

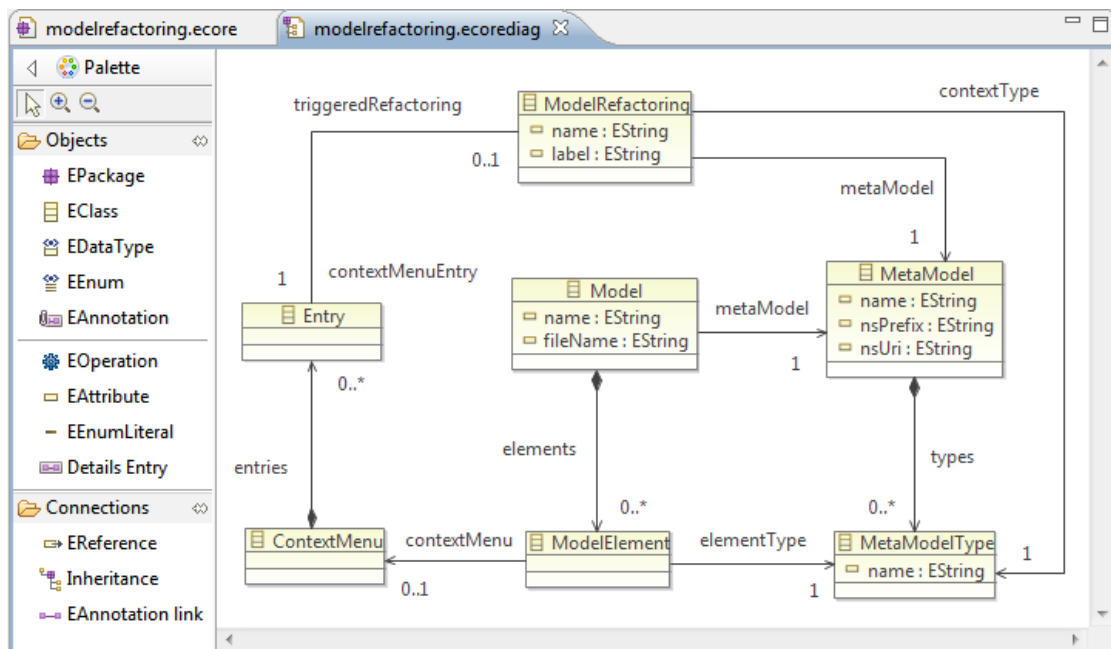
# How to apply EMF model refactorings

Thorsten Arendt

January 7, 2011

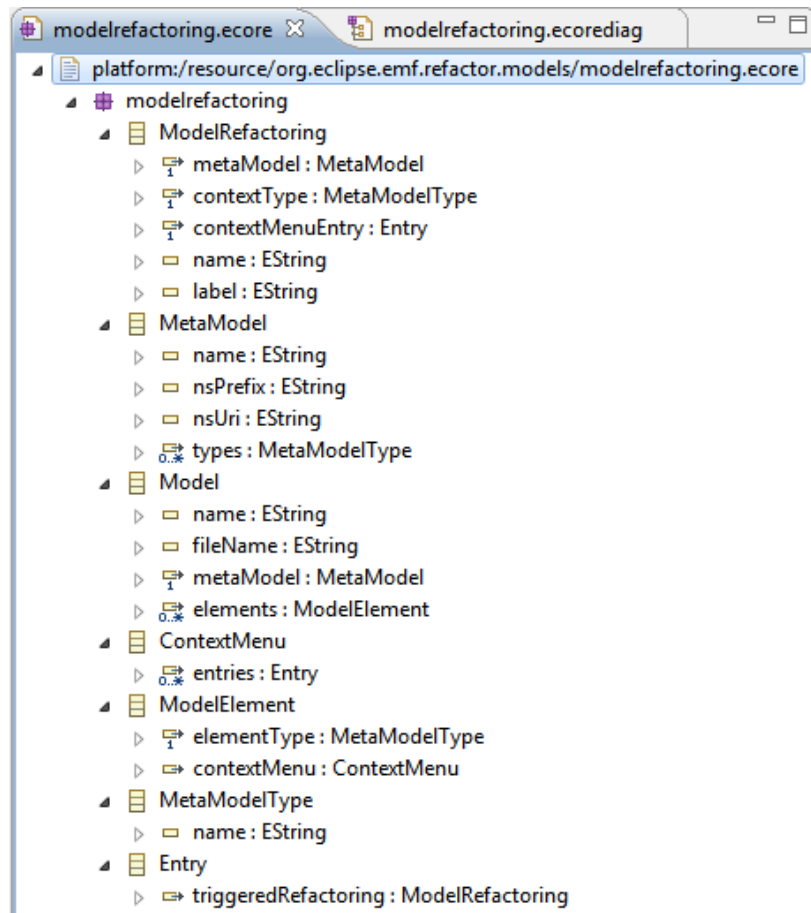
This manual presents the application of an EMF model refactoring using **EMF Refactor**. More precisely, we demonstrate the model refactoring **Move EAttribute** for Ecore models. Please note, that EMF Refactor can be used for refactorings of any models whose meta model is based on EMF Ecore.

