Agile Model Editing in EMF using Executable Metamodel Annotations

Dimitrios Kolovos, Richard Paige, and Fiona Polack

{dkolovos, paige, fiona}@cs.york.ac.uk

Department of Computer Science
The University of York

Introduction

- Domain Specific Languages (DSL)
  - DSLs are increasingly used in MDD
  - The abstract syntax of the DSL is the first and most important artefact
    - Based on it, concrete syntaxes and model management operations are defined
- Eclipse Modeling Framework (EMF)
  - EMF is the most widely-used open source modelling framework
  - In EMF, the abstract syntax of a DSL is defined using ECore (a variant of MOF 2.0)

The Need for an Agile Model Editing Approach

- Providing prototype model editing tool-support early in the process enables the engineers to locate problems in the abstract syntax by experimentation
- In research activities, DSLs are only prototypes and thus, not much effort must be spent to provide editing tool-support
- Model engineers typically work with a number of DSLs and they cannot spend too much effort in implementing and maintaining a separate editor for each DSL

Research Question

- Is there a technique that requires little customization and maintenance effort but still delivers usable prototype editors?
  - to make prototyping DSLs easier
  - to facilitate the establishment of Towers of DSLs (is that “a good thing”? ☺)

Model Editing in the EMF world

- Text-based Editors
  - E.g. using XText, TCS
- Diagrammatical Editors
  - E.g. using GMF, Topcased
- Tree-based Editors
  - Generated tree-based editors
  - Built-in reflective editor

Text-based Editors

- Tools like XText and TCS can be used to specify textual syntaxes for DSLs
- **Pros**
  - Users can edit models in a compact textual syntax
  - Such tools also support model-to-text serialization
- **Cons**
  - Tools of this kind require significant expertise with EBNF-like notations
  - Identifying and correcting errors in grammars is often challenging
  - Involves generating and maintaining additional artefacts (e.g. parsers)

Diagrammatical Editors

- Tools like GMF and Topcased enable engineers to define visual syntaxes for DSLs
- **Pros**
  - Enable engineers to edit models using visual tools
  - Enable non-technical stakeholders to provide feedback early in the process
- **Cons**
  - To achieve generality such tools are particularly complex and require significant expertise
  - Requires generation and maintenance of additional artifacts (i.e. plug-ins)

