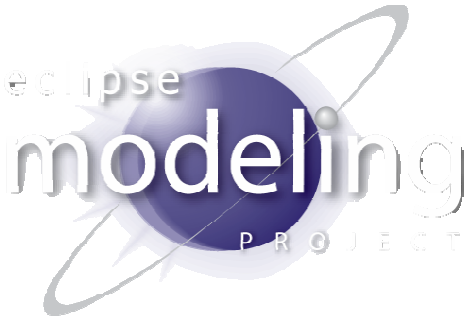
EclipseWorld 2007

November 6-8, 2007

Fundamentals of the Eclipse Modeling Framework

Dave Steinberg
IBM Rational Software
Toronto, Canada
EMF Project Committer



Goal

Learn about modeling and how the Eclipse Modeling Framework can help you write your application in significantly less time, simply by leveraging the data model you've probably already defined (although you might not know it yet)

Agenda

- What is Modeling?
- Defining a Model with EMF
- EMF Architecture
- Code Generation
- Programming with EMF
- Relational Persistence
- Demo
- Summary

Model Driven Architecture (MDA)

- A software architecture proposed by the OMG (Object Management Group)
- Application specified in high-level, Platform Independent Model (PIM)
- Transformation technologies used to convert PIM to Platform Specific Model (PSM), implementation code
- Includes several open modeling standards:
  - UML (Unified Modeling Language)
  - MOF (Meta-Object Facility)
  - XMI (XML Metadata Interchange)
  - CWM (Common Warehouse Model)

MDA – Are We There Yet?

- Doubts exist about the ability of MDA to deliver on its promises:
  - Ambitiousness of vision
  - Model expressiveness vs. complexity
  - Availability of implementations (“vaporware”)

Enter EMF!

- EMF is a simple, pragmatic approach to modeling:
  - Allows us to generate some of the code that we write over and over, paving the way for more complex systems (including more ambitious MDA tools)
  - Models are simple, but meant to be mixed with hand-written code
  - It's real, proven technology (since 2002)

Model Driven Development with EMF

- Contrary to the belief of many programmers, modeling is useful for more than just documentation
- Almost every program we write manipulates some data model
  - Defined using UML, XML Schema, some other definition language, or implicitly in Java™
- EMF aims to extract this intrinsic “model” and generate some of the implementation code
  - Can be a tremendous productivity gain

What Does EMF Provide?

- Ecore Metamodel
  - Model of models, with which other models are defined
- Tools for importing models and generating code
- Runtime model support
  - Reflection, notification, dynamic definition
- Persistence framework
  - XML/XMI resource implementations
- Validation framework
- Change model and change recorders
- EMF.Edit
  - UI-independent viewing and editing support
  - Integrated workbench or RCP model editors
- And more all the time…

EMF and Java 5.0

- EMF 2.3 began exploiting the capabilities of Java 5.0
  - Generics
  - Typesafe enums
  - Annotations
  - Enhanced for loop
  - Autoboxing
- EMF 2.3 and later provide 5.0-targeted runtime that only runs on a Java 5.0 or later VM
- The code generator can also produce application code that runs on a 1.4 VM against the 2.2 runtime
- This is a recognition that Java 5.0 is the way forward, but the transition need not be painful for the user

EMF at Eclipse.org

- Foundation for the Eclipse Modeling Project
  - EMF project incorporates core and additional mature components: Query, Transaction, Validation
  - EMF Technology project incubates complementary components: CDO, Teneo, Compare, Search…
  - Other projects build on EMF: Graphical Modeling Framework (GMF), Model Development Tools (MDT), Model to Model Transformation (M2M), Model to Text Transformation (M2T) …
- Other uses: Web Tools Platform (WTP), Test and Performance Tools Platform (TPTP), Data Tools Platform (DTP), Business Intelligence and Reporting Tools (BIRT)…
- Large open source user community

Agenda

- What is Modeling?
- Defining a Model with EMF
- EMF Architecture
- Code Generation
- Programming with EMF
- Relational Persistence
- Demo
- Summary

What is an EMF "Model"?

