

<h1>AMW Use Case</h1> <h2>Ontology Annotation Metamodel</h2>	Guillaume Hillairet <a href="mailto:g.hillairet@gmail.com">g.hillairet@gmail.com</a> SIDO Group - L3I
<h3>User Guide</h3>	Date: 05/04/2007 Version 0.1

## Installation

This use case required Eclipse + EMF + ATL + AM3 + AMW to be installed.

- Download ATL/AM3: <http://www.eclipse.org/m2m/atl/download/>
- Download AMW: <http://www.eclipse.org/gmt/amw/download/>

If you want to use latest version, you can download ATL and AMW sources from eclipse CVS. Read instructions on the ATL wiki: [http://wiki.eclipse.org/index.php/ATL/How\\_Install\\_ATL\\_From\\_CVS/](http://wiki.eclipse.org/index.php/ATL/How_Install_ATL_From_CVS/).

Download the archive file containing this use case. Unzip it into a new ATL project. The project structure should look like this:

### MOF\_OAM2OWL/

- **Metamodels/**
  - OAM/
    - OAM.km3
    - OAM.ecore -- The OAM Metamodel
  - OWL/
    - XMLSyntax/
      - OWL2XML.atl -- ATL Transformation OWL2XML
    - OWL.km3
    - OWL.ecore -- OWL Metamodel
    - XML.ecore -- XML Metamodel
- **Models/**
  - WeavingModels/
    - MySample.amw -- Sample Annotation Model
  - Museum.km3 -- Museum Model in km3
  - Museum.ecore -- Museum Model in Ecore
  - Museum-OWL.ecore -- Museum Model conform to OWL
  - Museum-XML.ecore -- Museum Model conform to XML
- MOF\_OAM2OWL.atl -- ATL Transformation MOF\_OAM2OWL

This project contains the various metamodels and transformations needed to perform the transformations between the MOF to the OWL/XML syntax.

## Ontology Annotation Metamodel

The Ontology Annotation Metamodel (OAM) lets you define annotations on models in order to produce ontologies. OAM defines a set of annotations representing the different concepts of the OWL language. In order to use it you may have to understand a little bit the concepts behind ontology languages like OWL. In this section we will first explain basic concepts of OWL and then explain how to use OAM annotations.

Ontology language like OWL lets you define ontologies, which are usually used to described and share information in a way that both humans and machines can understand. An ontology usually defines four types of concepts: Classes,

Properties, Relations and Individuals. An ontology Class does not represent a structure like in OO languages but is used to represent a set of resources that are of the same kind. Properties represent links that can exist between Classes and relations link Classes to Properties. Individuals are the representation of resources and are typed by a Class.

Classes can be composed with other classes with some set operators like union, intersection, and complement. Restrictions are kind of classes that represent a set of individuals that respect certain conditions. Restrictions can be made to set properties cardinalities (minimum, maximum) or to fix properties values with operators like someValuesFrom or hasValue.

Properties are of two types: Datatype and Object Properties. A property has a domain and a range (type). A Datatype property is a property that links a Class to a literal (e.g. String value). An object property links two Classes. A property can be functional if its range cardinality is fixed to one. An Object property can be symmetric property if domain and range are the same class. It also can be transitive or inverse functional if domain cardinality is one.

All these concepts are present in OAM in the form of annotations to be applied. The figure below shows an excerpt of the different ontology concepts present in OAM and also AMW Core elements that are extended to define OAM.

