
# EMF: Eclipse Modeling Framework

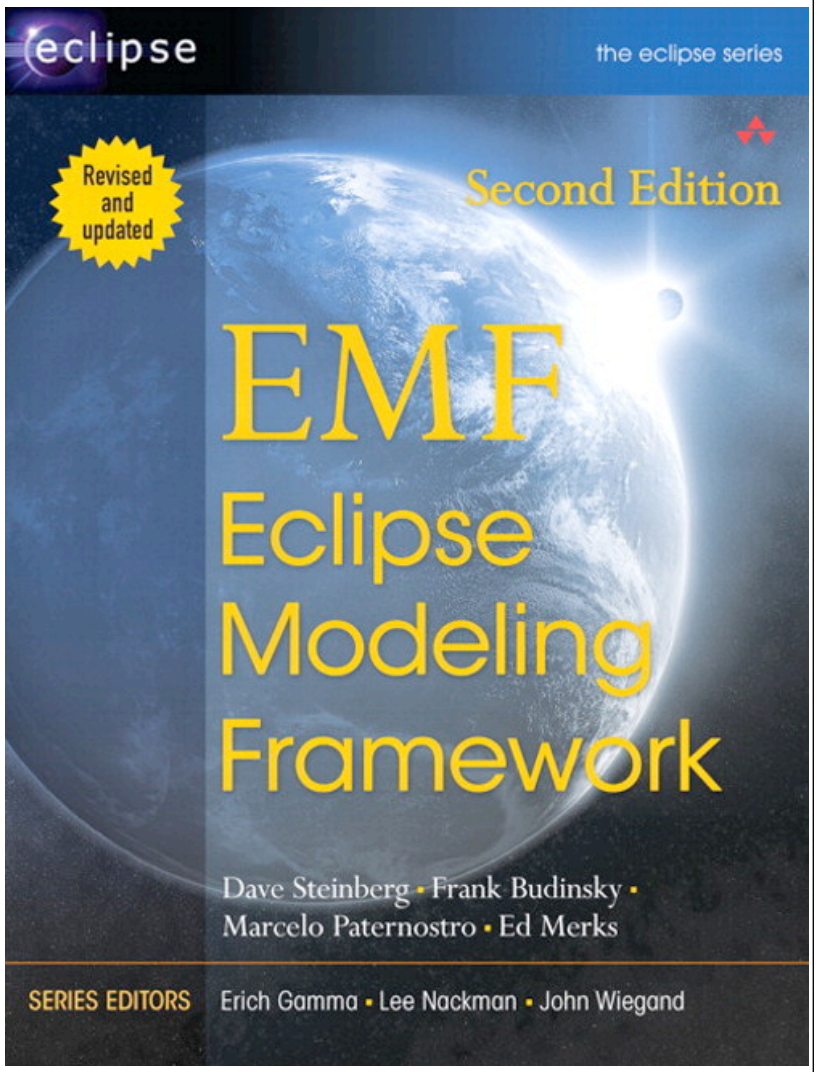
Prof. Dr. Ralf Lämmel
Dipl.-Math. Vadim Zaytsev
Universität Koblenz-Landau
Software Languages Team

# EMF — what is it?

- Modeling (think of UML)
- Interoperability (think of XMI)
- Editing tool support (think Eclipse)
- Code generation (think of MDA)

# Detailed reading

- The EMF book by Steinberg, Budinsky, Paternostro & Merks
- http://help.eclipse.org
- http://wiki.eclipse.org
- Gerber&Raymond, MOF to EMF: there and back again, OOPSLA'03,p.60-64



# Relation between EMF & UML

- UML concepts:
  - classes, members, relationships, …
- UML also covers dynamic aspects

