Procesadores de Lenguajes 2

Desarrollo de editores textuales con Xtext

Curso 2013-2014

Iván Ruiz Rube
Departamento de Ingeniería Informática
Escuela Superior de Ingeniería
Universidad de Cádiz

En la clase anterior…

- El framework GMF permite construir editores de modelos visuales, con capacidades avanzadas.
- Proceso de construcción basado en la transformación sucesiva de modelos.
- Mapping entre elementos del metamodelo, metáforas visuales y herramientas de creación.

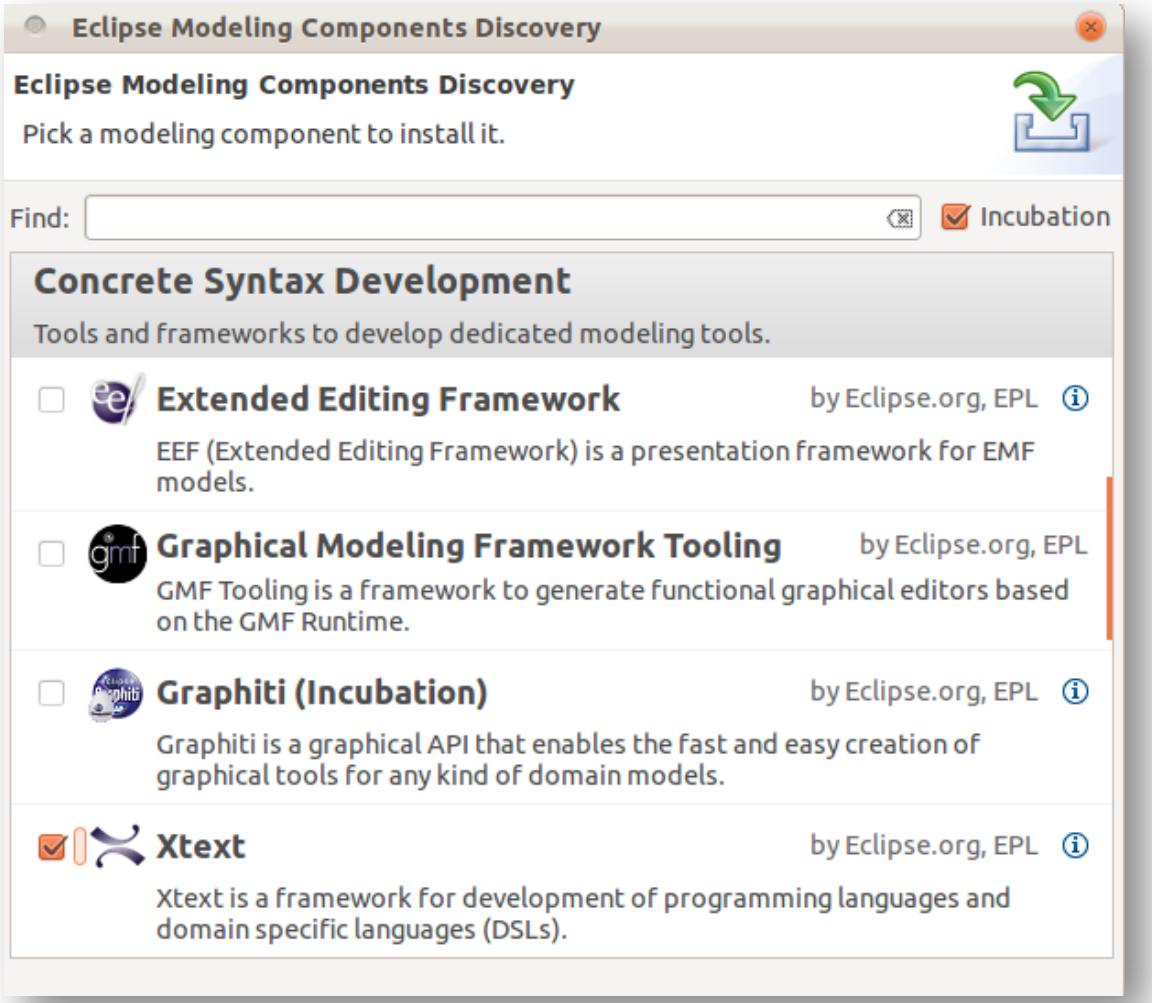
Contenidos

- Instalación
- Introducción
- Características
- EMF y Xtext
- Diseño de gramáticas
- Desarrollo de un editor textual

DESARROLLO DE EDITORES TEXTUALES CON XTEXT

INSTALACIÓN

Instalación Xtext



`Help → Install Modeling Components`

DESARROLLO DE EDITORES TEXTUALES CON XTEXT

INTRODUCCIÓN

Xtext

- Proyecto open source para el desarrollo de lenguajes textuales, liderados por la empresa Itemis AG.
- Xtext genera automáticamente los componentes necesarios para trabajar con los DSL: parser, analizador estático, formateador de código, generador de código, etc.
- Pueden utilizarse desde dentro de Eclipse o de forma independiente.

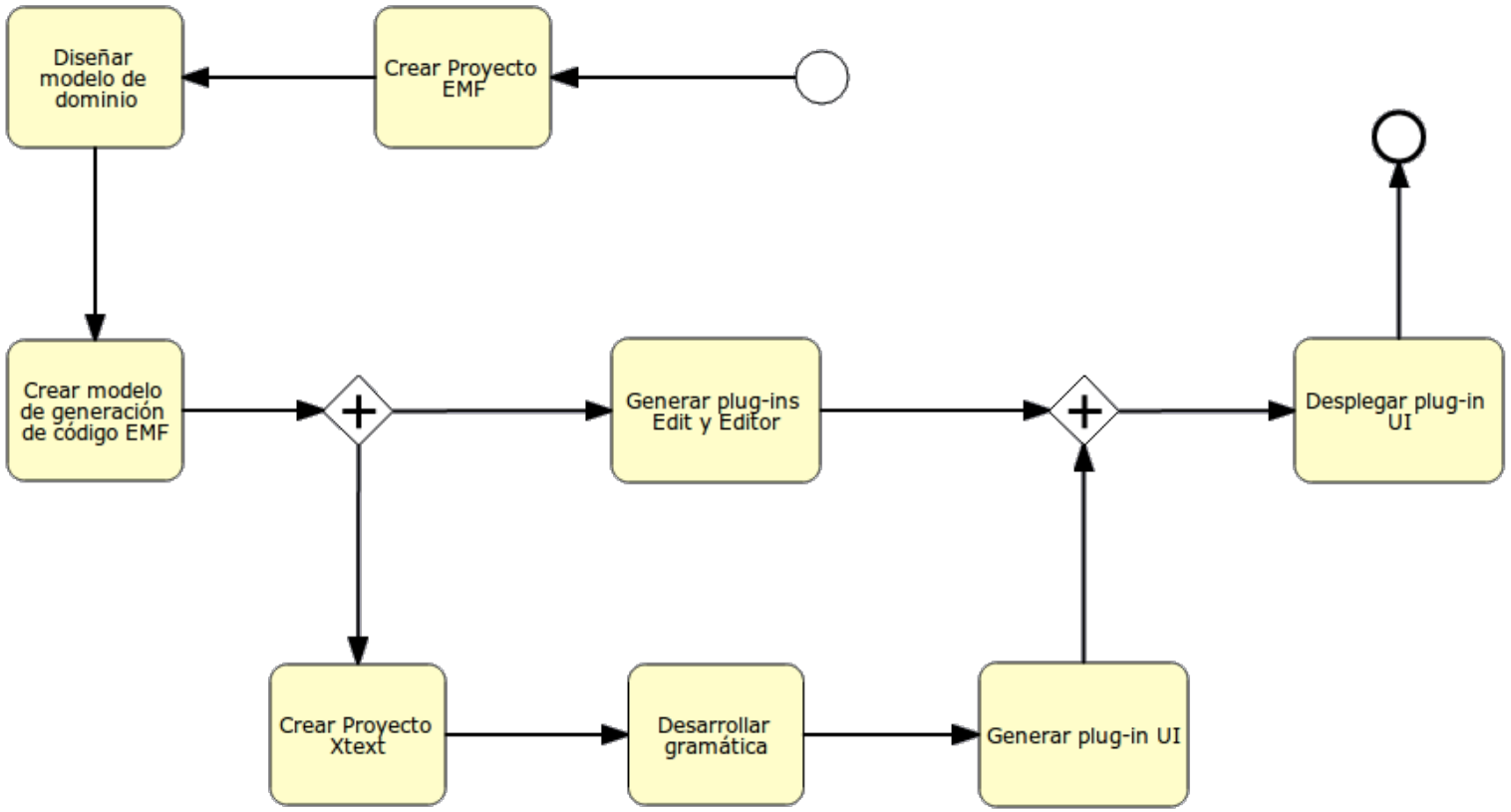
Xtext

- Xtext incluye un lenguaje para diseñar gramáticas y una amplia API para definir los diferentes aspectos de los DSL.
- Utiliza Google Guice, un framework ligero para la inyección de dependencias, con el cual podemos sustituir fácilmente cualquier comportamiento por defecto.
- Se basa en ANTLR para generar analizadores sintácticos descendentes LL(k).

