Análisis GWT para desarrollo visual en web con Java EE.

Sebastián Echarte, Dario Secualino, [1]Alejandro R. Sartorio,
[1] Centro de Altos Estudios en Tecnología Informática, Sede Rosario,
Universidad Abierta Interamericana, Ov. Lagos 944, 2000 Rosario, Argentina
secharte@gmail.com
sartorio@cifasis-conicet.gov.ar
dsecualino@telecom.com.ar

***Resumen*ð Esta investigación está enfocada en resolver problemas comunes de desarrollo y centrarse en que el desarrollador Java solo tenga que resolver problemáticas de manejo de clases y no tener que lidiar con la problemática de manejo de lenguajes como: HTML, JQuery, JSP, CSS, ActionScript, Flex, JavaScript etc. También resuelve el problema de la dependencia del Navegador, lo que significa que se debe adaptar el código JS, HTML, CSS, etc. para el intérprete de cada Navegador por ejemplo (FireFox, Internet Explorer x, etc.). El principal FrameWork utilizado para resolver esto es GWT propiedad de la comunidad de google, tambien existen otros como (Vaadin, JavaServer Faces (JSF)) que al ser de código libre diferentes empresas han modificado para ofrecer más componentes visuales en este caso se usara SmartGWT el cual se llevo a la práctica. En el caso de aplicación que consiste en la utilización de SmartGWT y diferentes FrameWork ós para resolver problemáticas de un proyecto a gran escala y para la implementación fueron incorporando diferentes Patrones de diseño los cuales serán explicados en detalle y con ejemplos prácticos. Este caso es una aplicación de Liquidación de sueldos llamado, ñbssyjò, el entorno para desarrollar es Eclipse-Galileo, Base de Datos MySQL 5 y Servidor de aplicaciones Tomcat 6, será explicado los Plug-In utilizados y los motivos por los cuales fueron usados.**

I. Introducción

La construcción de cualquier aplicación a gran escala tiene sus inconvenientes, y las aplicaciones (Google Web Toolkit) GWT no son una excepción. Que varios programadores trabajen simultáneamente en la misma base de código, mientras tanto el mantenimiento de las la funcionalidad, puede convertirse rápidamente en código desordenado, para evitar eso se platea una arquitectura adecuada. Hay varios patrones de diseño, Presentation-Abstraction-Control, modelo-vista-controlador, modelo-vista-presentador, etc. Y si bien cada modelo tiene sus ventajas, hemos encontrado que un modelo-vista-presentador (MVP) la arquitectura que funciona mejor para el desarrollo de aplicaciones GWT teniendo en cuenta las siguientes consideraciones. En primer lugar el modelo de MVP, al igual que otros patrones de diseño, desacopla de una manera que permite a varios desarrolladores trabajar simultáneamente. En segundo lugar, este modelo nos permite minimizar el uso de GWTTestCase (herramienta de generación de test de unidad sobre web browser o sobre clases java), que se basa en la presencia de un navegador.

Este patrón se basa es la separación de la funcionalidad en componentes que lógicamente tienen sentido, pero en el caso de GWT hay un propósito claro en hacer la vista tan simple como sea posible para minimizar nuestra dependencia de GWTTestCase y reducir el tiempo total de las pruebas de funcionamiento.

La creación de una aplicación basada en MVP puede ser sencilla conociendo los fundamentos detrás de este modelo de diseño.

A. Fundamentos MVP explicados brevemente:

1) Modelo

Un modelo incluye objetos de negocio, y en el caso de nuestra aplicación de contactos que tenemos:

2) Vista

Un punto de vista contiene todos los componentes de interfaz de usuario que componen nuestra aplicación. Esto incluye todas las tablas, etiquetas, botones, cuadros de texto, etc .. Las vistas son responsables de la distribución de los componentes de interfaz de usuario y no tienen conocimiento del modelo. Es decir, no sabe que está mostrando un objeto, simplemente sabe que tiene controles para usuarios, Labels (etiqueras), combo-box, que son controles de un Form de HTML.

3) Presentador

Un presentador contiene toda la lógica de nuestra aplicación de ejemplo, la sincronización de transición y datos a través de (Remote Procedure Call Llamada a Procedimiento Remoto) RPC de vuelta al servidor. Como regla general, por cada vista es necesario un presentador para conducir la vista y manejar los eventos que provienen de los controles de la interfaz de usuario dentro de la vista.

4) AppController

Para hacer frente a la lógica que no es específico de ninguna presentadora y en su lugar reside en la capa de aplicación, vamos a introducir el componente AppController. Este componente contiene la gestión de la del history del browser y la lógica de transición vista que hizo antes y después está directamente vinculada a la gestión del history del browser.

II. Requerimientos para correcta implementacion

Al momento de implementar una solución informática utilizando estas tecnologías, se va a describir para qué tipos de sistemas es posible aplicar el Framework GWT y los patrones descriptos.

A. *Tipo de sistemas de información*

Sistema de procesamiento de transacciones (TPS), tales como el control de inventarios, control de activos fijos o la nómina de sueldos o salarios, explotan poco las posibilidades de las máquinas y el software actual, es un tipo de sistema de información que recolecta, almacena, modifica y recupera toda la información generada por las transacciones producidas en una organización. Una transacción es un evento que genera o modifica los datos que se encuentran eventualmente almacenados en un sistema de información. Desde un punto de vista técnico, un TPS monitoriza los programas transaccionales (un tipo especial de programas). La base de un programa transaccional está en que gestiona los datos de forma que estos deben ser siempre consistentes (por ejemplo, si se realiza un pago con una tarjeta electrónica, la cantidad de dinero de la cuenta sobre la que realiza el cargo debe disminuir en la misma cantidad que la cuenta que recibe el pago, de no ser así, ninguna de las dos cuentas se modificará), si durante el transcurso de una transacción ocurriese algún error, el TPS debe poder deshacer las operaciones realizadas hasta ese instante. Otra función de los monitores de transacciones es la detección y resolución de interbloqueos (deadlock), y cortar transacciones para recuperar el sistema en caso de fallos masivos.

Los sistemas de planificación de recursos empresariales, o ERP (por sus siglas en inglés, Enterprise Resource Planning) son sistemas de información gerenciales que integran y manejan muchos de los negocios asociados con las operaciones de producción y de los aspectos de distribución de una compañía en la producción de bienes o servicios. La Planificación de Recursos Empresariales es un término derivado de la Planificación de Recursos de Manufactura (MRPII) y seguido de la Planificación de Requerimientos de Material (MRP); sin embargo los ERP han evolucionado hacia modelos de subscripción por el uso del servicio (SaaS, cloud computing) lo que se denomina Sistema Integral de Operación Empresarial (EOS) por sus siglas en inglés Enterprise Operating System.

