CASCON 2005 Workshop
October 17 & 18, 2005

Hands-On:
The Eclipse Modeling Framework

http://eclipse.org/emf/docs/presentations/CASCON/

Nick Boldt, Dave Steinberg, Ed Merks
IBM Rational Software
Toronto, Canada
EMF and XSD Projects

Agenda

- Introduction
  - ***EMF in a Nutshell***
  - EMF Components
  - The Ecore Metamodel

- Exercise 1: Code Generation, Regeneration and Merge
- Exercise 2: Runtime Framework: Loading & Saving Resources

  – break –

- Exercise 3: Change Model & Change Recorder
- Exercise 4: Validation Framework
- Exercise 5: XML Processor & Reflection *(optional)*

- Summary
- What's New in EMF 2.2 for Eclipse 3.2?
- Q&A

- End 4:45pm

What is EMF?

- A modeling & data Integration framework for Eclipse
- What is an EMF “model” (Ecore)?
  - A general model of models (metamodel) from which any model can be defined
  - Specification of an application’s data
    - Object attributes
    - Relationships (Associations) between objects
    - Operations available on each object
    - Simple constraints (ex: cardinality) on objects and relationships
  - Essentially the Class Diagram subset of UML
- For more on cardinality & entity relationships see:
  - http://www.datamodel.org/DataModelCardinality.html
  - http://www.smartdraw.com/tutorials/software-erd/erdcardinality.htm

What does EMF Provide?

- From a model definition -- Java interfaces, UML, XML Schema -- EMF can generate efficient, correct, and easily customizable implementation code
- EMF converts your models to Ecore (EMF Meta Model)
- Tooling support within the Eclipse framework (or command line), including support for generating Eclipse-base and RCP editors
- Reflective and dynamic model invocation
- Supports XML/XMI (de) serialization of instances of a model
- And more….

Why EMF?

- EMF is middle ground in the modeling vs. programming world
  - Focus is on class diagram subset of UML modeling (object model)
  - Transforms models into Java code
  - Provides the infrastructure to use models effectively in your code
- Very low cost of entry
  - Full scale graphical modeling tool not required
  - EMF is free
  - Reuse your knowledge of UML, XML Schema, or Java

EMF History

- Originally based on MOF (Meta Object Facility)
  - From OMG (Object Management Group)
  - Abstract language and framework for specifying, constructing, and managing technology neutral meta-models
- EMF evolved based on experience supporting a large set of tools
  - Efficient Java implementation of a practical subset of the MOF API
- 2003: EMOF defined (Essential MOF)
  - Part of OMG's MOF 2 specification; UML2 based
  - EMF is approximately the same functionality
    - Significant contributor to the spec; adapting to it

Who is using EMF today?

- IBM
  - Rational Application Developer (RAD), Software Architect (RSA)
  - Websphere Studio (WSAD), Lotus Workplace
  - alphaWorks projects, including:
    - XML Forms Generator (http://www.alphaworks.ibm.com/tech/xfg)
    - Emfatic Language for EMF (http://www.alphaworks.ibm.com/tech/emfatic)
- Eclipse
  - Eclipse Test & Performance Tools Platform (TPTP) [was Hyades]
  - Eclipse Web Tools Platform (WTP)
  - UML2, Visual Editor (VE), EMF Technology Projects (EMFT)
- Independent Software Vendors (ISVs)
  - Borland (TogetherSoft), InferData, Ensemble, Versata, Omondo and more
- Large open source community
  - 7 million *logged* file requests (downloads) in the past 30 days (235k/day), including 60,000 zip requests (2000/day), or
  - Equal to about 6000 EMF download requests per day, and growing!

What have people said about EMF?

- EMF represents the ***core subset*** that's left when the non-essentials are eliminated. It represents a ***rock solid foundation*** upon which the more ambitious extensions of UML and MDA can be built.
  - *Vlad Varnica, OMONDO Business Development Director, 2002*

- EMF ***provides the glue between the modeling and programming worlds***, offering an infrastructure to use models effectively in code by integrating UML, XML and Java. EMF thus fits well into [the] Model-Driven Development approach, and is ***critically important for Model-Driven Architecture***, which underpins service-oriented architectures [SOA].
  - *Jason Bloomberg, Senior analyst for XML and Web services, research firm ZapThink, 2003*

- The EMF […] with UML stuff is pretty cool in Eclipse. Maybe one day MDA will make its way into the NetBeans GUI.
  - *posted to theserverside.com, November 2004 (circa NetBeans 4.1 EA release)*

- "[As] a consultant with fiduciary responsibility to my customers, [...] given the ***enormous traction*** that Eclipse has gathered, we have to view the EMF metadata management framework as the ***de facto standard***."
  - *David Frankel, as seen in Business Process Trends, March 2005*

EMF Model Sources

- EMF models can be defined in (at least) three ways:
  1. Java Interfaces
  2. UML models expressed in Rose files
  3. XML Schema

- Choose the one matching your perspective or skills

EMF Model Sources (cont)

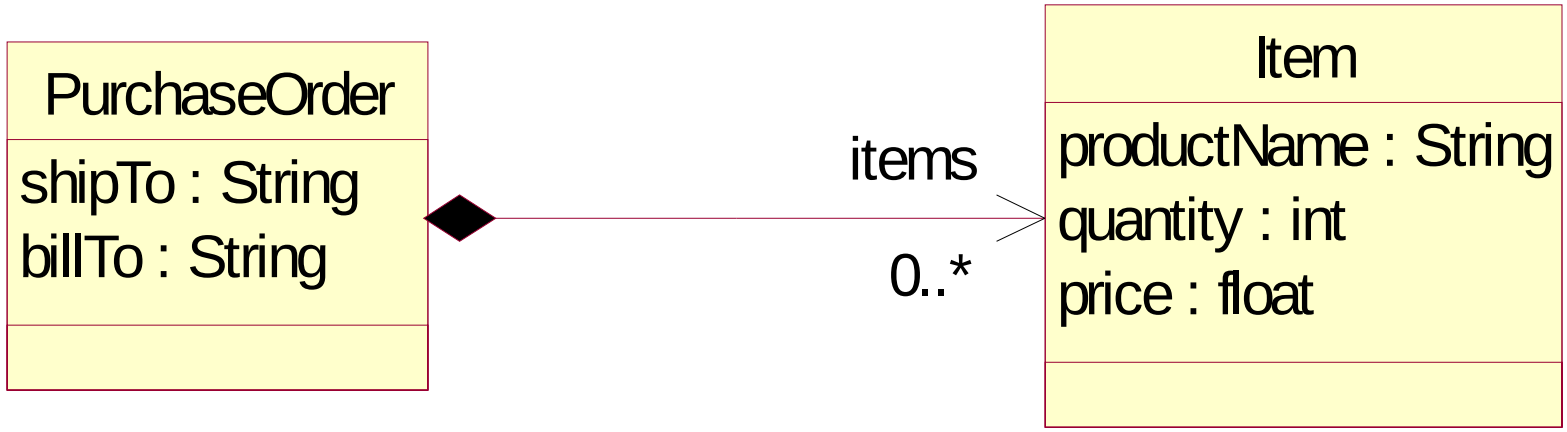
1. Java Interfaces

```
public interface PurchaseOrder {
  String getShipTo();
  void setShipTo(String value);
  String getBillTo();
  void setBillTo(String value);
  List getItems(); // List of Item
}

public interface Item {
  String getProductName();
  void setProductName(String value);
  int getQuantity();
  void setQuantity(int value);
  float getPrice();
  void setPrice(float value);
}
```

EMF Model Sources (cont)

2. UML Class Diagram



EMF Model Sources (cont)

3. XML Schema

```
<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>
```

EMF Model Sources (cont)

Unifying Java™, XML, and UML technologies

- All three forms provide the same information
  - Different visualization/representation
  - The application's "model" of the structure

- From a model definition, EMF can generate:
  - Java implementation code, including UI
  - XML Schemas
  - Eclipse projects and plug-ins