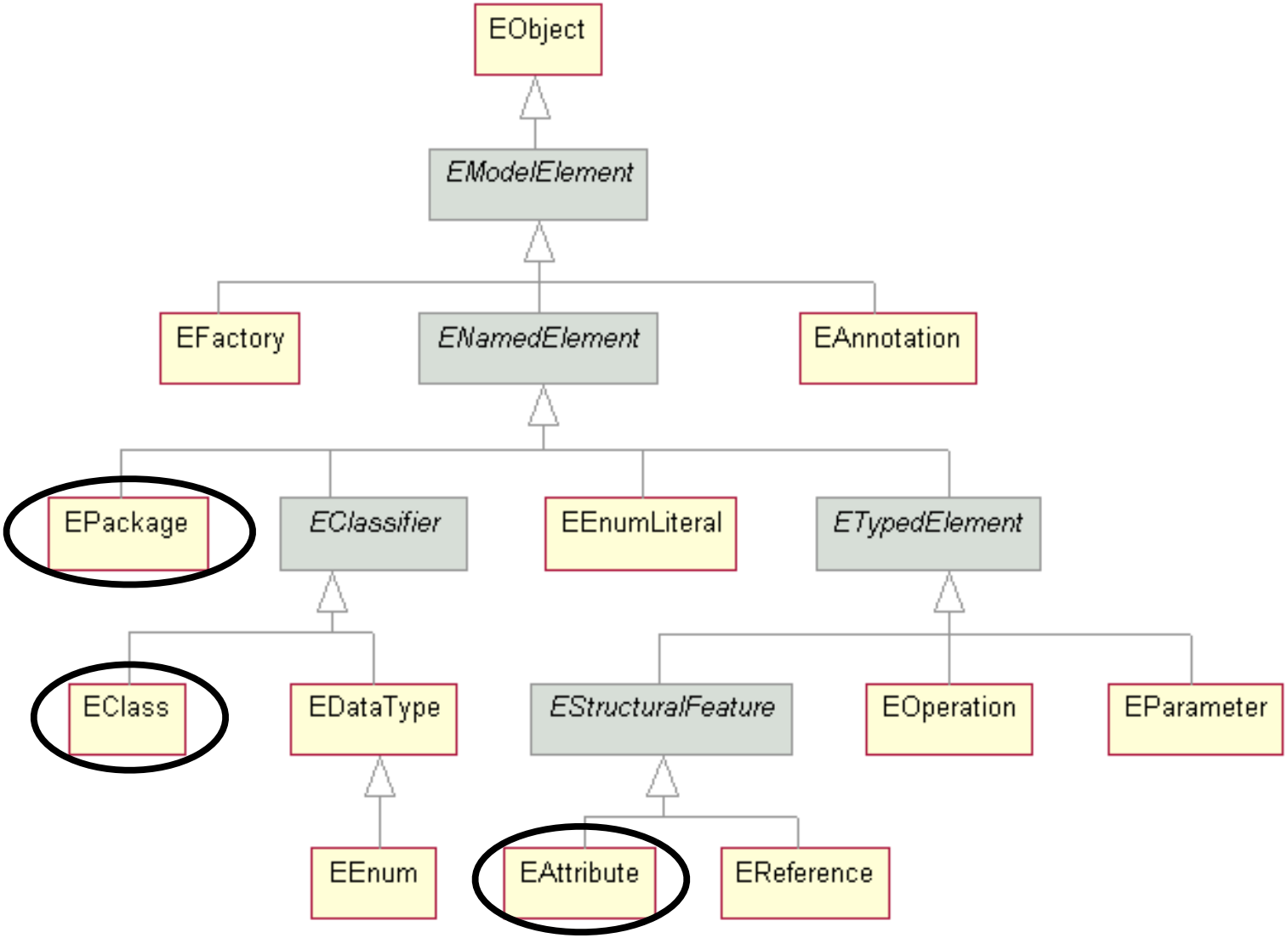
- EMF is a meta-model for a subset of UML
- (EMF is also a meta-model for EMF)

