Herramientas Eclipse para Desarrollo de Software Dirigido por Modelos

Cristina Vicente Chicote
Teléfono: (+34) 968 32 6448
E-mail: Cristina.Vicente@upct.es

Diego Alonso Cáceres
Teléfono: (+34) 968 32 5341
E-mail: Diego.Alonso@upct.es

División de Sistemas e Ingeniería Electrónica (DSIE)
Departamento de Tecnologías de la Información y Comunicaciones
Escuela Técnica Superior de Ingeniería de Telecomunicación
Edificio Antigones, Plza. del Hospital Nº 1, 30202 Cartagena
Universidad Politécnica de Cartagena

Vol. 1, Nº 8, Sistedes 2007, ISSN: 1988-3455

Tabla de contenidos 1/2

Tabla de contenidos 2/2

- Herramientas de transformación Modelo-A-Modelo (M2M)
  - Introducción al **Atlas Transformation Language** (ATL)
  - **Ejemplo práctico 3:** Transformando componentes en figuras
    - Creación del nuevo meta-modelo de figuras
    - Definición de la transformación ATL entre meta-modelos
    - Validación de la transformación
- Herramientas de transformación Modelo-A-Texto (M2T)
  - Introducción a **MOFScript**
  - **Ejemplo práctico 4:** Generando ficheros de texto a partir de los modelos de figuras.
    - Definición de la transformación MOFScript
    - Validación de la transformación

La plataforma Eclipse 1/4

- Eclipse es una plataforma abierta y de libre distribución

  http://www.eclipse.org

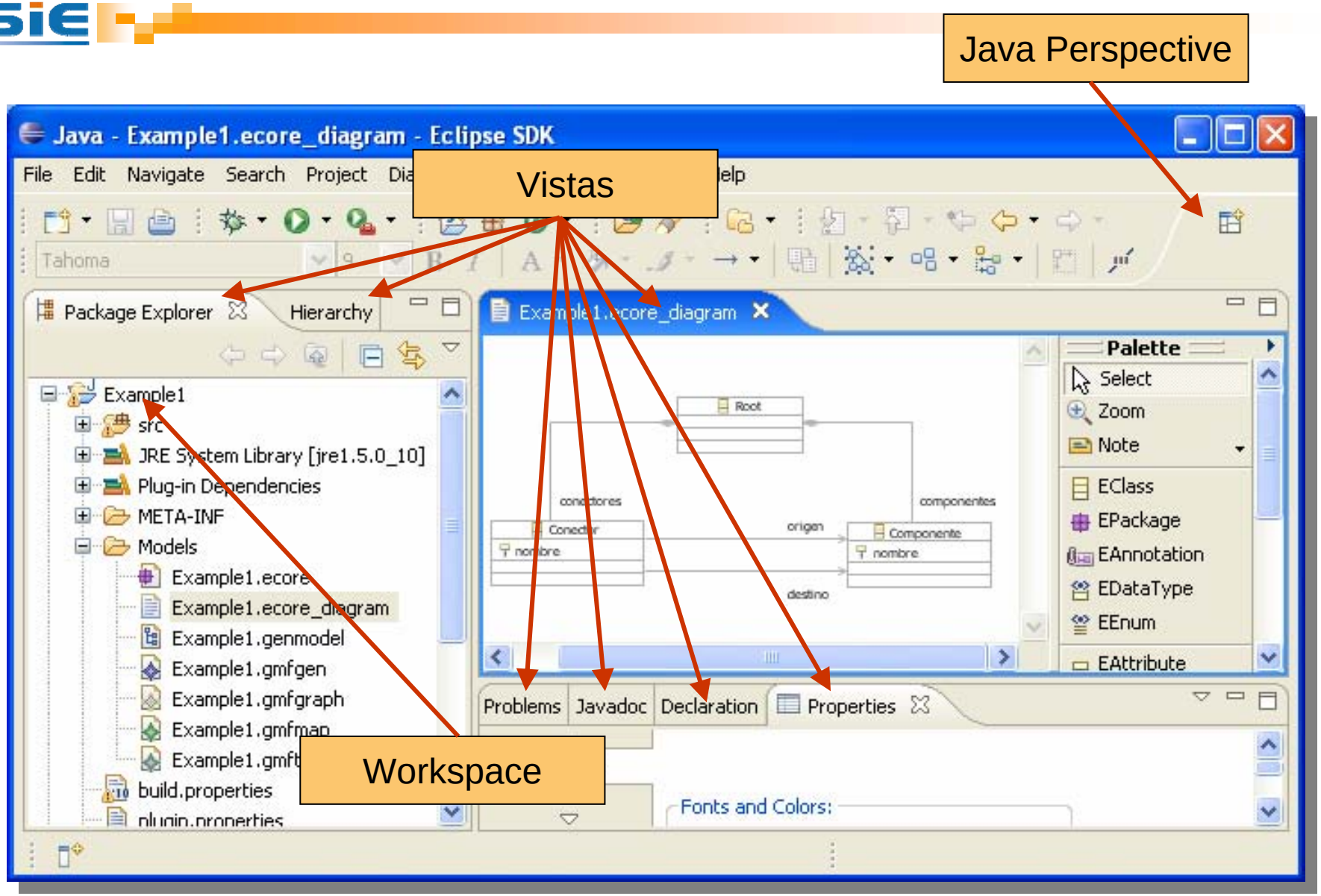
- En su desarrollo participan importantes empresas como Borland, IBM, Intel, Motorola, etc.
- Actualmente, la comunidad Eclipse se organiza en torno a múltiples proyectos que evolucionan en paralelo de manera independiente o cooperativa.
- Actualmente, los proyectos relacionados con la Ingeniería Dirigida por Modelos (MDE) se encuentran entre los más activos: **EMF, GMF, M2M, M2T,** ...

  http://www.eclipse.org/modeling/

La plataforma Eclipse 2/4

- Necesita run-time de Java (*jre*)
  - Aunque funciona con 1.4.2, se recomienda a partir de 1.5
- No necesita instalación (se descomprime en cualquier carpeta)
- Fácil de extender con distintos plug-ins:
  - Se descargan y descomprimen directamente en \eclipse
  - También utilizando el menú "Help → Software Updates"
- Entorno de trabajo:
  - **Workspace**: directorio donde se almacenan todos los proyectos relacionados. Mantienen sus propias propiedades.
  - **View**: ventanas de utilidad, como gestor de proyectos, propiedades, consola, etc.
  - **Perspective**: agrupación de vistas (*views*) que facilitan alguna tarea concreta, e.g. desarrollo Java.

La plataforma Eclipse



El meta-mundo… 1/4

M2T

EMF

MDA

DSL

PSM

M2M

Meta-Model

XMI

PIM

CIM

Model

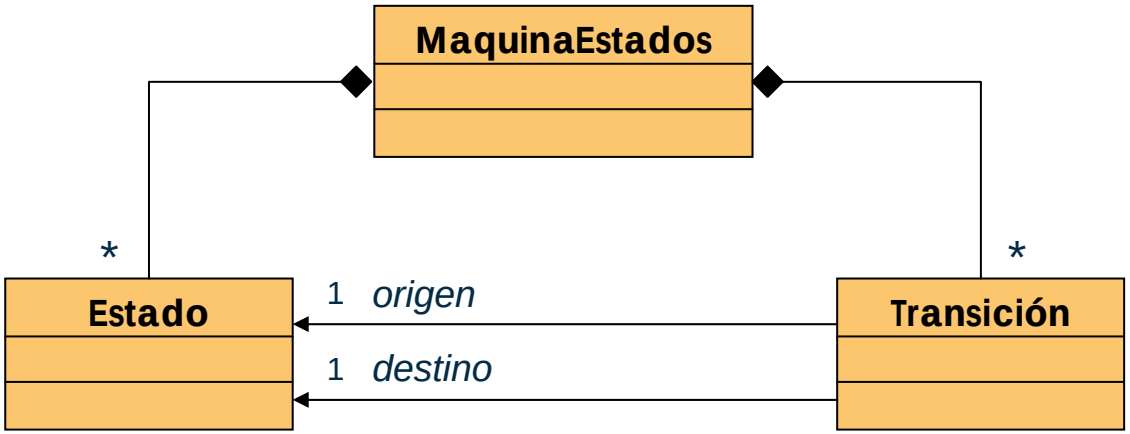
MOF

MDE

¡¡ Un “meta-infierno” de siglas !!

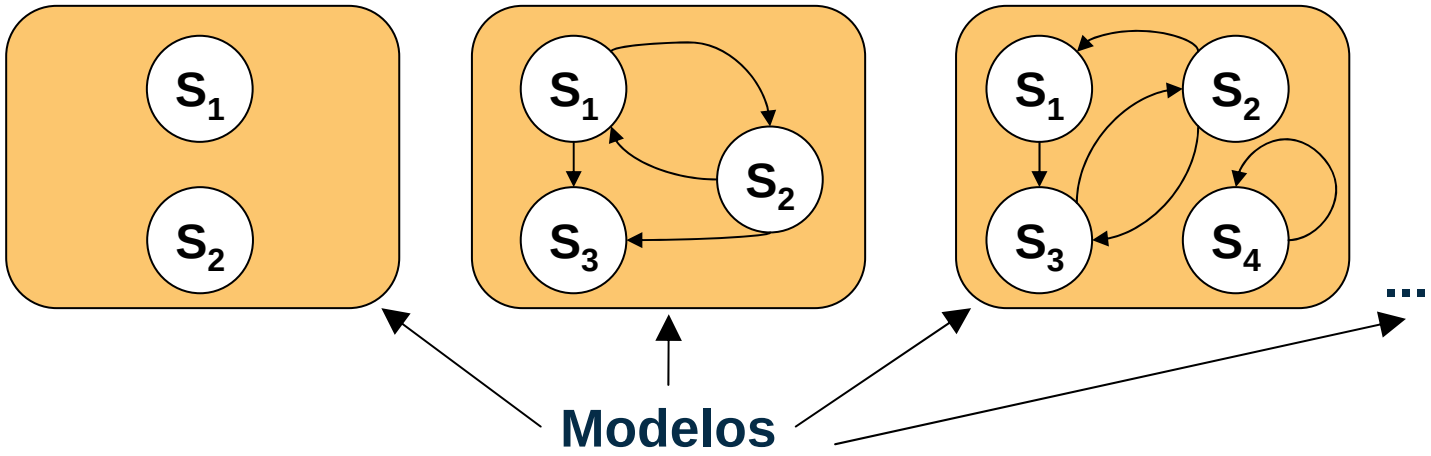
El meta-mundo… 2/4

- **Meta-Modelo**: conjunto finito de conceptos que se quieren modelar + conjunto de relaciones entre ellos.
  - Ejemplo 1: Palabras del diccionario de la lengua española + gramática de la lengua española.
  - Ejemplo 2: Meta-modelo para modelar máquinas de estados

