# Análisis Framework GWT para desarrollo visual en web con Java EE.

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

***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

GWT MVP Explicación y motivos de su implementación.

### *A. Motivo:*

La construcción de cualquier aplicación a gran escala tiene sus obstáculos, y las aplicaciones GWT no son una excepción. Que varios programadores trabajen simultáneamente en la misma base de código, mientras que el mantenimiento de las características heredadas y la funcionalidad, puede convertirse rápidamente en código desordenado. Para ayudar a mejorar el código introducimos los patrones de diseño para crear zonas compartidas de responsabilidad dentro de nuestro proyecto.

Hay varios patrones de diseño para elegir, 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) de la arquitectura funciona mejor cuando el desarrollo de aplicaciones GWT por dos razones principales. En primer lugar el modelo de MVP, al igual que otros patrones de diseño, desarrollo desacopla de una manera que permite a varios desarrolladores trabajar simultáneamente. En segundo lugar, este modelo nos permite minimizar el uso de GWTTestCase, que se basa en la presencia de un navegador, y por la mayor parte de nuestro código, escribir ligero (y rápido) las pruebas de JRE (que no requieren de un navegador).

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

Creación de una aplicación basada en MVP puede ser sencilla y fácil una vez que entienda los fundamentos detrás de este modelo de diseño. Formato para los objetos insertados.

### *B. 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 noción de la modelo. Es decir, un punto de vista no sabe que está mostrando un objeto, simplemente sabe que tiene, por ejemplo, 3 etiquetas, cuadros de texto 3, y 2 botones que se organizan en forma vertical. Cambio entre los puntos de vista está vinculado a la gestión de la historia dentro de la presentación de la capa.

#### *3) Presentador*

Un presentador contiene toda la lógica de nuestra aplicación de bssyj, incluyendo la gestión del history del browser , ver la sincronización de transición y datos a través de RPC de vuelta al servidor. Como regla general, por cada punto de vista usted querrá un presentador para conducir la vista y manejar los eventos que provienen de los widgets de la interfaz de usuario dentro de la vista.

#### *4) AppController*

*C. 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.*

*Vista la transición está directamente vinculada a la gestión del history del browser.*

## II. 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 y asociados vista juntos 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 anterior, 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, es a un nivel suficientemente alto para que una vista sea capaz de cambiar la implementación específica de la lista, sin efectos secundarios. Método SetData () es una forma sencilla de obtener modelos de datos en la vista sin el conocimiento intrínseco del propio modelo. Los datos que se muestran está directamente ligada a la complejidad de nuestro modelo. Un modelo más complejo puede llevar a un mayor número de datos que se muestran en una vista. La belleza de la utilización de bindDTO () es que los cambios en el modelo se puede hacer sin actualizar el código de la vista.*

*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 escuchar los eventos de la interfaz de usuario que tenemos lo siguiente:*

```
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 bindean con los eventos que se originan en la vista , tendrá que tomar alguna acción sobre estos eventos. Para ello, tendrá que contar con un bus de eventos que se construye en la parte superiora de la GWT HandlerManager . El bus de eventos es un mecanismo para que una serie de eventos que traspasen y registren información para recibir información de algún subconjunto de estos eventos.*

*Es importante tener en cuenta que no todos los eventos deben ser colocados en el bus de eventos. Si todos los eventos posibles dentro de su aplicación en el bus de eventos puede llevar a el dumping de y perderse en el manejo de eventos.*

*Eventos de altos nivel son realmente los únicos eventos que desea que se pasa alrededor del bus de eventos. La aplicación no está interesada en eventos como "el usuario hace clic en enter" o "un RPC está a punto de desarrollarse". 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 RPC uno que 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 estas piezas, veamos lo que ocurre cuando un usuario quiere loguarse. En primer lugar vamos a necesitar el AppController para inscribirse en el LoginEvent. Para ello, hacemos un llamamiento HandlerManager.addHandler () y pase 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 de la HandlerManager , llamado eventBus, y se está registrando una nueva LoginEventHandler. Este controlador pasar a la doLogin () método cada vez que un evento de LoginEvent.getAssociatedType () se dispara. Varios componentes se pueden escuchar por un solo evento, por lo que cuando se dispara un evento con el HandlerManager.fireEvent () , el HandlerManager busca cualquier componente que se ha 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 echar un vistazo al código que las fuentes de la LoginEvent. Como se mencionó anteriormente, nos 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 () de clase que se inicia con la identificación de los login.*

## III. *A CONTINUACIÓN VAMOS A ENUMERAR LAS TECNOLOGÍAS UTILIZADAS EN EL PROYECTO DE EJEMPLO.*

### A. *Implementación de Hibernate y explicación explicación de mapeo con esta Base de Datos de ejemplo y paquete de beans.*

1) *Por que uso Hibernate, resumen:*

a) *Modelo natural de Programacion*

b) *Persistencia Transparente*

c) *Alta Performance, lazy*

d) *Auto cache y Query cache*

e) *Fiabilidad y escalabilidad*

f) *Transactions, auto rollback y sessions.*

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: **ñ**org.hibernate.tool.hbm2ddl.***ò**, con la cual se necesita crear un XML de comportamiento **ñ**ant**ò** se corre y genera en base a la configuración de la base de datos con la cual esta creado genera el 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 archivo de comandos 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)
);
```