# Relation between EMF & MOF

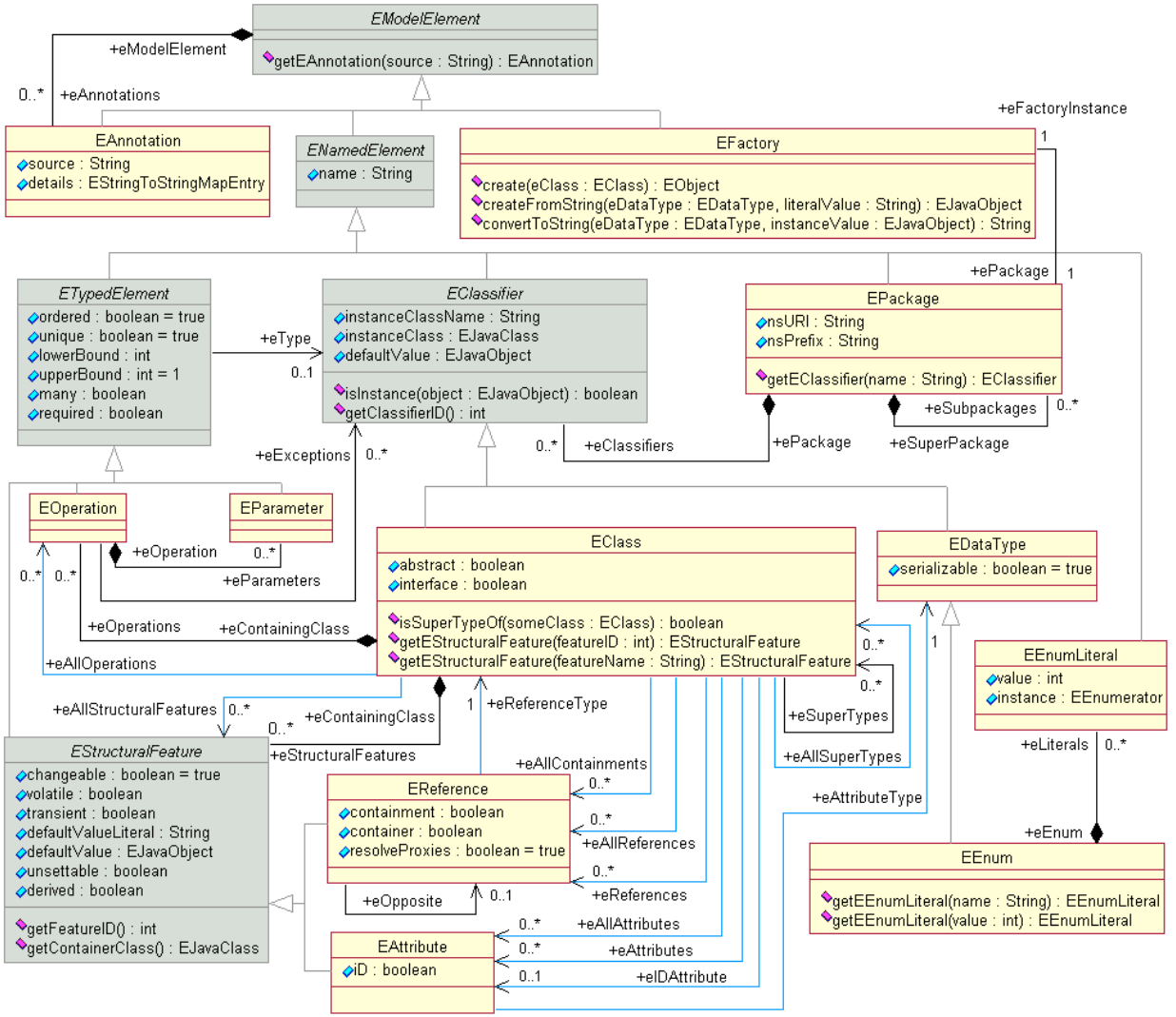
- MOF: Meta-Object Facility
  - OMG standard for MDE
  - EMOF (Essential), CMOF (Complete)
  - also ISO/IEC 19502:2005 standard