Generación de componentes

- Xtext construye automáticamente la infraestructura software necesaria para dar soporte a los nuevos lenguajes.
- Para configurar la generación automática del código de soporte, Xtext utiliza un DSL especial, denominado *MWE2*.
- MWE2 es un lenguaje declarativo para definir workflows formados por diferentes componentes que se encargarán de leer recursos EMF, transformarlos y generar artefactos de código.

Proceso generativo de desarrollo



DESARROLLO DE EDITORES TEXTUALES CON XTEXT

CARACTERÍSTICAS

Características de los DSL (I)

- Validation: para asegurar la corrección de los programas.
- Linking: declaración de enlaces dentro de la gramática.
- Scoping: definir el ámbito de aplicabilidad de elementos de la gramática.
- Quick Fixes: proporcionar soluciones rápidas para corregir un error o warning.

Características de los DSL (II)

- Formatting: manipular tokens (espacios en blanco, saltos de línea, comentarios, etc.)
- Syntax Coloring: utilización de diferentes colores o fuentes en diferentes elementos del fichero.
- Templates: incluir extractos de código de forma automática en el editor.
- Content Assist: auto-completado de código.

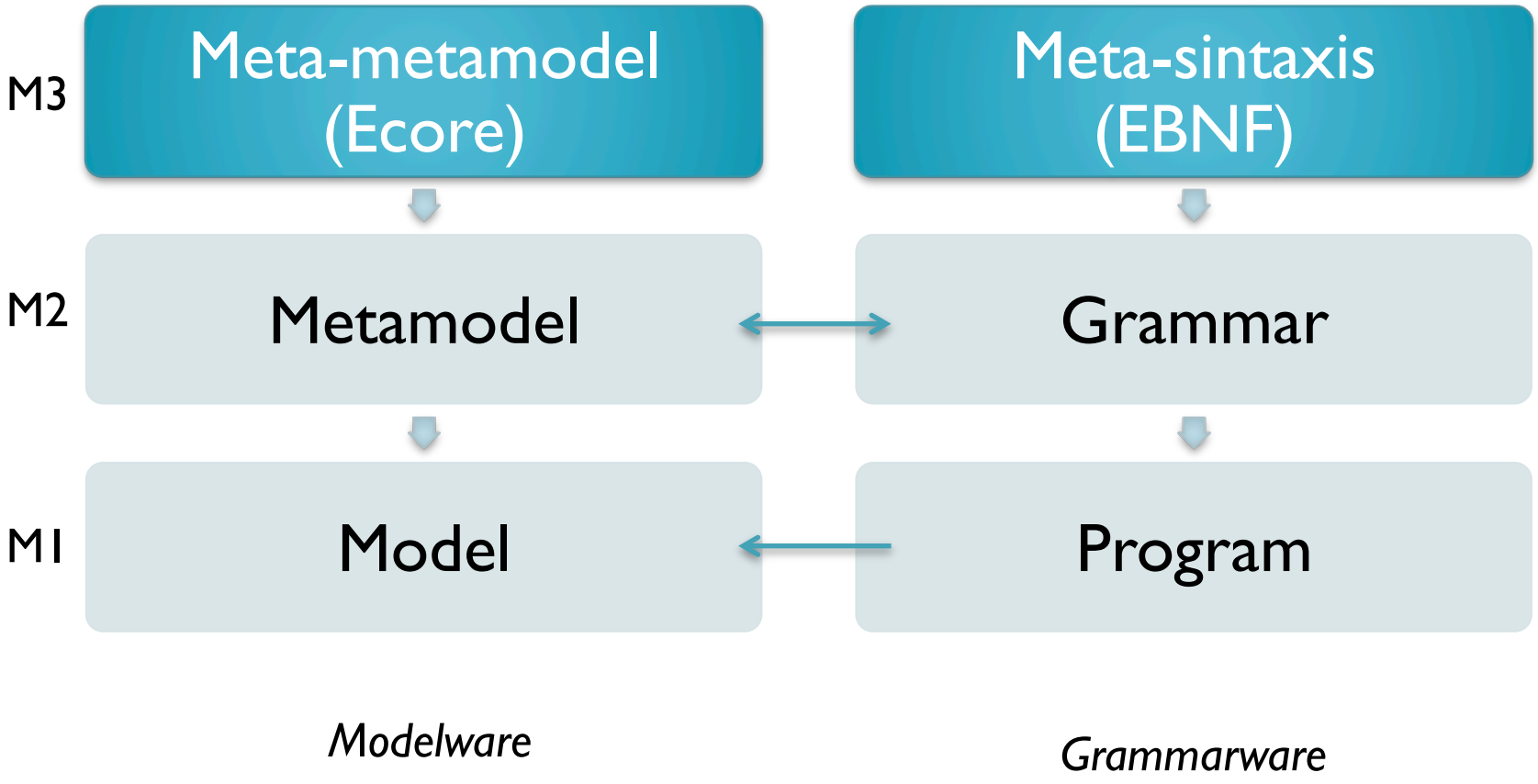
Características de los DSL (III)

- Outline View: vista jerárquica para navegar sobre nuestros modelos.
- Hyperlinking: mecanismo para navegar entre tokens del elementos del modelo (ctrl+click).
- Label Providers: personalizar la forma en el que los elementos del modelo son presentados en la interfaz gráfica.
- Name Refactoring: Renombrado seguro de elementos.

DESARROLLO DE EDITORES TEXTUALES CON XTEXT

EMF y XTEXT

EMF y Xtext


M3
Meta-metamodel (Ecore)
Meta-sintaxis (EBNF)
M2
Metamodel
Grammar
M1
Model
Program
Modelware
Grammarware

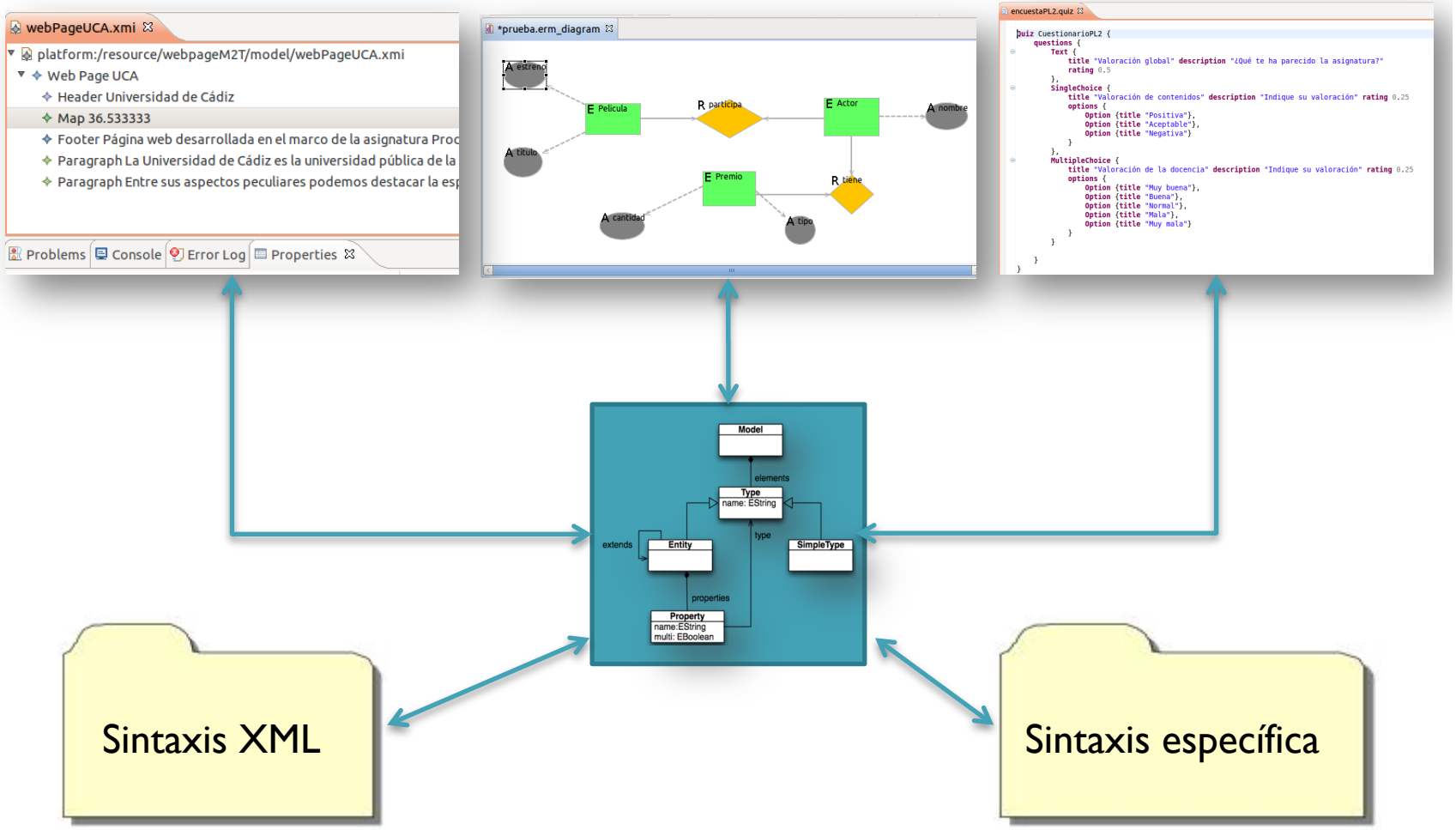
EMF y Xtext (II)

- Los parsers generados con Xtext analizan los ficheros de código fuente, generando una representación en memoria de los mismos.
- Los árboles de sintaxis abstracta (AST) son utilizados para representar la estructura sintáctica del fichero de texto.
- Los AST son modelos conformes al metamodelo de nuestro DSL. Así pues, el parser generado por Xtext se encarga de transformar el "texto" en un modelo (T2M).

