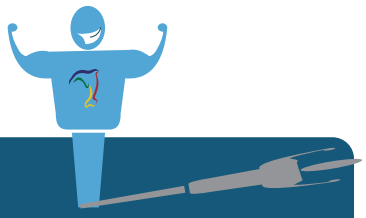
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

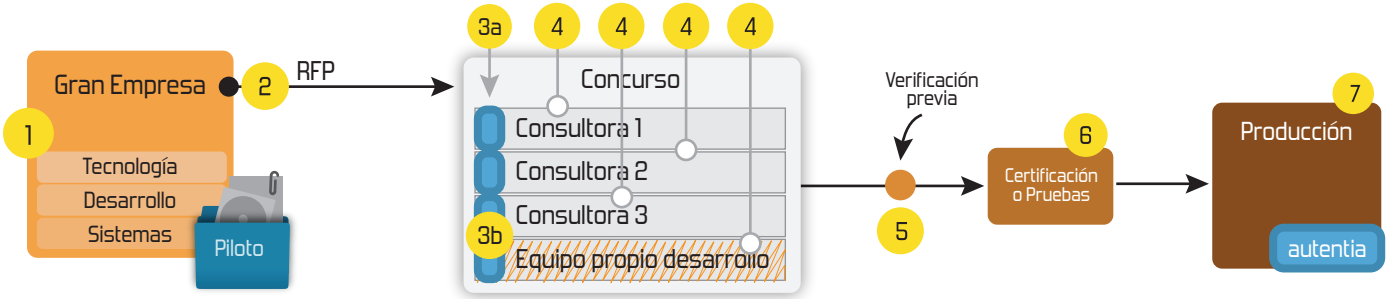
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Tutorial desarrollado por: Alejandro Perez García 2003-2007
Alejandro es Socio fundador de Autentia y nuestro experto en J2EE, Linux y optimización de aplicaciones empresariales.

Si te gusta lo que ves, **puedes contratarle** para impartir **cursos presenciales** en tu empresa o para ayudarte en proyectos (Madrid).

Contacta:
alejandropg@autentia.com

Fecha de creación del tutorial: 2007-07-13

BOUML,

una herramienta CASE de UML gratuita

Creación: 12-07-2007

Índice de contenidos

1. Introducción

Habitualmente suelo utilizar el UML en las aplicaciones que construimos en Autentia (www.autentia.com). El uso que le suelo dar es para usarlo como herramienta de análisis y diseño que me ayude a descubrir nuevos aspectos del sistema y debatir estos con mis compañeros, o para documentar ciertas partes importantes del sistema.

Para realizar este tipo de tareas suelo utilizar el ArgoUML (http://argouml.tigris.org/): herramienta Java gratuita (licencia BSD http://opensource.org/licenses/bsd-license.php), que, aunque sencilla, cubre mis necesidades (además tiene una cosa bastante curiosa que es que te va criticando el modelo intentando descubrir fallos o deficiencias).

Pero en este tutorial no os quiero hablar del ArgoUML, os quiero hablar del BOUML (http://bouml.free.fr/). Esta también es una herramienta CASE gratuita (licencia GPL) que he descubierto hoy y que me parece una muy buena alternativa porque:

- Permite trabajar con UML 2 (ArgoUML todavía no lo permite).
- Soporta gran cantidad de diagramas (incluidos los de secuencia que en el ArgoUML funcionan una versión si y otra no, a ver si terminan de estabilizarlo ;)
- Es rápida y apenas consume memoria.
- Es sencilla de utilizar.
- Puedes generar código para Java, C++ e IDL (y controlar bastante la generación), y puedes hacer reingeniería inversa (a partir del código sacar el modelo).
- También es capaz de generar documentación en varios formatos (HTML, XMI, ...)
- Puedes trabajar en grupo con sus módulos "Project Control" y "Project Synchro".

Y además, aunque no es Java, también es multiplataforma: Linux, MacOS y Windows.

En definitiva, todas estas características y su bajo precio (0 :) la convierten en una alternativa por lo menos digna de evaluar (ya veremos que nos dice el tiempo y el uso ;)

2. Entorno

El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Asus G1 (Core 2 Duo a 2.1 GHz, 2048 MB RAM, 120 GB HD).
- Sistema Operativo: GNU / Linux, Debian (unstable), Kernel 2.6.21, KDE 3.5
- BOUML 2.29

3. Instalación

Como siempre, en Debian, la instalación resulta sumamente sencilla. Basta con hacer:

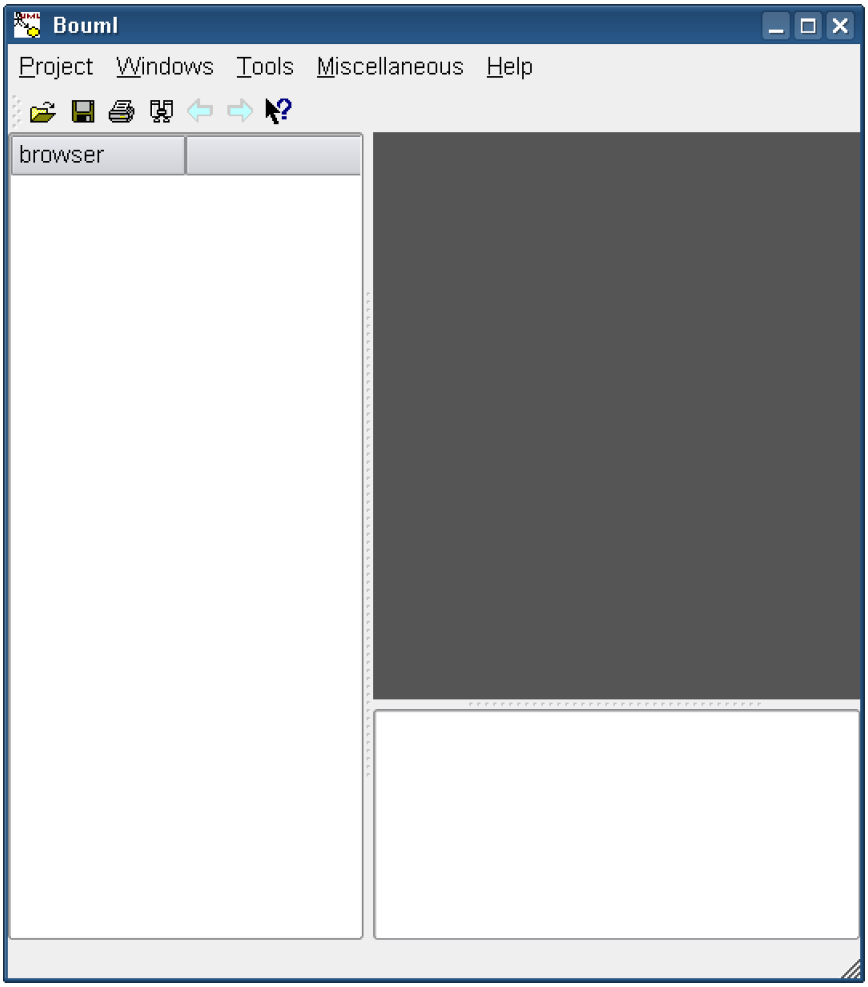
```
# apt-get bouml bouml-plugouts-src
```

Si tenemos otro sistema operativo, siempre podemos ir a la página http://bouml.free.fr/download.html y descargar la versión que nos interese.

En http://bouml.free.fr/ además también podemos encontrar mucha documentación que está bastante bien, y otros tutoriales (en ingles y francés).

4. Creando un proyecto

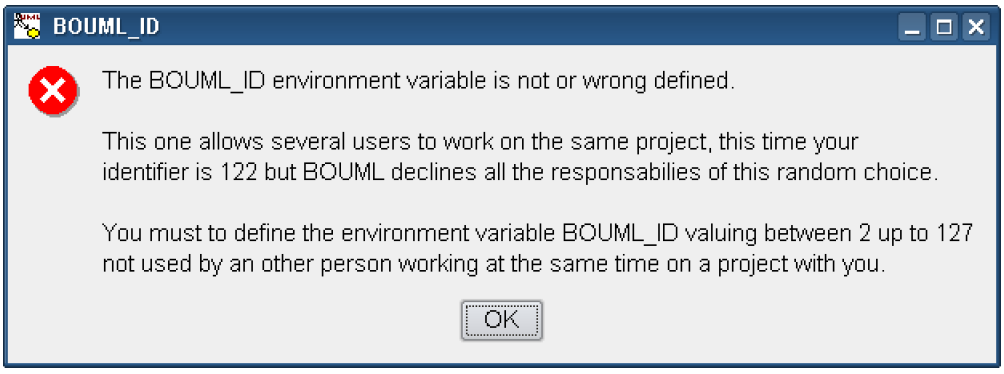
Cuando ejecutemos BOUML nos encontraremos con algo como:



A la derecha será donde nos aparezca el proyecto y el modelo que vayamos creando, a la izquierda nos aparecerán los distintos diagramas, y en la parte de abajo podremos editar la documentación del elemento que esté seleccionado.

Para crear el proyecto podemos hacer: *Project --> New*, y nos aparecerá un diálogo preguntando el directorio donde queremos guardar el proyecto. Yo le digo "autentia-project".