- Specification of an application's data
  - Object attributes
  - Relationships (associations) between objects
  - Operations available on each object
  - Simple constraints (e.g. multiplicity) on objects and relationships
- Essentially, the Class Diagram subset of UML

Model Sources

- EMF models can be defined in (at least) three ways:
  1. Java Interfaces
  2. UML Class Diagram
  3. XML Schema
- Choose the one matching your perspective or skills and EMF can create the others, as well as the implementation code

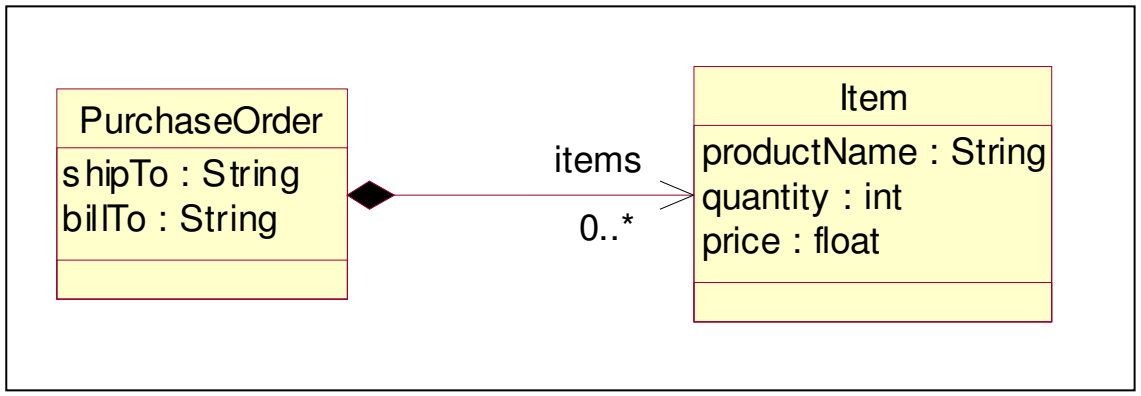
Java Interfaces

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

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

- Classes can be defined completely by a subset of members, supplemented by annotations

UML Class Diagram



- Built-in support for Rational Rose® (.mdl)
- UML2 support available with UML2 project

