UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

# Genaro: Generador de código gvHIDRA

Proyecto Final de Carrera

I.T.I.S.

**Autor**: Jose Morell Sendra

**Director**: Manuel Vázquez De Parga Andrade

28 de Septiembre de 2014

# Resumen

Este proyecto se centra en la creación de un generador de código de aplicaciones gvHIDRA (Generalitat Valenciana Herramienta Integral de Desarrollo Rápido de Aplicaciones), que mediante una interfaz gráfica, permite crear de forma rápida y sencilla mantenimientos funcionales (búsqueda, alta, baja y modificación). Para ello, únicamente requiere de la especificación de la conexión, la tabla de la BBDD y la forma de visualización (patrón de interfaz).

La mejora que se pretende obtener con esta herramienta es la creación de un aplicación web con el framework gvHIDRA en un tiempo mínimo.

# Tabla de contenidos

# 1. Introducción

## 1.1. ¿Qué es gvHIDRA?

Se trata de un proyecto OpenSource (GPL v.2) cuyo objetivo es simplificar el desarrollo de aplicaciones de gestión en entornos Web. Se trata de un proyecto OpenSource (GPL v.2) cuyo objetivo es simplificar el desarrollo de aplicaciones de gestión en entornos Web. Es un framework que, siguiendo una guía de estilo (unifica criterios de aspecto y usabilidad), sirve de base para las aplicaciones.

Trabaja sobre Web usando la arquitectura modelo vista controlador (MVC) con Phrame, Smarty y Pear. Utiliza la potencia del PHP (rapidez, sencillez en los despliegues, robustez, multiplataforma...) en su versión 5 (orientación a objetos, interfaces, iteradores, soporte SOAP...). Proporciona independencia de Base de Datos usando Pear::MDB2 más una capa de abstracción que trata otros aspectos como el manejo de fechas, números con decimales, control de transacciones,etc. Actualmente probado con PostgreSQL, Mysql y Oracle.

## 1.2. Funcionalidades

- Alta productividad.
- Estandarización de los desarrollos.
- Simplifica el entorno de trabajo Web (no necesitamos conocer HTML o JavaScript).
- Uniformidad en el aspecto y usabilidad de las aplicaciones.
- Separación de la lógica de negocio y de la presentación

- Independencia del gestor de base de datos empleado
- Control de acceso por usuarios y grupos
- Interfaz amigable.
- Componentes complejos: Ventanas de selección, Mensajes modales, listas enlazadas,...
- Patrones de interfaz: tabular, registro, maestro detalle, maestro n detalles, árbol.
- Integración con otros módulos
- utenticación de usuarios y permisos
- Listados PDF, ODT, CSV,...

# 2. Antecedentes

## 2.1. Generación de código en la programación web

La generación de código hoy en día se ha convertido en algo indispensable, ya que el ahorro en tiempo y recursos, la eficiencia en la programación y la estandarización son las bases fundamentales para el desarrollo de cualquier proyecto. La automatización de la programación requiere un gran cambio en la construcción del software ya que el programador solo tiene que preocuparse de la lógica de negocio. Esto se debe a que el resto de tareas pueden ser generadas de forma automática, ganando mucho tiempo al no tener que invertirlo en maquetación o implementación de la capa de acceso a datos. Estas tareas son simplificadas y con tan solo especificar algunos parámetros sin importar el lenguaje de programación conseguimos generar la estructura que tanto tiempo costaría hacer "a mano".

## 2.2. Necesidad de generar código

En el desarrollo de aplicaciones web siguiendo el modelo de 3 capas (Lógica de Negocio, Interfaz y Acceso a Datos) se dan una serie de patrones comunes en cualquiera de ellas, obligando al programador a repetir numerosas veces muchas lineas de código. Todo esto crea una programación ineficiente y provoca

En conclusión, la generación de código crea una estructura "base" de forma automática evitando que el programador tenga que escribir todo el código, suponiendo un ahorro de tiempo importante en el desarrollo.

Los generadores de código han acabado siendo una necesidad prácticamente obligatoria para la mayoría de programadores en el desarrollo de software. Existen muchas herramientas que permiten la generación a partir de diagramas.