Let's take a look to the following Ecore diagram presenting a first model concerning EMF model refactorings in an early stage of the EMF Refactor development process. A **ModelRefactoring** has a name and conforms to a **MetaModel** that is specified by name, namespace prefix, and namespace URI. Furthermore, it has a label that should be shown as an **Entry** in the **ContextMenu** of an arbitrary **ModelElement**. A **ModelElement** belongs to a **Model** that is specified by a name and stored in a file with a specific name. Furthermore, a **Model** conforms to a **MetaModel** and each **ModelElement** is typed over a specific **MetaModelType** belonging to the corresponding **MetaModel**. Besides the afore mentioned attributes, each **ModelRefactoring** is related to a **MetaModelType** representing the type of the contextual element the refactoring can be applied on.



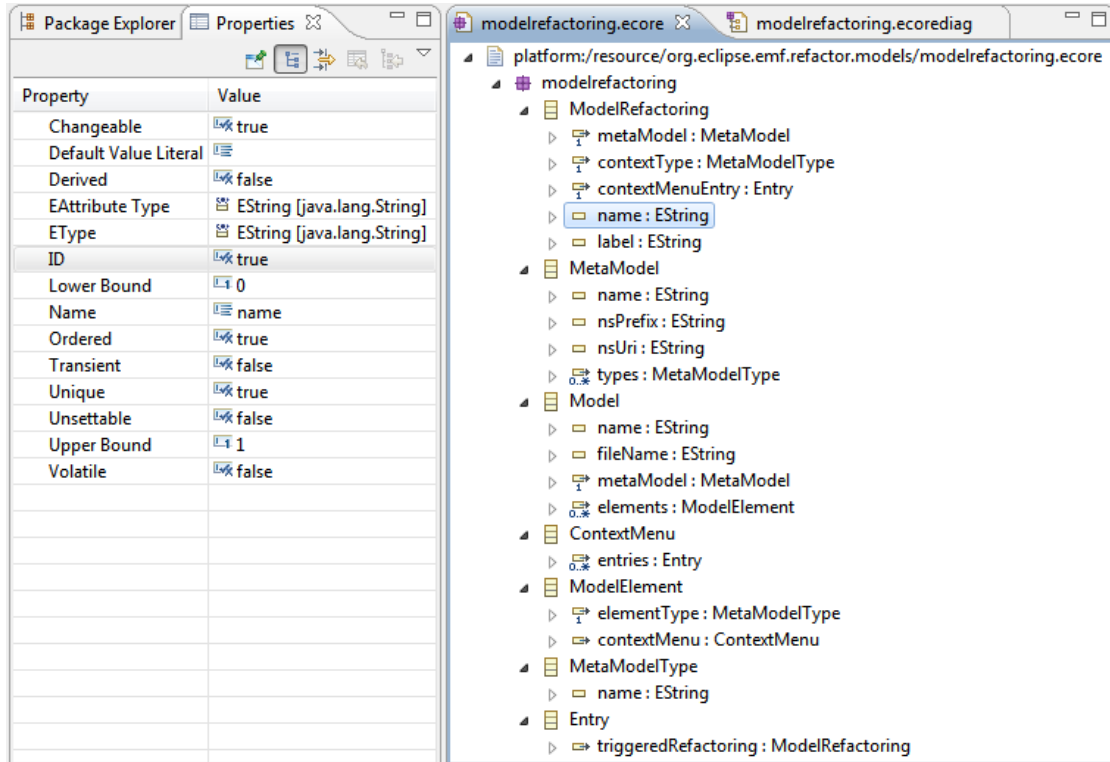
During software design it became questionable whether attribute `label` of class `ModelRefactoring` could be better placed in class `Entry`. So, model refactoring **Move EAttribute** is the next task to be performed.