EMF y Xtext (III)

- Los modelos desarrollados con los editores EMF o GMF son serializados en un formato XML.
- Los modelos desarrollados con los editores Xtext son serializados en un fichero de texto según una determinada gramática.
- La integración de EMF y Xtext permite que un mismo modelo pueda ser editado síncronamente con editores reflexivos (basados en árbol), visuales o textuales.

EMF y Xtext (IV)



DESARROLLO DE EDITORES TEXTUALES CON XTEXT

DISEÑO DE GRAMÁTICAS

Gramáticas en Xtext

- Xtext ofrece un lenguaje específico para diseñar gramáticas tipo EBNF.
- Las gramáticas se componen de un conjunto de reglas terminales y no terminales.
- Adicionalmente, se pueden definir predicados sintácticos, símbolos terminales ocultos y reglas para valores enumerados.

Declaración de la gramática

- Se define el nombre de la gramática y aquellas que queremos reutilizar. Indicaremos el metamodelo de nuestros AST que vayamos a utilizar o bien podemos generar (inferir) uno nuevo.

```
grammar grammar.qualified.file.name
    with other.grammar.qualified.file.name

import "input_metamodel_uri" as metamodel_name
generate metamodel_name 'metamodel_uri'
```

Expresiones EBNF (I)

- Caracteres: *'a'*
- Rango de caracteres: *'0'..'9'*
- Comodines: *'F'* **.** *'0'*
- Hasta… '/*' **->** '*/';
- Negación: ***!****'a'*
- Grupos: ***('0'..'9') ('A'..'F')***
- Alternativas: *rule1* **|** *rule2* **|** *rule3*

Expresiones EBNF (II)

- Cardinalidad:
  - Exactamente uno: por defecto
  - Cero o uno: 'a'**?**
  - Cero o más: *'a'******
  - Uno o más: *'a'***+**
- Grupos desordenados

  ('public' |'private' |'protected') ***&*** *'static'* ***&*** *'final'*
- Llamadas a reglas:

  terminal DOUBLE: ***INT*** *'.'* ***INT****;*
- Referencias cruzadas

  'Event:' name=ID 'from' ***[State]*** *'to'* ***[State]****;*

Expresiones EBNF (III)

- Operadores de asignación
  - Simple: permite asignar la información procesada a una característica simple del modelo.

    name **=** *ID*
  - Múltiple: permite añadir una lista de valores a una característica del modelo de tipo lista.

    options **+=** *OPTION*
  - Booleano: asigna true si hay algún valor y falso en caso contrario

    correct **?=** *'correct'*

Terminal Rules

- Estas reglas transforman secuencias de caracteres en tokens, cubriendo así el análisis léxico del texto. Pueden devolver un tipo EDataType (por defecto, ecore::EString) La gramática *org.eclipse.xtext.common.Terminals* incorpora una serie de patrones estándar, como ID.

```
terminal ID :
 ('^')?('a'..'z'|'A'..'Z'|'_') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')*;
```

Parser Rules

- Estas reglas, a diferencia de las anteriores, no produce un token atómico, sino que producen un árbol de tokens terminales y no terminales. Estas reglas pueden devolver un tipo EDataType.

```
BookRule returns booksMM::Book :
 'Book'  '{' 'title:' title=ID ','
             'author:' author=ID  '}' ;
```

DESARROLLO DE EDITORES TEXTUALES CON XTEXT

DESARROLLO DE UN EDITOR TEXTUAL

¿Qué vamos a hacer?

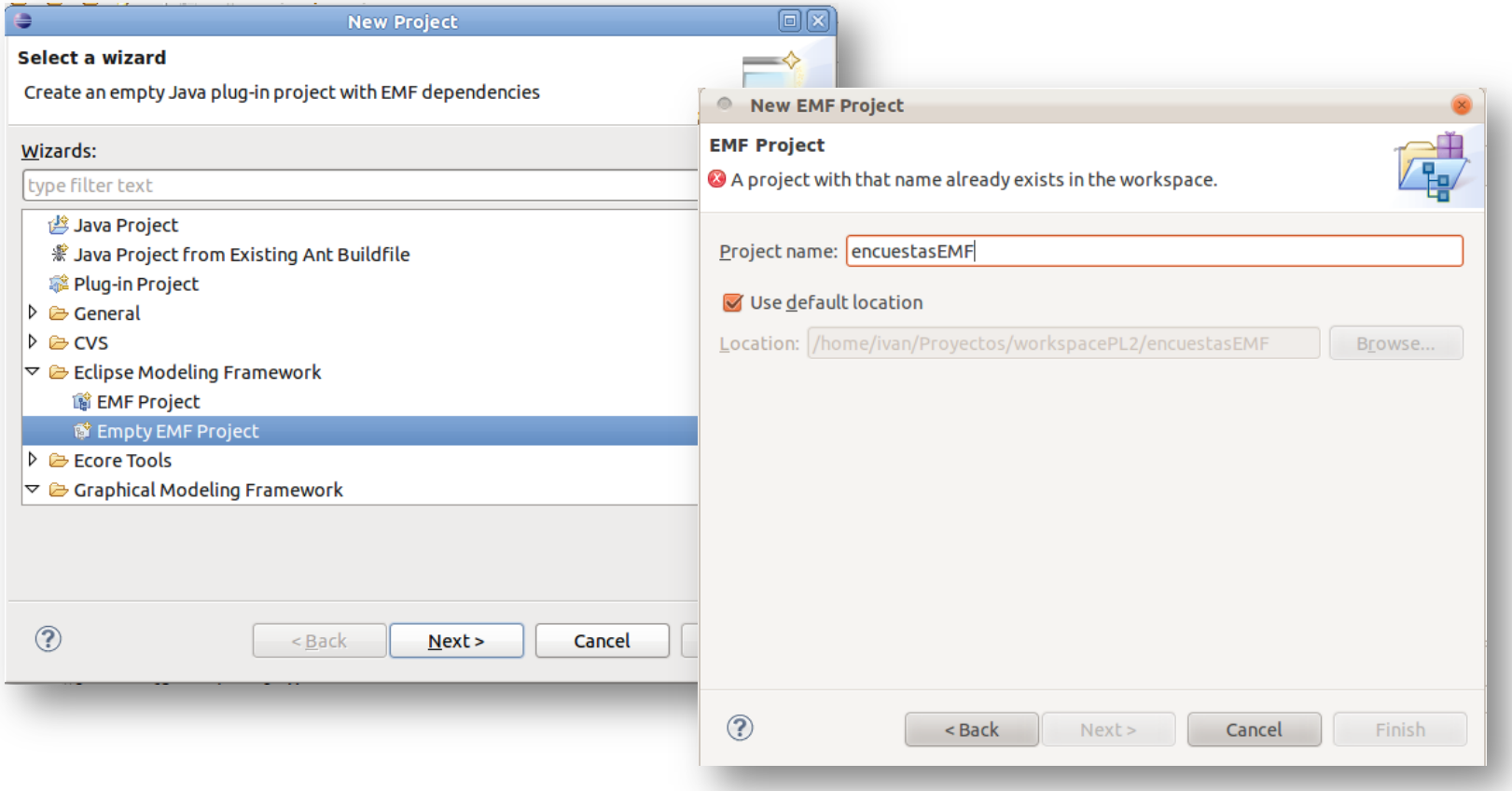
- Suponemos que muy frecuentemente tenemos que desarrollar formularios web para generar cuestionarios de evaluación.
- No queremos hacer uso una interfaz de administración, ni utilizar lenguajes visuales para su diseño.
- Solución: desarrollar un DSL textual para escribir cuestionarios utilizando una gramática sencilla.