A Typical EMF Usage Scenario

1. Create EMF model
   - Import UML (e.g. Rational Rose .mdl file)
   - Import XML Schema
   - Import annotated Java interfaces
   - Create Ecore model directly using EMF Ecore editor or Omondo's EclipseUML graphical editor
2. Generate Java code for model
3. Prime the model with instance data using generated EMF model editor
4. Iteratively refine model (and regenerate code) and develop Java application
5. Use EMF.Edit to build customized user interface

Agenda

- Introduction
  - EMF in a Nutshell
  - ***EMF Components***
  - The Ecore Metamodel

- Exercise 1: Code Generation, Regeneration and Merge
- Exercise 2: Runtime Framework: Loading & Saving Resources

  – break –

- Exercise 3: Change Model & Change Recorder
- Exercise 4: Validation Framework
- Exercise 5: XMLProcessor & Reflection *(optional)*

- Summary
- What's New in EMF 2.2 for Eclipse 3.2?
- Q&A

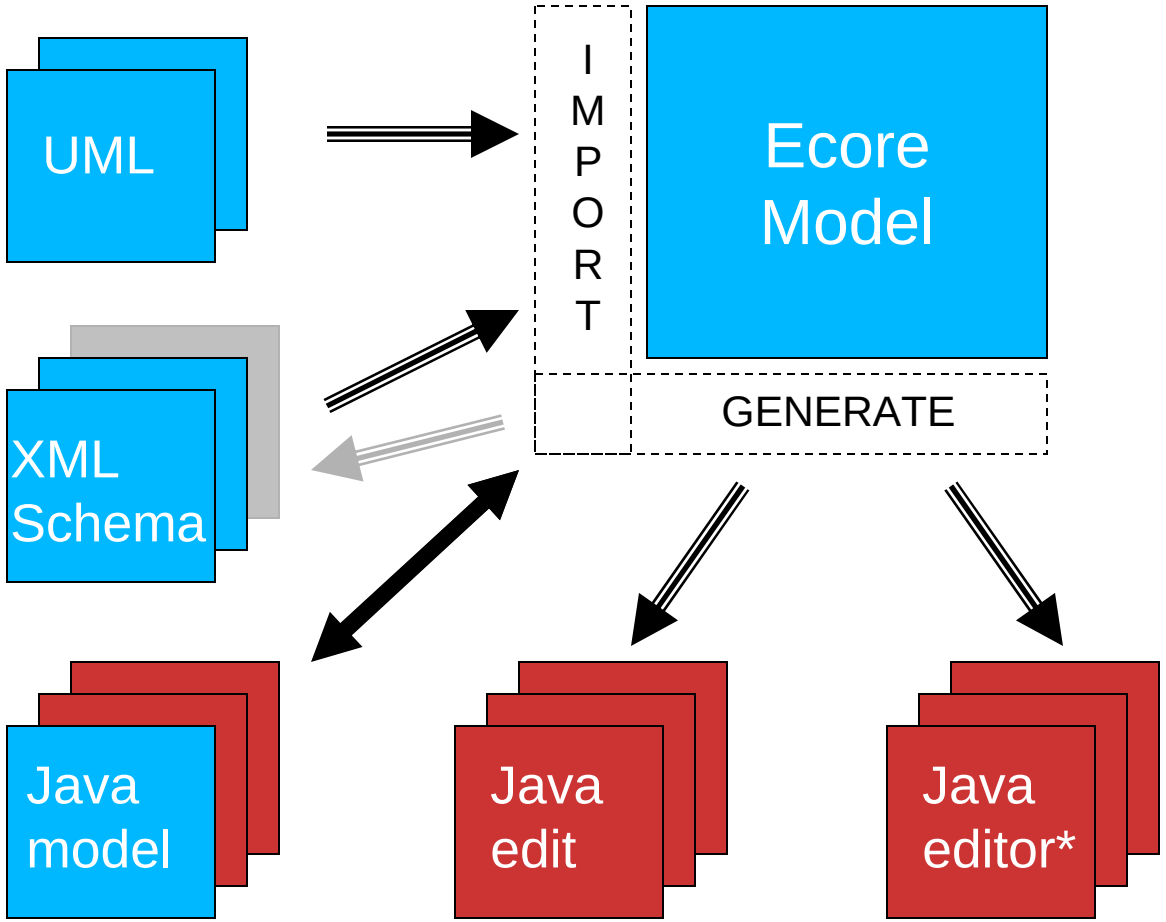
- End 4:45pm

EMF Components

- **EMF Core**
  - Ecore meta model
  - Model change notification & validation
  - Persistence and serialization
  - Reflection API
  - Runtime support for generated models
- **EMF Edit**
  - Helps integrate models with a rich user interface
  - Used to build editors and viewers for your model
  - Includes default reflective model editor
- **EMF Codegen**
  - Code generator for core and edit based components
  - Extensible model importer framework

EMF Tooling

Model Import and Generation



Generator features:

- Customizable JSP-like templates (JET)
- JDT-integrated, command-line, or Ant
- Fully supports regeneration and merge

* requires Eclipse to run

EMF Model Importers

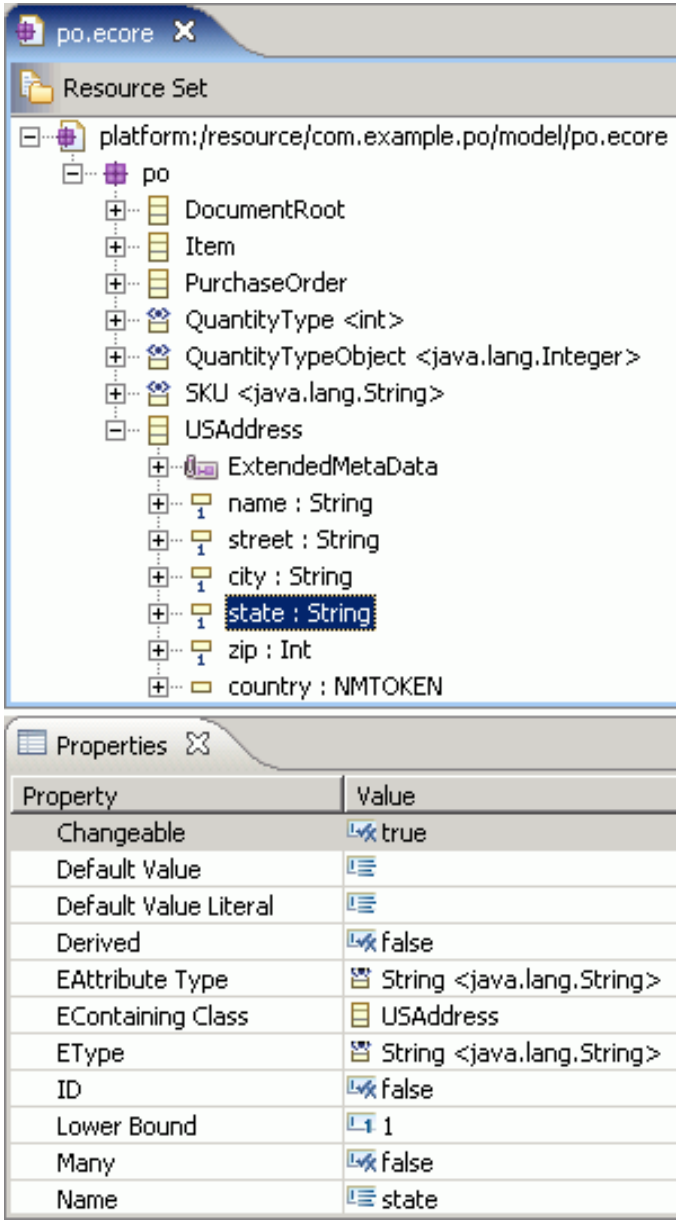
- UML
  - Rational Rose `.mdl` file
  - Eclipse UML2 project provides importer for `.uml2`
- Annotated Java
  - Consists of Java interfaces for each class model
  - Annotations using `@model` tags added to interface to express model definition not possible with code
  - Lowest cost approach
- XML Schema
  - Describes the data of the modeled domain
  - Provides richer description of the data, which EMF exploits
- Ecore model (`*.ecore` file)
  - Just creates the generator model (discussed later)
  - Also handles EMOF (`*.emof`)