Since **EMF Refactor** can be used on arbitrary EMF based models the application of a specific refactoring is mainly triggered from within the EMF instance editor. The next figure shows the example model from above using this tree-based editor.

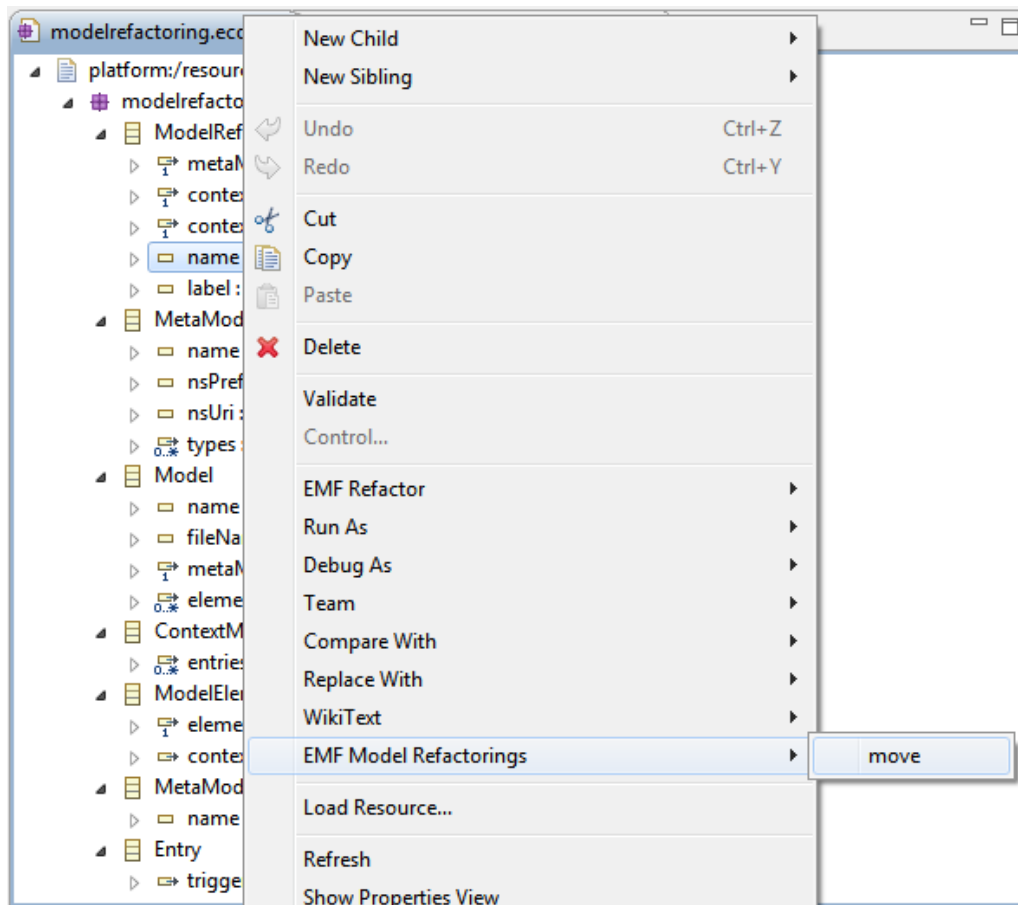


EMF model refactoring **Move EAttribute** can be specified in the following way: First, it has to be checked whether the contextual `EAttribute` is not marked as ID of the containing class, and whether this class has at least one referenced class. If these (initial) checks pass the user has to put in the name of the class the attribute has to be moved to. Then, it has to be checked whether the containing class has a referenced class with the specified name, and whether this class does not already owns an attribute with the same name as the contextual attribute. If these (final) checks pass the contextual attribute can finally be moved to the specified class.

