Introducción al DSDM

Abel Gómez Llana [agomez@dsic.upv.es].
Isidro Ramos Salavert [iramos@dsic.upv.es]
19 de Enero de 2010

eclipse

Eclipse y el soporte a modelado/metamodelado

19 de Enero de 2010

eclipse

Contenido

- Introducción a *Eclipse Modeling Framework*
  - Antecedentes
  - Vista general de EMF
  - El metamodelo Ecore
  - Creación de un metamodelo
  - Generación de código y editores
  - Uso del código generado y persistencia

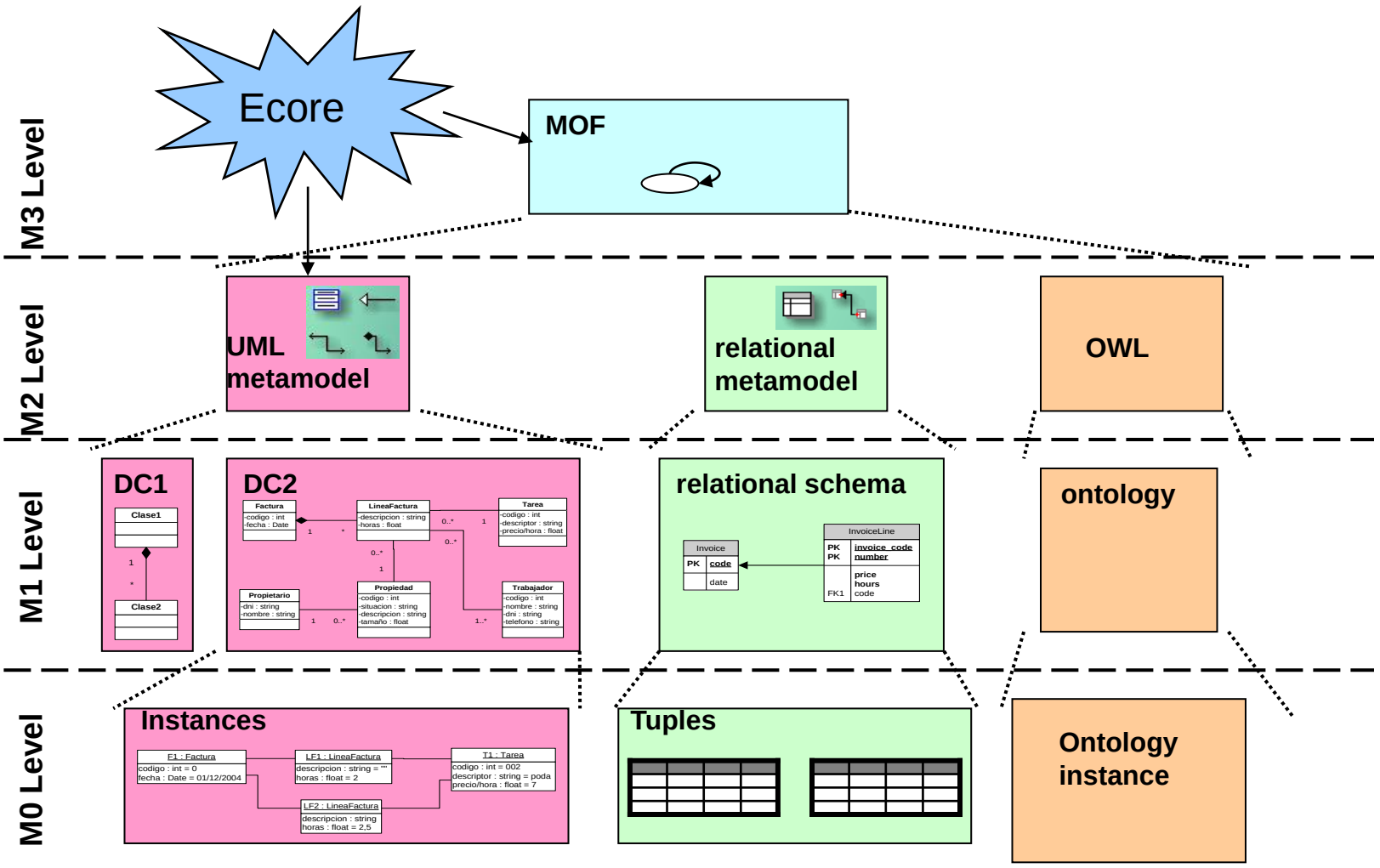
Eclipse Modeling Framework

Antecedentes

El estándar MOF

- MOF (*Meta-Object Facility*) es un estándar para ingeniería dirigida por modelos propuesto por el OMG (*Object Management Group)*.
- Entre los objetivos de MOF están la definición de un marco que dé soporte a modelado y metamodelado, así como proporcionar mecanismos de interoperabilidad e intercambio de modelos.
- El estándar proporciona una estricta arquitectura de 4 niveles, donde cada elemento del nivel *x* es instancia de un elemento del nivel *x+1*.
- El lenguaje MOF puede considerarse como un subconjunto del diagrama de clases UML, y en su versión actual, proporciona dos variantes: EMOF (*Essential MOF*) y CMOF (*Complete MOF*).
- MOF se relaciona con otros numerosos estándares: UML, OCL, XMI, JMI, QVT, etc.

Arquitectura de niveles MOF



Introducción a EMF

Eclipse Modeling Framework

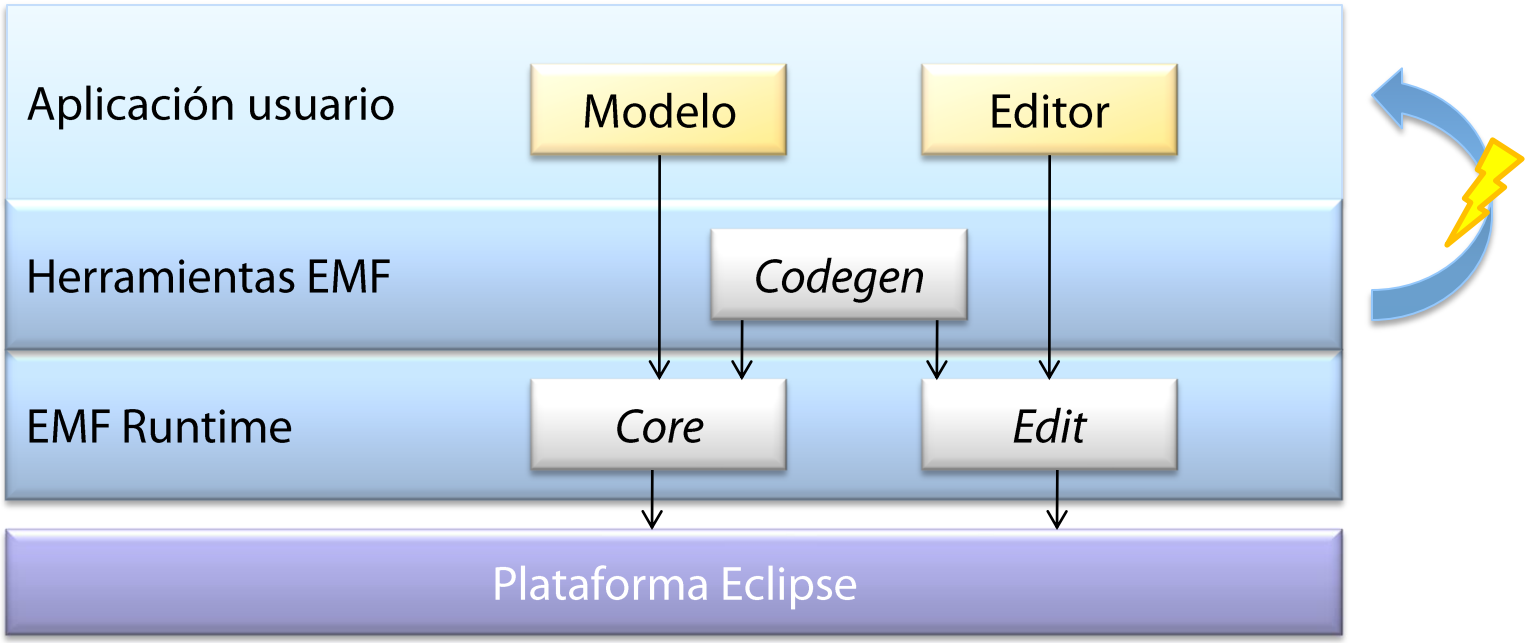
- EMF es un *framework* que proporciona herramientas para creación de modelos.
- EMF proporciona capacidades de generación de código Java de forma automática.
- Además, proporciona de forma automática los mecanismos de persistencia para las instancias de estos modelos en formato XMI.
- Como lenguaje de modelado (meta-metamodelo) proporciona el lenguaje *Ecore*. *Ecore* puede considerarse como una implementación del lenguaje EMOF.
- En la distribución clásica de Eclipse no se incluye EMF. Para poder usarse de instalarse mediante un sitio de actualizaciones.

Eclipse Modeling Framework

- Componentes EMF
  - *Core Runtime*
    - Provee la generación básica y el soporte de ejecución para la creación de la implementación de las clases Java de un modelo.
    - Metamodelo Ecore
    - Persistencia, validación y cambios del modelo.
  - *EMF.edit framework*
    - Extiende el modelo base de EMF con clases genéricas reusables para el soporte de:
      - Visores de modelos
      - Modificaciones del modelo basada en línea de comandos
      - Editores para los modelos EMF
  - *Codegen*
    - Proporciona los mecanismos de generación de código.