Software contable, son los programas de contabilidad o paquetes contables, destinados a sistematizar y simplificar las tareas de contabilidad.

B. *Requerimientos no funcionales*

Java EE versión 1.6 o superior

Sistemas Web, Arquitectura Cliente-servidor.

Un motor de base de datos.

Un servidor WEB como, tomcat, webLogic, WebSphere, jBoss etc.

Eclipse ultima versión estable, como (entorno de desarrollo integrado, sigla en inglés de integrated development environment) IDE.

III. ARQUITECTURA UTILIZADA

A continuación veremos la arquitectura utilizada y su relación entre el Framework GWT y los patrones implementados.

Aquí mostraremos los motivos de su utilización y en otra sección como llevarlos a cabo con un ejemplo práctico.

A. *Introducción a GWT*

Google Web Toolkit es un kit diseñado con la finalidad de crear páginas web dinámicas con mucha funcionalidad AJAX. GWT es un kit de desarrollo de software de google que compila/traduce código java en código javascript. De esta manera se permite a los desarrolladores implementar la parte cliente de su aplicación utilizando cualquier IDE para Java abstrayendo al desarrollador del JavaScript necesario para implementar dichas funcionalidades. El compilador de GWT (pieza central del kit de desarrollo GWT) permite al programador aislarse de los detalles y características propias de los navegadores web, haciendo posible un desarrollo para cualquier tipo de navegador del mercado (Explorer, Firefox, Opera, Chrome, ...).

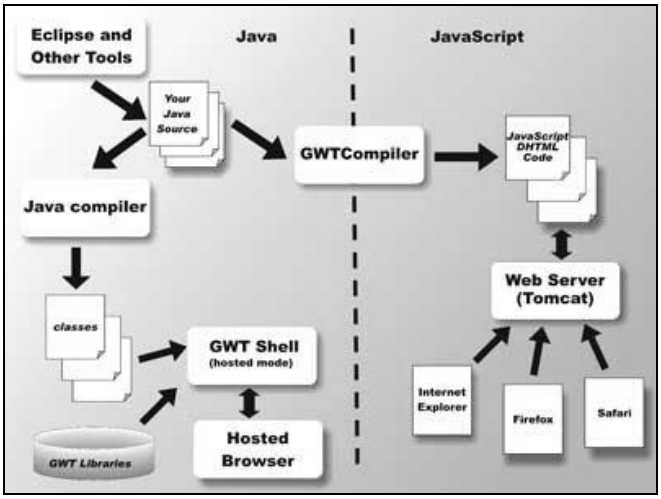


Fig. 1: Diagrama de GWT estándar.

B. *Integración MVP y GWT*

MVP se describió en la sección de Introducción, ahora veremos como se integran el patrón con GWT.

En la Fig. 2 podemos ver de izquierda a derecha Vista UI Binder (significa que se acoplan lo objetos de tipo UI con los eventos propios a usar) que va a ser la vista que se ejecuta solo en modo de compilación, con UiBinder , también tenemos el Template donde se definen las Vistas implícitamente como: Componentes UI, html, Labels, Text Box etc. se disponen declarativo usando XML.

Tenemos el Presentador que va a manejar los eventos que se utilizan de las vistas los que son ñBindeadosò a través de Display Interfase el Presentador los define y la Vista lo implementa, tambien el Presentador es el encagado de actualizar los datos de la vista ya se dato como eventos, por ejemplo cambiar el tamaño de una venta, llenar dartos a una grilla, etc.

El ñEvent Busò es un mecanismo para: pasar eventos y registrar eventos que serán utilizados a través de toda la aplicación, como por ejemplo un evento de un menu que habra un ventana este evento no esta ligado a ninguna vista.

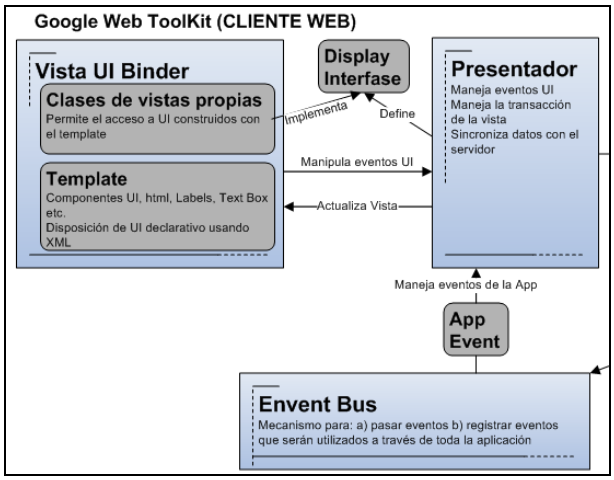


Fig. 2: Integración MVP y GWT parte 1 de 2

En la Fig. 3 veremos la otra parte del diagrama que va a la derecha de la Fig. 2 veremos RPC que es propio de GWT que maneja todos las llamadas GWT hacia servicio remotos. Tambien el ñRemote GWT Serviceò que convierte entidades Hibernate hacia el modelo GWT a través de DTOs y viceversa, convierte acciones a llamadas de servicios. ñCallback Handlerò que evalúa respuesta agrega los eventos mas relevantes o las excepciones al Event Bus. El Modelo que en este caso va a contener la representación de objetos de negocio (beans) como DTO y un representación de servicios livianos para el lado del Cliente.

Cabe destacar la division entre cliente y servidot que se define de la siguiente manera, lo que esta a la izaquierda del icono es ñGoogle Web ToolKit (CLIENTE WEB)ò y lo que esta a la derecha es òJava (SERVER)ò. Tenemos el ñRemote GWT Serviceò que convierte entidades Hibernate hacia el modelo GWT a través de DTOs y viceversa y convierte acciones a llamadas de servicios.

