

Procesadores de Lenguajes 2

# Transformaciones de modelo a modelo con ATL

Curso 2013-2014

Iván Ruiz Rube

Departamento de Ingeniería Informática

Escuela Superior de Ingeniería

Universidad de Cádiz



# Contenidos

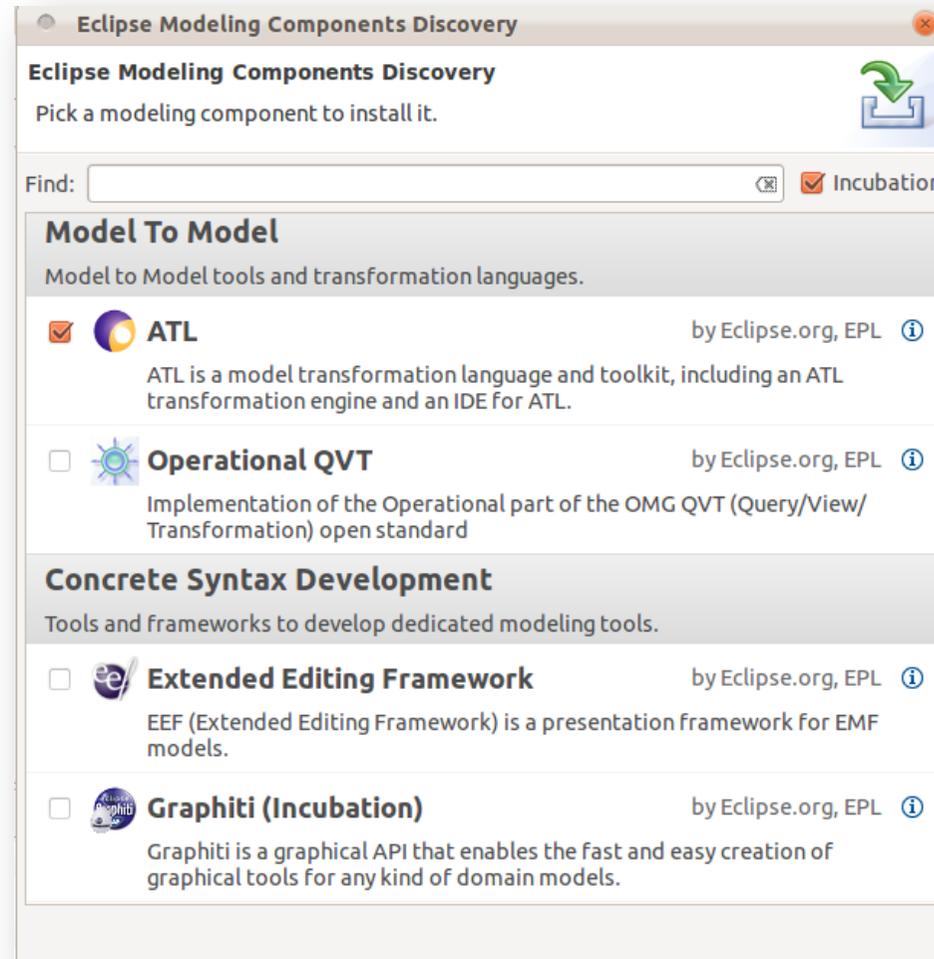
- Instalación
- Introducción
- Componentes
- Características del lenguaje
- Desarrollo de una transformación

TRANSFORMACIONES DE MODELO A MODELO CON  
ATL



# INSTALACIÓN

# Instalación ATL



Help → Install Modeling Components

TRANSFORMACIONES DE MODELO A MODELO CON  
ATL



# INTRODUCCIÓN

# Atlas Transformation Language

- Lenguaje y un motor de transformación (*máquina virtual*) para M2M.
- Desarrollado por el grupo ATLAS INRIA & LINA como una implementación de la propuesta QVT de la OMG.
- El lenguaje ATL incorpora tanto un metamodelo como una sintaxis textual para definir las reglas de transformación.

# Lenguaje ATL

- El lenguaje de ATL es declarativo e imperativo y está basado en OCL.
- De forma declarativa se pueden expresar los mappings entre los elementos origen y destino.
- En ciertas ocasiones es difícil de declarar ese mapping, por lo que se pueden utilizar ciertas construcciones imperativas para lograrlo.

# Modos de ejecución

- **Modo Normal**
  - Se debe especificar completamente la forma en que los elementos del modelo destino serán generados.
- **Modo Refinamiento**
  - Sólo se deben especificar las modificaciones a llevar a cabo entre el modelo origen y el destino.
  - El resto de elementos son copiados automáticamente (transformaciones endógenas).

# Eclipse ATL Zoo

The screenshot shows a web browser window with the title "ATL Transformations" and the URL "www.eclipse.org/m2m/atl/atlTransformations/". The page content is as follows:

**Modeling >>**

**M2M >>**

**About This Project**

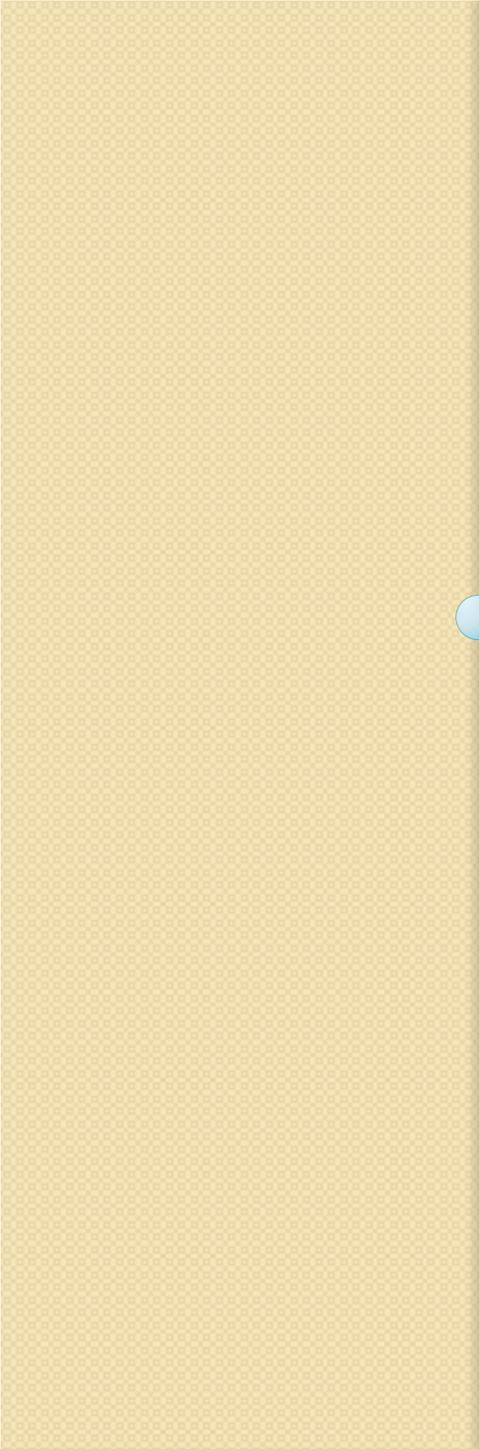
- Components
- Infrastructure
- ATL
- Operational QVT
- Declarative QVT

**ATL Transformations**

There are currently 103 model transformations scenarios in this zoo.