Algo asi…

encuestaPL2.quiz

```
Quiz CuestionarioPL2 {
    questions {
        Text {
            title "Valoración global" description "¿Qué te ha parecido la asignatura?"
            rating 0.5
        },
        SingleChoice {
            title "Valoración de contenidos" description "Indique su valoración" rating 0.25
            options {
                Option {title "Positiva"},
                Option {title "Aceptable"},
                Option {title "Negativa"}
            }
        },
        MultipleChoice {
            title "Valoración de la docencia" description "Indique su valoración" rating 0.25
            options {
                Option {title "Muy buena"},
                Option {title "Buena"},
                Option {title "Normal"},
                Option {title "Mala"},
                Option {title "Muy mala"}
            }
        }

    }
}
```

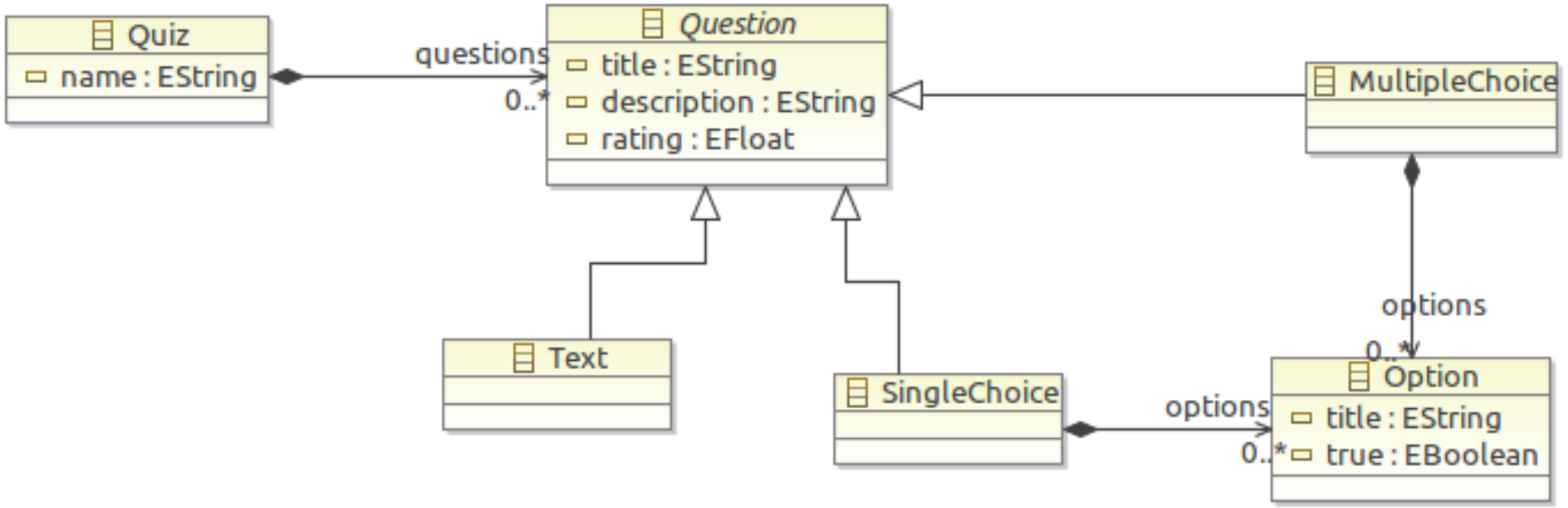
Creación de un proyecto EMF



`File → New Project → Empty EMF Project`

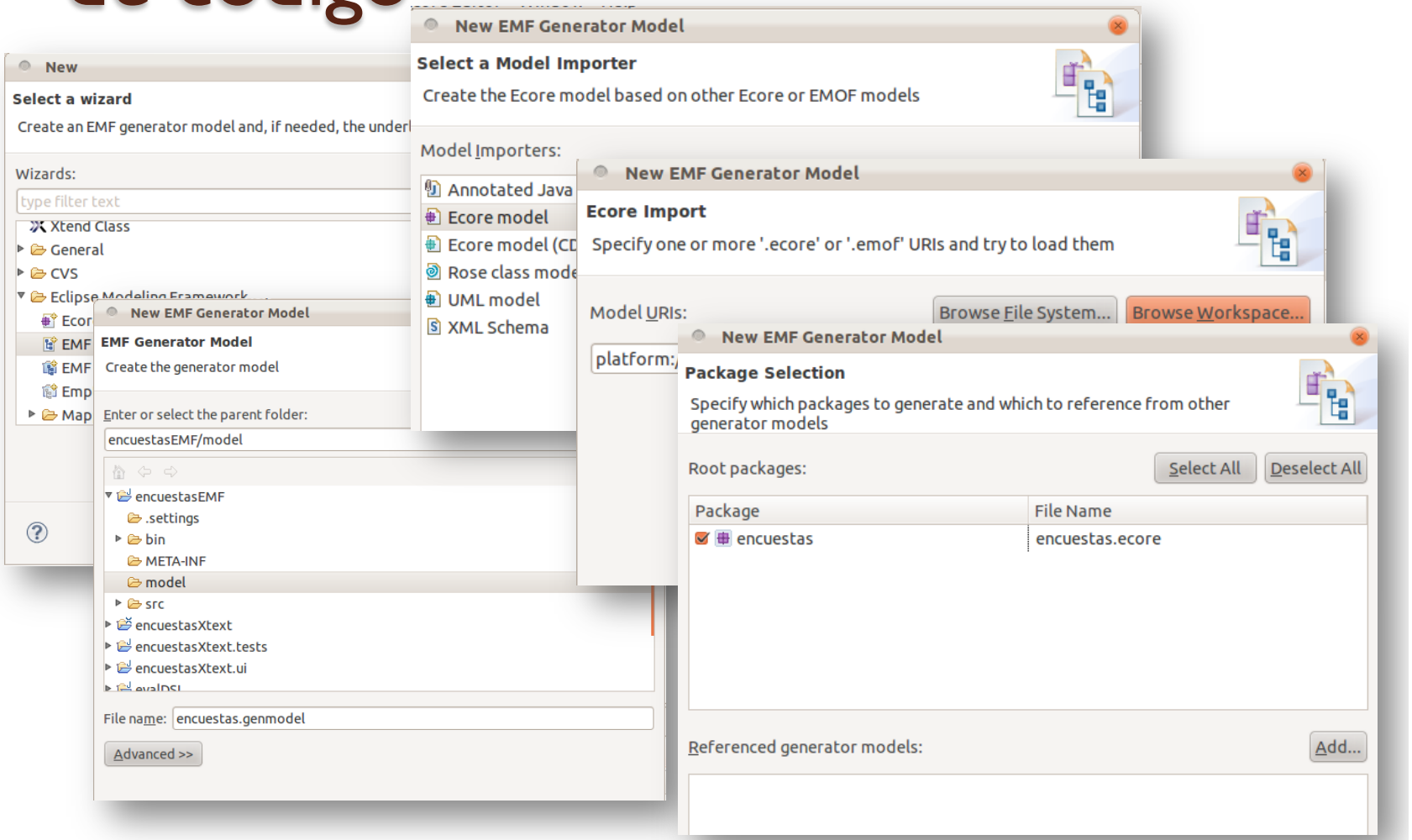
En primer lugar, crearemos un metamodelo del lenguaje dentro de un proyecto EMF