La generación automática de código existe desde los primeros compiladores. Actualmente se ha llevado hacía otro nivel, permitiendo que usuarios finales se beneficien de ello. No obstante, en la mayoría de casos suelen costar mucho dinero, y hoy en día sigue sin existir una herramienta que cumpla con todas las necesidades. Esto se debe a que aunque se sigue avanzando, el programador no suele ir más allá de elaborar el código y no está mentalizado en buscar una herramienta que pueda ayudarle.

## 2.3. Tipos de generadores de código

- Plantilla: Se genera una estructura o borrador de código fuente no funcional para ser editado por el programador con el que se ahorra tiempo al no tener que escribir todo el código desde cero.
- Parcial: Se genera una estructura de código fuente que implementa parcialmente la funcionalidad requerida. Esta estructura será editada por el programador para modificar, añadir o quitar alguna funcionalidad.

## 2.4. Creación de un generador de código

- La arquitectura de software para la que se va a crear.
- El lenguaje de programación y sus características.
- El lenguaje con el que se generará el propio generador.

- El punto de partida desde el que se va a generar el código (Uml, tablas de la base de datos, diagramas).

## 2.5. Desventajas de los generadores existentes

Existen muchos generadores de código y a pesar de facilitarnos el trabajo también tienen algunas desventajas como:

- Generan código referente unicamente a los métodos básicos de conexiones a la base de datos: insertar, modificar, eliminar y en algunos casos consultas.
- No están orientados a una arquitectura.
- A veces resulta ser demasiado complicado hacer los diagramas para generar un simple código y otras veces no es posible especificar las funcionalidades que se requieren mediante un diagrama.
- Casi todos los generadores de código son aplicaciones de escritorio.

## 2.6. Arquitectura idónea para la generación de código

Varios son los módulos que forman una aplicación. Cada uno de estos módulos se encarga de realizar una tarea diferente. Ya que todo el software está formado por unos módulos parecidos independiente de las funcionalidades requeridas, como la capa de acceso a datos, la lógica de negocio o la interfaz del usuario, la identificación de estos módulos facilitará en gran medida la creación del diseño de nuestro generador de código.

## 2.7. Tipos de módulos

Los componentes de módulos que forman una aplicación basada en el modelo por capas suelen ser siempre los mismos que indicamos a continuación:

- Módulo de interfaz de usuario: La interfaz de usuario se encarga de la interacción entre el usuario y los datos. El proceso debe estar definido por una serie de pasos que con la ayuda de formularios y controles van a

  procesar y dar formato a los datos del usuario, validando y adquiriendo la información necesaria sobre las funcionalidades que la estructura de código generada deberá satisfacer.

- Módulo de Lógica de negocio: Esta se encarga de las tareas relacionadas con todos los procesos que se realizan detrás de la aplicación visible para el usuario, entradas de datos, consultas de datos, generación de informes, etc.

- Módulo de acceso a datos: Prácticamente todas las aplicaciones necesitan en algún momento acceder a datos almacenados, por eso, es necesario generar la lógica necesaria para crear una capa independiente para obtener el acceso a los datos para facilitar su programación, configuración y mantenimiento.

## 2.8. Conclusión

Valorando todos los aspectos de los generadores de código, se puede afirmar que su uso es de gran ayuda en el desarrollo de software, mejorándose la calidad de la producción, el establecimiento de estándares de código y el tiempo de desarrollo de las aplicaciones.

Además, con el uso de una arquitectura en componentes se puede llegar a obtener un máximo rendimiento del generador de código que se utilice, así como una buena practica de lo métodos de construcción de software utilizando capas o componentes.

# 3. Análisis del framework gvHidra

## 3.1. Arquitectura por capas

Concretamente, el framework ofrece tres capas:

1. **Core**: es la capa propia de los ficheros del framework. Está ubicada en el directorio igep y contiene ficheros de configuración propios del proyecto.
2. **Custom**: destinado a las configuraciones propias de la entidad donde se despliega la aplicación. Generalmente, la organización donde se despliegue una aplicación tiene una serie de características propias:
    - aspecto: se puede configurar a partir de css e imágenes.
    - Definición de listas o ventanas de selección generales (municipios, provincias, terceros, ...).
    - clases propias, tipos, web services, ...
    - configuraciones propias: conexiones a BBDD corporativas, acceso a datos comunes, formato de las fechas...
3. **app**: cada aplicación tiene una configuración propia:
    - acceso a la BBDD propios de la aplicación.
    - Listas y ventanas de selección propias de la aplicación.

