Análisis Google Web Tollkit para desarrollo visual en web con Java EE.

Sebastián Echarte, Dario Secualino, 1Alejandro R. Sartorio,

1 Centro de Altos Estudios en Tecnología Informática, Sede Rosario,

Universidad Abierta Interamericana, Ovidio Lagos 944, 2000 Rosario, Argentina

[secharte@gmail.com](mailto:secharte@gmail.com)

[sartorio@cifasis-conicet.gov.ar](mailto:sartorio@cifasis-conicet.gov.ar)

[dsecualino@telecom.com.ar](mailto:dsecualino@telecom.com.ar)

*Resumen*— 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 ‘s 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, “bssyj”, 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.

# Introducción

**C**

omo parte del trabajo final de ingeniería (TFI) se decidió crear un sistema informático de gestión de sueldos y jornales llamado bssyj. Contiene, entre otros módulos, una nómina de empleado, con usuarios y roles, módulo de liquidación con fórmula para cálculos, administración de Conceptos, informes para la gestión, visualización a través del entorno Web de recibo de sueldo por empleado, etc.

Como se puede apreciar, es un sistema comercial estándar que abunda en el mercado, por lo que se decidió diferenciar este sistema de los competidores directos a través de la tecnología. Para ello, se utilizó GWT ya que es relativamente nuevo en el mercado, así como también pantallas dinámicas e innovadoras.

Para realizar una aplicación utilizando tecnología tradicional, necesitaríamos aprender varios lenguajes de programación, HTML, JavaScript y AJAX para la funcionalidad asíncrona, CSS para el formato de la web y SQL para la base de datos. Aprender bien todos estos lenguajes requiere mucho tiempo, y en el mundo del del software el tiempo no suele ser un compañero.

Google Web Toolkit (GWT) permite el uso de Java para el desarrollo web con todas las ventajas que ello implica: lenguaje orientado a objetos, uso de Hibernate para la persistencia de datos, posibilidad de usar un entorno de desarrollo integrado (IDE) como Eclipse. Con el uso de Java evitamos el tiempo necesario para tener que aprender todos los lenguajes citados anteriormente y hacemos que sea más sencillo el desarrollo y mantenimiento de la aplicación.

Esta publicación describe el diseño y desarrollo de una aplicación web para gestionar sueldos y jornales. La aplicación está desarrollada usando GWT y las librerías de SmartGWT [4] que aportan a GWT una gran cantidad de componentes visuales con funcionalidad extra como ordenación en tablas y en árbol, formularios con muchas opciones, lista de selección múltiple, interfaces atractivas, rendimiento fluido etc. Estas librerías se distribuyen bajo licencia GPL [11] .

Google propone para aplicaciones que utilicen GWT, la implementación del patrón de diseño llamado MVP [6].

# Definiciones de la aplicación

A continuación, se presentan algunas definiciones del sistema con el cual aplicaremos los patrones y la implementación.

Del ejemplo práctico “BS-Sueldos y jornales“(BSSYJ) se describe lo siguiente:

## 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 comerciales.

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.

## Modelo de distribución del software:

Se utilizará la computación en la nube [12,14] es un concepto el que incorporamos al [software como servicio](http://es.wikipedia.org/wiki/Software_como_servicio). Es decir, nuestro sistema se ofrece como [servicio](http://es.wikipedia.org/wiki/Servicio_Web), de modo que los usuarios puedan acceder al mismo a través de "la nube de Internet" sin conocimientos (o, al menos sin ser expertos) en la gestión de los recursos informáticos que usan.

La idea es proveer aplicaciones comunes de negocio en línea accesibles desde un [navegador web](http://es.wikipedia.org/wiki/Navegador_web), mientras el [software](http://es.wikipedia.org/wiki/Software) y los [datos](http://es.wikipedia.org/wiki/Datos) se almacenan en los servidores.

## Descripción sucinta del negocio.

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

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 nóminas, 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.

