Eusko Jaurlaritzaren Informatika Elkartea
Sociedad Informática del Gobierno Vasco

UDA – Utilidades de desarrollo de aplicaciones

Plugin UDA. Guía de uso de plantillas

Fecha: 06/06/2011

Referencia:

EJIE S.A.
Mediterráneo, 14
Tel. 945 01 73 00
Fax. 945 01 73 01
01010 Vitoria-Gasteiz
Posta-kutxatila / Apartado: 809
01010 Vitoria-Gasteiz
www.ejie.es

UDA – Utilidades de desarrollo de aplicaciones by *EJIE* is licensed under a *Creative Commons Reconocimiento-NoComercial-CompartirIgual 3.0 Unported License*

Control de documentación

Título de documento: Plugin UDA - Guía de Uso de Plantillas

Histórico de versiones

Código:	Versión:	Fecha:	Resumen de cambios:
	1.0.0	06/06/2011	Primera versión.
	1.1.0	14/09/2011	Cambios en la estructura y contenido del documento.

Cambios producidos desde la última versión

Cambios en la estructura y contenido del documento.

Control de difusión

Responsable: Ander Martinez

Aprobado por:

Firma: Fecha:

Distribución:

Referencias de archivo

Autor:

Nombre archivo:

Contenido

1 Introducción

UDA (Utilidades de Desarrollo de Aplicaciones) es el conjunto de utilidades, herramientas, librerías, plugins, guías, y recomendaciones funcionales y técnicas que permiten acelerar el proceso de desarrollo de sistemas software con tecnología Java impulsado por EJIE/EJGV. Sus principales señas de identidad frente a sus predecesores son la **simplicidad y productividad** en el desarrollo y sus capacidades de interfaz gráfico para la construcción de aplicaciones ricas (RIA, Rich Internet Applications).

UDA es un conjunto de utilidades de desarrollo para aplicaciones Java EE compuesto por

1. Normativa de uso: Define cómo encajar un conglomerado de frameworks, librerías y productos que han destacado por su éxito en el mundo Java EE para lograr la sinergia que favorezca la simplicidad de las aplicaciones y la robustez de la arquitectura.
2. Generadores: UDA proporciona distintos asistentes de tal manera que el programador parte de una base que ya funciona y cumple la normativa de uso de los diferentes componentes que se han integrado en UDA. Existen asistentes para:
    a. **Asistentes de generación de proyectos**: Es posible crear proyectos de dos tipos, según el de motor de persistencia deseado, dando opción a escoger entre JDBC o JPA 2.0. Una vez escogido el tipo de proyecto, UDA generará un proyecto de tipo EAR que estará vinculado a un proyecto JAR y a un proyecto WAR, obligatoriamente. Opcionalmente, se podrán añadir nuevos módulos WAR y EJB a esos proyectos iniciales.
    Los proyector EAR generados, están optimizados para ser desplegados en el servidor de aplicaciones Oracle WebLogic 11g automáticamente.
    b. **Asistentes de generación de código**: Partiendo de un proyecto UDA, se podrá autogenerar el código correspondiente partiendo de un esquema relacional de base de datos. UDA detectará automáticamente si el proyecto sobre el que se desea generar el código es de tipo JDBC o JPA. Basándose en el modelo relacional, generará el código necesario para realizar la gestión CRUD (Create, Read, Update and Delete) de dicho esquema relacional desde una aplicación JEE, con interfaces REST y RMI.
    c. **Asistentes de generación de vista**: Por último, una vez que se ha autogenerado el backend de la aplicación, UDA proporciona asistentes que generan automáticamente interfaces de usuario Web ricas (Rich Internet Application o RIA) que actúan como consumidores de servicios Web REST, proporcionando al usuario final interfaces atractivas y accesibles desde las que gestionar el esquema relacional cómodamente.

En el desarrollo de los asistentes agrupados en el plugin UDA se han utilizado diferentes estrategias para automatizar la generación del código.