Veremos que nos saca la siguiente advertencia:

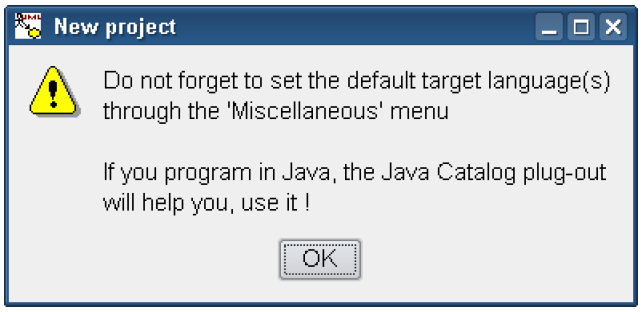


Con esto, BOUML, nos está indicando que no tenemos definida la variable de entorno `BOUML_ID`, esta variable de entorno representa el id del desarrollador para trabajar de forma simultánea sobre el mismo modelo. Es decir, cada desarrollador que trabaje de forma simultánea sobre el mismo modelo debería tener un `BOUML_ID` diferente.

Ahora no nos hace falte definir la variable (estoy probando yo solo ;) pero si queremos trabajar en grupo y queremos usar las herramientas "Project Control" y "Project Synchro", será necesario.

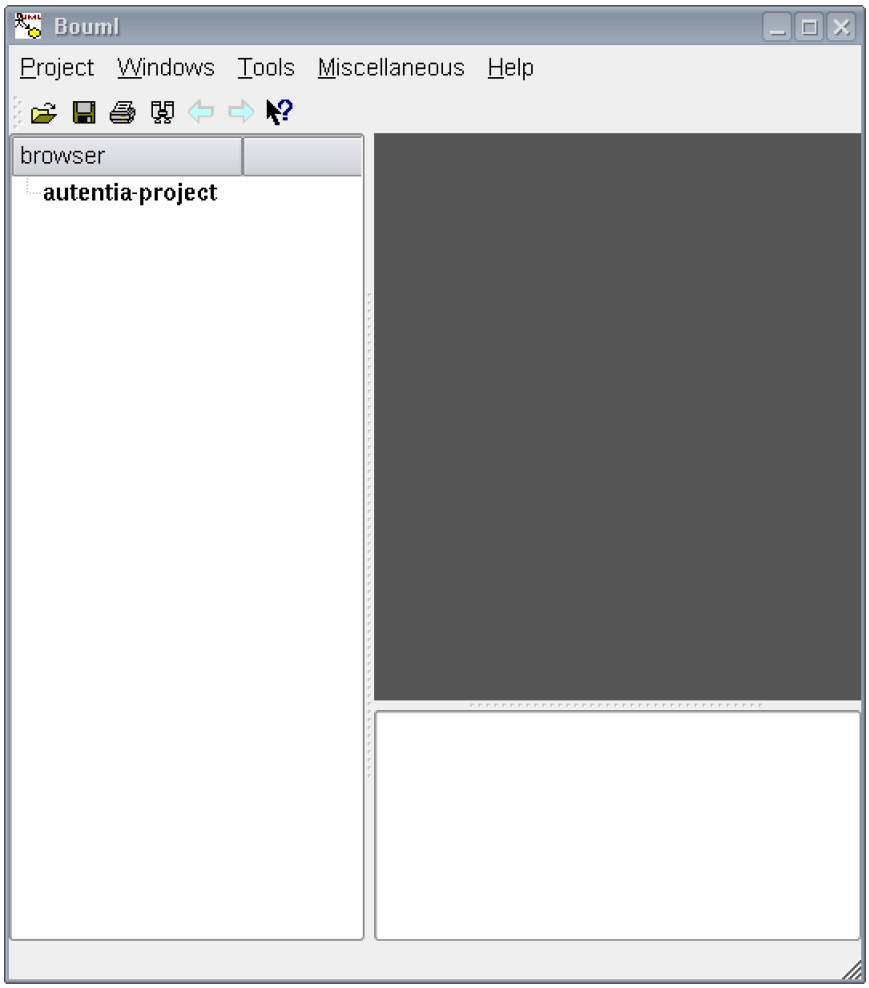
Por ahora le damos a OK y BOUML habrá elegido un número al azar como `BOUML_ID`.

A continuación nos presenta otra alerta:



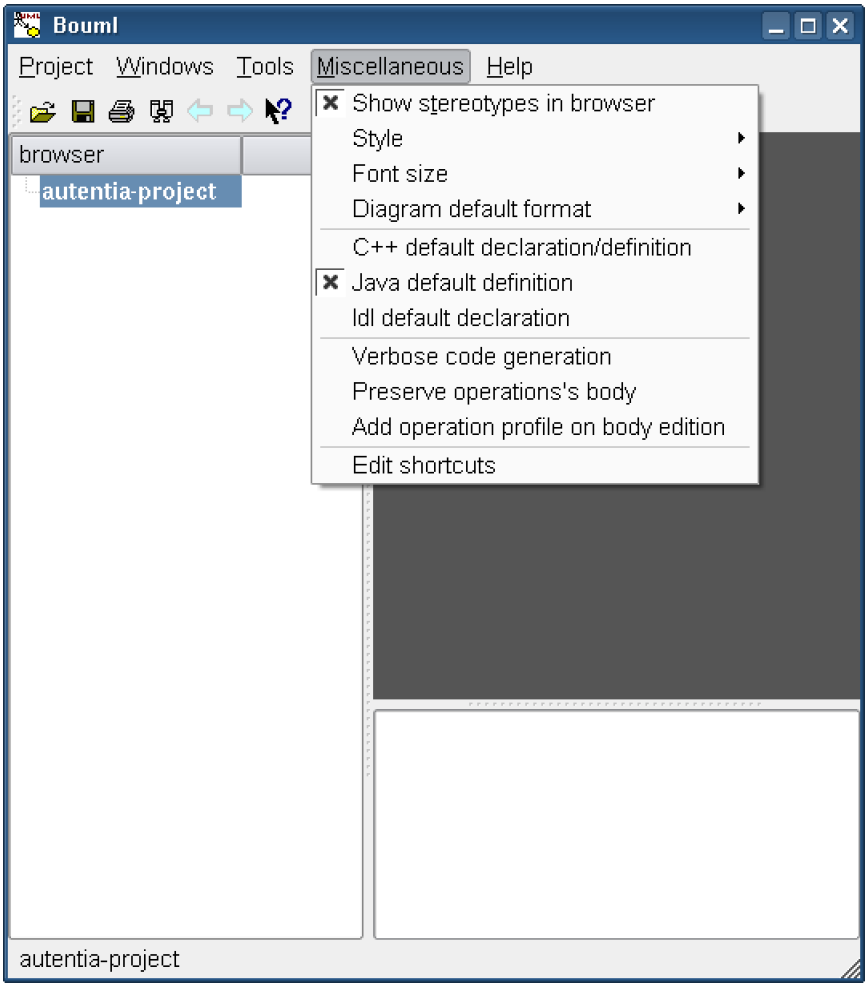
Con esto nos está recordando que tenemos que indicar los lenguajes destino para los que vamos a generar código.

Le damos a OK, y nos aparece la siguiente:



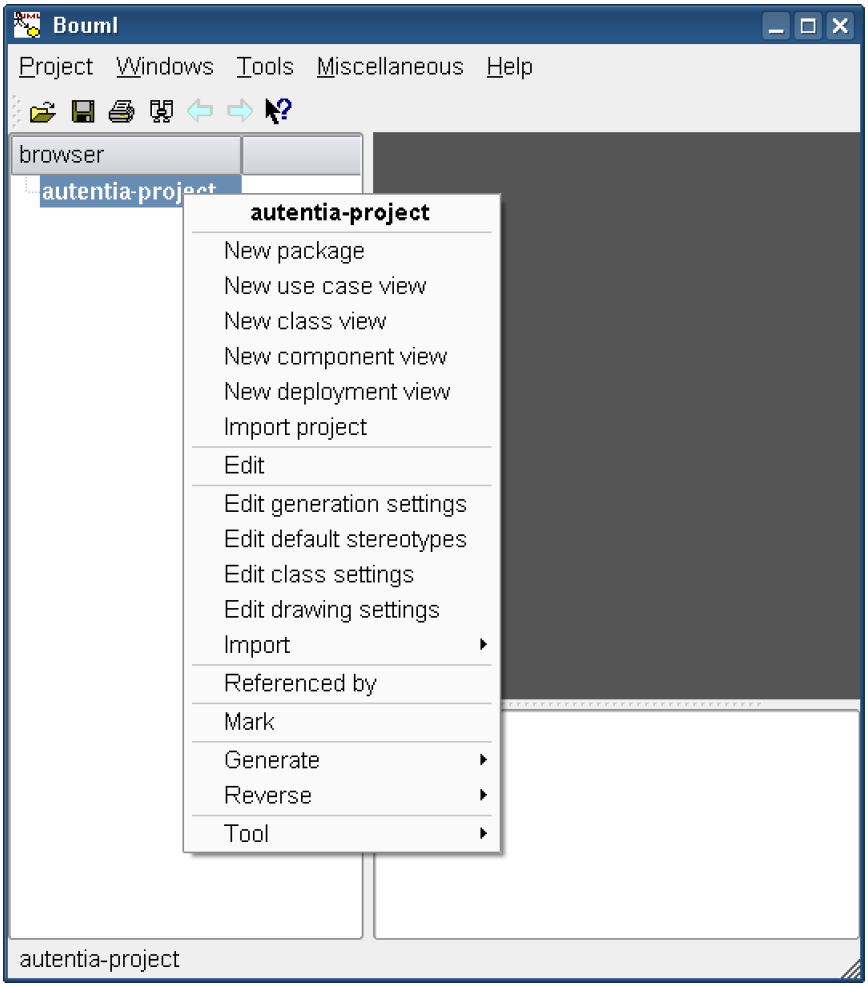
Vemos que hemos vuelto a la primera pantalla, pero ya tenemos el proyecto creado con el mismo nombre que le habíamos dado al directorio.