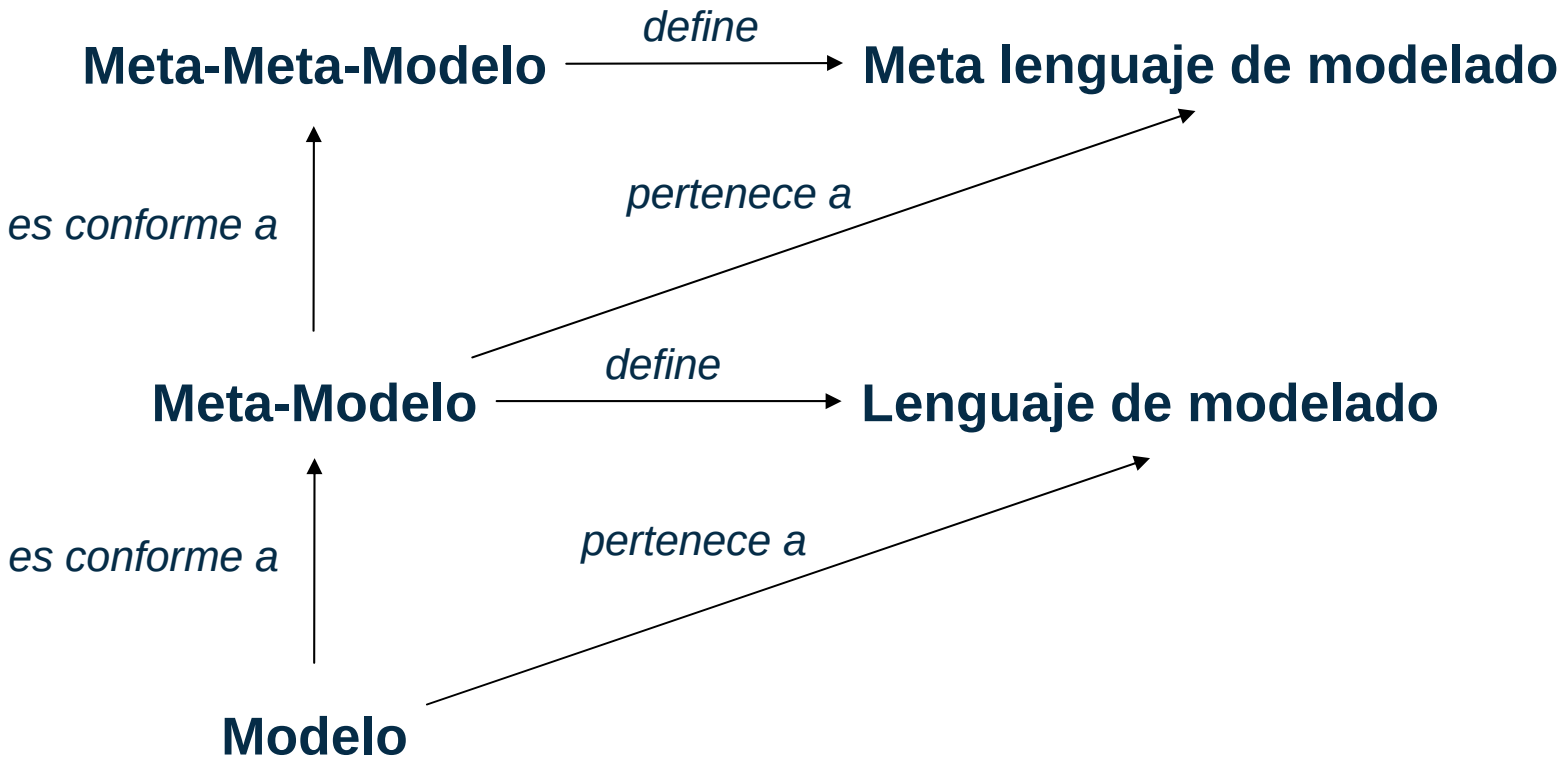


El meta-mundo… 3/4

- **Lenguaje de modelado**: conjunto infinito de todos los modelos válidos que se pueden construir a partir de un meta-modelo.
  - Ejemplo 1: Lenguaje español = conjunto de todas las posibles frases correctas que se pueden formar con las palabras del diccionario.
  - Ejemplo 2: Conjunto de todos los modelos de máquinas de estados que se pueden construir a partir del meta-modelo anterior. A continuación se muestran sólo algunos ejemplos:

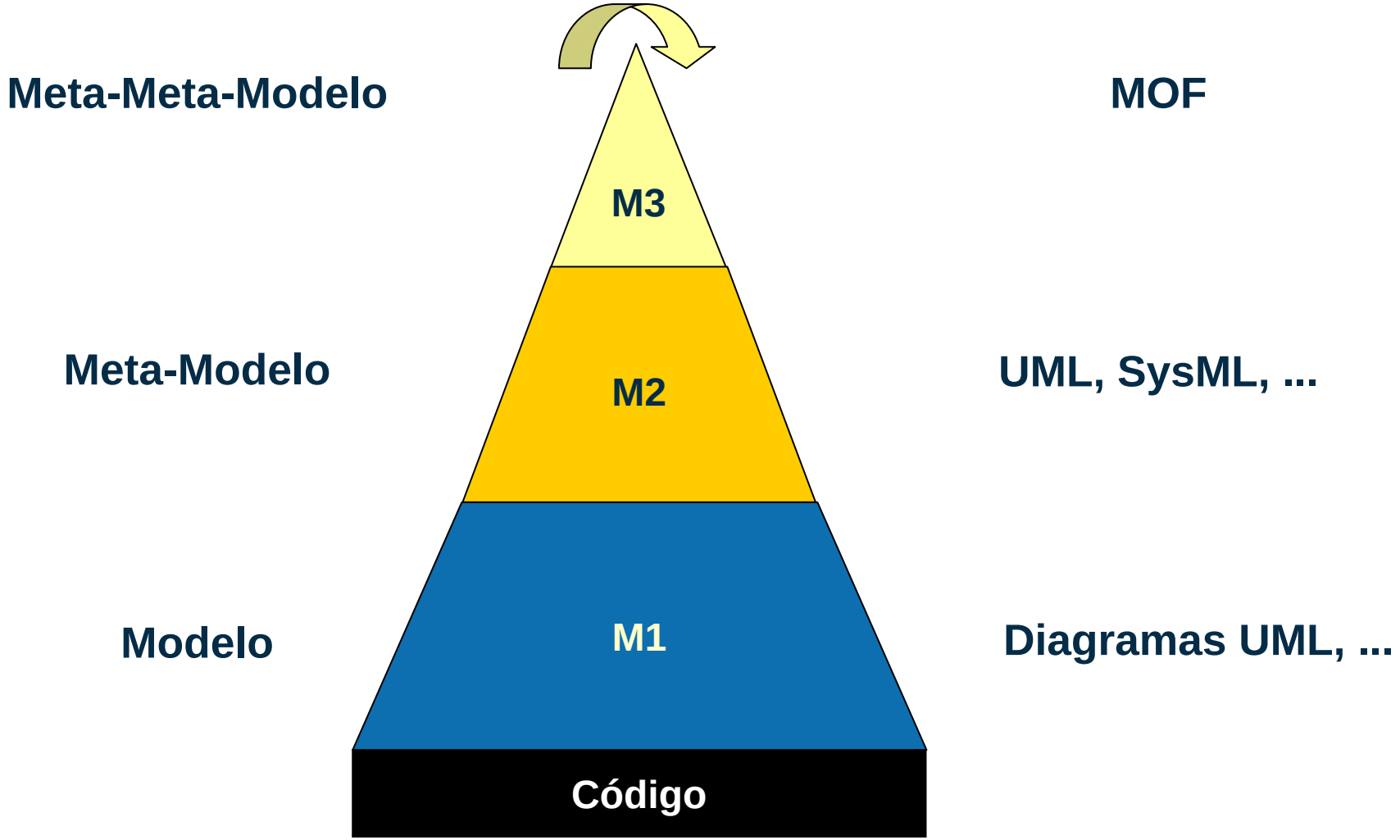


El meta-mundo… 4/4



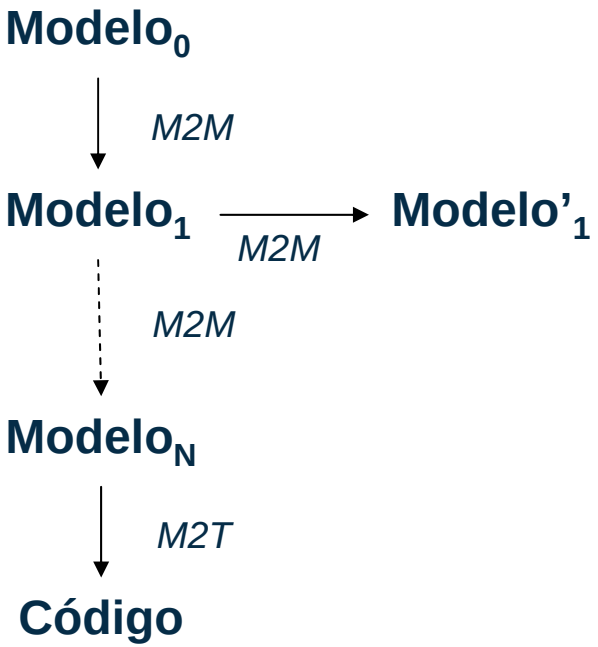
J. M. Favre, *Foundations of Meta-Pyramids: Languages vs. Metamodels Episode II: Story of Thotus the Baboon*

La visión MDE … 1/2



La visión MDE … 2/2

- Los modelos evolucionan mediante transformaciones definidas entre los correspondientes meta-modelos.
- Estas transformaciones pueden ser:
  - Modelo-A-Modelo (M2M)
    - Horizontales
    - Verticales
  - Modelo-A-Texto (M2T)



La visión MDA …

- MDA (*Model-Driven Architecture*) es la propuesta del OMG *(Object Management Group)* en el marco de MDE
- MDA gira en torno a otros estándares OMG:
  - MOF *(Meta-Object Facility)*
  - UML *(Unified Modelling Language)*
  - SysML *(Systems Modelling Language)*
  - OCL *(Object Constraint Language)*
  - XMI *(XML Metadata Interchange)*
  - *...*

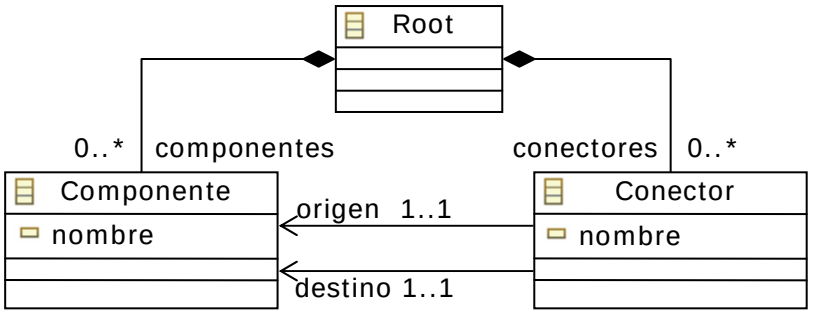
Profiling

- La propuesta MDA clasifica los (meta-) modelos en tres categorías:
  - CIM (*Computation Independent Model*)
  - PIM (*Platform Independent Model*)
  - PSM (*Platform Specific Model*)