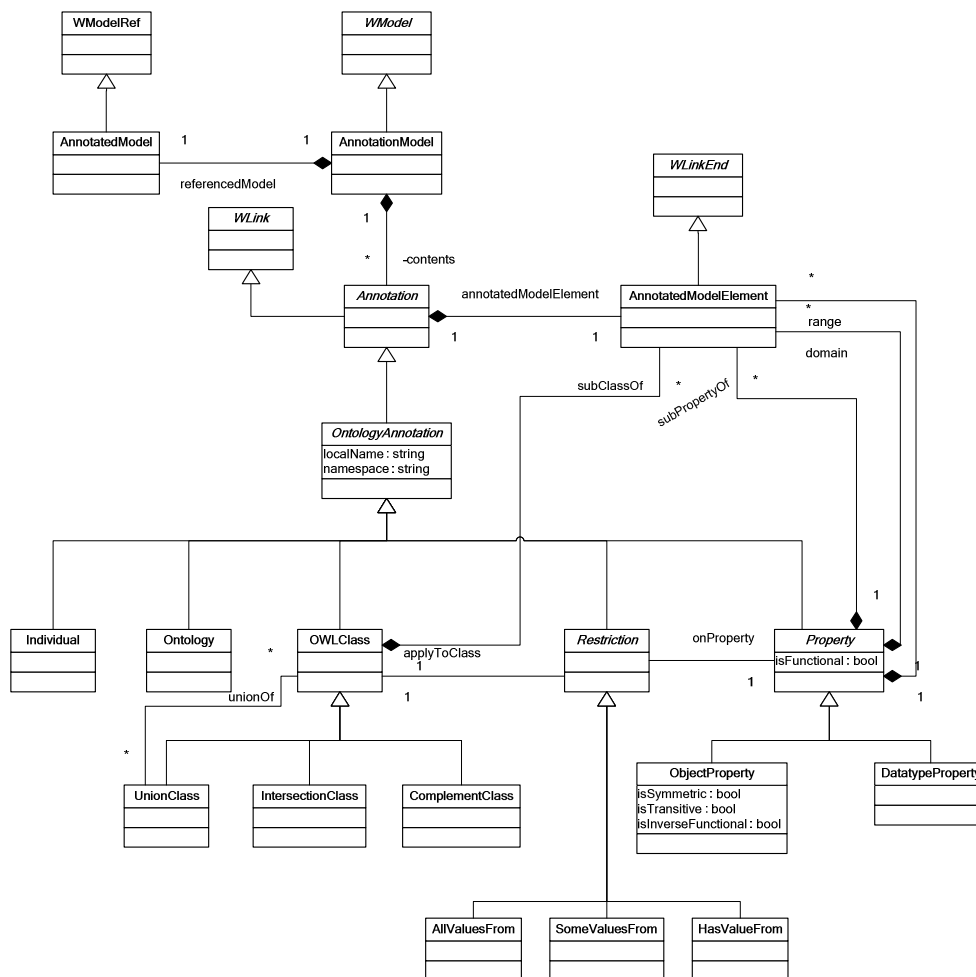


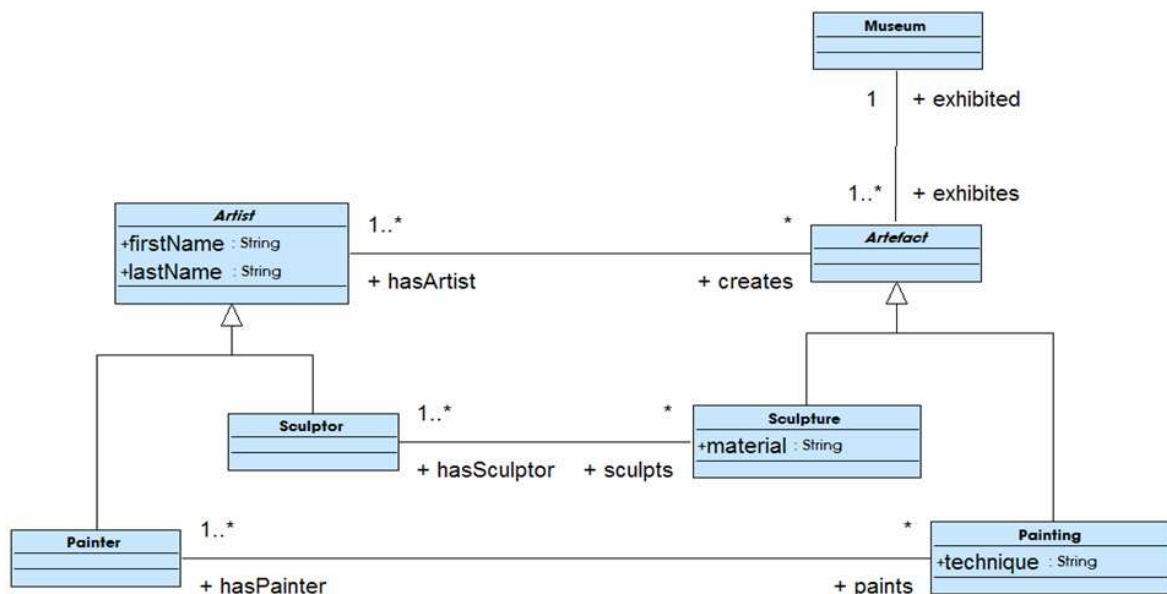
Figure 1 OAM Metamodel.