Eclipse Modeling Framework

- Arquitectura de EMF



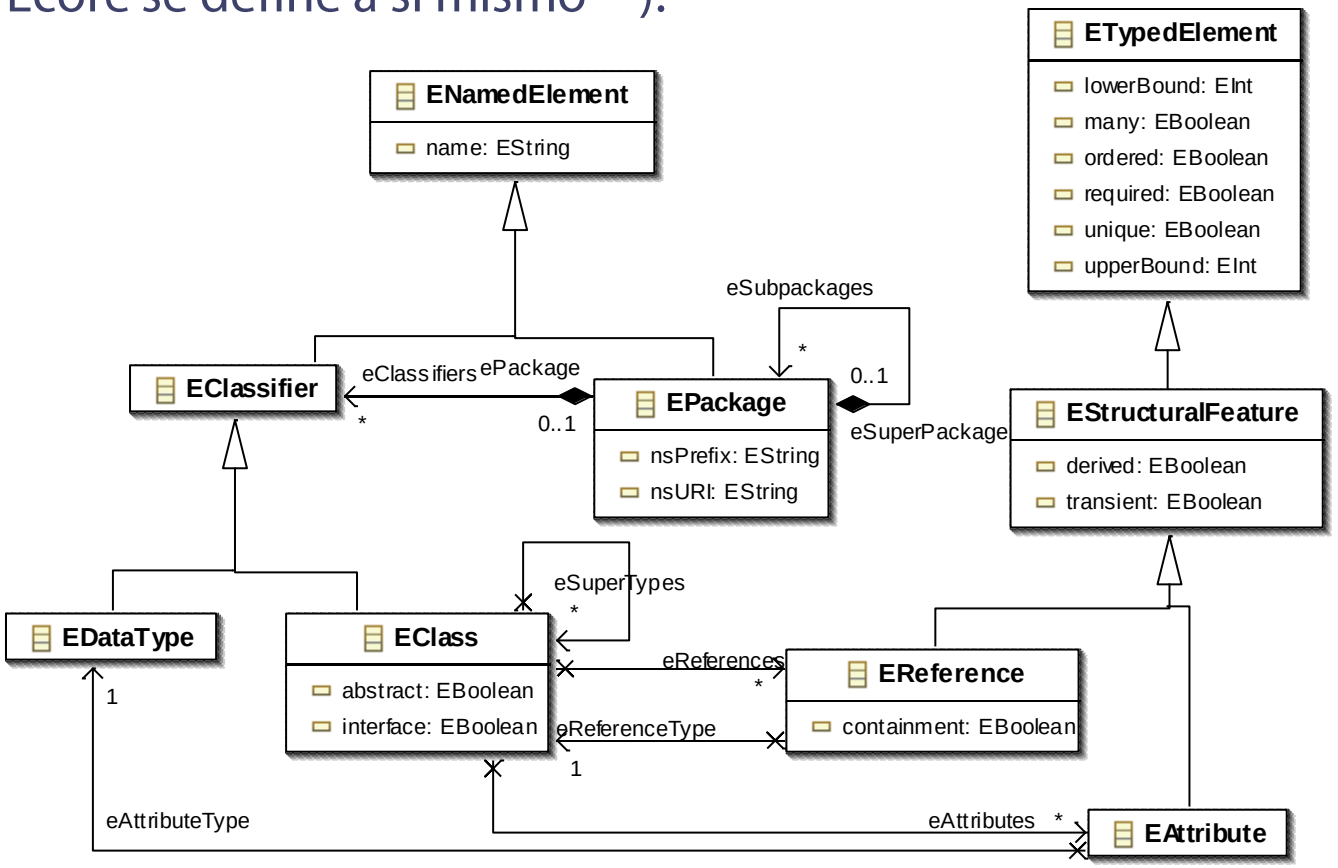
Eclipse Modeling Framework

- ¿Qué es un modelo EMF?
  - La especificación de los datos de una aplicación
    - Atributos de los objetos
    - Relaciones entre objetos
    - Operaciones que puede realizar cada objeto
    - Restricciones simples en objetos y relaciones (multiplicidad)
  - En esencia, el diagrama de clases de UML.
- EMF utiliza XMI como la forma canónica para la definición de un modelo. Existen varias formas para de definirlo:
  - Java anotado (interfaces).
  - Diagrama de clases UML (Proyecto UML2 de Eclipse o *Rational Rose*).
  - XML Schema.

El metamodelo Ecore

El metamodelo Ecore

- Subconjunto simplificado de Ecore (especificado en el mismo Ecore —Ecore se define a sí mismo—):

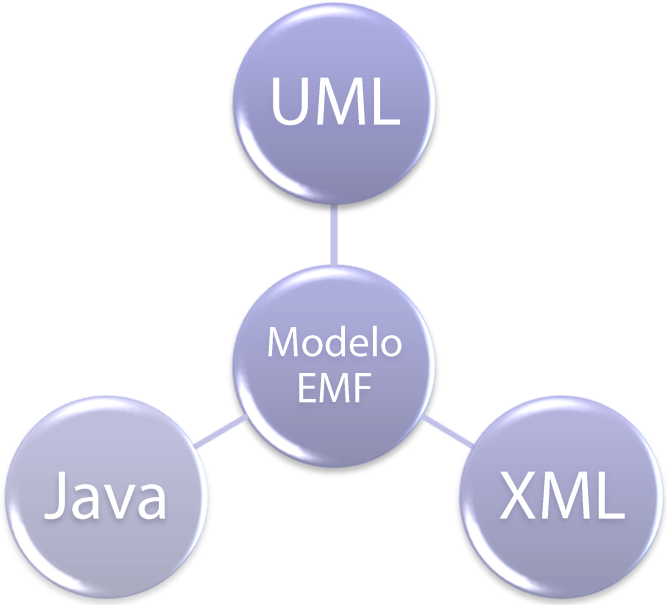


El metamodelo Ecore

- *EClassifier* — Tipo abstracto que agrupa a todos los elementos que describen conceptos
- *EDataType* — representa el tipo de un atributo. Un tipo de datos puede ser un tipo básico como *int* o *float*, o un objeto, como por ejemplo *java.util.Date*.
- *EAttribute* — Tipo que permite definir los atributos de una clase. Éstos tienen nombre y tipo. Como especialización de *ETypedElement*, *EAttribute* hereda un conjunto de propiedades como cardinalidad (*lowerBound*, *upperBound*), si es un atributo requerido o no, si es derivado, etc.
- *EReference* — Modelar las relaciones entre clases. En concreto *EReference* permite modelar las relaciones de asociación, agregación y composición que aparecen en UML. Al igual que *EAttribute*, es una especialización de *ETypedElement*, y hereda las mismas propiedades. Además define la propiedad *containment* mediante la cual se modelan las agregaciones disjuntas (denominadas composiciones en UML).
- *EPackage* — Agrupa un conjunto de clases en forma de módulo, de forma similar a un paquete en UML. Sus atributos más importantes son el nombre, el prefijo y la URI. La URI es un identificador único gracias al cual el paquete puede ser identificado unívocamente.

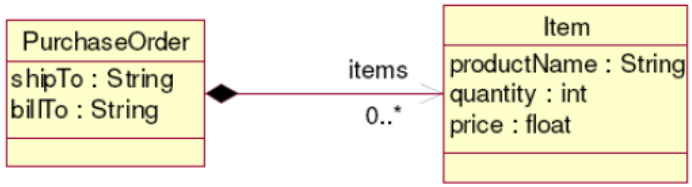
Definición de modelos Ecore

- Históricamente, un modelo *Ecore* se puede definir de diferentes maneras:

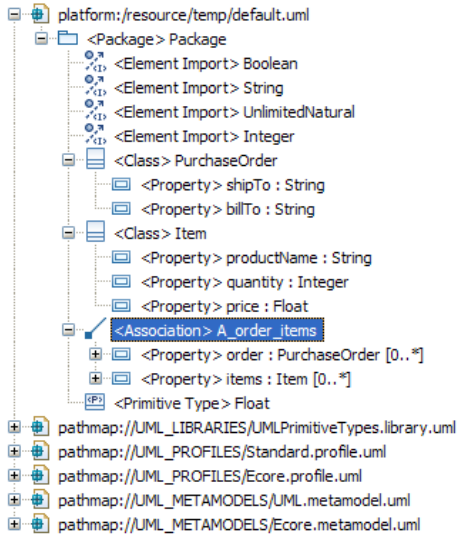


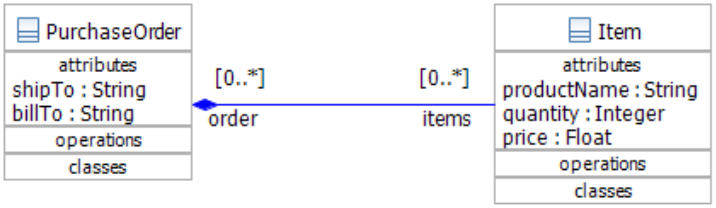
Definición de modelos Ecore

- Mediante UML:
  - Modelo de Rational Rose («*.mdl»)



  - Del proyecto de UML2 de Eclipse (últimas versiones)





Definición de modelos Ecore

- Mediante interfaces Java anotadas:

```java
/**
 * @model
 */
public interface Family {
   /**
    * Return the father
    * @return the father
    * @model
    **/
   Male getFather();

   /**
    * Return the mother
    * @return the mother
    * @model
    **/
   Female getMother();

   /**
    * Return children
    * @return list of child Individuals
    * @model type="Individual" containment="true"
    **/
   java.util.List getChildren();
}
```

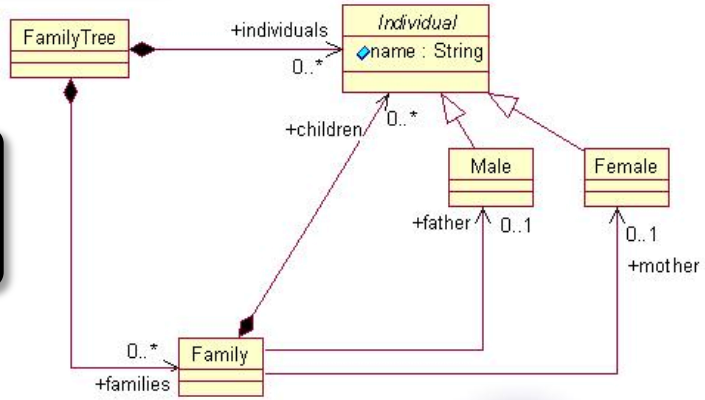
```java
/**
 * @model
 */
public interface FamilyTree {
   /**
    * Return a list of contained families
    * @model type="Family" containment="true"
    **/
   java.util.List getFamilies();

   /**
    * Return a list of contained individuals
    * @model type="Individual" containment="true"
    **/
   java.util.List getIndividuals();
}
```

```java
/**
 * @model abstract="true"
 */
public interface Individual {
   /**
    * Return the individuals name.
    * @return the name
    * @model
    **/
    String getName();
}
```

```java
/**
 * @model
 */
public interface Male extends Individual {
}
```

```java
/**
 * @model
 */
public interface Female extends Individual {
}
```



Definición de modelos Ecore

- Mediante esquema XML

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:element name="purchaseOrder" type="PurchaseOrderType"/>

  <xsd:element name="comment" type="xsd:string"/>

  <xsd:complexType name="PurchaseOrderType">
    <xsd:sequence>
      <xsd:element name="shipTo" type="USAddress"/>
      <xsd:element name="billTo" type="USAddress"/>
      <xsd:element ref="comment" minOccurs="0"/>
      <xsd:element name="items"  type="Items"/>
    </xsd:sequence>
    <xsd:attribute name="orderDate" type="xsd:date"/>
  </xsd:complexType>