**ATL Transformations list**

- Ant to Maven: Documentation, Source files
- AssertionModification: Documentation, Source files
- ATL to BindingDebugger: Documentation, Source files
- ATL to Problem: Documentation, Source files
- ATL to Tracer: Documentation, Source files
- BibTeXXML to DocBook: Documentation, Source files
- Book to Publication: Documentation, Source files
- CatalogueModelTransformations: Documentation, Source files
- Class to Relational: Documentation, Source files
- Code Clone Tools to SVG: Documentation, Source files
- CPL to SPL: Documentation, Source files
- Disaggregation: Documentation, Source files
- DSL to EMF: Documentation, Source files
- EliminateRedundantInheritance: Documentation, Source files
- EMF2KM3 (Ecore to KM3): Documentation, Source files
- EquivalenceAttributesAssociations: Documentation, Source files
- Families to Persons: Documentation, Source files
- Feature Models for representing runtime variability in BIS to Business Process Model Notation: Documentation, Source files
- Geometrical Transformations: Documentation, Source files



TRANSFORMACIONES DE MODELO A MODELO CON  
ATL



# COMPONENTES

# Componentes ATL

- *Module*: implementar transformaciones entre modelos. Se compone de *Headers*, *Helpers* y *Rules*.
- *Library*: construcción de bibliotecas reutilizables de código ATL.
- *Query*: especificar consultas sobre modelos.

# ATL Module: Headers

- Definen el nombre del módulo de transformación, los metamodelos origen y destino, modo de ejecución (normal o refinamiento) y opcionalmente la utilización de librerías ATL externas.

```
module module_name;  
create output_models [from|refining] input_models;  
uses extension_library_file_name;  
...
```

# ATL Module: Helpers

- Mecanismo para factorizar expresiones ATL, similar a los métodos en Java. Los helpers se definen mediante un nombre y un contexto de aplicación. Pueden incluir parámetros y tienen un tipo de retorno.

```
Helper [context context_type] def: helper_name(parameters) :  
    return_type=exp;
```

# ATL Module: Rules

- Definen cómo se transforman los elementos del modelo de entrada en los correspondientes elementos del modelo de salida. Permiten incluir condiciones, variables, asignaciones y otras sentencias.

```
rule rule_name {  
  from InputMetamodel!ModelElement(condition)  
  using (vars)  
  to OutputMetamodel!ModelElement(bindings)  
  do (statements)  
}
```

# ATL Module: Rules (II)

- Matched rules: reglas declarativas que se activan automáticamente mediante el matching de patrones [*from*].
- Lazy rules: igual que las anteriores, pero son invocadas desde otras reglas.
- Called rules: en lugar de tener matching con elementos del modelo origen, reciben parámetros.



# ATL Library

- Las bibliotecas ATL permiten definir un conjunto de helpers que son invocados desde otras bibliotecas, módulos o consultas.
- No pueden ejecutarse de forma independiente.
- Los helpers definidos dentro de una biblioteca ATL deben estar explícitamente asociados a un contexto determinado.

# ATL Query

- Las consultas ATL permiten obtener un valor de tipo primitivo (Boolean, String, Real, Integer) desde un modelo.
- Se suelen utilizar para generar una salida textual (valor de tipo String) a partir de elementos del modelo.

```
query query_name = exp;
```

TRANSFORMACIONES DE MODELO A MODELO CON  
ATL



# CARACTERÍSTICAS DEL LENGUAJE

# Expresiones ATL

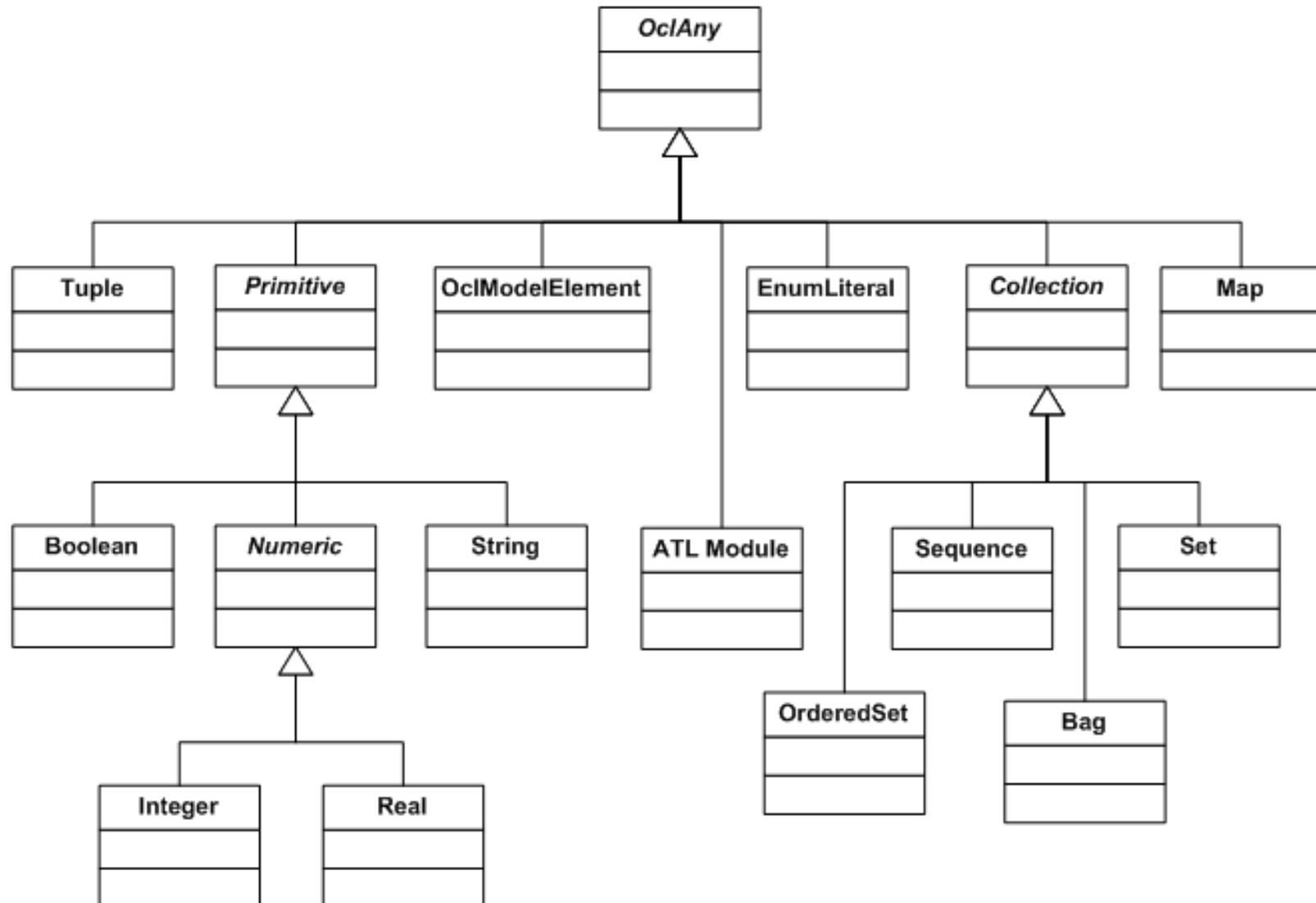
```
let var_name : var_type = var_init_exp in exp
```

```
using {  
    var_name : var_type = var_init_exp;  
}
```

```
if (condition)  
then exp1  
else exp2  
endif
```

```
for(iterator in collection) {  
    statements  
}
```