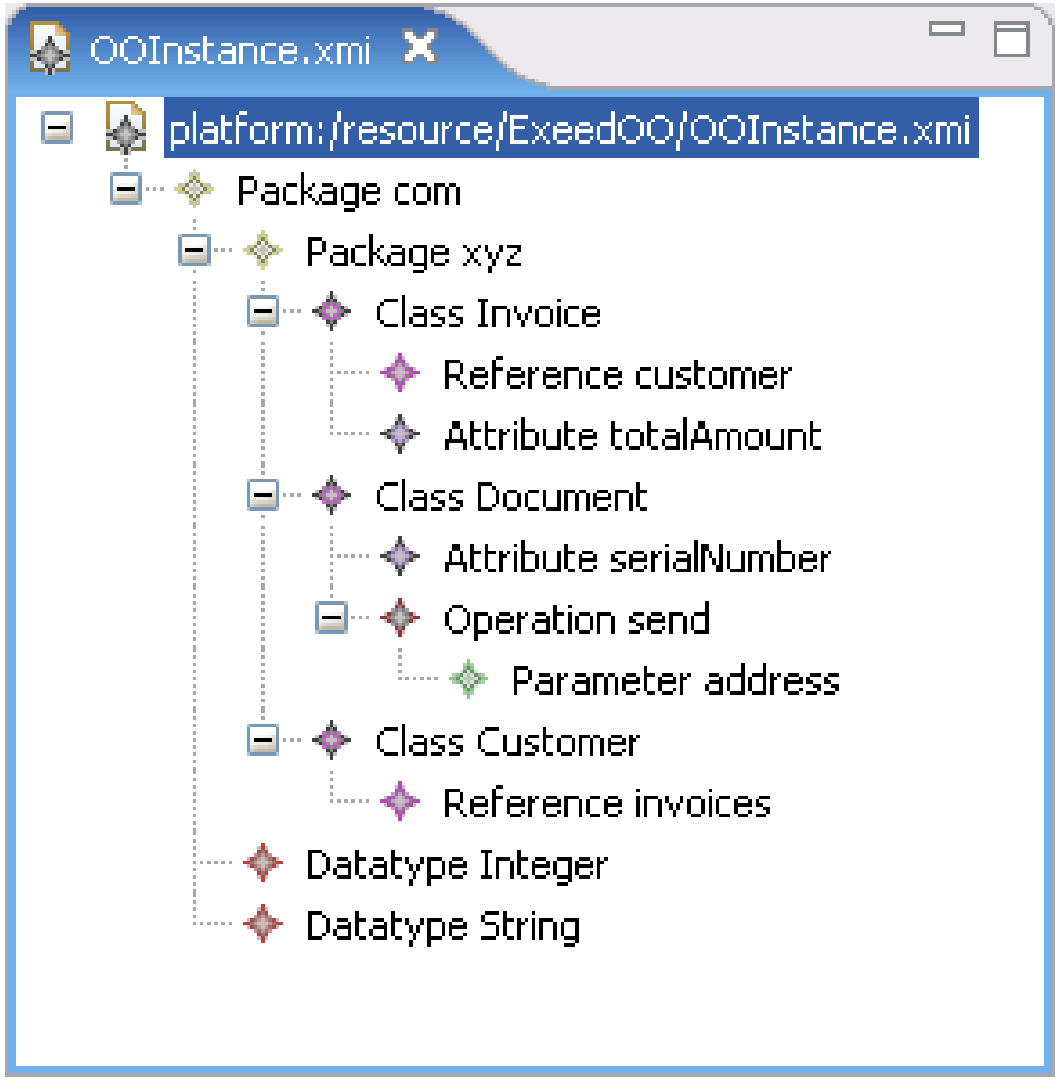
Generated Tree-based Editors

- EMF provides built-in tools for generating a basic tree-based editor and then customizing its appearance using Java
- **Pros**
  - Java can be used to customize many aspects of the generated editor
- **Cons**
  - Requires generation and maintenance of additional artefacts (I.e. plugins)
  - To customize, engineers must be familiar with the (non-trivial) underlying EMF.Edit framework

Reflective Tree-based Editor

- EMF provides a built-in reflective editor that can be used to edit models of arbitrary DSLs
- **Pros**
