The EMF: Modeling with Eclipse
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.
Three views depicting the same data model

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

Serialization through XMI (XML Metadata Interchange) connects model with XML documents.
Modeling – EMF Unifying UML
/** @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

```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>
```

[Budinsky et al., Eclipse Modeling Framework]
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

- EDataType
  - name : String

- EAttribute
  - name : String
  - eAttributeType : 1

- EReference
  - name : String
  - containment : boolean
  - lowerBound : int
  - upperBound : int
  - eOpposite : 0..1

- EClass
  - name : String
  - eSuperTypes : 0..*
  - eAttributes : 0..*
  - eReferences : 0..*

- EReferenceType : 1

[Budinsky et al., Eclipse Modeling Framework]
Modeling – Ecore in Use

- EClass name: "PurchaseOrder"
- EAttribute name: "shipTo"
- EAttribute name: "billTo"
- EClass name: "Item"
- EReference name: "items" containment: true lowerBound: 0 upperBound: -1
eOpposite
- EReference name: "owner" containment: false lowerBound: 0 upperBound: 1 eOpposite
- EAttribute name: "productName"
- EAttribute name: "quantity"
- EAttribute name: "price"
- EString
- EInt
- EFloat
EMF Demonstration

[Budinsky et al., Eclipse Modeling Framework]
Generating – Data Model Classes

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

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

public class PurchaseOrderImpl extends EObjectImpl implements PurchaseOrder {
    ...
}
```
Generating – Accessor Methods

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

```java
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));
}
```
Classes with «datatypes» stereotype are given string conversion methods.

```java
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.

```java
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.

```java
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.

```java
/**
 * @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

```java
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

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

class SupplierImpl {
    public SupplierImpl() {
        SupplierImpl supplier = new SupplierImpl();
        return supplier;
    }

class Supplier extends EObject {
    public static Supplier createSupplier() {
        switch (eClass.getClassifierID()) {
            case PoPackage.SUPPLIER:
                return createSupplier();
            case PoPackage.SUPPLIER:
                return createSupplier();
            default:
                return null;
        }
    }
```
Generated, reflective API allows for generic access to the model (e.g., EMF.Edit):

```java
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_set = order.eUnset(ship_att);
```

[Budinsky et al., Eclipse Modeling Framework]
Generating – EMF.Edit

JFace viewers can be generated for your model.

- TreeViewer
- TableViewer
- ListViewer
Application Demonstration
The End: Questions?
References

- Eric Gamma and Kent Beck, Contributing to Eclipse: Principles, Patterns, and Plug-ins, Addison-Wesley, 2004
- www.eclipse.org