Annotations can not be applied to any model elements. Some conditions must be respect in order to execute the transformation MOF\_OAM2OWL. These constraints are detailed here:

- Ontology annotation must be applied to a MOF!EPackage
- OWLClass annotation must be applied to MOF!EClass
- DatatypeProperty annotation must be applied to a MOF!EAttribute
- ObjectProperty annotation must be applied to a MOF!EReference
- Property Domain must be of type MOF!EClass
- DatatypeProperty Range must be a datatype
- ObjectProperty Range must be a MOF!EClass

## Execute the Use Case

We will now explain the creation of an annotation model and the execution of the transformation MOF\_OAM2OWL. We will use the example model (Museum.ecore) provided within the use case. A class diagram representation of the museum model is provided below.



**Figure 2 Museum Model.**

From this model we want to extract some information in order to produce an ontology written in the OWL language. For that we will follow several steps: first we create an annotation model with AMW. Second step, we execute the transformation MOF\_OAM2OWL which takes as input the Museum Model and the annotation Model and produce as output an OWL Model representing the Ontology. Finally in order to obtain an OWL file in XML format we use an OWL/XML extractor. The complete scenario execution is represented by the following figure.

In the next section we explain the following steps for creating an annotation model with AMW.

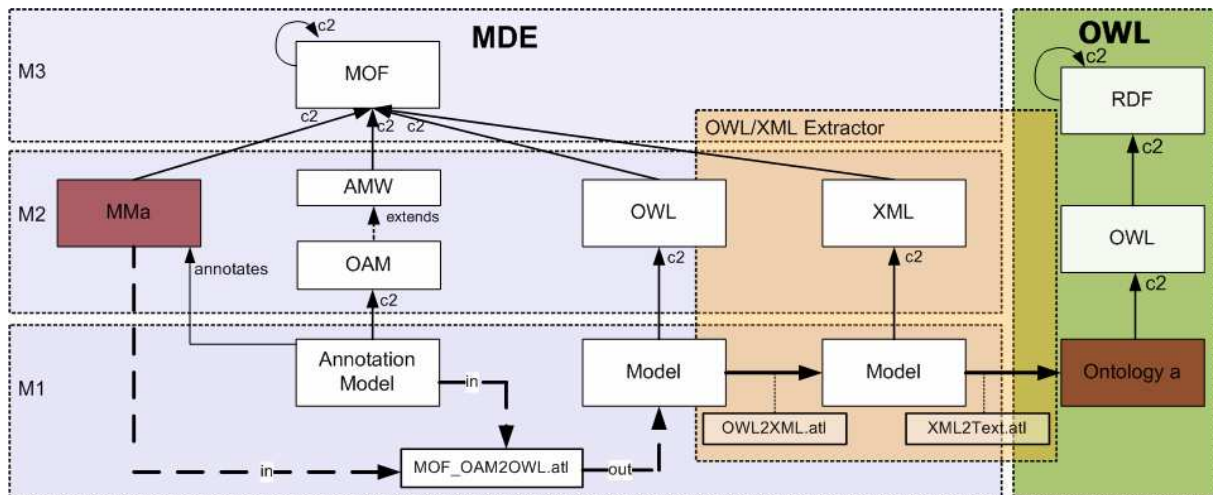


Figure 3 From a MOF Model to a OWL Ontology.

