

The EMF: Modeling with Eclipse



metamodel
“I never ~~met a man~~ I didn’t like.”
– Will Rodgers

Modeling – Model Driven Architecture

Quick Summary of MDA:

- Software development architecture proposed by the OMG using open modeling standards
 - UML, MOF, XMI, CWM, etc.
- Desired system is specified as a platform independent model (PIM)
- PIM is transformed into a platform specific model (PSM), i.e., implementation code.

Modeling – On the Way to MDA

Aims of EMF include:

- the extraction of a description's intrinsic “data model” (i.e., entity objects);
- the relating of these modeled concepts directly to their implementations;
- the unification of the system descriptions (i.e., the various models);
- and, the interoperability of EMF-based tools and applications.

Modeling – EMF and Model Unification

Using a data model in a project typically requires (in *some* order):

- a documented representation of the entities and their relationships;
- code for implementing the entities;
- and, a mechanism for persisting the entities.

Of course, the team keeps these artifacts perfectly in sync. **Not!**

Modeling – EMF and Model Unification

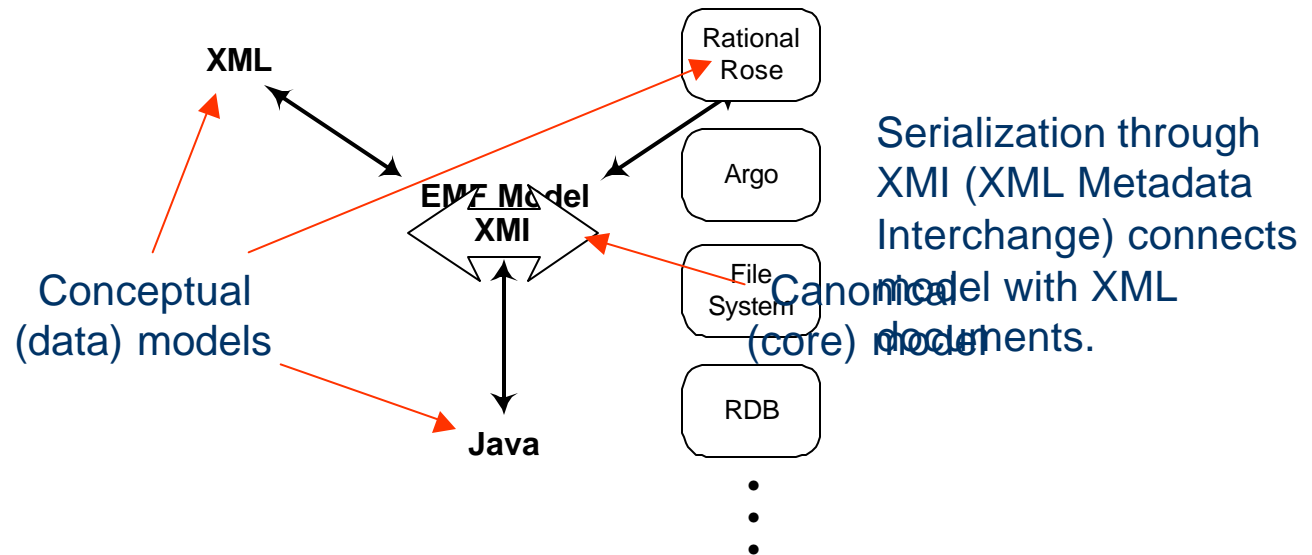
The framework **models** a given data **model** from which it can in turn, generate other equivalent **models**:

- Annotated Java code
- UML class diagrams
- XML schema

The framework also provides viewers for manipulating objects of the data model.