- Configuraciones propias de la aplicación: nombre, versión, modo de debug y auditoría, comportamiento de la búsqueda, ...

Resumiendo, debemos tener en cuenta que hay varios niveles dentro de la arquitectura de una aplicación gvHIDRA.

## 3.2. Configuración de una aplicación gvHidra

Ya que la configuración se realiza a nivel de contexto de la aplicación no necesitamos tener muchas cosas en cuenta de la configuración a la hora de generar código. Tan solo tenemos que tener en cuenta que la configuración se carga en dos fases consecutivas siguiendo en cada una de ellas el orden (core/custom/app).

Las dos fases son:

1. Carga estática: Esta fase carga los parámetros que estén ubicados en el fichero de configuración gvHidraConfig.inc.xml. Tenemos un fichero de este tipo en cada uno de las capas que se cargarán por orden.
    - **Core**: igep/gvHidraConfig.inc.xml (Configuración propia del framework. No debe ser modificado)
    - **Custom**: igep/custom/xxx/gvHidraConfig.inc.xml. (Configuración propia de la organización)
    - **App**: gvHidraConfig.inc.xml (Configuración propia de la aplicación)

2. Carga dinámica: Esta fase se realiza una vez acabada la anterior, con lo que reemplazará las configuraciones que se hayan realizado. Puede ser muy útil para fijar configuraciones de forma dinámica (dependiendo del entorno de trabajo), pero es peligroso, ya que puede ser difícil depurar un error.

    - **Core**: igep/actions/gvHidraMainWindow.php. (Configuración dinámica del framework. No debe modificarse)

    - **Custom**: igep/custom/xxx/ actions/CustomMainWindow.php. (Configuración dinámica de la organización. Aquí se definen las listas y ventanas de selección de la organización)

    - **App**: actions/principal/AppMainWindow.php. (Configuración dinámica de la aplicación. Aquí se definen listas y ventanas de selección de la aplicación)

## 3.3. Aspecto visual

Una ventana gvHIDRA está dividida en cuatro secciones:

- Barra superior: En esta barra aparece alineado a la izquierda el título de la ventana, y, alineado a la derecha aparecerán, si lo hay, botones tooltip. Estos botones efectúan acciones relacionadas con la interfaz.

- Contenedor: Donde se ubicará el formulario con los datos y campos con los que trabajaremos.

- Barra inferior: En ella, alineado a la izquierda, se ubicará el paginador y, alineados a la derecha aparecerán los botones.

- Contenedor de pestañas: En esta zona irán apareciendo pestañas con las que podremos cambiar el modo de trabajo (búsqueda, listado, registro…)

## 3.4. Modos de trabajo

La forma de trabajar con las ventanas gvHidra se ha clasificado por modos de trabajo. Definiendo modo de trabajo como la forma de representación de la información con la que vamos a trabajar. Estos modos de trabajo se verán reflejados tanto en la zona contenedor del panel con en la la zona de Contenedor de pestañas.

Los modos de trabajo que tenemos son res que se corresponden con las pestañas:

1. Búsqueda o filtro
2. Tabular o listado
3. Registro o edición

Partiendo de todo lo comentado, tenemos dos formas de trabajar en gvHIDRA:

- Dos modos de trabajo: Se refiere a la forma de trabajar, en este caso partimos de una búsqueda y el resultado se muestra en una tabla o ficha, donde los datos ya son accesibles en cada uno de estos modos.

  Ejemplos: Modo búsqueda/tabla o búsqueda/ficha

- Tres modos de trabajo: Se trata de una combinación de los dos anteriores. Se parte de un panel de búsqueda, el resultado de dicha búsqueda se mostrará en un panel tabular. En este panel tabular los

datos no son accesibles, solamente se podrá efectuar el borrado de los registros que se seleccionen, para inserción o modificación se pasa a un panel modo ficha.

Ejemplo: Modo búsqueda/tabla/ficha

## 3.5. Patrones de interfaz

Los patrones de interfaz que se pueden implementar con gvHIDRA tienen en común el modo búsqueda y son los siguientes:

- Tabular: Se corresponde con dos modos de trabajo, panel de búsqueda y panel tabular donde se trabajará con los datos.
- Registro: Se corresponde con dos modos de trabajo, panel de búsqueda y panel registro donde se trabajará con los datos.
- Tabular-registro: Se corresponde con tres modos de trabajo, panel de búsqueda, panel tabular donde se tendrá el resultado de la búsqueda y panel registro donde se podrán editar los campos de las tuplas seleccionadas en el panel tabular o insertar nuevas tuplas.
- Maestro-detalle: La parte del maestro será dos modos de trabajo, un panel de búsqueda y un tabular o registro. La parte del detalle se puede plantear como la forma de trabajar de dos modos (tener un sólo tabular o registro) o tres modos (tabular y registro), con la excepción de que en el detalle no tenemos búsqueda.