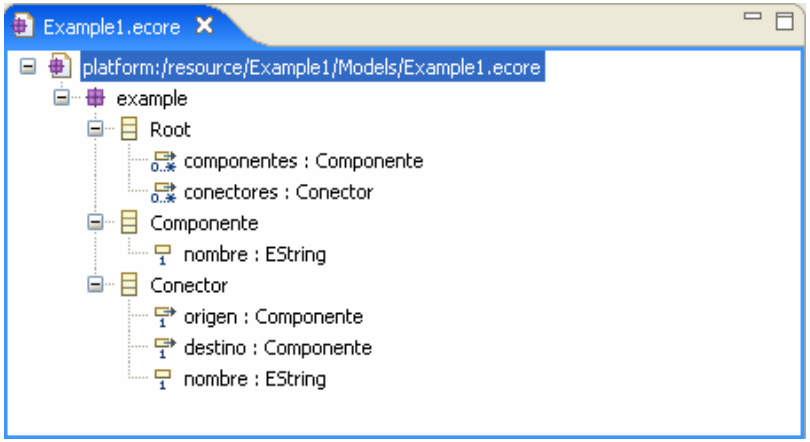
Introducción a EMF (= EMOF)

- **Meta-modelo EMF = diagrama de clases UML**
  - EClass = concepto del dominio
  - EAttribute = propiedad de una EClass en forma de tipo primitivo (int, boolean, String, float, enum)
  - EReference = relación entre conceptos:
    - Multiplicidad
    - Rol
    - Contención
    - Navegabilidad
  - EMF soporta herencia (especialización) múltiple.
- **Las relaciones de contención controlan la serialización de los modelos y la posición de los elementos en el editor.**

Un ejemplo de meta-modelo EMF



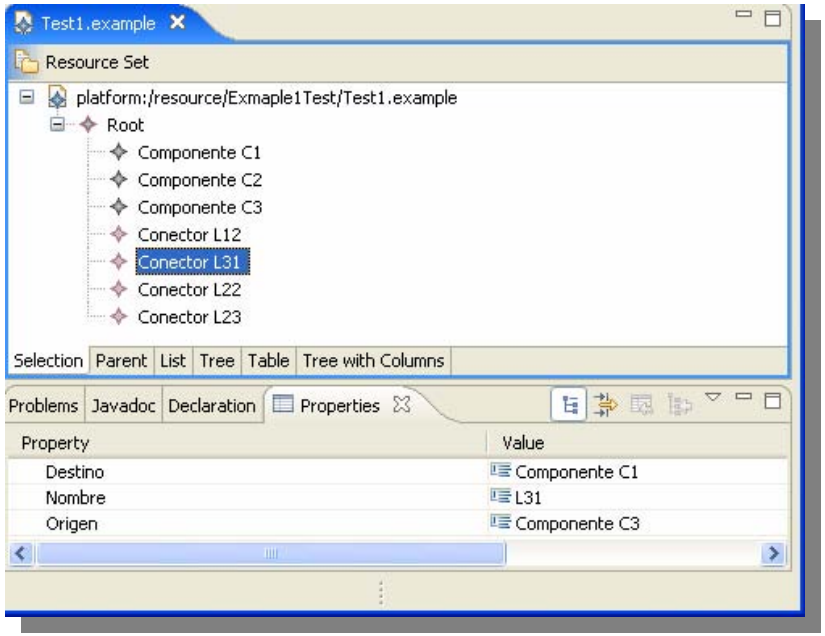
Example1.ecore_diagram

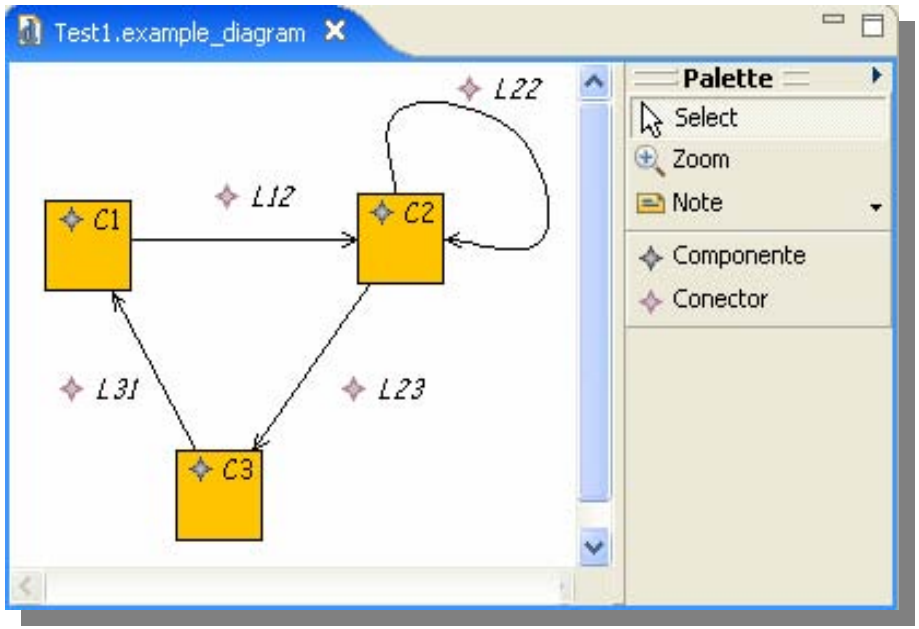


Example1.ecore

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
  …
 <eClassifiers xsi:type="ecore:EClass" name="Root">
      <eStructuralFeatures xsi:type="ecore:EReference"
        name="componentes" upperBound="-1"
        eType="#//Componente" containment="true"/>
      <eStructuralFeatures xsi:type="ecore:EReference"
        name="conectores" upperBound="-1"
        eType="#//Conector" containment="true"/>
 </eClassifiers>
 <eClassifiers xsi:type="ecore:EClass“ name="Componente">
       <eStructuralFeatures xsi:type="ecore:EAttribute"
        name="nombre" lowerBound="1"
        eType="ecore:EDataType
 </eClassifiers>
 <eClassifiers xsi:type="ecore:EClass" name="Conector">
       <eStructuralFeatures xsi:type="ecore:EReference"
        name="origen" lowerBound="1"
        eType="#//Componente"/>
       <eStructuralFeatures xsi:type="ecore:EReference"
        name="destino" lowerBound="1"
        eType="#//Componente"/>
      <eStructuralFeatures xsi:type="ecore:EAttribute"
       name="nombre" lowerBound="1"
       eType="ecore:EDataType
  </eClassifiers>
</ecore:EPackage>
```

Un ejemplo de modelo