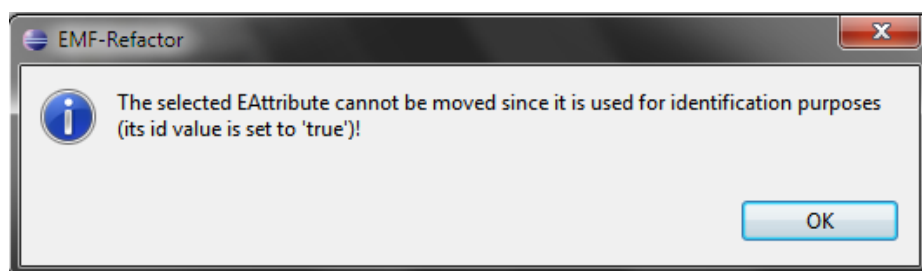
Before applying the refactoring let us first present some erroneous situations. The following figure shows that attribute name of class ModelRefactoring represents the ID of this class. So, the application of refactoring **Move EAttribute** on attribute name is not possible.



Each refactoring can be triggered from within the context menu of a specific contextual model element. The next figure shows the context menu of attribute `name`.

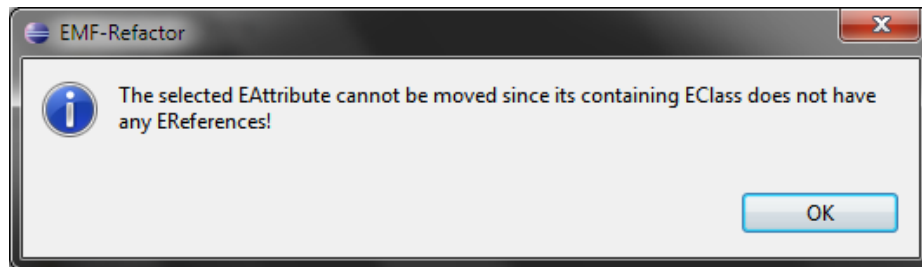


As expected, **EMF Refactor** does not apply refactoring **Move EAttribute** because of the violated precondition. The following figure shows the corresponding error message.