## Casos de uso de ejemplo

Veremos los casos de uso para el ejemplo que se desarrollará en esta publicación. La notación usada es UML.

## 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 aplicativo. 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. | | | |

## Caso de uso: Alta Usuario

Diagrama de caso de uso:



Especificaciones de caso de uso:

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **02** | **Nombre** | **Alta Usuario** |
| Descripción | Permitirá a los distintos Usuarios del Sistema registrarse al sistema. | | |
| Actores | Usuario | | |
| Precondición | El Usuario debe haber ingresado al aplicativo web. | | |
| Post-condición | 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, fecha nacimiento, domicilio, Email apellido y nombre.  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 “usuarios”.  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 “Borrar”.  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. | | | |

# Arquitectura

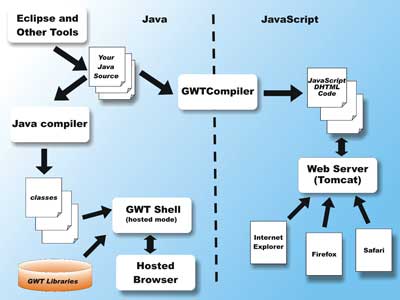
A continuación veremos la arquitectura utilizada y la relación entre GWT y los patrones implementados al aplicativo.

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

## 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 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 programador del JavaScript necesario para implementar dichas funcionalidades. El compilador de GWT ~~que es la pieza central~~ 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, etc.).

Vemos en la Fig. 1 que desde un IDE Eclipse u otra herramienta se genera código Java para crear las vistas y lógica de negocio, el compilador Java transforma un código compilado en modo local (hosted mode), el cual trabajamos constantemente con un navegador local y con esto podemos hacer debug. Luego tenemos el GWTCompiler que es la versión final que transforma lo que se encuentra del lado del cliente, objetos visuales en JavaScript que pueden ser visualizados en cualquier Navegador Web. Se utiliza hosted mode para trabajar y GWTCompiler cuando se quiere llevar la aplicación a un ambiente productivo.



1. Diagrama de GWT estándar.

## Fundamentos MVP explicados brevemente:

### Modelo Vista Presentador y AppController

Modelo: incluye objetos de negocio.

Vista: contiene todos los componentes de interfaz de usuario que componen nuestra aplicación. Esto incluye todas las tablas, etiquetas, botones, cuadros de texto, etc.

Presentador: tiene toda la lógica de aplicación, la sincronización de transición y datos a través de (Remote Procedure Call o 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.

Por último, AppController hace frente a la lógica que no es específica de ningún Presentador; por ejemplo, este componente podría contener la gestión de la historia del navegador.

## Integración MVP y GWT

En la Fig. 2 podemos ver de izquierda a derecha la Vista, con componentes SmartGWT [16] se ven los componentes solo en modo de compilación, con SmartGWT, tenemos herramientas como, componentes HTML, Labels, Text Box etc. y se establecen usando clases java.

El Presentador gestiona los eventos de las Vistas que son implementadas a través de “Display Interfase”. El Presentador los define y la Vista los implementa. A su vez, el Presentador es el encargado de actualizar la Vista, ya sean datos como eventos, por ejemplo, cambiar el tamaño de una venta, llenar datos a una grilla, etc.

Continuando con la Fig. 2: el “Event Bus” es un mecanismo para pasar y registrar eventos que serán utilizados a través de toda la aplicación, almacena eventos que utiliza la aplicación y se implementa a través de “HandlerManager”, clase proporcionada por GWT. Es parte de la clase APPController.