Diseño del metamodelo



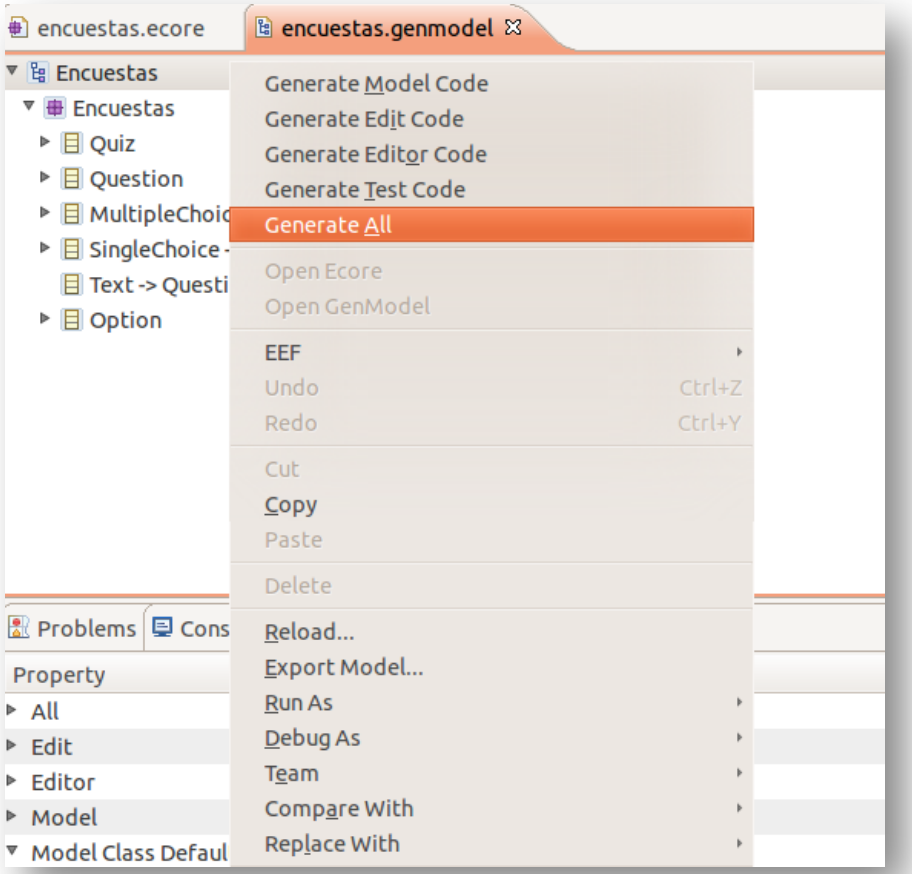
File → New → Ecore Diagram

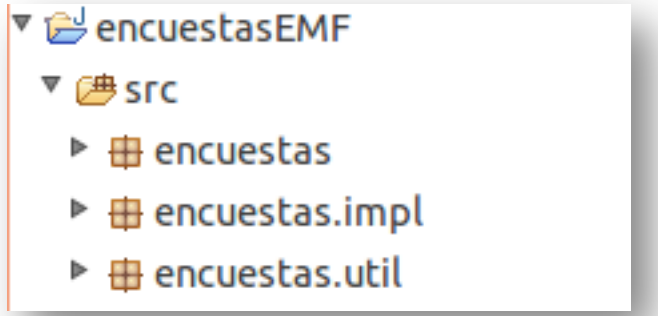
Creación del modelo de generación de código

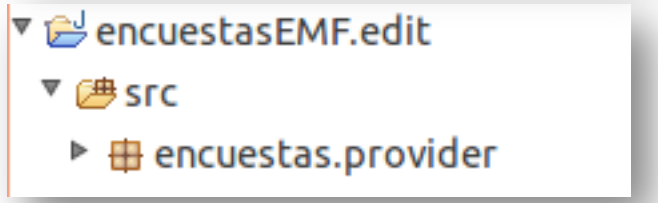


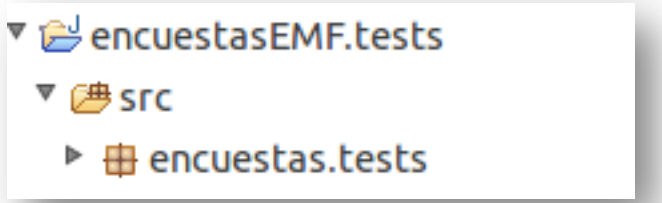
`File → New → EMF Generator Model`

Generación del código Java

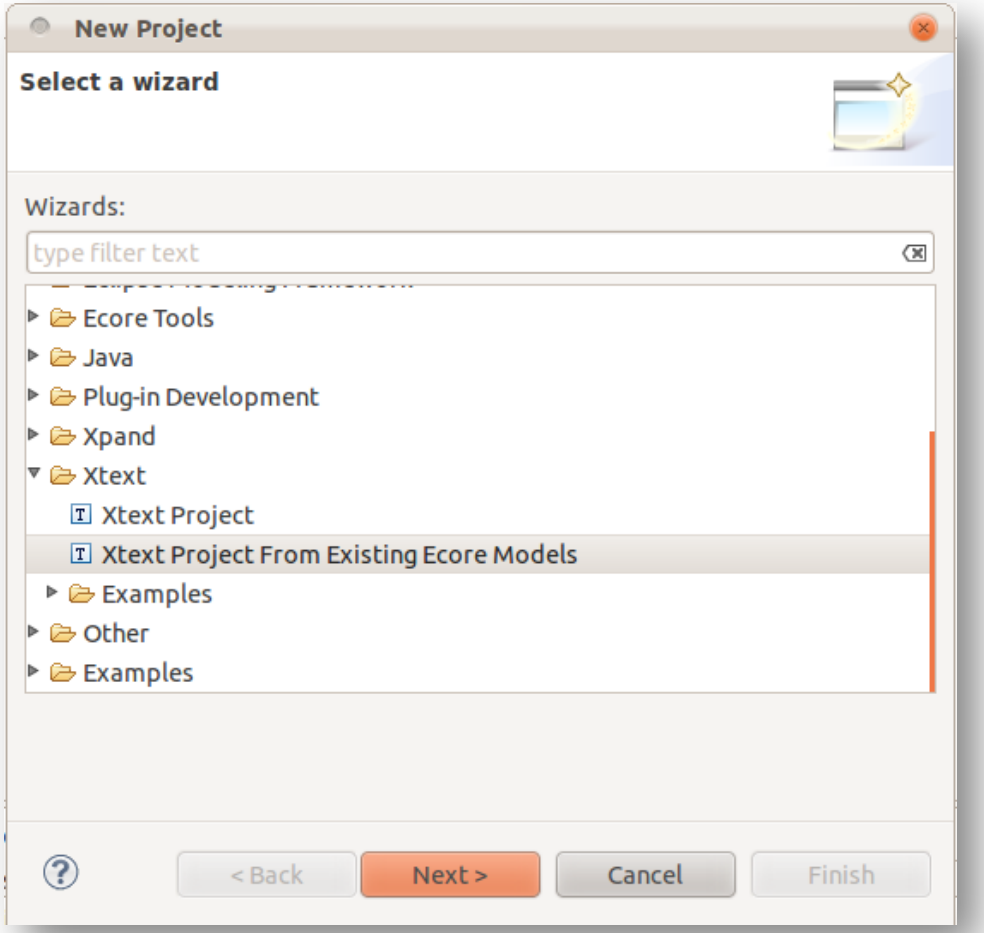






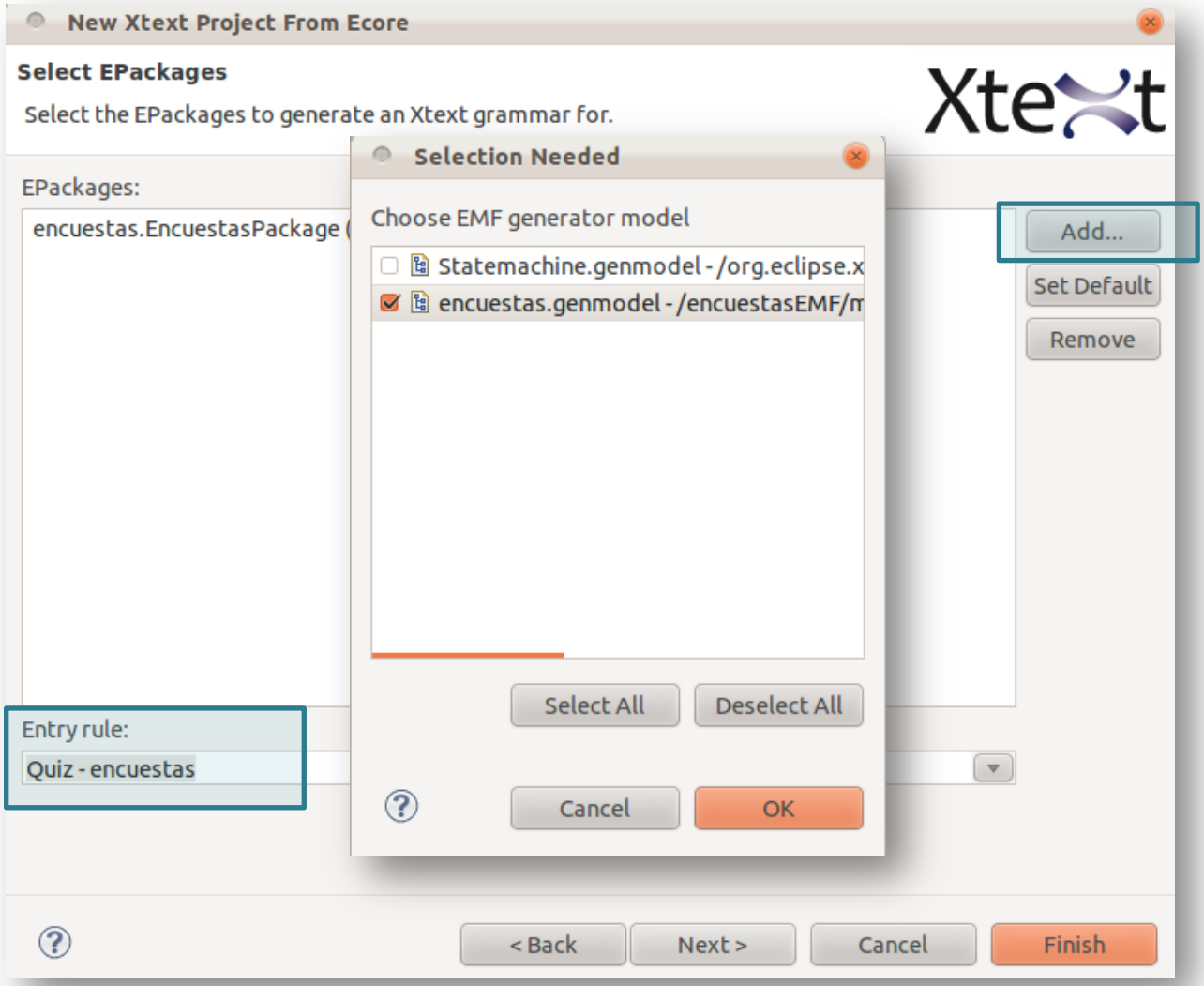


Creación de un proyecto Xtext (I)