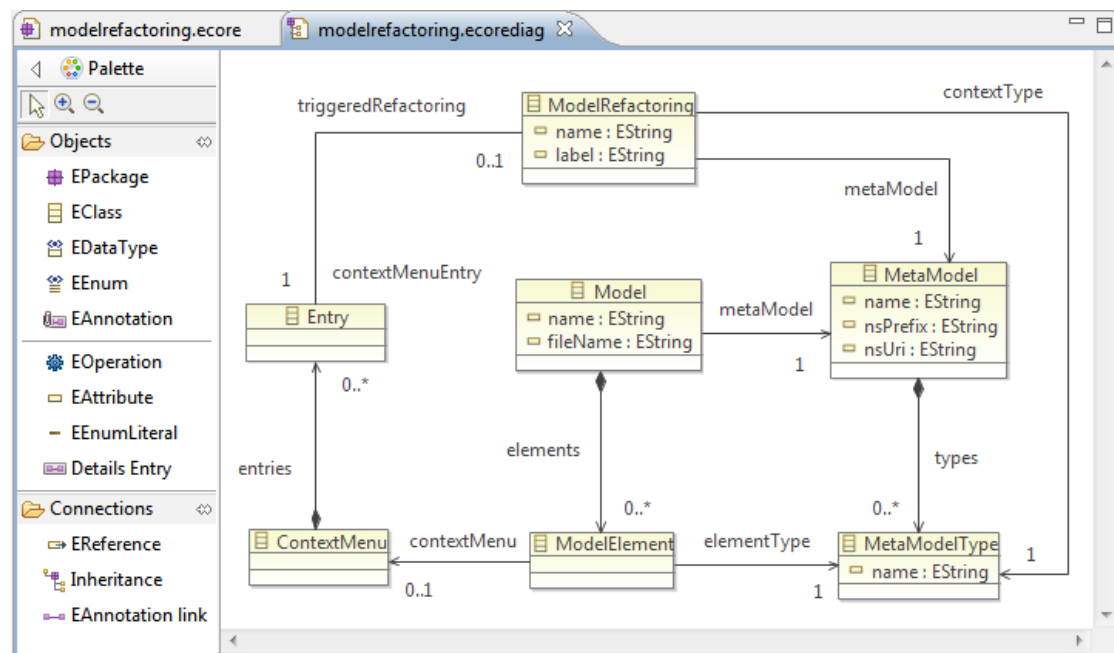


The second erroneous situation occurs if we try to apply refactoring **Move EAttribute** on attribute name of class `MetaModelElement`. As you see in the diagram on page 1, this class does not have any (outgoing) references to other classes in our example model.

We trigger refactoring **Move EAttribute** from within the context menu of attribute name of class `MetaModelElement` and again, **EMF Refactor** does not apply refactoring **Move EAttribute** because of the violated precondition. The following figure shows the corresponding error message.

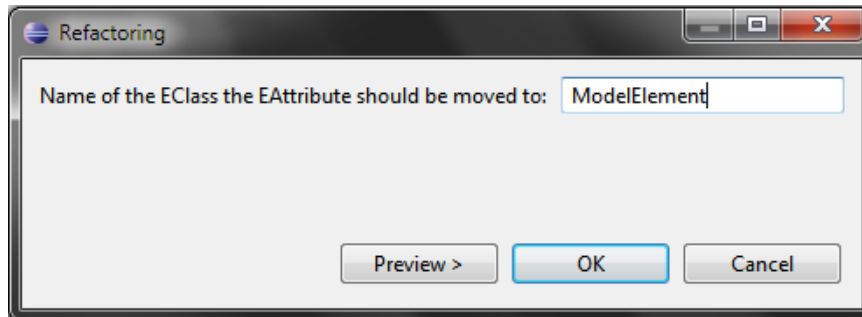


Now, let's try to apply refactoring **Move EAttribute** on attribute name of class `MetaModel1`. The initial checks pass, i.e. the attribute is not the ID of its containing class and this class is referenced to class `MetaModelType` (see following figure).

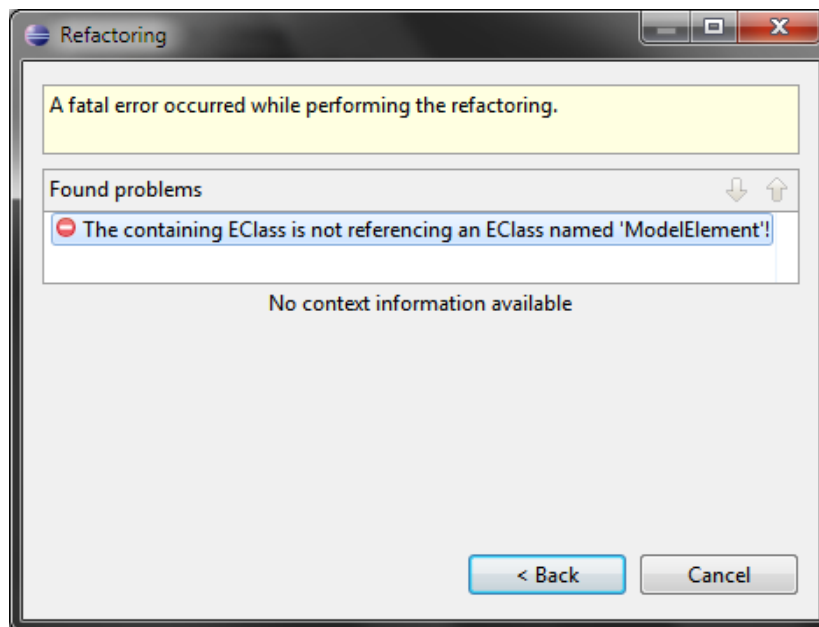


We trigger refactoring **Move EAttribute** from within the context menu of attribute name of class `MetaModel1`. The initial precondition checks pass, i.e. **EMF Refactor** does not display any error messages, and a user input form appears specific to the triggered EMF model refactoring. For refactoring **Move EAttribute** we now have to input the