## Creation of an Annotation Model

Create a new Weaving Model via the menu File -> New. A wizard will appear like the one in screenshot below. Select load KM3 extensions and Browse to select Km3 local file. Select the file OAM.km3 in your project. This file is the OAM metamodel in KM3 format. Then press load selected file, now your wizard should look like this:

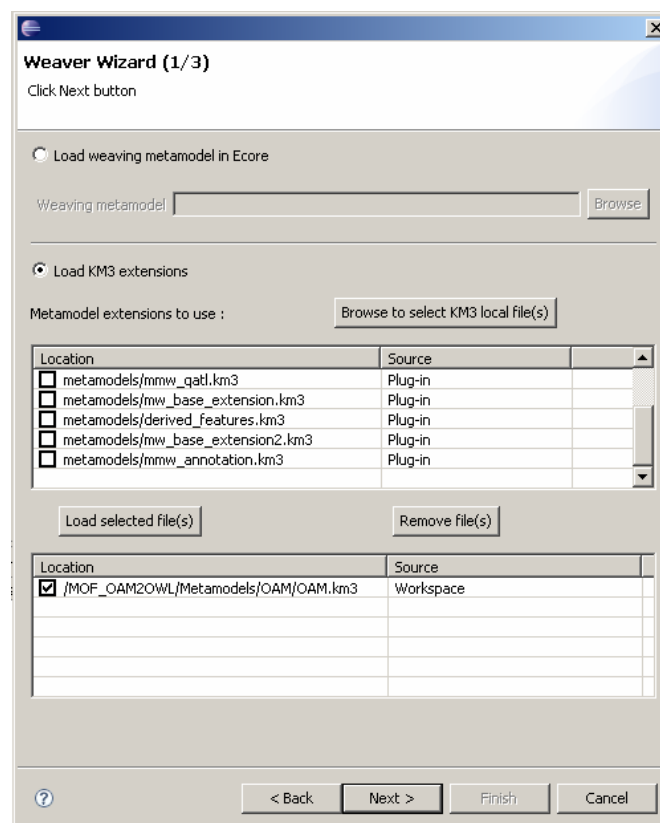
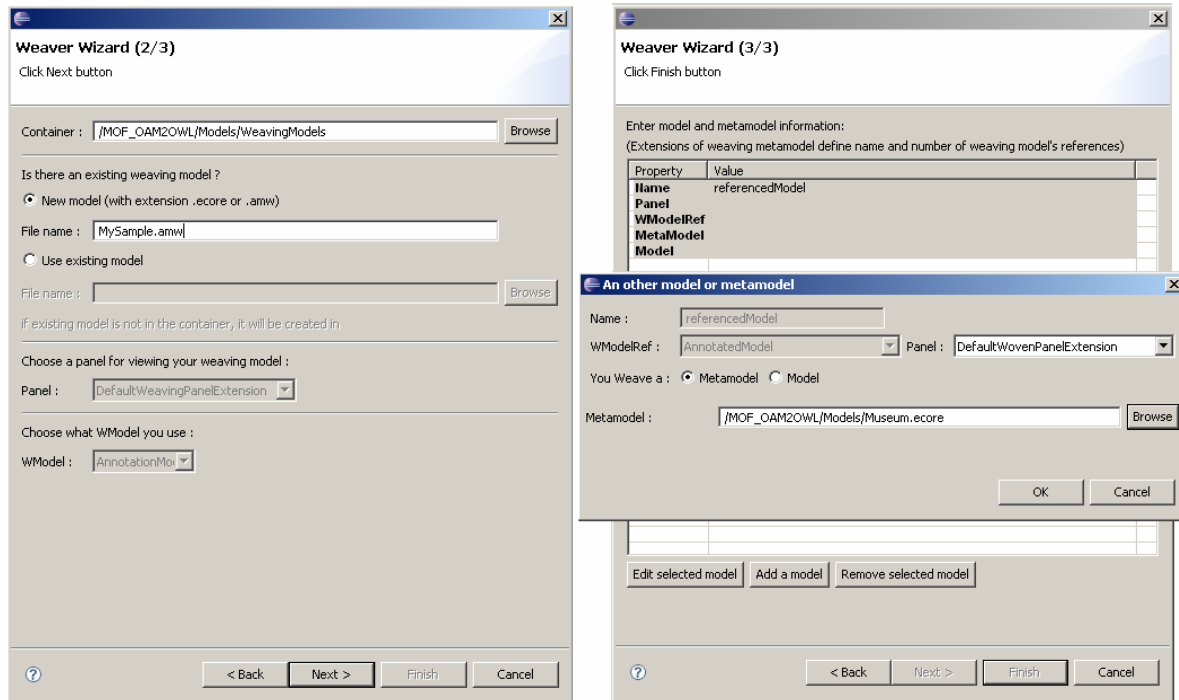