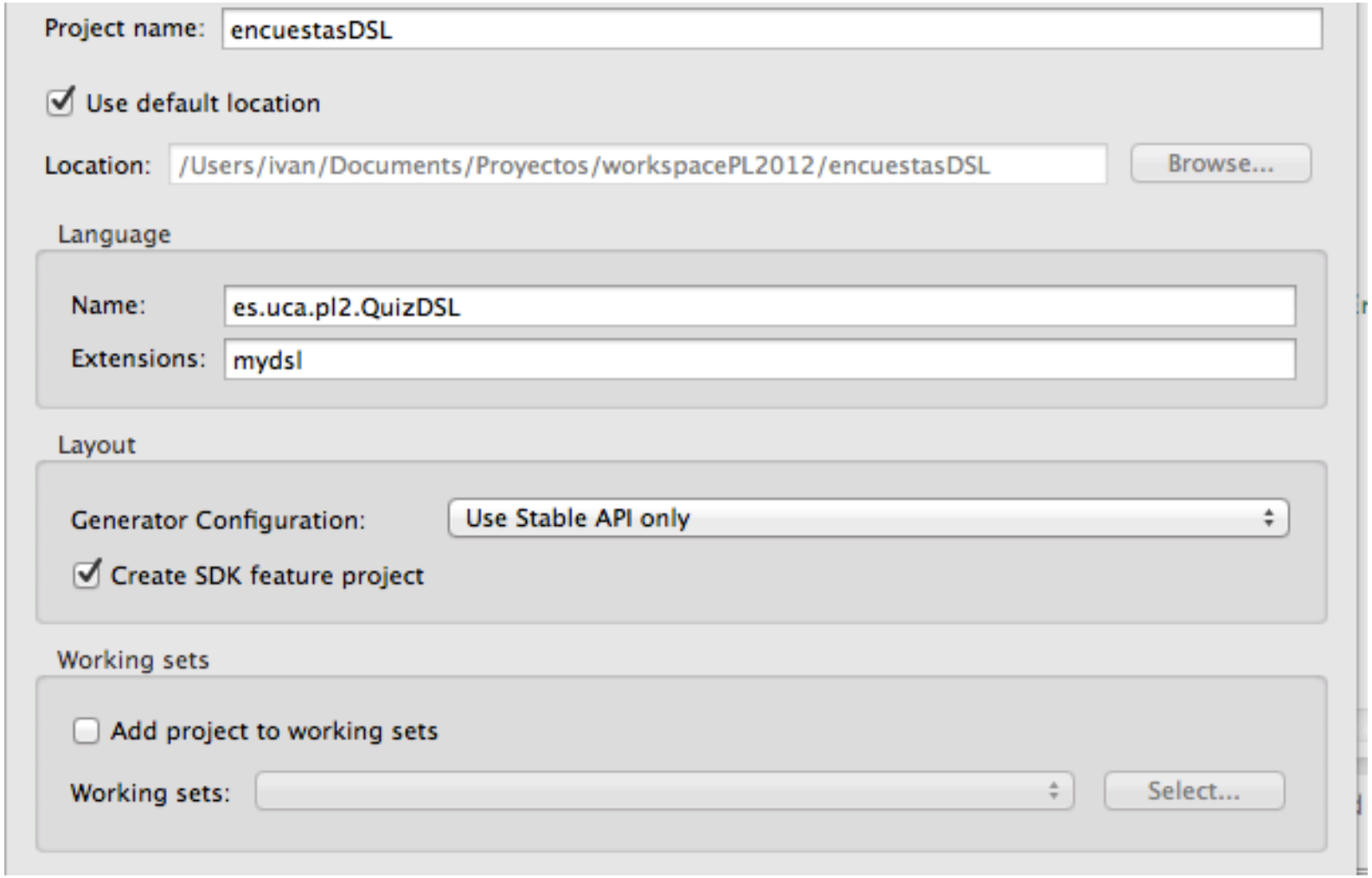


File → New Project → Xtext Project From Existing Ecore Models

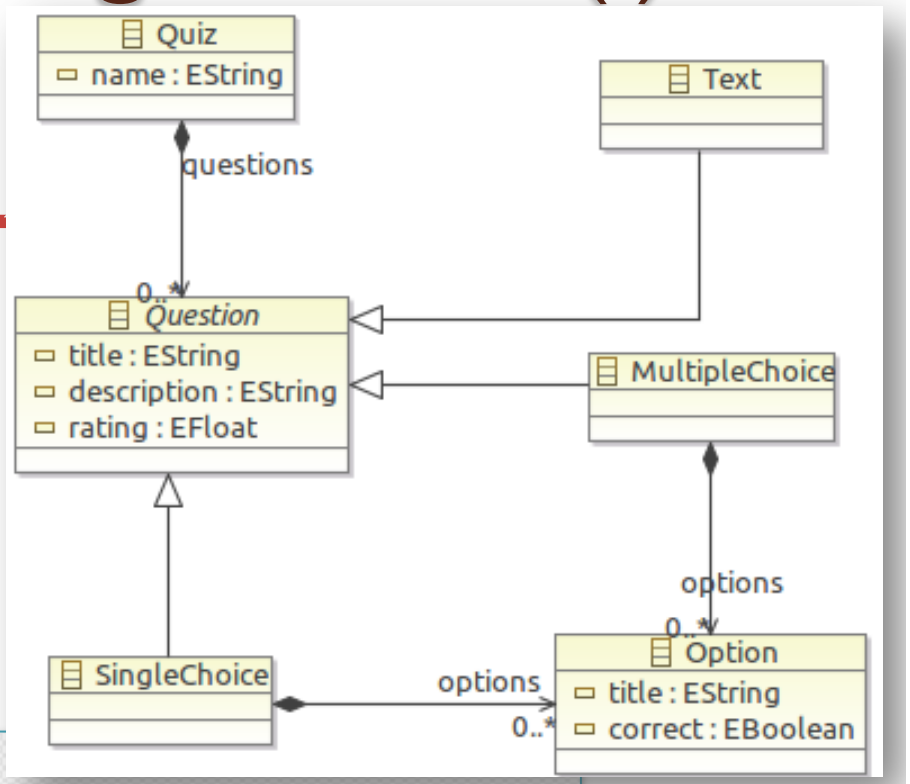
Creación de un proyecto Xtext (II)



Creación de un proyecto Xtext (III)



Derivación de la gramática (I)

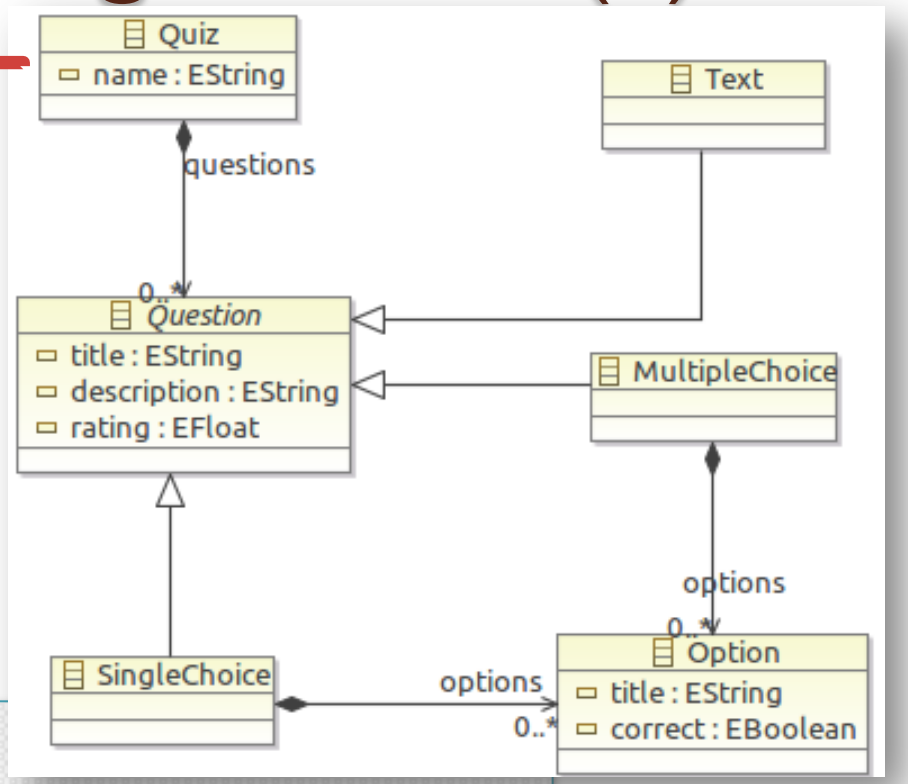


De un **paquete** del modelo a una nueva **gramática**

```
grammar es.uca.pl2.QuizDSL with
        org.eclipse.xtext.common.Terminals

import "platform:/resource/encuestasEMF/model/encuestas.ecore“
import "http://www.eclipse.org/emf/2002/Ecore" as ecore
```