  <xsd:complexType name="USAddress">
    <xsd:sequence>
      <xsd:element name="name"   type="xsd:string"/>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city"   type="xsd:string"/>
      <xsd:element name="state"  type="xsd:string"/>
      <xsd:element name="zip"    type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="country" type="xsd:NMTOKEN"
                   fixed="US"/>
  </xsd:complexType>

  <!--
     ...
   -->

</xsd:schema>
```

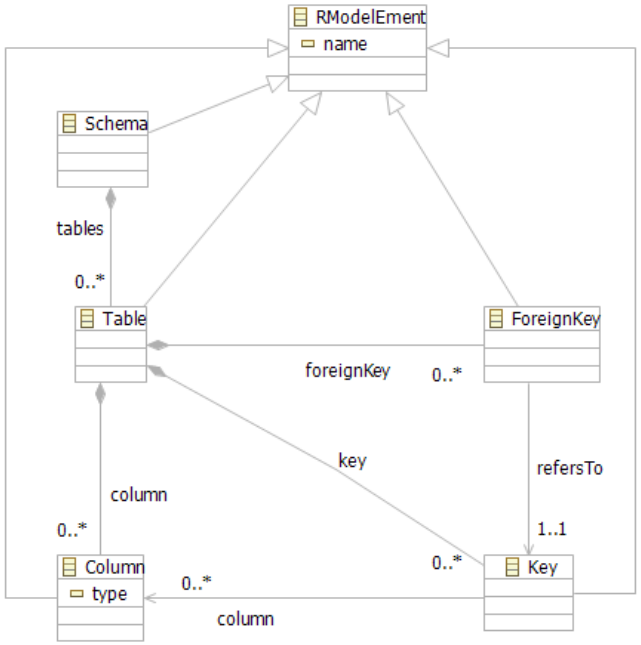
Definición de modelos Ecore

- Todas las representaciones previas, son equivalentes entre sí.
- No obstante, EMF aún proporciona una cuarta forma de representar un modelo Ecore, en XMI.
- XMI es el formato de «*representación canónica*» de un modelo Ecore.
- Un modelo Ecore puede editarse también directamente en XMI (o mediante los editores que proporciona Eclipse por defecto).
- En la práctica, en la mayoría de los casos, un modelo Ecore se creará y editará directamente de esta manera.

Creación de un nuevo proyecto de EMF

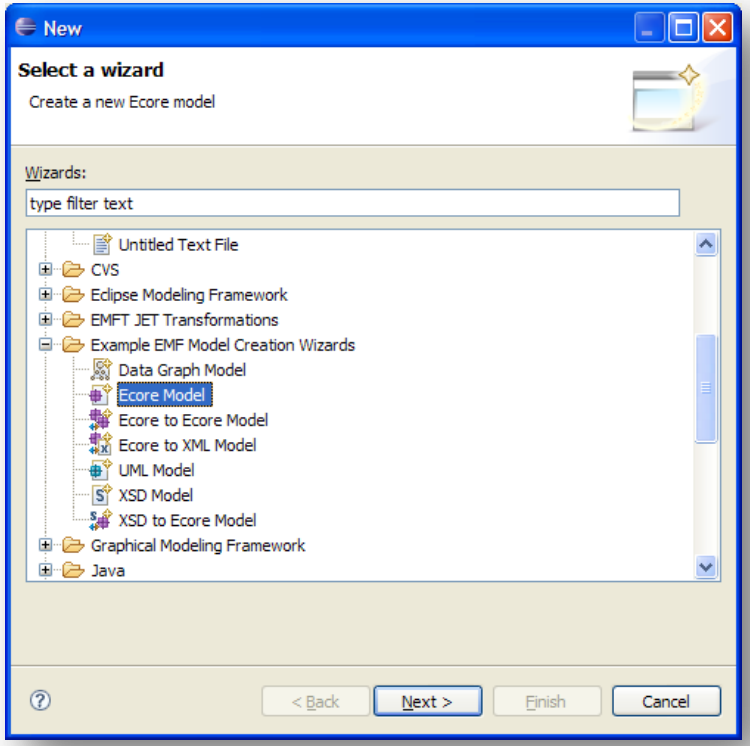
Creación de un nuevo proyecto de EMF

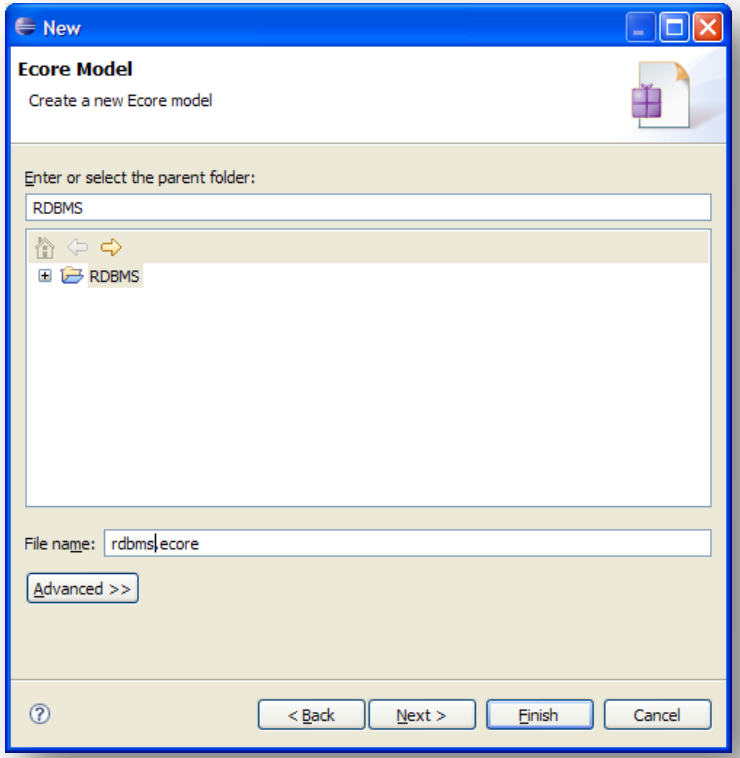
- Para crear un nuevo proyecto EMF se debe partir de un modelo Ecore pre-existente, en cualquiera de las representaciones anteriores.
- A lo largo de la sesión de hoy emplearemos como ejemplo de modelo Ecore el siguiente metamodelo simplificado de bases de datos relacionales:



Creación de un nuevo proyecto de EMF