Lo primero que vamos a hacer es lo que nos decía la segunda advertencia: fijar los lenguajes para los que se va a generar código. Así que, como nos decía la alerta, pinchamos sobre el menú *Miscellaneous*, y seleccionamos los que nos interesen. En mi caso he marcado *Java default definition*.

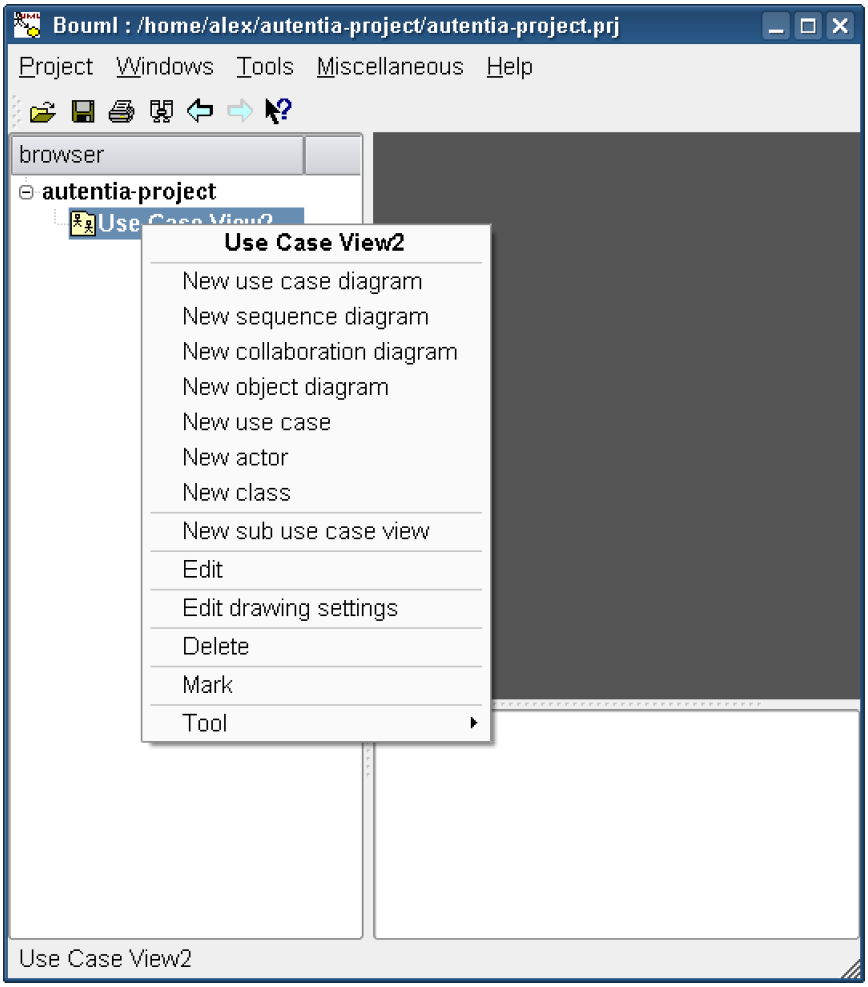


5. Creando un diagrama de casos de uso

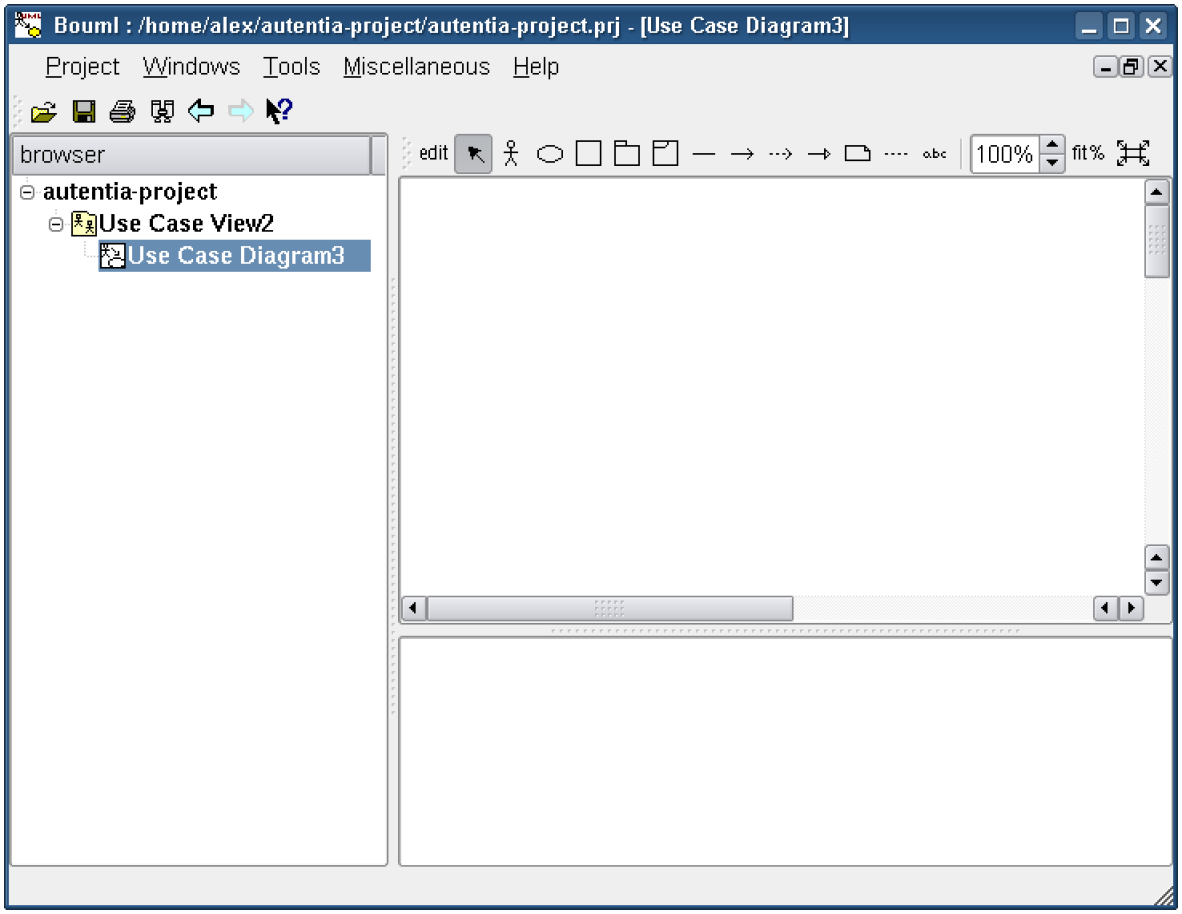
Sobre el nombre del proyecto pinchamos con el botón derecho y nos aparece el siguiente menú:



Vemos como las primeras opciones nos permiten crear paquetes para organizar nuestro modelo, y diferentes vistas (podríamos decir que son diferentes puntos de vista sobre nuestro modelo). Elegimos *New use case view*. Sobre el elemento que nos ha aparecido pulsamos de nuevo botón derecho y seleccionamos la opción *New use case diagram* para crear un nuevo diagrama de casos de uso (si pincháramos sobre *Edit* podríamos cambiar el nombre al elemento)

