

Everything you always wanted to do with EMF*

*But were afraid to ask

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

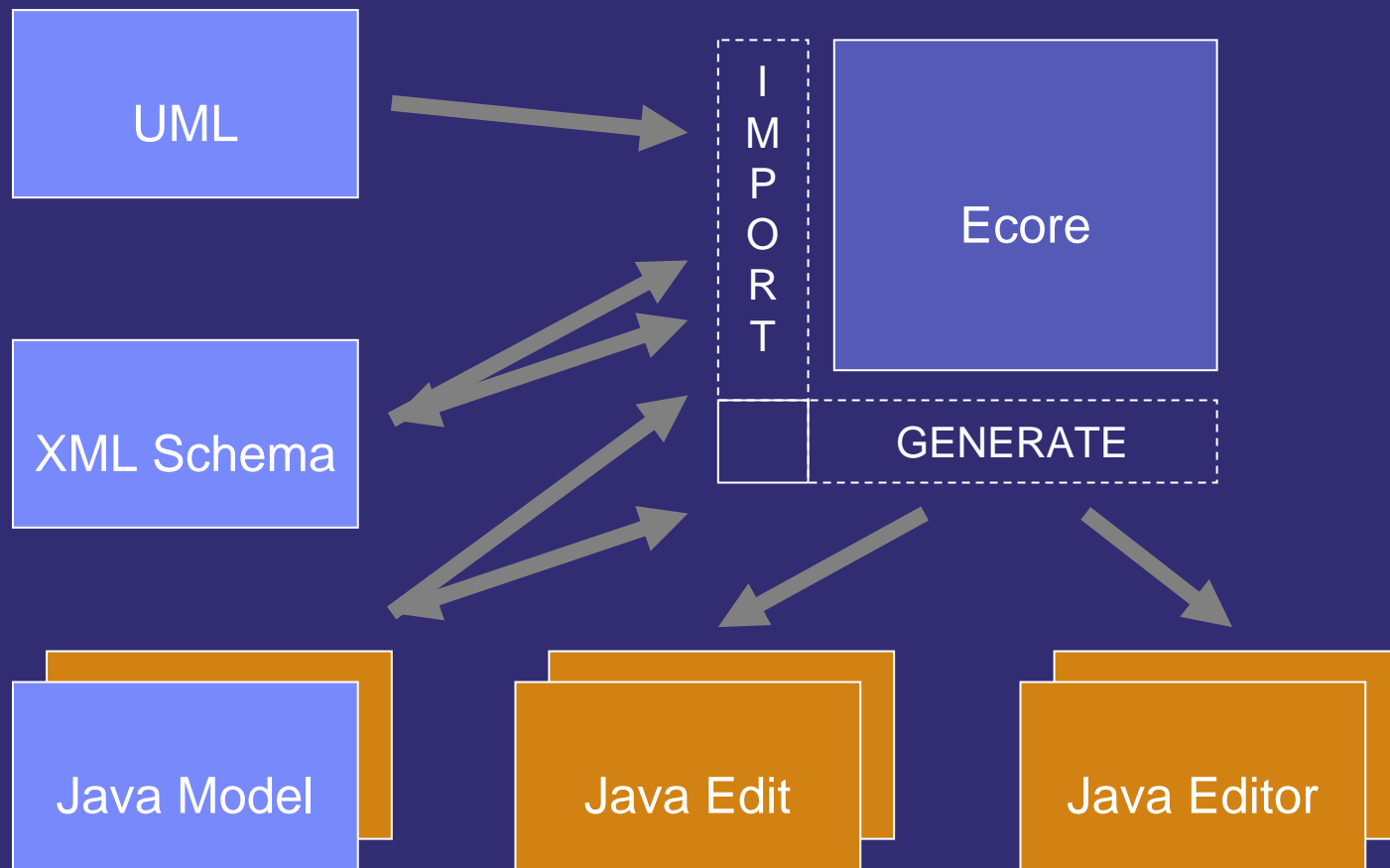
Topics

- Introduction to EMF
- Simple Customizations to Generated Code
- Extending Behavior with Adapters
- Validating Model Data
- Customizing XML Persistence

Introduction to the Eclipse Modeling Framework

- Low-cost modeling for the Java mainstream
- Leverages the intrinsic “model” in an application
 - Java interfaces or XML Schema
 - No high-level modeling tool 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”

Model Conversion and Code Generation



Generated Code

- For each modeled class...