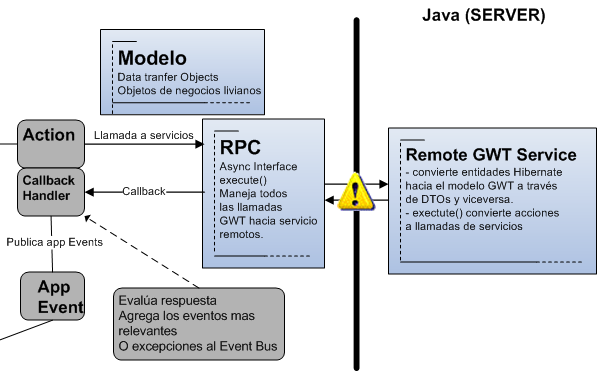


1. Integración MVP y GWT parte 1 de 2

La Fig. 3: es la continuación de la Fig. 2: y muestra la implementación RPC propia de GWT que maneja todas las llamadas GWT hacia servicios remotos. El “Remote GWT Service” convierte entidades de negocio hacia el modelo GWT a través de DTOs, y convierte acciones o eventos (del lado del cliente) a llamadas de servicios. El “Callback Handler” agrega los eventos más relevantes al Event Bus en modo de compilación y publica eventos de cada controladora.

El Modelo contiene la representación de objetos de negocio (beans) como DTO y una representación de servicios livianos del lado del Cliente.

Hay algunas particularidades de GWT, como “conexión asíncrona” [18] se refiere a que sólo se conectan en tiempo de ejecución sin haber una relación implícita en el código. La conexión vía Ajax se hace implícitamente sin necesidad de crear lo objetos de Ajax en la vista, únicamente manipulando eventos es suficiente.



1. Integración MVP y GWT parte 2 de 2

En la Fig. 4 tenemos una implementación más específica de cómo el modelo, GWT y MVP están siendo integrados para implementar Bssyj.

En la Vista tenemos las clases BssyjWindow de la cual heredan todas las ventanas que se deseen crear. Aquí se incluye EmpleadoView, que es el ABM de la entidad Empleados. Luego vemos como EmpleadoView implementa los métodos de la interface DisplayEmpleado. EmpleadoPresenter define qué van a hacer los eventos de EmpleadoView.

Desde la perspectiva del Presentador, tenemos BssyjPresenter de la cual heredan todos las clases presentadoras que resuelven comportamientos específicos de la aplicación, con el objetivo de compartir Internacionalización, constates y comportamientos que puedan surgir con el avance del desarrollo.

En la sección inferior la clase LogionEvent está contenidoen la clase LoginPresenter y va a recibir eventos externos desde el servidor. Por ejemplo, el login correcto del usuario o la expiración de la sesión.



1. Integración MVP, GWT y Ejemplo práctico.

# Implementación GWT en bssyj

Veremos en los puntos A y B el inicio de la aplicación y como se establecen todos los objetos fundamentales, explicado en detalle con código fuente.

## Proceso de arranque de GWT.

En la Fig. 5 veremos el código del método onModuleLoad() de la clase Bssyj crea el servicio RPC, el bus de eventos, y AppController, en la siguiente línea se instancia RootPanel (va a ser en inicio para la crearon de todo lo visual), a partir de entonces el AppController está en control de la creación de determinados [presentadores](https://developers.google.com/web-toolkit/articles/mvp-architecture#presenter).



Fig.5 Inicio o Main de la aplicación.

## Binding presenters y Vista

Por ejemplo el EmpleadoView:

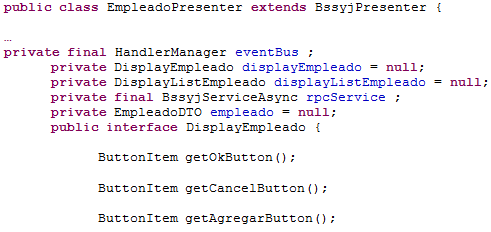
Este punto de la vista tiene 3 widgets [17]: una tabla y botones. Para que la aplicación pueda hacer algo significativo, el presentador se va a necesitar:

### Responder a clics en los botones

### Llenar la lista

### Responder a un usuario al hacer clic en la lista

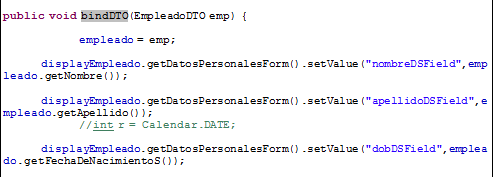
En el caso de nuestro EmpleadoPresenter, se define la interfaz de visualización como por ejemplo:

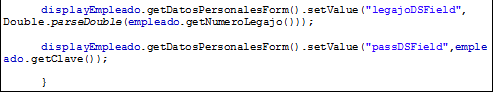


1. Relación entre eventos de vista y presentador

Mientras que el EmpleadoView implementa la interfaz DisplayEmpleado, definiendo los widgets y EmpleadoPresenter va a decidir que hacer con cada widget. 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. El motivo de la utilización de bindDTO() Fig. 6 de la clase EmpleadoPresenter es que los cambios en el modelo se puede hacer sin actualizar el código de la vista como se ve en la próxima porción de código Fig. 6. Para mostrar cómo funciona, veamos el código que se ejecuta al recibir 1 empleado desde el servidor que ya fue transformado a DTO, - el método setEmpleadoById() de la Fig. 6:

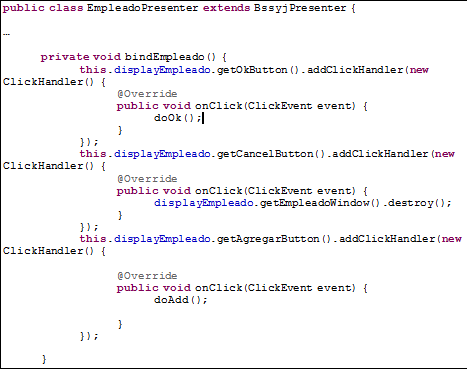
**





1. llenado de datos con información de un DTO

Para capturar los eventos de la interfaz de usuario, tenemos el siguiente código en EmpleadoPresenter:



1. Capturar eventos de la Vista

## Eventos y el bus de eventos

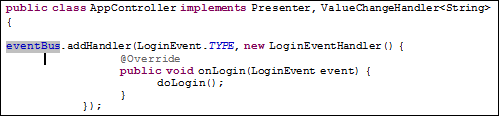
Para crear un bus de eventos se construye heredando de la clase de GWT “HandlerManager”.

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

Los eventos de alto nivel son realmente los únicos que deben pasar por del bus. 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 y en bssyj, 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: LoadMenuEvent, LoadMenuEventHandler, LoginEvent, LoginEventHandler.

Para demostrar cómo encajan los eventos del bus, veamos lo que ocurre cuando un usuario quiere logearse (caso de uso xx). En primer lugar vamos a necesitar el AppController (sección xx) para agregar el LoginEvent al event bus. 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.

**

1. Agregar un evento de aplicación al event bus

Aquí el AppController tiene una instancia del HandlerManager, llamado eventBus, y se está registrando el evento LoginEventHandler, que tiene el método “onLogin” , este controlador pasar a ejecutar el método privado doLogin(), cada vez que un evento de LoginEvent se dispara.

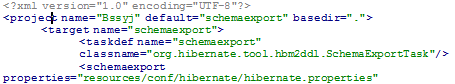
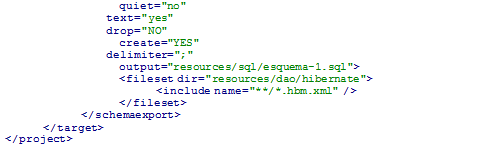
# Modelo de datos

## Implementación de Hibernate y explicación de mapeo con esta Base de Datos de bssyj y paquete de beans.

Hibernate es un es una API de tipo Object Relational mapping (ORM) para Java.

Los Motivos de utilización de Hibernate son: un modelo natural de Programación, Persistencia Transparente, Alta Performance, lazy, Auto cache y Query cache, Fiabilidad y escalabilidad, Transactions, auto rollback y sessions [3].