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.example.com/SimplePO"
            xmlns:PO="http://www.example.com/SimplePO">
  <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>
</xsd:schema>
```

Unifying UML, XML and Java

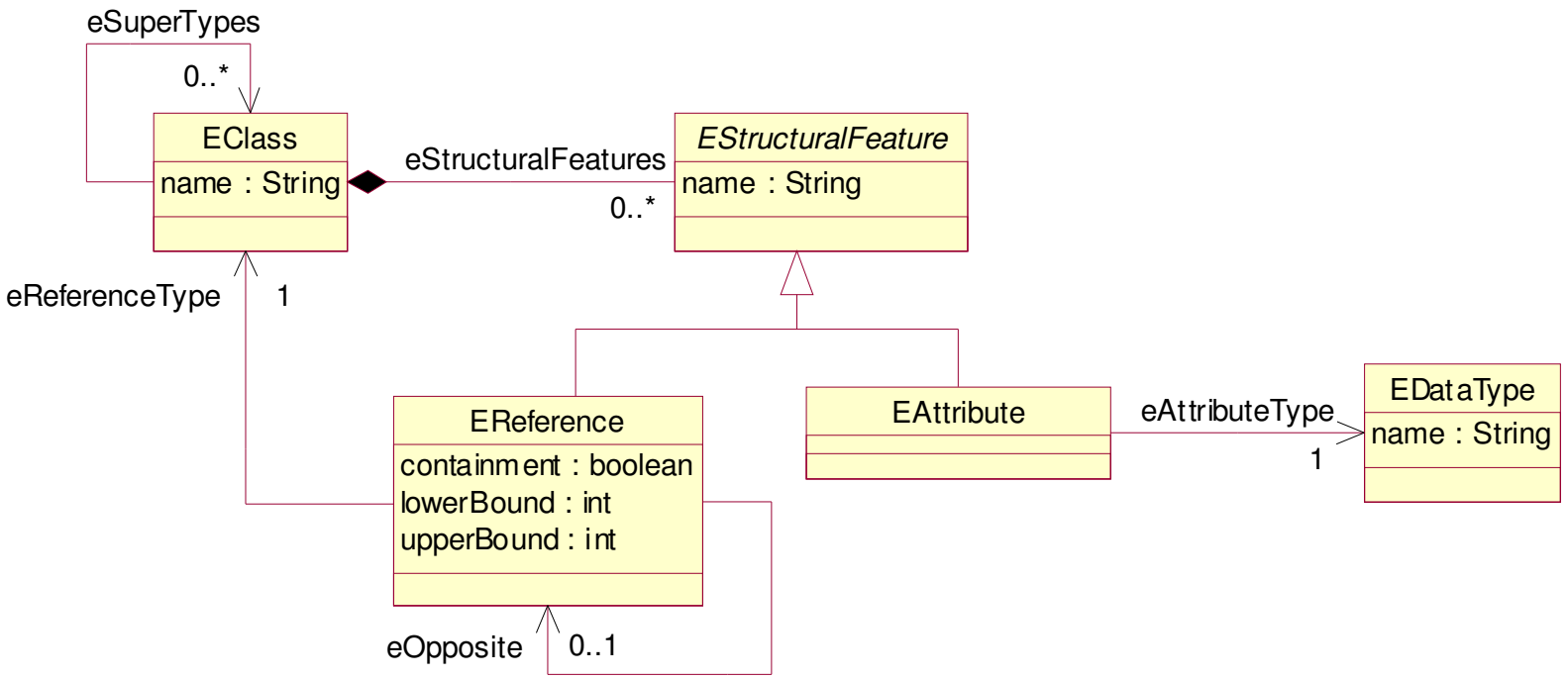
- All three forms provide the same information
  - Different visualization/representation
  - The application's data "model" or structure
  - Model importers can be added for different model representations (e.g. RDB Schema)
- From a model definition, EMF can generate…
  - Java implementation code, including UI
  - XML Schemas
  - Eclipse "plug-in" artifacts

Agenda

- What is Modeling?
- Defining a Model with EMF
- EMF Architecture
- Code Generation
- Programming with EMF
- Relational Persistence