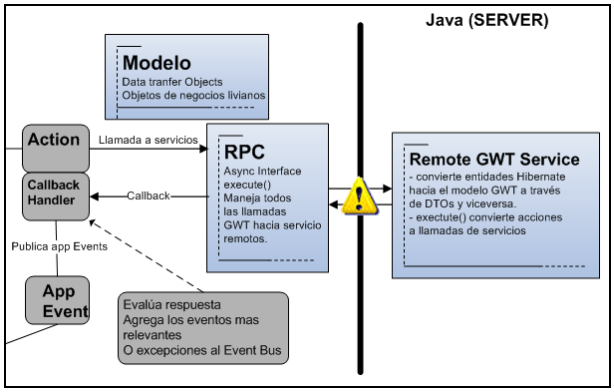


Fig 3 Integración MVP y GWT parte 2 de 2

En la Fig 4 tenemos una noción general de cómo interactúa el modelo MVP y en cada capa del modelo MVP lo patrones que se aplicaron, cabe destacar algunos puntos particulares como ñconexión asíncronaò que se refiere a que solo se conectan en tiempo de ejecución sin haber una relación implícita en el código, debido a esta separación se decide aplicar DTO para separar por completo los toda conexión. La conexión vía Ajax se hace implícitamente configurando correctamente GWT sin necesidad de crearlos en la vista, únicamente manipulando eventos es suficiente.

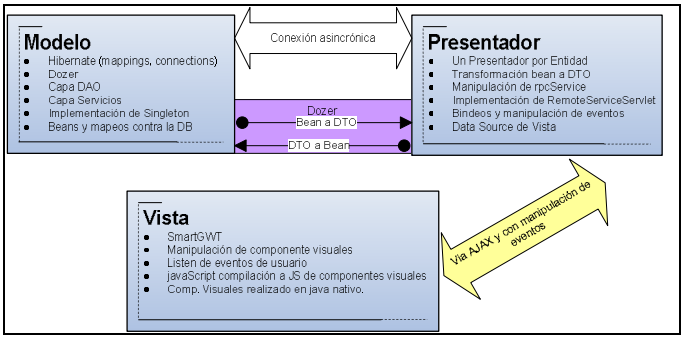


Fig 4: Integracion MVP y Patrones

IV. Implementación de Patrones.

Antes de empezar vamos a dar unas definiciones del sistema con el cual vamos a aplicar el los patrones y la implementación.

Del ejemplo práctico (BSSYJ) se describe lo siguiente:

A. *Alcance del sistema:*

El sistema a construir, tiene por objetivo, aportar soluciones a la problemática de la gestión de liquidación de sueldos jornales de empresas de todo tipo de envergadura.

Este sistema de gestión on-line apuntará a pequeñas y medianas empresas que tengan grandes dotaciones de personal, también será totalmente apto para Estudios Contables.

Se utilizará la computación en nube es un concepto el que incorporamos al software como servicio.

Es decir todo lo que ofrece nuestro sistema se ofrece como servicio, de modo que los usuarios puedan acceder a estos servicios disponibles "en la nube de Internet" sin conocimientos (o, al menos sin ser expertos) en la gestión de los recursos que usan.

La idea es proveen aplicaciones comunes de negocio en línea accesibles desde un navegador web, mientras el software y los datos se almacenan en los servidores.

B. *Descripción sucinta del negocio.*

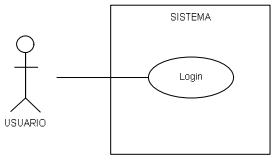
Con el Sistema Bssyj se podrá lograr la unificación y máxima operatividad concerniente a la liquidación de Nominas.

Permitirá a sus usuarios acceder a un servicio integral vía web, que centraliza la información de empleados con el objeto de llevar a cabo la liquidación de nominas, quincenales, jornales y/o mensuales de los empleados.

Además podrán ofrecerles a los clientes la posibilidad de acceder en cualquier instante, y obtener los Reportes necesarios para el proceso de liquidación.

C. *Caso de uso: Login*

Diagrama de caso de uso:

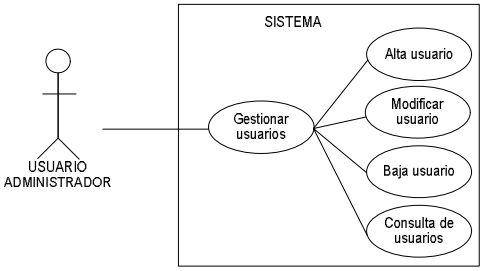


Especificaciones de caso de uso:

ID	01	Nombre	Login
Descripción	Permitirá a los distintos Actores del Sistema loguearse al mismo. Es decir, establecer sus credenciales alimentadas desde la base de datos del sistema.		
Actores	Usuario		
Precondición	El actor cuenta con un juego usuario y contraseña, estando registrado en el Sistema.		
Post-condición	El actor establecerá al Sistema claramente su ID, Rol, Credenciales, y quedará identificado para el Sistema.		
Validaciones	El sistema validará que el juego de nombre de usuario y contraseña coincida con los almacenados en la base de datos.		
Flujo Normal			
1) El Sistema muestra un formulario de ingreso de Usuario y Clave. 2) El Usuario ingresa su nombre de usuario y clave. 3) El Sistema valida acorde a Validaciones, y verifica que el Actor exista en el sistema. 4) El Sistema realiza la consulta y le otorga las credenciales correspondientes.			
Extensiones			
3.1) El sistema informa que el Usuario/Clave no corresponden a un Actor registrado. 4.1) Si el actor es un Administrador el sistema le permitirá ingresar a las opciones del backend.			

D. Caso de uso: Alta Usuario

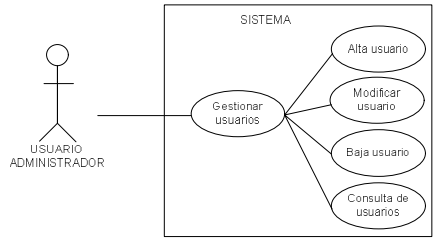
Diagrama de caso de uso:



Especificaciones de caso de uso:

ID	02	Nombre	Alta de Usuario
Descripción	Permitirá a los distintos Usuarios del Sistema registrarse al mismo.		
Actores	Usuario		
Precondición	El Usuario debe haber ingresado al aplicativo web.		
Postcondición	El Usuario quedará registrado en el sistema.		
Validaciones	Todos los campos deben estar completos. Las dos contraseñas ingresadas deben coincidir.		
Flujo Normal			
1) El Sistema muestra un formulario de ingreso de datos personales. 2) El Usuario ingresa sus datos personales: nombre de usuario, clave, clave 2, domicilio, Email. 3) El Sistema valida acorde a Validaciones, y verifica que el Usuario no exista en el sistema. 4) El Sistema da de alta al nuevo Usuario. 5) El Sistema muestra un mensaje de confirmación al usuario.			
Extensiones			
3.1) El sistema informa que el Usuario ya existe. 3.2) El sistema informa que algún campo está incompleto o incorrecto.			