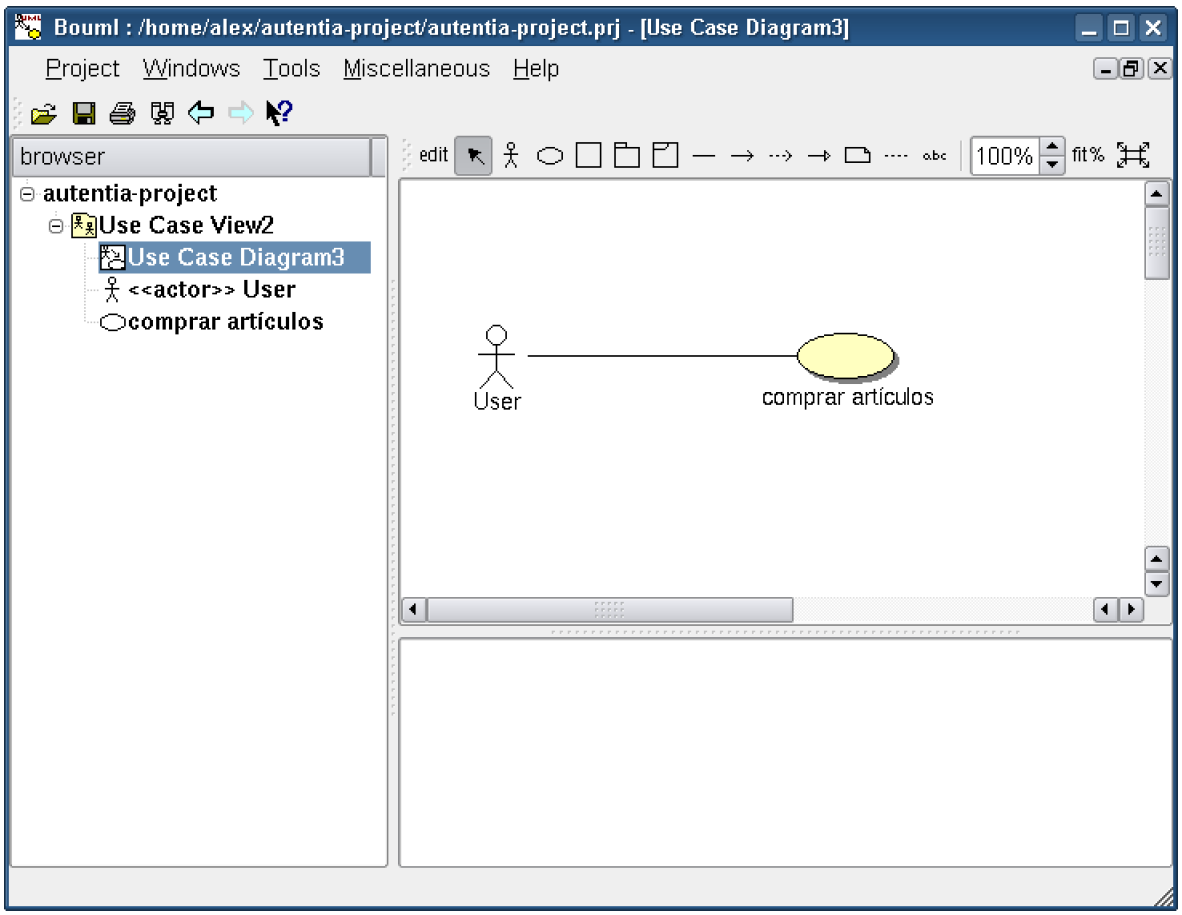


Sobre el nuevo elemento que se ha creado, hacemos doble click y a la derecha nos aparecerá la ventana para pintar el diagrama:



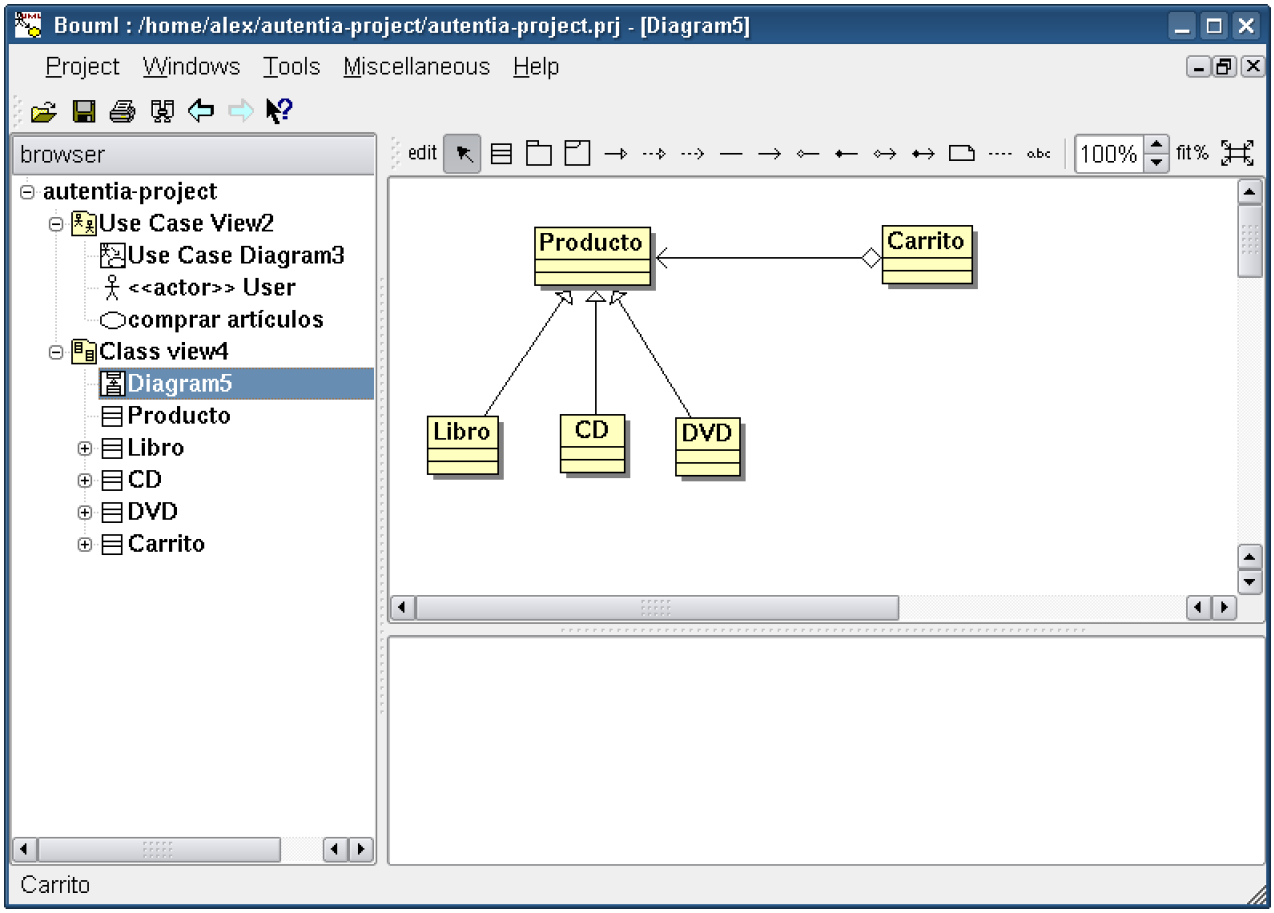
Vemos como arriba tenemos una "paleta" con los posibles elementos que podemos pintar en nuestro diagrama. Según vamos dibujamos

los diferentes elementos, estos aparecen representados en el modelo de la izquierda (con doble click sobre ellos, ya sea en el modelo o en el diagrama, podemos editar sus propiedades).

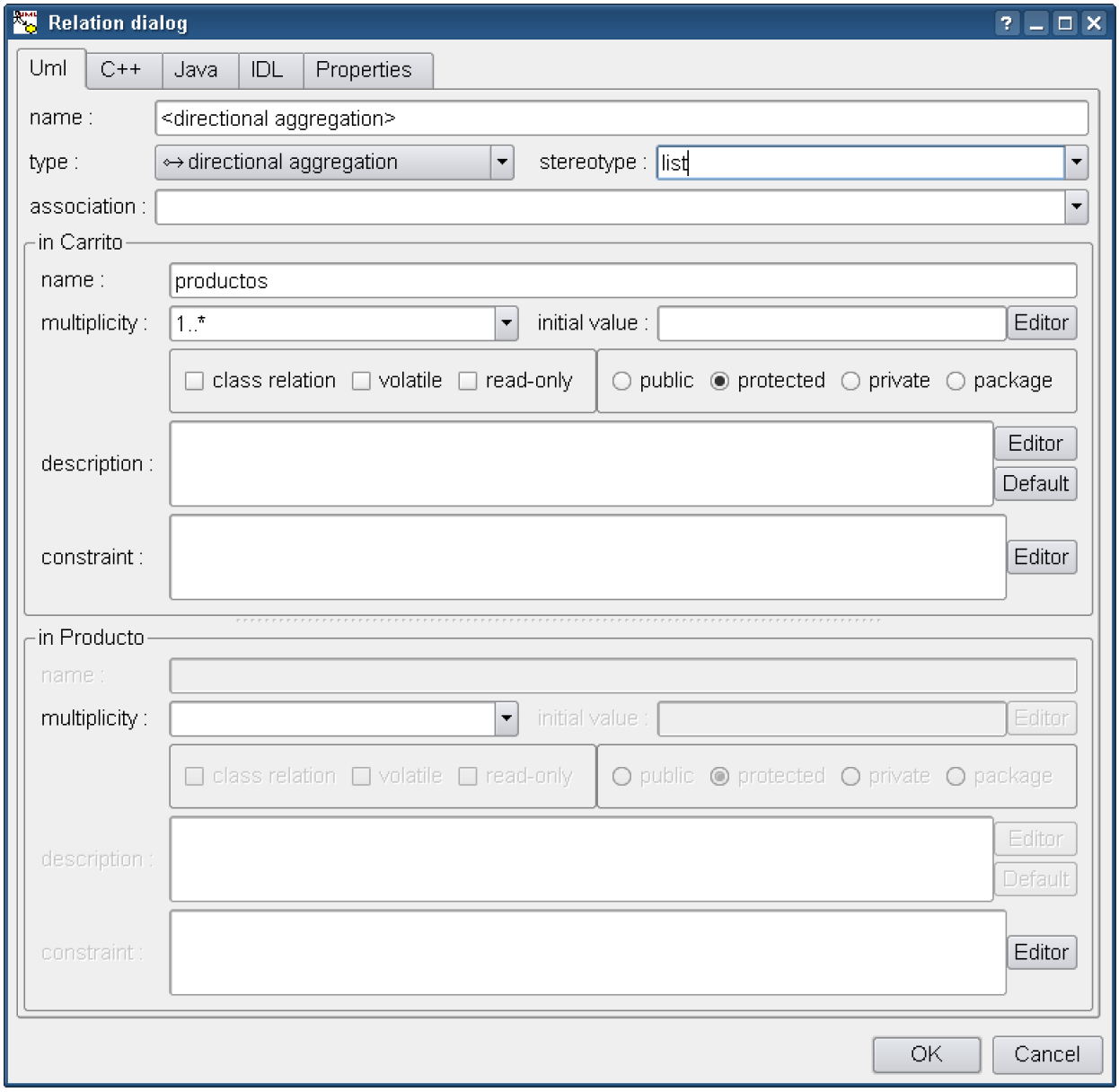


6. Creando un diagrama de clases

De forma similar a como hemos hecho antes, sobre el proyecto pulsamos botón derecho, y en el menú seleccionamos *New class view*. Ahora sobre el elemento que nos ha aparecido pulsamos con el botón derecho y seleccionamos la opción *New class diagram*. Sobre el nuevo elemento que nos ha aparecido hacemos doble click y a la derecha nos aparecerá la ventana para pintar el diagrama de clases, con la correspondiente paleta de elementos en su parte superior.