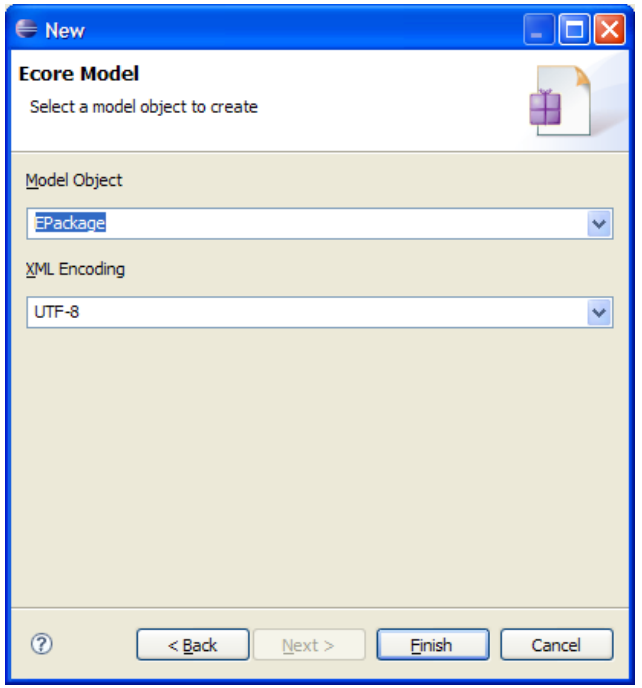
- Para crear un nuevo modelo Ecore, podemos emplear el asistente de ejemplo, al que se accede desde el menú habitual («*New*…»).





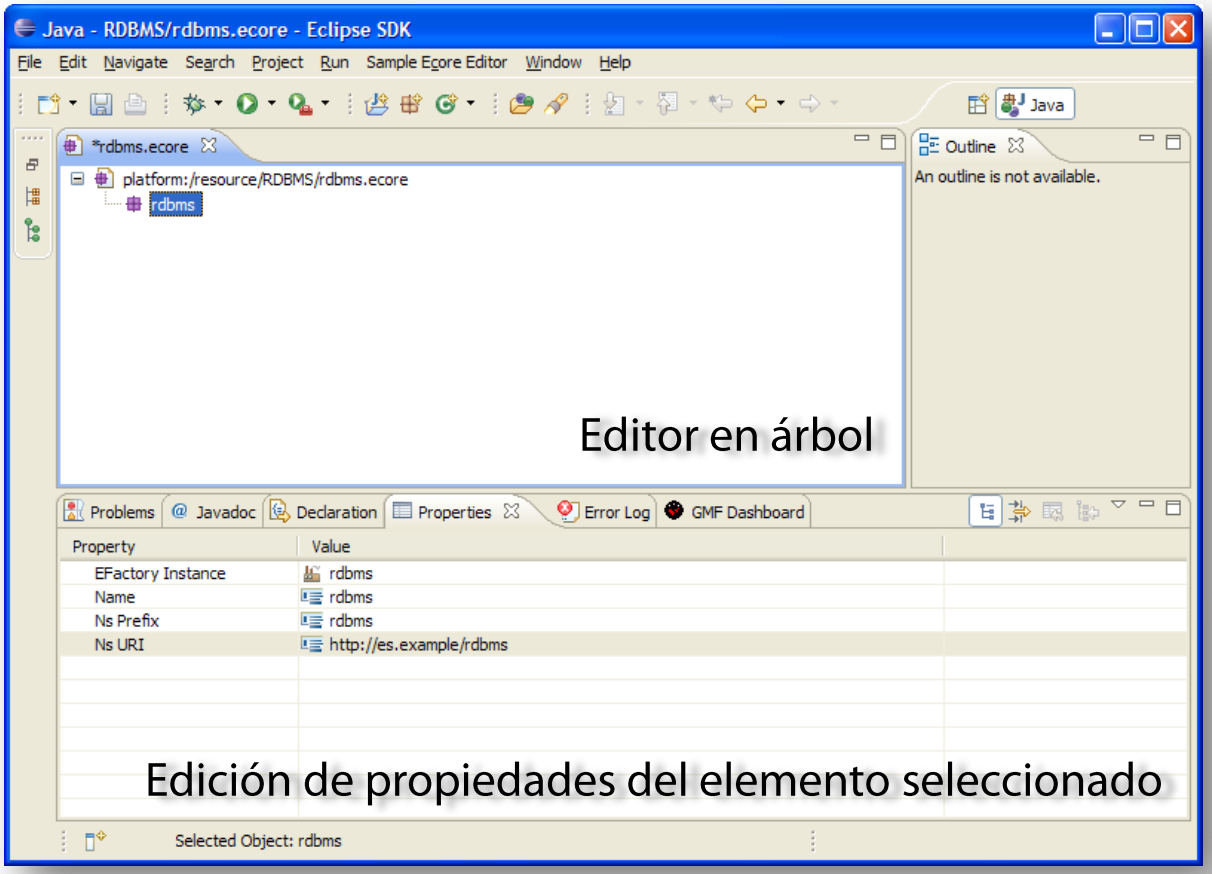
Creación de un nuevo proyecto de EMF

- Generalmente, un modelo Ecore tiene un único elemento raíz. El metamodelo Ecore, puesto que también está definido en Ecore, proporciona de forma natural el elemento *EPackage* como el elemento raíz.
- Un modelo Ecore se representa de forma arbórea. Esto se debe a que su representación está íntimamente relacionada con el formato de persistencia (XMI).
- Esta consideración debe tenerse en cuenta también al definir modelos Ecore propios, para evitar problemas en la generación de editores y en la persistencia en XMI.
- De esta manera, toda clase (salvo la que se vaya a considerar como raíz) debe estar contenida en otra (debe recibir una *referencia* de tipo *containment*).



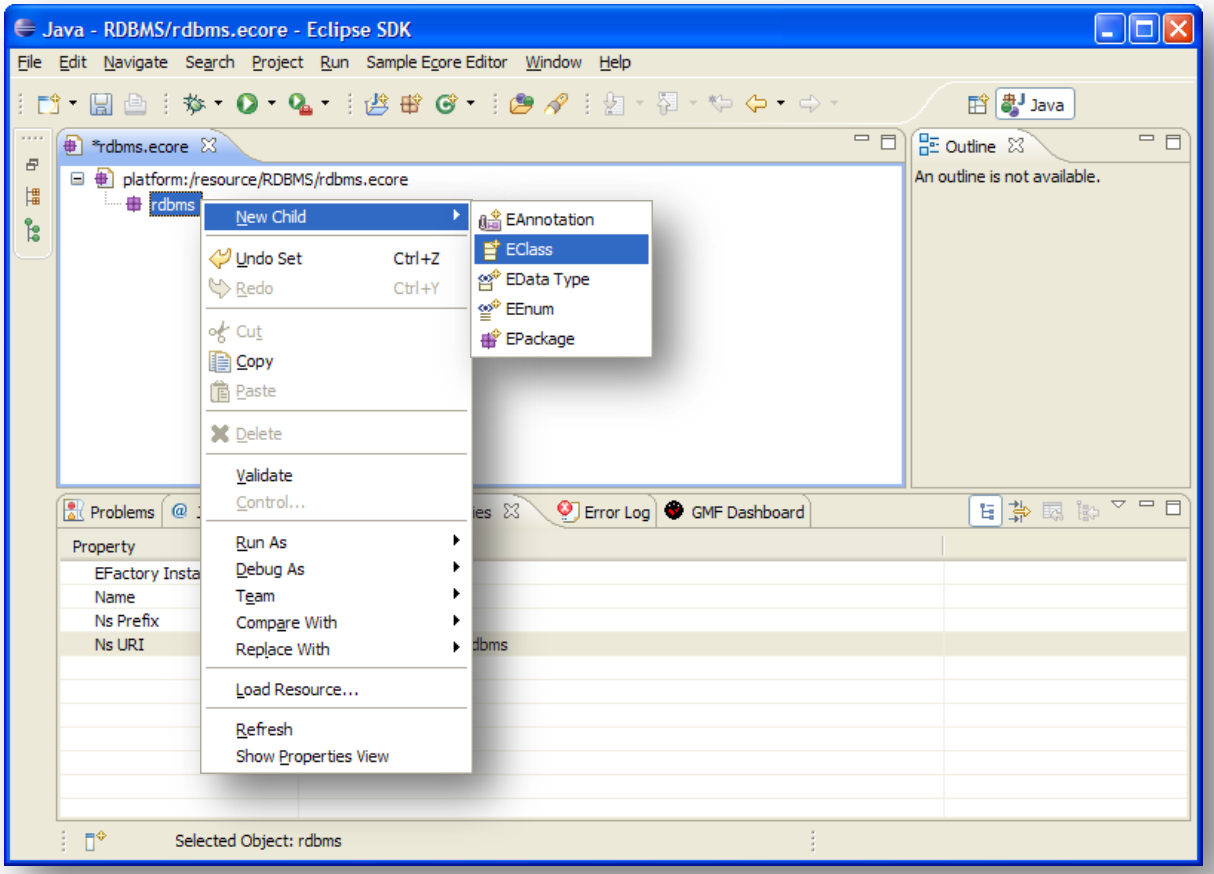
Creación de un nuevo proyecto de EMF

- Una vez creado el modelo, se abre automáticamente con el editor en árbol por defecto de EMF



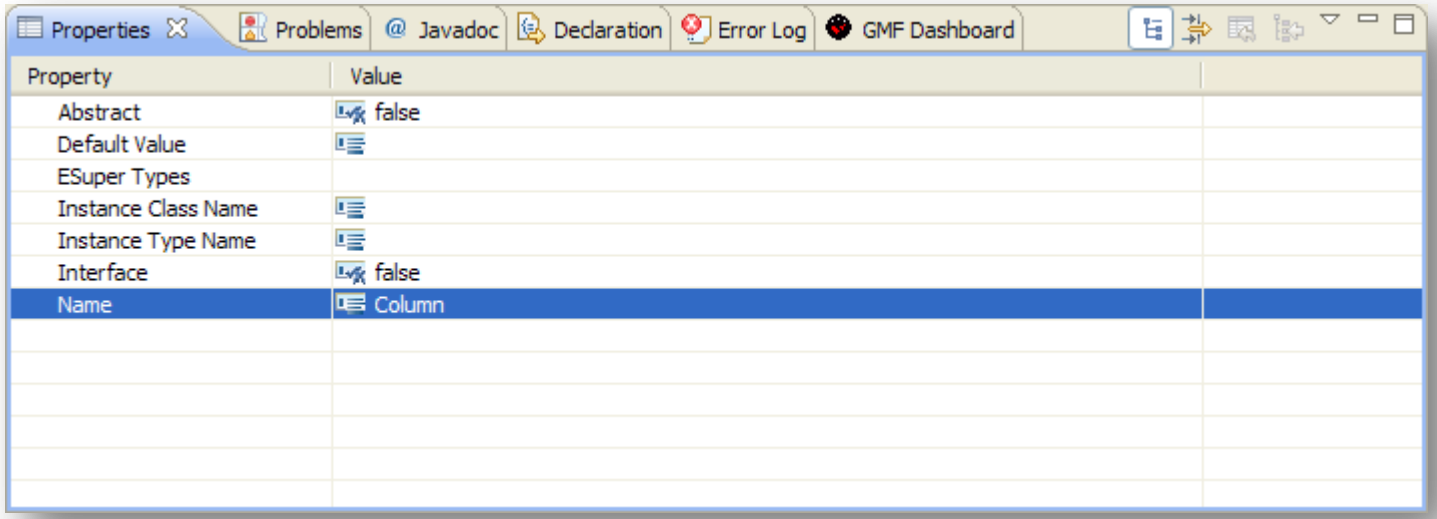
Creación de un nuevo proyecto de EMF

- Para añadir nuevos elementos en el editor en árbol, se hace uso del menú contextual «*New Child*».



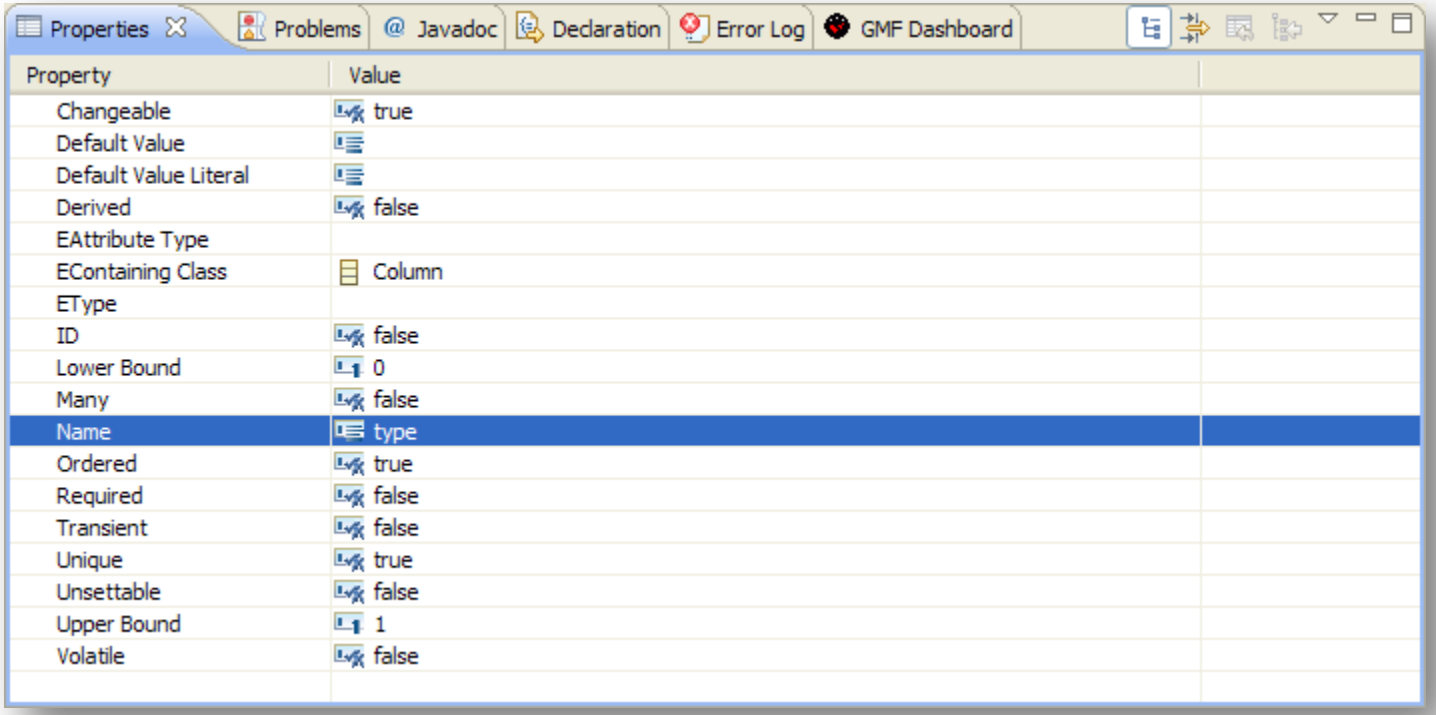
Creación de un nuevo proyecto de EMF

- Propiedades de una clase



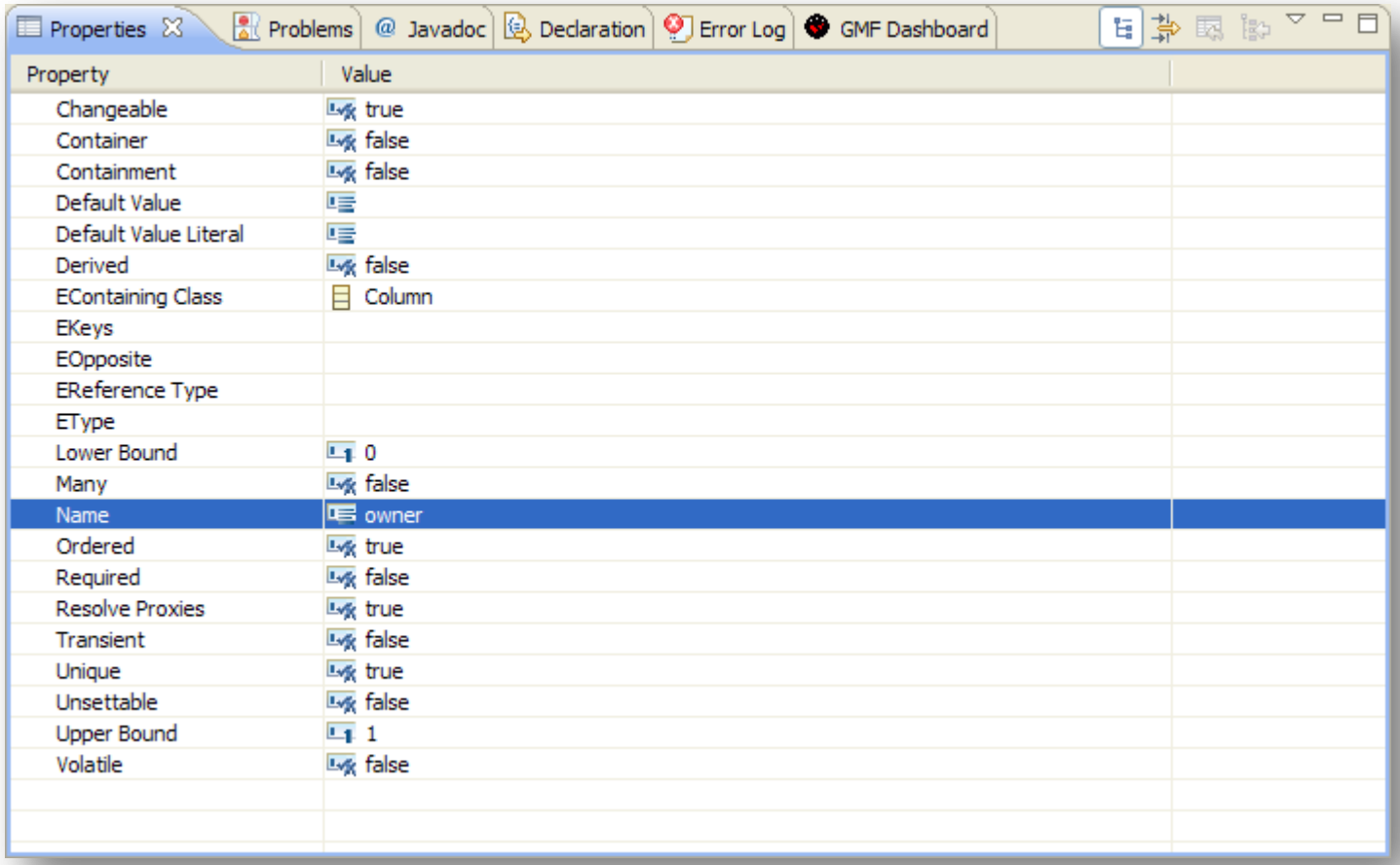
Creación de un nuevo proyecto de EMF