```
<?xml version="1.0" encoding="UTF-8"?>
<example:Root xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:example="example">
        <componentes nombre="C1"/>
        <componentes nombre="C2"/>
        <componentes nombre="C3"/>
        <conectores origen="//@componentes.0" destino="//@componentes.1" nombre="L12"/>
        <conectores origen="//@componentes.2" destino="//@componentes.0" nombre="L31"/>
        <conectores origen="//@componentes.1" destino="//@componentes.1" nombre="L22"/>
        <conectores origen="//@componentes.1" destino="//@componentes.2" nombre="L23"/>
</example:Root>
```

Ejemplo práctico 1

- **Paso 1: Crear el meta-modelo (.ecore)**
  - Podemos utilizar el *tree-editor* de EMF o un editor gráfico de ecore como los proporcionados por GMF o TOPCASED.

- **Paso 2: Creación de un modelo a partir del meta-modelo**
  - Seleccionar la clase *Root* del meta-modelo con el botón derecho del ratón y pinchar en la opción "*Create dynamic instance*"
  - Dar un nombre al fichero xmi en el que se guardará el modelo
  - Añadir elementos al modelo seleccionando la opción "*New child*"
  - Editar las propiedades de dichos elementos en la vista "*Properties*"

- **Paso 3: Validación del modelo contra el meta-modelo**
  - Seleccionar el elemento *Root* del modelo con el botón derecho del ratón y pinchar en la opción "*Validate*"

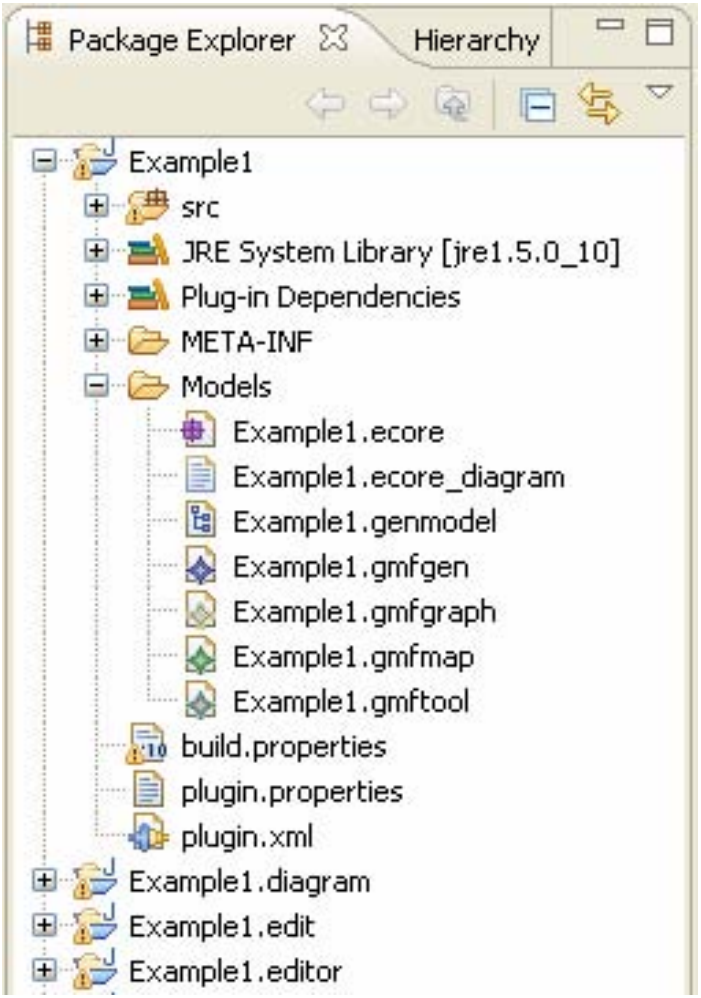
Introducción a GMF

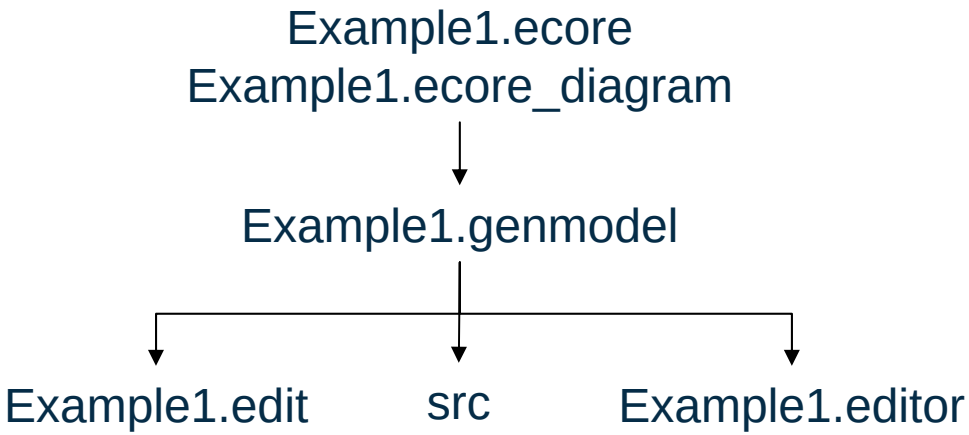
- **GMF es un plug-in Eclipse que permite crear editores gráficos de modelos a partir de meta-modelos EMF**

  http://www.eclipse.org/gmf

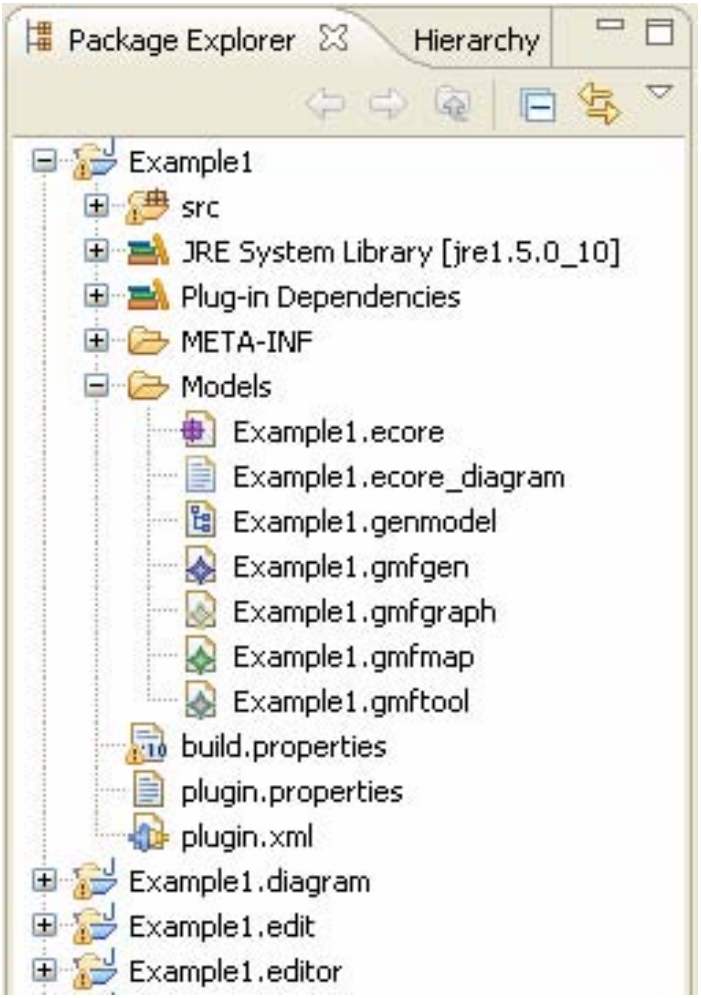
- **GMF depende de otros plug-ins Eclipse:**
  - **EMF** (Eclipse Modelling Framework)
    - Definición de meta-modelos
  - **GEF** (Graphical Editing Framework)
    - Definición de componentes gráficos
  - **EMF OCL/Query/Validation/Transaction**

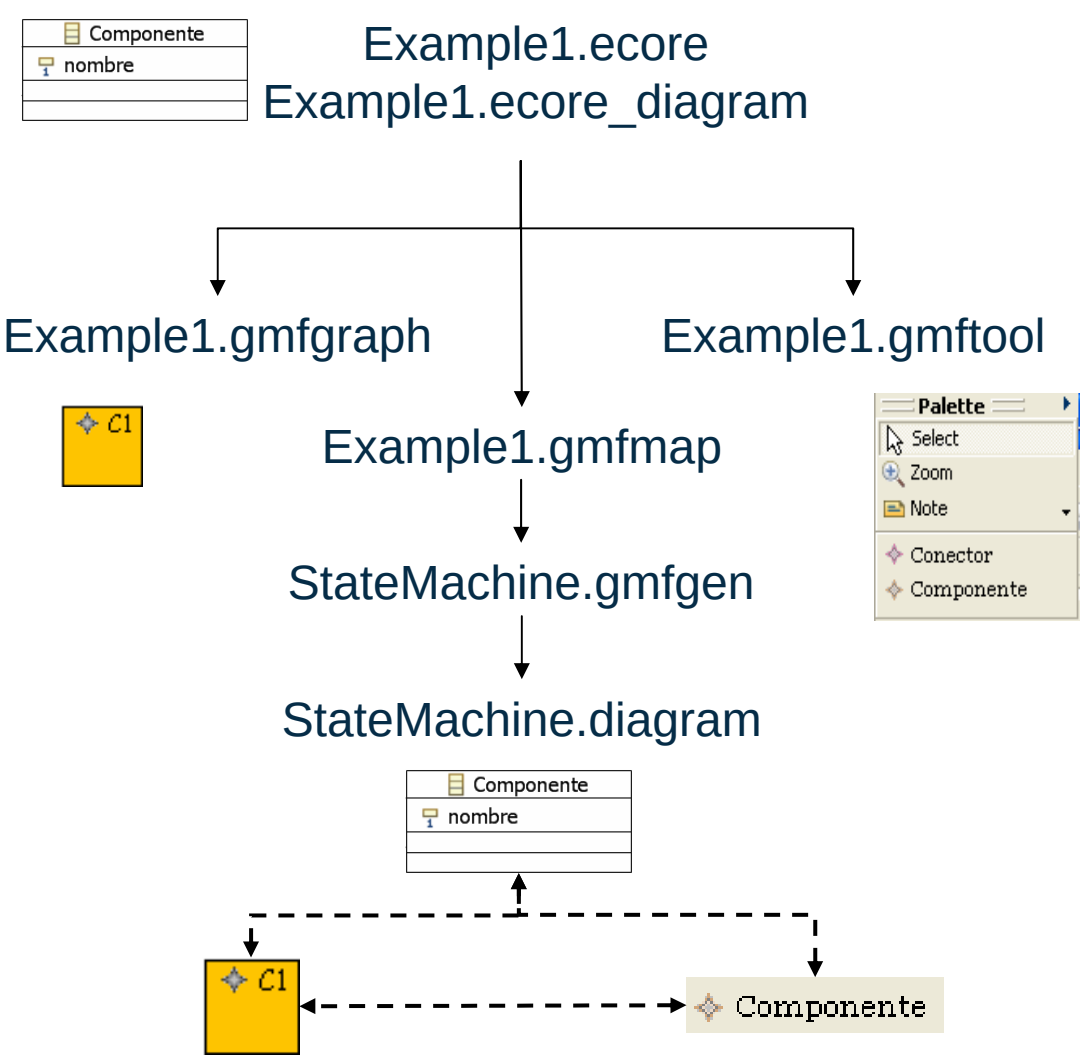
Estructura de un proyecto GMF 1/2





Estructura de un proyecto GMF 2/2





Ejemplo práctico 2

- **Paso 1: Generación de código**
  - Generar el fichero .genmodel a partir del .ecore
  - Generar el código asociado al meta-modelo (src), y a los editores NO GRÁFICOS (.edit y .editor) a partir del genmodel
- **Paso 2: Diseño de la aplicación GMF a partir del .ecore**
  - Elementos gráficos (.gmfgraph)
  - Paleta de herramientas (.gmftool)
  - Mapping de todos los elementos anteriores (.gmfmap)
- **Paso 3: Generación de la herramienta gráfica**
  - A partir del fichero .gmfmap generar el .gmfgen
  - A partir del .gmfgen generar la carpeta .diagram
- **Paso 4: Creación de un modelo gráfico**
  - Arrancar un nuevo Eclipse a partir del proyecto GMF (**Run→Run**)
  - **File → New → Other → Examples →** Example1 Diagram

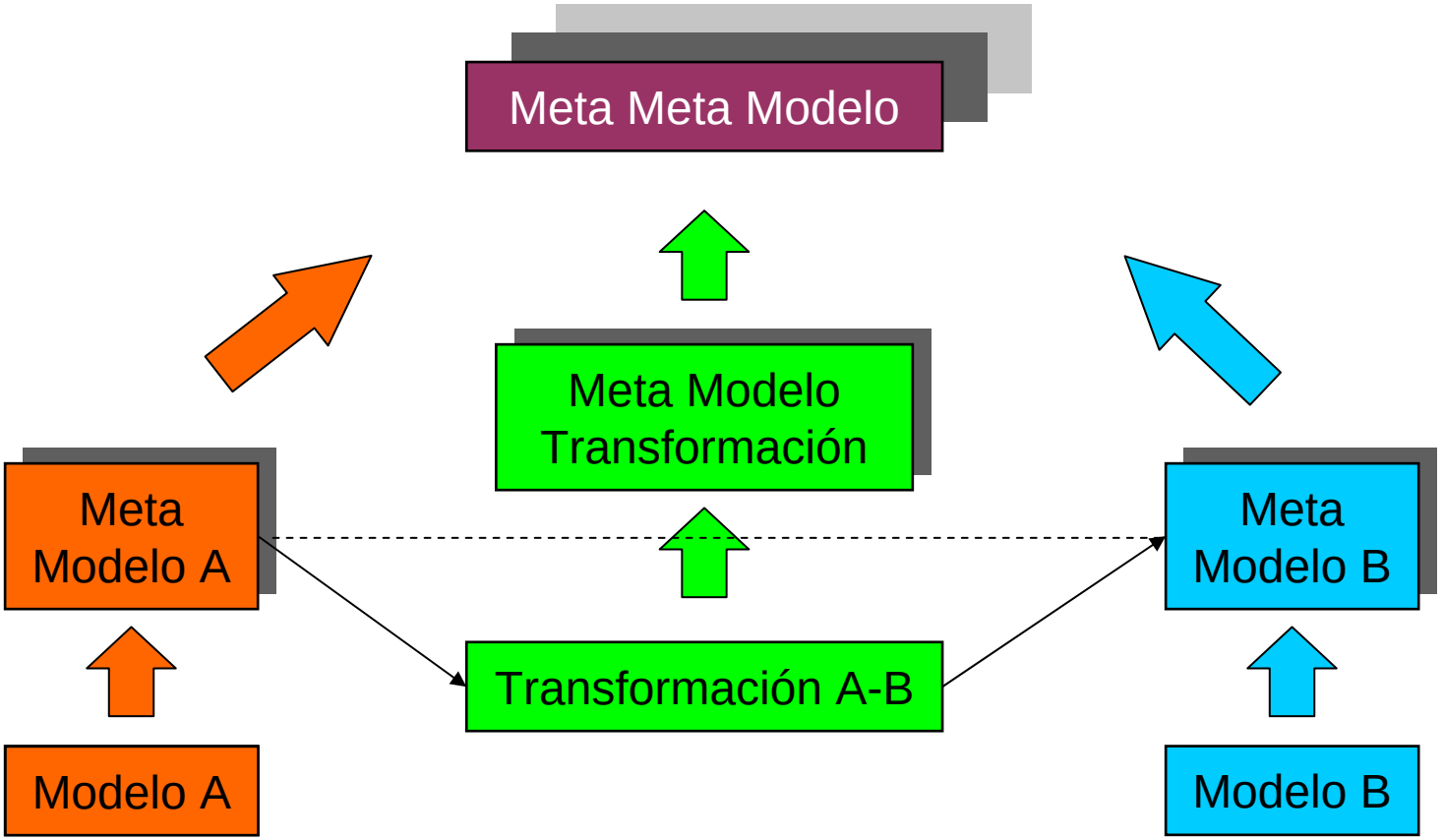
Restricciones OCL y validación

- Abrir el fichero Example1.gmfmap