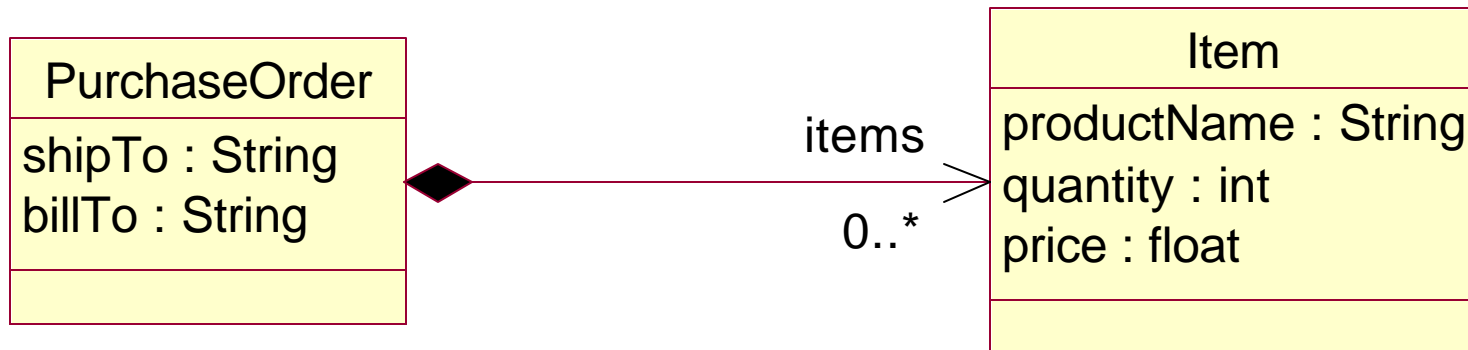
Modeling – EMF and Model Unification

Three views depicting the same data model



Essence of the data model is captured in Ecore – EMF’s metamodel language.

Modeling – EMF Unifying UML



Modeling – EMF Unifying Java

```
/** @model */ // Annotation marks class as part of model definition.
public interface PurchaseOrder {
    /** @model */
    String getShipTo();
    /** @model */
    String getBillTo();
    /** @model type="Item" containment="true" */
    List getItems();
}

/** @model */
public interface Item {
    /** @model */
    String getProductName();
    /** @model */
    int getQuantity();
    /** @model */
    float getPrice();
}
```

Modeling – EMF Unifying XML

```
<xsd:complexType name="PurchaseOrder">
  <xsd:sequence>
    <xsd:element name="shipTo" type="xsd:string"/>
    <xsd:element name="billTo" type="xsd:string"/>
    <xsd:element name="items" type="PO:Item"
      minOccurs="0" maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Item">
  <xsd:sequence>
    <xsd:element name="productName" type="xsd:string"/>
    <xsd:element name="quantity" type="xsd:int"/>
    <xsd:element name="price" type="xsd:float"/>
  </xsd:sequence>
</xsd:complexType>
```