- In EMF Ecore is aligned with EMOF

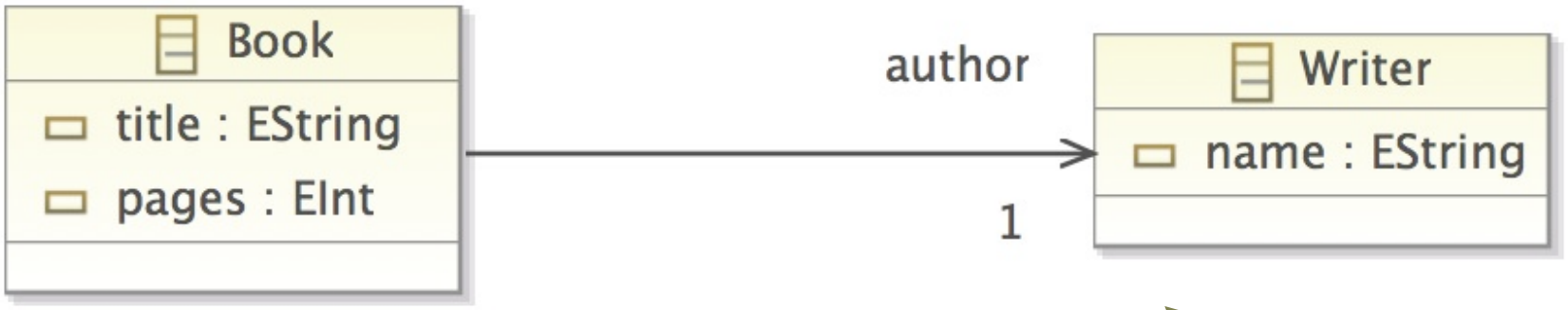
# Ecore components

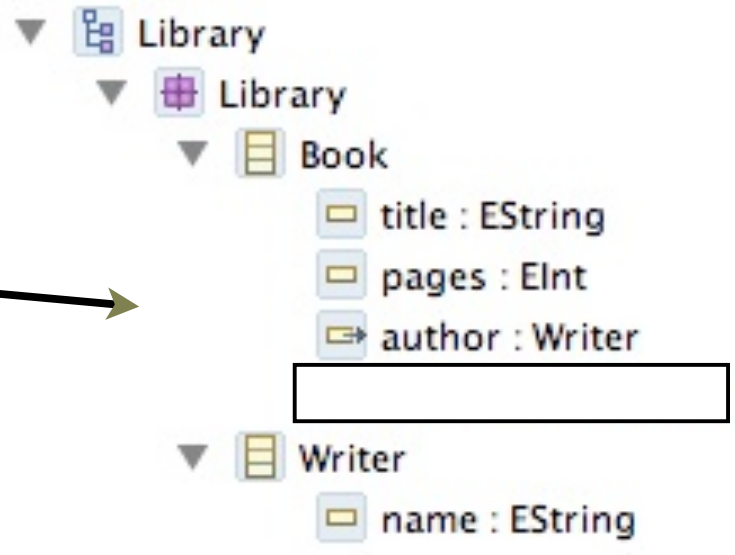


# Ecore components hierarchy



# UML → EMF




Ecore Model Editor

Ecore Diagram (=UML class diagam)

# UML & EMF

- Conceptually a subset
  - no State Diagrams
  - no Collaboration Diagrams
- No graphical designer [needed]
- EMF does not substitute all OOA/D tools

# XML Metadata Interchange

- XMI is XML standard for meta-data interchange
  - If the meta-model can be expressed in MOF, its meta-data can be serialised as XMI
- Ecore models are stored/shared in XMI
- Ecore/UML diagrams are in XMI
- EMF generator models are in XMI

# XMI → EMF

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore" name="library"
    nsURI="http:///library.ecore" nsPrefix="library">
  <eClassifiers xsi:type="ecore:EClass" name="Book">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="title"
     eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="pages"
     eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="author"
     eType="#//Writer"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Writer">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name"
     eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
</ecore:EPackage>
```

# XMI → EMF

```
<?xml version="1.0" encoding="UTF-8"?>
<genmodel:GenModel xmi:version="2.0"
    xmlns:xmi="http://www.omg.org/XMI" xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
    xmlns:genmodel="http://www.eclipse.org/emf/2002/GenModel" modelDirectory="/Library/src"
    modelPluginID="MusicLibrary" modelName="Library" importerID="org.eclipse.emf.importer.java"
    complianceLevel="5.0" copyrightFields="false">
  <foreignModel>@model</foreignModel>
  <genPackages prefix="Library" disposableProviderFactory="true" ecorePackage="library.ecore#/">
    <genClasses ecoreClass="library.ecore#//Book">
      <genFeatures createChild="false"
                ecoreFeature="ecore:EAttribute library.ecore#//Book/title"/>
      <genFeatures createChild="false"
                ecoreFeature="ecore:EAttribute library.ecore#//Book/pages"/>
      <genFeatures notify="false" createChild="false" propertySortChoices="true"
                ecoreFeature="ecore:EReference library.ecore#//Book/author"/>
    </genClasses>
    <genClasses ecoreClass="library.ecore#//Writer">
      <genFeatures createChild="false"
                ecoreFeature="ecore:EAttribute library.ecore#//Writer/name"/>
    </genClasses>
  </genPackages>
</genmodel:GenModel>
```

# XMI & EMF

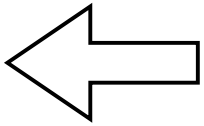
- EMF meta-model is EMF
  - XMI is a default serialised form

- Write manually (in any XML editor)
- Generate with other tools
  - e.g. Rational Rose
  - less specific than .mdl

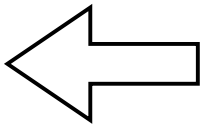
# EMF so far

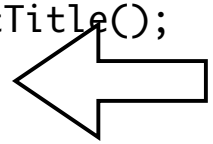
- UML models the class hierarchy
- EMF models UML
- EMF Ecore relates to OMG EMOF
- EMF generator model is for binding
  - what code needs to be generated and how

# Java → EMF

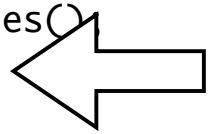