En los siguientes apartados se describen cuáles son esas estrategias y los recursos necesarios para la generación del código.

Además, ante posibles cambios que pudieran surgir en la evolución de UDA, también se describe cómo modificar las plantillas de código que se utilizan en función de la estrategia de automatización.

2 Estrategias de generación de ficheros

El Plugin UDA, a través de diferentes asistentes que se ejecutan en el IDE de desarrollo (Helios Oepe), genera la estructura de proyectos y parte del código de los mismos atendiendo a las consideraciones de la arquitectura de aplicaciones.

Para la generación tanto de la estructura de los proyectos como del código se siguen diferentes estrategias:

- Uso de plantillas mediante exporters y FreeMarker, es la estrategia principal y más frecuentemente utilizada a la hora de generar los ficheros de código. Además, en aquéllos casos en los que es necesario obtener el modelado de base de datos, a parte de FreeMarker, se utiliza también Hibernate Tools.
- Copia de los ficheros, otra forma de generación, menos usada debido a que no se utilizan prácticamente ficheros estáticos. Es decir, desde la raíz de las plantillas se copia al proyecto el fichero estático. Se utiliza en reducidas ocasiones, como por ejemplo en el fichero “messages.properties”.
- Edición de ficheros, en algunas automatizaciones que posee UDA es necesario editar ficheros. En estos casos no se puede utilizar una plantilla debido a que ésta eliminaría toda la información existente. El responsable de añadir la información al fichero a generar será el propio asistente UDA que se instala en el eclipse. Un ejemplo de estas modificaciones es la inserción de una etiqueta en un XML.

A continuación se procede a explicar cada una de ellas más en profundidad.

2.1 Generación mediante exporters

En la generación de código desde el modelo de base de datos, UDA utiliza herramientas tales como Hibernate Tools o FreeMarker. Por ejemplo a la hora de generar la capa del modelo de datos (debido a que se genera un fichero por entidad con la información asociada a ella misma) o ciertos ficheros como puede ser el service-config.xml (al elegir generación mediante anotaciones) se realiza la generación mediante exporters.

Los exporter son una utilidad de Hibernate Tools que proporciona el acceso al modelado de base de datos pudiendo así realizar cualquier tipo de consulta sobre dicho él. A través de los exporter se puede realizar la lectura de la base de datos generando una clase por entidad correspondiente a la tabla origen obtenida de la base de datos.