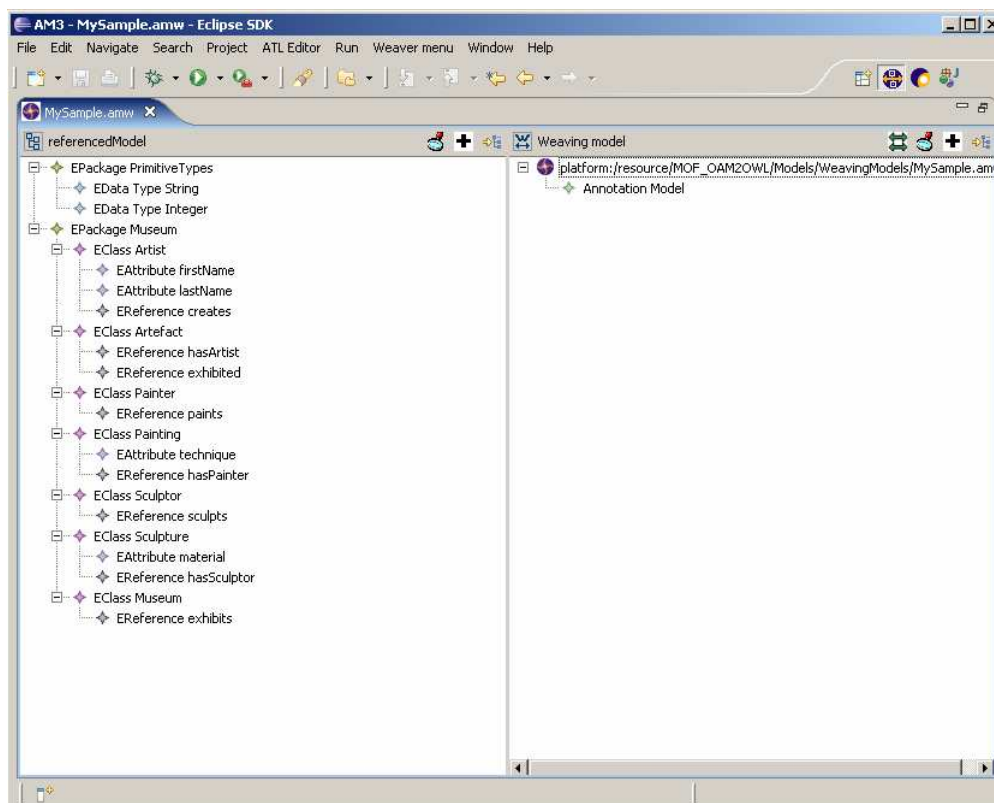
Figure 4 Weaver Wizard

Go to the next page of the wizard, in this one you have to enter a name for your annotation model, for example MySample.amw. Proceed to the third and last wizard page.