- Propiedades de un atributo



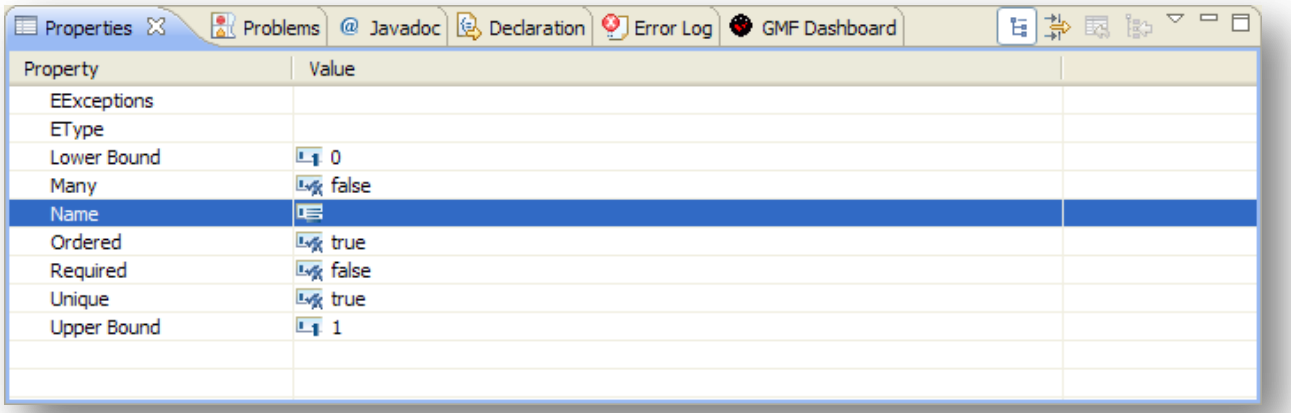
Creación de un nuevo proyecto de EMF

- Propiedades de una referencia

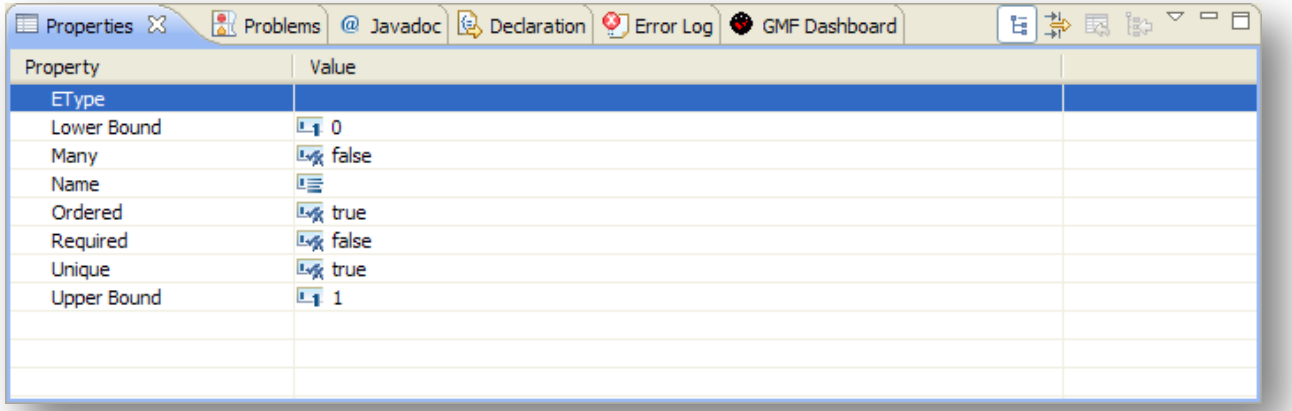


Creación de un nuevo proyecto de EMF

- Propiedades de una operación



- Propiedades de un parámetro



Creación de un nuevo proyecto de EMF

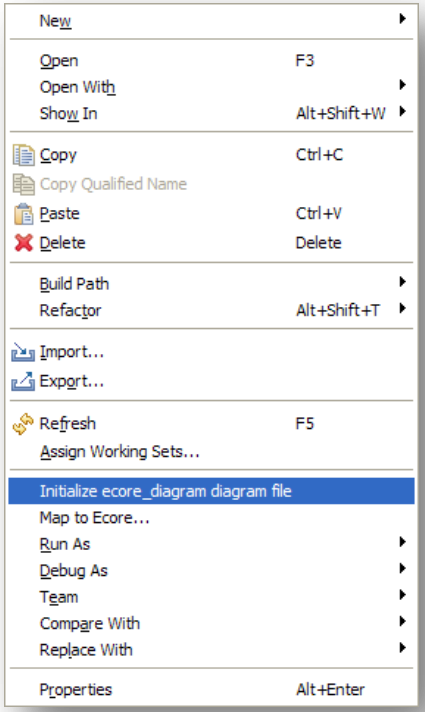
- El editor en árbol de EMF permite además crear todos los demás elementos que se especifican en el modelo de Ecore de forma similar:
  - Anotaciones (EAnnotation)
    - Detalle de entrada (Details)
  - Tipos de datos (EDataType)
  - Enumeraciones (EEnum)
    - Literales (EEnumLiteral)

Creación de un nuevo proyecto de EMF

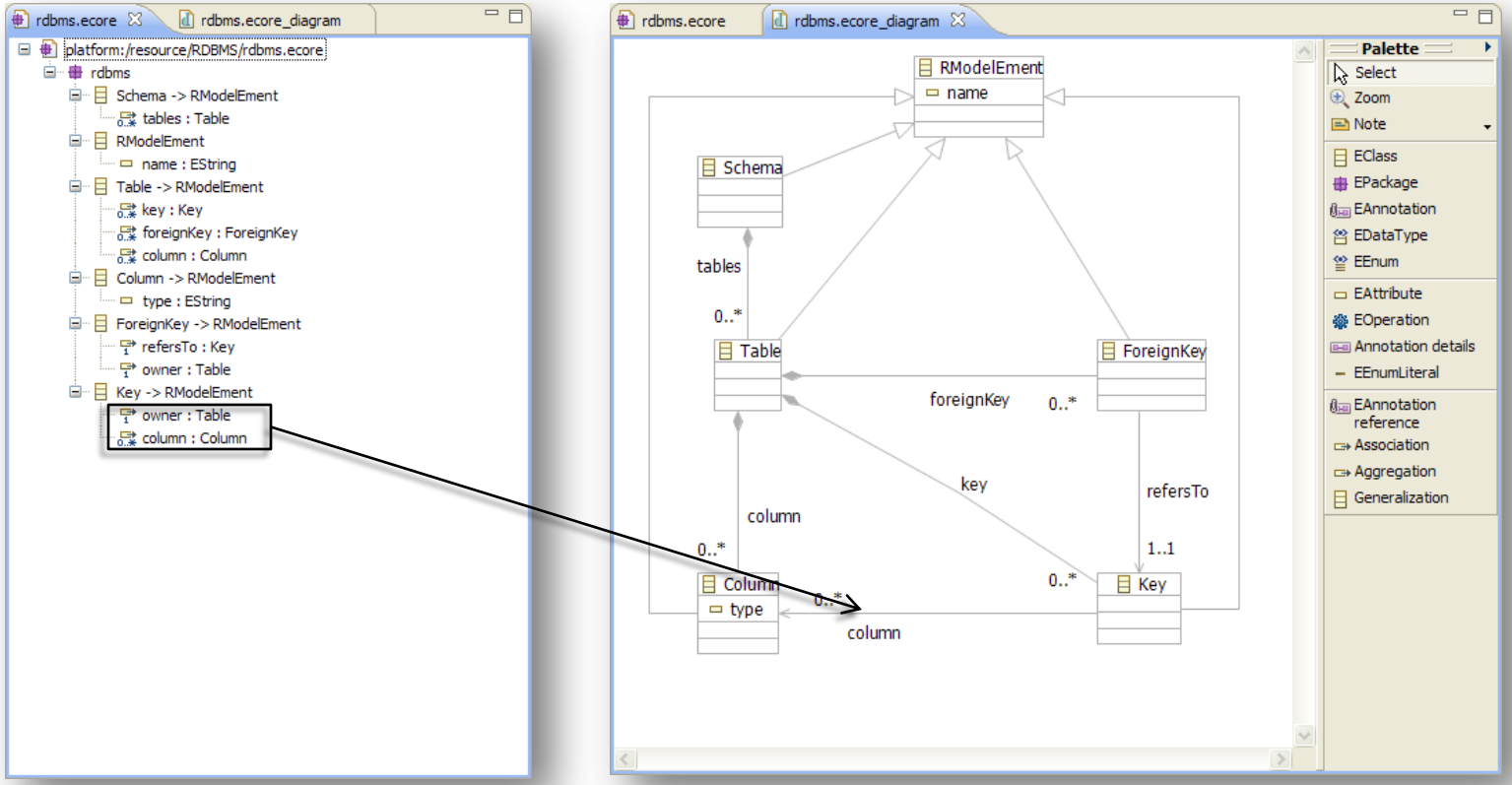
- El editor en árbol es la forma más segura de editar un modelo Ecore, no obstante puede resultar muy poco amigable para editar modelos complejos.
- Siendo Ecore un subconjunto del diagrama de clases de UML, lo ideal es poder emplear una metáfora visual similar a éste diagrama.
- Hasta la introducción de GMF no existía ningún editor gráfico propio del proyecto eclipse para la edición de modelos Ecore, a pesar de esto existía el editor Omondo (comercial).
- Tras la introducción de GMF existe un editor gráfico (generado de forma automática, como se verá más adelante) para la edición de modelos Ecore.
- El editor de GMF es útil y funcional, aunque esconde ciertos detalles del modelo, por ello, siempre es recomendable revisar (o al menos tener en cuenta) la representación en árbol del modelo bajo edición.