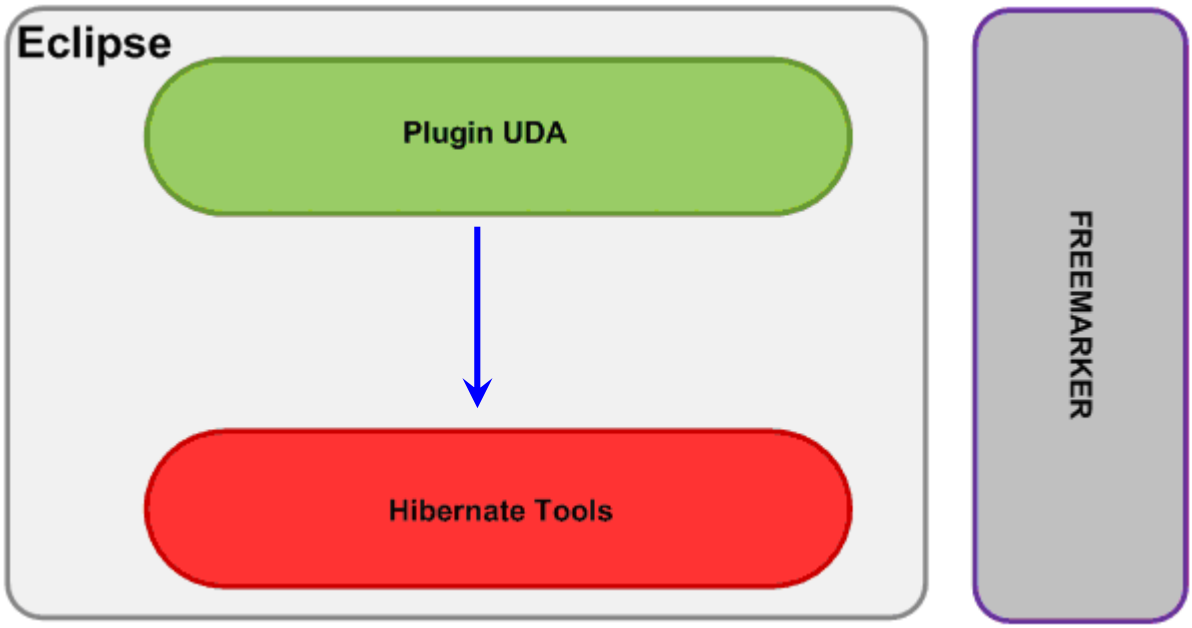

Ilustración 1. Arquitectura para la generación de código.

- La capa Plugin UDA, es la encargada de solicitar los parámetros al usuario, y desembocar el proceso de generación deseado. Internamente se apoya en unos exporters que mediante plantillas de FreeMarker generan el código de la aplicación.
- Otra de las capas es la denominada “Hibernate Tools”. Dicha capa es la responsable de la lectura del modelado de base de datos utilizado en varias capas de la arquitectura de las aplicaciones que se construirán haciendo uso del plugin UDA, tales como la capa de DAO, model, service o controller. Hibernate Tools, a su vez, proporciona un API para poder manipular dicha información.
- Y el último componente del esquema es FreeMarker que se usa para generar las plantillas.

El proceso que se desencadena en la invocación de la generación de código sigue los siguientes pasos:

1. Desde Eclipse se arranca el Asistente UDA.
2. El asistente de generación de código UDA, en función de la opción seleccionada, invoca a los exporters existentes.
3. Los exporters utilizan Hibernate Tools para obtener el modelado y a través de plantillas generar el código deseado.

2.2 Generación mediante FreeMarker

La generación exclusiva mediante FreeMarker ha sido utilizada en aquellos ficheros que no son estáticos y que esperan una serie de valores exclusivos para lo que se está generando; por ejemplo, cuando no es necesario tener una lectura de base de datos y se necesitan valores obtenidos (o calculados) durante la generación.

Los ficheros generados de esta manera pueden ser el fichero “build.xml” generado en el proyecto EARClasses o el fichero “weblogic.xml” generado en el war entre otros.

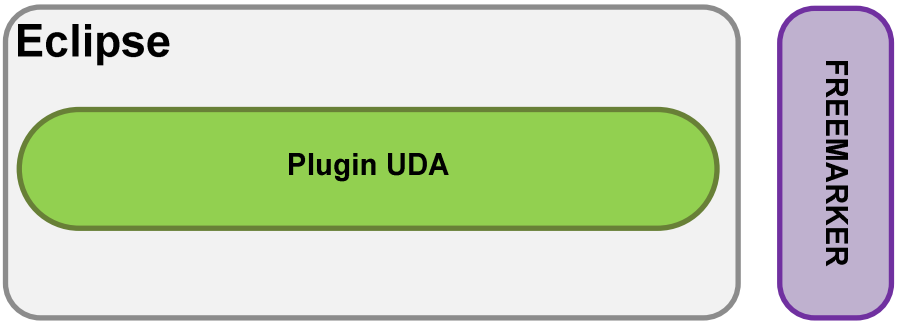

Ilustración 2. Arquitectura generación plantillas

La capa Plugin UDA, es la encargada de solicitar los parámetros al usuario, y genera el contexto FreeMarker (contexto necesario para el motor de FreeMarker). Una vez creado éste, realiza la llamada a la creación de plantillas.

La capa FreeMarker, nuevamente, se usa para generar las plantillas.