**Figure 5 Weaver Wizard page 2 and 3**

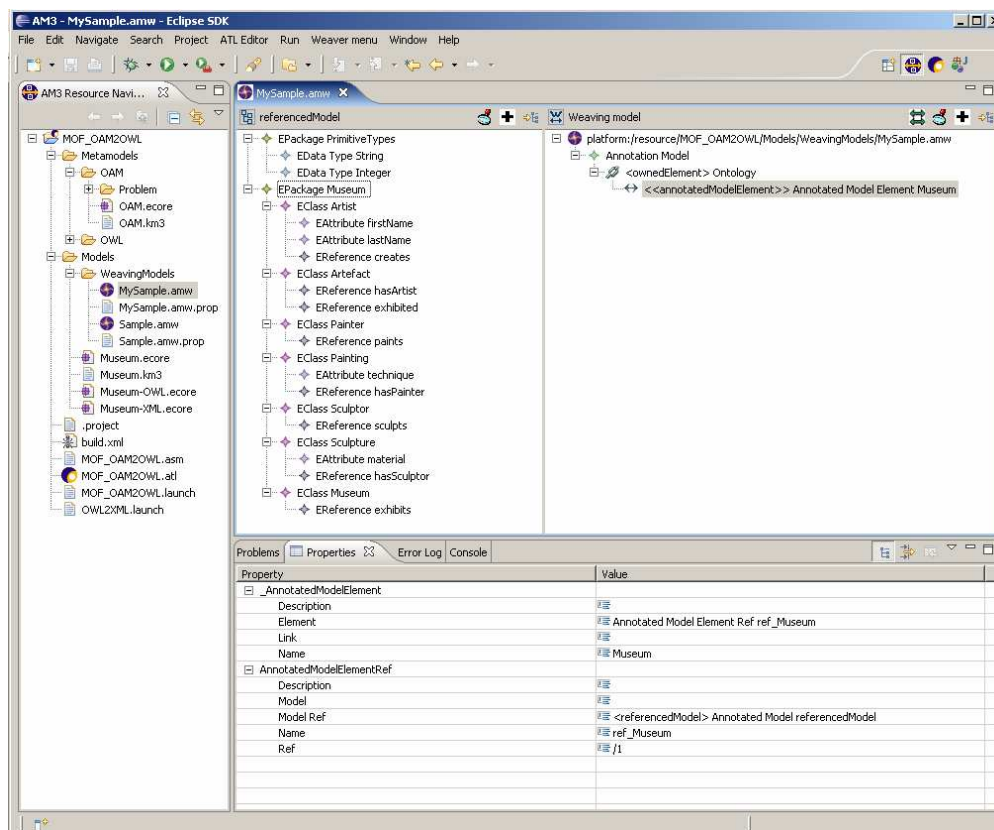
Enter Edit selected model, a window appear to help you selecting the input model, the Museum for our example. Press Browse and select the file Museum.ecore in Models folder. Then press Ok and finish, you have created your annotation model, now you can edit it with the AMW interface. Double click on your model file (MySample.amw) you should have an AMW editor which look like this one.



**Figure 6 Open your annotation model file with AMW**

As you can see, the editor contains on the left the referring model (Museum) and on the right the annotation model. We are now able to create annotations and applied them to elements on the left model.

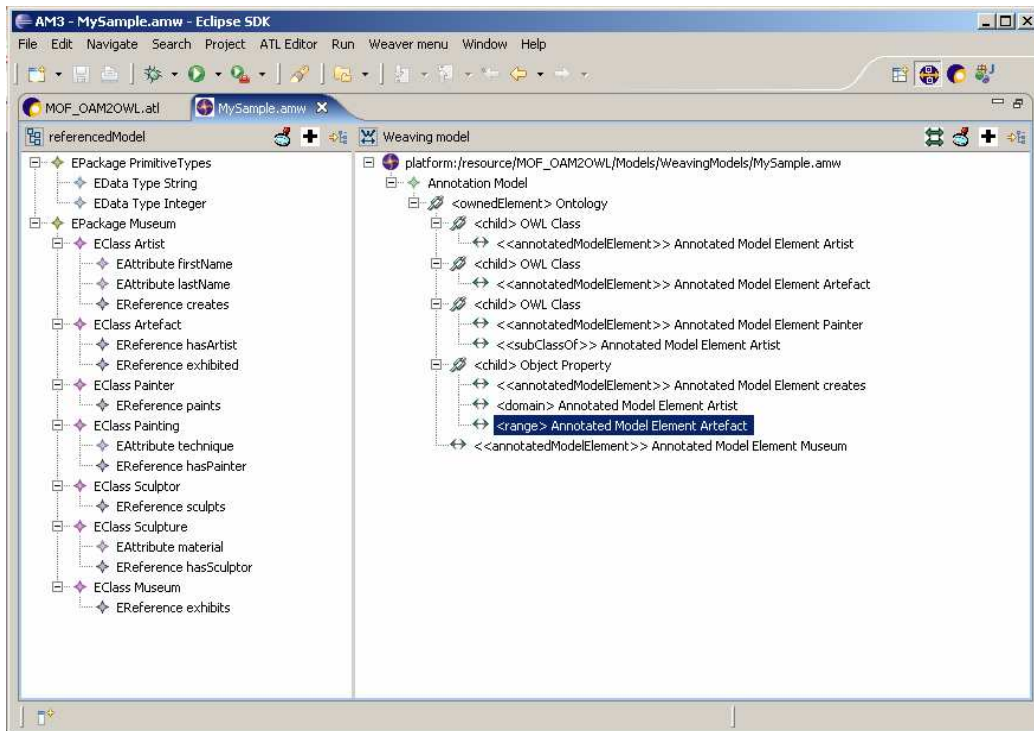
First we will create an Ontology annotation. Make a right click on the Annotation Model element on right model. Choose New Child -> Ontology. Now select the package Museum on the left and drag and drop it to the ontology annotation. You should see a menu appearing, select annotated Model Element. This action creates the link between the annotation and the model element. You should now have an annotation model like the now on figure below. Select the Ontology annotation and go to the properties view. Enter a namespace for the ontology, for example *http://www.example.org/Museum*. This information is important because each ontology have a unique identifier, its namespace.