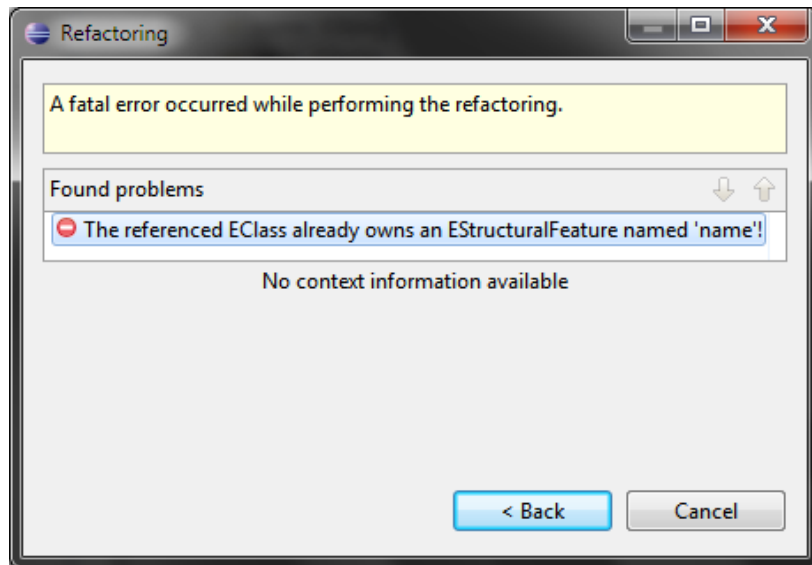
name of the class the contextual attribute should be moved to. The following figure shows the input dialog. Here, another erroneous situation arises if we type in a name of a class that is not referenced by the containing class of the contextual attribute (or even a complete invalid name), for example ModelElement.



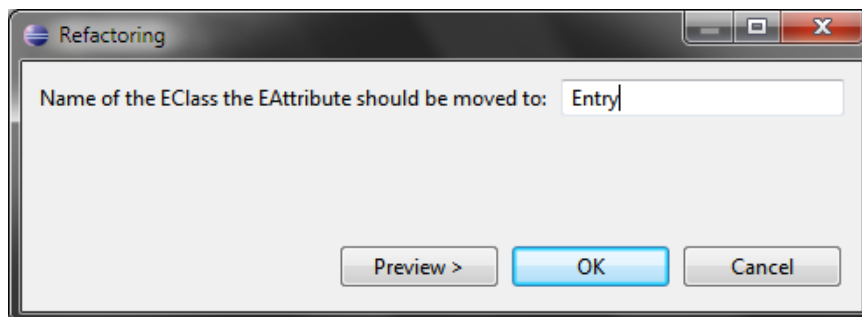
Again, **EMF Refactor** does not apply refactoring **Move EAttribute** because of the violated precondition. The following figure shows the corresponding error message.