Model Creation

- Ecore model created within an Eclipse project via wizard using one of the sources described in previous slide
- Output is:
  - *modelname*`.ecore` file
    - Ecore model file in XMI format
    - Canonical form of the model
  - *modelname*`.genmodel` file
    - A “generator model” for specifying generator options
    - Decorates `.ecore` file
    - EMF code generator is an EMF `.genmodel` editor
    - `.genmodel` and `.ecore` files automatically kept in sync

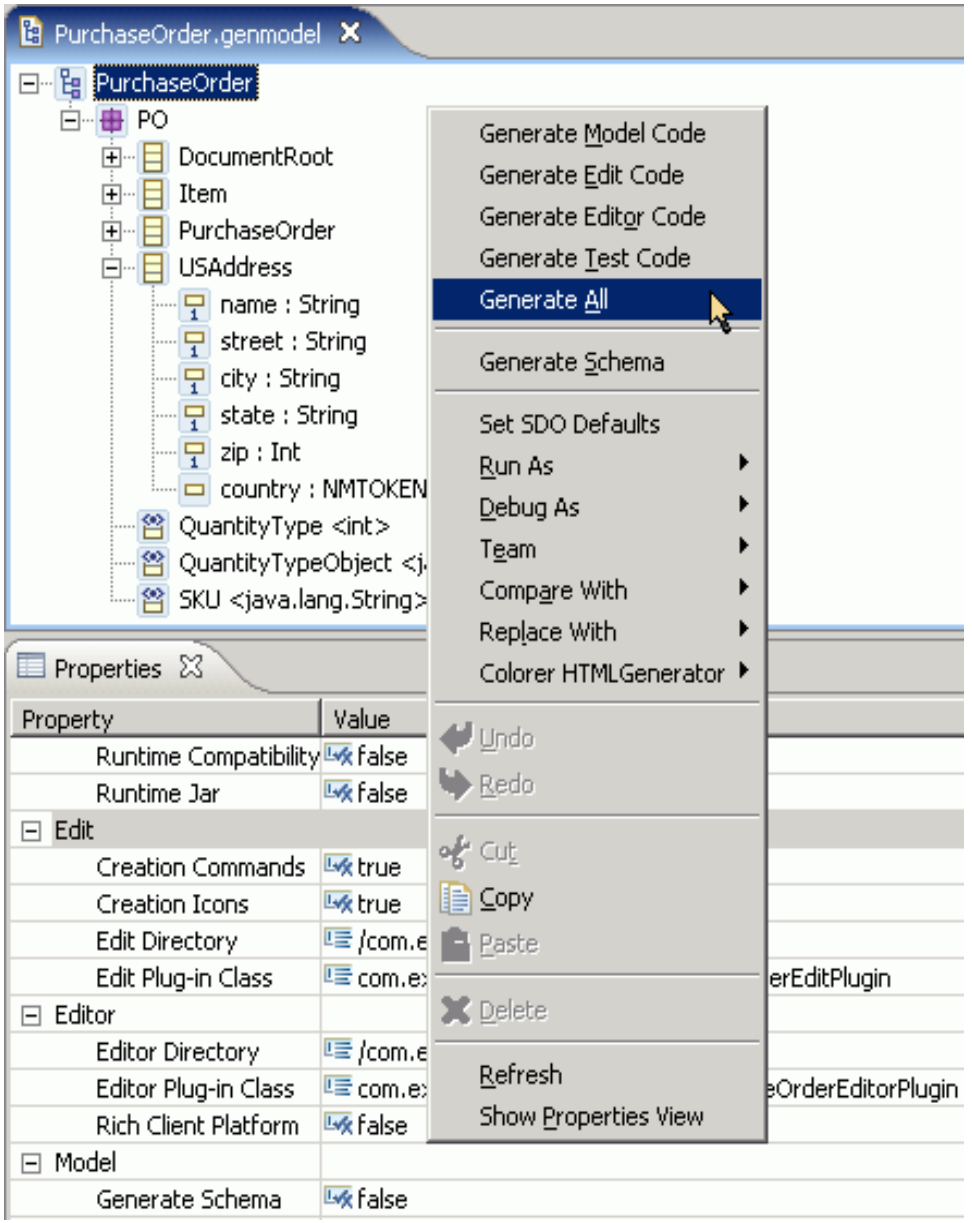
Ecore Model Editor

- A generated (and customized) EMF editor for the Ecore model
- Models can be edited using tree view in conjunction with property view
  - New components (EClass, EAttribute, EReference, etc.) created using popup actions in tree view
  - Set names, etc., in property view
- Note: a graphical editor (e.g., Omondo) is better approach



EMF Generator

- Similar layout to the Ecore model editor, and kept in sync with `.ecore` changes
- Use context menu actions or Generator menu) to generate code
  1. The **Generate Model Code** action produces Java code to implement the model
  2. The **Generate Edit Code** action produces adapter code to support viewers
  3. The **Generate Editor Code** action produces a fully functional Eclipse editor
  4. The **Generate Test Code** action produces a set of JUnit tests stubs
  5. The **Generate All** action produces all four of the above.
  6. The **Generate Schema** produces an XML Schema document representing the `.ecore` file
- Generation options expressed in Properties view
- Use **Generator > Reload** to push changes in source model to `.ecore / .genmodel`
- Command line API also available



Agenda

- Introduction
  - EMF in a Nutshell
  - EMF Components
  - ***The Ecore Metamodel***

- Exercise 1: Code Generation, Regeneration and Merge
- Exercise 2: Runtime Framework: Loading & Saving Resources

  – break –

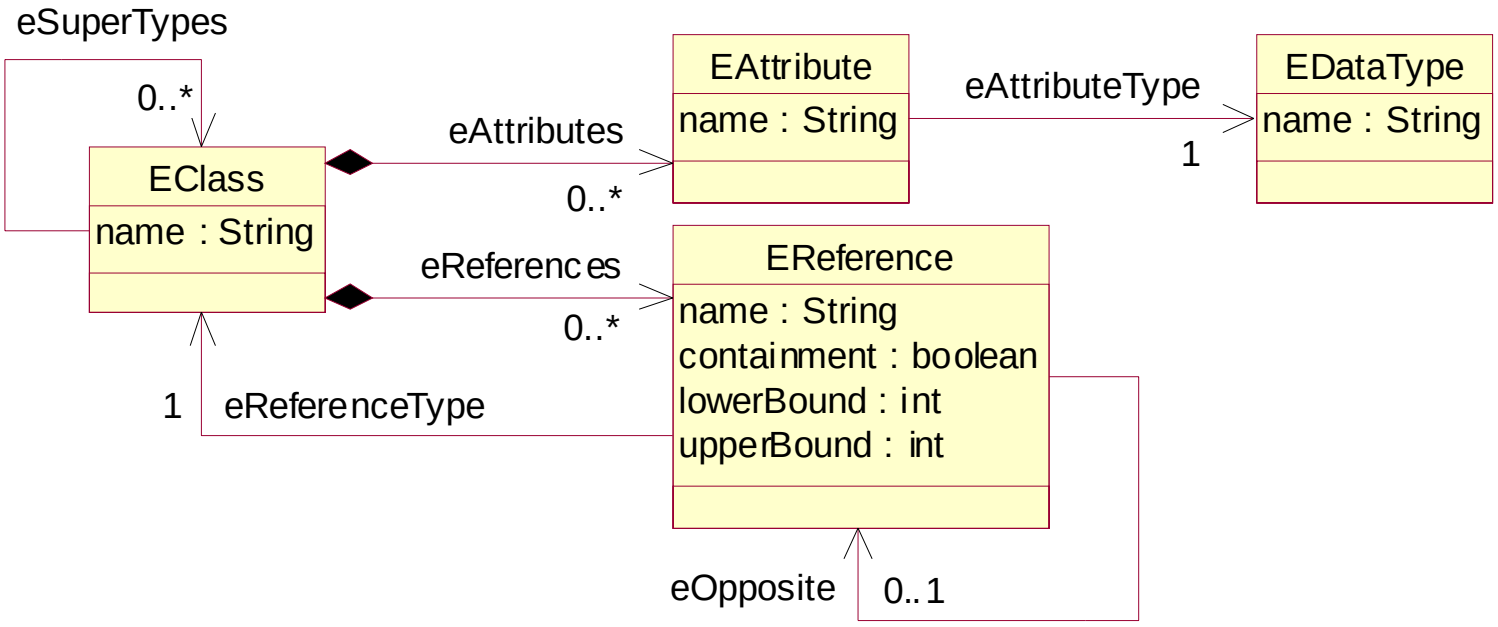
- Exercise 3: Change Model & Change Recorder
- Exercise 4: Validation Framework
- Exercise 5: XMLProcessor & Reflection *(optional)*