Derivación de la gramática (II)

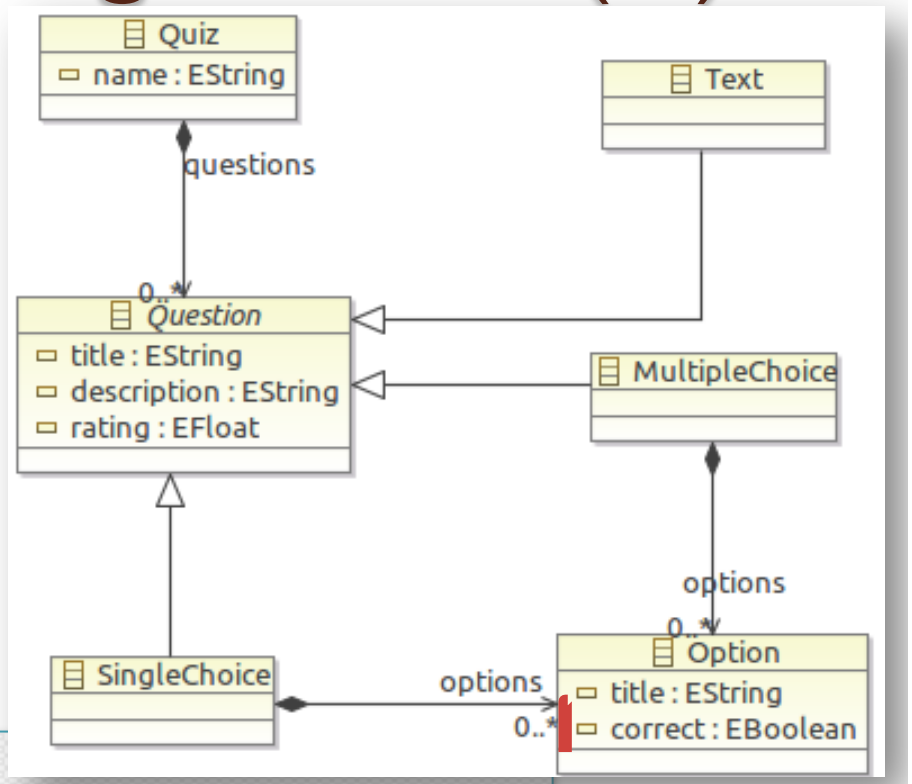
De **clases** del modelo a **reglas** de la gramática




```
Quiz returns Quiz:
    'Quiz'
    '{'
    ...
    '}'
    ;
```

Derivación de la gramática (III)

De **atributos** de clase a **asignaciones simples** en una regla



Option returns Option:

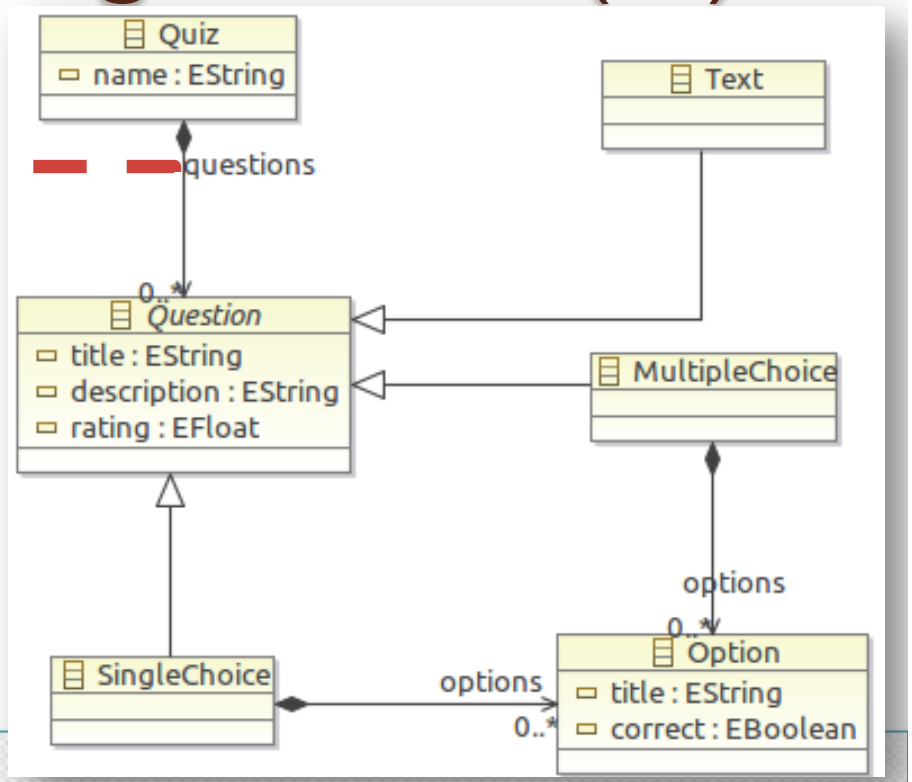
…

('title' title=EString)?

…

;

Derivación de la gramática (IV)



De **composición a asignaciones múltiples** en una regla

Quiz returns Quiz:

…

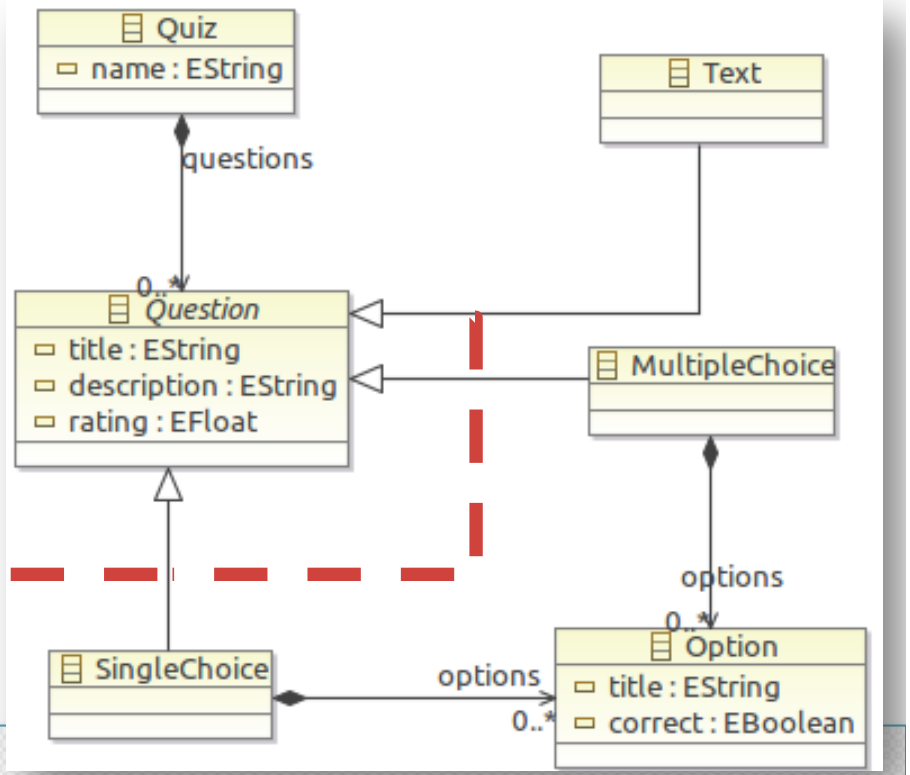
('questions' '{' questions+=Question ("," questions+=Question)* '}')?