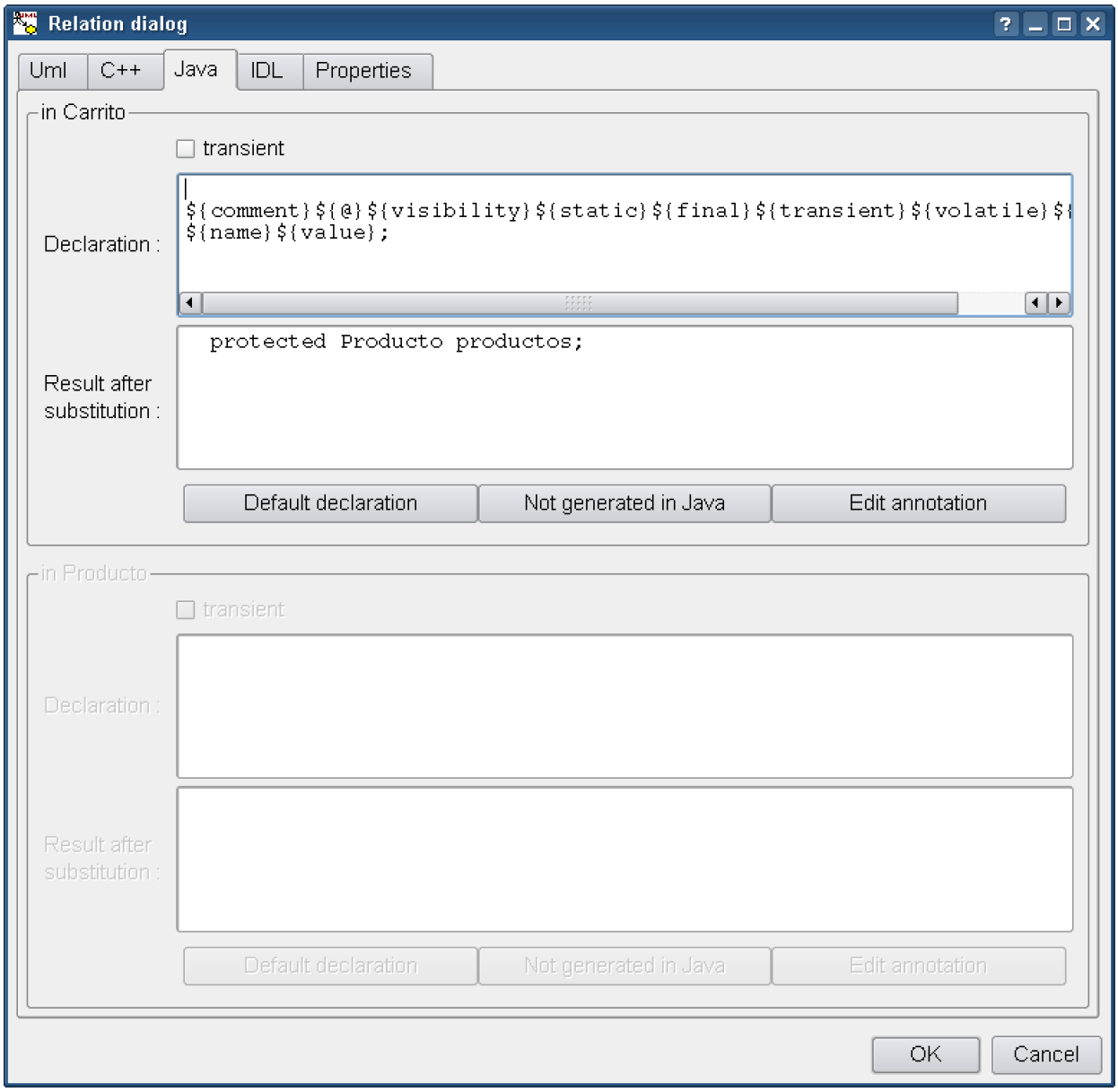
Fijémonos sobre la agregación entre Carrito y Producto. Si hacemos doble click sobre esta relación nos aparecerá una ventana donde podremos cambiar las propiedades de la relación:



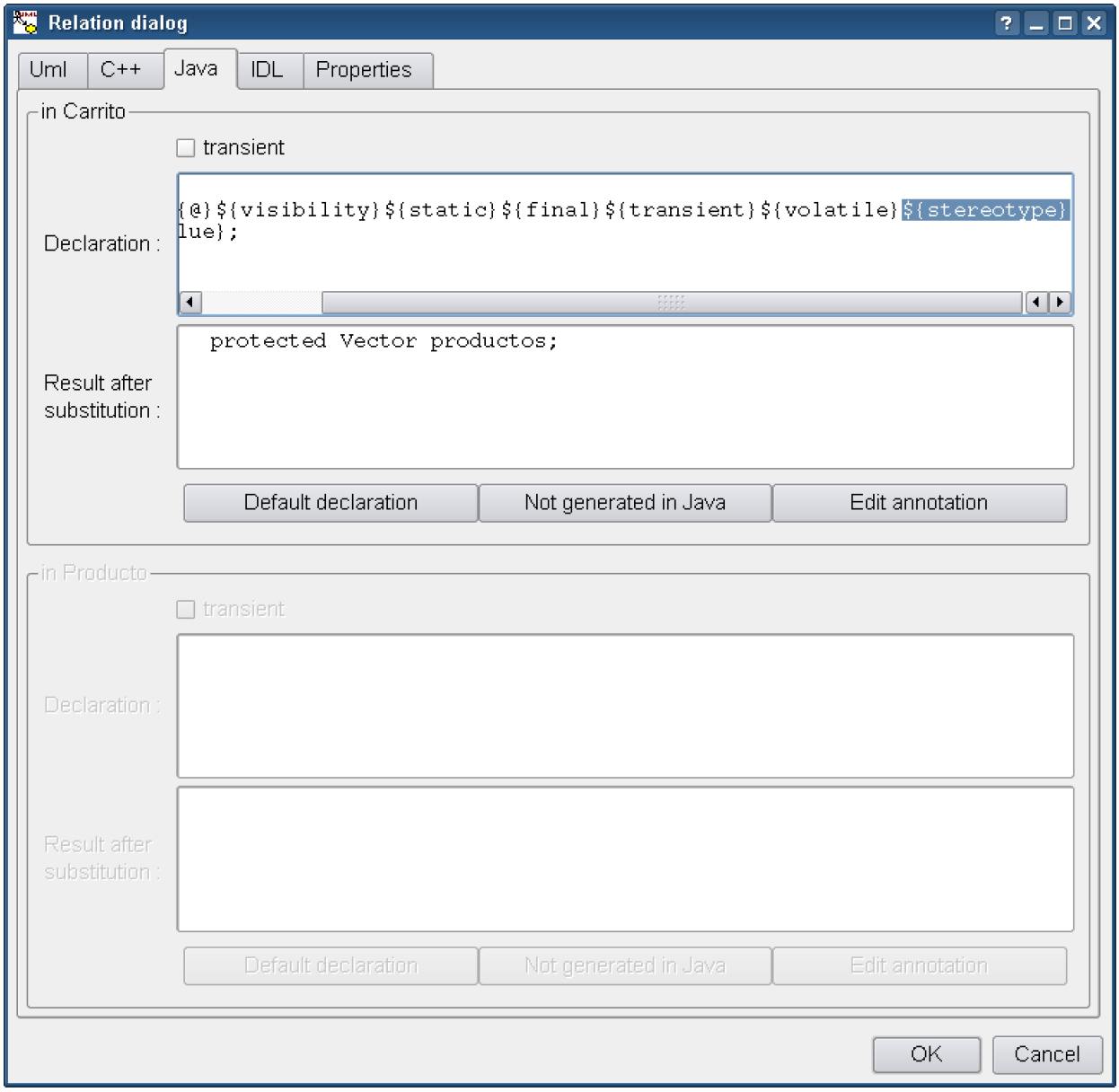
Hemos indicado que el *stereotypo* es *list*, y el nombre que tendrá el atributo en la clase Carrito y la multiplicidad.