- An interface:

```
public interface PurchaseOrder extends EObject
{
    EList getItems();
    String getComment();
    void setComment(String value);
    ...
}
```

- An implementation:

```
public class PurchaseOrderImpl extends EObjectImpl
    implements PurchaseOrder
{
    protected EList items = null;
    ...
}
```

Reflective Interface for Model Objects

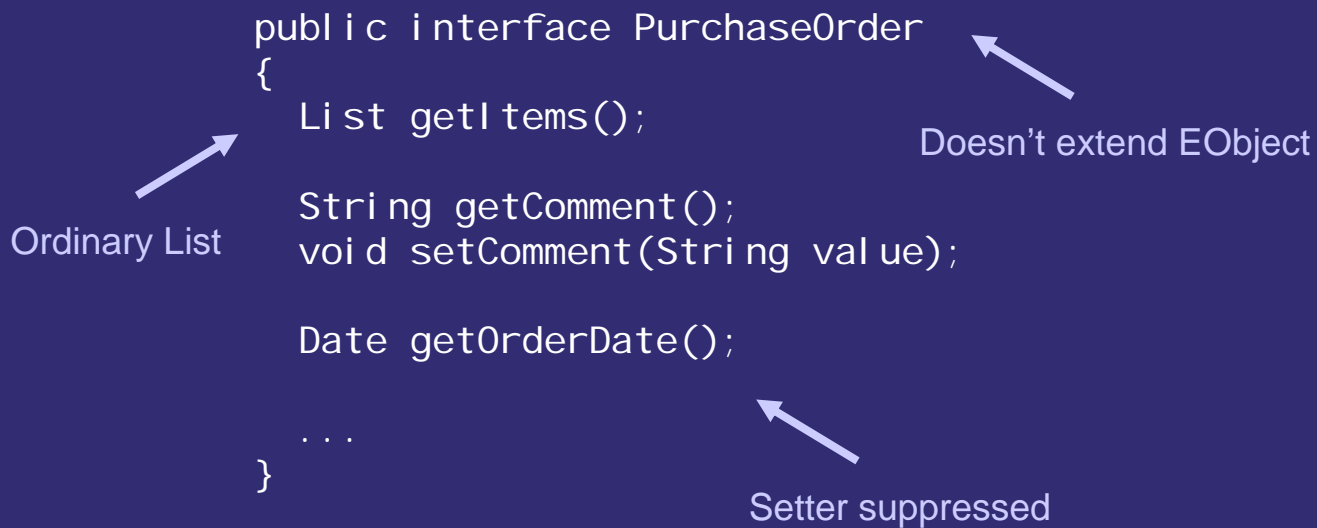
- All model classes derive from EObject:

```
public interface EObject extends Notifier
{
    EClass eClass();
    Resource eResource();
    EObject eContainer();
    EReference eContainmentFeature();
    boolean eIsProxy();
    Object eGet(EStructuralFeature feature, boolean resolve);
    void eSet(EStructuralFeature feature, Object newValue);
    boolean eIsSet(EStructuralFeature feature);
    void eUnset(EStructuralFeature feature);
    ...
}
```

- This reflective EObject API lets us write generic code, like in EMF's editing, persistence, validation frameworks

Customizing Interfaces

- In some applications, you may prefer to have your modeled classes present a simpler interface:



Customizing Interfaces: Taking Out the “E”