Generación automática de Base de datos (esquema) basándose en el modelo de datos. Para que se pueda realizar esto Hibernate permite usar una conjunto de herramientas llamadas: “hbm2ddl”, 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.

**

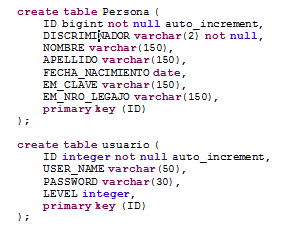
1. Para la creación de SQL en base a mapeo

El diagrama de Entidades de Negocio será esto responde a los casos de uso 01, 02 y 03:



1. Diagrama de clases de modelo de negocio.

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



1. Esquema creado desde mapeos

## Capa DAO

Se encuentra en el “Modelo” del patrón de diseño MVP.

### 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.

### Solución

Utilizar un Data Access Object (DAO) [19] 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.

### Ventajas 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 Transparencia

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

### Desventajas DAO

a) Añade una Capa Extra

b) Necesita Diseñar un Árbol de Clases

Diagrama de clases de la aplicación de ejemplo Para DAO:

**

1. DAO genéricos

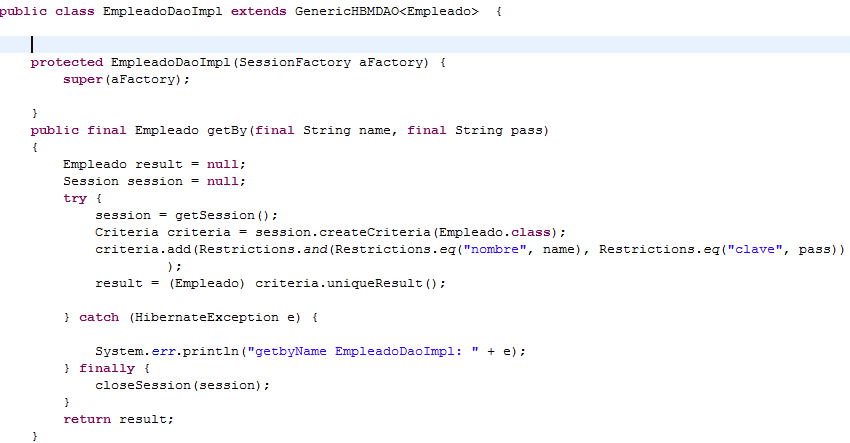
Ejemplos de clase genérica (Create Update delete) Crud, en esta clase GenericHBMDAO<T>, podemos ver que indicándole un bean en T como parte de generics de Java, todas las clases que heredan de esta clase abstracta pueden usar métodos comunes como add, delete, detall etc.

Los métodos genéricos Crud están implementados de la siguiente forma:

**

1. *DAO con genéricos*

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



1. Genérico de modelo de datos implementado

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

Tenemos por ejemplo el método getAllEmpleados() : List<Empleado> que lo único que hace es utilizar el método getAll(Empleado.class) pasándole como parámetro la clase el método es de la clase GenericHBMDAO, al igual que el método createOrUpdate(Empleado) : Empleado, solo contiene una línea de código que la que llama a saveOrUpdate(Object) pasándole como parámetro la instancia a crear o editar, el método es de la clase GenericHBMDAO. En la clase GenericHBMDAO en los métodos saveOrUpdate(Object) y getAll(Class).