- Añadir un nuevo elemento al *mapping* de tipo *Audit Container*
- Añadir un nuevo elemento al *Audit Container* de tipo *Audit Rule*
- Añadir un nuevo elemento al *Audit Rule* de tipo *Domain Element Target* y seleccionar en la vista de propiedades *Element = EClass Root*
- Añadir un nuevo elemento al *Audit Rule* de tipo *Constraint* y seleccionar Body en la vista de propiedades. Introducir la siguiente regla OCL:

  self.componentes->forAll (c1, c2 | c1<>c2 implies

  c1.nombre <> c2.nombre)
- Regenerar el ficheros .gmfgen a partir del .gmfmap
- Activar las propiedades de validación en el fichero .gmfgen
- Regenerar el código de la carpeta /diagram a partir del .gmfgen
- Volver a arrancar el entorno de modelado (F11 ó *Run Last Launched*)
- Añadir al modelo un componente con un nombre repetido
- Validar el modelo (***Diagram → Validate***) y comprobar que detecta error

Transformaciones de modelos



MDE: *«Todo es un modelo»*

Jean Bézivin

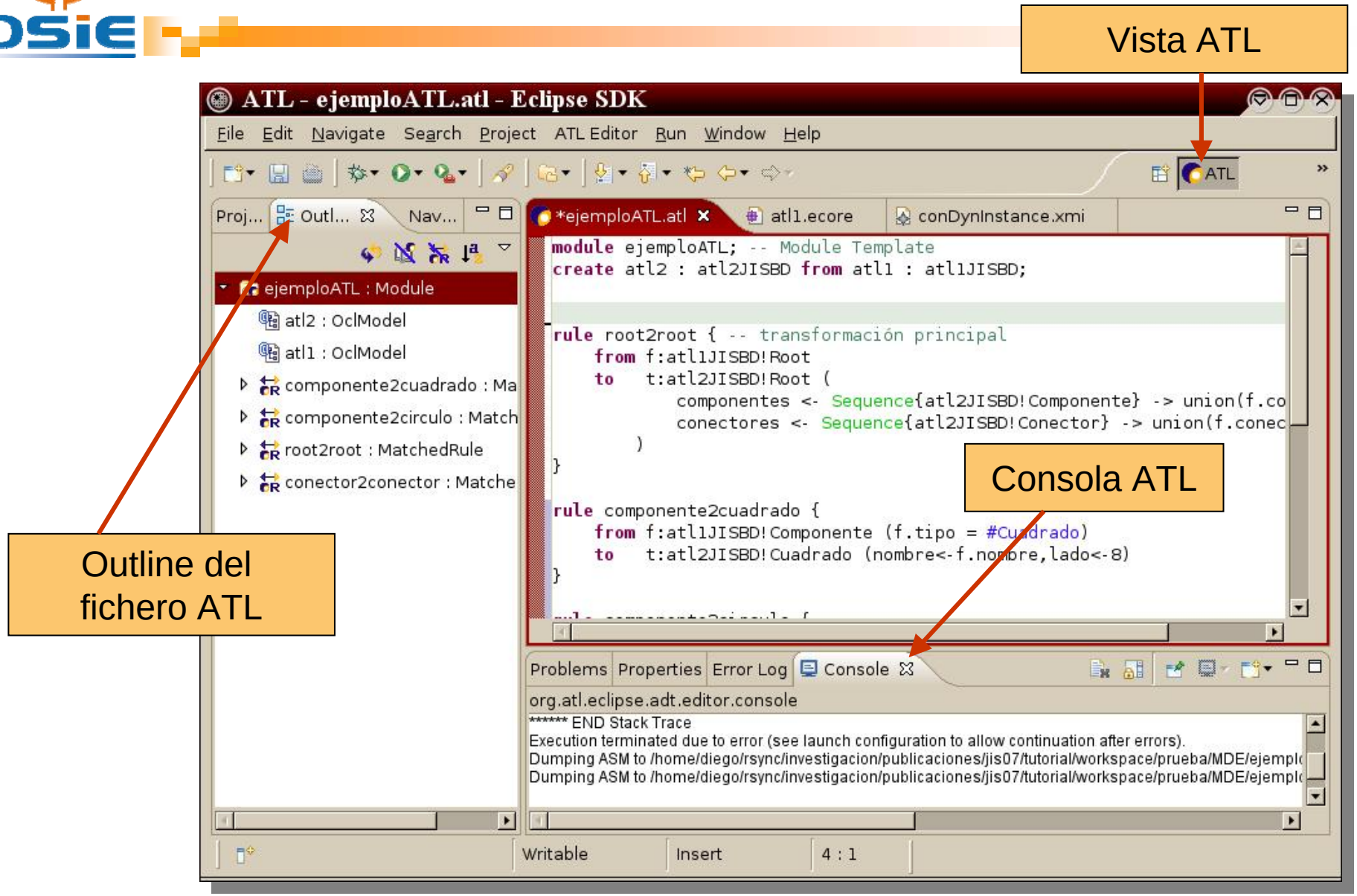
Herramientas Model-To-Model (M2M)

- **ATL** (*ATLAS Transformation Language*)
  - Desarrollado por el grupo ATLAS como respuesta al RFP-M2M del OMG
- **SmartQVT**
  - Primera implementación del estándar QVT 2.0 desarrollado en el marco del proyecto europeo Modelware
- **Xpand**
  - Integrado en el proyecto openArchitectureWare (oAW)
- Herramientas de manipulación XML

Breve descripción de ATL

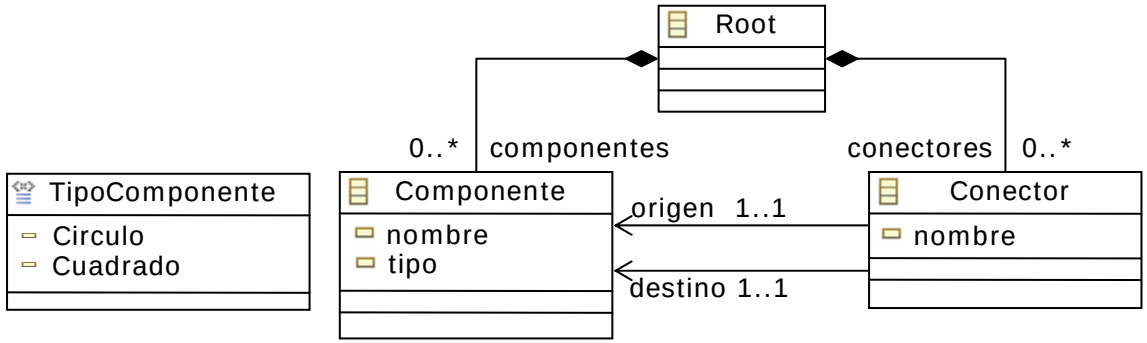
- ATL es un lenguaje mixto: imperativo/declarativo
- En ATL se pueden definir tres tipos de ficheros:
  - **Module**: transformación modelo-a-modelo
  - **Library**: funciones auxiliares reutilizables
  - **Query**: devuelve los elementos del modelo que cumplen determinadas propiedades o restricciones
- ATL permite transformaciones MIMO.
- Dos tipos de transformación: normal y refinamiento
- ATL tiene un depurador de transformaciones.
- ATL distingue mayúsculas de minúsculas.
- Comprobación de tipos en tiempo de ejecución. Difícil de depurar.

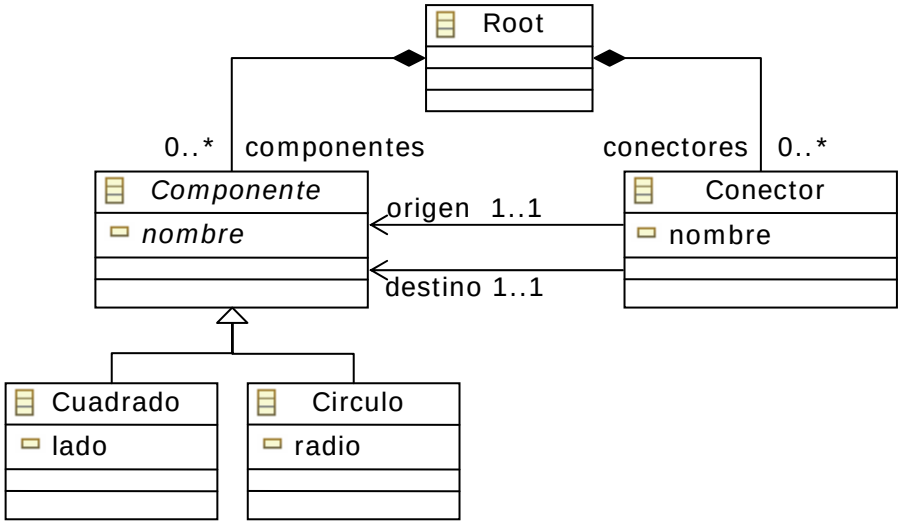
Entorno ATL en Eclipse



Adelanto del ejemplo práctico 3

Modelo de componentes → modelo de figuras





Fichero transformación .atl

1. Nombre de la transformación

   module *nombre;*

2. Definición de la transformación y de los modelos de entrada y salida:

   create modeloSal:MM_Sal [from|refines] modeloEnt:MM_Ent;

3. Definición de las reglas de transformación:
   - **Matched** rules: declarativas. Reglas principales, especifican cómo se transforma un elemento del meta-modelo de entrada en uno o varios elementos del meta-modelo de salida
   - **Helper** rules: auxiliares. Invocadas por el usuario. No crean elementos de meta-modelo destino
   - **(Lazy) Called** rules: auxiliares. Invocadas por el usuario. Crean elementos de meta-modelo destino

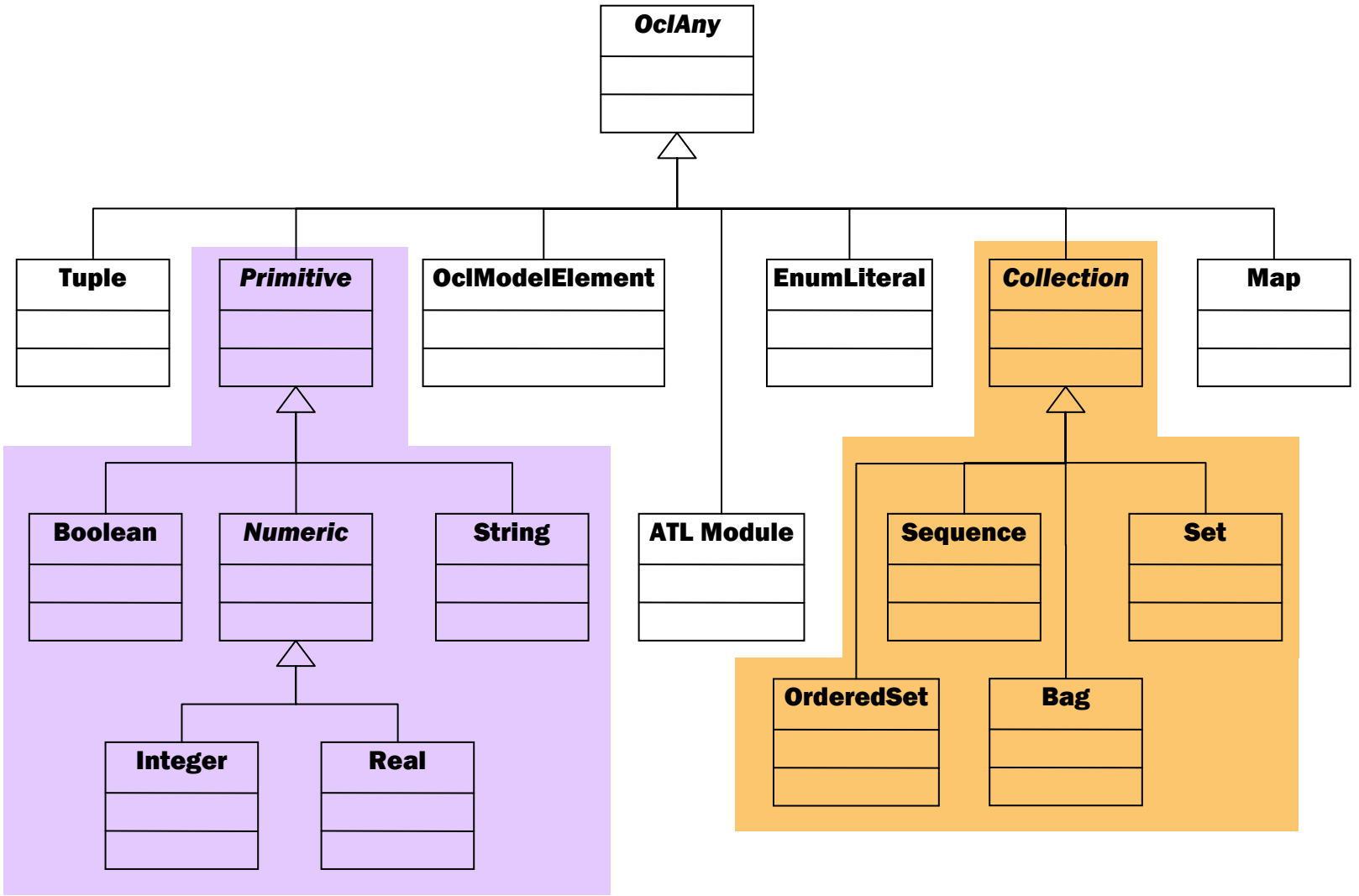
Definición de reglas (matched rules)