```
import org.eclipse.emf.ecore.EObject;
package lib;

/**
 * @model
 */
public interface Book extends EObject
{
    /**
     * @model
     */
    String getTitle();

    /**
     * @model
     */
    int getPages();
}
```





```
/**
 * Sets the value of the '{@link library.Book#getTitle <em>Title</em>}' attribute.
 * @param value the new value of the '<em>Title</em>' attribute.
 * @see #getTitle()
 * @generated
 */
void setTitle(String value);
```



```
/**
 * Sets the value of the '{@link library.Book#getPages <em>Pages</em>}' attribute.
 * @param value the new value of the '<em>Pages</em>' attribute.
 * @see #getPages()
 * @generated
 */
void setPages(int value);
```

# Java & EMF

- Start with the code
- Annotate model elements
- Generate the model
- Generate the code
  - model
  - implementation
- Continue development
- Regenerate when necesary

# Eclipse demonstration

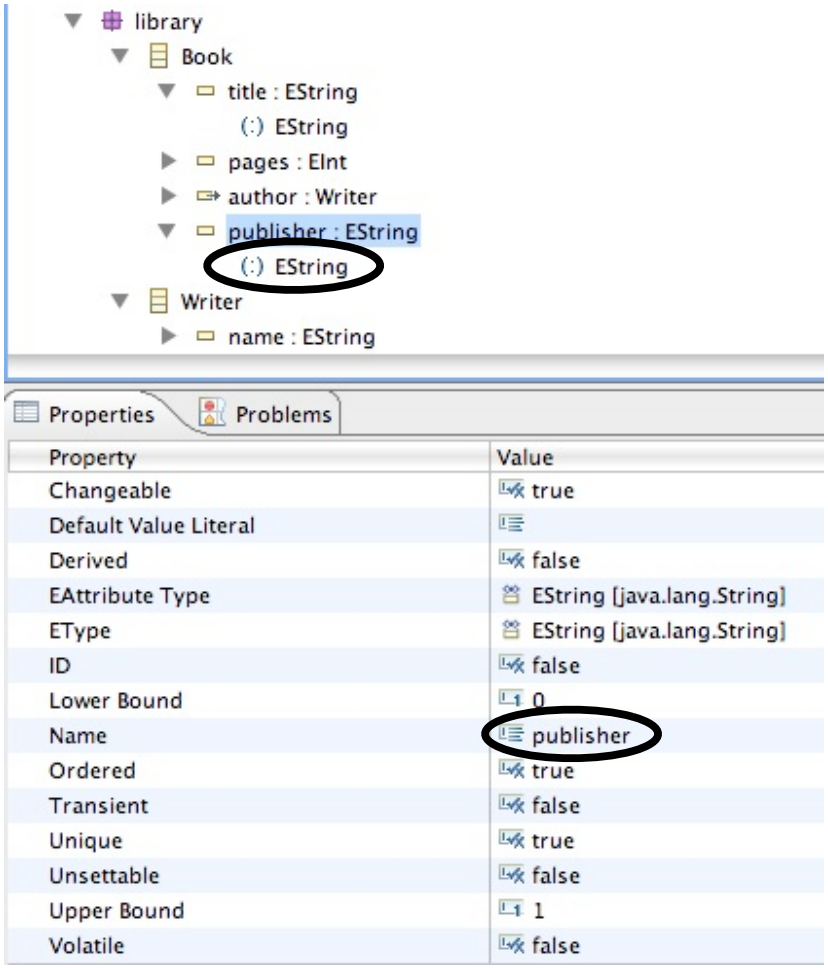
eclipse-modeling-galileo-incubation-macosx-cocoa.tar.gz

# Changing the Java code

```
        . . .