  De esto podemos obtener diferentes combinaciones para un maestro-detalle:

- Maestro tabular – Detalle tabular
- Maestro tabular – Detalle registro
- Maestro registro – Detalle tabular
- Maestro registro – Detalle registro
- Maestro tabular – Detalle tabular-registro
- Maestro registro – Detalle tabular-registro

Además de estos patrones para el maestro-detalle contamos con un patrón más complejo, el **Maestro – N Detalles**. Este patrón es una extensión de cualquiera de los indicados anteriormente, tendremos un maestro que tendrá varios detalles asociados. Estos detalles están accesibles mediante unas solapas que aparecerán en la cabecera de la zona del detalle.

- Árbol: El patrón árbol se compone de dos zonas. En la zona de la izquierda encontramos el árbol en sí, una jerarquía de menús, seleccionando cualquiera de ellos accedemos a su información que aparecerá representada en la zona de la derecha. Esta información podemos decidir representarla con un patrón Tabular, Registro o Tabular-Registro.

## 3.6. Lógica de negocio

La lógica de negocio transcurre mediante acciones y operaciones. Son dos conceptos fundamentales que hay que conocer para desarrollar con gvHidra.

Una acción es el proceso que se inicia tras la solicitud de un usuario. Las acciones están divididas en dos grupos:

1. **Acciones genéricas**

   Las acciones genéricas resuelven procesos comunes de un mantenimiento (altas, bajas, modificaciones, búsquedas...)

   - Iniciar Ventana

     Esta acción se ejecutará al iniciar la ventana.

   - Buscar

     Realiza la consulta del panel basándose en los datos que hay en el panel filtro.

   - Editar

     Esta acción genérica se ejecuta cuando estamos trabajando con tres modos. Como su propio nombre indica realiza la consulta de edición a partir de los datos seleccionados en el panel tabular.

- Recargar

  Acción genérica que actúa en un patrón maestro-detalle. Se ejecutará cuando se efectúe un cambio de maestro, recargará los detalles correspondientes.

- Modificar

  Esta acción genérica se ejecuta desde el modo de trabajo edición, cuando, tras haber editado unas tuplas, se pulsa el botón guardar.

- Borrar

  Esta acción genérica se lanza normalmente desde el modo de trabajo tabular para eliminar las tuplas seleccionadas.

- Insertar

  Esta acción genérica se dispara desde el modo de trabajo inserción cuando, tras haber introducido los datos se pulsa el botón guardar.

- Nuevo

  Esta acción genérica se lanza para preparar la presentación de datos en pantalla antes de una inserción.

2. **Acciones particulares**

   Con las acciones particulares podremos implementar necesidades concretas no resueltas por el framework.

Las operaciones son métodos internos del framework que serán llamados a para cumplir lo que cada acción requiere, algunas de estas operaciones son públicas por lo tanto el programador podrá invocarlas desde las acciones particulares. El programador podrá modificar el compoortamiento de por defecto de la operación sobrescribiendo con una serie de métodos abstractos (virtuales).

Todas las acciones genéricas tienen una serie de métodos que el programador puede sobrescribir completando el comportamiento de la acción. Generalmente hay un método virtual “pre”, antes de la acción (inserción, modificación, borrado), y un método “post”, después de la acción.

# 4. Construcción del generador de código

## 4.1. Tecnologías empleadas

Para la construcción de este generador de código para gvHidra, se ha tenido muy en cuenta la arquitectura que compone el framework.

Esta arquitectura esta basada en el modelo 3 capas MVC (Modelo Vista Controlador). Se trata de un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y módulo encargado de gestionar la comunicación y los procesos. Por un lado tenemos los componentes para la representación de la información y por otro lado la interacción del usuario. Se basa principalmente en la re-utilización de código y separación de conceptos.

Como es una arquitectura fuertemente desacoplada ha sido fácil realizar la separación por módulos para construir cada uno por separado.

