

# Improvements to the Emfatic Editor

Miguel Garcia, 2007-02-14

<http://www.sts.tu-harburg.de/~mi.garcia/>

## Table of Contents

Architecture.....	1
New GUI features .....	2
Show in EMF Type Hierarchy .....	8
ToDo List, Part 1 (priority-ranked).....	9
ToDo Part 2.....	9
Possibilities for future work.....	9
Sample new productions in the grammar.....	10

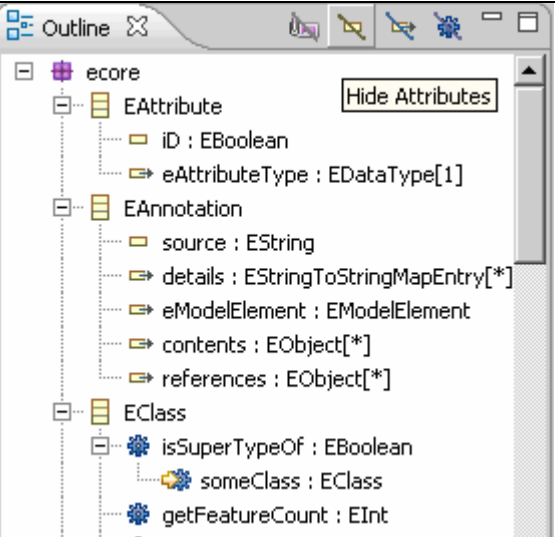
## **Architecture**

The main motivation behind the GUI improvements was adding support for EMF Generics to Christopher J Daly's Emfatic, <http://www.alphaworks.ibm.com/tech/emfatic>. The conversion between .ecore and .emf now understands them, as a consequence Emfatic now requires EMF 2.3 or higher. Examples of new productions in the grammar appear at the end of this document, the new grammar is backwards compatible.

After each keystroke the document is parsed, a CST is built and (if possible) an AST too. A bidirectional multimap keeps the correspondence between textual representation (CST nodes) and AST node. The performance impact is acceptable. Functionality such as "Mark Occurrences" do not resort to exploring the textual document. Instead, the (offset, length) pair is used to get a CST node, from which an AST node can be found, making for robust navigation.

The availability of CST, AST and their bridging will likely simplify the introduction of new functionality (see ToDo List section).