- NUNCA editar simultáneamente un modelo mediante el editor en árbol y el editor gráfico.

Creación de un nuevo proyecto de EMF

- Si GMF está instalado, un modelo Ecore puede crearse directamente mediante el editor en árbol.
- Los asistentes de creación de un nuevo modelo mediante GMF se encuentran por defecto en la categoría «*Other*» dentro del menú de creación de nuevo fichero.
- Un modelo gráfico en GMF requiere de dos ficheros:
  - El modelo de datos (en este caso, un «*.ecore») donde se salva la información fundamental del modelo.
  - El modelo del diagrama (en este caso, un «*.ecore_diagram») donde se almacena únicamente la información gráfica sobre como se representará el modelo en pantalla.
- De esta forma, se mantiene compatibilidad de los modelos editados de esta manera, y se mantiene bien diferencia la información propia del modelo de su representación.
- Si el editor gráfico para un determinado tipo de modelos está instalado, se puede inicializar el modelo del diagrama a partir de un modelo de datos pre-existente mediante el menú contextual.



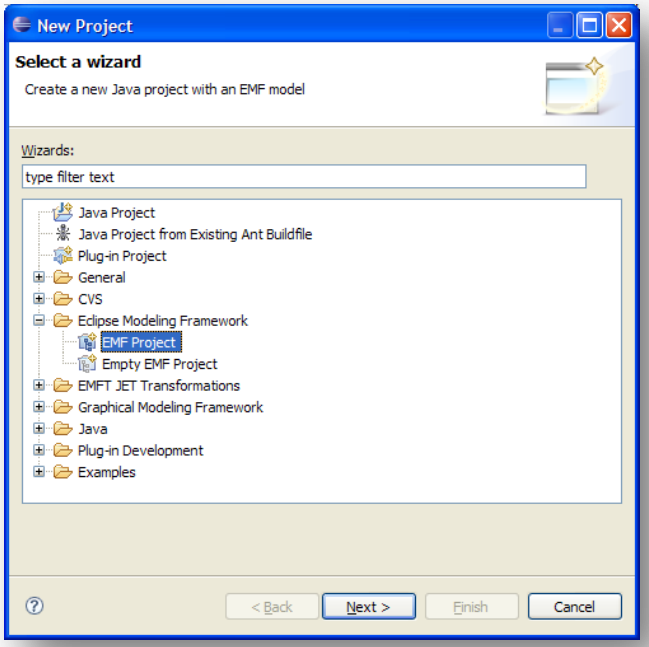
Creación de un nuevo proyecto de EMF



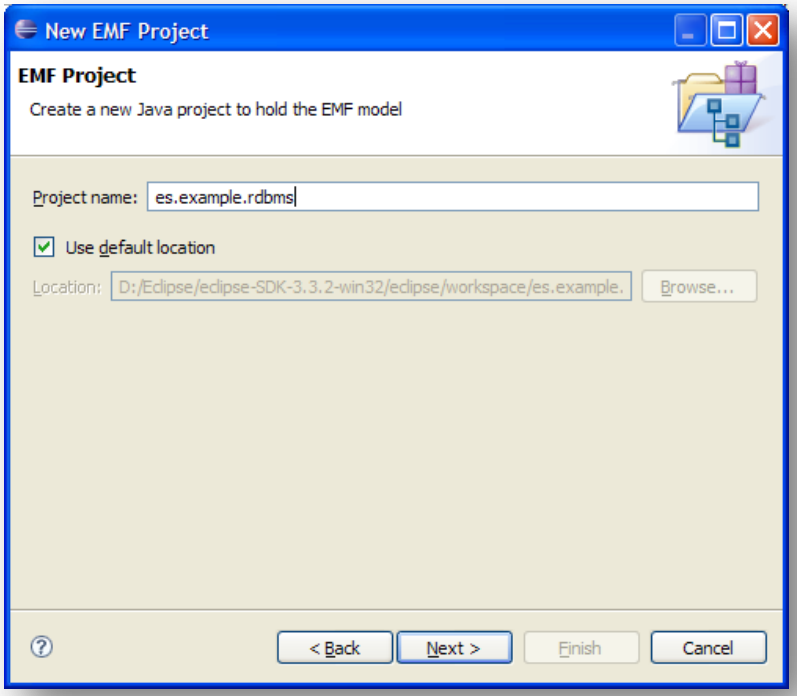
¿… y owner…?

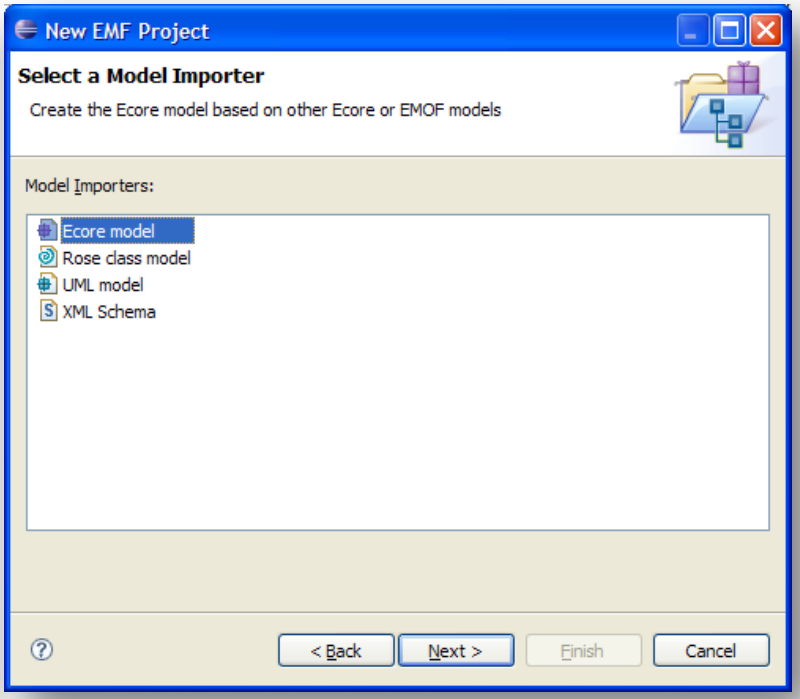
Creación de un nuevo proyecto de EMF

- Una vez tenemos un primer modelo Ecore, podemos generar un nuevo proyecto EMF.
- Para ello invocaremos el asistente de nuevo proyecto «*EMF Project*» desde la ubicación habitual (New → Project).