Modeling – UML, MOF and Ecore

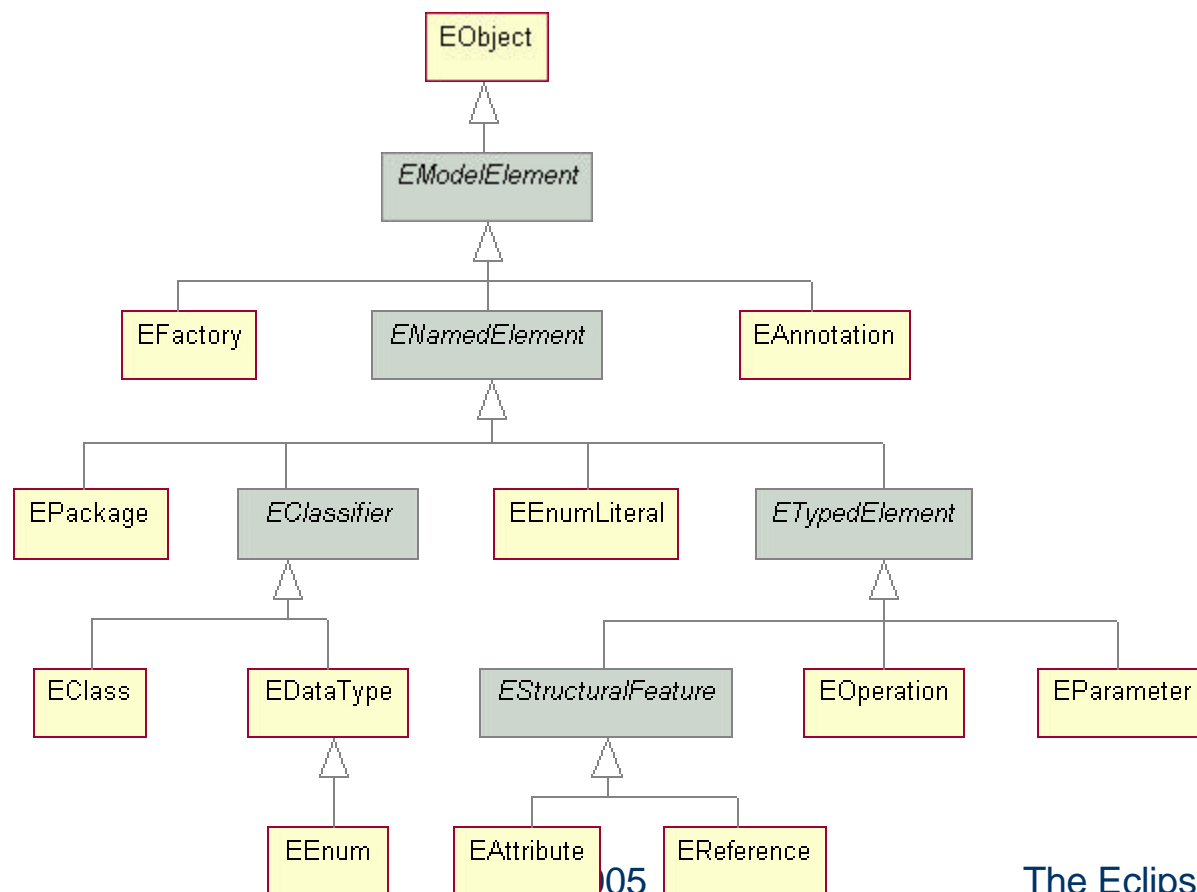
The Meta-Object Facility specializes the class modeling aspects of UML for managing and implementing metadata repositories.

Ecore is a highly efficient Java implementation of a core subset of the MOF API.