# Tipos de datos ATL



# Operaciones ATL

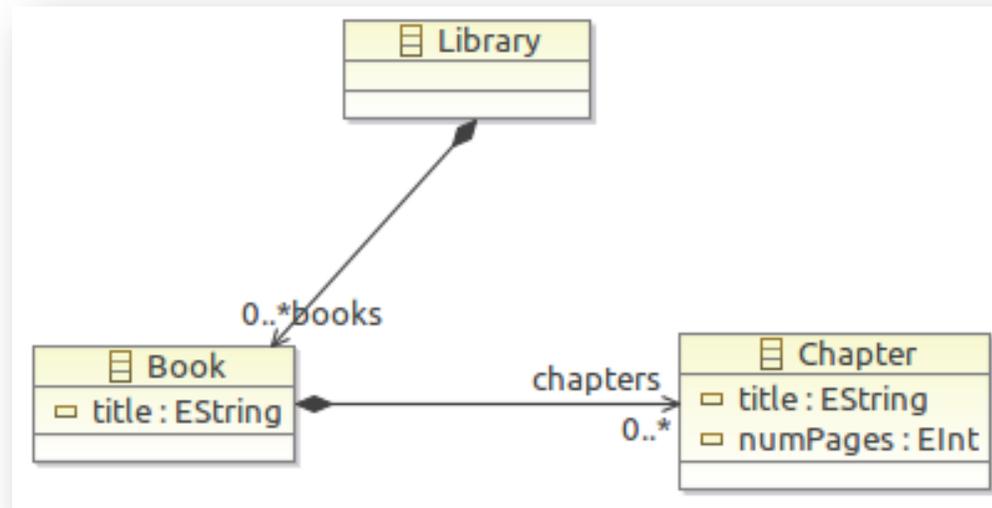
- **OclAny:**
  - *toString()*, *oclType()*, *output()*
- **Primitive:**
  - Boolean: *and*, *or*, *xor*, *not*
  - Numeric: *abs()*, *floor()*, *round()*, *cos()*
  - String: *concat()*, *substring()*, *startsWith()*
- **Collection:**
  - Comunes: *size()*, *count()*, *sum()*
  - Iteradores: *forAll()*, *collect()*, *select()*
  - Sequence: *first()*, *last()*, *insertAt()*
  - Bag: *including()*, *excluding()*, *flatten()*
  - Set: *union()*, *intersection()*, *excluding()*