```
rule nombre {
    from nombreLocalFrom : MM_Ent!EClass (condicion)
    to   nombreLocalTo : MM_Sal!EClass (constructor)
}
```

- La *condición* es opcional y controla la ejecución de la regla. Utiliza *nombreLocalFrom*
- El *constructor* es obligatorio. Inicializa todos los campos del elemento. Usa el elemento '*from*'
- Cada elemento del modelo sólo pueden ser seleccionado por una única regla (determinista). Ojo con la herencia!

```
rule componente2cuadrado {
    from f : MM_origen!Componente (f.tipo = #"Cuadrado")
    to   t : MM_Destino!Cuadrado (nombre<-f.nombre, lado<-8)
}
```

Tipos de datos en ATL



Operaciones sobre colecciones

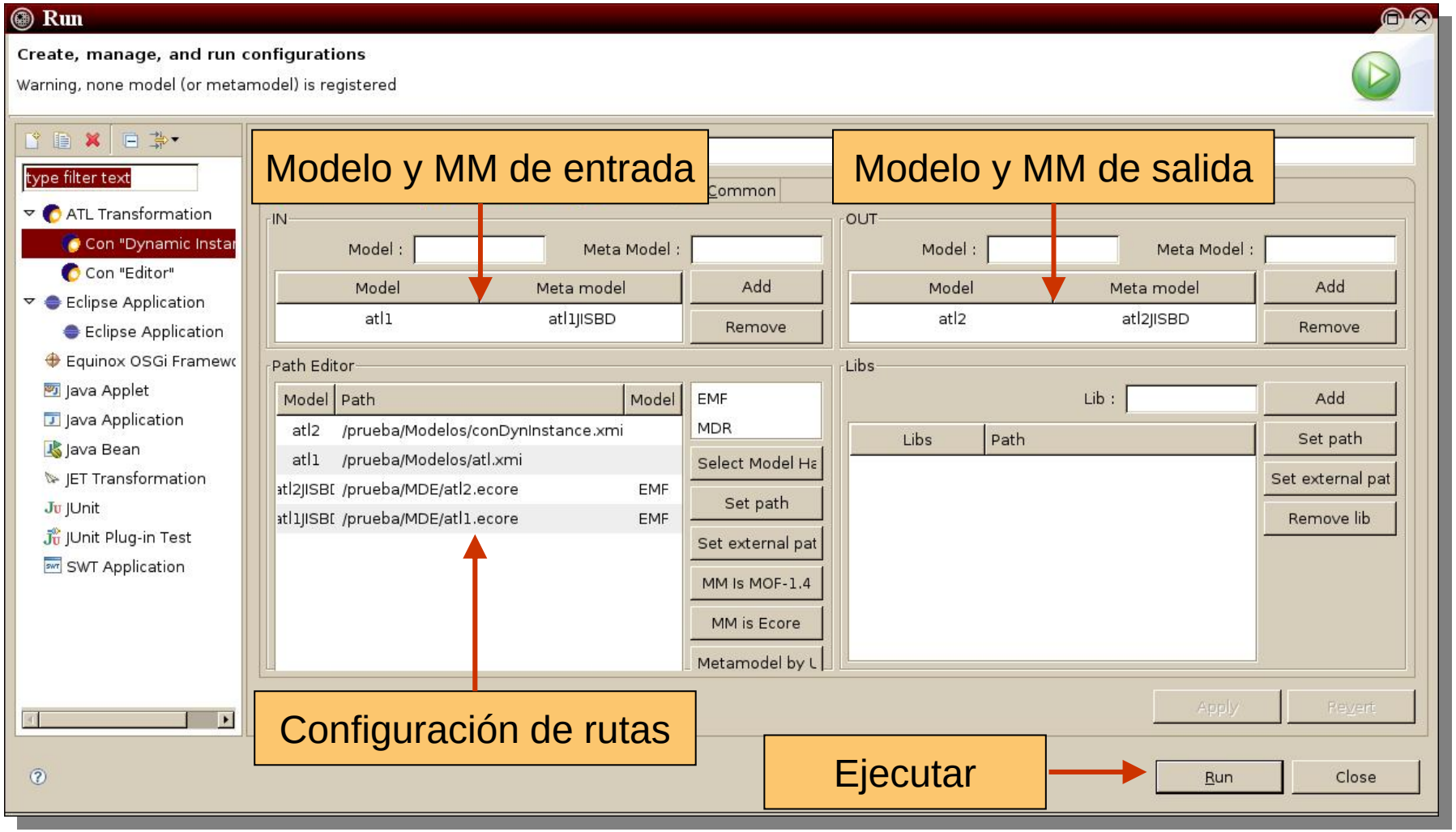
- Operaciones generales:
  - **size, isEmpty, count, includes, …**
- Sobre colecciones:
  - *Set*: **union, intersection, …**
  - *OrderedSet*: **append, insertAt, last, …**
  - *Sequence*: **union, prepend, indexOf, first, …**
- Iteradores (aplicables a cualquier colección):

  colección -> *operador* (iterador | cuerpo)

  Operadores: **exists, forAll, select, collect, any, …**

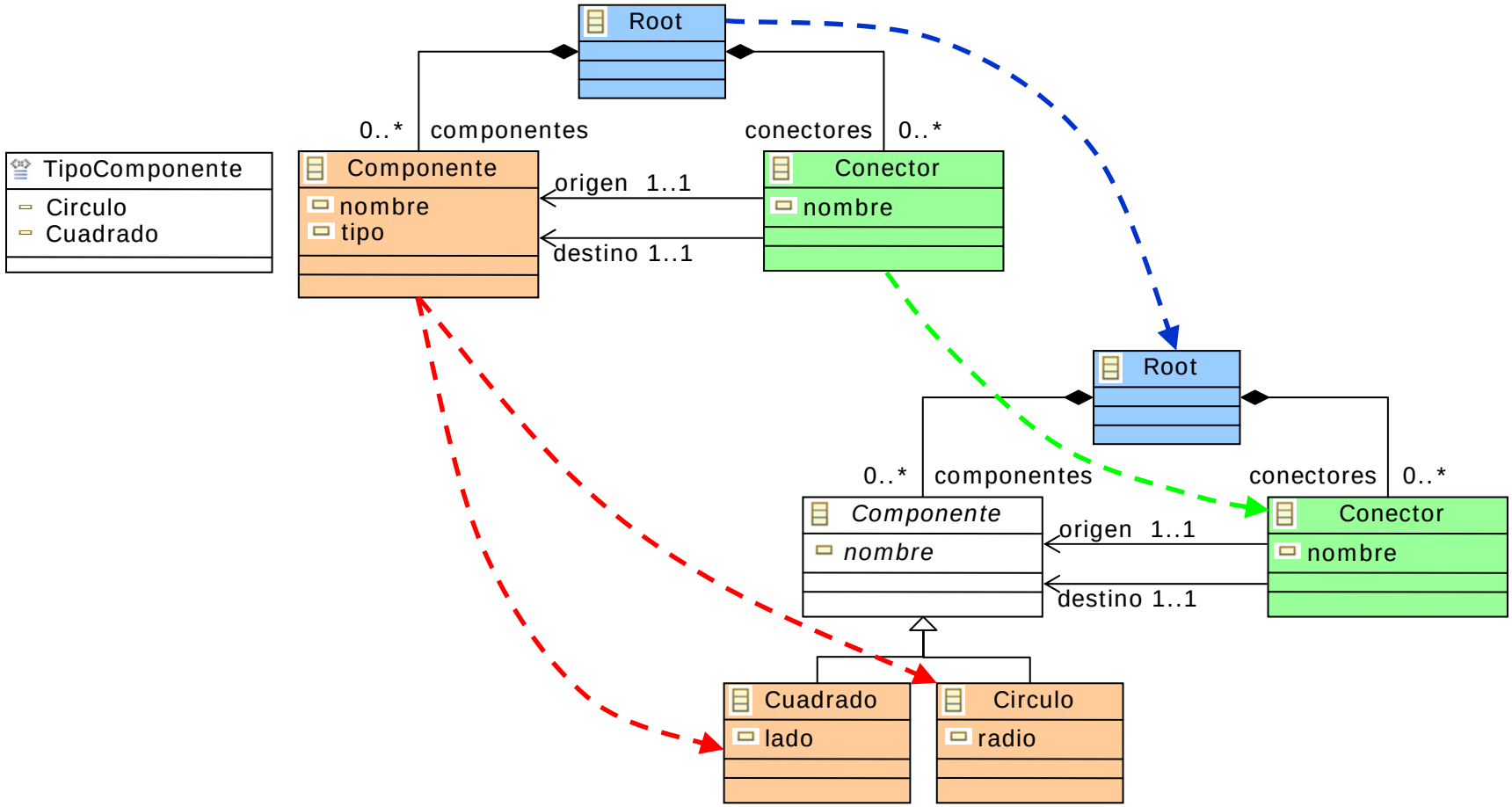
  Root.componentes -> exists (i | i.name='caja 1')

Ejecución de transformaciones ATL



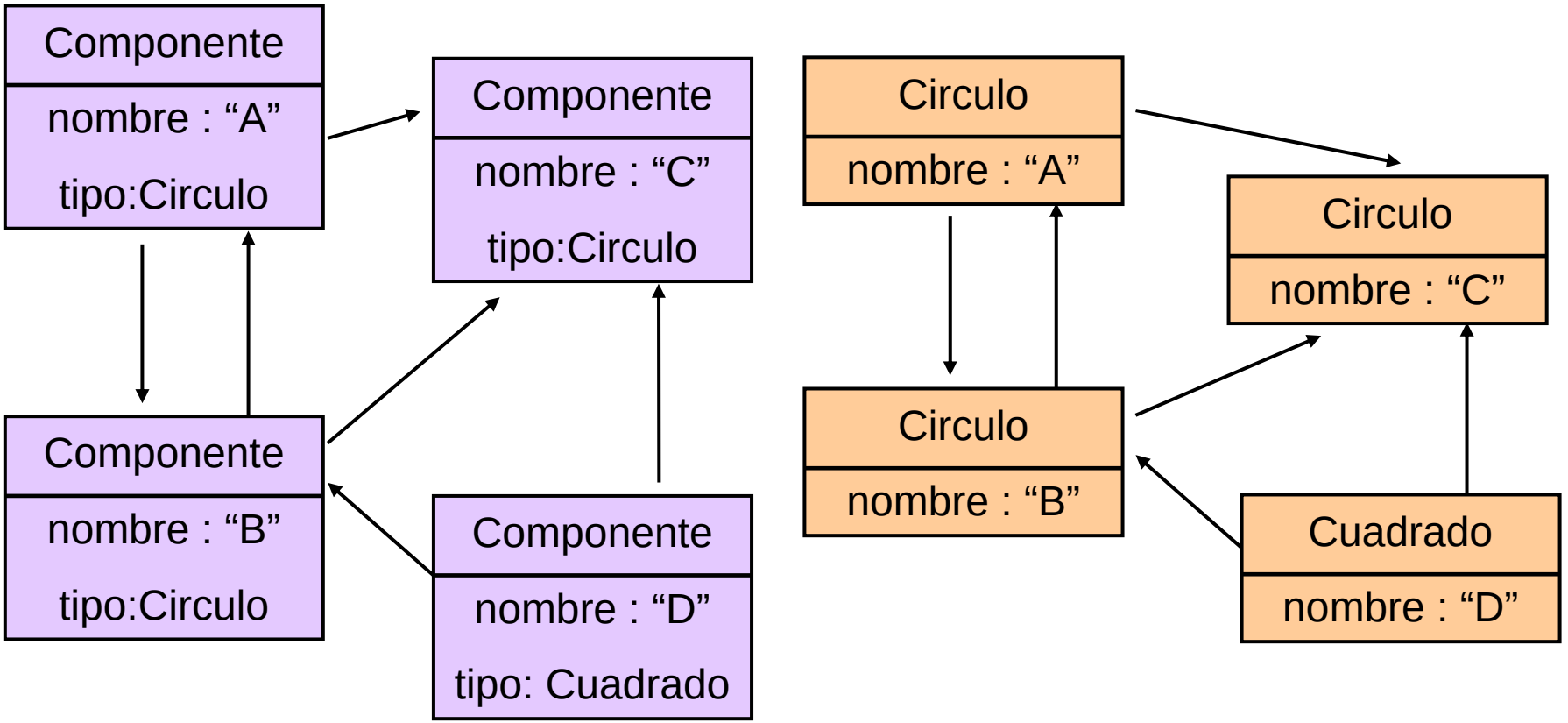
Ejemplo Práctico 3 1/2

Modelo de componentes → Modelo de figuras



Ejemplo Práctico 3 2/2

Transformación de modelos de componentes en modelos de figuras