Para centralizar aplicación y generador de código en un mismo proyecto, con el fin de facilitar el uso del generador de código para añadir nuevas funcionalidades a la aplicación, y todo ello “en caliente”, es decir, sin tener que regenerar toda la aplicación, sino que se pueden generar nuevas funcionalidades en cualquier momento, tanto la aplicación como el generador comparten exactamente las mismas tecnologías. Esto supone una gran mejora respecto a otros generadores de código ya que no es necesario ninguna instalación extra de la propia instalación inicial para el uso del framework gvHidra.

A continuación vamos a detallar sin profundizar cuales son estas tecnologías:

- El lenguaje utilizado tanto por la aplicación como por el generador es el lenguaje PHP (Hypertext Preprocessor). Es un lenguaje de código abierto muy popular y especialmente adecuado para el desarrollo web. Algo muy característico de este lenguaje es su simplicidad sin obviar todas las características avanzadas que ofrece a los programadores profesionales.

- PEAR: Es una extensión de PHP y repositorio para componentes re-utilizables PHP. Uno de los componentes utilizados en el framework es el de autenticacióncación de usuarios PEAR Auth.

- MDB2: Se trata de un paquete que se encuentra dentro de la extensión de PEAR. Este paquete es uno de los más importantes y otorga una de las características más grande de Genaro ya que funciona bajo cualquier tipo de base de datos ya sea Mysql, postgres u oracle.

- SMARTY: Es un motor de plantillas para PHP. Esta herramienta facilita la manera de separar la aplicación lógica y el contenido en la presentación. Múltiples plantillas son empleadas para escribir de forma automática el código generado.

## 4.2. Diseño del generador

Para diseñar el generador de código para gvHidra había que partir de como funciona este framework, para poder automatizar la creación de código funcional, que hasta entonces era tarea del programador.

Una vez claro el punto de partida, era necesario establecer hasta donde iba a ser posible generar código, y es gracias al diseño del propio framework, siguiendo la arquitectura modelo vista controlador, como se conseguiría llegar a generar una estructura base completamente funcional en la que el programador tan solo iba a tener que preocuparse de añadir las funcionalidades extra que la aplicación requería.

El framework cuenta con patrones de diseño predefinidos con los que puede satisfacerse prácticamente cualquier requisito funcional de un mantenimiento (tabla/s de la base de datos). Los patrones comprendidos son:

- Tabular
- Registro
- Tabular-Registro
- Maestro-Detalle

Para crear estos patrones con todas sus funcionalidades hay que generar los ficheros necesarios y disponerlos en la ubicación correspondiente teniendo en cuenta la estructura del framework.

Dependiendo del tipo de pantalla que queramos generar el contenido de los ficheros será diferente. GvHidra está basado en la arquitectura MVC, por tanto al crear una pantalla tenemos que crear y modificar varios ficheros que controlen las diferentes partes de la arquitectura:

- Modelo: En el directorio *actions*, se creará un fichero que controla toda la lógica de negocio del panel.

- Vista: Directorio *views* y *plantillas*. Se crearán unos ficheros que por un lado controlan la interfaz antes de presentar los datos, y por otro lado, la distribución de los componentes en la pantalla.

- Controlador: Ficheros *mappings.php* y *menuModulos.xml*. Se tienen que modificar estos ficheros para indicar que acción se resuelve con que clase.

Dentro del modelo, cabe tener en cuenta que el programador interaccionará con el usuario a través de dos flujos: flujo de entrada y flujo de salida. El flujo de entrada se compone de la información que viene de la pantalla y se gestiona con:

- Tablas de trabajo: Son las tablas con las que el framework va a trabajar. Es decir, las tablas sobre las que vamos a realizar operaciones con los datos que recibamos.

- Matching: Este método nos sirve para indicarle al framework en que se traduce cierto campo de la pantalla. Se utiliza únicamente en los flujos de entrada: construcción de la Where de la operación, del Insert, Update, …

El flujo de salida se compone de la información a mostrar por pantalla y se gestiona a través de las consultas que definimos en el constructor de la clase.

## 4.3. Funcionamiento del generador

Esta herramienta, Genaro, a través de 5 parámetros básicos, se conecta a la base de datos y construye una ventana funcional. Los mantenimientos tienen la lógica necesaria para realizar la gestión de búsquedas, altas, bajas y modificaciones, pero no tienen lógica específica de la aplicación (validaciones, funciones específicas, ...)