TRANSFORMACIONES DE MODELO A MODELO CON  
ATL



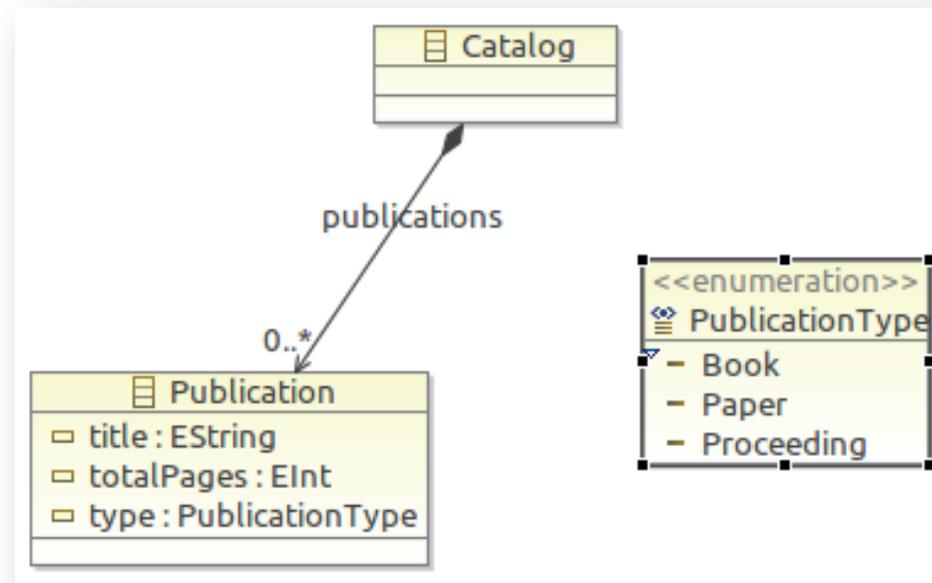
# DESARROLLO DE UNA TRANSFORMACIÓN

# Ejemplo



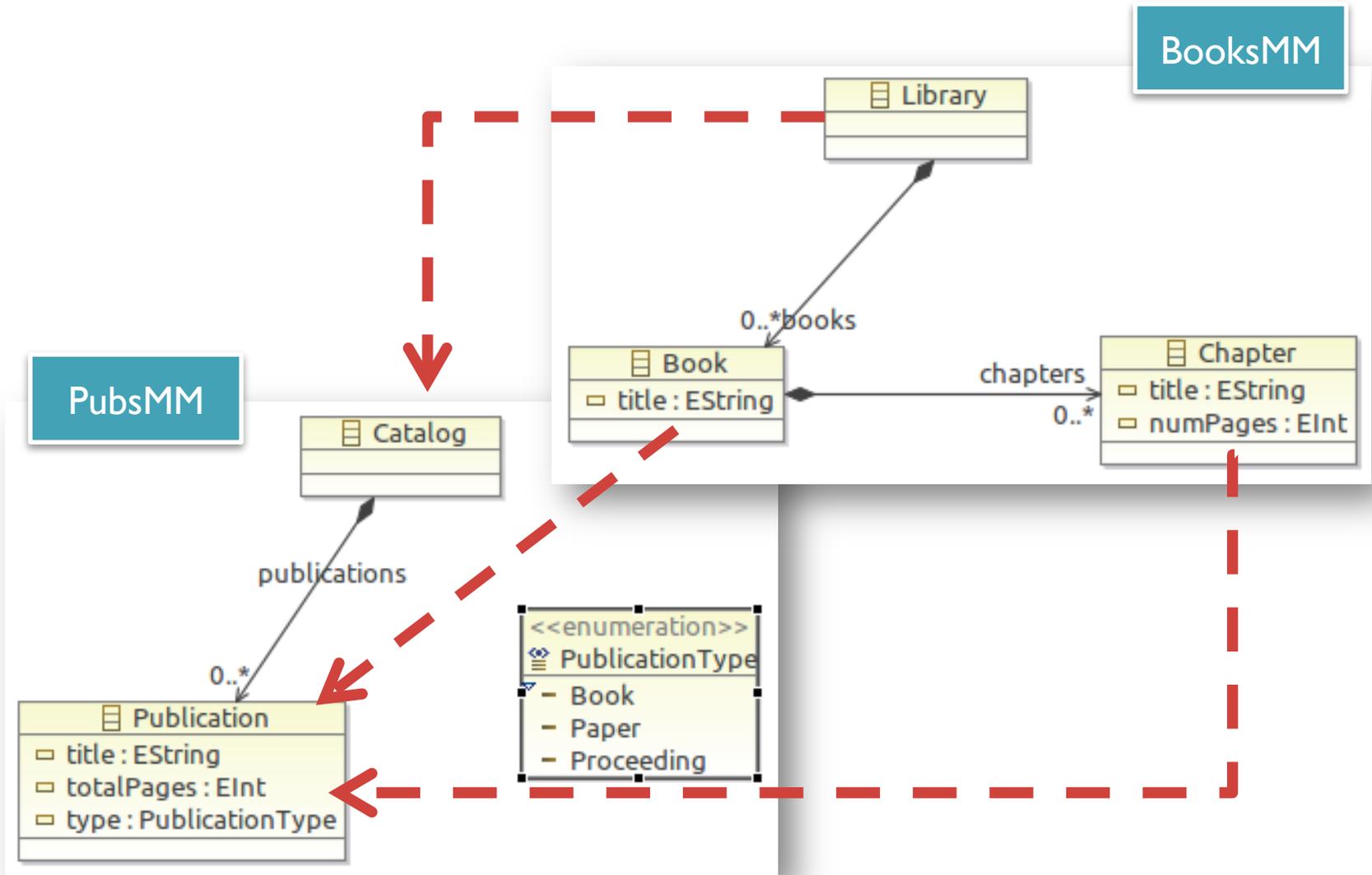
Metamodelo de libros

# Ejemplo

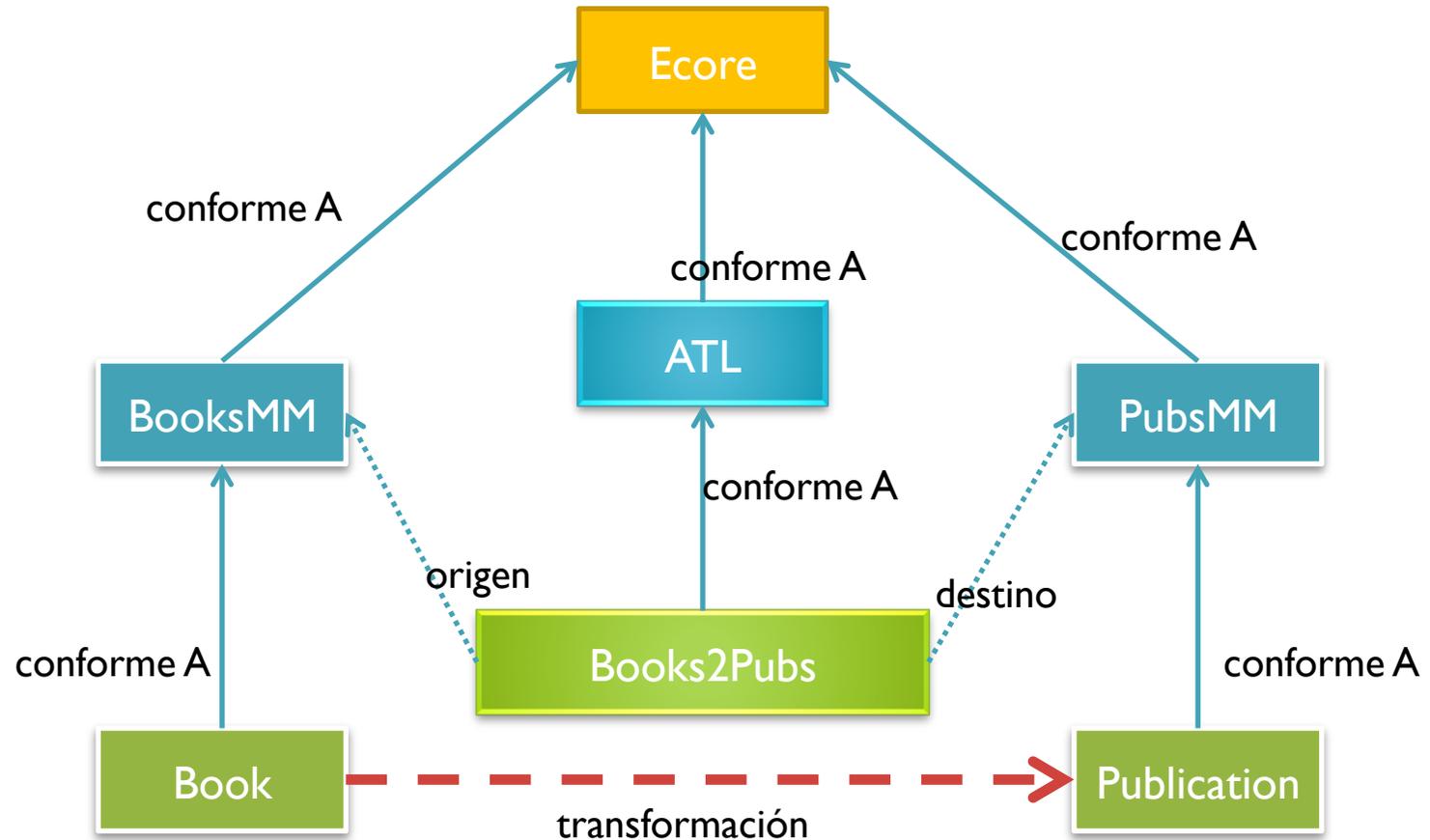


Metamodelo de publicaciones

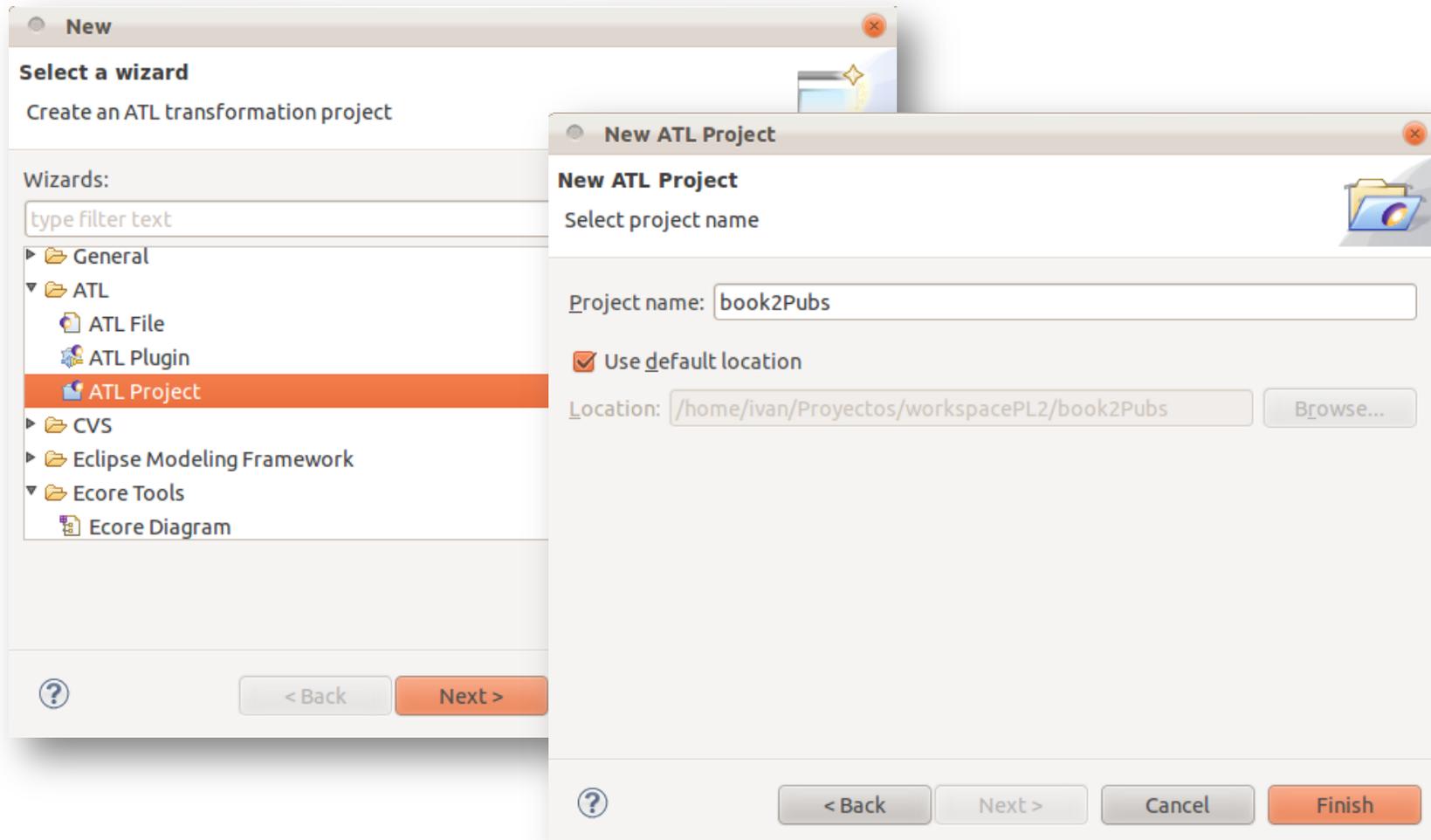
# Metamodelos y correspondencias



# Arquitectura del ejemplo

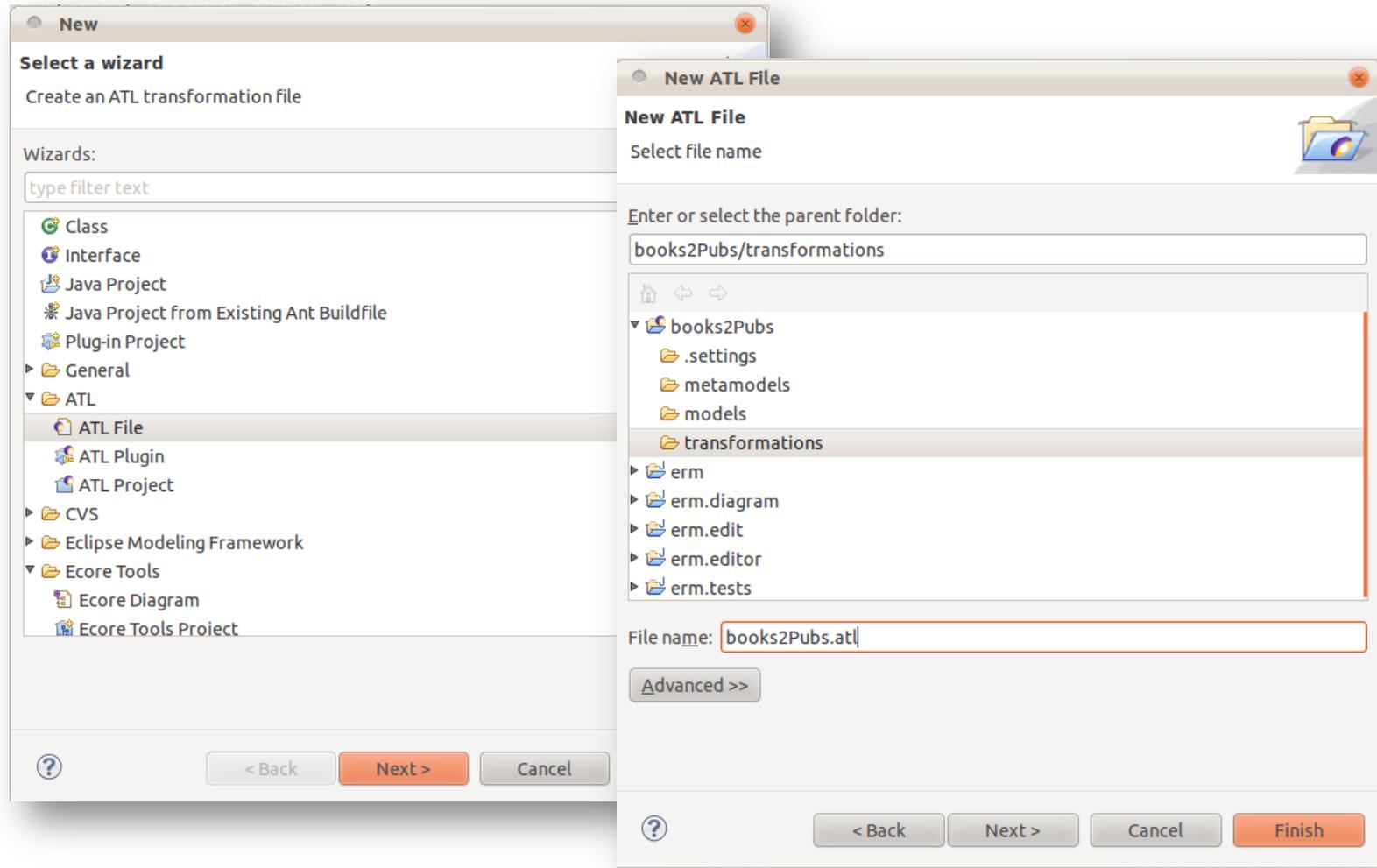


# Creación de un proyecto ATL



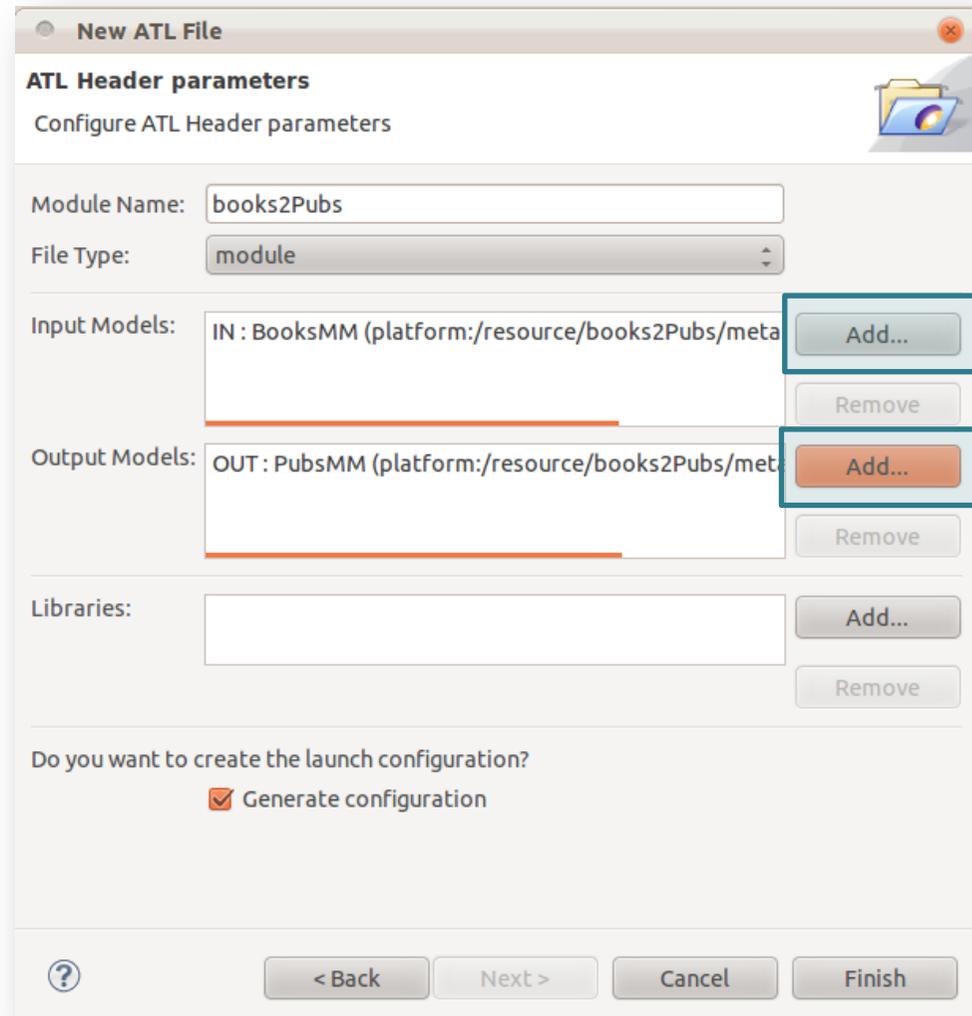
File → New → ATL Project

# Crear una transformación ATL

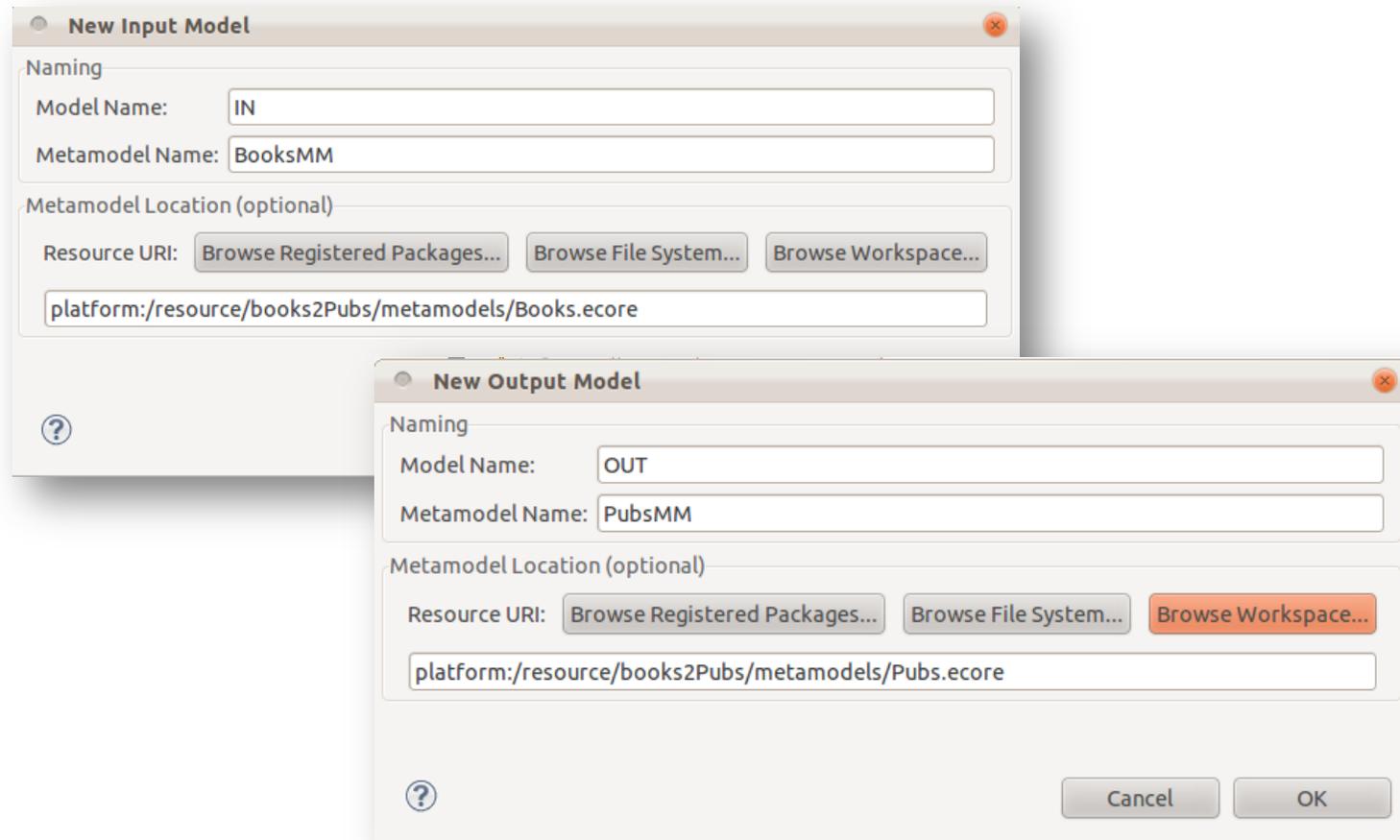


File → New → ATL File