7. Modificando como se genera el código

Si en la ventana anterior pinchamos sobre la pestaña Java, veremos el código que va a generar:

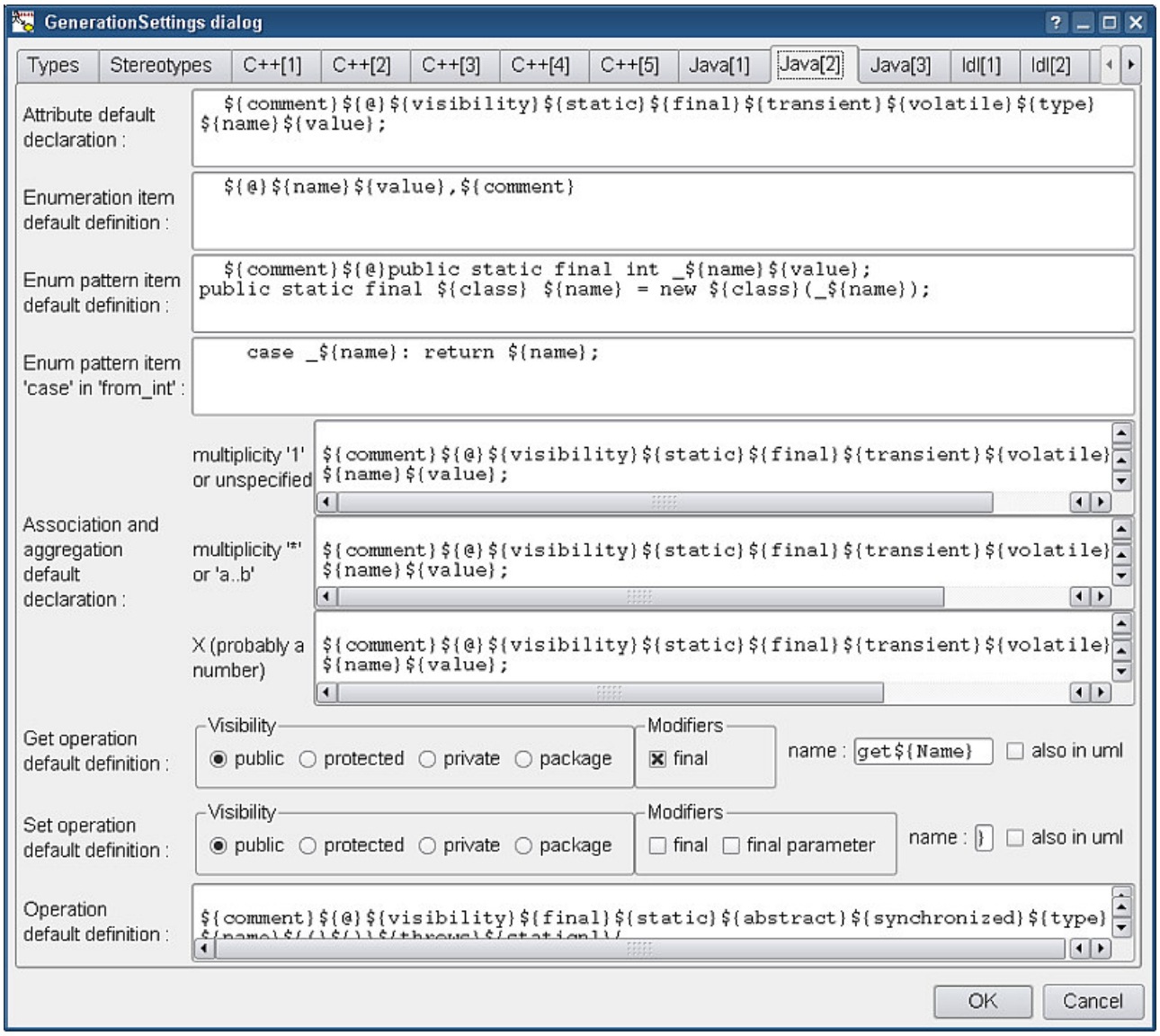


El código no me convence ya que le he dicho que el *stereotype* es *list* y que la multiplicidad es *1..**, y a pesar de todo me está declarando una variable de tipo Producto, en vez de una lista como sería de esperar. Pinchemos sobre *Default declaration* y veamos como cambia:



Vemos que ahora le ha añadido la etiqueta *${stereotype}* y que el resultado es un atributo de tipo Vector. Hemos mejorado, pero todavía no tenemos el código que nos gustaría: quiero sintaxis de Java 5 y ¿por qué Vector si le indiqué *list*?

Para definir como se genera el código lo podríamos hacer en esta ventana cambiando la macro de la opción *Declaration*. Pero este cambio sería local a esta agregación, y quiero que el cambio afecte a todas las asociaciones. Así que debemos hacer el cambio en *Project --> Edit --> Edit generation settings*, y pinchamos sobre la pestaña *Java[2]*:

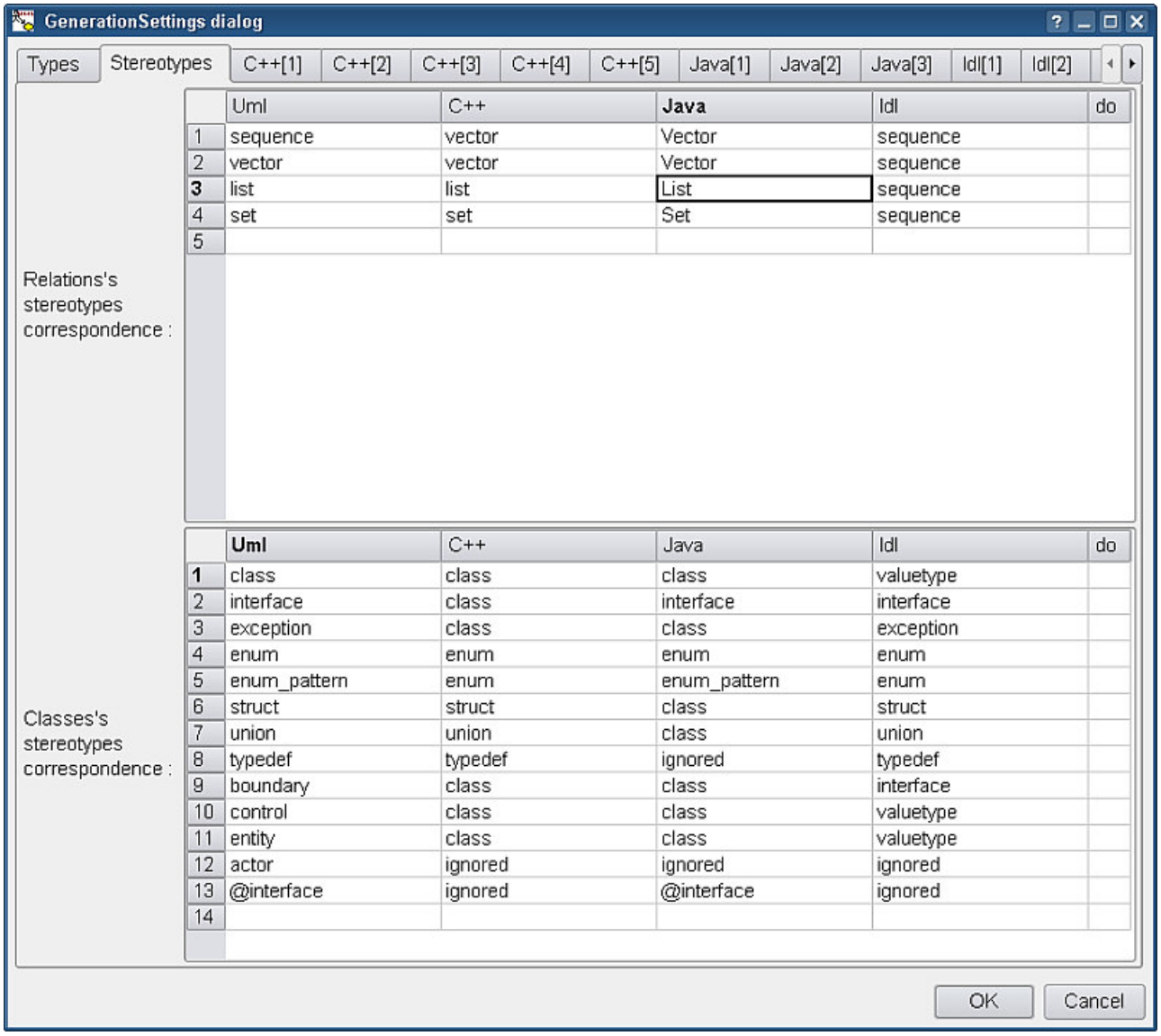


En el cuadro de texto *multilicity '*' or 'a..b'* es donde se define el código que nos interesa. Lo vamos a cambiar por:

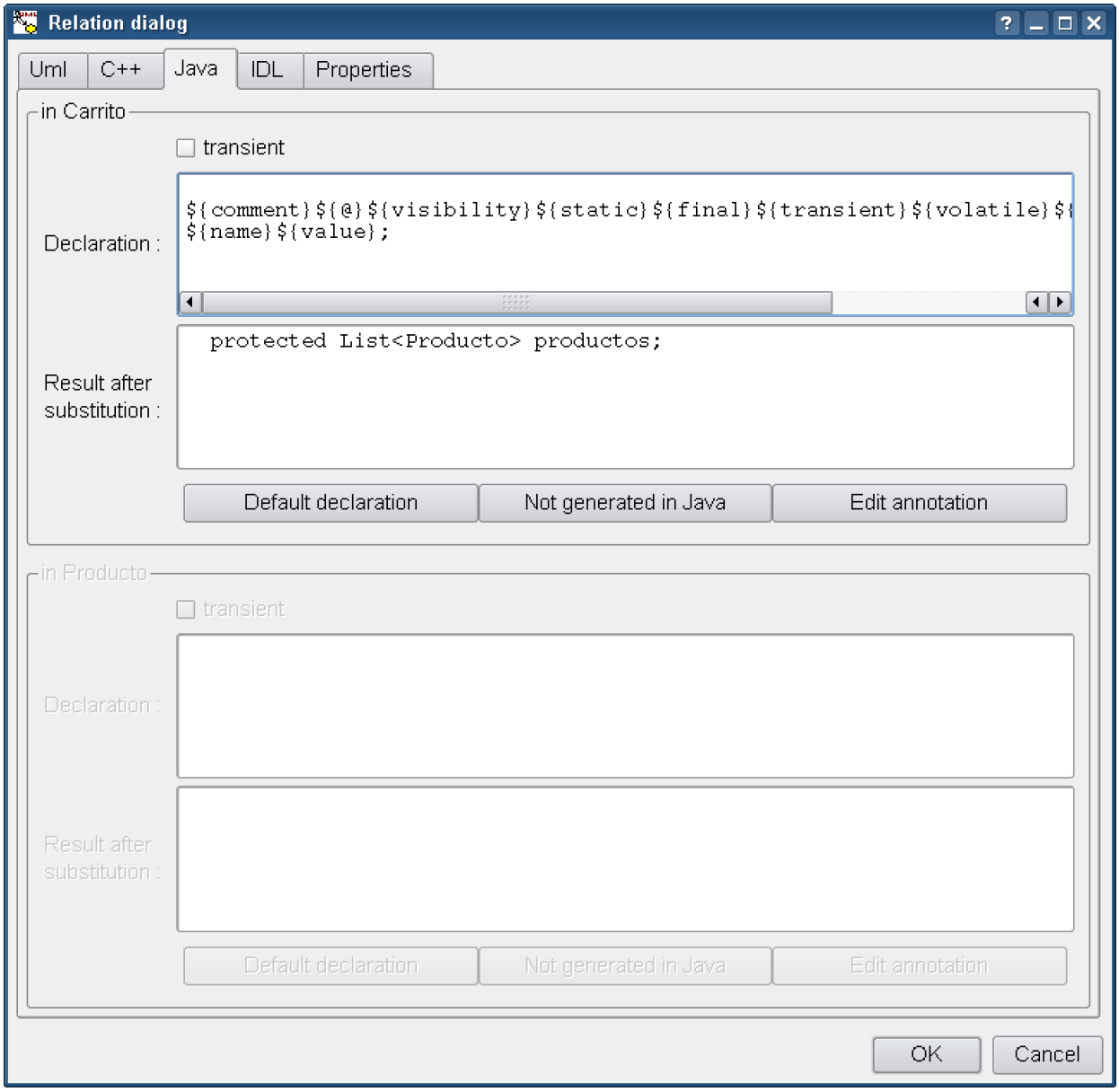
```
${comment}${@}${visibility}${static}${final}${transient}${volatile}${stereotype}<${type}> ${name}
${value};
```

En negrita he remarcado lo que he añadido para conseguir la sintaxis de Java 5.

Ahora, en este misma ventana vamos a pinchar sobre la pestaña *Stereotypes*. En esta pestaña es donde se indica como se debe interpretar el *stereotype list* en cada uno de los lenguajes. Vemos que en Java está todo como Vector, así que lo cambiamos por List (también aprovechamos y cambiamos *set* por Set):



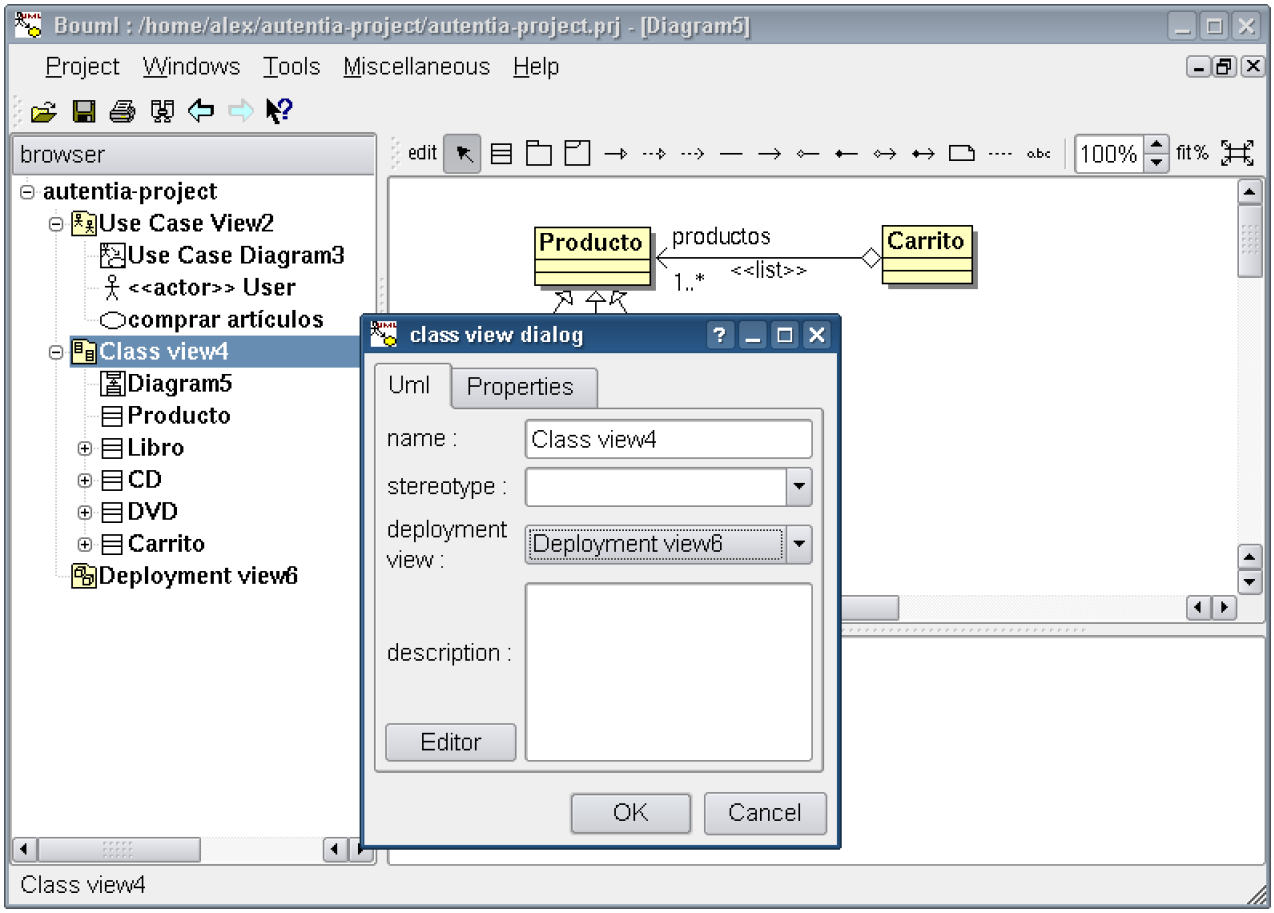
Damos a OK y volvemos a la ventana de propiedades de la agregación, nos situamos en la pestaña Java, y volvemos a pulsar sobre *Default declaration* y vemos que ahora el código queda como nos interesa:



8. Generando el código

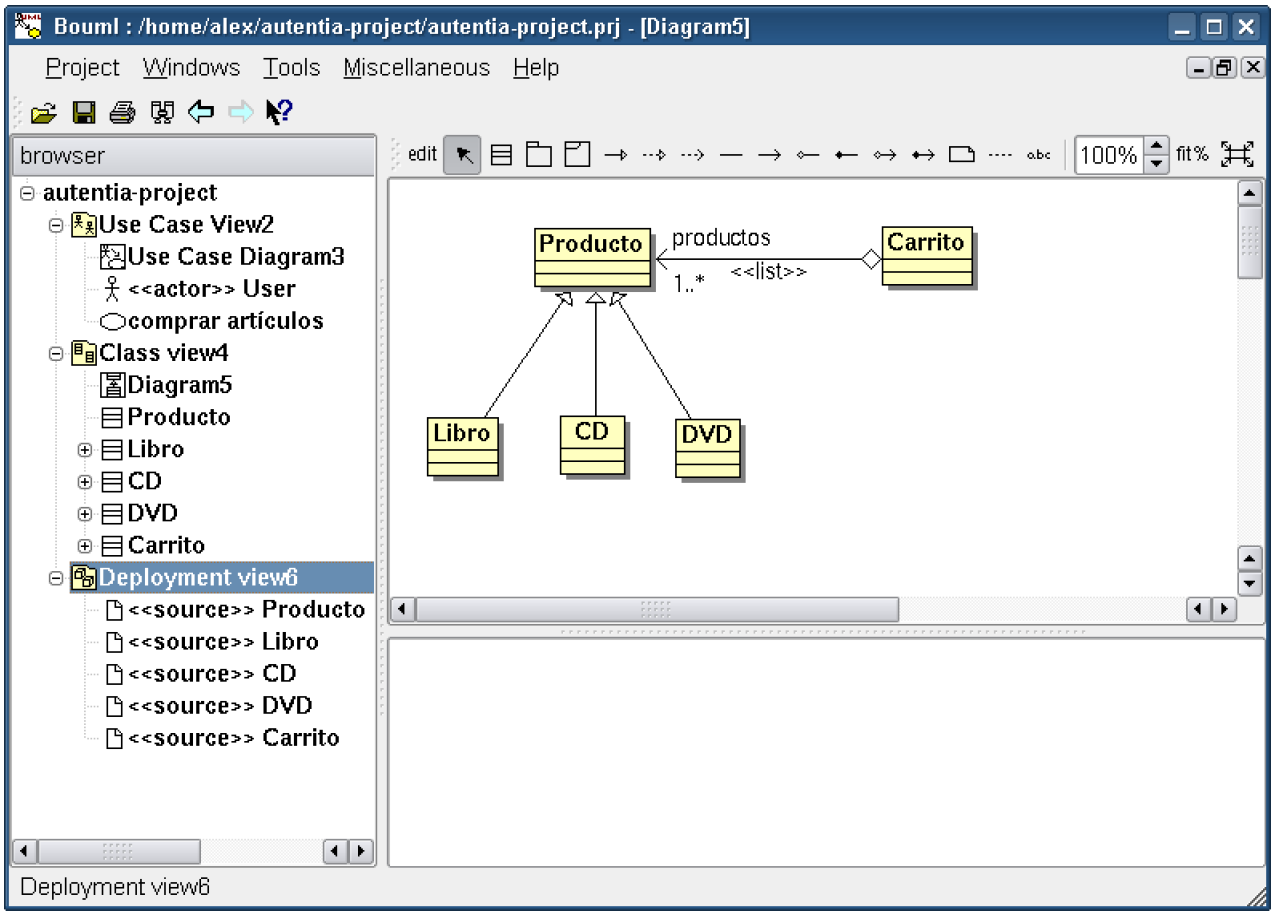
Ya hemos ajustado como queremos que se nos genere el código. Ahora vamos a generar el código. Para ello, y siguiendo el UML 2, necesitamos definir un artefacto (un .java) para cada una de nuestras clases. Esto lo haremos en una vista de despliegue. Para ello vamos a crear una como hemos hecho con las anteriores: sobre el proyecto botón derecho y seleccionamos *New deployment view*.