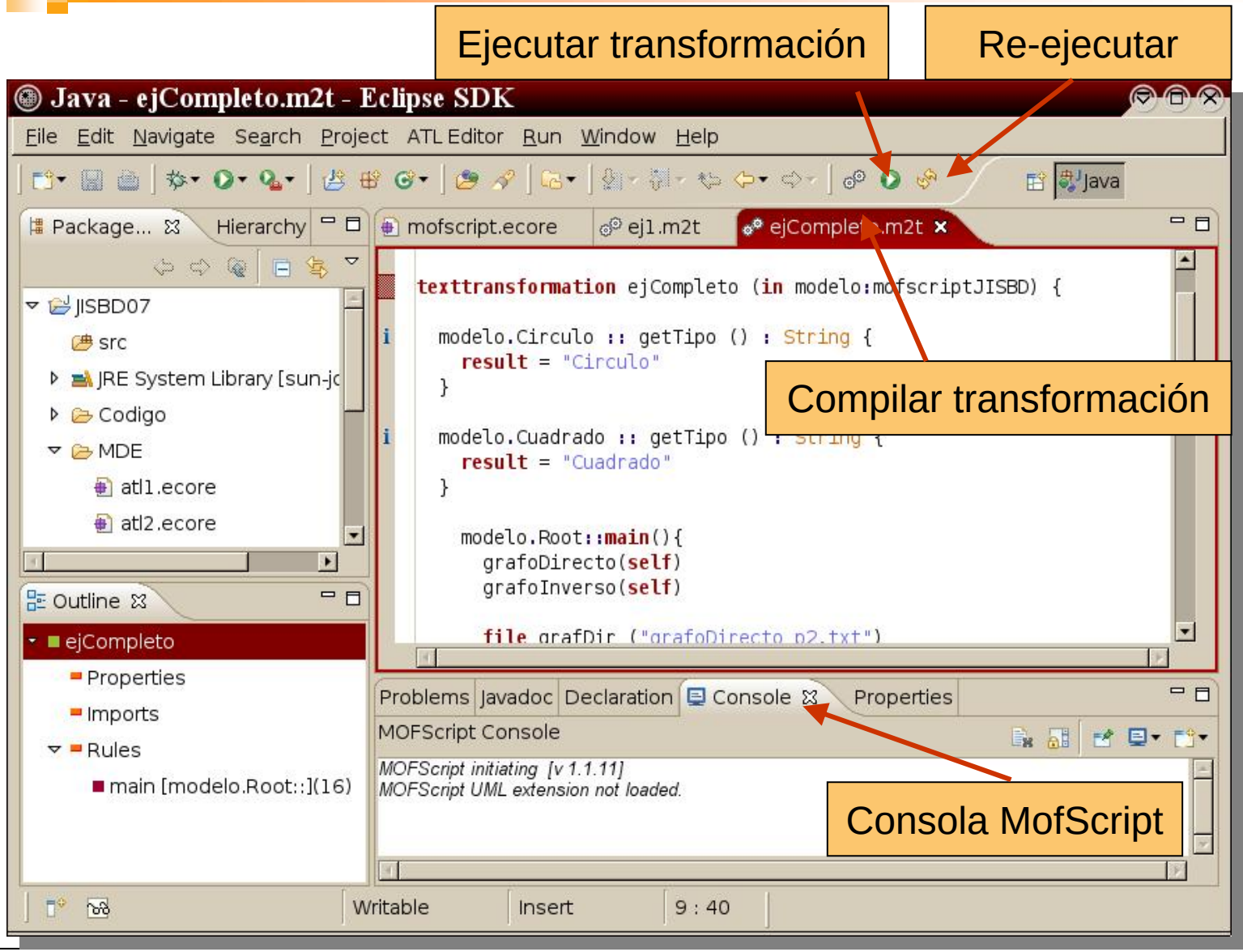
Herramientas Model-to-Text (M2T)

- **Soluciones basadas en plantillas:**
  - **JET** *(Java Emmiter Templates):* integrado en EMF, está inspirado en JSP
  - **VT** (*Velocity Templates*): proyecto de la fundación Apache, integrable en Eclipse
- **Soluciones basadas en lenguajes:**
  - **MofScript**: parte del proyecto Modelplex (6th FP), única contribución al RFP-M2T de la OMG.
  - **XPand**: integrado en openArchitectureWare (oAW)

Breve descripción de MofScript

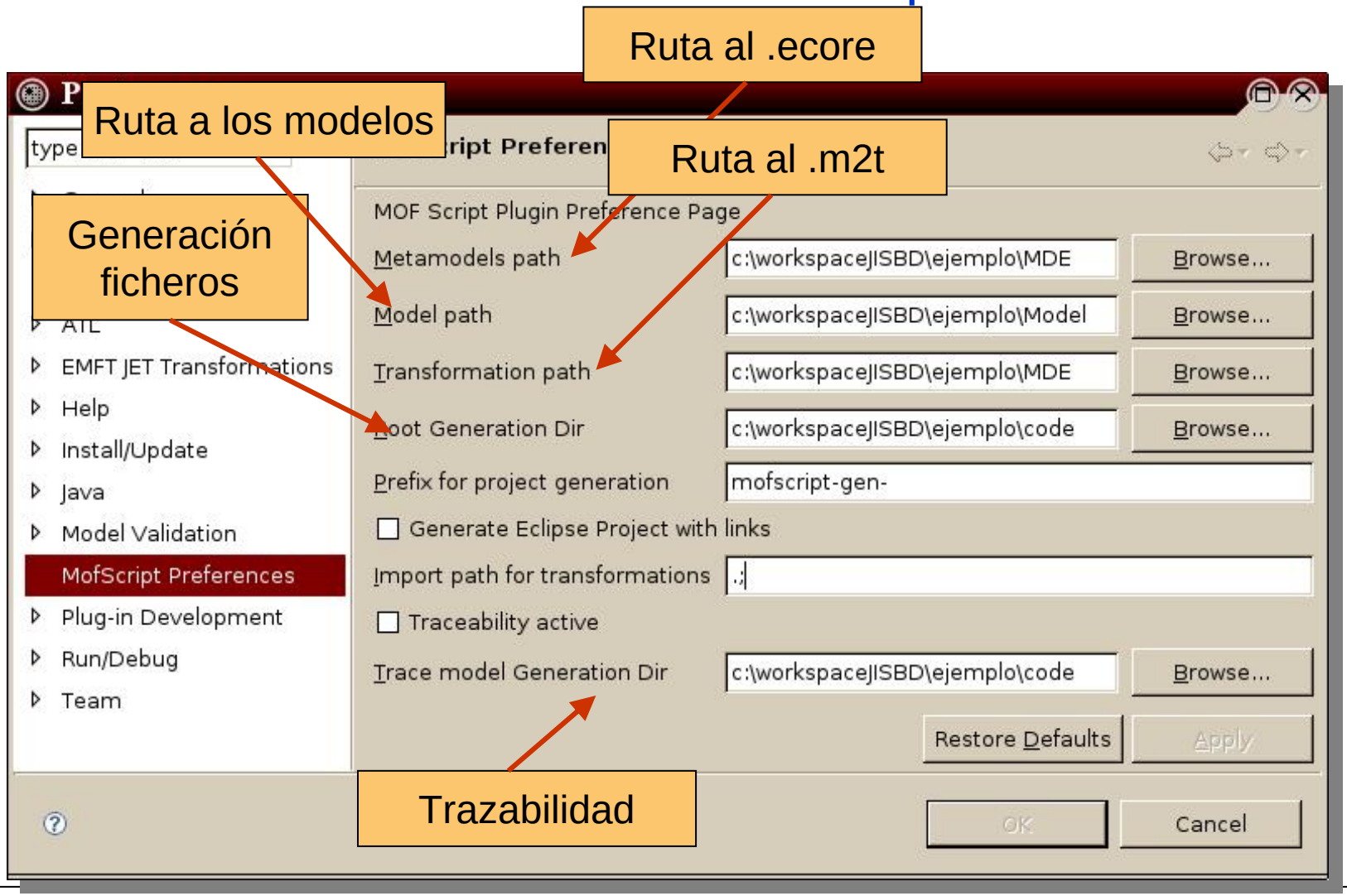
- Depende del plug-in **ANTLR**
- Proporciona: editor con coloración de sintaxis, compleción de código y sistema de trazabilidad para las transformaciones
- Admite múltiples modelos de entrada
- Lenguaje mixto: declarativo/imperativo
  - *Declarativo*: definición de reglas de transformación
  - *Imperativo*: descripción de las reglas de transformación y funciones auxiliares
- Distingue mayúsculas de minúsculas
- Las líneas pueden acabar en punto y coma

Entorno MofScript en Eclipse



Diálogo de preferencias de MofScript

Window → Preferences → MofScript Preferences



Cabecera fichero transformación .m2t

1. **Inclusión de otras transformaciones (op):**

   import "*nombre_fichero.m2t*"

2. **Definición de la transformación y de los meta-modelos de entrada:**

   texttransformation *ejemplo* (in *modelo1: MM1*, in *modelo2: MM2*) {

3. **Transformación principal (sólo una):**