…

;

Derivación de la gramática (V)

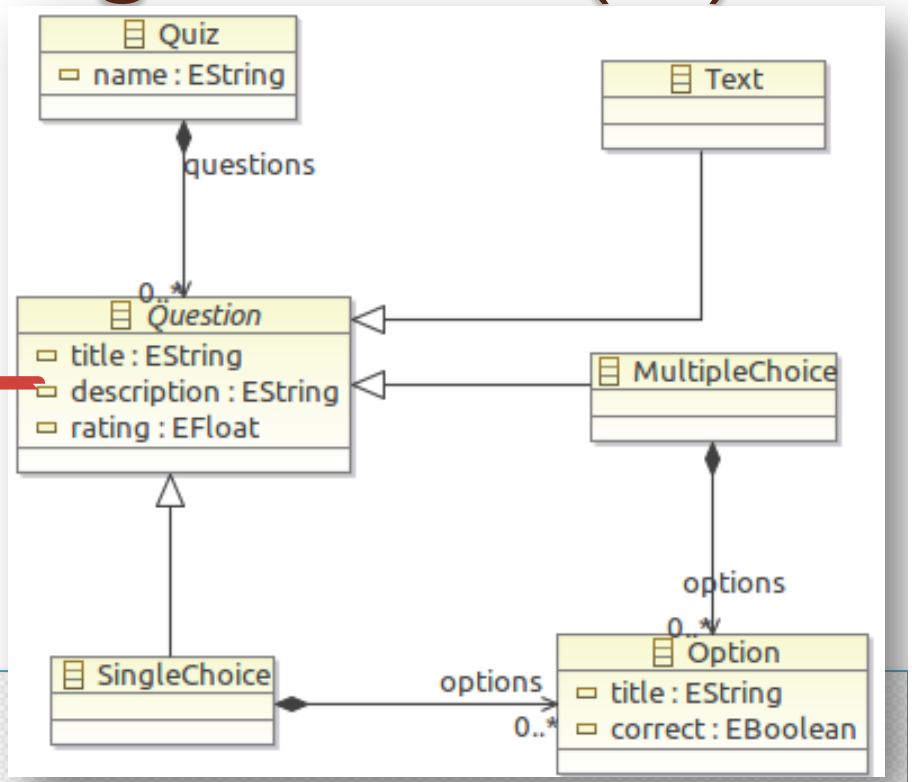
De una **herencia** de clases a una regla con **alternativas**



Question returns Question:
MultipleChoice | SingleChoice | Text;

Derivación de la gramática (VI)

Des-factorización de **atributos** de clase base sobre **asignaciones**




```
Text returns Text:
        'Text'
        '{'
                ('title' title=EString)?
                ('description' description=EString)?
                ('rating' rating=EFloat)?
        '}';
```

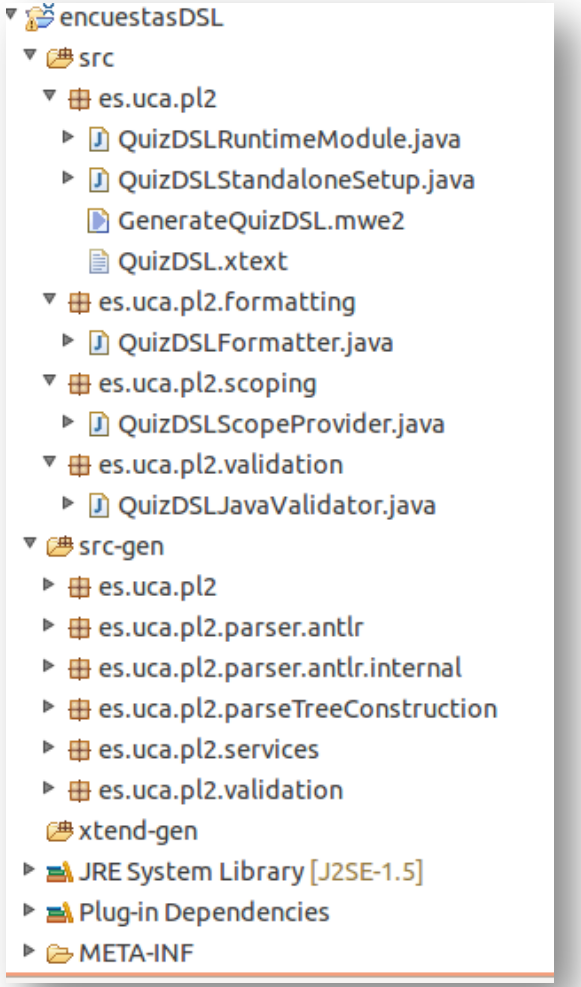
Generar artefactos del editor

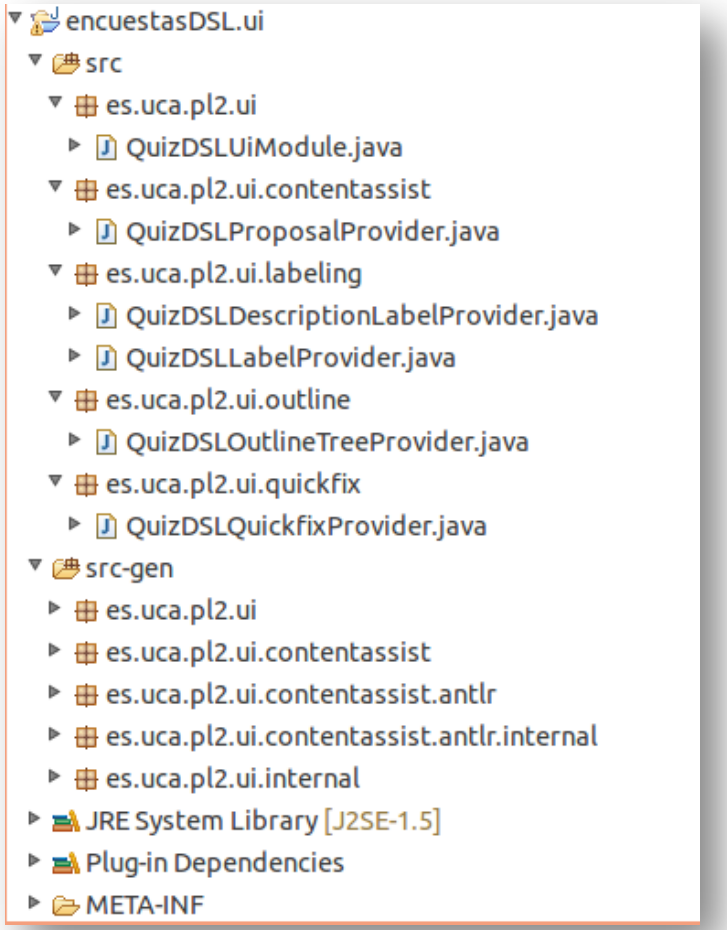
GenerateQuizDSL.mwe2

```
module es.uca.pl2.QuizDSL

import org.eclipse.emf.mwe.utils.*
import org.eclipse.xtext.generator.*
import org.eclipse.xtext.ui.generator.*

var grammarURI = "classpath:/es/uca/pl2/QuizDSL.xtext"
var file.extensions = "quiz"
var projectName = "encuestasDSL"
var runtimeProject = "../${projectName}"

Workflow {
    bean = StandaloneSetup {
        scanClassPath  = true
        platformUri = "${runtimeProject}/.."
        registerGeneratedEPackage = "encuestas.EncuestasPackage"

        // registerGenModelFile = "platform:/resource/encuestasEMF/model/encuestas.genmodel"

    }

    component = DirectoryCleaner {
        directory = "${runtimeProject}/src-gen"
    }

    component = DirectoryCleaner {
        directory = "${runtimeProject}.ui/src-gen"
    }

    component = Generator {
        pathRtProject = runtimeProject
        pathUiProject = "${runtimeProject}.ui"
```

[File *.mwe2] Run As → MWE2 Workflow

Estructura de código del editor





Desarrollo de una validación

QuizDSLJavaValidator.java

```
package es.uca.pl2.validation;

import org.eclipse.xtext.validation.Check;

import encuestas.EncuestasPackage;
import encuestas.Question;
import encuestas.Quiz;


public class QuizDSLJavaValidator extends AbstractQuizDSLJavaValidator {


    @Check
    public void checkTotalRating(Quiz quiz) {

        float count=0;

        for(Question question:quiz.getQuestions()){
            count += question.getRating();
        }

        if(count>1){
             error("La puntuación máxima es de 1 punto",EncuestasPackage.Literals.QUIZ__NAME);
             return;
        }

    }
}
```

Test del editor textual

encuestaPL2.quiz

```
Quiz CuestionarioPL2 {
    questions {
        Text {
            title "Valoración global" description "¿Qué te ha parecido la asignatura?"
            rating 0.5
        },
        SingleChoice {
            title "Valoración de contenidos" description "Indique su valoración" rating 0.25
            options {
                Option {title "Positiva"},
                Option {title "Aceptable"},
                Option {title "Negativa"}
            }
        },
        MultipleChoice {
            title "Valoración de la docencia" description "Indique su valoración" rating 0.25
            options {
                Option {title "Muy buena"},
                Option {title "Buena"},
                Option {title "Normal"},
                Option {title "Mala"},
                Option {title "Muy mala"}
            }
        }

    }
}
```

[Project *UI*] Run as → Eclipse Application

DESARROLLO DE EDITORES TEXTUALES CON XTEXT

RESUMEN

¿Qué hemos aprendido hoy?

- Desarrollar DSLs textuales con Xtext.
- Xtext genera automáticamente los componentes necesarios para usar nuestros DSLs dentro de Eclipse.
- Sólo necesitamos diseñar una gramática e implementar algún comportamiento adicional para nuestro editor.
- La gramática BNF puede ser derivada a partir de un metamodelo Ecore.

Procesadores de Lenguajes 2

Desarrollo de editores textuales con Xtext

Iván Ruiz Rube
ivan.ruiz@uca.es