# Headers de la transformación ATL



# Headers de la transformación ATL (II)



Tenemos que indicar los metamodelos origen y destino, así como los nombres para referirnos tanto a los modelos como a los metamodelos en el código ATL.

# Desarrollo de la transformación

- Nuestro fichero ATL definirá un módulo de transformación, el cual transformará un modelo *IN* de libros en un modelo *OUT* de publicaciones.

```
-- @path BooksMM=/books2Pubs/metamodels/Books.ecore  
-- @path PubsMM=/books2Pubs/metamodels/Pubs.ecore  
  
module books2Pubs;  
create OUT : PubsMM from IN : BooksMM;
```

# Desarrollo de la transformación (II)

- Transformación del elemento *Library* a un elemento *Catalog*.

```
rule Library2Catalog {  
  from  
    lib : BooksMM!Library  
  to  
    cat : PubsMM!Catalog (  
      publications <- lib.books  
    )  
}
```

# Desarrollo de la transformación (III)

- Transformación de los elementos *Book* a elementos *Publication*.

```
rule Book2Publication {  
  from  
    book : BooksMM!Book  
  to  
    pub : PubsMM!Publication (  
      title <- book.title,  
      type <- #Book,  
      totalPages <- book.countPages()  
    )  
}
```

## Desarrollo de la transformación (IV)

- Método auxiliar (helper) para calcular el total de páginas de un libro, a partir de las páginas de cada capítulo.