- Summary
- What's New in EMF 2.2 for Eclipse 3.2?
- Q&A

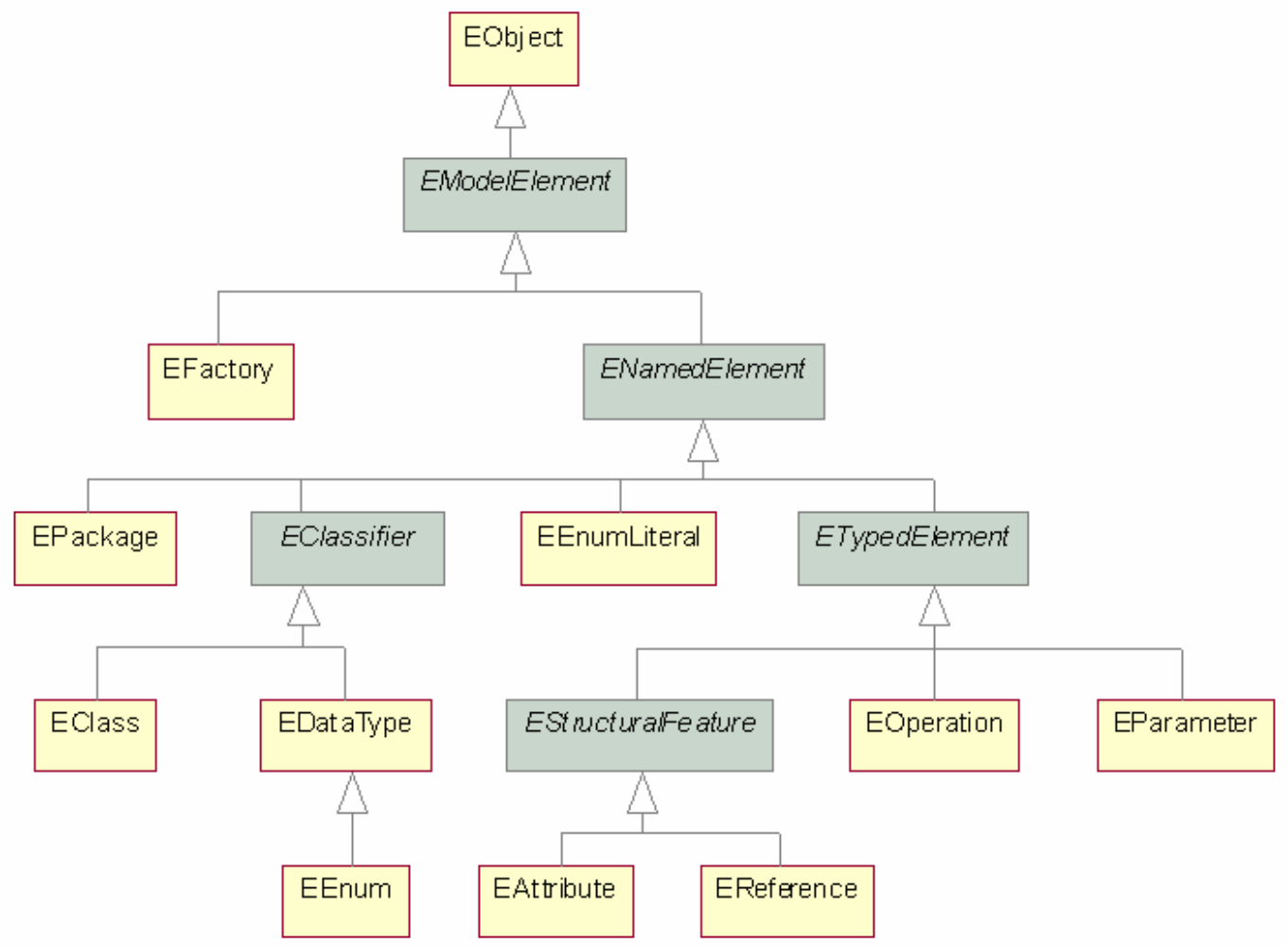
- End 4:45pm

The Ecore (Meta) Model

- Ecore is EMF's model of a model (metamodel)
  - Persistent representation is XMI



Ecore Meta Model



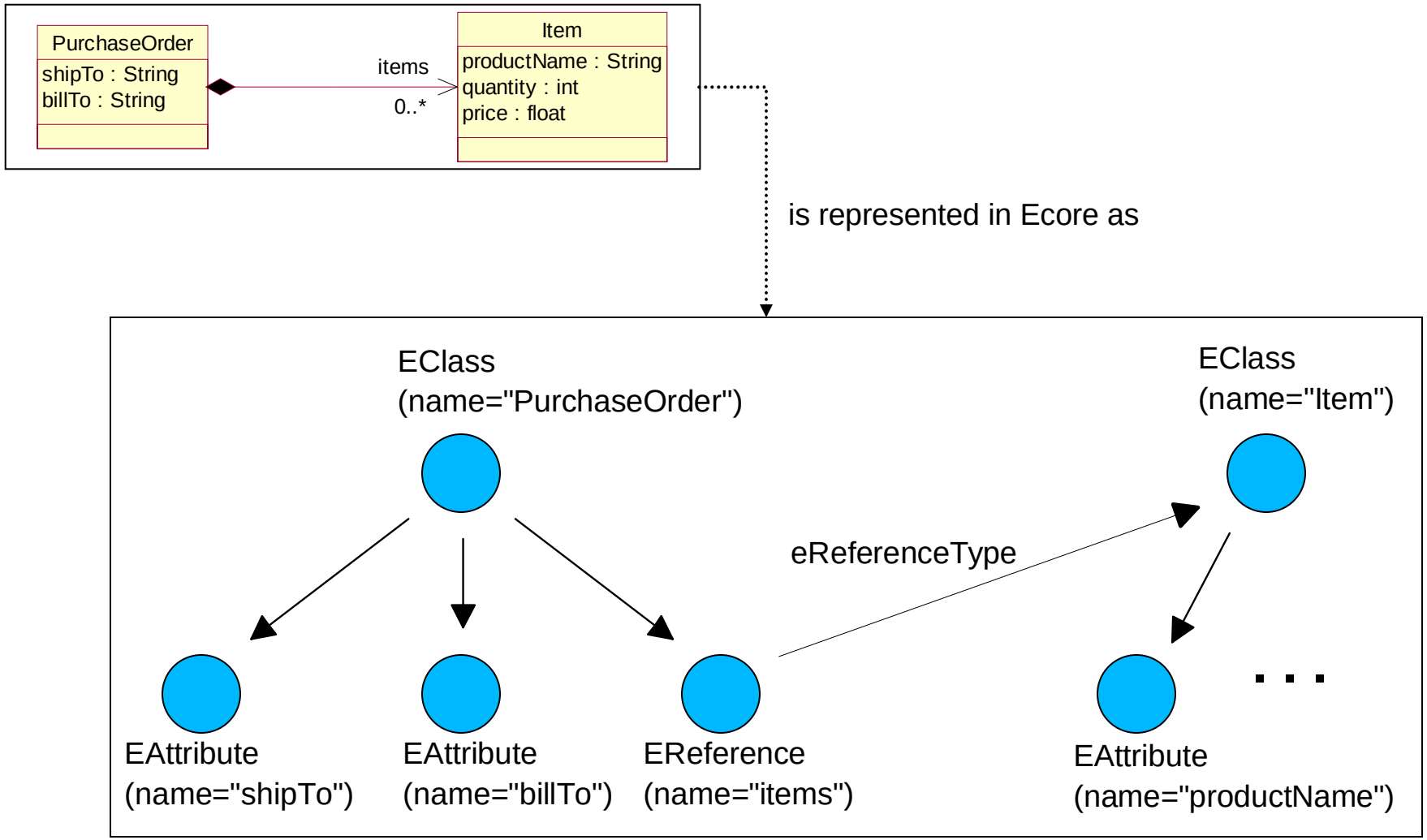
EObject is the root of every model object. Equivalent to java.lang.Object