  - Does not require generating or maintaining additional artefacts
- **Cons**
  - Model elements are represented on the editing tree using very simple labels and practically indistinguishable icons
  - No customization is possible

The Reflective Editor in Action



Comparison of Editing Approaches

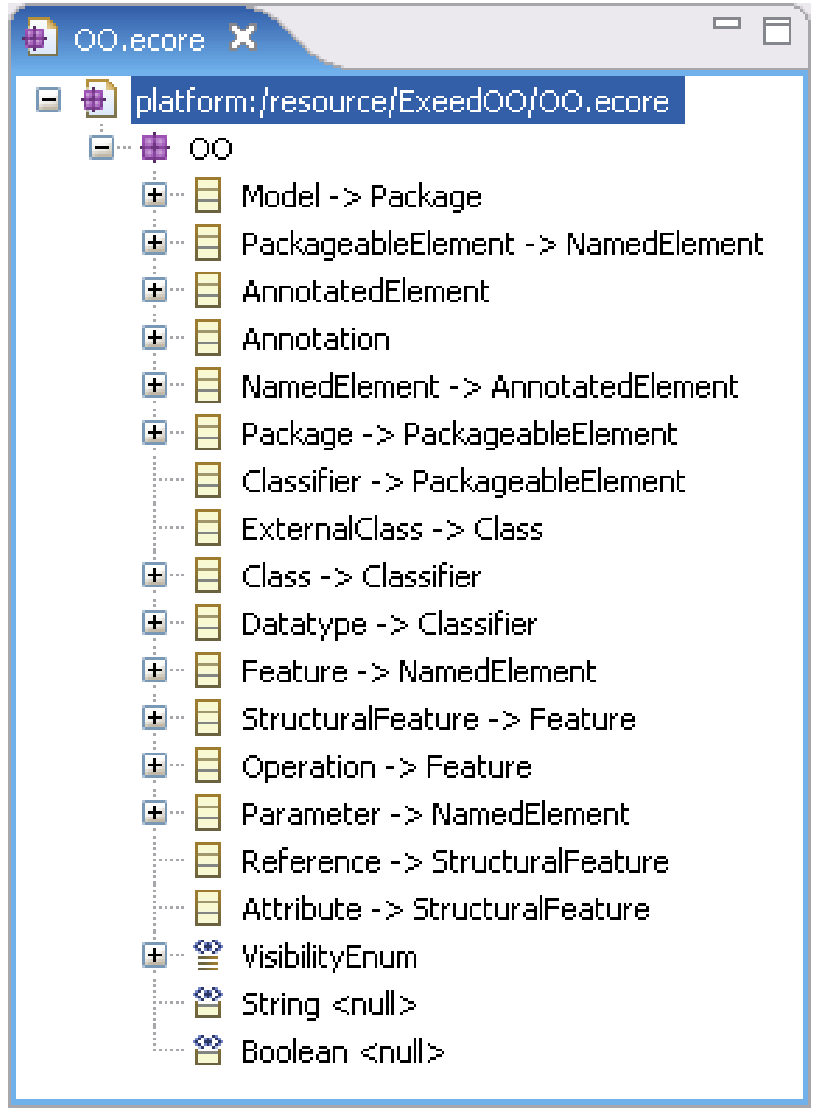
- Text-based and diagrammatical editors require significant effort and expertise to implement
- Except for the reflective approach, all others require generation and maintenance of additional artefacts
- The reflective editor is the most agile one (but it cannot be customized and is not really usable as-is)