## New GUI features

 <p>The Outline view now displays the same elements as the Sample Ecore Editor, toolbar actions are available for hiding/showing annotations, attributes, references, operations.</p>	<p>The Outline view now displays the same elements as the Sample Ecore Editor, toolbar actions are available for hiding/showing annotations, attributes, references, operations.</p>
<pre>class Stem { }  class FruitUtil {   op Fruit[*] processOrderedSet(Fruit[*]   !ordered op Fruit[*] processSet(!order;   !unique !ordered op Fruit[*] processBa;   !unique op Fruit[*] processSequence(!u   val Fruit[*] orderedSet;   !ordered val Fruit[*] set;   !unique !ordered val Fruit[*] bag;   !unique val Fruit[*] sequence; }</pre>	<p>Mark occurrences highlights usages of the same EClassifier, range indication on the vertical bar spans the EClassifier declaration.</p>

```

@namespace(uri="nw", prefix="nw")
package nw;

+class Util <J> {
}

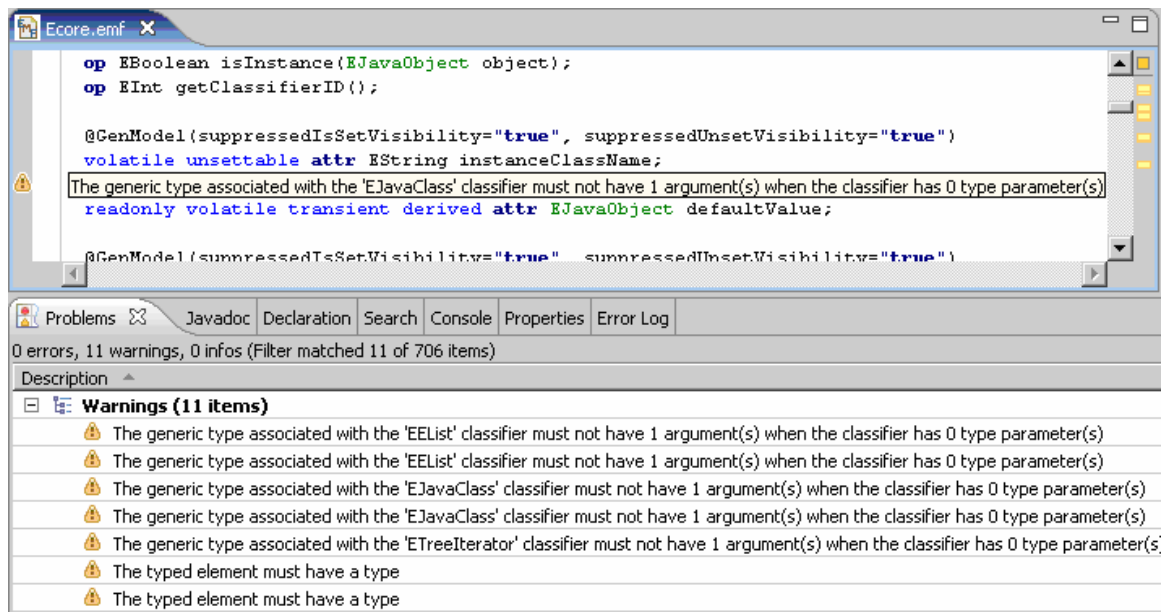
@namespace(uri="nw", prefix="nw")
package nw;

+class Util <J> {
    op <K> EList<?> asList(EJavaObject[
    op <T> void copy(EList<? extends T>
    op Util<?> r();
    transient attr EList<EBigDecimal> a;
}

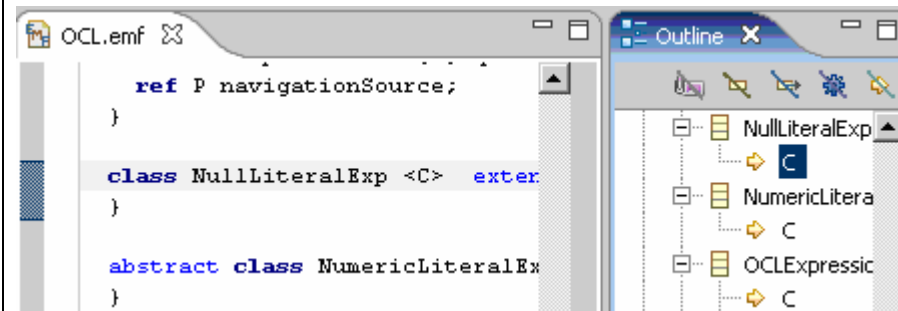
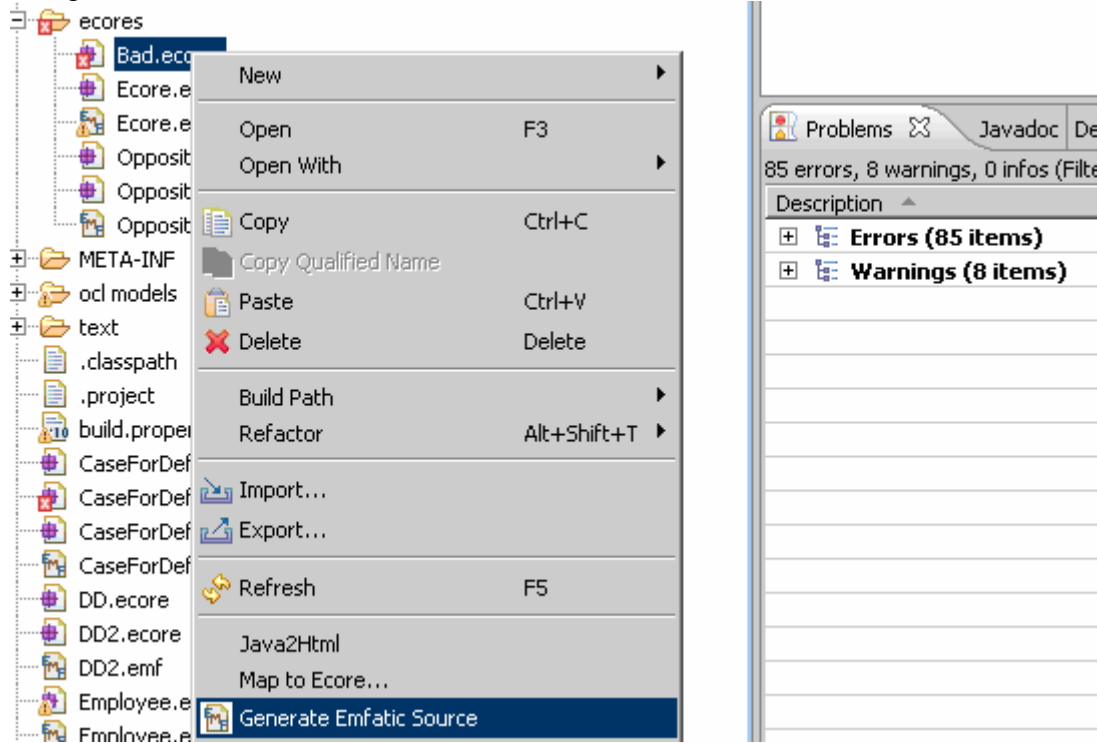
```

Folding is supported, with an annotation hover for collapsed regions.