Ahora para vamos a indicar que nuestra vista de clases está asociada con esta nueva vista de despliegue que acabamos de crear. Para ello, sobre la vista de clases pulsamos botón derecho y seleccionamos *Edit*:



En *deployment view:* hemos seleccionado la vista de despliegue que acabábamos de crear. Y damos a OK.

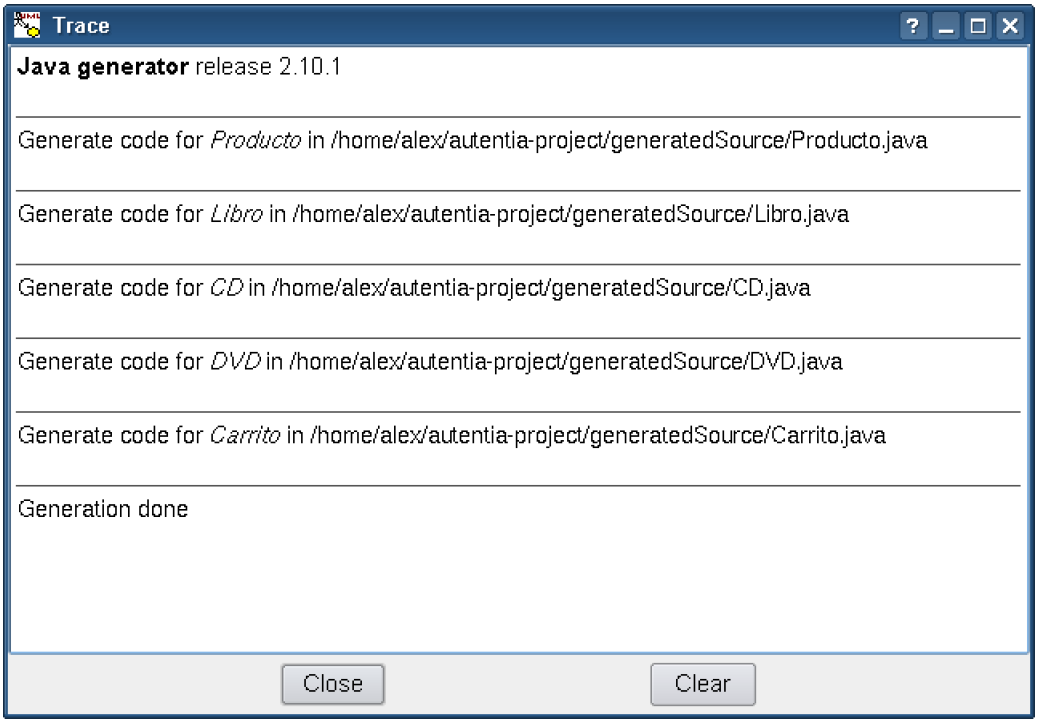
Ahora sobre cada clase de la que queremos generar código pulsamos botón derecho y seleccionamos *Create source artifact*. Vemos como nos van apareciendo los artefactos en la vista de despliegue que acabamos de asociar:



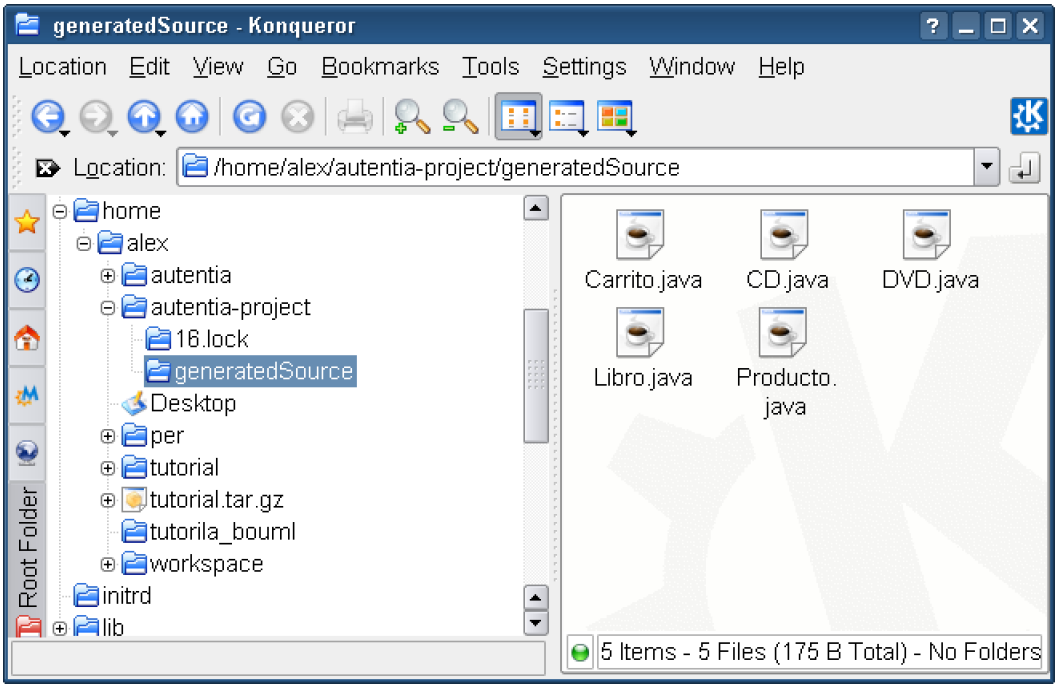
Lo último que necesitamos es definir cual será el directorio de salida del código generado. Para ello pulsamos botón derecho sobre el

proyecto y seleccionamos *Edit generation settings*, elegimos la pestaña *Directory*, indicamos el directorio que nos interese, y pulsamos OK.

Ahora para hacer la generación pinchamos sobre *Tools --> Generate Java*. Nos aparece una ventana con el resultado de la operación:



Vemos que todo es correcto. Podemos comprobarlo abriendo el directorio donde hemos hecho la generación:



9. Conclusiones

Con esto no hemos más que empezar a rascar ya que hay muchos más diagramas y opciones de visualización, generación de código, ... y está claro que todavía le faltan faltan unos cuantos detalles, como por ejemplo la típica rejilla para alinear de forma sencilla los elementos en los diagramas. Pero desde luego es bastante completa y funcional.

Igualmente el sistema de generación de código sería más interesante que estuviera basado en plantillas, ya que esto permitiría dar soporte a más lenguajes de forma muy sencilla, y nos permitiría personalizar más aun el código que se genera. Aun así es bastante flexible y podemos influir mucho en cómo se genera el código para que cumpla con nuestros estándares.

En definitiva, os animo a que la probéis.

10. Sobre el autor

Alejandro Pérez García, Ingeniero en Informática (especialidad de Ingeniería del Software)

Socio fundador de Autentia (Formación, Consultoría, Desarrollo de sistemas transaccionales)

mailto:alejandropg@autentia.com

Autentia Real Business Solutions S.L. - "Soporte a Desarrollo"

http://www.autentia.com

Nota: Los tutoriales mostrados en este Web tienen como objetivo la difusión del conocimiento.

Los contenidos y comentarios de los tutoriales son responsabilidad de sus respectivos autores.

En algún caso se puede hacer referencia a marcas o nombres cuya propiedad y derechos es de sus respectivos dueños. Si algún afectado desea que incorporemos alguna reseña específica, no tiene más que solicitarlo.

Si alguien encuentra algún problema con la información publicada en este Web, rogamos que informe al administrador rcanales@adictosaltrabajo.com para su resolución.