An Agile and Usable Editing Approach

- Our aim is to provide a means of customizing the appearance of the reflective editor without needing to generate and maintain additional artefacts
- We achieved this by enriching the abstract syntax of the DSL with presentation-specific executable annotations
- We have implemented a prototype that realizes this approach: the **EXtended Emf EDitor (Exeed)**

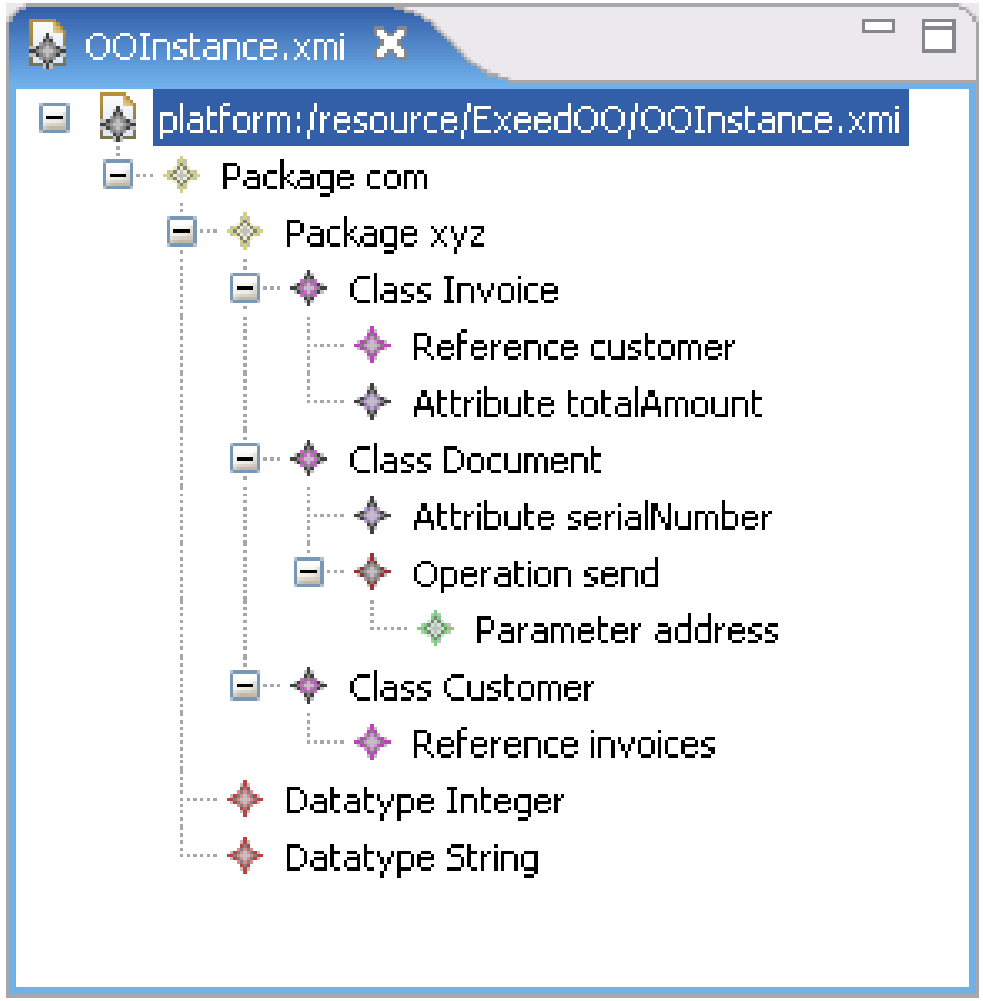
A Motivating Example

- We have designed a prototype DSL for specifying simple Object Oriented designs



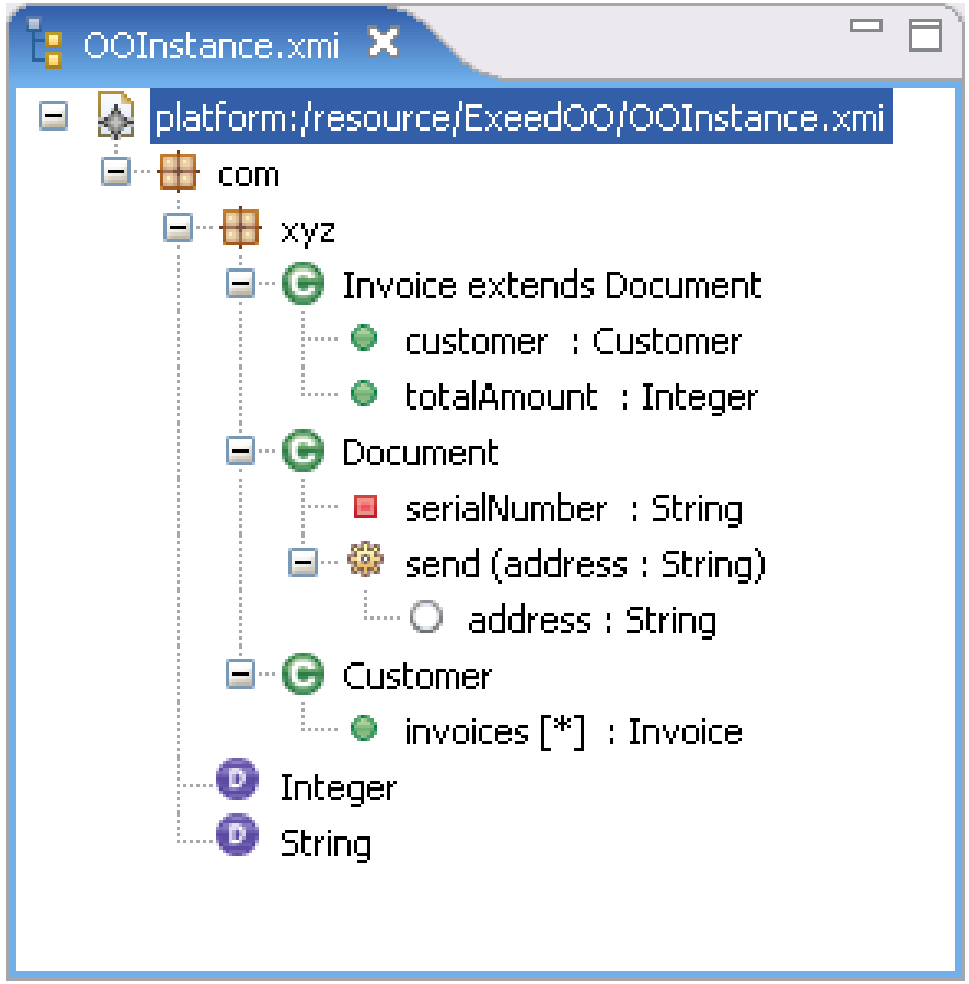
Editing OO Models with the Reflective Editor

- This is the appearance of an OO model in the reflective editor



Preview of the Result

- In the end of this example, the appearance of the same OO model will be like that:



Presentation-Specific Annotations

- Presentation-specific annotations are added to constructs of the ECore metamodel as **`EAnnotationDetail`**s contained in **`EAnnotation`**s named **`exeed`**
- There are two types of annotations
  - Static strings
  - Executable blocks of EOL statements
- EOL is an OCL-based imperative language, part of the Epsilon GMT project

EClass Annotations (1/2)

- **`label`**
  - A block of EOL statements that calculates a label for each instance of the EClass on the editing tree
- **`referenceLabel`**
  - A block of EOL statements that calculates a label for each instance of the EClass when referenced by another object (e.g. in the properties view)

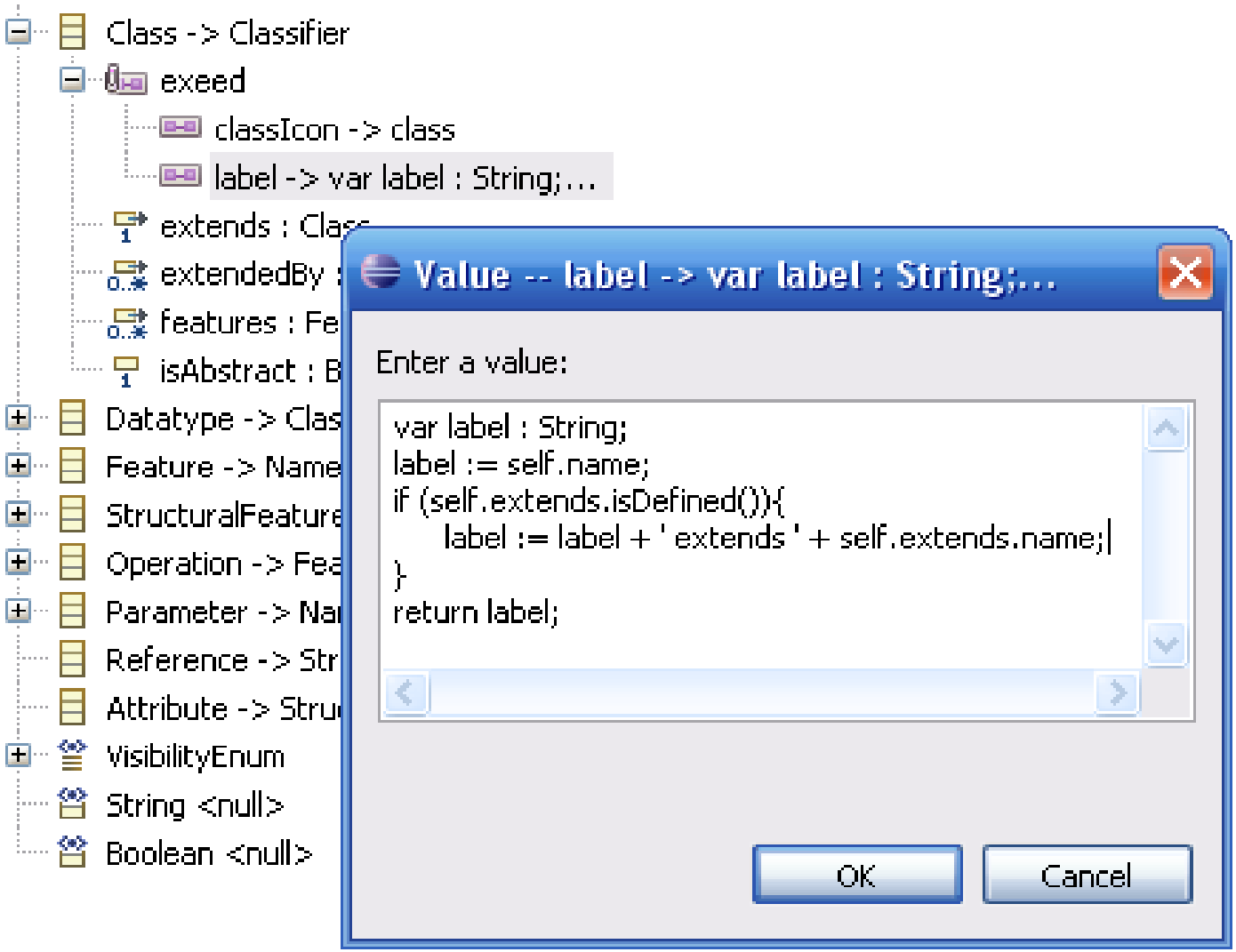
EClass Annotations (2/2)