*B. Capa DAO parte de la capa M:Modelo del patron de diseño MVP*

*1) Motivos*

*El acceso a los datos 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 Protocol (IIOP) o sockets de bajo nivel. Los componentes de negocio que tratan con el DAO utilizan un interface simple expuesto por el DAO para sus clientes. El DAO oculta completamente los detalles de implementación de la fuente de datos a sus clientes. Como el interface expuesto por el DAO no cambia cuando cambia la implementación de la fuente de datos subyacente, este patrón permite al DAO adaptarse a diferentes esquemas de almacenamiento sin que esto afecte a sus clientes o componentes de engocio. Esencialmente, el DAO actúa como un adaptador entre el componente y la fuente de datos.*

*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.*

*4) 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*

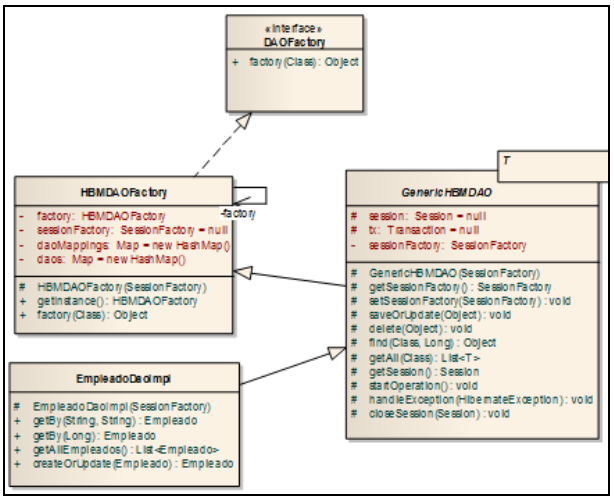
*d) Centraliza Todos los Accesos a Datos en un Capa Independiente*

*5) Desventajas*

*a) Añade una Capa Extra*

*b) Necesita Diseñar un Árbol de Clases*

*Diagrama de clases de la aplicación de ejemplo:*




*Ejemplos de clase genérica 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.*

```java
/**
 * 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 metodos genricos Crud estan implementados de la siguiente forma:*

```java
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:*

```java
@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 aca que implementamos el DAO en base a el bean Empleado que tiene métodos propios como getB(Sring, String) con hibérnate*

```java
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", pa
                    );
            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*

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

*C. Patron Singleton y Factory como parte de la capa M:Modelo del patron 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.*

*Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.*

*El patrón singleton se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con atributos como protegido o privado).*

*La instrumentación del patrón puede ser delicada en programas con múltiples hilos de ejecución y en nuestro ejemplo para evitar que múltiples usuarios creen muchas instancias de la clase Servicios. Si dos hilos de ejecución intentan crear la instancia al mismo tiempo y esta no existe todavía, sólo uno de ellos debe lograr crear el objeto. La solución clásica para este problema es utilizar exclusión mutua en el método de creación de la clase que implementa el patrón.*

*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.*
*La clase factoría devuelve una instancia de un objeto según los datos que se le pasan como parámetros. Para que la creación centralizada de objetos sea lo más "útil y eficaz" posible, es de esperar que todos los objetos creados desciendan de la misma clase o implementen el mismo interfase (es decir, hagan una operación similar pero de distintas formas.*

*3) Implementación en el Ejemplo bssyj*

*La idea es que las clases de Serviciosôs y DAOôs sean 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 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 HBMDAOFactory.getInstance(); lo hace a nivel de interfase con solo hacer la Factoría ya devuelve el árbol de herencia*

```
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 guardandola en un variable a reutilizar que se llama factory, aca vemos como se uno 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.*

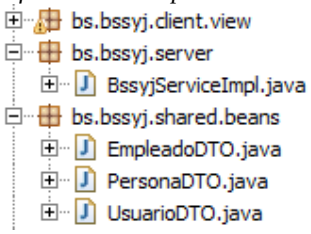
```
public  class HBMDAOFactory implements DAOFactory {

	/**
	 * 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;

	}
```

*D. Implementación de estratégia de DTO para poder resolver la probelmatica de Cliente Servidor de GWT com DTO Dozer*

*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 acomodan 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 JS transformado como por el servicio real.*




*Como vemos en este caso utilizo una instancia de DozerBeanMapper, 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 etc, como se menciono anteriormente obtengo la instancia del bean y el caso siguiente es invocar a dozer para que se encargue de trasformarlo automáticamente, si tener la necesidad escribir mucho código.*