Partial List of Ecore Data Types

Ecore Data Type	Java Primitive Type or Class
EBoolean	boolean
EChar	char
EFloat	float
EString	java.lang.String
EByteArray	byte[ ]
EBooleanObject	java.lang.Boolean
EFloatObject	java.lang.Float
EJavaObject	java.lang.Object

Note: Ecore datatypes are serializable; support for custom datatypes

PurchaseOrder Ecore Model



PurchaseOrder Ecore XMI

```
<eClassifiers xsi:type="ecore:EClass"
    name="PurchaseOrder">
  <eStructuralFeatures xsi:type="ecore:EReference"
      name="items" upperBound="-1" eType="#//Item"
      containment="true"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute"
      name="shipTo"
      eType="ecore:EDataType http:...Ecore#//EString"/>
  <eStructuralFeatures xsi:type="ecore:EAttribute"
      name="billTo"
      eType="ecore:EDataType http:...Ecore#//EString"/>
</eClassifiers>
```

- Alternate serialization format is EMOF (Essential MOF) XMI
  - Part of OMG Meta-Object Facility (MOF) Standard
  - For more, see http://www.omg.org/docs/ad/05-08-01.pdf

Agenda

- Introduction
  - EMF in a Nutshell
  - EMF Components
  - The Ecore Metamodel

- ***Exercise 1: Code Generation, Regeneration and Merge***
- Exercise 2: Runtime Framework: Loading & Saving Resources

  – break –

- Exercise 3: Change Model & Change Recorder
- Exercise 4: Validation Framework
- Exercise 5: XMLProcessor & Reflection *(optional)*

- Summary
- What's New in EMF 2.2 for Eclipse 3.2?
- Q&A

- End 4:45pm

Generated Java Code

- EMF framework is lightweight
  - Generated code is clean, simple, efficient
- EMF can generate
  - Model implementation
  - UI-independent edit support
  - Editor (Eclipse IDE-integrated or RCP application)
  - JUnit test skeletons
  - Manifests, plug-in classes, properties, icons, etc.

Generated Model Code

- Interface and implementation for each modeled class
  - Includes get/set accessors for attributes and references

```
public interface PurchaseOrder extends EObject {
  String getShipTo();
  void setShipTo(String value);
  String getBillTo();
  void setBillTo(String value);
  EList getItems();
}
```

  - Usage example

```
order.getItems().add(item);
```

Generated Model Code

- Factory to create instances of model objects

```
POFactory factory = POFactory.eINSTANCE;
PurchaseOrder order = factory.createPurchaseOrder();
```

- Package class provides access to metadata

```
POPackage poPackage = POPackage.eINSTANCE;
EClass itemClass = poPackage.getItem();

EAttribute priceAttr = poPackage.getItem_Price();
//or itemClass.getEStructuralFeature(POPackage.ITEM__PRICE)
```

- Also generated: switch utility, adapter factory base, validator, custom resource, XML processor

Generated Edit/Editor Code

- Viewing/editing code divided into two parts:
  - UI-independent code (placed in edit plug-in)
    - Item providers (adapters)
    - Item provider adapter factory
  - UI-dependent code (placed by default in a separate editor plug-in)
    - Model creation wizard
    - Editor
    - Action bar contributor
    - Advisor (RCP)

Regeneration and Merge

- Hand-written code can be added to generated code and preserved during regeneration
- All generated classes, interfaces, methods include `@generated` marker
- Replace generated code by removing `@generated` marker or appending
  - or include additional text like

    `@generated NOT`
- Methods without `@generated` marker are left alone during regeneration

Regeneration and Merge

- Extend (vs. replace) generated method through redirection

- To override the `getQuantity()` generated method:
  - Add suffix `Gen` to generated method

    `getQuantity()` becomes `getQuantityGen()`
  - During regen, the generated body will be redirected to the `...Gen()` method
  - Create your own `getQuantity()` method and then can call generated `getQuantityGen()`

Exercise 1:
Code Generation, Regeneration and Merge

See:

- **CASCON 2005/Exercise1_CodeGen_Regen_Merge/_Exercise1_Instructions.html**
- **CASCON 2005/Exercise1_CodeGen_Regen_Merge/PurchaseOrder.xsd (XML Schema)**
- **CASCON 2005/Exercise1_CodeGen_Regen_Merge/PurchaseOrder.mdl (Rose model)**

Agenda

- Introduction
  - EMF in a Nutshell
  - EMF Components
  - The Ecore Metamodel

- Exercise 1: Code Generation, Regeneration and Merge
- ***Exercise 2: Runtime Framework: Loading & Saving Resources***

  – break –

- Exercise 3: Change Model & Change Recorder
- Exercise 4: Validation Framework
- Exercise 5: XMLProcessor & Reflection *(optional)*

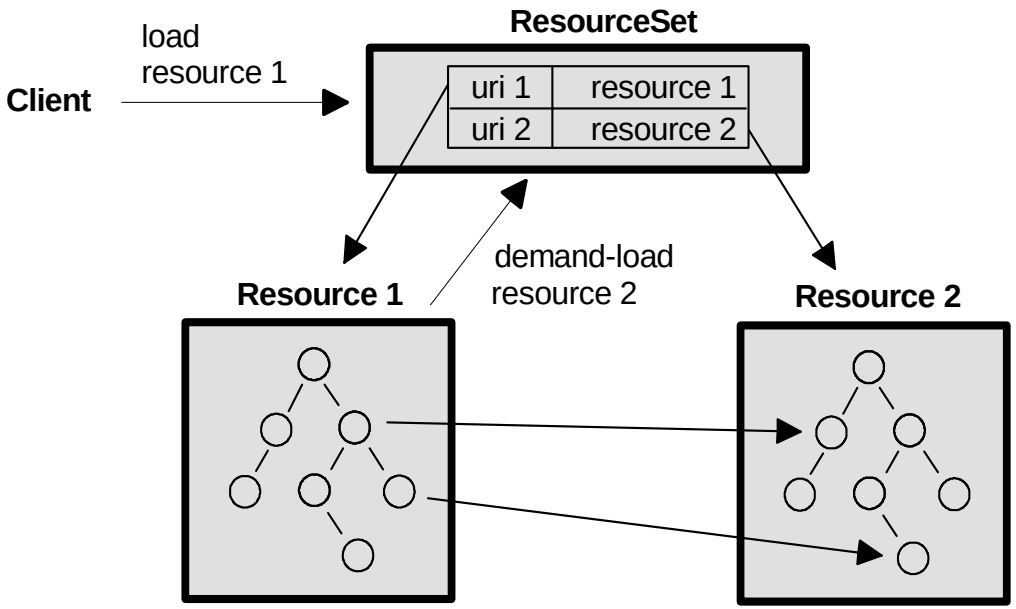
- Summary
- What's New in EMF 2.2 for Eclipse 3.2?
- Q&A

- End 4:45pm

EMF Runtime Framework

- Persistence and serialization of model data
  - Proxy resolution and demand load
- Automatic notification of model changes
- Bi-directional reference handshaking
- Dynamic object access through a reflection API
- Runtime environments
  - Eclipse
  - RCP
  - Standalone Java