**Figure 7 Create an ontology annotation**

Now you can create OWLClass annotations and properties annotations. These annotations must be child of an Ontology annotation. In the next steps we will annotate the classes *Artist* and *Artefact* as OWLClass, the attribute *firstName* as DatatypeProperty and the reference *creates* as ObjectProperty.

So first make a right click on the Ontology annotation and select New Child -> OWLClass. Reproduce that to create another OWLClass annotation, a DatatypeProperty and ObjectProperty annotation. Drag and drop the class *Artefact* to an OWLClass annotation and select annotated Model Element like for the ontology annotation. Do the same for the class *Artist*. Select the attribute *firstName* and do a drag and drop to the DatatypeProperty annotation. Also do the same for the *creates* reference. You have now link each left element to an annotation.



## Execute the transformation

When you have finished editing your annotation model, the time has come to run the MOF\_OAM2OWL transformation to produce the desired ontology. This transformation takes as input a MOF Model and an OAM Model. In our example these models are the Museum.ecore and MySample.amw. A configuration example is provided in the following picture. It is important to note that this transformation required AMW as Model Handler for the AMW metamodel (see red square in the screenshot).

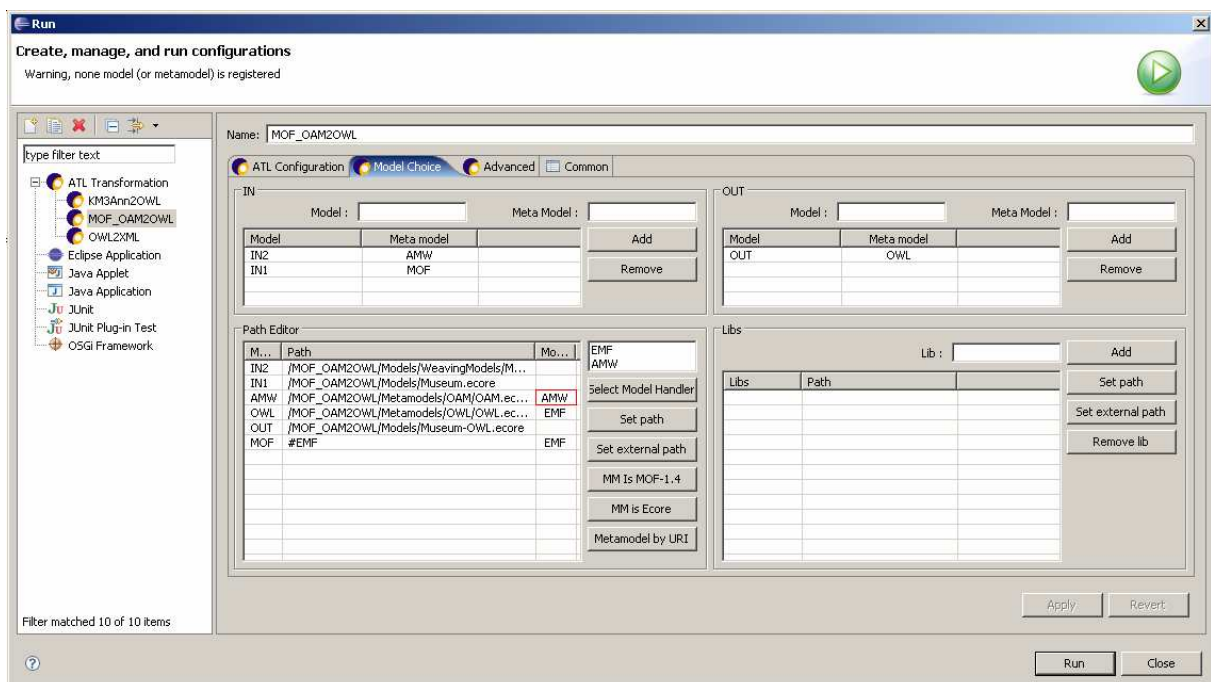


Figure 8 MOF\_OAM2OWL configuration

## Annexe

### OAM Metamodel in KM3

```

package OAM {
  class AnnotationModel extends WModel {
    -- @subsets ownedElement
    reference contents[*] ordered container : Annotation;
    -- @subsets wovenModel
    reference referencedModel container : AnnotatedModel;
  }

  -- @welementRefType AnnotatedModelElementRef
  class AnnotatedModel extends WModelRef {
  }

  -- @wmodelRefType AnnotatedModel
  class AnnotatedModelElementRef extends WElementRef {
  }

  --@begin ANNOTATIONS
  abstract class Annotation extends WLink {
    -- @subsets end
    reference annotatedModelElement[0-1] container : AnnotatedModelElement;
  }

  class AnnotatedModelElement extends WLinkEnd {
  }

  --@begin ONTOLOGY ANNOTATIONS
  abstract class OntologyAnnotation extends Annotation {
