Advanced Features of the Eclipse Modeling Framework

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

Tutorial

Marcelo Paternostro and Kenn Hussey
IBM Rational Software

 March 20, 2006

Agenda

- ***EMF in a Nutshell***

- Working with URIs
- Working with Resources and ResourceSets
- Customizing the Code Generator
- Tuning for Performance and/or Memory Footprint
- Importers and Exporters
- Supporting Backward and Forward Compatibility

- Summary

What is EMF?

- A modeling & data integration framework
- Exploits the facilities offered in Eclipse to...
  - Generate code without losing user customizations (merge)
  - Automate important tasks (such as registering runtime information)
  - Improve extensibility
  - Provide a UI layer

What is EMF “Model”?

- Specification of your application’s data
  - Object attributes
  - Relationships (associations) between objects
  - Operations available on each object
  - Simple constraints (e.g. cardinality) on objects and relationships
- Essentially it represents the class diagram of the application

What does EMF Provide?

- From a model specification, EMF can generate efficient, correct, and easily customizable implementation code
- Out of the box, EMF provides support for
  - Java™ interfaces
  - UML™
  - XML Schema
- EMF converts your models to Ecore (EMF metamodel)

What does EMF Provide?

- Tooling support within the Eclipse framework (UI, headless mode, Ant and standalone), including support for generating Eclipse-based and RCP editors
- Reflective API and dynamic model definition
- Persistence API with out of box support for XML/XMI (de)serialization of instances of a model
- And much more….

Why EMF?

- EMF is middle ground in the modeling vs. programming worlds
  - 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 application
- Very low cost of entry
  - EMF is free and open source
  - Full scale graphical modeling tool not required
  - Reuses your knowledge of UML, XML Schema, or Java
- It’s real, proven technology (since 2002)

EMF History

- First version was released in June, 2002
- Originally based on MOF (Meta Object Facility)
  - From OMG (Object Management Group)
  - Abstract language and framework for specifying, constructing, and managing technology neutral metamodels
- 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 2.0 specification; aligned with UML 2.0
  - EMF provides approximately the same functionality
    - Significant contributor to the specification; adapting to it

Who is Using EMF Today?

- Eclipse projects
  - Tools projects: UML2 and Visual Editor (VE)
  - Web Tools Platform (WTP) Project
  - Test and Performance Tools Platform (TPTP) Project
  - Business Intelligence and Reporting Tools (BIRT) Project
  - Data Tools Platform (DTP) Project
  - Technology project: Graphical Modeling Framework (GMF)
- Commercial offerings
  - IBM, Borland, Oracle, Omondo, Versata, MetaMatrix, Bosch, Ensemble...
- Large open source community
  - Estimated 125,000 download requests in January 2006

EMF at IBM

- Pervasive usage across product lines
  - IBM® Rational® Software Architect
  - IBM® Rational® Application Developer for WebSphere® Software
  - IBM® WebSphere® Integration Developer
  - IBM® WebSphere® Application Server
  - IBM® Lotus® Workplace
- Emerging technology projects: alphaWorks®
  - Emfatic Language for EMF Development (http://www.alphaworks.ibm.com/tech/emfatic)
  - Model Transformation Framework (http://www.alphaworks.ibm.com/tech/mtf)
  - XML Forms Generator (http://www.alphaworks.ibm.com/tech/xfg)

Agenda

- EMF in a Nutshell

- ***Working with URIs***
- Working with Resources and ResourceSets
- Customizing the Code Generator
- Tuning for Performance and/or Memory Footprint
- Importers and Exporters
- Supporting Backward and Forward Compatibility

- Summary

URI – Uniform Resource Identifier

- A formatted string that serves as an identifier for a resource
- Specification:
  - Definition: http://www.ietf.org/rfc/rfc1630.txt
  - Generic Syntax: http://www.ietf.org/rfc/rfc2396.txt
- Can assume two forms
  - URL – Uniform Resource Locators
    - Example: http://www.ietf.org/rfc/rfc1738.txt, http://rfc1737.x42.com/, http://rfc2141.x42.com/
    - In Java is always associated with a stream handler
  - URN – Uniform Resource Name
    - Example: urn:oasis:member:A00024:x

URI Syntax

- Generic
  - [*scheme*:]*scheme-specific-part*[*#fragment*]
    - Absolute vs. Relative
    - Opaque vs. Hierarchical
    - Examples: file:///tmp/calendar, mailto:marcelop@ca.ibm.com, ../tmp/presentation.ppt
- Hierarchical
  - [*scheme*:][*//authority*][*path*][*?query*][*#fragment*]
    - Server-based vs. Registry based authority
    - Example:
      http://download.eclipse.org
      /tools/emf/scripts/downloads-viewer.php
      ?s=2.1.0/R200507070200
      #quicknav