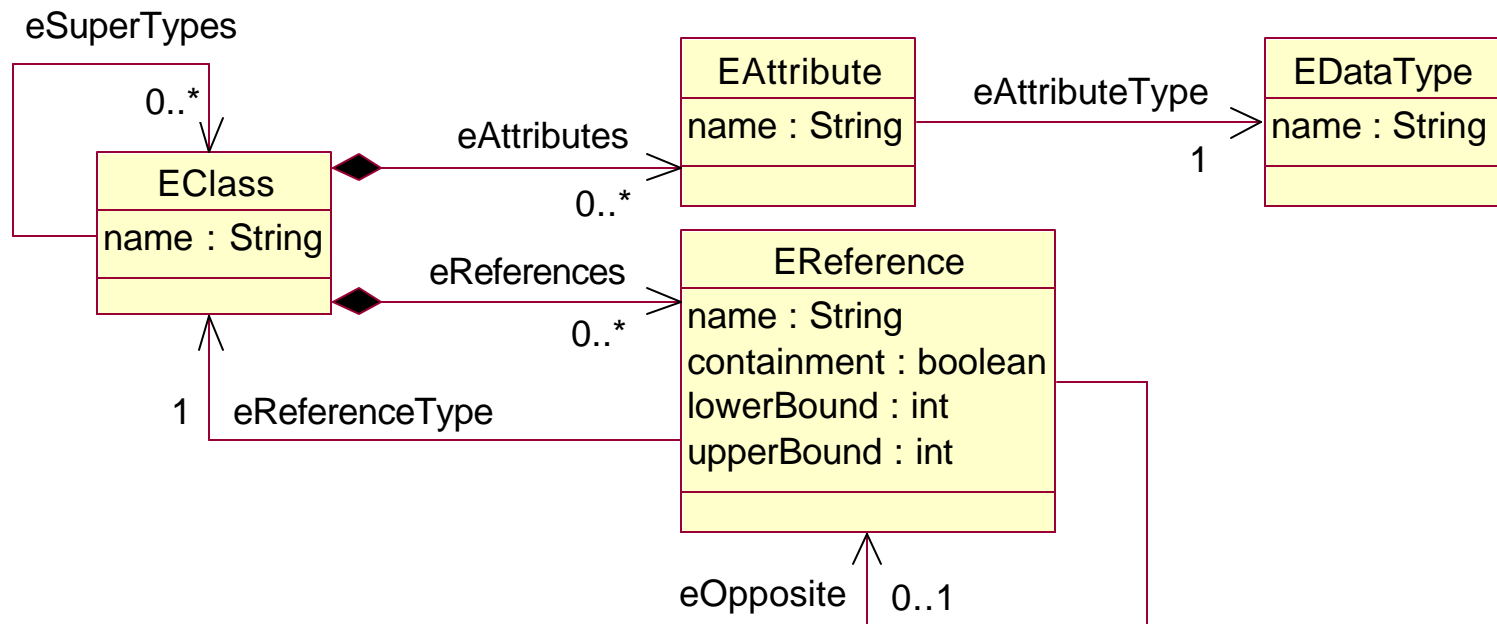
Ecore defines structure of EMF core models, which in turn define our application models.

Ecore is itself an EMF model, and thus is its own metamodel.