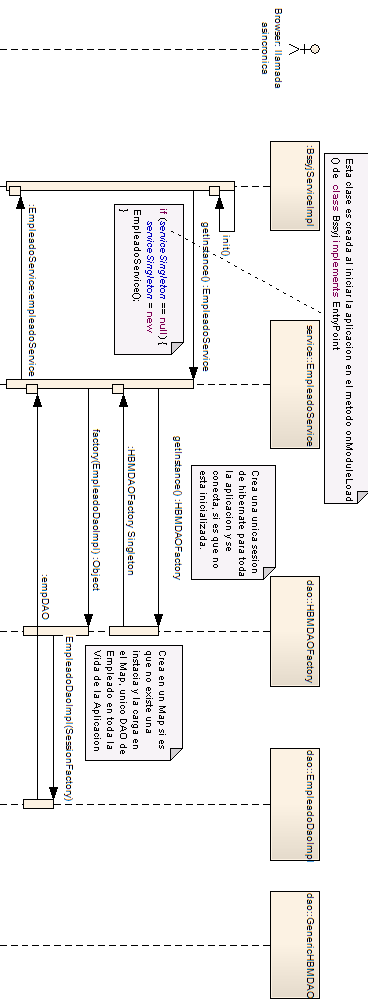
## Patrón Singleton y Factory.

Se encuentran en la parte de la capa “Modelo” [19] del patrón de diseño MVP. 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. La idea que se esconde detrás del patrón Factoria 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.

### Implementación en el Ejemplo bssyj

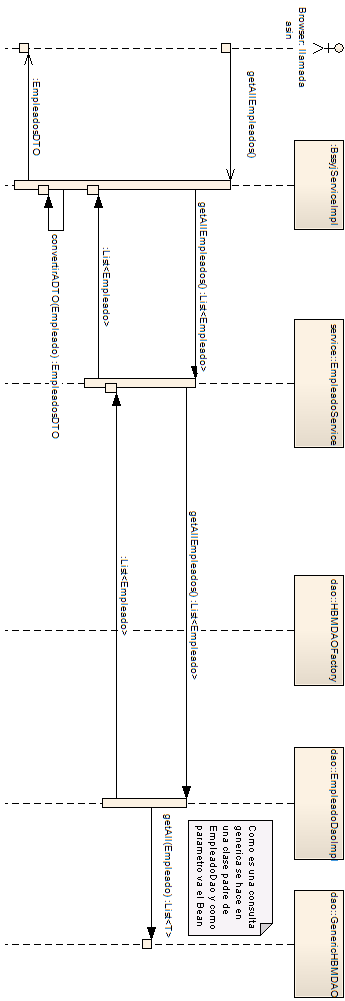
Las clases de Servicios y DAO son Singleton e inicializadas por la Factoría todo guardado en un contenedor de objetos llamado HashMap único para toda la vida de la aplicación en el servidor.

Veremos un Diagrama de secuencia con el ejemplo, correspondiente al el CU 02 obtener todos los usuarios para poder listarlos, pasando por las N capas, pero primero vemos que al iniciar la aplicación sucede lo siguiente:



1. Creación de clases para uso, singleton

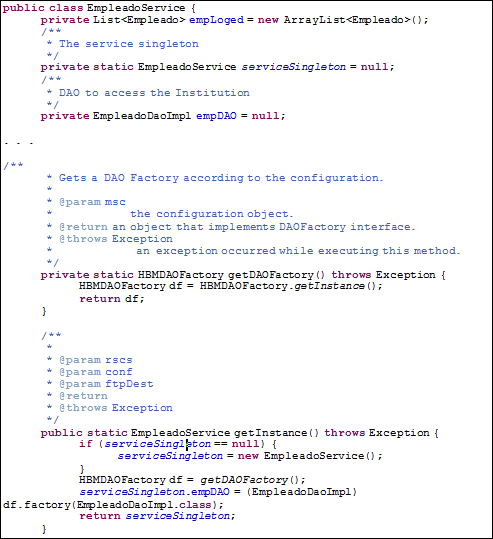
Luego una vez que las instancias estén en la memoria de la aplicación solo queda, obtener todos lo empleado con el método getAllEmpleados().



1. Utilización de clases singleton, ya iniciadas

Veremos a continuación como las variables “serviceSingleton” de tipo: “EmpleadoService” y

“empDAO” de tipo EmpleadoDaoImpl, se mantienen singleton tan solo manejando una Factoría de instancias, que solo verifica 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.