URIs in EMF

- URIs are used in EMF to identify:
  - A resource
  - An object in a resource
  - An Ecore package
    - The package's namespace URI
- Important facilities:
  - URI class
    - org.eclipse.emf.common.util.URI
  - URI Converter
  - Eclipse Utilities

org.eclipse.emf.common.util.URI

- Predates java.net.URI
- Instances are “immutable objects” that represent URIs
- Besides all the URI required operations, this class also provides Eclipse related methods and other features

Static factory methods	Simplify the instantiation of a URI by providing the appropriate scheme, encoding, …
Syntax query methods	Allow retrieving the individual elements that form the URI, such as the scheme, authority, and device
Resolve and Deresolve methods	Create a new absolute or relative URI using another URI as base

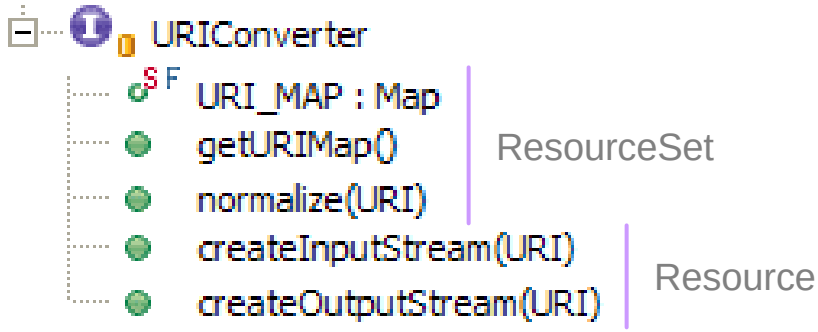
org.eclipse.emf.common.util.URI

```
{
  URI fileURI = URI.createFileURI("c:\\tmp\\foo.txt");
  URI platformURI = URI.createPlatformResourceURI("/Project A/MyModel.ecore", true);

  System.out.println(fileURI);
  System.out.println(platformURI);

  URI relativeURI = URI.createURI("../bar.txt");
  URI absoluteFileURI = relativeURI.resolve(fileURI);
  URI absolutePlatformURI = relativeURI.resolve(platformURI);

  System.out.println(relativeURI);
  System.out.println(absoluteFileURI);
  System.out.println(absolutePlatformURI);

  URI sameDirectoryFileURI = URI.createFileURI("c:\\tmp\\A File.txt");

  System.out.println(fileURI.deresolve(sameDirectoryFileURI));
}
```

```
file:/c:/tmp/foo.txt
platform:/resource/Project%20A/MyModel.ecore

../bar.txt
file:/c:/bar.txt
platform:/resource/bar.txt

foo.txt
```


URI Converter

- Provides an interface for customized treatment of URIs within a Resource Set
- It is responsible for
  - URI normalization
  - Creation of input and output streams
- org.eclipse.emf.ecore.resource.URIConverter



URI Normalization

- Process of converting an input URI into an actual URI of a resource
- The URIConverter default implementation
  - Checks if the input URI is mapped to a different URI
    - The global map can be populated by accessing *URI.URI_MAP* or through the *uri_mapping* extension point

```
<extension point="org.eclipse.emf.ecore.uri_mapping">
  <mapping source="//special/" target="special/"/>
</extension>
```

  - Turns a relative URI into a Platform Resource URI or a file URI
  - Applies this algorithm recursively

URI Normalization

```
{
  URI logicalURI1 = URI.createURI("myscheme:/identifier/");
  URI logicalURI2 = URI.createURI("myscheme:/identifier/myFile.xml");
  URI physicalURI = URI.createPlatformResourceURI("prj/fld/");

  URIConverter uriConverter = new URIConverterImpl();
  uriConverter.getURIMap().put(logicalURI1, physicalURI);

  System.out.println(uriConverter.normalize(logicalURI2));

  URI relativeURI = URI.createURI("myFile.xml");

  System.out.println(uriConverter.normalize(relativeURI));
}
```

```
platform:/resource/prj/fld/myFile.xml

file:/C:/eclipse/workspace/myFile.xml
```


Eclipse Utilities

- When developing Eclipse applications, you can use several utilities and helpers available in EMF
  - org.eclipse.emf.common.CommonPlugin.resolve(URI)
    - Resolves a platform URI into a local URI, if possible
  - org.eclipse.emf.ecore.plugin.EcorePlugin
    - computePlatformURIMap()
      - Computes a map so that plug-ins in the workspace will override those in the environment
    - getPlatformResourceMap() and resolvePlatformResourcePath(String)
      - Defines an association between a name and a URI

Eclipse Utilities

```
{
  URI existingResourceURI = URI.createPlatformResourceURI("MyProject");
  URI nonExistingResourceURI = URI.createPlatformResourceURI("NotMyProject");

  System.out.println(CommonPlugin.resolve(existingResourceURI));
  System.out.println(CommonPlugin.resolve(nonExistingResourceURI));

  URI workspacePluginURI = URI.createURI("platform:/plugin/my.new.plugin/");
  Map platformURIMap = EcorePlugin.computePlatformURIMap();

  System.out.println(platformURIMap.get(workspacePluginURI));

  URI projectFileURI = URI.createFileURI(“c:\\MyProjectDir”);
  EcorePlugin.getPlatformResourceMap().put("MyProject", projectFileURI);

  System.out.println(EcorePlugin.resolvePlatformResourcePath("MyProject"));
  System.out.println(EcorePlugin.resolvePlatformResourcePath("MyProject/foo.txt"));
}
```

```
file:/C:/eclipse/workspace/MyProject
platform:/resource/NotMyProject

platform:/resource/my.new.plugin/

file:/C:/MyProjectDir
file:/C:/MyProjectDir/foo.txt
```