- Demo
- Summary

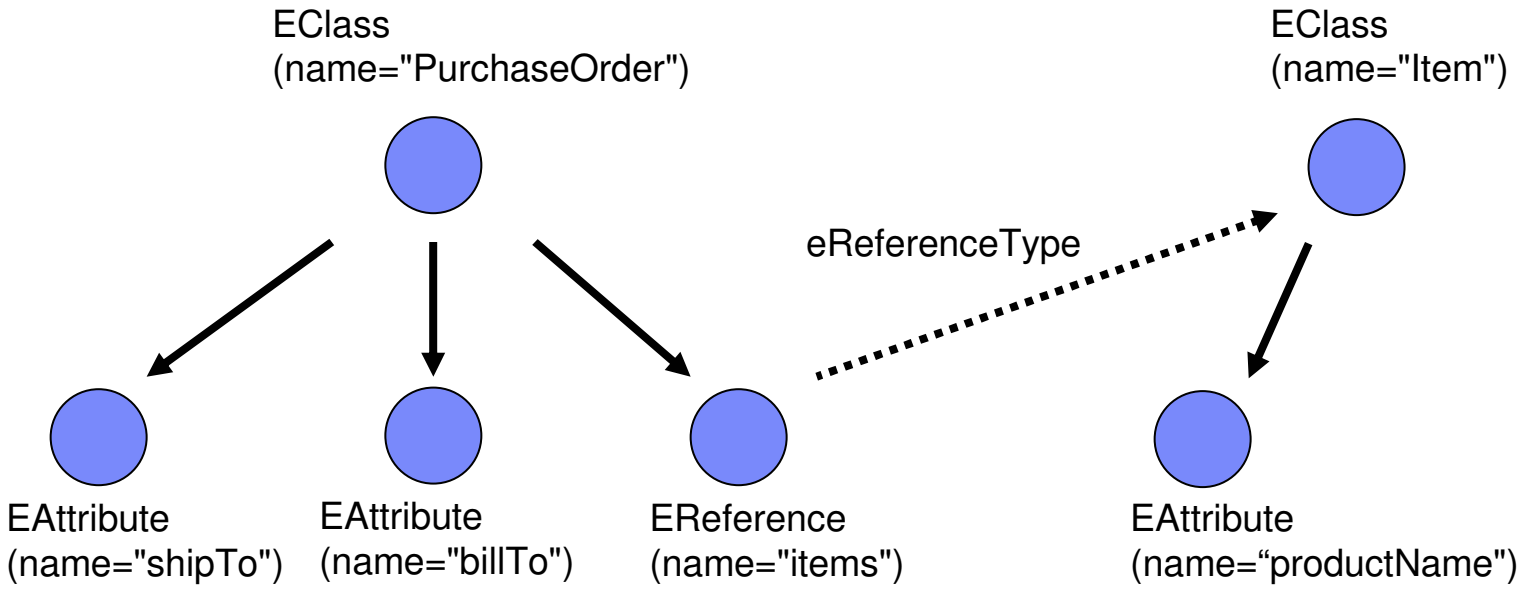
Ecore

- EMF's metamodel (model of a model)



Ecore

- Application models (e.g. purchase order model) are instances of Ecore



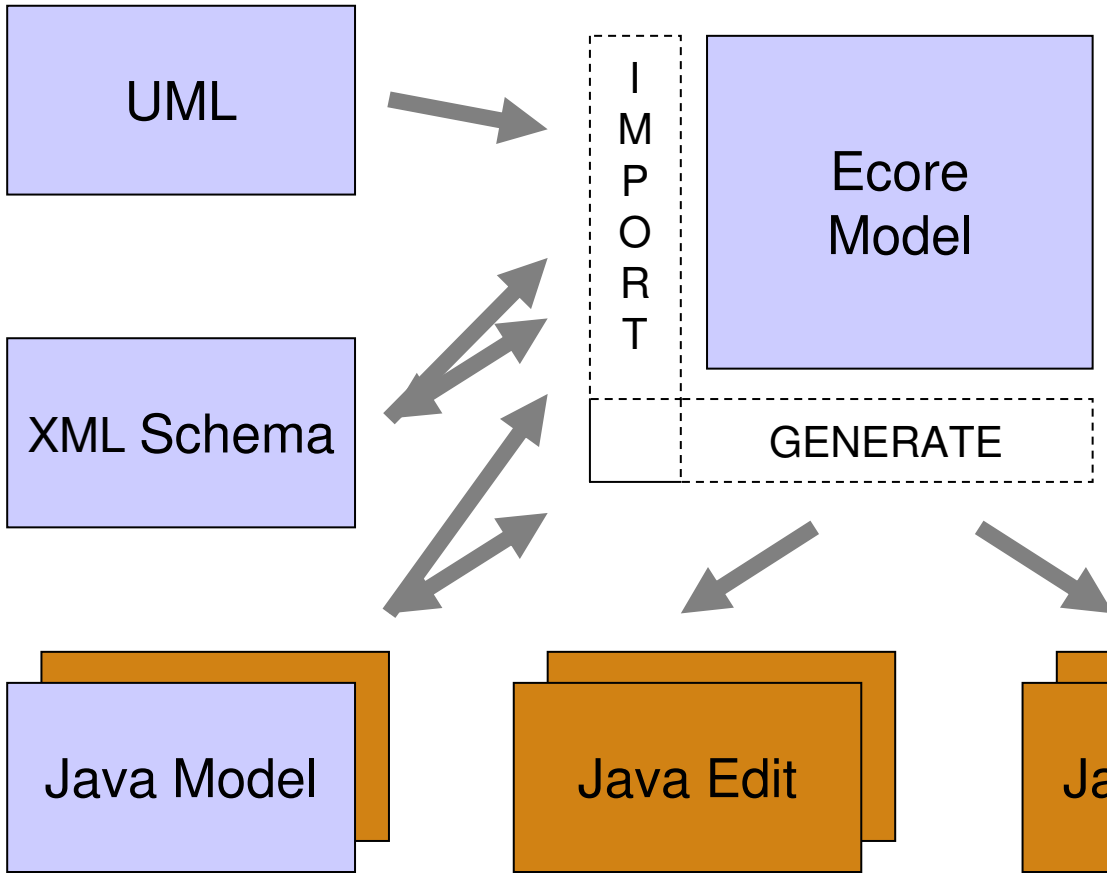
Ecore

- Persistent format is XMI

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

- Alternate format is EMOF (Essential MOF) XMI

Model Import and Generation



Generator features:

- Customizable, JSP-like templates (JET)
- JDT-integrated, command-line or Ant
- Full support for regen and merge

Agenda

- What is Modeling?
- Defining a Model with EMF
- EMF Architecture
- Code Generation
- Programming with EMF
- Relational Persistence
- Demo
- Summary

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<Item> getItems();
}
```