Problem and warning markers (with squiggles) are updated after each keystroke.



Attempting to generate an .emf for an .ecore triggers an EcoreValidation. Errors and warnings are shown in the Problems view, generation is still allowed if there are only warnings.



The selected outline node follows the cursor position in the text

```
class OperationCallExp <C, O> extends FeatureCallExp<C> {  
    val OCLExpression<C>[*] argument;  
    ref O referredOperation;  
    volatile transient attr int operationCode;  
}
```

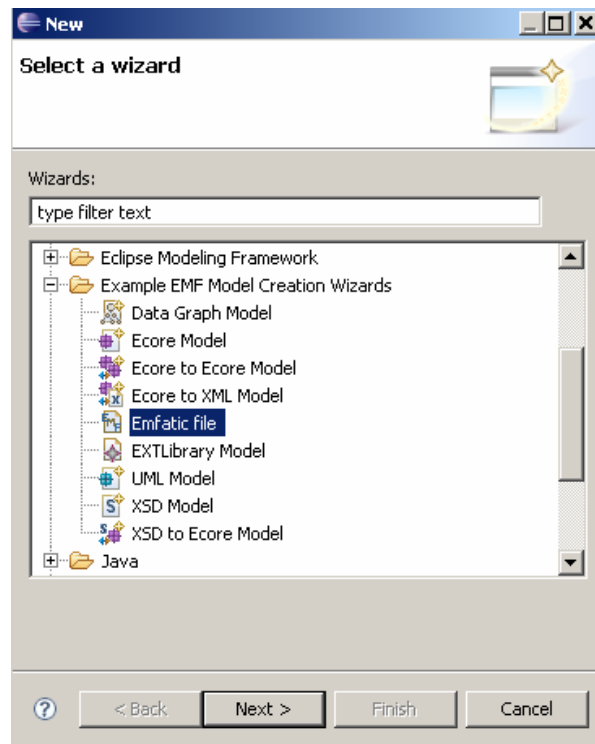
Navigable hyperlinks (from usages of classifiers and type variables to their declarations)

Hyperlinks are very handy for example to follow an eOpposite to its declaration (alternatively, letting the mouse cursor rest over an eOpposite displays its definition in a text hover, see below)

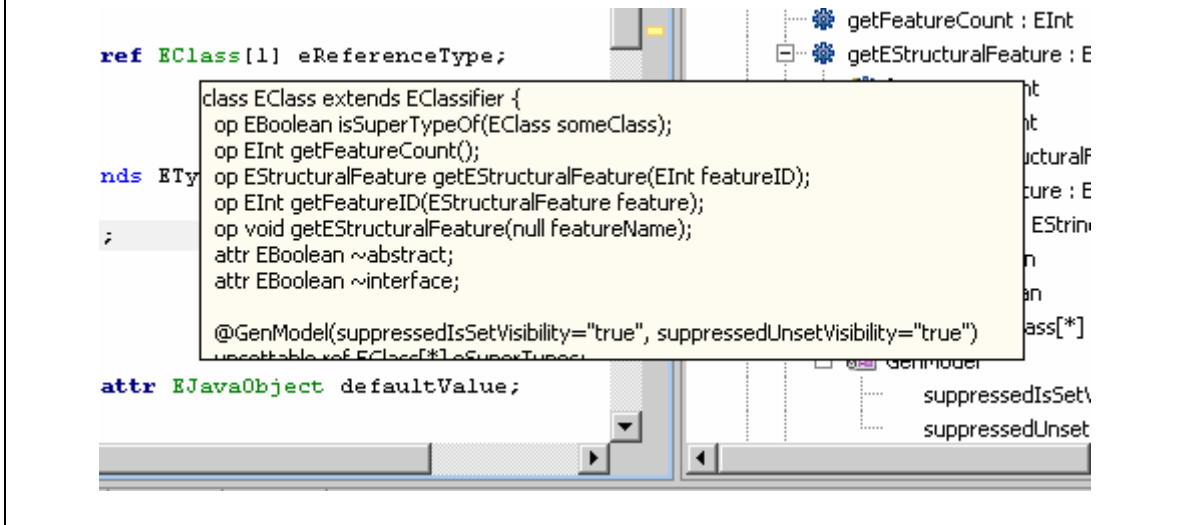
Open declaration is also supported, i.e. pressing F3 (or whatever key binding is in effect for it) moves the cursor to the declaration.