Caso de uso: Baja Usuario



Especificaciones de caso de uso:

ID	03	Nombre	Baja Usuario
Descripción	Permitirá al Administrador del sistema dar de baja a usuarios registrados.		
Actores	Usuario Administrador.		
Precondición	El administrador debe haberse logueado en el sistema.		
Post-condición	Se dará de baja al Usuario seleccionado.		
Validaciones	El usuario no debe haber sido utilizado en otro registro del sistema.		
Flujo Normal			
1) El sistema muestra una lista de opciones 2) El actor selecciona ñusuariosò. 3) El sistema muestra una lista de todos los usuarios almacenados. 4) El actor selecciona el usuario que desea eliminar. 5) El Sistema muestra un formulario de ingreso de datos personales. 6) El Usuario selecciona ñBorrarò. 7) El Sistema valida acorde a Validaciones. 8) El Sistema elimina al usuario seleccionado.			

Extensiones
7.1) El sistema muestra un mensaje de error porque el usuario se encuentra Inactivo.

E. Implementación de Hibernate y explicación de mapeo con esta Base de Datos de ejemplo y paquete de beans.

Hibernate es un es una Application Programming Interface (API es el conjunto de funciones y métodos, que ofrece cierta biblioteca para ser utilizado por otro software) de tipo Object Relational mapping (ORM) para Java.

1) Porque uso Hibernate, en este proyecto:

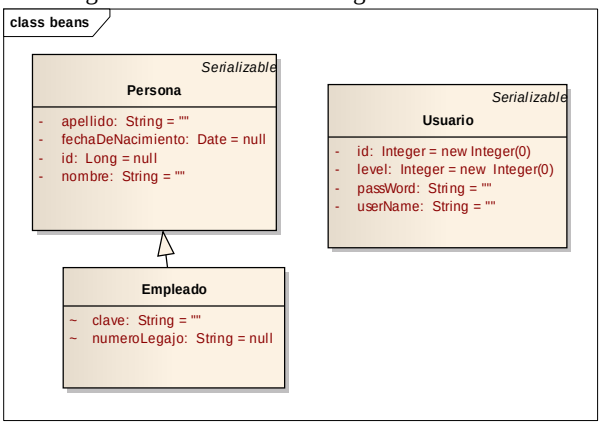
Modelo natural de Programación, Persistencia Transparente, Alta Performance, lazy, Auto cache y Query cache, Fiabilidad y escalabilidad, Transactions, auto rollback y sessions [3].

2) Generación automática de Base de datos (esquema) basándose en el esquema de Beans.

Para que se pueda realizar esto Hibernate permite usar una conjunto de herramientas llamadas: ñhbm2ddlò, con lo cual se necesita crear un XML que ejecuta y genera en base a la configuración de mapeos de Hibernate, la base de datos correspondiente con el diagrama de clases de la aplicación y genera el Lenguaje de definición de datos (DDL) correspondiente.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="Bssyj" default="schemaexport" basedir=".">
      <target name="schemaexport">
            <taskdef name="schemaexport"
            classname="org.hibernate.tool.hbm2ddl.SchemaExportTask"/>
            <schemaexport
properties="resources/conf/hibernate/hibernate.properties"
              quiet="no"
            text="yes"
            drop="NO"
              create="YES"
            delimiter=";"
              output="resources/sql/esquema-1.sql">
              <fileset dir="resources/dao/hibernate">
                    <include name="**/*.hbm.xml" />
              </fileset>
          </schemaexport>
    </target>
</project>
```

El diagrama de Entidades de Negocio será:



El archivo de sentencias DDL que genera para nuestro caso será:

```
create table Persona (
    ID bigint not null auto_increment,
    DISCRIMINADOR varchar(2) not null,
    NOMBRE varchar(150),
    APELLIDO varchar(150),
    FECHA_NACIMIENTO date,
    EM_CLAVE varchar(150),
    EM_NRO_LEGAJO varchar(150),
    primary key (ID)
);

create table usuario (
    ID integer not null auto_increment,
    USER_NAME varchar(50),
    PASSWORD varchar(30),
    LEVEL integer,
    primary key (ID)
);
```

F. Capa DAO

Se encuentra en el Modelo del patrón de diseño MVP

1) Contexto

El acceso a la información varía dependiendo de la fuente de los datos. El acceso al almacenamiento persistente, como una base de datos, varía en gran medida dependiendo del tipo de almacenamiento (bases de datos relacionales, bases de datos orientadas a objetos, ficheros planos, etc.) y de la implementación del vendedor.

2) Solución

Utilizar un Data Access Object (DAO) para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.

El DAO implementa el mecanismo de acceso requerido para trabajar con la fuente de datos. Esta fuente de datos puede ser un almacenamiento persistente como una RDMBS, un servicio externo como un intercambio B2B, un repositorio LDAP, o un servicio de negocios al que se accede mediante CORBA Internet Inter-ORB.

3) Resumen DAO

a) Abstraer y encapsular todos los accesos a la fuente de datos.

b) El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.

c) El DAO oculta completamente los detalles de implementación de la fuente de datos a sus clientes.

2) Ventajas de capa DAO

a) Permite la Transpariencia

b) Permite una Migración más Fácil

c) Reduce la Complejidad del Código de los Objetos de Negocio

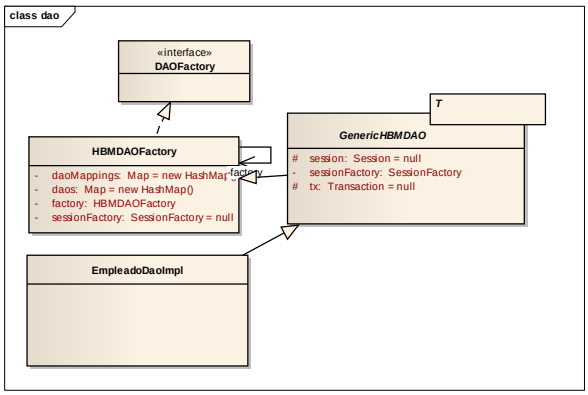
d) Centraliza Todos los Accesos a Datos en un Capa Independiente

4) Desventajas

a) Añade una Capa Extra

b) Necesita Diseñar un Árbol de Clases

Diagrama de clases de la aplicación de ejemplo Para DAO:



Ejemplos de clase genérica (Create Update delete) Crud, en esta clase GenericHBMDAO<T> , podemos ver que indicándole un bean en T como parte de generics de Java, todas las clases que heredan de esta clase abstracta pueden usar metodos comunes como add, delete, detall etc.