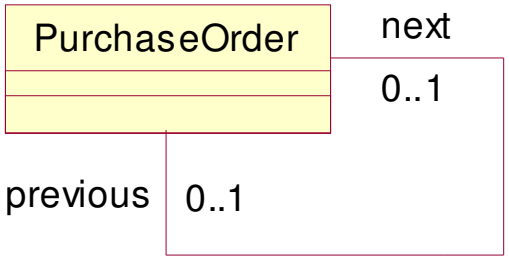
```
public class PurchaseOrderImpl extends EObjectImpl
  implements PurchaseOrder
{
  ...
}
```

Feature Accessors: Change Notification

- Efficient notification from set methods
  - Observer design pattern

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

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

Bidirectional Reference Handshaking



Bidirectional reference imposes invariant:

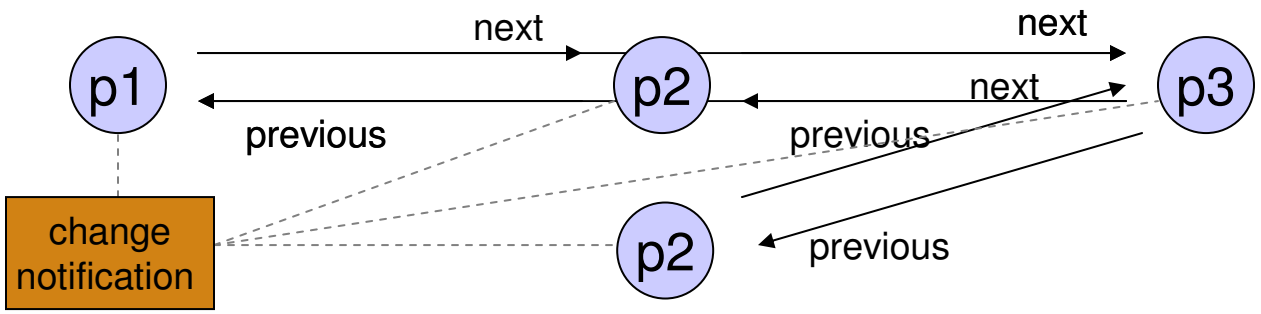
po.getNext().getPrevious() == po

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

Bidirectional Reference Handshaking

- Multi-step procedure implemented using InternalEObject methods:
  - eInverseRemove()
  - eInverseAdd()
- Framework list implementations and generated setters directly invoke them on target objects as needed
- Notfifications accumulated using a NotificationChain and fired off at the end
  - Not used to implement the handshaking

Bidirectional Reference Handshaking




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

Typesafe Enums

- Typesafe enum pattern for enumerated types

```
public enum Status implements Enumerator
{
  PENDING(0, "Pending", "Pending"),
  BACK_ORDER(1, "BackOrder", "BackOrder"),
  COMPLETE(2, "Complete", "Complete");

  public static final int PENDING_VALUE = 0;
  public static final int BACK_ORDER_VALUE = 1;
  public static final int COMPLETE_VALUE = 2;

  private final int value;
  private final String name;
  private final String literal;

  private Status(int value, String name, String literal)
  { ... } ...
}
```

Reflective EObject API

- All EMF classes implement EObject interface
- Provides an efficient API for manipulating objects reflectively
  - Used by framework (e.g. persistence framework, copy utility, editing commands)
  - Key to integrating EMF-based tools and applications

```
public interface EObject
{
  Object eGet(EStructuralFeature sf);
  void eSet(EStructuralFeature sf, Object val);
}
```

Reflective EObject API

- Efficient generated switch-based implementation of reflective methods

```
public Object eGet(int featureID, ...)
{
  switch (featureID)
  {
    case POPackage.PURCHASE_ORDER__ITEMS:
      return getItems();
    case POPackage.PURCHASE_ORDER__SHIP_TO:
      return getShipTo();
    case POPackage.PURCHASE_ORDER__BILL_TO:
      return getBillTo();
    ...
  }
  ...
}
```

Other Generated Artifacts

- Model
  - Package (metadata)
  - Factory
  - Switch utility
  - Adapter factory base
  - Validator
  - Custom resource
  - XML Processor
- Edit
  - Item providers
  - Item provider adapter factory
- Editor
  - Model Wizard
  - Editor
  - Action bar contributor
  - Advisor (RCP)
- Tests
  - Test cases
  - Test suite
  - Stand-alone example
- Manifests, plug-in classes, properties, icons...

Regeneration and Merge

- The EMF generator is a merging generator

```
/**
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated
 */
public String getName()
{
  return name;
}
```

- @generated elements are replaced/removed
- To preserve changes, mark @generated NOT