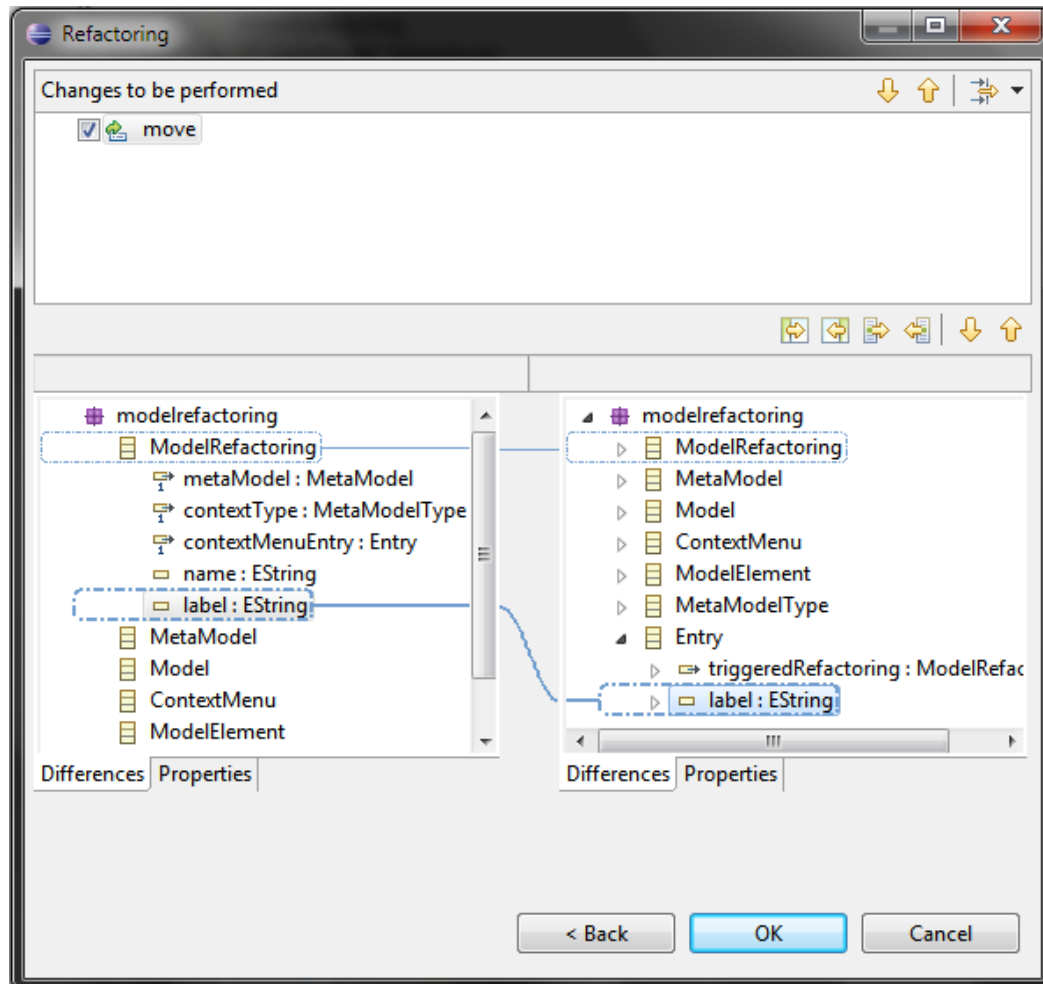
The last erroneous situation occurs if we try to move attribute `name` of class `MetaModel` to the referenced class `MetaModelType`. Here, class `MetaModelType` already owns an attribute `name` (see diagram on page 5). We trigger refactoring **Move EAttribute** from within the context menu of attribute `name` of class `MetaModel`. In the refactoring parameter dialog we type in name `MetaModelType`. Again, **EMF Refactor** does not apply refactoring **Move EAttribute** because of the violated precondition. The following figure shows the corresponding error message.



Now, all possible erroneous situations of EMF refactoring **Move EAttribute** are presented and we can continue with a successful refactoring application. As already mentioned above, attribute `label` of class `ModelRefactoring` could be better placed in class `Entry` and should be moved (see diagrams on page 1 and 5). We trigger refactoring **Move EAttribute** from within the context menu of attribute `label` of class `ModelRefactoring`. Since the initial precondition checks pass the parameter dialog appears and we type in class name `Entry` (see following figure).