- **`icon`**
  - A block of EOL statements that calculates an icon for each instance of the Eclass
- **`classIcon`**
  - A string that specifies the icon that represents the EClass (in context menus or where an **`icon`** annotation is not provided)

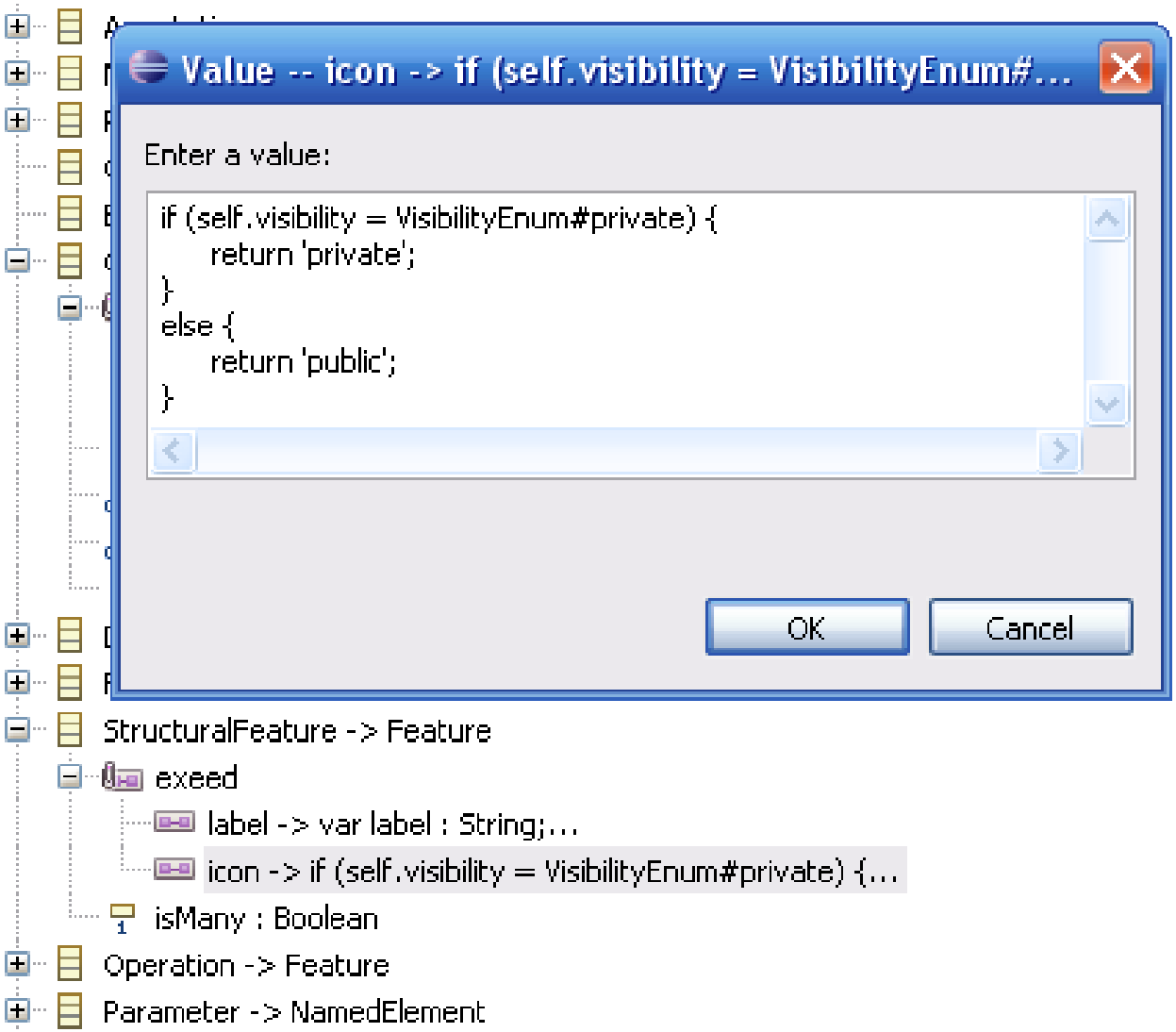
EStructuralFeature Annotations

- **`featureLabel`**
  - A string that specifies a human-understandable label for the feature (in context menus, property view etc)
- **`multiline`**
  - A true/false value that specifies if the value of the EAttribute must be edited in a single-line or a multi-line mode (applies to **`EAttribute`**s)