Regeneration and Merge

- Generated methods can be extended through redirection

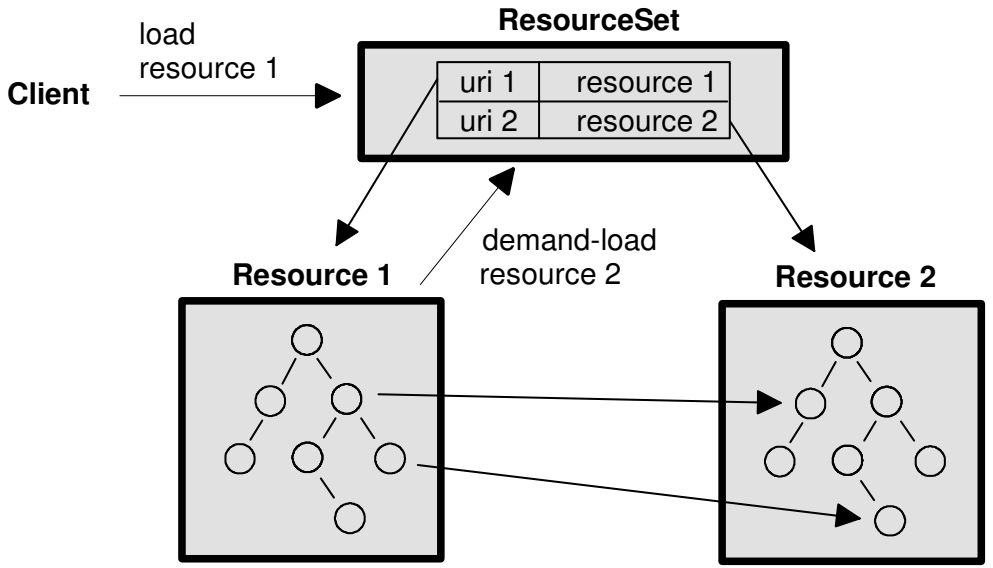
```
public String getName()
{
  return format(getNameGen());
}

/**
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated
 */
public String getNameGen()
{
  return name;
}
```

Agenda

- What is Modeling?
- Defining a Model with EMF
- EMF Architecture
- Code Generation
- Programming with EMF
- Relational Persistence
- Demo
- Summary

Model Persistence

- Persisted data is referred to as a resource
- Data can be spread out among a number of resources, becoming a resource set
- Proxies represent referenced objects in other resources



Resource Implementations

- EMF provides a generic XML resource implementation

```
resource = resourceSet.createResource(..."p1.xml"...);
resource.getContents().add(p1);
resource.save(...);
```

p1.xml:

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

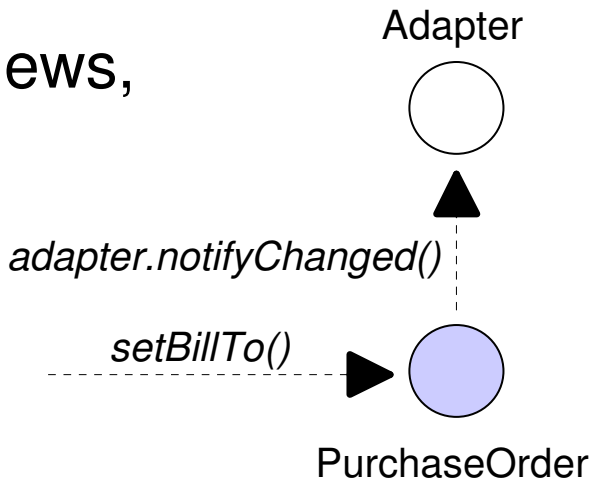
- Other resource implementations are possible

Proxy Resolution and Demand Load


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

Change Notification

- Every EObject is also a notifier
  - Sends notification whenever an attribute or reference is changed
  - Observers can update views, dependent objects
  - Observers are also adapters




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

Reflection

- Setting an attribute using generated API:

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

- Using reflective API:

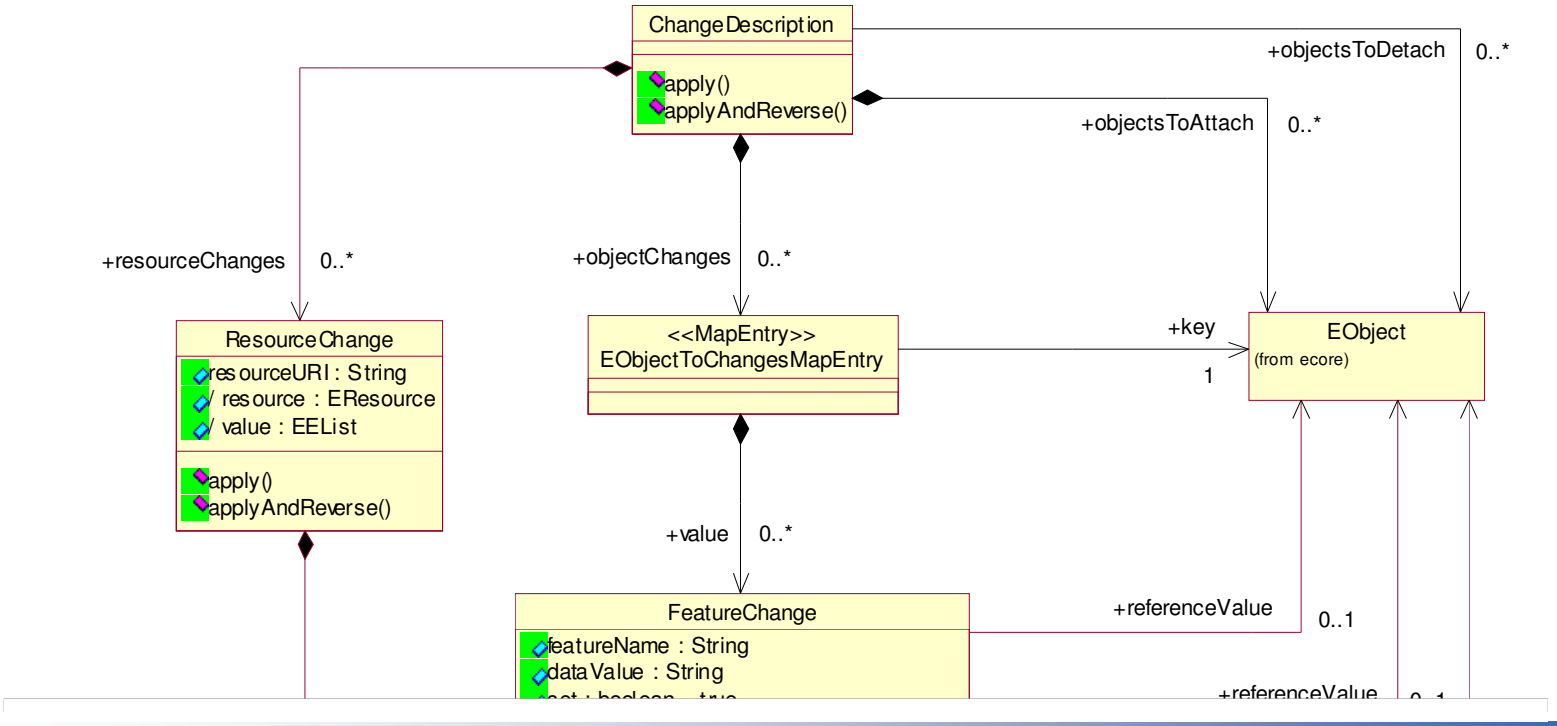
```
EObject po = ...
EClass poClass = po.eClass();
po.eSet(poClass.getEStructuralFeature("billTo"),
        "123 Elm St.");
```

Dynamic EMF

- An Ecore model can also be defined at runtime using Ecore API or loaded from persistent form
  - No generated code required
  - Dynamic implementation of reflective EObject API provides same runtime behavior as generated code
  - Can create dynamic subclasses of generated classes
- The framework treats all model instances the same, whether generated or dynamic

Change Recording

- The Change model represents changes to any EMF model



Change Recording

- Change recorder
  - Adapter that populates change model based on notifications, to describe reverse delta
  - Provides transaction capability

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

Validation

- Models can define named constraints and invariants for batch validation
  - Invariant: defined directly on class, as <<inv>> operation
  - Constraint: externally defined via a validator
- In either case, body must be hand-coded
- Diagnostician walks a containment tree of objects, dispatching to package-specific validators

```
Diagnostianostic diagnostic =
  Diagnostician.INSTANCE.validate(eObject);
if (diagnostic.getSeverity() == Diagnostic.ERROR)
  ...
```

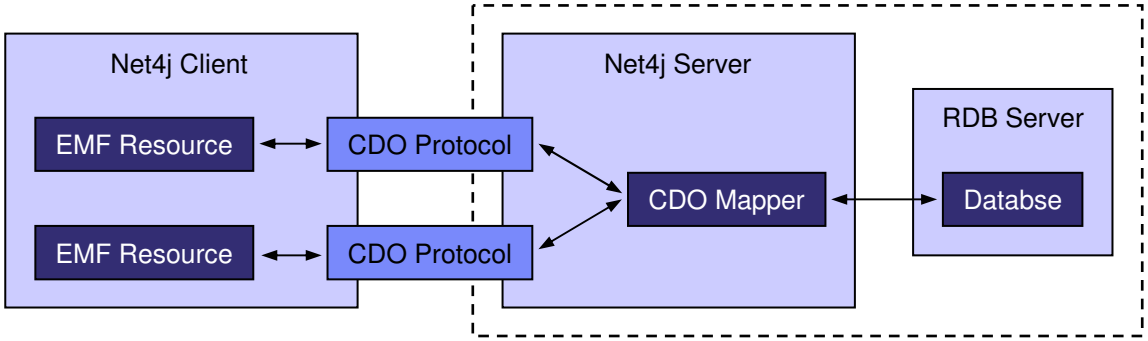
Agenda

- What is Modeling?
- Defining a Model with EMF
- EMF Architecture
- Code Generation
- Programming with EMF
- Relational Persistence
- Demo
- Summary