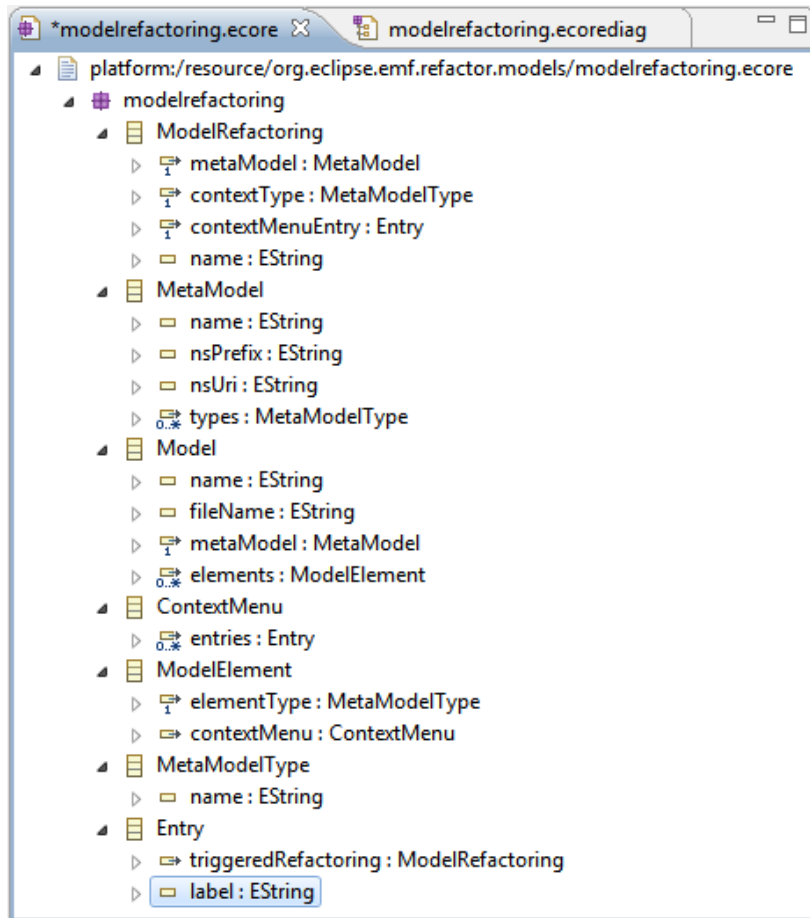


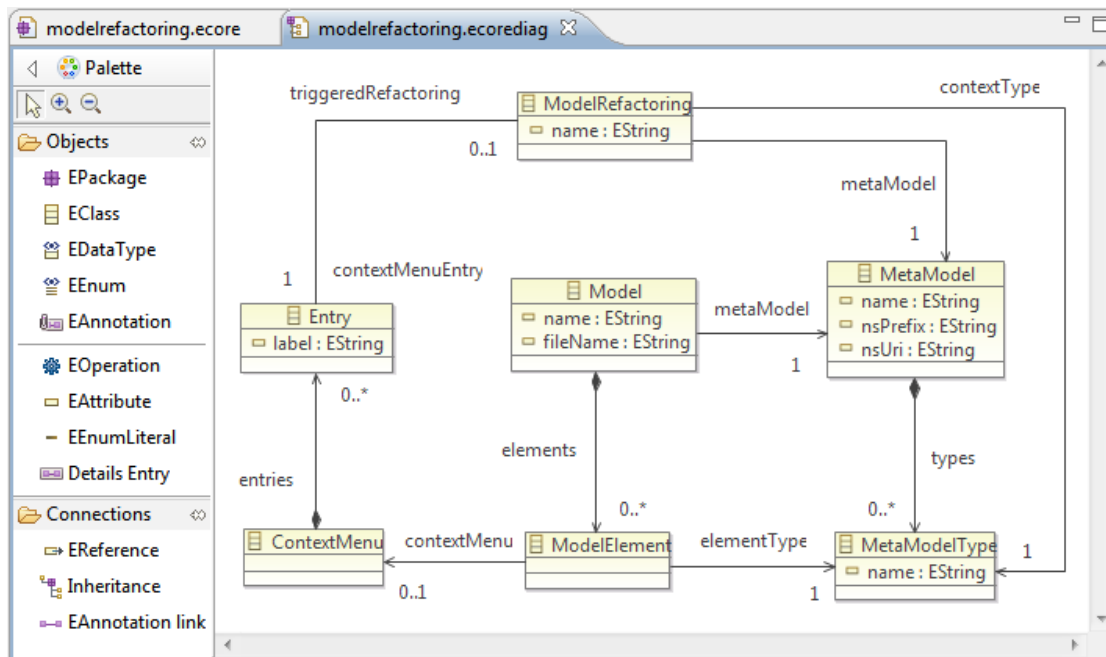
It is possible to obtain a preview of the refactoring action. Here, **EMF Refactor** uses **EMF Compare**. The left hand side of the following figure shows the original model whereas the right hand side presents the refactored model. Model changes are highlighted by colored connections. Here, the dialog shows that attribute `label` was removed from class `ModelRefactoring` and inserted into class `Entry`, i.e. the attribute was moved.



Now, these changes can be committed and refactoring **Move EAttribute** can take place. Of course, **EMF Refactor** provides undo and redo functionality. The following figures show the refactored model using the tree-based EMF instance editor respectively the graphical diagram view of **Ecore Tools**.







- END -