```
/**
 * Clase con operaciones basicas de CURD o ABM para ser extendido
 * implementaciones DAO, incluye Singletong y sinconized
 *
 * @author Sebastian Echarte Base DAO class.
 */
public abstract class GenericHBMDAO<T>  extends HBMDAOFactory {

    protected GenericHBMDAO(SessionFactory aFactory) {
        super(aFactory);
    }

    protected Session session = null;
    protected Transaction tx = null;
```

Los métodos genéricos Crud están implementados de la siguiente forma:

```
protected void saveOrUpdate(Object obj) {

    try {
        session = getSession();
        tx = session.beginTransaction();
        session.saveOrUpdate(obj);
        tx.commit();
    } catch (HibernateException e) {
        handleException(e);
    } finally {
        closeSession(session);
    }
}

protected void delete(Object obj) {
    try {
        startOperation();
        session.delete(obj);
        tx.commit();
    } catch (HibernateException e) {
        handleException(e);
    } finally {
        closeSession(session);
    }
}
```

Y también mas métodos genéricos:

```
@SuppressWarnings("unchecked")
protected Object find(Class clazz, Long id) {
    Object obj = null;
    try {
        startOperation();
        obj = session.load(clazz, id);
        tx.commit();
    } catch (HibernateException e) {
        handleException(e);
    } finally {
        closeSession(session);
    }
    return obj;
}

@SuppressWarnings("unchecked")
protected List<T> getAll(Class clazz) {
    List<T> result = null;
    startOperation();
    try {

        Query query = session.createQuery("from " + clazz.getName());
        result = query.list();
        tx.commit();

    } catch (HibernateException e) {

        handleException(e);
    } finally {
        closeSession(session);
    }
```

Ahora vamos a usar la clase genérica de la siguiente manera, vemos que acá que implementamos el DAO en base a el bean Empleado que tiene métodos propios como getBy(Sring, String) con hibérnate

```
public class EmpleadoDaoImpl extends GenericHBMDAO<Empleado>  {

    protected EmpleadoDaoImpl(SessionFactory aFactory) {
        super(aFactory);

    }
    public final Empleado getBy(final String name, final String pass)
    {
        Empleado result = null;
        Session session = null;
        try {
            session = getSession();
            Criteria criteria = session.createCriteria(Empleado.class);
            criteria.add(Restrictions.and(Restrictions.eq("nombre", name), Restrictions.eq("clave", pass))
                    );
            result = (Empleado) criteria.uniqueResult();

        } catch (HibernateException e) {

            System.err.println("getbyName EmpleadoDaoImpl: " + e);
        } finally {
            closeSession(session);
        }
        return result;
    }
```

Como utilizo los métodos genéricos, aquí va un ejemplo esto es dentro de EmpleadoDaoImpl

```
public List<Empleado> getAllEmpleados() {
    return getAll(Empleado.class);
}
public Empleado createOrUpdate(Empleado emp) {
    saveOrUpdate(emp);
    return emp ;
}
```

G. Patrón Singleton y Factory.

Se encuentran en la parte de la capa M:Modelo del patrón de diseño MVP

1) Definición Singleton

El patrón de diseño singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

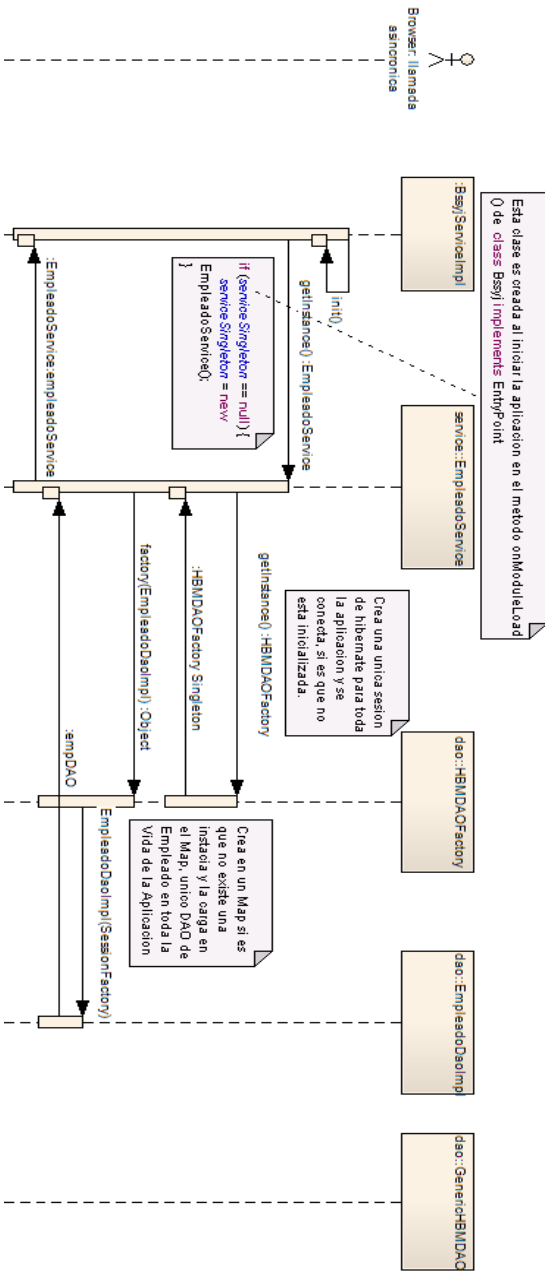
2) Definición de Factoría

La idea que se esconde detrás de este patrón es la de centralizar el sitio donde se crean los objetos, normalmente donde se crean objetos de una misma "familia", sin dar una definición clara de lo que nuestro software puede entender como familia, como podría ser componentes visuales, componentes de la lógica del negocio, o objetos concurrentes en el tiempo.

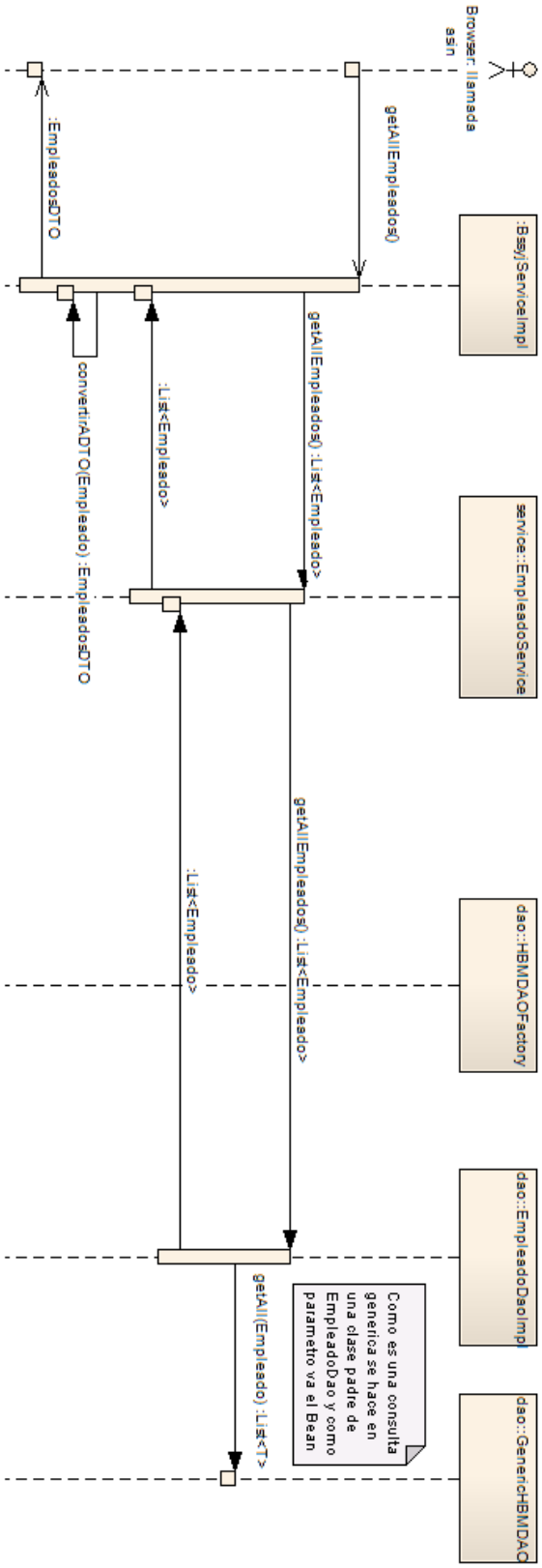
3) Implementación en el Ejemplo bssyj

Las clases de Servicios y DAO son Singleton e inicializadas por la Factoría todo guardado en un contenedor de objetos llamado HashMap único para toda la vida de la aplicación en el servidor.

Veremos un Diagrama de secuencia con el ejemplo de obtener todos los empleados pasando por las N capas, primero vemos que al iniciar la aplicación sucede los siguiente:



Luego una vez que las instancias estén en la memoria de la aplicación solo queda, obtener todos lo empleado con el método getAllEmpleados():



Veremos a continuacion como las variables EmpleadoService serviceSingleton = null y EmpleadoDaoImpl empDAO, se mantienen Singleton tan solo manejando una Factoría de instancias que solo averiguan si el objeto ya fue creado anteriormente y si es así lo usa. El método getInstance() de la clase HBMDAOFactory lo hace a nivel de interface con solo hacer la Factoría ya devuelve el árbol de herencia de HBMDAOFactory como ser EmpleadoDAOImp.