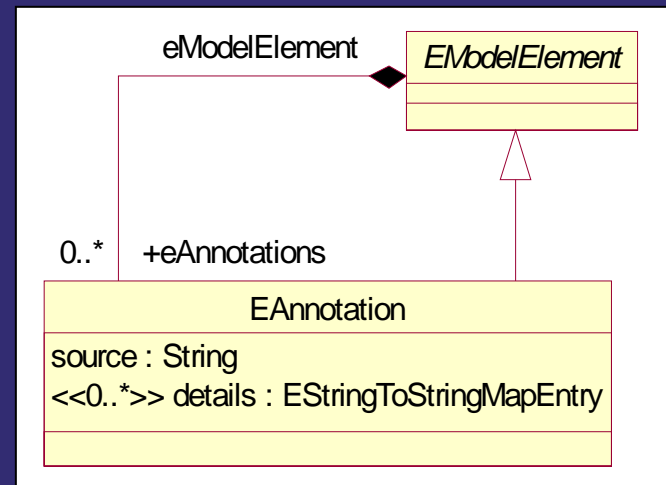
- Generated interfaces customized via generator options
 - Root Extends Interface
 - Suppress EMF Types (EList, EMap, EObject)

Property	Value
[-] Model Class Defaults	
Root Extends Class	org.eclipse.emf.ecore.impl.EObjectImpl
Root Extends Interface	org.eclipse.emf.ecore.EObject
Root Implements Interface	
Static Packages	
[-] Model Feature Defaults	
Feature Map Wrapper Class	
Feature Map Wrapper Interface	
Feature Map Wrapper Internal Interface	
Suppress EMF Types	false

- Generated classes must still implement reflective EObject API

Customizing Interfaces: Suppressing Accessors

- Individual accessors (get, set, isSet, unset) can be suppressed in generated interfaces
 - Modeled via EAnnotations on features, with source of “<http://www.eclipse.org/emf/2002/GenModel>”
- Only affects the interface; features still fully available reflectively



Demo: Customizing Generated Interfaces

- Annotate purchase order interface to suppress a setter
- Set generator options to suppress EMF base class and types
- Editor and serialization still work, via reflective API

Notifiers and Observers

- In EMF, every model object is a change notifier:

```
public interface Notifier
{
    EList eAdapters();
    boolean eDeliver();
    void eSetDeliver(boolean deliver);
    void eNotify(Notification notification);
}
```

- An observer is registered to receive notifications:

```
item.eAdapters().add(itemObserver);
```

Notifications

- Notifications describe the change that occurred:

```
public interface Notification
{
    Object getNotifier();
    int getEventType();
    int getFeatureID(Class expectedClass);
    Object getFeature();
    Object getOldValue();
    Object getNewValue();
    boolean wasSet();
    boolean isTouch();
    boolean isReset();
    int getPosition();
    boolean getOldBooleanValue();
    boolean getNewBooleanValue();
    ...
}
```

Example: Simple Observer

- ItemChangeCounter
 - An observer that counts changes to features of an Item
 - Extends AdapterImpl, the framework base class for adapters/observers
 - Handles notifications in notifyChanged()
 - Tests isTouch() to ignore non-changes
 - Switches on feature ID for efficiency
 - Singleton instance is added to model objects

Adapters

- In addition to receiving notifications, observers in EMF are also adapters:

```
public interface Adapter
{
    void notifyChanged(Notification notification);
    Notifier getTarget();
    boolean isAdapterForType(Object type);
    ...
}
```

- Adapters can be used to extend the behavior of the objects they are attached to.

Adapter Factories

- Adapter factories are used to obtain an adapter providing a particular type of behavior extension:

```
public interface AdapterFactory
{
    boolean isFactoryForType(Object type);
    Object adapt(Object object, Object type);
    Adapter adapt(Notifier target, Object type);
    ...
}
```

- By convention, the type is often the interface that the adapter implements:

```
POAdapter adapter =
    (POAdapter)poAdapterFactory.adapt(order, POAdapter.class);
```

Example: Adapter and Adapter Factory

- **ChangeCounter**
 - Simple interface for counting and comparing changes
- **ChangeCounterAdapter**
 - Adapter-based implementation that `isAdapterFor(ChangeCounter.class)`
- **ChangeCounterAdapterFactory**
 - Singleton adapter factory that creates a `ChangeCounterAdapter` as an adapter of this type for any model object

Content Adapters

- EContentAdapter
 - Framework base class for adapters that respond to changes from a complete containment tree of model objects
 - Automatically adds and removes itself as objects are added to and removed from the tree
 - Subclasses' override for `notifyChanged()` must call `super.notifyChanged()`

Example: Content Adapter

- ContentChangeCounter
 - Extends EContentAdapter to count the number of changes to a purchase order and its contents