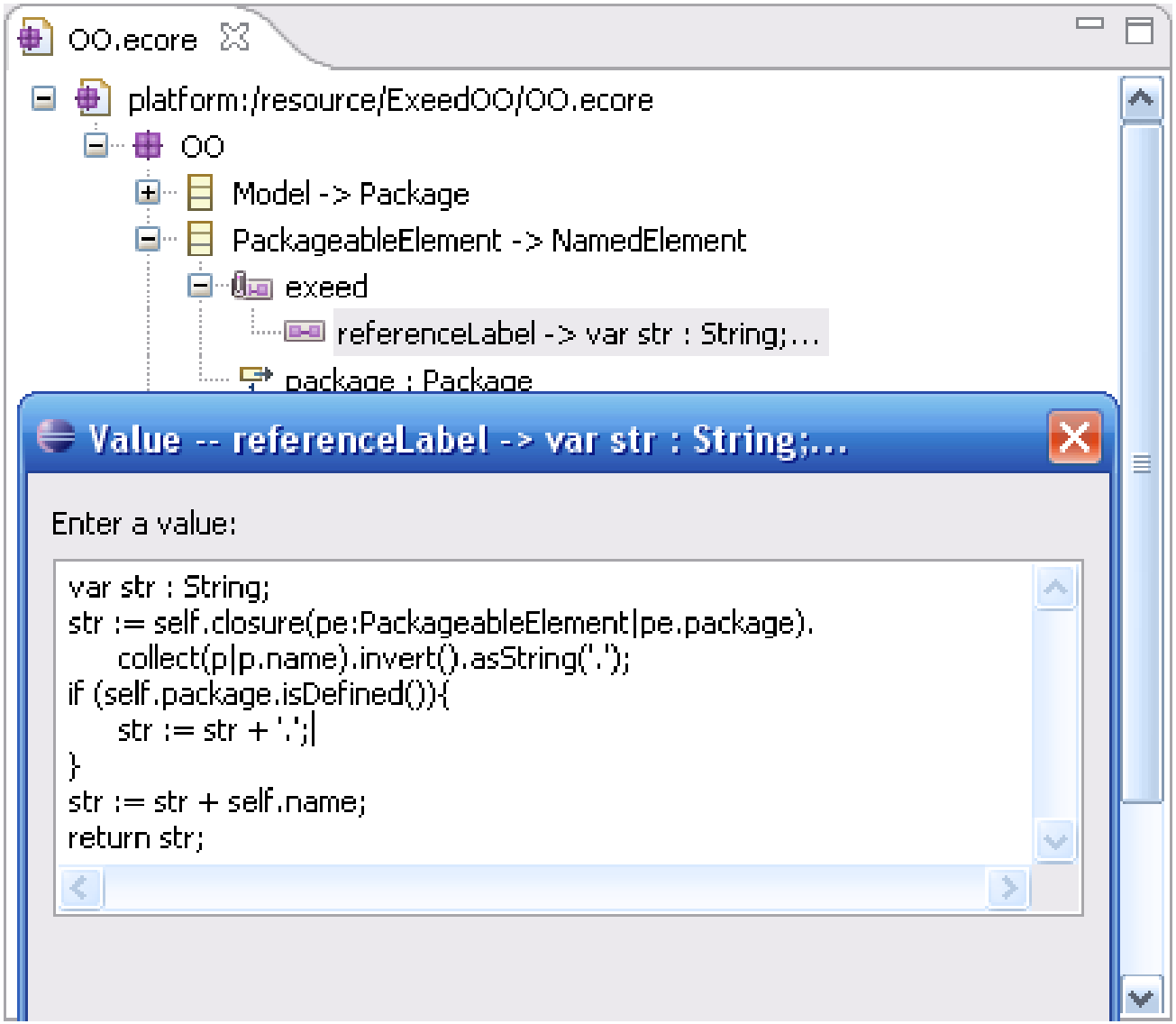
A Label Annotation Example



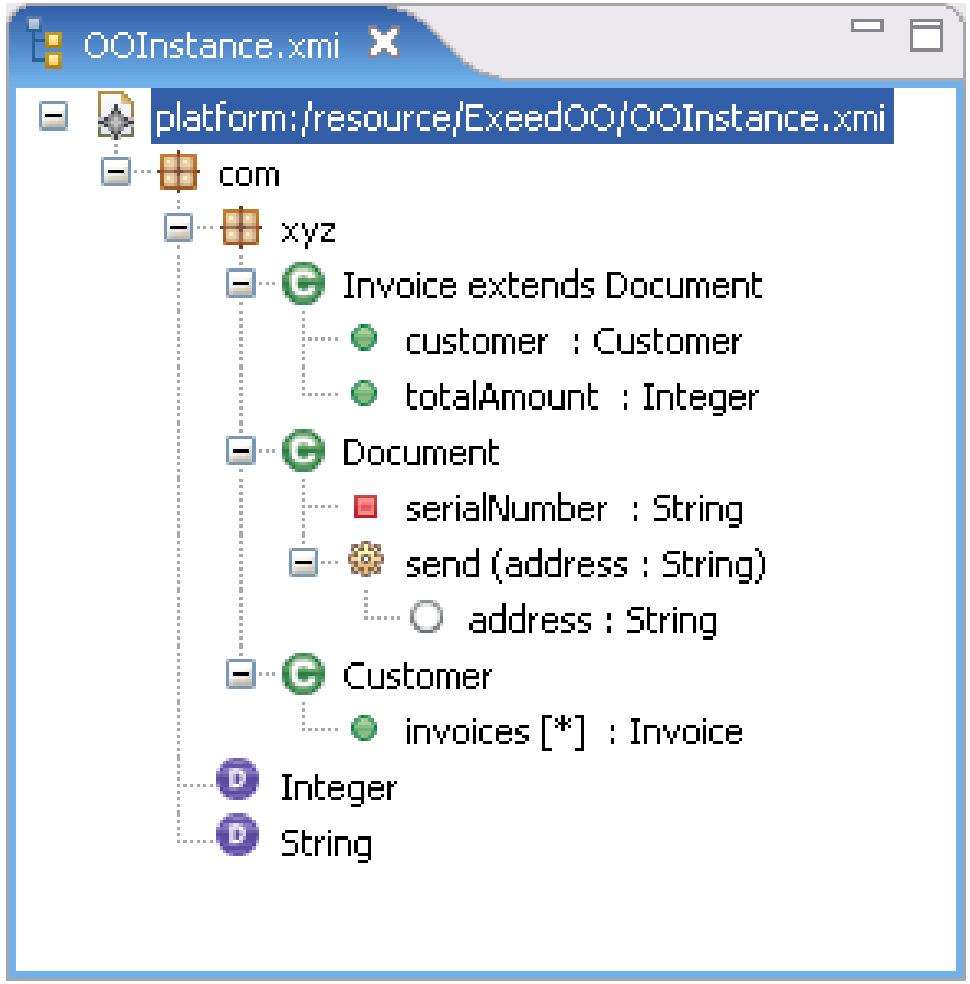
An Icon Annotation Example



A ReferenceLabel Annotation Example



The Result


OOInstance.xmi
platform:/resource/ExeedOO/OOInstance.xmi
com
xyz
Invoice extends Document
customer : Customer
totalAmount : Integer
Document
serialNumber : String
send (address : String)
address : String
Customer
invoices [*] : Invoice
Integer
String

Conclusions

- Executable metamodel annotations can be used to add presentation-specific information to the abstract syntax of a DSL
- The approach is particularly agile as it does not require generation and maintenance of language-specific editors
- We consider our approach to be particularly helpful in the initial stages of DSL development and in cases of prototype languages constructed for research purposes

Further Work

- We are using this approach for providing editors for DSLs we define for our research and have already identified new annotations that can further customize the appearance of the editor
- When support for dynamic EMF is provided in GMF*, we shall attempt to apply a similar approach in order to support definition of simple diagrammatical syntaxes

* https://bugs.eclipse.org/bugs/show_bug.cgi?id=150177

Acknowledgements

This work was supported by the ModelPlex EU IST project

www.modelplex-ist.org

Resources

- Epsilon (including Exeed) can be obtained at:
  - http://www.eclipse.org/gmt/epsilon/download.php
- The OO DSL example can be downloaded at:
  - http://www.eclipse.org/gmt/epsilon/doc/examples.php