/**
 * @model
 */
public interface Book
{
        . . .

    /**
     * @model
     */
    String getPublisher();

        . . .

}
```

- Edit the source code
- Add a getter
- Reload the model
  - based on annotated Java
- Generate model code
- Modify the setter (if needed)

# Changing the Ecore



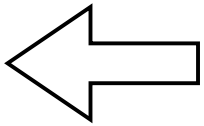

- Open in Sample Ecore model editor
- Add EAnnotation child
  - add EGeneric Type
- Reload the model
  - based on Ecore model
- Generate model code
- Modify the getter and the setter (if needed)

# Changed Ecore → Java

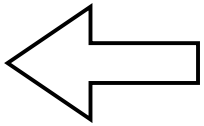
```
    /**
 * Returns the value of the '<em><b>Publisher</b></em>' attribute.
 * <!-- begin-user-doc -->
 * <p>
 * If the meaning of the '<em>Publisher</em>' attribute isn't clear,
 * there really should be more of a description here...
 * </p>
 * <!-- end-user-doc -->
 * @return the value of the '<em>Publisher</em>' attribute.
 * @see #setPublisher(String)
 * @see library.LibraryPackage#getBook_Publisher()
 * @model
 * @generated
 */
String getPublisher();

/**
 * Sets the value of the '{@link library.Book#getPublisher <em>Publisher</em>}' attribute.
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @param value the new value of the '<em>Publisher</em>' attribute.
 * @see #getPublisher()
 * @generated
 */
void setPublisher(String value);
```

# EMF → implementation

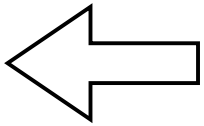
```
     /**
 * @generated
 */
public class BookImpl extends EObjectImpl implements Book {
     . . .
     /**
      * @see #getPublisher()
      * @generated
      * @ordered
      */
     protected static final String PUBLISHER_EDEFAULT = null;
     /**
      * @see #getPublisher()
      * @generated
      * @ordered
      */
     protected String publisher = PUBLISHER_EDEFAULT;
     /**
      * @generated
      */
     public String getPublisher() {
          return publisher;
     }
     /**
      * @generated
      */
     public void setPublisher(String newPublisher) {
          String oldPublisher = publisher;
          publisher = newPublisher;
          if (eNotificationRequired())
           eNotify(new ENotificationImpl(this, Notification.SET, LibraryPackage.BOOK__PUBLISHER, oldPublisher, publisher));
     }