Validation Framework

- Model objects validated by external EValidator:

```
public interface EValidator
{
    boolean validate(EObject eObject,
                    DiagnosticChain diagnostics, Map context);
    boolean validate(EClass eClass, EObject eObject,
                    DiagnosticChain diagnostics, Map context);
    boolean validate(EDatatype eDataType, Object value,
                    DiagnosticChain diagnostics, Map context);
    ...
}
```

- Detailed 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

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
- EObjectValidator validates basic EObject constraints
 - Multiplicities are respected
 - Proxies resolve
 - All referenced objects are contained in a resource
 - Data type values are valid

Generated EValidator Implementations

- Dispatch validation to type-specific methods
- For model objects, one method is called for each...
 - Invariant: defined directly on the class, as an operation with <<inv>> stereotype
 - Constraint: externally defined for the class via the validator method
- In either case, method body must be hand-coded
- Constraints generated, with implementation, for simple type facets defined in XML Schema
- Basic constraints inherited from EObjectValidator

Demo: Constraints and Validation

- Add constraints and an invariant to the purchase order model
 - Write implementations
- Use the generated editor to create and validate an instance of the model

Model Persistence

- Resource is EMF's unit for persistence
 - `Resource.getContents()` returns the list of objects to be persisted as part of the resource
 - `Resource.save()` converts the model to its persistent form and writes it out
 - The complete contents of the resource includes containment trees – objects are always persisted along with their container
- Included Resource implementations:
 - Highly customizable `XMLResource`
 - `XMIResource` for XML Metadata Interchange 2.0

Customizing XML Persistence

- Extended Metadata
 - Annotations added to Ecore elements to customize XML persistence
 - Modeled as EAnnotations with source of “http:///org/eclipse/emf/ecore/util/ExtendedMetaData”
 - Programmatically accessed via ExtendedMetaData interface
- An XMLResource option specifies that extended metadata should be used during save or load:

```
Map options = new HashMap();
options.put(XMLResource.OPTION_EXTENDED_META_DATA,
            ExtendedMetaData.INSTANCE);
resource.save(options);
```

Demo: Customizing XML Persistence

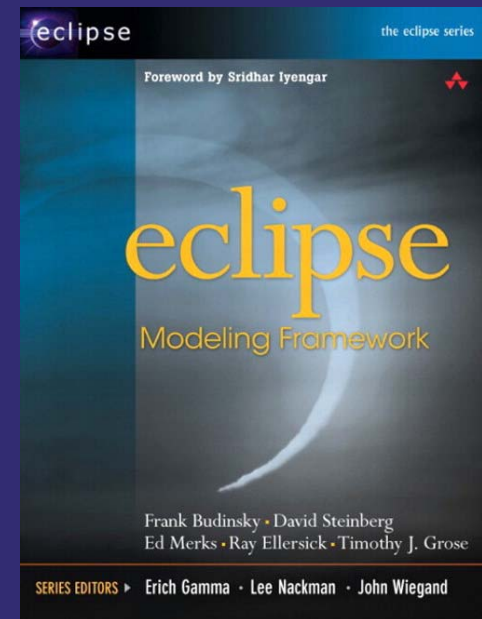
- Add extended metadata annotations to purchase order model
- Generate code
 - PPOPackageImpl adds the annotations to the model that it builds at runtime
 - PPOResourceFactoryImpl creates and initializes resource to use extended metadata
 - Plug-in manifest file globally registers the factory against “.ppo” file extension

Summary

- EMF is low-cost modeling for the Java mainstream, leveraging the intrinsic model in an application
- Generated interfaces can expose simpler API for your model, while uniform EObject API enables integration “under the covers”
- Adapters provide dynamic behavior extension
- Validation framework tests objects against invariants and external constraints
- Extended metadata can customize XML persistence

Additional Information

- EMF documentation in Eclipse Help
 - Overviews, tutorials, API reference
- EMF project Web site
 - <http://www.eclipse.org/emf/>
 - Downloads, documentation, FAQ, newsgroup, Bugzilla, EMF Corner
- *Eclipse Modeling Framework*, by Frank Budinsky et al.
 - ISBN: 0131425420