```
public class EmpleadoService {
	private List<Empleado> empLoged = new ArrayList<Empleado>();
	/**
	 * The service singleton
	 */
	private static EmpleadoService serviceSingleton = null;
	/**
	 * DAO to access the Institution
	 */
	private EmpleadoDaoImpl empDAO = null;

. . .

/**
	 * Gets a DAO Factory according to the configuration.
	 *
	 * @param msc
	 *            the configuration object.
	 * @return an object that implements DAOFactory interface.
	 * @throws Exception
	 *             an exception occurred while executing this method.
	 */
	private static HBMDAOFactory getDAOFactory() throws Exception {
		HBMDAOFactory df = HBMDAOFactory.getInstance();
		return df;
	}

	/**
	 *
	 * @param rscs
	 * @param conf
	 * @param ftpDest
	 * @return
	 * @throws Exception
	 */
	public static EmpleadoService getInstance() throws Exception {
		if (serviceSingleton == null) {
			serviceSingleton = new EmpleadoService();
		}
		HBMDAOFactory df = getDAOFactory();
		serviceSingleton.empDAO = (EmpleadoDaoImpl)
df.factory(EmpleadoDaoImpl.class);
		return serviceSingleton;
	}
```

Como veremos a continuación la factoría implementada para HBMDAO que configura para conectar las tablas de la base con Hibernate, creando una única conexión y guardándola en un variable a reutilizar que se llama factory, acá vemos como se une DAO, Hibernate, Singleton y Factoria.

```
/**
 * Factory of DAOs.
 * @author Sebastian Echarte
 */

public  class HBMDAOFactory implements DAOFactory {

...

/**
	 * Builds and returns a daoFactory for the requested configuration.
	 * @param hbmCfgFileName
	 * the configuration file.
	 * @return a dao factory.
	 * @throws Exception
	 * an exception occurred.
	 */
	public static HBMDAOFactory getInstance() throws Exception {

		if (factory == null) {
			try {

				Configuration cfg = new Configuration();
				SessionFactory ssfactory = null;
				ssfactory =
cfg.configure("/conf/hibernate/hibernate.cfg.xml")
						.buildSessionFactory();
				factory = new HBMDAOFactory(ssfactory);

			} catch (Throwable ex) {

				// Make sure you log the exception, as it might be
swallowed
				System.err.println("Initial SessionFactory creation
failed."
						+ ex);
				throw new ExceptionInInitializerError(ex);
			}

		}
		return factory;
```

Acá vemos como se guarda y se mantiene un conjunto de DAOS en la variable (no global) daoMappings, de nuevo combinamos todo, DAO, Singleton y factoría.

```
public  class HBMDAOFactory implements DAOFactory {

      /**
       * stores the factory singleton.
       */
      private static HBMDAOFactory factory;
      /**
       * The session factory.
       */
      private SessionFactory sessionFactory = null;
      /**
       * The interface - implementation definition container.
       */
      @SuppressWarnings("unchecked")
      private Map daoMappings = new HashMap();

. . .