Persistence and Serialization



- Serialized data is referred to as a **Resource**
  - Data can be spread out among a number of resources in a **Resource Set**
- One resource is loaded at a time, even if it has references to objects in other resources in the resource set
  - Proxies exist for objects in other resources
  - Lazy or demand-loading of other resources as needed
  - A resource can be unloaded

Resource Set

- Context for multiple resources that may have references among them
- Usually just an instance of ResourceSetImpl, or a customized subclass
- Provides factory method for creating new resources in the set:

```
ResourceSet rs = new ResourceSetImpl();
URI uri = URI.createFileURI("C:/data/po.xml");
Resource resource = rs.createResource(uri);
```

- Also provides access to the registries, URI converter, and default load options for the set

Resource Factory Registry

- Returns a resource factory for a given type of resource
  - Based on the URI scheme or filename extension
  - Determines the type of resource, hence format for save/load

```
Resource.Factory.Registry registry =
  rs.getResourceFactoryRegistry();
registry.getExtensionToFactoryMap().put(
  "xml", new XMLResourceFactoryImpl());
```

- For models created from XML Schema, the generated custom resource factory implementation should be registered to ensure schema-conformant serialization
  - When running model as a plug-in under Eclipse, EMF provides an extension point for registering resource factories
  - Generated plugin.xml registers generated resource factory against a package-specific extension (e.g. “po”)
- Global registry: Resource.Factory.Registry.INSTANCE
  - Consulted if no registered resource factory found locally

Package Registry

- Returns the package identified by a given namespace URI
  - Used during loading to access the factory for creating instances

```
EPackage.Registry registry = rs.getPackageRegistry();
registry.put(POPackage.eNS_URI, POPackage.eINSTANCE);
```

- Global registry: EPackage.Registry.INSTANCE
  - Consulted if no registered package found locally
- Running in Eclipse, EMF provides an extension point for globally registering generated packages
- Even standalone, a package automatically registers itself when accessed:

```
POPackage poPackage = POPackage.eINSTANCE;
```

Resource

- Container for objects that are to be persisted together
  - Convert to and from persistent form via save() and load()
  - Access contents of resource via getContents()

```
URI uri = URI.createFileURI("C:/data/po.xml");
Resource resource = rs.createResource(uri);
resource.getContents().add(p1);
resource.save(null);
```

- EMF provides XMLResource:

```
<PurchaseOrder>
  <shipTo>John Doe</shipTo>
  <next>p2.xml#p2</next>
</PurchaseOrder>
```

- Other, customized XML resource implementations, provided, too (e.g. XMI, Ecore, EMOF)

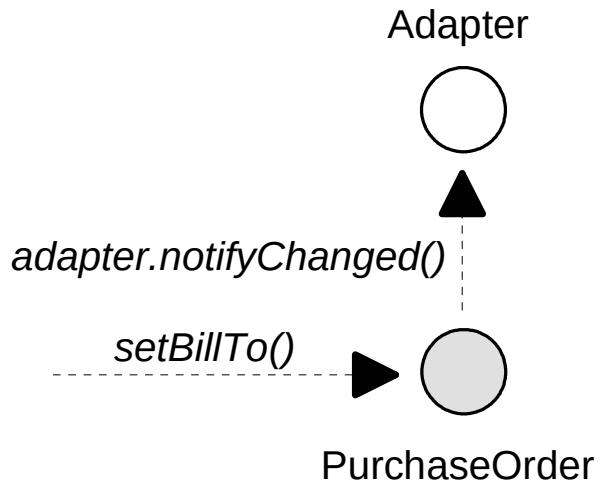
Proxy Resolution and Demand Load


```
PurchaseOrder p2 = p1.getNext();
```

Model Change Notification

- Every EMF object is also a Notifier
  - Send notification whenever an attribute or reference is changed
  - EMF objects can be “observed” in order to update views and dependent objects

```
Adapter poObserver = ...
purchaseOrder.eAdapters().add(poObserver);
```



Model Change Notification

- Observers or listeners in EMF are called adapters
  - Adapter can also extend class behavior without subclassing
  - For this reason they are typically added using an AdapterFactory

```
PurchaseOrder purchaseOrder = ...
AdapterFactory somePOAdapterFactory = ...
Object poExtensionType = ...

if (somePOAdapterFactory.isFactoryForType(poExtensiontype))
{
  Adapter poAdapter = somePOAdapterFactory.adapt(
    purchaseOrder, poExtensionType);
  ...
}
```

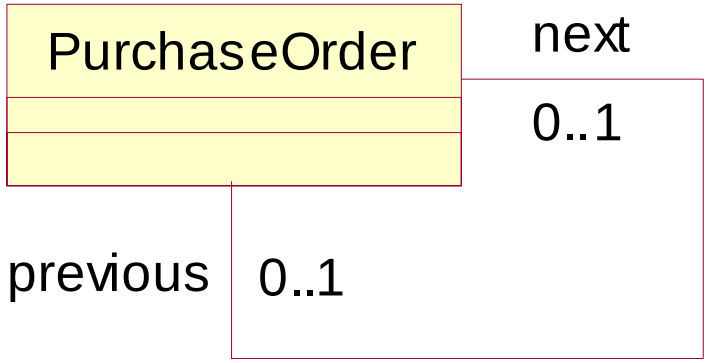
Model Change Notification

- Efficient notification calls in “set” methods
  - Checks for listeners before sending

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

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

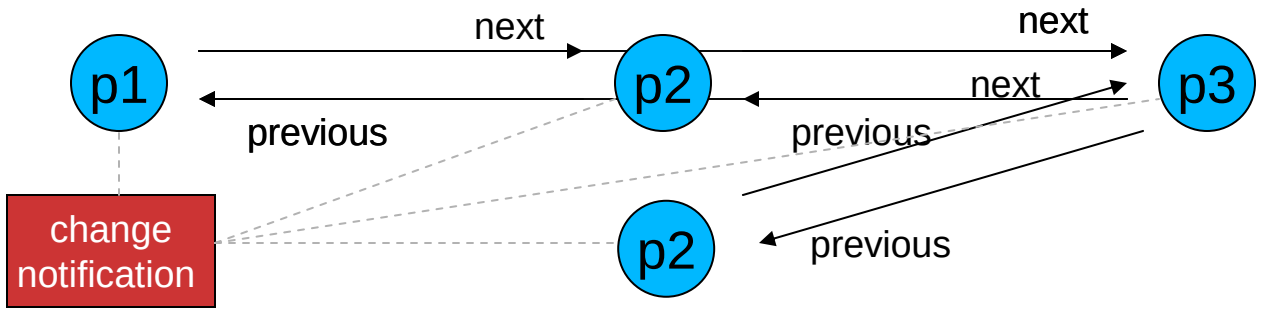
Bidirectional Reference Handshaking




```
po.getNext().getPrevious() == po
```

```
public interface PurchaseOrder {
  ...
  PurchaseOrder getNext();
  void setNext(PurchaseOrder value);
  PurchaseOrder getPrevious();
  void setPrevious(PurchaseOrder value);
}
```

Bidirectional Reference Handshaking




```
p1.setNext(p3);
```

Reflection

- All EMF classes implement interface EObject
- Provides an efficient API for manipulating objects reflectively
  - Used by the framework (e.g., generic serializer, copy utility, generic editing commands, etc.)
  - Also key to integrating tools and applications built using EMF

```
public interface EObject {
  EClass eClass();
  Object eGet(EStructuralFeature f);
  void eSet(EStructuralFeature f, Object v);
  ...
}
```

Reflection Example

- Setting an attribute using generated API:

```
PurchaseOrder po = ...
po.setBillTo("123 Elm St.");
```

- Using reflective API:

```
EObject po = ...
EClass poClass = po.eClass();
po.eSet(poClass.getEStructuralFeature("billTo"),
  "123 Elm St.");
```

Reflection Performance

- Efficient generated switch implementation of reflective methods

```
public Object eGet(EStructuralFeature eFeature) {
  switch (eDerivedStructuralFeatureID(eFeature))
  {
    case POPackage.PURCHASE_ORDER__SHIP_TO:
      return getShipTo();
    case POPackage.PURCHASE_ORDER__BILL_TO:
      return getBillTo();
    ...
}
```