Everything you always wanted to do with EMF*

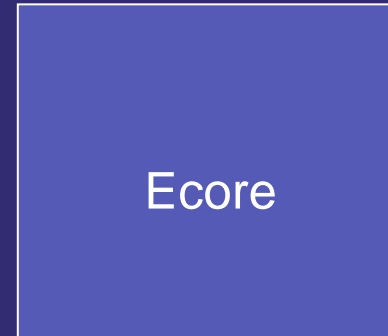
*But were afraid to ask

Additional Slides

EMF Components

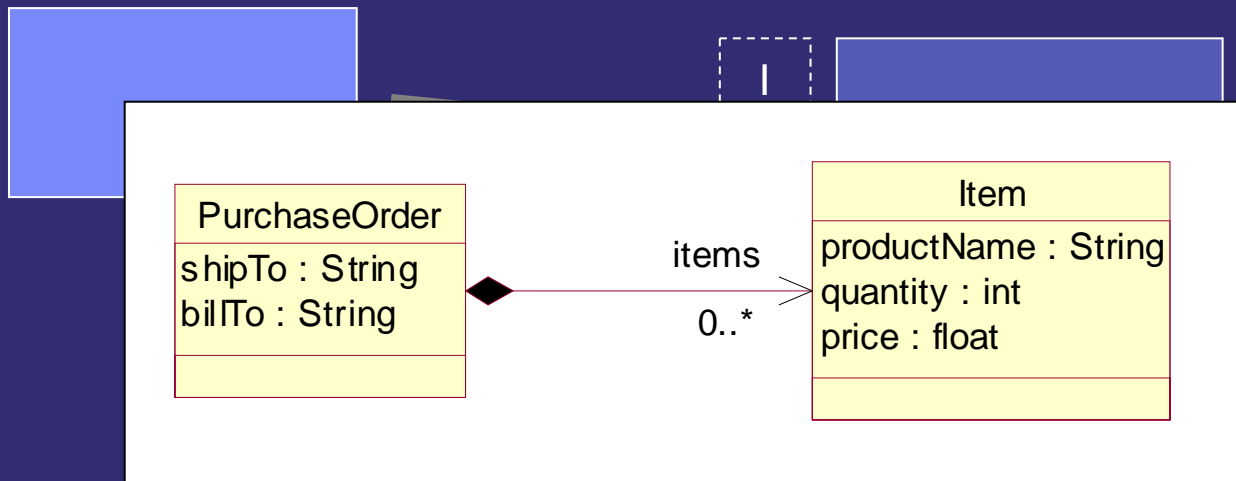
- Ecore, EMF's meta-model
- Model conversion and code generation tools
- Runtime model support
 - reflection, notification, dynamic definition
- Persistence framework
 - XML/XMI resource implementations
- Validation framework
- Change model
- EMF.Edit
 - UI-independent viewing and editing support
 - Integrated workbench or RCP model editors
- And more all the time...

Model Forms: Ecore



- The canonical representation of a model in EMF
- Its persistent form is XML Metadata Interchange (XMI)