Relational Persistence

- EMF Technology (EMFT) project provides two emerging relational persistence technologies
  - Connected Data Objects (CDO): custom client-server based persistence framework for distributed shared EMF models
  - Teneo: EMF integration with existing persistence frameworks
- See http://www.eclipse.org/modeling/emft/

Connected Data Objects (CDO)



- Flexible client-server architecture
- Proprietary binary protocol for object operations between CDO client and server, over socket or embedded transport
- CDO server does custom object/relational mapping, uses JDBC for backend storage

Connected Data Objects (CDO)

- Change notification broadcasting to enable "hot-connected" clients
- O/R Mapping
  - Default mapping automatically generated into mapping file
  - Can be edited to change table names for classes, and column names and types for attributes
- Requires a generated EMF model at client, in which classes extend CDOPersistent

Teneo

- Provides EMF integration with existing object/relational mapping frameworks
  - Hibernate
  - JPOX/JDO
- Documentation at http://www.elver.org/

Teneo Hibernate O/R Mapping

- Hibernate data store sets up mapping and creates tables in the database

```
HbDataStore ds =
  HbHelper.INSTANCE.createRegisterDataStore("POStore");
ds.setEPackages(new EPackage[] { POPackage.eINSTANCE });
ds.initialize();
```

- Automatic mapping can be controlled via JPA/EJB3 annotations on the Ecore model or in a separate XML document
- Custom mapping file can be specified instead

Teneo Hibernate Integration

- Data store provides session factory, which can be used for typical Hibernate transactions

```
SessionFactory sf = ds.getSessionFactory();

Session session = sf.openSession();
session.beginTransaction();

List orders =
  session.createQuery("FROM PurchaseOrder").list();
((PurchaseOrder)orders.get(0)).setShipTo("123 Elm St.");

session.getTransaction().commit();
session.close();
```

- EJB3 EntityManager API also available

Teneo Hibernate Resource

- Hibernate resource implementation provides familiar EMF API

```
URI uri = URI.createURI("hibernate://?dsname=POStore");
Resource resource = resourceSet.createResource(uri);
resource.getContents().add(po);
resource.save(null);
```

- Handles sessions and transactions under the covers

Agenda

- What is Modeling?
- Defining a Model with EMF
- EMF Architecture
- Code Generation
- Programming with EMF
- Relational Persistence
- Demo
- Summary

Agenda

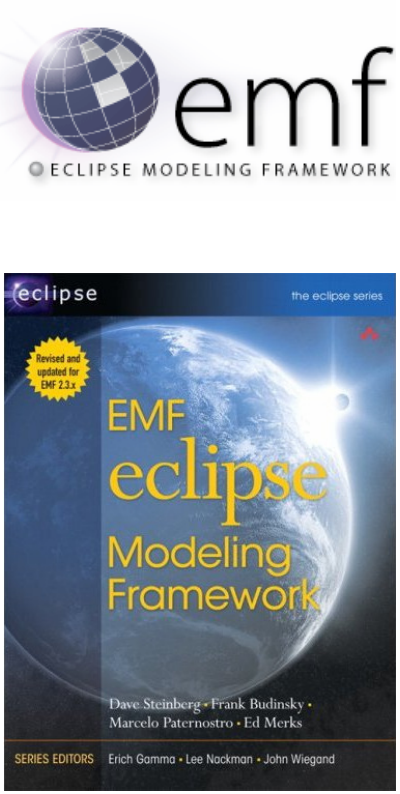
- What is Modeling?
- Defining a Model with EMF
- EMF Architecture
- Code Generation
- Programming with EMF
- Relational Persistence
- Demo
- Summary

Summary

- EMF is low-cost modeling for the Java mainstream
- Leverages the intrinsic model in an application
  - No high-level modeling tools required
- Mixes modeling with programming to maximize the effectiveness of both
- Boosts productivity and facilitates integration
- The foundation for model-driven development and data integration in Eclipse

Resources

- EMF documentation in Eclipse Help
  - Overviews, tutorials, API reference
- EMF project Web site
  - http://www.eclipse.org/modeling/emf/
  - Downloads, documentation, FAQ, newsgroup, Bugzilla, wiki
- *Eclipse Modeling Framework*, by Frank Budinsky et al.
  - ISBN: 0131425420
  - Second edition coming soon…



Legal Notices

IBM, Rational, and Rational Rose are registered trademarks of International Business Machines Corp. in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.