     . . .
}
```



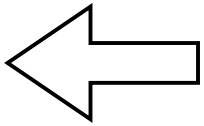
default value defined



default value used



generated getter impl.



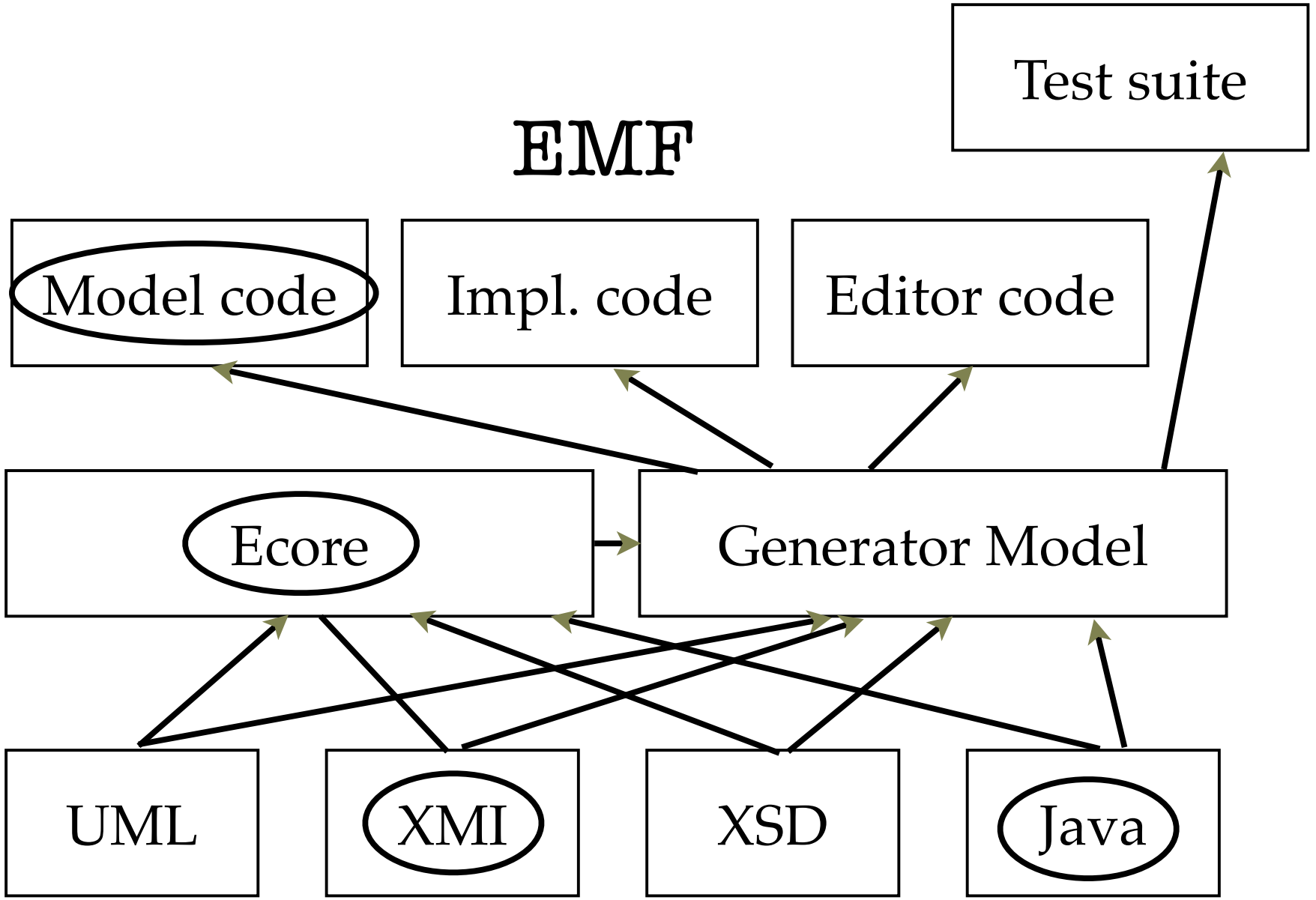
generated setter impl.

# XSD → EMF

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" >
  <xsd:element name="Book">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="title" type="xsd:string" />
        <xsd:element name="pages" type="xsd:integer" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Writer">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
```

# XSD & EMF

- Schema as form of a serialisation
- Think of object instances
- Think of how we want them serialised
- Think of using external standard in your app
- The generated model might be different



# Grammar case study

- Language as a model
- Schema as a grammar
- Editor for trees (instances)

# FL: Factorial Language

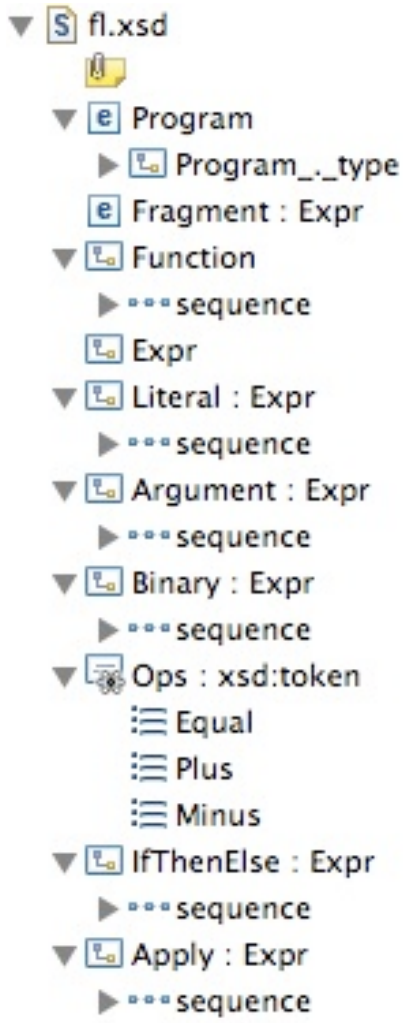
Syntax

```
inc x = (x + 1)
dbl x = (x + x)
mult n m = if (n==0) then 0 else (m + (mult (n - 1) m))
fac n = if (n==0) then 1 else (mult n (fac (n - 1)))
```