Model Forms: UML Class Diagram



Model Forms: Java Interfaces

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

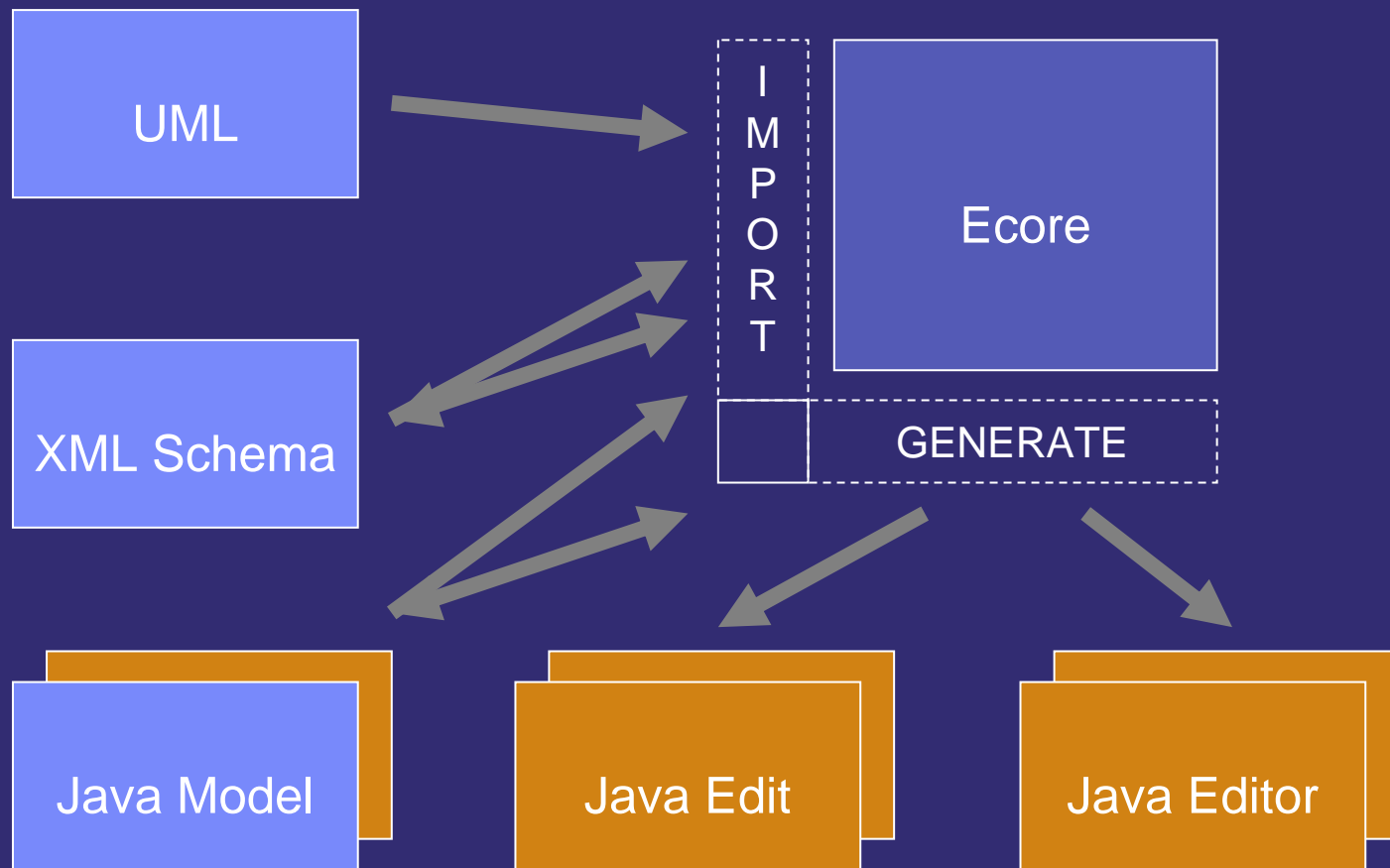
```
public interface Item
{
    String getProductName();
    int getQuantity();
    float getPrice();
}
```

Java Model

Model Forms: 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>
```

Model Conversion and Code Generation



Demo: Generating the Purchase Order Model

- The “Hello world” of EMF
- Generate the purchase order model and an editor for it from various model forms:
 - Java Interfaces
 - XML Schema
- Generate two different kinds of editor:
 - RCP-based application
 - Workbench-integrated editor

Type-Specific Adapter Factories

- EMF generates an adapter factory base class for each package, which creates an adapter by switching and dispatching based on model object type:

```
public class PPOAdapterFactory extends AdapterFactoryImpl
{
    ...
    public Adapter createAdapter(Notifier target)
    {
        return (Adapter)modelSwitch.doSwitch((EObject)target);
    }

    public Adapter createItemAdapter() { return null; }
    public Adapter createUSAddressAdapter() { return null; }
    public Adapter createPurchaseOrderAdapter() { return null; }
    public Adapter createEObjectAdapter() { return null; }
}
```

Example: Type-Specific Adapter Factory

- `WeightedChangeCounter`
 - Subclass of `ChangeCounterAdapter` that counts each change twice
- `WeightedChangeCounterAdapterFactory`
 - Type-specific adapter factory that returns a `WeightedChangeCounterAdapter` for `PurchaseOrders` and a `ChangeCounterAdapter` for everything else