A *File > New > Other ...* wizard is available, which initializes the just created .emf file to a syntactically correct state:

```
@namespace(  
    uri="http://a.b.c/x/y/Z",  
    prefix="p")  
package top;
```

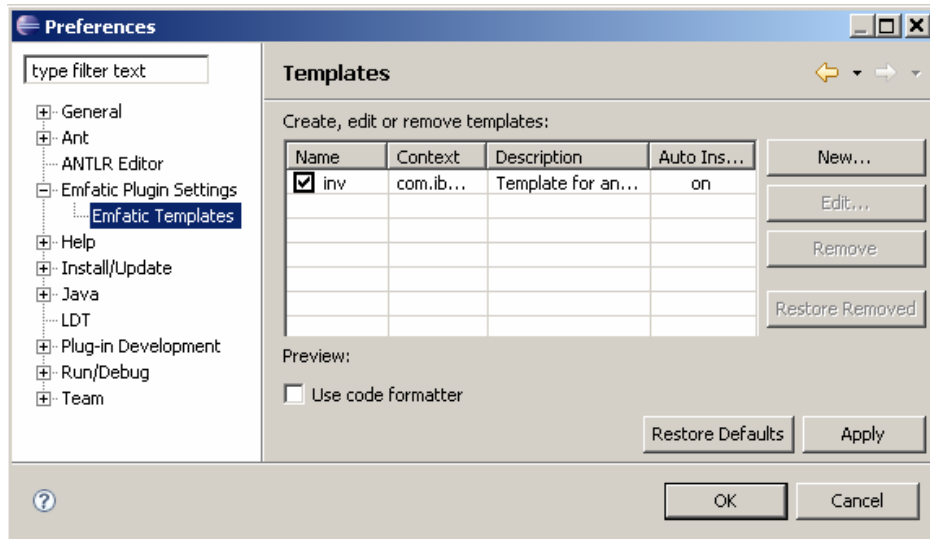


Hovers for usages of classifiers, type parameter variables, and eOpposite references: the definition is shown.



<pre>class c {     }</pre>	<p>AutoEdits</p> <p><b>SmartBrace:</b> typing an opening brace automatically adds its closing brace and leaves the cursor indented on its own line (end result shown left)</p> <p>Besides that strategy the usual ones:</p> <ul style="list-style-type: none"><li>- DefaultIndentLine</li><li>- Closing quote for opening quote</li></ul>
----------------------------	---

Preference pages, to control AutoEdits and templates (see example below)



A template for each kind of annotation saves typing. In the example, the template definition for adding OCL invariants is shown (the error marker is transient, once the template is inserted it disappears)