Reflection Benefits

- Reflection allows generic access to any EMF model
  - Similar to Java's introspection capability
  - Every EObject (which is every EMF object) implements the reflection API
- An integrator need only know your model!
- A generic EMF model editor uses the reflection API
  - Can be used to edit any EMF model

Dynamic EMF

- Ecore models can be defined dynamically in memory
  - No generated code required
  - Dynamic implementation of reflective EObject API provides same runtime behavior as generated code
  - Also supports dynamic subclasses of generated classes
- All EMF model instances, whether generated or dynamic, are treated the same by the framework

Dynamic EMF Example

- Model definition using Ecore API:

```
EPackage poPackage = EcoreFactory.eINSTANCE.createEPackage();
poPackage.setName("po");
poPackage.setNsURI("http://www.example.com/PurchaseOrder");

EClass poClass = EcoreFactory.eINSTANCE.createEClass();
poClass.setName("PurchaseOrder");
poPackage.getEClassifiers().add(poClass);


EAttribute billTo = EcoreFactory.eINSTANCE.createEAttribute();
billTo.setName("billTo");
billTo.setEType(EcorePackage.eINSTANCE.getEString());
poClass.getEStructuralFeatures().add(billTo);
...

EObject po = poPackage.getEFactoryInstance().create(poClass);
po.eSet(billTo,"123 Elm St.");
```

- Or load from an .ecore file

Exercise 2:
Runtime Framework: Loading & Saving Resources

See:

- **CASCON 2005/Exercise2_Create_Load_Save/_Exercise2_Instructions.html**
- **CASCON 2005/Exercise2_Create_Load_Save/CreatePOInstance.java (example of how to create xml instance, but not save)**
- **CASCON 2005/Solution2_Optional/src/exercises/CreatePOInstance.java (includes save)**

Break

Agenda

- Introduction
  - EMF in a Nutshell
  - EMF Components
  - The Ecore Metamodel

- Exercise 1: Code Generation, Regeneration and Merge
- Exercise 2: Runtime Framework: Loading & Saving Resources

  – break –

- ***Exercise 3: Change Model & Change Recorder***
- Exercise 4: Validation Framework
- Exercise 5: XMLProcessor & Reflection *(optional)*

- Summary
- What's New in EMF 2.2 for Eclipse 3.2?
- Q&A

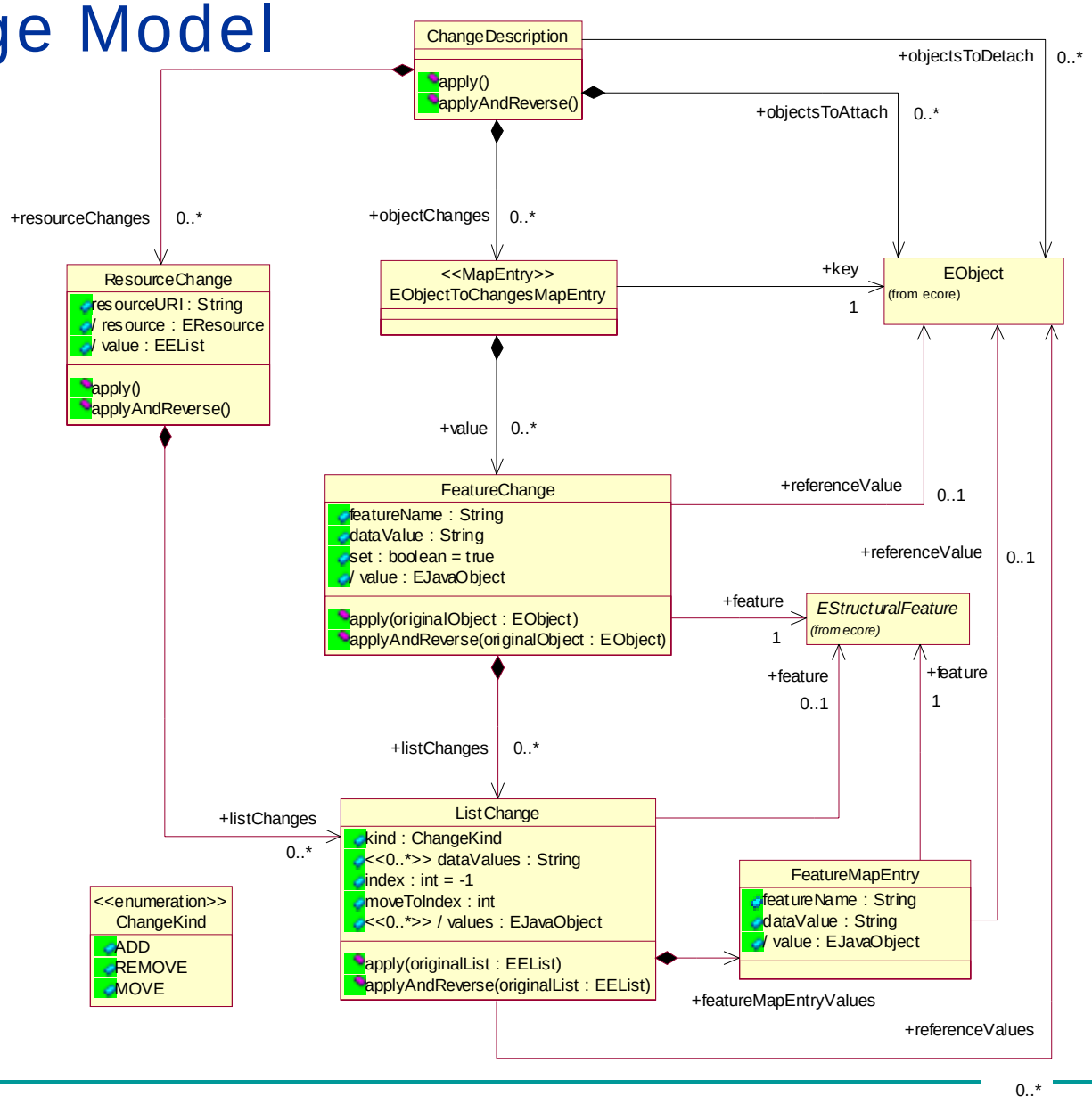
- End 4:45pm

Recording Changes

- Change Model
  - An EMF model for representing changes to objects
  - Directly references affected objects
  - Includes “apply changes” capability

- Change Recorder
  - EMF adapter
  - Monitors objects to produce a change description (an instance of the change model)

Change Model



Change Recorder

- Can be attached to EObjects, Resources, and ResourceSets
  - Monitors changes to the objects and their contents trees
- Produces a description of the changes needed to return to the original state (a reverse delta)

```
PurchaseOrder order = ...
order.setBillTo("123 Elm St.");

ChangeRecorder recorder = new ChangeRecorder();
recorder.beginRecording(Collections.singleton(order));
order.setBillTo("456 Cherry St.");
ChangeDescription change = recorder.endRecording();
```

- Result: a change description with one change, setting billTo to “123 Elm St.”