```
helper context BooksMM!Book def : countPages() : Integer =  
  self.chapters->collect(b|b.numPages).sum()  
;
```

# Desarrollo de un modelo origen

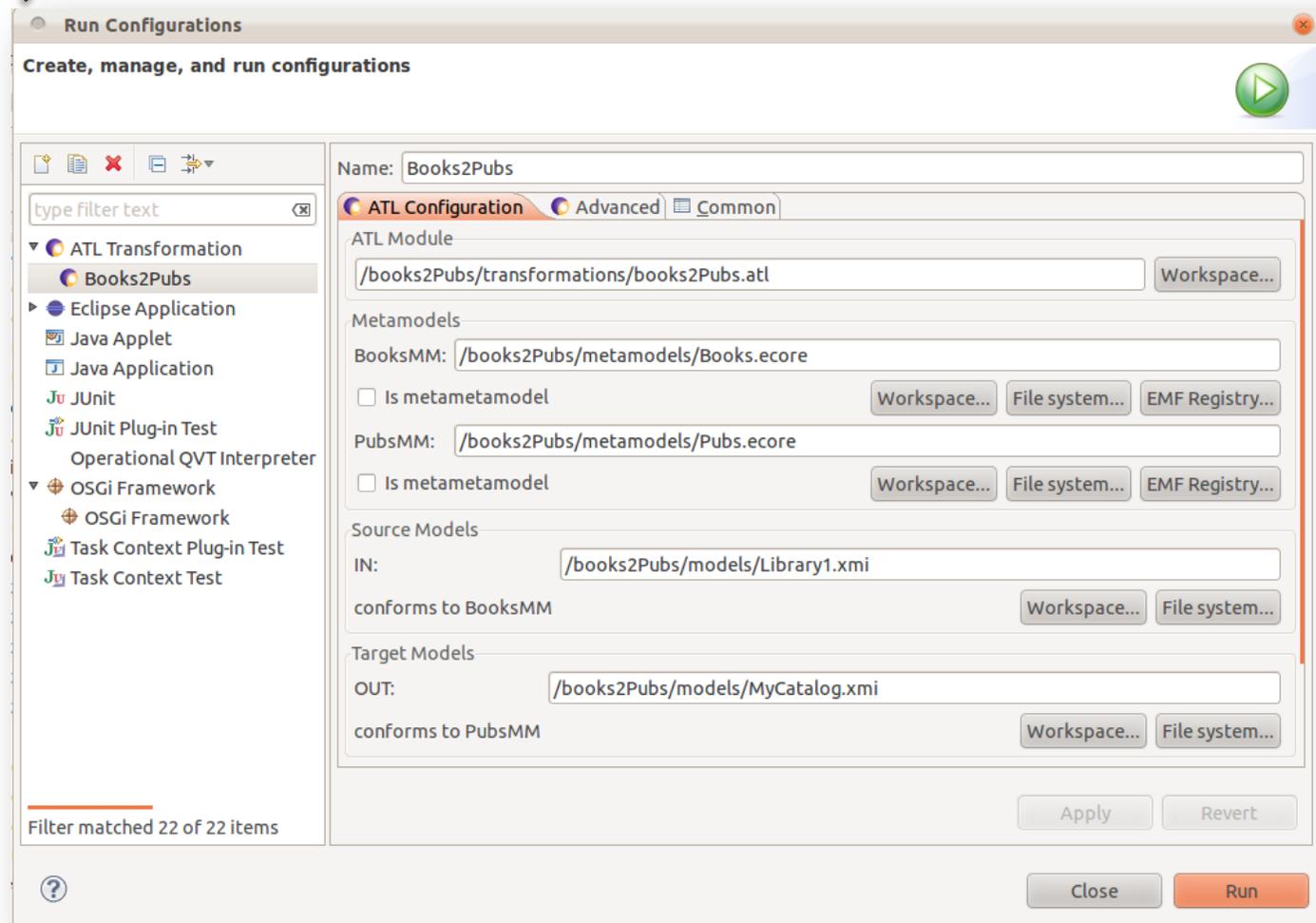
The screenshot displays an IDE window titled 'Library1.xmi'. The project structure is as follows:

- platform:/resource/books2Pubs/models/Library1.xmi
  - Library
    - Book UML y patrones
    - Book MDA Explained: The Model Driven Architecture
      - Chapter The MDA Development Process
      - Chapter Defining Your Own Transformations
  - platform:/resource/books2Pubs/metamodels/Books.ecore

The Properties view at the bottom shows the following table:

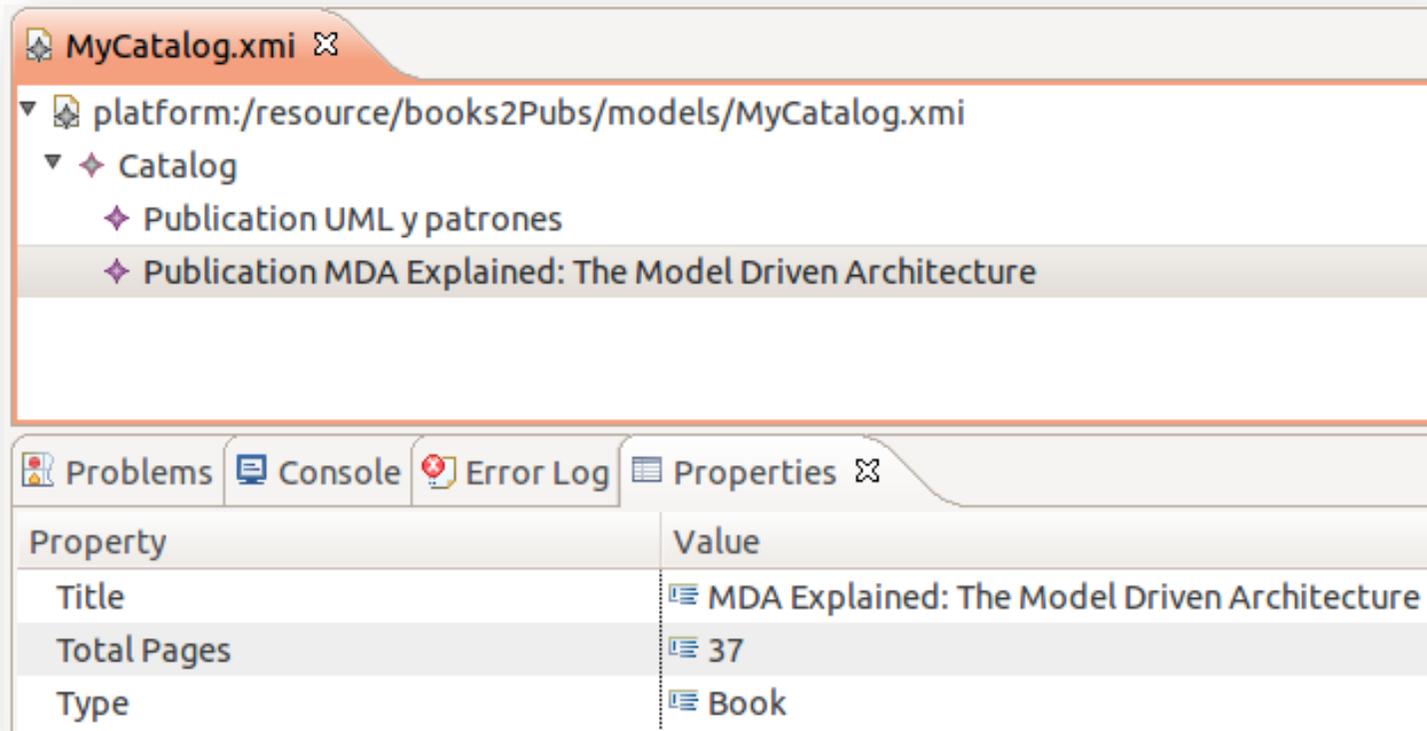
Property	Value
Title	MDA Explained: The Model Driven Architecture

# Ejecución de la transformación



Run Configurations...

# Resultado: modelo destino



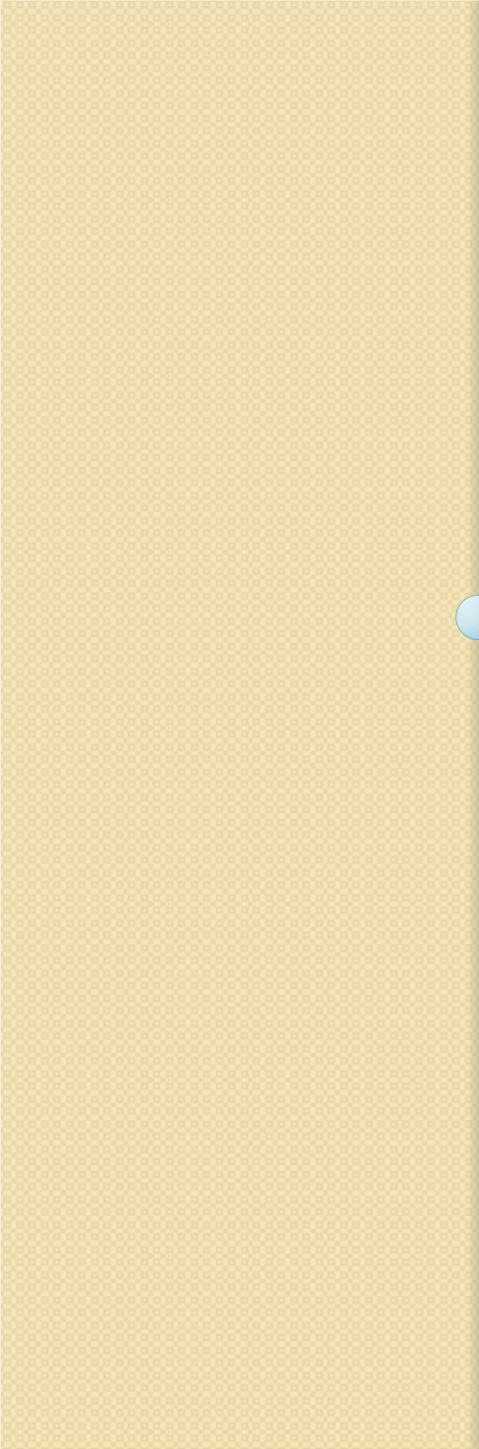
The screenshot shows an IDE window titled 'MyCatalog.xmi'. The tree view displays the following structure:

- platform:/resource/books2Pubs/models/MyCatalog.xmi
  - Catalog
    - Publication UML y patrones
    - Publication MDA Explained: The Model Driven Architecture

The 'Properties' view is active, showing the following table:

Property	Value
Title	MDA Explained: The Model Driven Architecture
Total Pages	37
Type	Book

Nota: Para poder visualizar el modelo destino hay que registrar previamente su metamodelo



TRANSFORMACIONES DE MODELO A MODELO CON  
ATL



# RESUMEN

# ¿Qué hemos aprendido hoy?

- El framework ATL incorpora un lenguaje basado en OCL para diseñar reglas de transformación y una máquina virtual sobre la cual ejecutarlas.
- Desarrollar transformaciones de modelo a modelo (M2M).
- Las transformaciones pueden ser tanto endógenas como exógenas y con multiplicidad N:M.

# Transformaciones de modelo a modelo con ATL

**Iván Ruiz Rube**

ivan.ruiz@uca.es