    attribute namespace : String;
    attribute localName[0-1] : String;
  }

  class Ontology extends OntologyAnnotation {
    attribute nsPrefix : String;
    attribute importsOntology[*] : String;
  }

  class OWLClass extends OntologyAnnotation {
    -- @subsets end
    reference subClassOf[*] container : AnnotatedModelElement;
    reference equivalentClass[*] : OWLClass;
  }

  class UnionClass extends OWLClass {
    reference unionOf [*] : OntologyAnnotation;
  }

  class IntersectionClass extends OWLClass {
    reference intersectionOf [*] : OntologyAnnotation;
  }

  class ComplementClass extends OWLClass {
    reference complementOf : OntologyAnnotation;
  }

  abstract class Property extends OntologyAnnotation {
    attribute isFunctional[0-1] : Boolean;
    -- @subsets end
    reference subPropertyOf[*] container : AnnotatedModelElement;
    -- @subsets end
    reference domain[*] container : AnnotatedModelElement;
    -- @subsets end
    reference range[*] container : AnnotatedModelElement;
  }
}

```

```

class DatatypeProperty extends Property {
}

class ObjectProperty extends Property {
  attribute isTransitive : Boolean;
  attribute isSymmetric : Boolean;
  attribute isInverseFunctional[0-1] : Boolean;
}

abstract class Restriction extends OntologyAnnotation {
  reference onProperty : Property;
  reference subClass : OWLClass;
}

class SomeValuesFrom extends Restriction {
  reference someValueFromClass[0-1] : OWLClass;
  reference someValueFromDataRange[0-1] : DataType;
}

class AllValuesFrom extends Restriction {
  reference allValueFromClass[0-1] : OWLClass;
  reference allValueFromDataRange[0-1] : DataType;
}

class HasValueFrom extends Restriction {
  reference hasIndividualValue[0-1] : Individual;
  reference hasLiteralValue[0-1] : String;
}

class DataType extends OntologyAnnotation {
  attribute xsdType : XsdType;
}

class EnumeratedClass extends OWLClass {
  reference oneOf [*]: Individual;
}

class Individual extends OntologyAnnotation {
}

enumeration XsdType {
  literal "http://www.w3.org/2001/XMLSchema#string";
  literal "http://www.w3.org/2001/XMLSchema#nonNegativeInteger";
  literal "http://www.w3.org/2001/XMLSchema#boolean";
  literal "http://www.w3.org/2001/XMLSchema#integer";
}
}

```