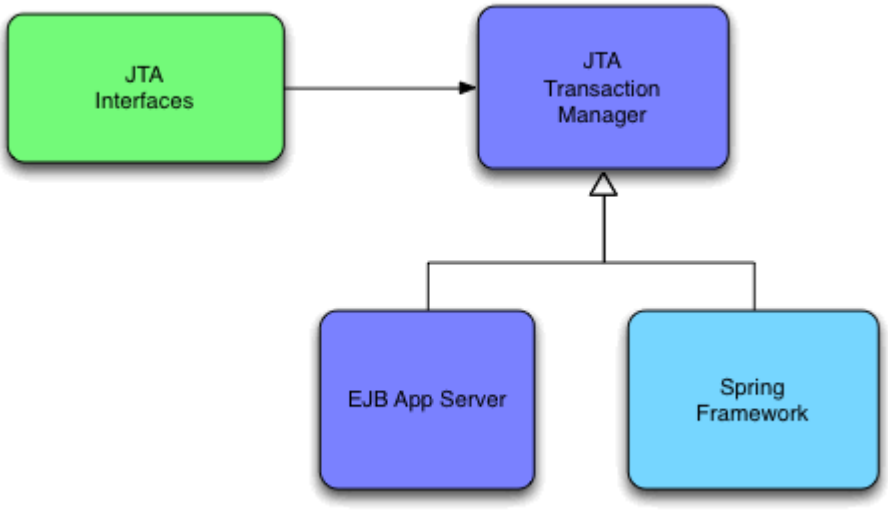
El concepto de Java JTA genera muchas dudas y el otro día a través del blog me han preguntado si podía escribir un artículo sobre el tema. JTA (Java Transaction API) existe para generar una abstracción sobre la gestión de transacciones entre varios sistemas, permitiendo transacciones distribuidas.



Interfaces y Java Transacional Manager

JTA gestiona dos conceptos fundamentales: Uno es el de Transactional Manager que define como ha de implementarse un gestor de transacciones para ser compatible con JTA. Esto es algo de lo que deben preocuparse las empresas que implementan servidores de aplicaciones. El otro concepto son los JTA interfaces que delimitan los interfaces y anotaciones que los desarrolladores usan para gestionar las transacciones a nivel de código.



Java JTA Interfaces

Existen dos formas de trabajar con JTA a través de anotaciones lo que se denomina en el mundo de EJB Container Managed Transaction (CMT) o bien a través del uso de interfaces clásicos lo que se denomina Bean Managed Transaction (BMT). Este último usa el interface de UserTransaction que permite un control de la transacción mas a detalle. En nuestro caso vamos a implementar un ejemplo sencillo que use anotaciones. Empezaremos por construir un proyecto JPA que se encargue de mapear un Java Bean en la base de datos.

```
package com.arquitecturajava.bo;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Persona {

@Id
```

```
private String nombre;

public String getNombre() {
return nombre;
}

public void setNombre(String nombre) {
this.nombre = nombre;
}
}
```

Realizada esta primera operación el siguiente paso es configurar el persistence.xml para que se apoye en un dataSource previamente configurado con cada servidor de aplicaciones .

```
&amp;lt;?xml version="1.0" encoding="UTF-8"?&amp;gt;
&amp;lt;persistence version="2.1"
xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"&amp;gt;
&amp;lt;persistence-unit name="Persistencia"&amp;gt;
&amp;lt;provider&amp;gt;org.hibernate.ejb.HibernatePersistence&amp;lt;
/provider&amp;gt;
&amp;lt;jta-data-
source&amp;gt;java:jboss/datasources/MySqlDS&amp;lt;/jta-data-
source&amp;gt;
&amp;lt;/persistence-unit&amp;gt;
&amp;lt;/persistence&amp;gt;
```

Java JTA y EJBs

Queda por construir un EJB que gestione una transacción a través del uso de anotaciones:

```
package com.arquitecturajava.ejb;

import javax.ejb.LocalBean;
import javax.ejb.Stateless;
import javax.ejb.TransactionAttribute;
import javax.ejb.TransactionAttributeType;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;

import com.arquitecturajava.bo.Persona;

/**
* Session Bean implementation class ServicioPersona
*/
@Stateless
@LocalBean
public class ServicioPersona {
@PersistenceContext
private EntityManager em;

@TransactionAttribute(TransactionAttributeType.REQUIRED)
public void insertar(Persona personaA,Persona personaB) {

em.persist(personaA);
em.persist(personaB);
```

```
}

}
```

Como se puede ver hemos definido un único método que se ejecuta de forma transaccional utilizando la anotación @TransactionAttribute que delega en JTA . En nuestro caso hemos construido un método que inserta dos Personas. Al ser el nombre la clave primaria de nuestra tabla si insertamos dos personas idénticas se producirá un rollback y ninguna de las personas será salvada en la base de datos.Invocaremos el EJB a través de un Servlet:

```
package com.arquitecturajava;

import java.io.IOException;
import java.io.PrintWriter;

import javax.ejb.EJB;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.arquitecturajava.bo.Persona;
import com.arquitecturajava.ejb.ServicioPersona;

&amp;amp;nbsp;
@WebServlet("/ServletTransaccion")
public class ServletTransaccion extends HttpServlet {
private static final long serialVersionUID = 1L;
```

```
@EJB
ServicioPersona miservicio;

protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

Persona a= new Persona();
a.setNombre("ana");
Persona b= new Persona();
b.setNombre("pedro");
miservicio.insertar(a, b);
PrintWriter pw= response.getWriter();
pw.println("insertado");

}
}
```

Como podemos ver usar JTA con nuestros EJBs es realmente sencillo.

Otros artículos relacionados:

Introducción a EJB

EJB Remotos

EJB Cliente