# Obstáculos Técnicos Encontrados

# En Tempore

## Arquitectura y encapsulamiento aplicaciones GWT

Se requiera separar en diferentes proyectos la aplicación GWT de manera que sea más fácil de mantener y trabajar entre varias personas. Lo que se intento es separar entre los servicios que acceden a BBDD (DAO y Entities), la vista de la aplicación, los servicios requeridos por la vista para funcionar y los componentes requeridos para la integración entre la vista y servicios.

Los obstáculos que se presentaron es que el IDE de Eclipse no permitía la integración de proyectos de la manera convencional de la aplicación, sino que hay que colocarlos como link del proyecto para que funcionen. Este link emula que el código que esta en otro proyecto, se encuentra en el mismo.

## Servicios de GWT y SmartGWT

Se eligió el framework GWT para desarrollo, con el objetivo de facilitar el diseño y velocidad dela aplicación ya que termina trabajando con JavaScript.

SmartGWT trabaja en base al framework GWT y este trabaja con la particularidad básica de trabajar todos sus servicios asincrónicamente (AJAX).

Se tuvo muchos obstáculos al tener que pensar la aplicación de esa manera y tener en cuenta cuando requeríamos la sincronización de servicios debido a que se requerían unos a otros. Por ejemplo para solicitar las tareas de un proyecto primero se tenia que descargar los proyectos de ese usuario y luego las tareas del mismo, de lo contrario el servicio retornaba error por que no sabia que tareas bajar. Hubo dos soluciones distintas para este obstáculo. La primera fue la encadenar servicios de manera que cuando un servicio entregaba la respuesta se realizaba la solicitud del segundo y así tantos servicios como se requiriera unir.

Pero encontramos que la solución anterior no era prolija cuando se trataba de sincronizar paneles diferentes ya que eran paquetes de códigos diferentes de manera que se eligió una segunda solución, la de utilizar el patrón *Observer*, que avisaba a los observadores cuando debían actualizarse, sincronizando de esta manera los servicios.

## Spring y GWT

Remplazo del uso del framework Spring para uso de patrón MVC. Se termino utilizando el propio GWT una estructura de implementación de servicios y vistas bien encapsuladas.

## Framework Hibernate y Jetty:

Integración del framework Hibernate con GWT y Servidor Web Jetty (servidor por defecto de GWT). La versión del framework Jetty que venia por defe3cto con la versión de GWT utilizaba una librería (JPA) de hibernate desactualizada y que traiga conflictos con el mismo imposibilitando levantar la aplicación. Para solucionar el problema se tuvo que cambiar el servidor de aplicaciones por el Tomcat.

## DBUnit

Testeo de todos los servicios DAO mediante el uso de DBUnit.

## Framework Dozer:

Utilización de framework Dozer para mapeos entre entities y DTO. Dificultad para la correcta implementación e integración con el servicio.

## Hibernate Session:

Errores al utilizar la sesión de hibernate durante servicios concurrentes de la aplicación. Se estaba utilizando en forma incorrecta la sesión de hibernate, de manera que si dos servicios accedían simultáneamente al la solicitud no identificaba que eran el mismo solicitante y entraban en conflicto. Se soluciono inicialiazando los DAO en cada servicio y dejando de lado la creación de un singleton para el DAO.

## Framework para la generación de Reportes (GWT-Visaulization)

Los reportes que ofrecía el framework smartgwt eran pagos. Por lo que se decidió investigar otros frameworks gratuitos. El problema es que los que se encontraron estaban desarrollados sobre GWT y no sobre smartGWT. Este inconveniente trajo el problema que el reporte no podía ser adherido a los layout de smartgwt.

La solución al problema fue la de escribir código JSNI (JavaScript Native Interface) para realizar la adaptación entre los dos componentes. Se escribió código nativo de manera que los dos lo puedan entender y no tener problemas.

## Carga de imágenes en GWT y SmartGWT

La carga de imágenes en GWT es complicada, debido a que todos los servicios son asincrónicos e independientes. El problema que encontramos fue que teníamos el manejo de los usuarios y queríamos que estos puedan llegar a cargar imágenes a sus perfiles de manera que luego en la asignación pueda realizarse en forma mucho más amigable. Al intentar realizar esto con un form de Smartgwt nos dimos cuenta que este no soporta imágenes como ítems del form. Investigando encontramos una manera de resolver este problema y que era la misma manera en que lo resuelve aplicaciones como gmail u otras. Y es la de trabajar la imagen en forma independiente, cargar la imagen y subirla directamente a un servidor independiente del formulario. Luego cuando se completa el resto de la información del formulario se le asocia el ID de la imagen subida y se guarda como otro campo en la BBDD y luego con este ID se puede recuperar la imagen con un servicio simple (servlet para visualizar imágenes)

## Manejo de ABM en GWT

Se entendió que se requería desarrollar un framework propio para el manejo de todos los servicios tipo ABM. Se desarrollo interfaces que implementan los servicios y los datasource requeridos para que sean integrados rápidamente con el framework GWT y los servicios de background con pocas líneas de código.

## Utilización del Tree en SmartGWT

Al trabajar con el framework propio de ABM, no se comportaba correctamente cuando trabajábamos con el componente Tree de manera que se volvió a trabajar con servicios en forma indirecta.