```
modelo1.Root_EClass :: main() {   // regla de transformación
    // bloque imperativo
}
/*   Resto de reglas de transformación (declarativo)   */
}
```

Tipos de datos en MofScript

- Existen dos familias de tipos:
  - Tipos predefinidos:

    String, Integer, Real, Boolean, Hashtable, List y *Object*
  - Tipos definidos en los meta-modelos de entrada:

    EClass y *EEnum*
- Existe un tipo especial 'fichero':

  file *nombre* ("*nombreFich*")
  - Sólo existen en el ámbito de declaración de una regla.
  - **NO** se pueden utilizar como variables globales ni como parámetros.
- Para imprimir se utiliza print o println:

```
file fichero ("prueba.txt")
fichero.println ("Esto se escribe en el fichero")
stdout.println("Esto se escribe en la salida de la consola")
```

Instrucciones imperativas

- Variables y constantes (globales o locales):

```
var nombre : tipo = valorInicial;
property nombre : tipo = valorInicial;
```

- Operadores lógicos: not, or, and, =, !=, <,>, >=, <=

```
if (condLógica) {
    // bloque código
} else {
    // bloque código
}
```

```
while (condLógica) { … }

5->forEach (i) { … }

“tutorial ”->forEach (c) { … }
```

Operaciones sobre colecciones

- Colecciones predefinidas:
  - *List*: **add, remove, isEmpty, forEach, …**
  - *Hashtable*: **put, get, size, forEach, …**
- Colecciones de elementos de un modelo (multiplicidad distinta de 1):
  - **size, first, isEmpty** y **forEach**
- Iterador sobre colecciones (forEach):

```
self.conectores->forEach (c : modelo.Conector) {
    stdout.println (“Nombre conector = ”+c.nombre);
}
```

Definición de reglas (parte declarativa)

contexto :: nombreRegla (parámetros) : tipoRetorno

- *Contexto*: elemento del meta-modelo al que nos referimos cuando utilizamos self en una regla:
  - Con contexto:

    modelo.Componente :: add (valor : Integer) : Boolean

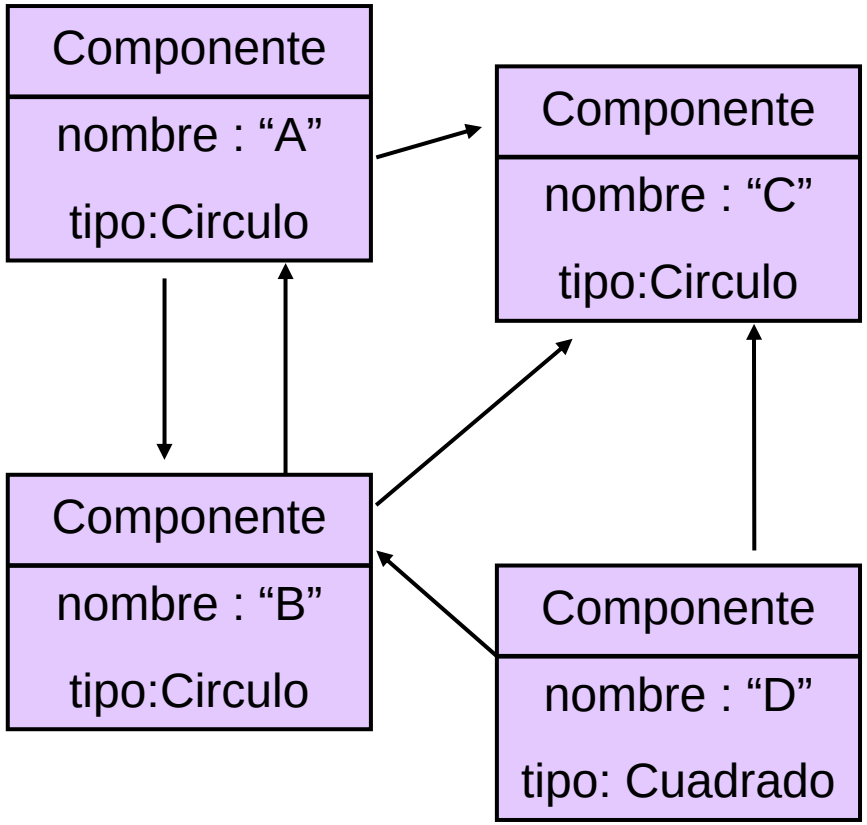
  - Sin contexto: module (función auxiliar)

    module :: cabecera (nombre : String, item : modelo.Conector)

- Parámetros: nombreLocal:Tipo
- Tipo de retorno (opcional): los predefinidos o los incluidos en el meta-modelo.
- Variable de retorno: result.

Ejemplo Práctico 4

Creación de un fichero de texto que enumera todos los componentes directamente conectados a otro en el modelo de entrada.



grafoDirecto.txt
A :: B, C
C ::
D :: B, C
B :: A, C

Cabecera fichero transformación .m2t

1. **Creación de un fichero de transformación:**

   File → New → Other → MofScript file

2. **Declaración cabecera y regla principal:**

```
texttransformation ej1 (in modelo:mofscriptJISBD) {
        modelo.Root :: main ( ) {
        }
}
```

3. **Definición de la regla principal:**

```
file grafDir (“grafoDirecto.txt”);
self.componentes->forEach (c : modelo.Componente) {
        grafDir.print (c.nombre + “ :: ”);
        …
}
```

Otras posibles transformaciones

1. Obtener todos los componentes que apuntan a otro (grafo inverso del ejemplo anterior).
2. Modificar ambas transformaciones para añadir, entre paréntesis, el tipo de componente (círculo o cuadrado):

```
self.componentes->forEach (c : modelo.Component) {
        if (c.oclIsTypeOf (modelo.Circulo))  { // completar }
        else if (c.oclIsTypeOf (modelo.Cuadrado))  { // completar }
}
```

3. Definir cada transformación como una regla e incluirlas todas en el mismo fichero

Algunas referencias bibliográficas

- F. Budinsky, et al., *Eclipse Modelling Framework*, Addison-Wesley Professional, 2003.
- J. Bézivin, "*On the Unification Power of Models*", Journal of Software and Systems Modelling 4(2), pp. 171-188, 2005.
- D. Schmidt, *"Model-Driven Engineering",* IEEE Computer 39(2), pp. 25-31, 2006.
- S. Sendall, and W. Kozaczynski, "*Model transformation: the hart and soul of model-driven software development*", IEEE Software 20(5), pp. 42-45, 2003.
- S. Beydeda, M. Book, and V. Gruhn (Eds.)*, Model-Driven Development*, Springer-Verlag, 2005.
- T. Stahl, M. Völter, *Model-Driven Software Development,* John Wiley & Sons, Ltd., 2006.

Más sobre estos plug-ins…

- Todas estas herramientas de soporte MDE están en fase de desarrollo y se liberan nuevas versiones cada poco tiempo.
- Existen numerosas incompatibilidades entre las distintas versiones que se liberan.
- Conflicto de versiones.
- Los manuales están, como mínimo desactualizados
- News de Eclipse (¡¡ la mejor fuente de ayuda !!)

news://news.eclipse.org/

Herramientas Eclipse para Desarrollo de Software Dirigido por Modelos

¡¡ Gracias por su atención !!

Cristina Vicente Chicote
Teléfono: (+34) 968 32 6448
E-mail: Cristina.Vicente@upct.es

Diego Alonso Cáceres
Teléfono: (+34) 968 32 5341
E-mail: Diego.Alonso@upct.es

División de Sistemas e Ingeniería Electrónica (DSIE)
Departamento de Tecnologías de la Información y Comunicaciones
Escuela Técnica Superior de Ingeniería de Telecomunicación
Edificio Antigones, Plza. del Hospital Nº 1, 30202 Cartagena
Universidad Politécnica de Cartagena