```
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 ;
    }
```

## IV. Comparación entre GWT y Struts 2

*Esta comparación está enfocada en desarrollo de una interface de un ABM quitando la ingeniería de de obtención de datos y aplicación de modelos MVC o MVP, en síntesis JSP vs Clases Windows y componentes*

*A. Para JSP se utilizó:*

*1) Plug in de JQuery*

*a) jquery-1.3.2*

*b) jquery.form*

*c) jquery.maskedinput-1.2.2*

*d) jquery.gridSelection*

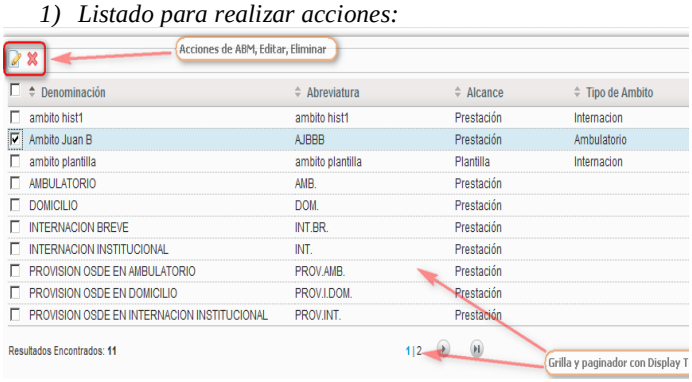
*e) tabs*

*f) jquery.autocomplete*

*2) HTML*
*3) JavaScript*
*4) Display Tag, para armar listas*
*5) JSTL de Struts 2*
*6) Adaptacion para Diferentes Browsers, código específico para lograr un front-end correcto, en este caso IE6 FireFox y Chrome*
*7) Manejo de CSS*
*8) Manejo de archivos de archivos JSP con include*

*B. Veremos un caso simple*

*1) Listado para realizar acciones:*




*Al presionar editar envía a la siguiente pantalla:*


*Al eliminar aparece los siguiente con el plug in showMessage de jQuery:*


| Tarea | Tiempo en horas Aprendizaje | Tiempo en horas de Implementación | Obs |
|---|---|---|---|
| Plug in de JQuery | 45 | 32 | 77 HS de utilización de jquery |
| jquery-1.3.2 | 16 | 8 | |
| jquery.form | 4 | 7 | |
| jquery.gridSelection | 15 | 4 | |
| tabs | 5 | 8 | |
| jquery.autocomplete | 5 | 5 | |
| HTML | 0 | 18 | |
| JavaScript | 0 | 8 | |
| Display Tag, para armar listas | 16 | 18 | |
| JSTL de Struts 2 | 0 | 4 | |
| Adaptacion para IE6 FireFox y Chrome | 0 | 24 | |
| Manejo de CSS | 0 | 5 | |
| Manejo de archivos de archivos JSP con include | 0 | 2 | |
| | 61 | 111 | |

### C. *Para GWT se utilizo:*

*Se omiten capturas de pantallas debido a que fue presentado con anterioridad.*

1) *SmartGWT*
   a) *Manejo de TAB*
   b) *Dynamic Forms*
   c) *Validators*
   d) *Buttons ítems*
   e) *List Grid*
   f) *Implementación de DataSource*
2) *ImageResource de GWT*
3) *Comprender el concepto de Cliente servidor y RPC*
4) *Manejo de Eventos e Implantación*

*Esta es la conclusión de la investigación de todo el documento.*

| Tarea | Tiempo en horas Aprendizaje | Tiempo en horas de Implementación |
|---|---|---|
| **SmartGWT** | 21 | 20 |
| o **Manejo de TAB** | 2 | 8 |
| o **Dynamic Forms** | 4 | 2 |
| o **Validators** | 2 | 3 |
| o **Buttons ítems** | 1 | 1 |
| o **List Grid** | 6 | 4 |
| o **Implementación de DataSource** | 6 | 2 |
| - **ImageResource de GWT** | 6 | 2 |
| - **Comprender el concepto de Cliente servidor y RPC** | 12 | 0 |
| | 39 | 42 |

## V. *Conclusión de comparación*

*Tiempos insumidos, para un desarrollador Semi-senior con 3 años de desarrollo java web, se divide en 2 Aprendizaje y Implementación.*
*Cabe destacar que se piensa en incorporar tiempo para personas que absorban este conocimiento y que luego cuando se quiera hacer otro ABM los tiempos de Aprendizaje serán un 80% menor y los de Implementación un 20% menores. También incluye tiempos de optimización de html y css.*

## 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

*Si los hubiere, a quienes corresponda.*