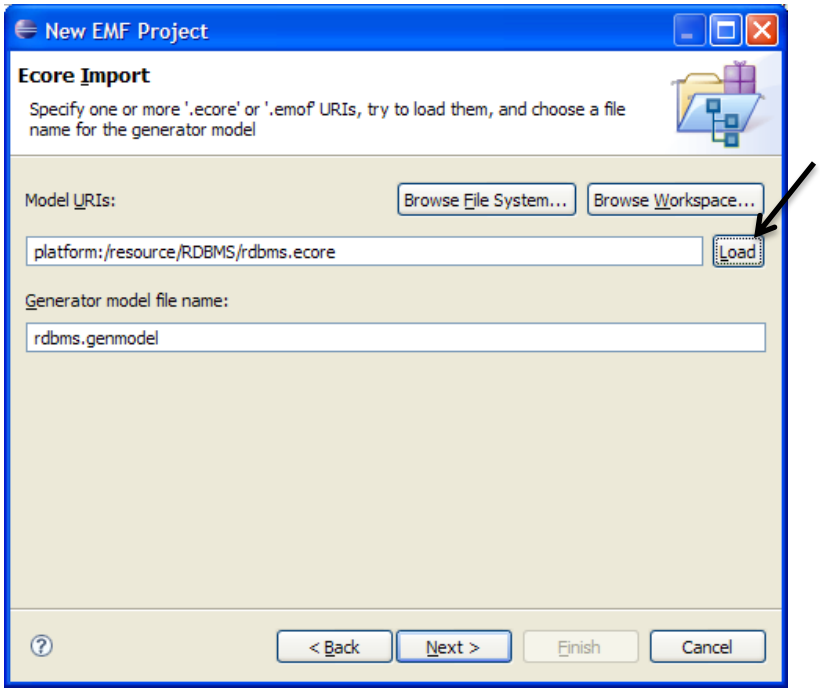


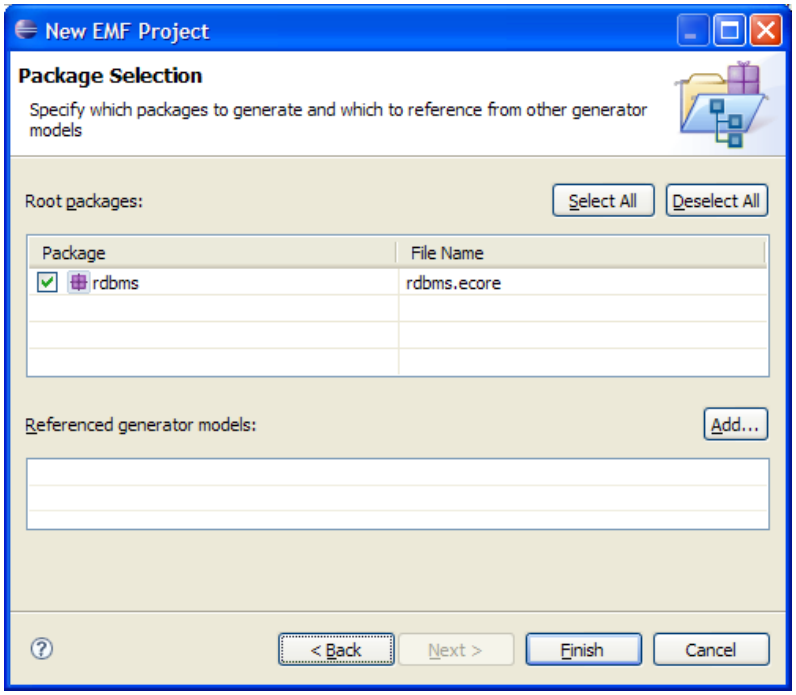
Creación de un nuevo proyecto de EMF





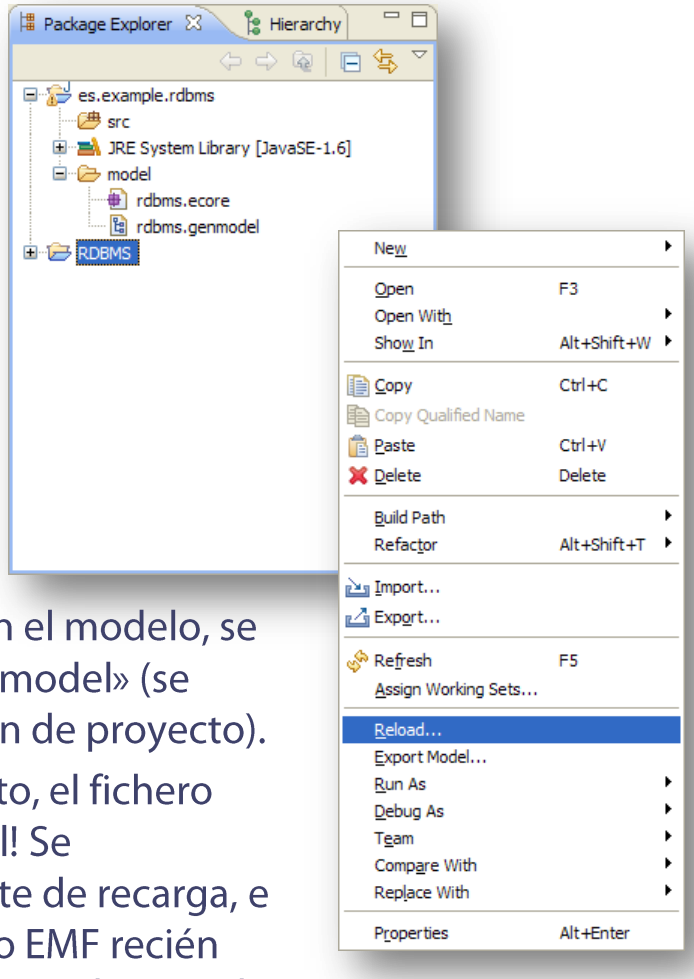
Creación de un nuevo proyecto de EMF



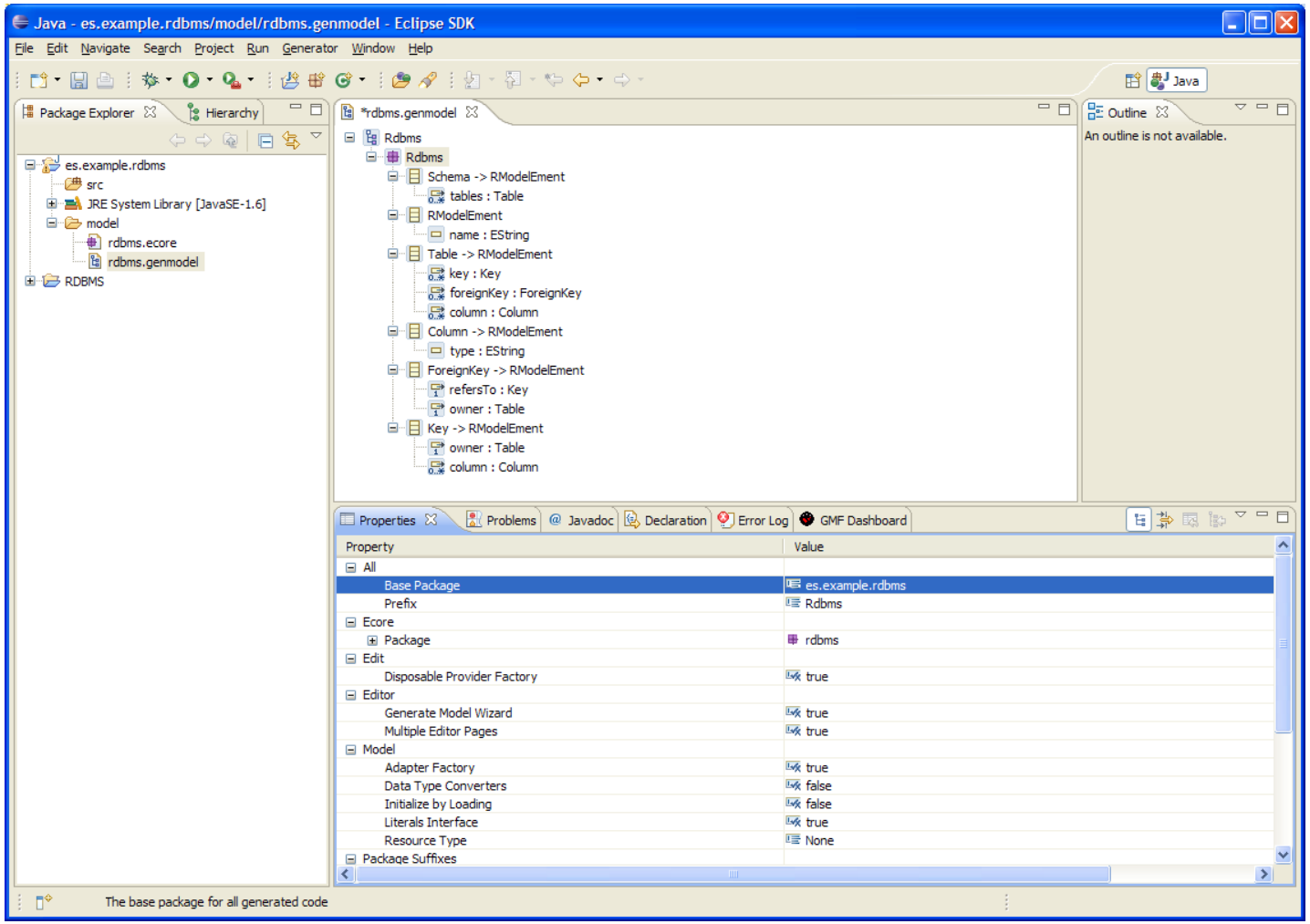


Creación de un nuevo proyecto de EMF

- Se creará un nuevo proyecto con una carpeta de código fuente vacía, y una carpeta «*model/*» con el fichero de nuestro modelo (una copia exacta) y el fichero «*.genmodel».
- A partir de ahora, las ediciones deberemos hacerla sobre el modelo recién creado.
- El fichero «*.genmodel» es el que servirá para la generación de código. Enlaza y consulta al fichero «*.ecore» para saber el código de qué clases debe generar, y debe de mantenerse sincronizado con el modelo Ecore.
- En caso de que se hagan cambios importantes en el modelo, se recomienda recargar (*Reload…*) el fichero «*.genmodel» (se iniciará un asistente similar al anterior de creación de proyecto).
- ¡Atención! ¡La primera vez que se crea un proyecto, el fichero «*.genmodel» enlaza al modelo «*.ecore» original! Se recomienda encarecidamente ejecutar el asistente de recarga, e indicando el nuevo fichero «*.ecore» del proyecto EMF recién creado para evitar confusiones (y pudiendo eliminar el original).



Creación de un nuevo proyecto de EMF



Generación de código Java

Clases generadas para el modelo

- Para cada clase del modelo (*EClass*):
  - Una interfaz Java:

```
public interface Column …
```

  - Y la correspondiente clase que implementa la interfaz:

```
public class ColumnImpl extends … implements Column
```

- Esta separación es una decisión de implementación impuesta por EMF, que permite simular la herencia múltiple que permite EMF en Java.
- Las interfaces se agrupan en el paquete «*prefijo.nombremodelo*».
- Las implementaciones en el paquete «*prefijo.nombremodelo.impl*».

Clases generadas para el modelo

- Además se observa que todas las interfaces extienden directa o indirectamente de la interfaz *EObject* (y las clases extienden la clase *EObjectImpl*):

```
public interface RModelEment extends EObject
```

- *EObject* es el equivalente de EMF a la clase *java.lang.Object*, esto es, una clase base para todos los objetos modelados. *EObject* introduce tres comportamientos básicos:
  - *eClass()* — devuelve el metaobjeto del objeto (una EClass).
  - *eContainer()* y *eResource* — devuelve el objeto contedor y el recurso del objeto.
  - *eGet()*, *eSet()*, *eIsSet()* y *eUnset()* — proporcionan una API para acceder al objeto de forma reflexiva. Éstos son ligeramente más ineficientes que los métodos generados, pero abren el modelo a un acceso completamente genérico.

Clases generadas para el modelo

- La clase *EObject* proporciona una funcionalidad más:

```
public interface EObject extends Notifier {
```

- La interfaz *Notifier* introduce notificaciones acerca del cambio del modelo.
- El patrón exacto que se utilizada en los métodos generados para la implementación de una característica dada (atributo o referencia) depende del tipo y otras propiedades ajustables por el usuario. Veamos como ejemplo el atributo *type* de una *Columna* (para otros atributos se emplean patrones más complejos, especialmente en *referencias bidireccionales*).

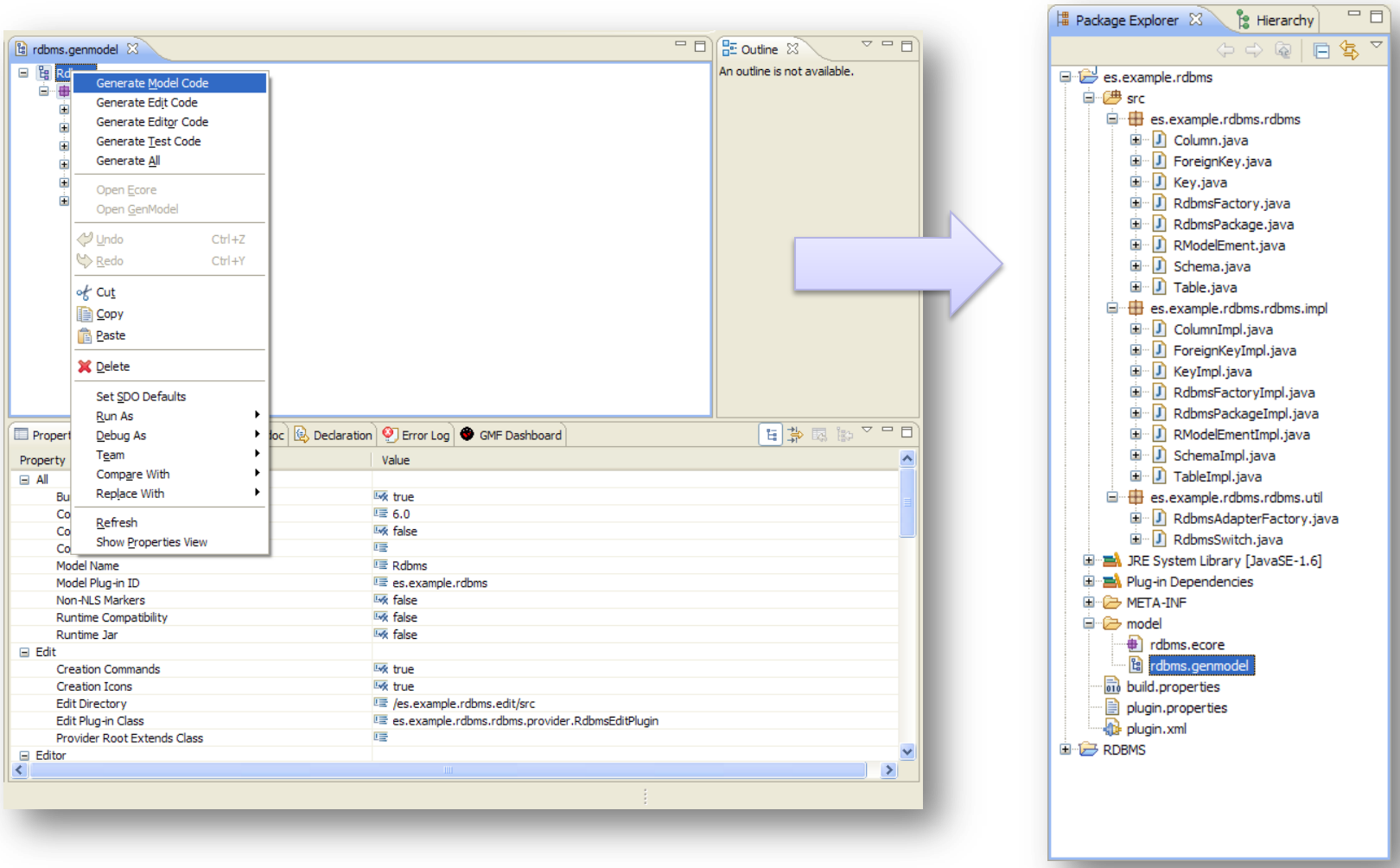
```
public String getType() {
        return type;
}
```

```
public void setType(String newType) {
        String oldType = type;
        type = newType;
        if (eNotificationRequired())
              eNotify(new ENotificationImpl(this, Notification.SET
                                    RdbmsPackage.COLUMN__TYPE, oldType, type));
}
```

Clases generadas para el modelo

- Para cada modelo, además, se crean dos clases adicionales:
  - Una *factoría*, que proporciona un método *create* para cada clase del modelo.
    - El modelo de programación de EMF recomienda encarecidamente el uso de la factoría (aunque no lo obliga).
    - Se recomienda escribir «*RdbmsFactory.eINSTANCE.createColumn()*» en lugar de «*new ColumnImpl()*».
  - Una clase que representa al *paquete*. Esta clase contiene todos los metadatos Ecore del metamodelo (nombre, ns prefix, URI, etc) así como métodos de acceso a las *EClasses*, *EReferences*, etc. del modelo.

Invocación del generador



Generación de editores en árbol

- Además del código del modelo, EMF permite generar editores en árbol específicos para un modelo Ecore concreto (registrando una extensión por defecto para ellos, iconos personalizados, etc).
- Para este caso, el código generado no se genera en el mismo proyecto que el del modelo, sino que se genera por separado, y además en dos *plug-ins*.
  - **.edit*, en este *plug-in* se incluyen clases de ayuda a la edición, pero independientes del editor concreto (etiquetas que identifican un determinado elemento del modelo, icono asociado, hijos para los que se deben dar opciones de creación, etc).
  - **.editor*, este *plug-in* implementa el propio editor, y hace uso del *plug-in* anterior. Registra la extensión por defecto, etc.

Generación del código EMF.Edit

- «*Generate edit code*» creará un plug-in completo conteniendo la parte independiente del editor concreto, esto es:
  - Un conjunto de *item providers* (etiquetas, iconos, etc), uno para cada clase en el modelo.
  - Una clase *item provider adapter factory* que creará los *item providers* generados.
  - Una clase *Plugin* que incluye los métodos para localizar los recursos de texto e iconos del plug-in.
  - Archivos de manifiesto del plug-in, *(plug-in.xml, MANIFEST.MF)*, especificando las dependencias requeridas.
  - Un archivo de propiedades, *plugin.properties*, conteniendo las cadenas de texto externalizadas necesarias por las clases generadas y el *framework*.
  - Un directorio de iconos, uno para cada clase del modelo.

Generación del editor

- En el plug-in del editor, se generarán los siguientes elementos:
  - Un editor integrado en el *workbench* de Eclipse.
  - Un asistente para crear documentos con nuevas instancias del modelo (se incluirá en el menú por defecto, bajo la categoría «*Example EMF Model Creation Wizards*»).
  - Una contribución a las barras de menús, de herramientas y menús contextuales.
  - Una clase *Plugin* que incluye los métodos para localizar los recursos de cadenas de texto e iconos.
  - Archivos de manifiesto del *plug-in*, que especifican las dependencias requeridas y extensiones del editor, asistente, y puntos de extensión de las acciones del *workbench*.
  - Un archivo de propiedades, *plugin.properties*, conteniendo las cadenas de texto externalizadas necesarias para las clases generadas y para el *framework*.
  - Un directorio conteniendo iconos para el editor y el asistente de «*Nuevo modelo*».

Uso del código generado

Uso del código generado

- Además de los *plug-ins* del modelo, del soporte a la edición, y del editor, se puede generar automáticamente un *plug-in* de pruebas mediante la opción «*Generate test code*» del fichero «**.genmodel*».
- Este *plug-in* de pruebas sirve como ejemplo de cómo se usa el código generado tanto desde un *plug-in* que se ejecuta dentro de Eclipse, como de una aplicación Java independiente.

Creación/carga-edición de una instancia

1. Crear un nuevo ResourceSet:

```
ResourceSet resourceSet = new ResourceSetImpl();
```

2. Registrar la ResourceFactory:

```
resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap().put(Resource.Factory.Registry.DEFAULT_EXTENSION, new XMIResourceFactoryImpl());
```

3. Registrar el paquete:

```
resourceSet.getPackageRegistry().put(RdbmsPackage.eNS_URI, RdbmsPackage.eINSTANCE);
```

4. Crear/Cargar:
   - Crear un nuevo *Resource:*

```
Resource resource = resourceSet.createResource(URI.createURI("http:///My.rdbms"));
```

   - Cargar un nuevo *Resource*:

```
Resource resource = resourceSet.getResource(uri, true);

Resource.load();
```

5. Editar el modelo:
   - Se accede a las instancias del Resource mediante el método resource.getContents().
6. Salvar el modelo:

```
resource.save(outputStream, null);
```