XML Schema

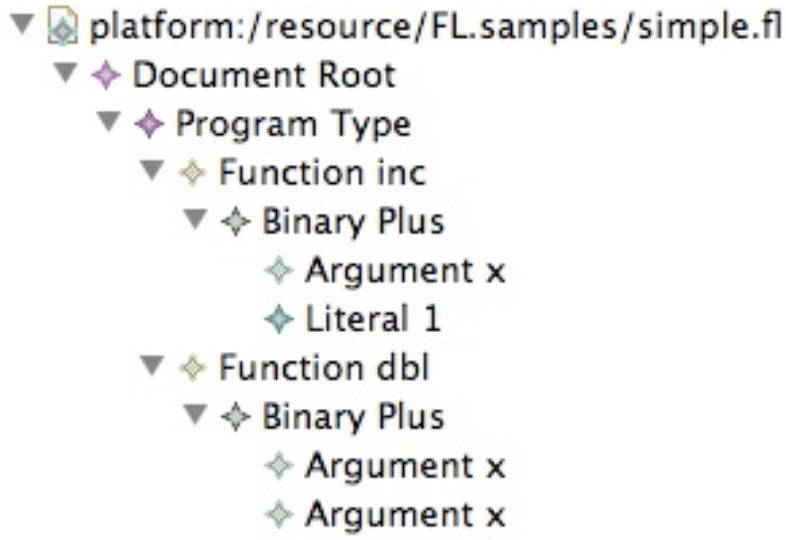
```
<xsd:complexType name="Function">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="arg" type="xsd:string" maxOccurs="unbounded"/>
    <xsd:element name="rhs" type="Expr"/>
  </xsd:sequence>
</xsd:complexType>

. . .
```

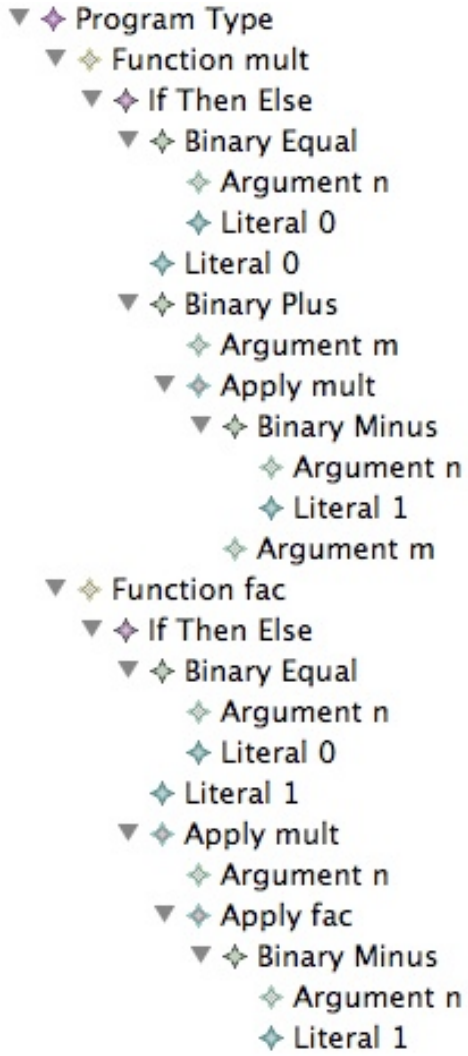
# FL XSD walkthrough




- Start with the schema
- Generate a generator model
  - underlying Ecore
  - xsd2ecore mapping
- Generate code
- Generate editor code
- Run the editor as Eclipse app (run-time workbench)

# Programming in FL!



# Eclipse demonstration

eclipse-modeling-galileo-incubation-macosx-cocoa.tar.gz

# Conclusion

- EMF is a meta-model for UML
- Ecore implements EMOF
- Ecore can be generated from XSD, UML, Java, …
- Generator model defines bindings
- Code is generated from the Ecore model
- Implementation code binding is persistent
- Editor code is generated and can be run as Eclipse