/**
       * {@inheritDoc}
       */
      @SuppressWarnings("unchecked")
      public Object factory(final Class daoInterface) throws Exception {

            GenericHBMDAO dao = (GenericHBMDAO)
daos.get(daoInterface.getName());

            if (dao == null) {
                  try {
                        Class daoClass = (Class) daoMappings
                                    .get(daoInterface.getName());
                        dao = (GenericHBMDAO) daoClass.newInstance();
                        dao.setSessionFactory(sessionFactory);
                        daos.put(daoClass.getName(), dao);

                  } catch (Exception e) {
                        System.err.println("Initial Factory." + e);
                        throw new Exception(e);
                  }

            }

            return dao;

      }
```

H. Implementación de estrategia de DTO.

Es para poder resolver la problemática de Cliente Servidor de GWT con DTO Dozer (API para poder transformar un objeto de negocio en su similar de DTO, configurable para soportar diferentes tipos de objetos de bean a beanDTO)

1) Motivo

Se debe utilizar objetos de datos de transferencia (DTO), porque DTOs sólo pueden ser transferidos a otro la capa de cliente de GWT.

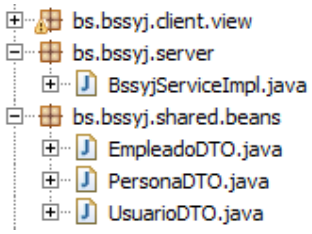
En este ejemplo se crean tantos DTO como beans existan y después de recibir el objeto de Hibernate que tenía para convertirlo en EntidadDto.

En el lado de cliente que va a operar sólo con DTOs. Si desea guardar un bean con Hibernate es necesario convertir de DTO a bean.

Para convertir a DTOs Beans se usa dozer en el ejemplo bssyj.

Para utilizar el dozer es necesario asignar DTOs beans y con las asignaciones de dozer. O usar las asignaciones personalizadas a través de archivos XML dozer.

Vamos a ver un ejemplo sencillo de transformación de bean a DTO y viceversa, se establecen del lado del servidor y los dto en un paquete de objetos simples que pueden estar en el medio y ser accedido tanto por el lado del Cliente como del Servidor.



Como vemos en este caso utilizo una instancia de DozerBeanMapper (que viene con la Api d Dozer), para pasar de bean a DTO dozer automáticamente va a comparar las variables de y si coincide el nombre y la firma asume que son los mismos datos.

En los casos de los métodos saveEmpleado() y getAllEmpleados() los pasos son similares se invoca al servicio real que va contra hibernate a través de los DAO, como se menciono anteriormente obtengo la instancia del bean y el caso siguiente es invocar a dozer para que se encargue de trasformarlo automáticamente.

```
public class BssyjServiceImpl extends RemoteServiceServlet implements
            BssyjService {

      private static final long serialVersionUID = 5336553158893222 6L;
      private EmpleadoService empleadoService = null;
      private Mapper mapper = new DozerBeanMapper();
...

@Override
      public EmpleadoDTO saveEmpleado(EmpleadoDTO emp) {
            Empleado ret = new Empleado();
            ret = mapper.map(emp, Empleado.class);
            ret = empleadoService.saveEmpleado(ret);
            return mapper.map(ret, EmpleadoDTO.class);
      }

      @Override
      public List<EmpleadoDTO> getAllEmpleados() {
            List<Empleado> real = new  ArrayList<Empleado>();
            List<EmpleadoDTO> ret = new  ArrayList<EmpleadoDTO>();
            //TODO realizar el maping via XML
            real = empleadoService.getAllEmpleados();
            for (Empleado empleado : real) {
                  ret.add(mapper.map(empleado, EmpleadoDTO.class));
            }

            return ret ;
      }
```

V. Implementación de ejemplo BSSYJ

A. Proceso de arranque de GWT llamadas onModuleLoad().

1) onModuleLoad() crea el servicio RPC, bus de eventos, y AppController del

2) El AppController se pasa la instancia RootPanel y se hace cargo

3) A partir de entonces el AppController está en control de la creación de determinados presentadores y el suministro de un punto de vista de que el presentador va a conducir.

```
public class Bssyj implements EntryPoint {

      private BssyjServiceAsync rpcService;


      public void onModuleLoad() {
            BssyjConstants bssyjConstants =
GWT.create(BssyjConstants.class);
            BssyjMessajes bssyjMessajes = GWT.create(BssyjMessajes.class);
            BssyjResources bssyjResources =
GWT.create(BssyjResources.class);

            rpcService = GWT.create(BssyjService.class);

            HandlerManager eventBus = new HandlerManager(null);
            AppController appViewer = new AppController(rpcService,
eventBus,
                        bssyjConstants, bssyjMessajes,bssyjResources);
            appViewer.go(RootPanel.get());


      }
}
```

B. Binding presenters y Vista

Con el fin de obligar a la presentadora a asociar la vista vamos a depender de interfaces de pantalla que se definen en el presentador. Tomemos por ejemplo el EmpleadoView:

Este punto de vista tiene 3 widgets: una tabla y botones. Para que la aplicación para hacer algo significativo, el presentador se va a necesitar:

1) *Responder a clics en los botones*
2) *Llenar la lista*
3) *Responder a un usuario hacer clic en la lista*

En el caso de nuestro EmpleadoPresenter, se define la interfaz de visualización como por ejemplo:

```
public class EmpleadoPresenter extends BssyjPresenter {

…
private final HandlerManager eventBus ;
      private DisplayEmpleado displayEmpleado = null;
      private DisplayListEmpleado displayListEmpleado = null;
      private final BssyjServiceAsync rpcService ;
      private EmpleadoDTO empleado = null;
      public interface DisplayEmpleado {

            ButtonItem getOkButton();

            ButtonItem getCancelButton();

            ButtonItem getAgregarButton();
```

Mientras que el EmpleadoView implementa la interfaz EmpleadoPresenter.DisplayEmpleado, utilizando botones y el BssyjPresenter no es el más sabio. Además, si queremos ejecutar esta aplicación en un navegador móvil que podría cambiar los puntos de vista sin tener que cambiar ningún código de la aplicación. Para ser transparentes, el presentador está haciendo la suposición de que una vista va a mostrar los datos en la forma de una lista. Dicho esto, de otra manera, que una vista sea capaz de cambiar la implementación específica de la lista, sin efectos secundarios. Método SetData() es una forma sencilla de adicionar datos en la vista sin el conocimiento en detalle del propio modelo de datos. Los datos que se muestran están directamente ligados a la complejidad de nuestro modelo. Un modelo más complejo puede llevar a un mayor número de datos que se muestran en una vista. El motivo de la utilización de bindDTO() es que los cambios en el modelo se puede hacer sin actualizar el código de la vista como se ven en la próxima porción de código. Para mostrar cómo funciona esto, veamos el código que se ejecuta al recibir 1 empleado desde el servidor:

```
public class EmpleadoPresenter extends BssyjPresenter {

. . .

protected synchronized void setEmpleadoById(Long id) {

            rpcService.getEmpleadoById(id, new AsyncCallback<EmpleadoDTO>()
{
                  @Override
                  public void onSuccess(EmpleadoDTO result) {
                        bindDTO(result);
                  }