**

1. DAO, Singleton, y factoría en acción 1 de 2

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.

**

1. DAO, Singleton, y factoría en acción 2 de 2

Acá vemos como se guarda y se mantiene un conjunto de DAOS en la variable daoMappings, de nuevo combinamos todo, DAO, Singleton y factoría.



1. Singleton de DAOs

## Implementación de estrategia de DTO.

Es para poder resolver la problemática de Cliente Servidor de GWT con DTO Dozer [9] (API para poder transformar un objeto de negocio en su similar de DTO, configurable para soportar diferentes tipos de objetos de bean a beanDTO)

### Motivo

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

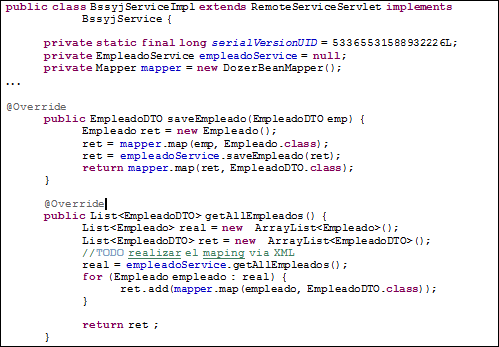
En este ejemplo se crean tantos DTO como beans existan y después de recibir el objeto de Hibernate que tenía para convertirlo en EntidadDto.

Para convertir a DTOs Beans se usa dozer [9] 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.

Como vemos en este caso se utiliza 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.

Obtengo la instancia del bean y el caso siguiente es invocar a dozer para que se encargue de trasformarlo automáticamente.

**

1. Utilización de Dozer

# 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, “Harnessing Hibernate”, 2008.
4. SmartGWT (<http://www.smartclient.com/smartgwt/showcase/>)
5. MVP (https://developers.google.com/web-toolkit/doc/latest/DevGuideMvpActivitiesAndPlaces)
6. GWT y MVP (<http://code.google.com/intl/es-ES/webtoolkit/articles/mvp-architecture.html>)
7. DAO (<http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html>)
8. Singleton (<http://es.wikipedia.org/wiki/Singleton>)
9. DTODozer (<http://dozer.sourceforge.net/documentation/gettingstarted.html>)
10. GWT (https://developers.google.com/web-toolkit/overview)
11. Licencia GPL (<http://es.wikipedia.org/wiki/GNU_General_Public_License>)
12. Kalagiakos, P. Cloud Computing learning Application of Information and Communication Technologies (AICT), 2011
13. Piotr Pawlak, Bartosz Sakowicz, Piotr Mazur, Andrzej Napieralski Social Network Application based on Google Web Toolkit.
14. Cloud Computing (http://en.wikipedia.org/wiki/Cloud\_computing)
15. Santiago Meliá, Jaime Gómez Sandy Pérez, Oscar Díaz A Model-Driven Development for GWT-Based Rich Internet Applications with OOH4RIA
16. Dongjin Yu y Pei Zhang Rapid Development of Web Applications Based on SmartGWT ISBN 978-952-5726-11-4 Proceedings of the Third International Symposium on Electronic Commerce and Security Workshops(ISECS ’10) Guangzhou, P. R. China, 29-31,July 2010
17. Widgets de SmartGWT (<http://www.smartclient.com/smartgwtee/javadoc/com/smartgwt/client/widgets/BaseWidget.html>)
18. Conexión asincrónica (<https://developers.google.com/web-toolkit/doc/latest/DevGuideServerCommunication?hl=en>)
19. Li Zhang, Design and realization persistence framework of Tourism E-business System based on J2EE Multimedia Technology (ICMT), 2011 International Conference on 26-28 July 2011 Coll. of Comput. Eng., Jiangsu Teachers Coll. of Technol., Changzhou, China

# Agradecimientos