Este generador de código realiza por nosotros la generación de los siguientes ficheros:

- Actions: genera las clases manejadoras (1 o N dependiendo del patrón). Estas clases tienen las consultas, las asociaciones con los campos, las asociaciones entre los maestros y sus detalles (en el caso de estos patrones) y los tipos de datos (con las validaciones de los mismos).
- Views: genera el fichero de la vista.
- Plantillas: genera la plantilla tpl de la ventana
- Include: edita el fichero include para incluir en el proyecto las nuevas clases.
- Mappings: edita el fichero de mapeos para asociar las acciones con la clase que las gestiona (las nuevas clases manejadoras).
- MenuModulos.xml: edita el menu para que aparezca la nueva entrada. Agrupa esta entrada por el módulo.

## 4.4. Instalación

En primer lugar tenemos que descargar el paquete de la web de gvHidra o directamente la última versión desde la forja de OSOR.

- Desde la página de gvHidra: www.gvhidra.org
- Desde la forja de OSOR: http://forge.osor.eu/projects/genaro/

Una vez descargado, lo dejamos en el directorio include de nuestra aplicación gvHidra.

Genaro, recoge la información de las conexiones de nuestra aplicación, por lo que necesitamos que estén definidas en el fichero gvHidraConfig.inc.xml de la aplicación.

Una vez definidas, debemos acceder a la herramienta http://<ubicacionAplicacion>/include/genaro. Por comodidad, se recomienda añadir esta entrada de menu en el menu de herramientas. Al acceder a esta url nos aparecerá el formulario donde tendremos que introducir los datos sobre nuestra aplicación que necesita Genaro para poder generar el código.

## 4.5. Manual de usuario

En primer lugar, tenemos que escoger la conexión de base de datos que queremos utilizar. Para ello, en el desplegable de la parte superior derecha, seleccionamos el dsn deseado. Los ids que aparecen, son los que hemos configurado en nuestra aplicación.

Una vez seleccionada la conexión, debemos decidir si vamos a generar un patrón simple o un patrón maestro detalle. Vamos a revisar cada una de las ventanas.

Para generar un patrón simple debemos indicar los siguientes parámetros:

- Nombre del módulo: es un agrupador de mantenimientos. Nos permite agrupar los mantenimientos de nuestra aplicación de forma que sea más sencillo su manejo.
- Clase Manejadora: el nombre de la clase.
- Tabla de la base de datos: tabla sobre la que queremos realizar el mantenimiento.
- Patrón: seleccionamos la forma de visualización de la información (Tabular, Registro o Tabular-Registro).

Pulsamos generar y ya tendremos nuestro mantenimiento creado.

Vamos a nuestra aplicación. Entramos de nuevo desde la validación (así borramos caches y cookies). Vemos que se ha creado una nueva entrada de menú con el módulo que hemos especificado y dentro el mantenimiento creado.

Para generar patrones maestro detalle debemos indicar los siguientes parámetros:

- Nombre del módulo: es un agrupador de mantenimientos. Nos permite agrupar los mantenimientos de nuestra aplicación de forma que sea más sencillo su manejo
- Sección Maestro:
    - Clase Manejadora: el nombre de la clase del maestro.
    - Tabla: tabla de la base de datos del maestro.
    - Clave Primaria: campos que componen la clave primaria del maestro.
    - Patrón: Seleccionamos la forma de visualizción del maestro (Tabular o Registro).
    - Numero de detalles: número de detalles que va a tener nuestra ventana.
- Sección Detalle:
    - Clase Manejadora: el nombre de la clase del detalle.
    - Tabla: tabla de la base de datos del detalle.

- Clave Ajena: campos que relacionan la tabla con el maestro. Corresponde con los campos que son clave ajena referenciada a los campos de la clave primaria de la tabla maestro. Si son varios, debe respetarse el orden que se ha indicado en el maestro.

- Patrón del detalle: Seleccionamos la forma de visualización del detalle (Tabular, Registro o Tabular-Registro).

Pulsamos generar y ya tendremos nuestro mantenimiento creado.

Vamos a nuestra aplicación. Entramos de nuevo desde la validación (así borramos caches y cookies). Vemos que se ha creado una nueva entrada de menú con el módulo que hemos especificado y dentro el mantenimiento creado.