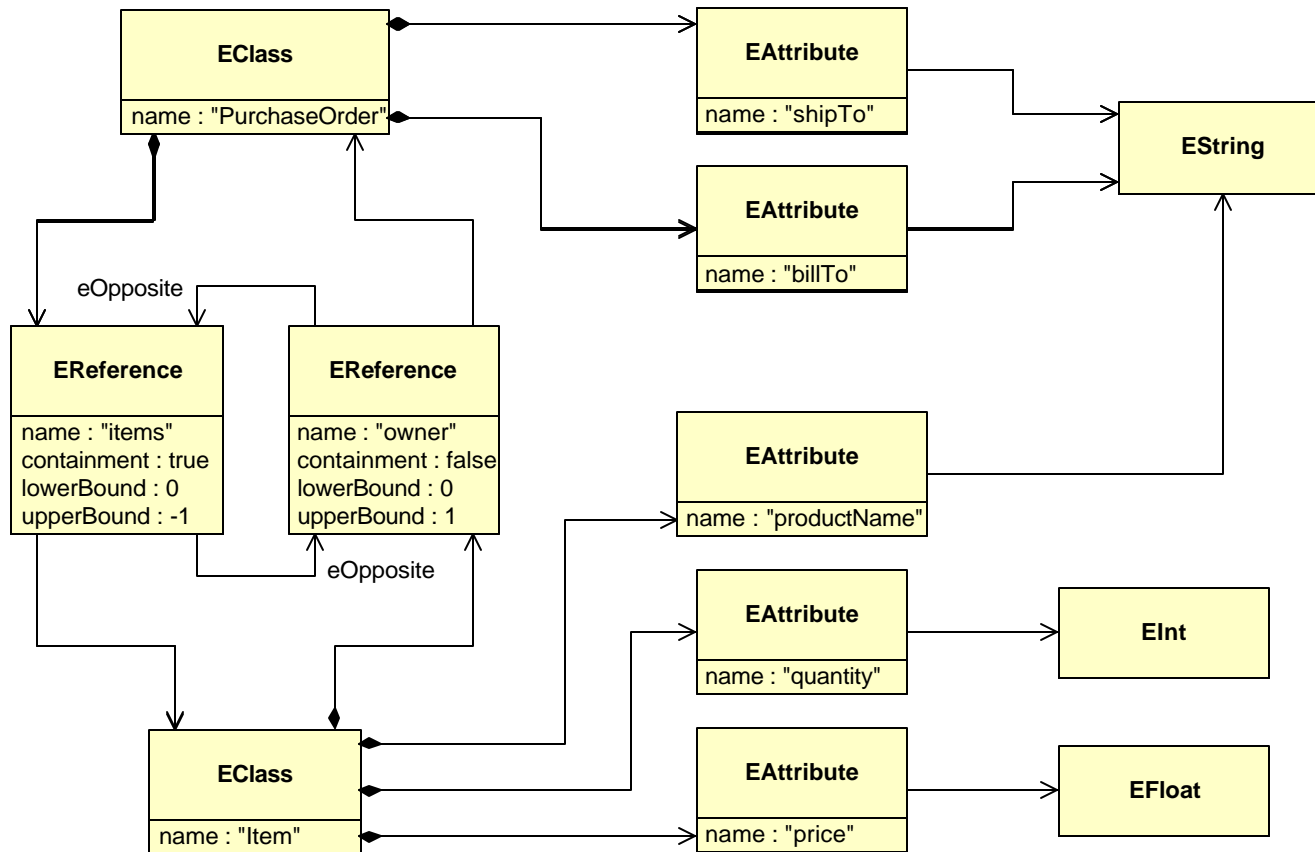
Modeling – The Ecore Hierarchy



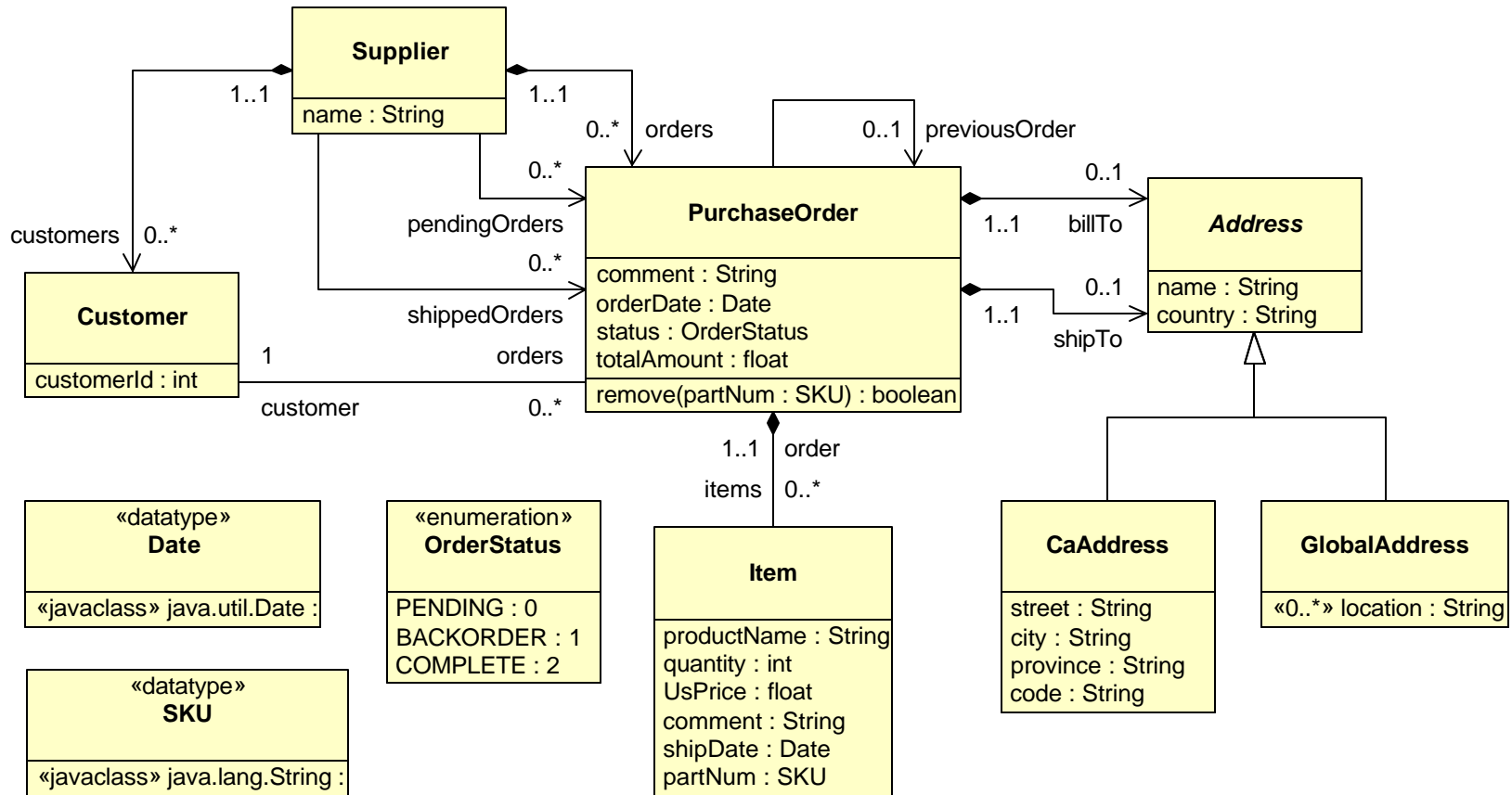
Modeling – The Ecore Kernel



Modeling – Ecore in Use



EMF Demonstration



Generating – Data Model Classes

Each model class produces a Java interface and an implementation class.

```
public interface PurchaseOrder extends EObject
{
    ...
}

public class PurchaseOrderImpl extends EObjectImp
    implements PurchaseOrder
{
    ...
}
```

Generating – Accessor Methods

Setters are generated with feature change notification using Observer design pattern.

```
public String getShipTo()
{
    return shipTo;
}

public void setShipTo(String newShipTo)
{
    String oldShipTo = shipTo;
    shipTo = newShipTo;
    if (eNotificationRequired())
        eNotify(new ENotificationImpl(this, Notification.SET,
            PoPackage.PURCHASE_ORDER__SHIPTO, oldShipTo, newShipTo));
}
```

Generating – Other Model Classes

Classes with «datatypes» stereotype are given string conversion methods.

```
public String convertDateToString(EDataType eDataType,
    Object instanceValue)
{
    ...
}

public Date convertDateFromString(EDataType eDataType,
    String initialValue)
{
    ...
}
```

Enumerated types are created using the Type-safe Enum pattern.

Generating – One-way References

Persistence proxies resolved by lazy loading.

```
public PurchaseOrder getPreviousOrder() {
    if (previousOrder != null && previousOrder.eIsProxy())
    {
        PurchaseOrder oldPreviousOrder = previousOrder;
        previousOrder = (PurchaseOrder)
            eResolveProxy((InternalEObject) previousOrder);
        if (previousOrder != oldPreviousOrder)
        {
            if (eNotificationRequired())
                eNotify(new ENotificationImpl(this,
                    Notification.RESOLVE, PoPackage...));
        }
    }
    return previousOrder;
}
```

Generating – Containment Refs

Contained classes are persisted with the containing owner class, in the same resource. No proxies need to be resolved.

```
public EList getItems()
{
    if (items == null)
    {
        items = new ObjectContainmentWithInverseEList(Item.class,
            this, PoPackage.PURCHASE_ORDER__ITEMS,
            PoPackage.ITEM__ORDER);
    }
    return items;
}
```

Generating – Operations

EOperation classes result in generation of empty method bodies tagged with a task list reminder to provide the code.

```
/**
 * @generated
 */
public boolean remove(String partNum)
{
    // TODO: implement this method
    // Remove @generated or mark it @generated NOT
    throw new UnsupportedOperationException();
}
```