                  @Override
                  public void onFailure(Throwable caught) {
                        SC.warn(getBssyjMessajes().failure(
                                    "setEmpleadoById: in: " +
this.toString()
                                                + " ERROR: " +
caught.getMessage()));

                  }
            });

            // return empleado;

      }

}

…

public void bindDTO(EmpleadoDTO emp) {

            empleado = emp;

      displayEmpleado.getDatosPersonalesForm().setValue("nombreDSField",emp
leado.getNombre());

      displayEmpleado.getDatosPersonalesForm().setValue("apellidoDSField",e
mpleado.getApellido());
            //int r = Calendar.DATE;

      displayEmpleado.getDatosPersonalesForm().setValue("dobDSField",emplea
do.getFechaDeNacimientoS());

      displayEmpleado.getDatosPersonalesForm().setValue("legajoDSField",
Double.parseDouble(empleado.getNumeroLegajo()));

      displayEmpleado.getDatosPersonalesForm().setValue("passDSField",emple
ado.getClave());

      }
```

Para capturar/escuchar los eventos de la interfaz de usuario tenemos el siguiente código en la presentadora:

```
public class EmpleadoPresenter extends BssyjPresenter {

…

      private void bindEmpleado() {
            this.displayEmpleado.getOkButton().addClickHandler(new
ClickHandler() {
                  @Override
                  public void onClick(ClickEvent event) {
                        doOk();
                  }
            });
            this.displayEmpleado.getCancelButton().addClickHandler(new
ClickHandler() {
                  @Override
                  public void onClick(ClickEvent event) {
                        displayEmpleado.getEmpleadoWindow().destroy();
                  }
            });
            this.displayEmpleado.getAgregarButton().addClickHandler(new
ClickHandler() {

                  @Override
                  public void onClick(ClickEvent event) {
                        doAdd();

                  }
            });
      }
```

C. *Eventos y el bus de eventos*

Una vez que tenga los presentadores que se acoplan con los eventos de la vista, tendrá que tomar alguna acción sobre estos eventos. Para ello, deberá contar con un bus de eventos que se construye heredamdo de la clase de GWT ñHandlerManagerò. El bus de eventos es un mecanismo para que una serie de eventos que traspasen y registren información de algún subconjunto de estos eventos es un administrador de eventos generales de la aplicación.

Es importante tener en cuenta que no todos los eventos deben ser colocados en el bus de eventos. Si todos los eventos existentes dentro de una aplicación son incluidos en el bus de eventos, puede llevar a el dumping de memoria y perderse el manejo de eventos.

Los eventos de alto nivel son realmente los únicos eventos que desean que se pasan por del bus de eventos. Una aplicación no está interesada en eventos como "el usuario hace clic en enter" o "un RPC está a punto de ejecutarse". En su lugar (al menos en nuestra aplicación de ejemplo), se pasa en torno a acontecimientos tales como un empleado en proceso de actualización, el usuario de cambiar a la vista de edición, o el RPC notifica que se elimina un usuario, ha devuelto desde el servidor.

A continuación se muestra una lista de los eventos que hemos definido para el bus.

1) *LoadMenuEvent*
2) *LoadMenuEventHandler*
3) *LoginEvent*
4) *LoginEventHandler*

Para demostrar cómo encajan los eventos del bus, veamos lo que ocurre cuando un usuario quiere logearse. En primer lugar vamos a necesitar el AppController para inscribirse en el LoginEvent. Para ello, hacemos un llamado HandlerManager.addHandler() y pasarle el GwtEvent.Type así como el controlador que se debe llamar cuando se activa el evento. El código siguiente muestra cómo el AppController registra para recibir LoginEvents.

```
public class AppController implements Presenter, ValueChangeHandler<String>
{

eventBus.addHandler(LoginEvent.TYPE, new LoginEventHandler() {
                @Override
                public void onLogin(LoginEvent event) {
                      doLogin();
                }
          });
```

Aquí el AppController tiene una instancia del HandlerManager , llamado eventBus, y se está registrando una nueva LoginEventHandler. Este controlador pasar a ejcutar el método privado doLogin() método cada vez que un evento de LoginEvent.getAssociatedType() se dispara. Varios componentes pueden ser escuchados por un solo evento, por lo que cuando se dispara un evento con el HandlerManager.fireEven(), el HandlerManager busca cualquier componente que se le haya añadido un controlador para event.getAssociatedType(). Para cada componente que tiene un controlador, el HandlerManager llama event.dispatch() con la interfaz de EventHandler de ese componente.

Para ver cómo se dispara un evento, vamos a analizar al código fuente del LoginEvent. Como se mencionó anteriormente, hemos añadido como un controlador, haga clic en la lista de LoginView. Ahora, cuando un usuario hace clic en login, vamos a notificar al resto de la aplicación mediante una llamada al HandlerManager.fireEvent() con un LoginEvent() clase que se inicia con la identificación de los login.

VI. Detalles técnicos

A. Plug instalado utilizados para Eclipse Galileo:

1) Google Web Toolkit
2) GWT Designer
3) Schema export (ant con hibernate)
4) FileSync (Sincronización de archivos para deployar)

B. FrameWorks y herramientas utilizados:

1) Tomcat 6.0 como server de aplicaciones.
2) Hibernate 3
3) GWT
4) SmartGWT(http://www.smartclient.com/smartgwt/showcase/)
5) MySQL 5
6) 2 proyectos separados por capas /bssyj-war y /bssyj-data (aquí esta la lógica de la aplicación)
7) J2EE 1.6
8) Google Web Toolkit 1.4.62
9) JRE 1.6
10) Common Collection 3.1

C. Arquitectura utilizada (Patrones)

1) GWT MVP (http://code.google.com/intl/es-ES/webtoolkit/articles/mvp-architecture.html)
2) DAO (http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html)
3) Singleton (http://es.wikipedia.org/wiki/Singleton)
4) DTODozer (http://dozer.sourceforge.net/documentation/gettingstarted.html)

D. Implementaciones propias de la aplicación de ejemplo.

1) Multilenguaje implementado con el frmework GWT
2) Manejo de Imágenes con ImageResource de GWT

VII. Referencias

[1] Adam Tacy, Robert Hanson, Jason Essington, Ian Bambury, Christopher Ramsdale, *Gwt in Action*, Editorial: O'Reilly Media, 2012.
[2] Daniel Guermeur, Amy Unruh, Amy Unruh, Google App Engine Java and GWT Application, 2010.
[3] James Elliott, Timothy O'Brien, Ryan Owler, ñHarnessing Hibernateò, 2008.

VIII. Agradecimientos