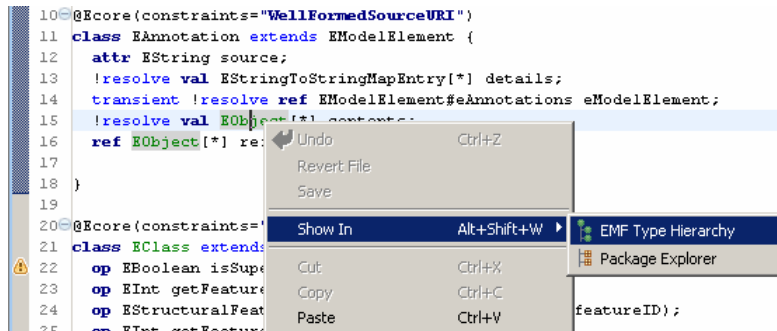


after choosing the template above

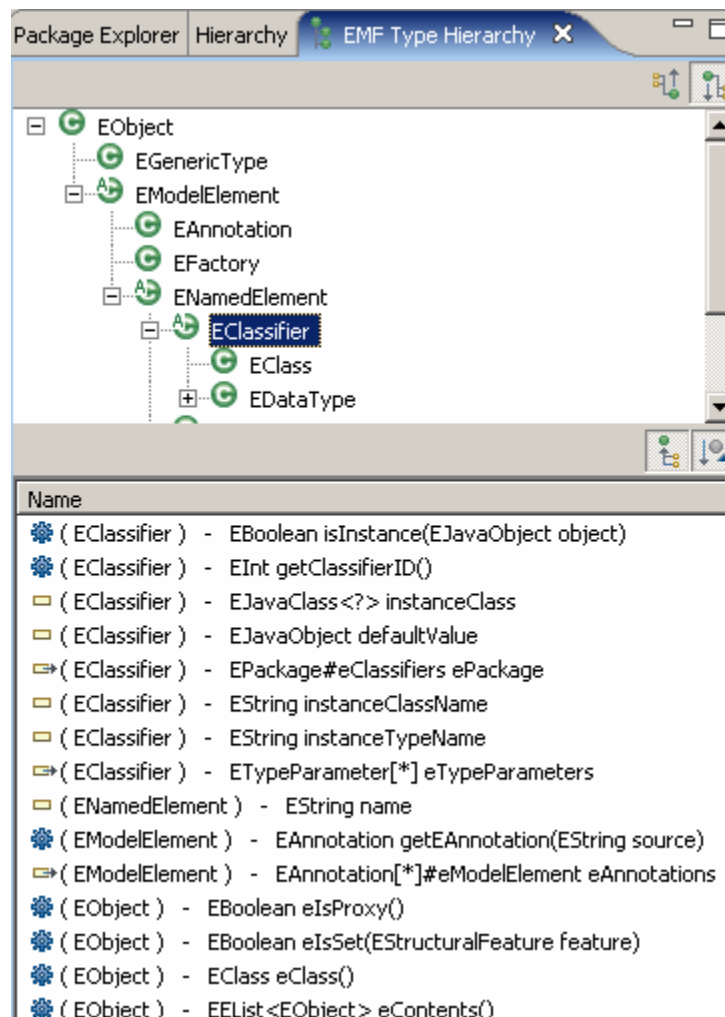


## Show in EMF Type Hierarchy

“Show in EMF Type Hierarchy” can be activated from the editor and from the outline view



The supertype and subtype relationships can be followed, with a list of (inherited) class members displayed on the lower pane. Double-clicking on any of these elements navigates in the editor to its declaration.



## ***ToDo List, Part 1 (priority-ranked)***

- a) content assist beyond templates for annotations, i.e. write an additional ContentProcessor to deliver context proposals for things such as:
  - visible type params and declared types,
    - in a type argument position,
    - in a parameter type position,
    - in an operation result type (together with void),
    - in a supertype position,
    - in an exception type position
  - and where the type of a ref or val is declared
  - visible type params whose first bound is an EDataType and visible EDataType where the type of an attr is declared
- b) Right-click menu > References, or Control-Shift-G

## ***ToDo Part 2***

Some refactorings could be implemented with the infrastructure in place, namely,

- a) Rename a classifier declaration, and usages get renamed
- b) Move a classifier declaration to another package, and usages get rewritten

Given that an AST is available, checking in advance that they leave the resulting AST in a well-formed state is possible (using EcoreValidator). Yet looking into the future, once OCL annotations are also present in an .emf document, what would happen? The implementation of these and other refactorings (developed for Ecore alone) would not be aware about them.

## ***Possibilities for future work***

Many extensions are possible. An obvious one is adding Emfatic-level shorthand annotations for OCL expressions. To the Emfatic editor they are just uninterpreted strings, yet the AST-building background process can trigger the MDT OCL infrastructure for parsing and validation, thus allowing for feedback to the user.

Another possibility that would rely on previous work at STS is support for Java Persistence (JSR-220). Assuming that EMF-level annotations for persistence were given alongside the class model (in the .emf document edited by Emfatic) then a generation of Java-level annotations for persistence could be carried out automatically by an Emfatic extension. The resulting code runs with an EMF-aware persistence runtime, e.g. Teneo.

The third possibility we'll mention are hints for diagram layout specified along text, with an accompanying visualization (as a second tabbed page in MultiPageEditor), following the ideas of

Diomidis Spinellis. On the declarative specification of models. IEEE Software, 20(2):94–96, March/April 2003

[http://www.spinellis.gr/pubs/jrnl/2003-IEEE\\_SW-umlgraph/html/article.html](http://www.spinellis.gr/pubs/jrnl/2003-IEEE_SW-umlgraph/html/article.html)

A tool exemplifying the proposed functionality is UMLGraph:

<http://www.spinellis.gr/sw/umlgraph/>

Martin Fowler on this topic,

<http://martinfowler.com/bliki/UmlSketchingTools.html>

Another, layout only tool, whose syntax for layout hints we're following is

MetaUML: <http://metauml.sourceforge.net/>

### Sample new productions in the grammar