Generating – Packages

- Singleton instance for each model package
- Provides access to model's metadata objects

```
PoPackage eINSTANCE = ca.encs.po.impl.PoPackageImpl.init();  
... int CUSTOMER = 1; int CUSTOMER__CUSTOMER_ID = 0;  
... private EClass customerEClass = null;  
... customerEClass = createEClass(CUSTOMER);  
... createEAttribute(customerEClass, CUSTOMER__CUSTOMER_ID);
```

```
public EClass getCustomer()  
{ return customerEClass; }
```

```
public EAttribute getCustomer_CustomerId()  
{ return (EAttribute)  
    customerEClass.getEStructuralFeatures().get(0); }
```

Generating – Factories

- Singleton instance for each model package
- For creation of model objects

```
PoFactory eINSTANCE = ca.encs.po.impl.PoFactoryImpl();


public Supplier createSupplier() {
    SupplierImpl supplier = new SupplierImpl();
    return supplier;
}

public EObject create(EClass eClass) {
    switch (eClass.getClassifierID())
    {
        case PoPackage.SUPPLIER:
            return createSupplier();
        ...
    }
}
```

Generating – Model Reflection

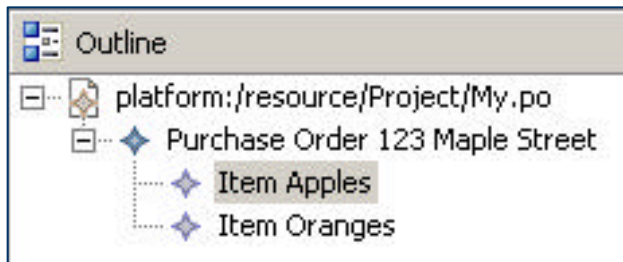
Generated, reflective API allows for generic access to the model (e.g., EMF.Edit):

```
PoPackage po_package = PoPackage.eINSTANCE;  
PoFactory po_factory = PoFactory.eINSTANCE;  
EClass po_class = po_package.getPurchaseOrder();  
EAttribute ship_att = po_package.getPurchaseOrder_ShipTo();  
  
EObject order = po_factory.create(po_class);  
order.eSet(ship_att, "123 Maple Street");  
String ship_to = (String) order.eGet(ship_att);  
  
boolean is_set = order.eIsSet(ship_att);  
  
boolean is_unset = order.eUnset(ship_att);
```

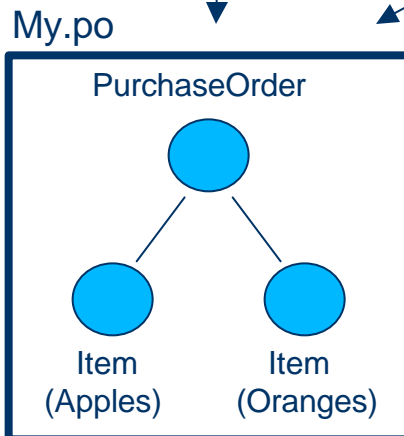


The diagram shows the text "EStructuralFeature" in teal on the right. Two red arrows originate from it: one points to the `eSet` method in the line `order.eSet(ship_att, "123 Maple Street");` and the other points to the `eGet` method in the line `String ship_to = (String) order.eGet(ship_att);`

Generating – EMF.Edit



Property	Value
Price	0.45
Product Name	Apples
Quantity	12



JFace viewers can be generated for your model.

- TreeViewer
- TableViewer
- ListView

Application Demonstration



The End: Questions?



References

- Frank Budinsky et al, Eclipse Modeling Framework: A Developer's Guide, Addison-Wesley, 2004
- Eric Gamma and Kent Beck, Contributing to Eclipse: Principles, Patterns, and Plug-ins, Addison-Wesley, 2004
- Ed Merks, The Eclipse Modeling Framework: Introducing Modeling to the Java Technology Mainstream, JavaOne presentation, 2004
- Bill Moore et al, Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework, IBM Redbook, 2004
- www.eclipse.org