Applying Changes

- Given a change description, the change can be applied:
  - ChangeDescription.apply()
    - consumes the changes, leaving the description empty
  - ChangeDescription.applyAndReverse()
    - reverses the changes, leaving a description of the changes originally made (the forward delta)
  - Note: the model is always left in an appropriate state for applying the resulting change description

Example: Transaction Capability

- If any part of the transaction fails, undo the changes

```
ChangeRecorder changeRecorder =
  new ChangeRecorder(resourceSet);

try
{
  // modifications within resource set
}
catch (Exception e)
{
  changeRecorder.endRecording().apply();
}
```

Exercise 3:
Change Model & Change Recorder

Agenda

- Introduction
  - EMF in a Nutshell
  - EMF Components
  - The Ecore Metamodel

- Exercise 1: Code Generation, Regeneration and Merge
- Exercise 2: Runtime Framework: Loading & Saving Resources

  – break –

- Exercise 3: Change Model & Change Recorder
- ***Exercise 4: Validation Framework***
- Exercise 5: XMLProcessor & Reflection *(optional)*

- Summary
- What's New in EMF 2.2 for Eclipse 3.2?
- Q&A

- End 4:45pm

Validation Framework

- Model objects validated by external EValidator

```
public interface EValidator
{
  boolean validate(EObject eObject,
                   DiagnosticChain diagnostics, Map Context);

  boolean validate(EClass eClass, EOjbect eObject,
                   DiagnosticChain, diagnostics, Map context);

  boolean validate(EDataType eDataType, Object value,
                   DiagnosticChain diagnostics, Map context);

  ...
}
```

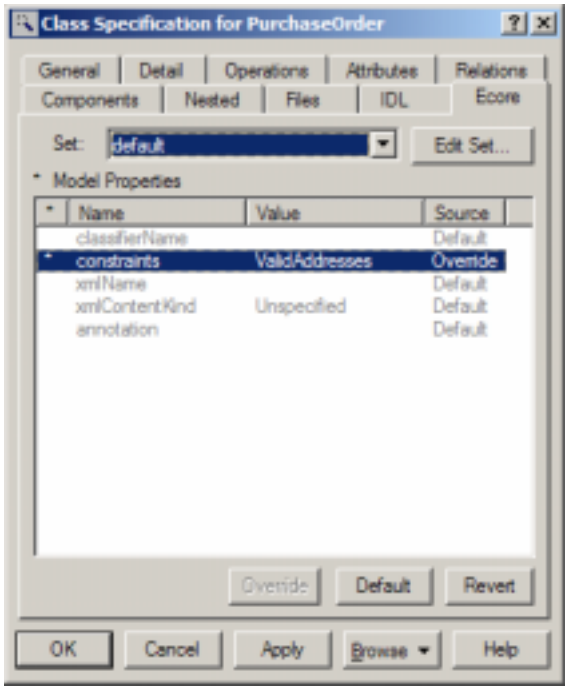
- Details results accumulated as Diagnostics
  - Essentially a non-Eclipse equivalent to IStatus
  - Records severity, source plug-in ID, status code, message, other arbitrary data, and nested children

Invariants and Constraints

- Invariant
  - Defined directly on the class, as an operation with <<inv>> stereotype
  - Stronger statement about validity than constraint

PurchaseOrder
shipTo : String billTo : String
<<inv>> validAddresses()

- Constraint
  - Externally defined for the class via a method on the validator



Generated EValidator Implementations

- Generated for each package that defines invariants or constraints
- Dispatches validation to type-specific methods
- For classes, a validate method is called for each invariant and constraint
  - Method body must be hand coded for invariants and named constraints

Schema-Based Constraints

- In XML Schema, named constraints are defined via annotations:

```
<xsd:annotation>
  <xsd:appinfo source="http://www.eclipse.org/emf/2002/Ecore"
    ecore:key="constraints">VolumeDiscount</xsd:appinfo>
</xsd:annotation>
```

- Also, constraints can be defined as facets on simple types, and no additional coding is required
  - Constraint method implementation generated

```
<xsd:simpleType name="SKU">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="\d{3}-[A-Z]{2}"/>
  </xsd:restriction>
</xsd:simpleType>
```

Framework EValidator Implementations

- EObjectValidator validates basic EObject constraints:
  - Multiplicities are respected
  - Proxies resolve
  - All referenced objects are contained in a resource
  - Data type values are valid
- Used as base of generated validators and for packages without

Framework EValidator Implementations

- Diagnostician walks a containment tree of model objects, dispatching to package-specific validators
  - Diagnostician.validate() is the usual entry point
  - Obtains validators from its EValidator.Registry

```
Diagnostician validator = Diagnostician.INSTANCE;
Diagnostic diagnostic = validator.validate(order);

if (diagnostic.getSeverity() == Diagnostic.ERROR) {
  // handle error
}

for (Iterator i = diagnostic.getChildren().iterator();
     i.hasNext(); )
{
  Diagnostic child = (Diagnostic)i.next();
  // handle child diagnostic
}
```

Exercise 4:
Validation Framework

Agenda

- Introduction
  - EMF in a Nutshell
  - EMF Components
  - The Ecore Metamodel

- Exercise 1: Code Generation, Regeneration and Merge
- Exercise 2: Runtime Framework: Loading & Saving Resources

  – break –

- Exercise 3: Change Model & Change Recorder
- Exercise 4: Validation Framework
- ***Exercise 5: XMLProcessor & Reflection (optional)***

- Summary
- What's New in EMF 2.2 for Eclipse 3.2?
- Q&A

- End 4:45pm

XML Processor

- New in EMF 2.2 (from M2)
- Simplified API for loading and saving XML
  - Handles resource set, registries, etc. under the covers
- Can automatically create a dynamic Ecore representation of a schema
  - Load/save instance documents without generating code
  - Manipulate using reflective API

```
URI schemaURI = ...
String instanceFileName = ...

XMLProcessor processor = new XMLProcessor(schemaURI);
Resource resource = processor.load(instanceFileName);

EObject documentRoot = (EObject)resource.getContents().get(0);
```

Exercise 5: XMLProcessor & Reflection

Agenda

- Introduction
  - EMF in a Nutshell
  - EMF Components
  - The Ecore Metamodel

- Exercise 1: Code Generation, Regeneration and Merge
- Exercise 2: Runtime Framework: Loading & Saving Resources

  – break –

- Exercise 3: Change Model & Change Recorder
- Exercise 4: Validation Framework
- Exercise 5: XMLProcessor & Reflection *(optional)*

- ***Summary***
- What's New in EMF 2.2 for Eclipse 3.2?
- Q&A

- End 4:45pm

Summary

- EMF is low-cost modeling for the Java mainstream
- Boosts productivity and facilitates integration
- Mixes modeling with programming to maximize the effectiveness of both

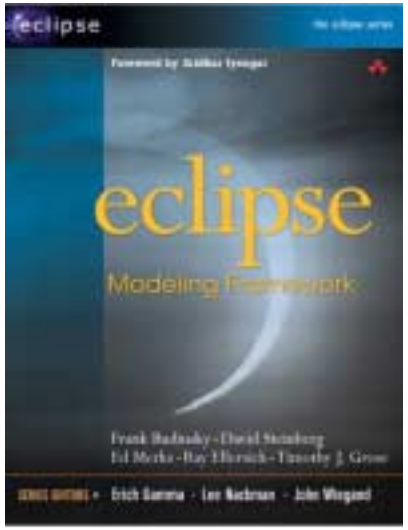
Summary (cont)

- EMF provides:
  - A model (Ecore) with which your models can be built
    - Model created from Rose, XML Schema, or annotated Java interfaces
  - Generated Java code
    - Efficient and straight forward
    - Code customization preserved
  - Persistence and Serialization
    - Default is XMI (XML metadata interchange) but can be overridden
    - Serialized to resources
  - Model change notification is built in
    - Just add observers (listeners) where needed
  - Reflection and Dynamic EMF
    - Full introspection capability

Sneak Peak: What's New in EMF 2.2?

- Recent Enhancements:
  - XMLProcessor (Exercise 5)
- Plan Items [Bugzillas]:
  - Cross resource containment [105937]
  - JMerger to move from JDOM to AST [78076]
  - Enhancements to the Java Emitter Template engine [105966]
  - Improve XSD generation and (Model Exporter) [104893]
  - Define constraints on valid Ecore model and diagnose violations [75933]
  - Improve code generation error reporting and handling [104727]
- Community Involvement:
  - EMFT: Incubating new EMF Technology Projects
- For more, see: http://www.eclipse.org/emf/docs.php#plandocs

Resources



- EMF Project Website
  - http://www.eclipse.org/emf/
  - Overviews, tutorials, newsgroup, Bugzillas
- Eclipse Modeling Framework by Frank Budinsky et al.
  - Addison-Wesley; 1st edition (August 13, 2003)
  - ISBN: 0131425420.
- This presentation, and its accompanying Eclipse project files, can be downloaded here:
  - http://eclipse.org/emf/docs/presentations/CASCON/

Any questions?