En este caso, el asistente UDA, genera un contexto de FreeMarker donde inyecta los valores deseados en las plantillas. Posteriormente invoca a las plantillas deseadas que extraerán del contexto los valores deseados.

2.3 Copia ficheros

La generación mediante copia de fichero se utiliza para aquellos ficheros que son estáticos. Es decir, ficheros comunes para todas las aplicaciones que no varían a medida que se genera código. La copia se realiza obteniendo el fichero en las plantillas y copiándolo en destino directamente.

Los ficheros generados mediante esta técnica son “beanRefContext.xml” o “log-config.xml” de los proyectos EARClasses entre otros.

2.4 Edición de ficheros

La edición de ficheros se aplica en casos en los que es necesario añadir algo de información por cada generación. Por ejemplo, en el fichero de “*nombreAplicacion*.properties”, que se ubica en el proyecto “Config” de nuestra aplicación, por cada Stub generado en un proyecto EJB se actualiza el fichero con los parámetros del servidor remoto.

A continuación se procede a explicar este caso:

Al generar una aplicación nueva, por ejemplo, x21b, se ha generado un módulo EJB denominado “x21bRemoting1EJB”. El fichero “application.xml” del proyecto EAR tiene el siguiente aspecto:

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:application="http://java.sun.com/xml/ns/javaee/application_5.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="Application_ID"
version="5" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/application_5.xsd">
  <display-name>x21bEAR</display-name>
  <module>
    <web>
      <web-uri>x21bX21bWAR.war</web-uri>
      <context-root>x21bX21bWAR</context-root>
    </web>
  </module>
```

```
<module>
        <ejb>x21bRemoting1EJB.jar</ejb>
    </module>
</application>
```

UDA permite añadir un módulo EJB nuevo a dicha aplicación (x21bRemotin2EJB). Mediante esta técnica de edición de ficheros, se logra añadir la información de dicho módulo quedando su nuevo aspecto:

```
<?xml version="1.0" encoding="UTF-8"?>
<application xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:application="http://java.sun.com/xml/ns/javaee/application_5.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" id="Application_ID"
version="5" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/application_5.xsd">
  <display-name>x21bEAR</display-name>
  <module>
    <web>
      <web-uri>x21bX21bWAR.war</web-uri>
      <context-root>x21bX21bWAR</context-root>
    </web>
  </module>
   <module>
        <ejb>x21bRemoting1EJB.jar</ejb>
    </module>
    <module>
        <ejb>x21bRemoting2EJB.jar</ejb>
    </module>
</application>
```

Tal y como se ha indicado, es el propio plugin UDA el encargado de inyectar dicha información en el fichero.

3 Plantillas

Las plantillas utilizadas en el proyecto son plantillas con extensión .ftl (FreeMarker Template Language). A fin de agilizar su búsqueda y modificación, se han agrupado en función de la generación que aportan al proyecto.

3.1 Generación de proyectos

En la generación de proyectos, se crean varios proyectos. A continuación se procede a mostrar el origen de las plantillas por proyecto generado:

- Proyecto EARClasses: las plantillas necesarias para la generación se encuentran, partiendo de la ruta configurada en el campo 'Preferences' en el eclipse, dentro de una carpeta denominada "earclasses".
- Proyecto WAR: las plantillas necesarias para la generación se encuentran, partiendo de la ruta configurada en el campo 'Preferences' en el eclipse, dentro de una carpeta denominada "war".
- Proyecto Config: las plantillas necesarias para la generación se encuentran, partiendo de la ruta configurada en el campo 'Preferences' en el eclipse, dentro de una carpeta denominada "config".
- Proyecto Statics: las plantillas necesarias para la generación se encuentran, partiendo de la ruta configurada en el campo 'Preferences' en el eclipse, dentro de una carpeta denominada "statics".
- Proyecto EJB: las plantillas necesarias para la generación se encuentran, partiendo de la ruta configurada en el campo 'Preferences' en el eclipse, dentro de una carpeta denominada "ejb".
- Proyecto EAR: las plantillas necesarias para la generación se encuentran, partiendo de la ruta configurada en el campo 'Preferences' en el eclipse, dentro de una carpeta denominada "ear".

3.2 Capa Presentación

3.2.1. Vista

En la generación del apartado Vista, dentro de la capa de presentación, las plantillas necesarias se obtienen de una carpeta denominada "maint", siempre partiendo de la ruta raíz que se encuentra configurada en el eclipse. Esta configuración se obtiene del menú Window → Preferences → UDA.

3.2.2. Control

En la generación del apartado Control, dentro de la capa de presentación, las plantillas necesarias se obtienen de una carpeta denominada "generateCode/controller", siempre partiendo de la ruta raíz que se encuentra configurada en el eclipse. Esta configuración se obtiene del menú Window → Preferences → UDA.

3.3 Capa Servicios de Negocio

En la generación del apartado Servicios de negocio, las plantillas necesarias se obtienen de una carpeta denominada "generateCode/service", siempre partiendo de la ruta raíz que se encuentra configurada en el eclipse. Esta configuración se obtiene del menú Window → Preferences → UDA.

En la capa de Servicios de negocio, se hace distinción entre las plantillas para tecnología Spring JDBC y JPA 2.0. Las plantillas utilizadas en Spring JDBC son aquellas que se encuentran en la subcarpeta “springJDBC” y las utilizadas en JPA las que se encuentran en la subcarpeta “JPA”.

Aquellas plantillas que no se encuentren en ninguna de las subcarpetas serán utilizadas por ambos.

3.4 Capa Acceso a Datos

En la generación del apartado capa de Acceso a datos, las plantillas necesarias se obtienen de una carpeta denominada “generateCode/dao”, siempre partiendo de la ruta raíz que se encuentra configurada en el eclipse. Esta configuración se obtiene del menú Window → Preferences → UDA.

En la capa de Acceso a datos, se hace distinción entre las plantillas para tecnología Spring JDBC y JPA 2.0. Las plantillas utilizadas en Spring JDBC son aquellas que se encuentran en la subcarpeta “springJDBC” y las utilizadas en JPA las que se encuentran en la subcarpeta “JPA”.

3.5 Capa Modelo de datos

En la generación de la capa de Modelo, las plantillas necesarias se obtienen de una carpeta denominada “generateCode/model”, siempre partiendo de la ruta raíz que se encuentra configurada en el eclipse. Esta configuración se obtiene del menú Window → Preferences → UDA.

En la capa de Modelo de datos, se hace distinción entre las plantillas para tecnología Spring JDBC y JPA 2.0. Las plantillas utilizadas en Spring JDBC son aquellas que se encuentran en la subcarpeta “springJDBC” y las utilizadas en JPA las que se encuentran en la subcarpeta “JPA”.

Aquellas plantillas que no se encuentren en ninguna de las subcarpetas serán utilizadas por ambos.

3.6 Capa Remoting

3.6.1. Skeleton

En la generación de la capa de Remoting, al generar el Skeleton más concretamente, las plantillas necesarias se obtienen de una carpeta denominada “ejb”, siempre partiendo de la ruta raíz que se encuentra configurada en el eclipse. Esta configuración se obtiene del menú Window → Preferences → UDA.

Las plantillas que intervienen en su generación son las que poseen el literal “Skeleton” en su nombre.

3.6.2. Stub

En la generación de la capa de Remoting, al generar el Stub más concretamente, las plantillas necesarias se obtienen de una carpeta denominada “ejb”, siempre partiendo de la ruta raíz que se encuentra configurada en el eclipse. Esta configuración se obtiene del menú Window → Preferences → UDA.

Las plantillas que intervienen en su generación son aquellas que poseen el literal “Stub” en su nombre.

4 Modificación plantillas

La modificación de las plantillas, en base a la estrategia usada para su generación, requiere diferentes normas de actuación.

4.1 Generación mediante exporters

La generación a través de FreeMarker se realiza mediante ficheros ".ftl".

Se trata de ficheros de texto plano en los que aparecen literales similares a ${xxxx}. Estos literales indican que lo que se está usando, en este caso, es el contexto de FreeMarker para obtener dicha variable ("xxxx" en este caso).

Por otra parte, los literales con el siguiente aspecto ${xxx.yyy} pueden ser o bien invocaciones a llamadas de funciones de Hibernate Tools o bien llamadas a funciones existentes en el plugin.

4.1.1. Modificación de texto plano

El código no perteneciente a FreeMarker, al ser texto plano, se puede modificar sin mayor problema. Los cambios, en este caso, se reflejarán en la posterior utilización de UDA.

Por ejemplo, si se desea cambiar un método public a private, la modificación quedaría de la siguiente manera:

```
public ${pojo.getDeclarationName()} add(${pojo.getDeclarationName()}
${ctrTl.stringDecapitalize(pojo.getDeclarationName())}) {
      return
this.${nombreDao}.add(${ctrTl.stringDecapitalize(pojo.getDeclarationName())});
    }
```

Simplemente es sustituir el public por el private, y cambiará la generación en los service.

```
private ${pojo.getDeclarationName()} add(${pojo.getDeclarationName()}
${ctrTl.stringDecapitalize(pojo.getDeclarationName())}) {
      return
this.${nombreDao}.add(${ctrTl.stringDecapitalize(pojo.getDeclarationName())});
    }
```

4.1.2. Modificación de código Freemaker

El código perteneciente a FreeMarker, sin embargo, requiere el API de FreeMarker. Por ello, los cambios deben seguir el lenguaje de dicha herramienta.

Hay que tener especial cuidado con las variables (aquellas que van entre ${…}), debido a que estas deben existir en el contexto que se transfiere desde UDA a FreeMarker. Una variable inexistente produce un error en la generación de plantillas provocando así que no se generen.

En el siguiente ejemplo, se muestra un caso de invocación de un método de Hibernate Tools desde la plantilla, y por otra parte se observa la utilización de una función del plugin inyectada a la plantilla de FreeMarker:

```
public ${pojo.getDeclarationName()} add(${pojo.getDeclarationName()}
${ctrTl.stringDecapitalize(pojo.getDeclarationName())}) {
     return
this.${nombreDao}.add(${ctrTl.stringDecapitalize(pojo.getDeclarationName())});
    }
```

En este caso, aquellas funciones con el formato ${pojo.xxx} son funciones de Hibernate, mientras que ${ctrTL.xxx} es una utilidad que dispone el plugin. Concretamente, el ejemplo realiza la labor de decapitalizar la cadena que se le pasa.

El contexto de FreeMarker, es un contexto calculado por el asistente UDA, y es el propio asistente el que invoca la llamada a FreeMarker, pasándole el contexto anteriormente calculado. Si se desea introducir un campo nuevo al usuario, que se plasme en este tipo de plantillas, habrá que “inyectarla” en el contexto de FreeMarker. Debido a que es el asistente UDA el que lanza las plantillas, está ampliación implica la modificación del propio asistente UDA.

Tanto si se desea introducir una variable en el contexto de FreeMarker, como si se desea introducir una clase de utilidades o modificar las existentes, se deberá modificar el asistente UDA. La modificación de este asistente implica que la calidad del código que se genera, tanto como los posibles errores resultantes, pasa a ser responsabilidad de la persona que haya realizado la modificación.

4.1.3. Creación de plantillas nuevas mediante exporters

Todo lo implementado en el plugin, se puede obtener mediante la el API que ofrece Hibernate Tools. En las plantillas FreeMarker, existe la posibilidad de usar directamente dicho API, y así poder lograr la creación de las plantillas deseadas.

Hibernate Tools trabaja con el objeto POJO en el que está mapeada la entidad que se está generando el momento, desde el cual se puede obtener el mapeo completo de dicha entidad (sus relaciones, campos, tipos,…).

Posteriormente, se utiliza FreeMarker para poder editar, almacenar,… toda esta información pudiendo hacer a nivel de plantilla todo lo que se desee.

Un ejemplo de esto puede ser listar en la misma plantilla todas las propiedades de la entidad que se está generando. Para ello, la estructura utilizada tiene el siguiente aspecto:

```
<#assign classbody>
     …
</#assign>

${pojo.generateImports()}
${classbody}
```

Dentro de la etiqueta <#assign classbody> se debe meter todo aquel código que se desea generar.

En este caso, para listar todas las propiedades que dispone una entidad, se realiza de la siguiente forma:

```
<#assign classbody>
    <#foreach property in pojo.getAllPropertiesIterator()>
              ${property}
    </#foreach>
</#assign>

${pojo.generateImports()}
${classbody}
```

Mediante la combinación de API HibernateTools y Freemaker, como arriba se indicaba, se puede realizar todo el desarrollo para este tipo de plantillas logrando cualquier tipo de generación deseada.

4.2 Generación mediante FreeMarker

La generación a través de FreeMarker se realiza mediante ficheros ".ftl". Este caso es para aquellas plantillas que utilizando FreeMarker, no utilizan Hibernate Tools ni sus exporters. Se trata de ficheros que no necesitan modelado de base de datos en su generación.

En estos ficheros de texto plano aparecen literales similares a ${xxxx} o <#xxxx>. Estos literales indican que lo que se está usando es el contexto de FreeMarker para obtener dicha variable ("xxxx" en este caso).

El código no perteneciente a FreeMarker, al ser texto plano, se puede modificar sin mayor problema. Los cambios, en este caso, estarán reflejados en la posterior utilización de UDA.

El código perteneciente a FreeMarker, sin embargo, requiere el api de FreeMarker. Por ello, los cambios deben seguir el lenguaje de dicha herramienta.

En este caso, la modificación se realizará siguiendo las pautas que se indican en el punto 4.1.2.

4.3 Copia ficheros

La copia de ficheros siempre se realiza a través de plantillas estáticas. El fichero es un fichero de texto normal y corriente, como puede ser el fichero de internacionalización. No es un fichero con extensión .ftl, sino es un fichero que se copia directamente de origen a destino. En éstos no entra en juego ninguna peculiaridad de la propia aplicación ni siquiera el código de aplicación en los nombres de los ficheros.

Por ello, la modificación de la plantilla se realizará como cualquier otro fichero de texto. Al cambiar el código de la plantilla, los resultados se reflejarán en la posterior generación realizada mediantel el plugin UDA.

4.4 Edición de ficheros

En los casos en los que no es necesaria la generación de ficheros, sino que lo único que se necesita es la inserción de parte de él, se utiliza la edición de ficheros. Esta funcionalidad está implementada en el propio plugin UDA que se instala en eclipse, y es él en encargado de realizarla.

Un caso claro, es la inserción de una etiqueta XML en un fichero existente, tal y como se explica en el apartado 2.4 de este documento.

Por ello, modificar el código que genera cualquier asistente del plugin UDA instalado en el Eclipse supone modificar el propio motor del plugin. La modificación de este asistente implica que la calidad del código que se genera, tanto como los posibles errores resultantes, pasa a ser responsabilidad de la persona que haya realizado la modificación.

5 Bibliografia

- Hibernate Tools: http://docs.jboss.org/tools/3.0.0.CR1/en/hibernatetools/html_single/index.html
- POJOClass Api: http://www.docjar.com/docs/api/org/hibernate/tool/hbm2x/pojo/POJOClass.html
- FreeMarker: http://fmpp.sourceforge.net/FreeMarker/alphaidx.html