Agenda

- EMF in a Nutshell

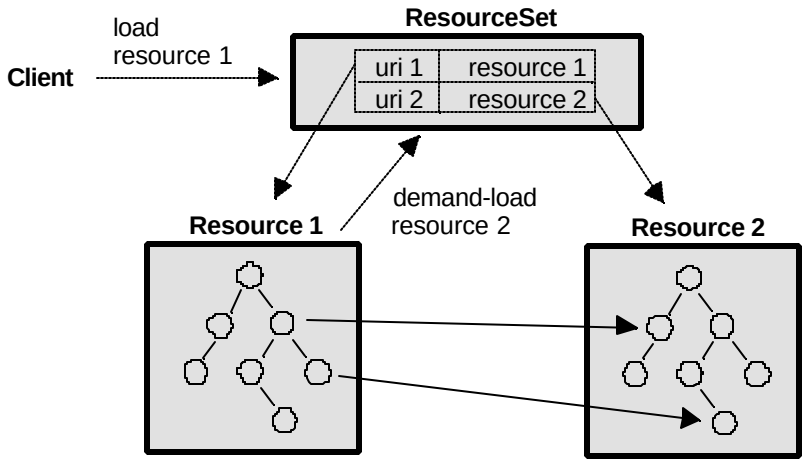
- Working with URIs
- ***Working with Resources and ResourceSets***
- Customizing the Code Generator
- Tuning for Performance and/or Memory Footprint
- Importers and Exporters
- Supporting Backward and Forward Compatibility

- Summary

Working with Resources and ResourceSets

- Working with proxies
  - Identifying proxies
  - Using the Validation Framework to detect unresolved proxies
  - Using “fragment queries” to handle unresolved proxies
- Cross-Resource Containment
- Inverse References
- Resource Tips & Tricks
  - Unloading
  - Tracking modification
  - Am I loading something?

Persistence and Serialization Review



- 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

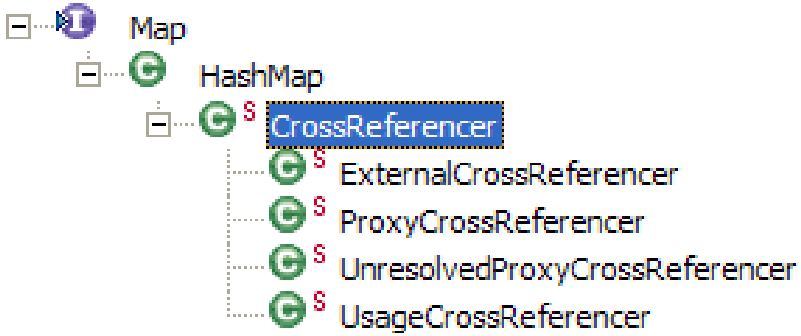
Identifying Proxies

- In some scenarios it may be important to know up-front what are the proxies referenced by a given object
  - Check whether the user has everything that will be necessary to load the model
  - Extract missing resources from the repository
  - Decide whether loading a specific object is a long or short operation

- org.eclipse.emf.ecore.util.EcoreUtil.ProxyCrossReferencer
  - A CrossReferencer that finds proxies without resolving them
  - For each object in a given context (resource, resource set, EObject, collection), checks if the referenced objects are in the same resource or not

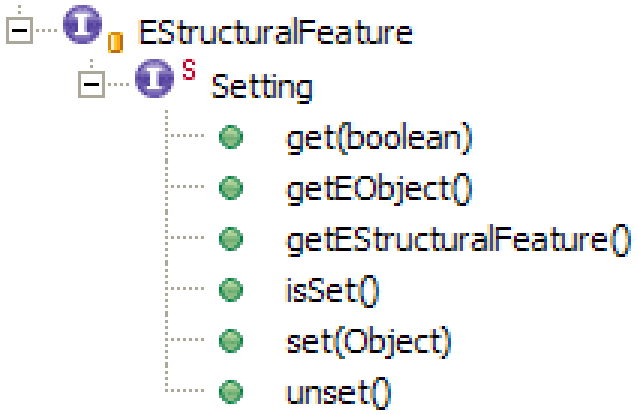
Going Deeper: EcoreUtil's CrossReferencers

- Utility classes that find "uses" of an object
- Also the data structures that hold the objects they find
  - A CrossReferecer is a java.util.Map
  - On each Map.Entry
    - Key: reference target
    - Value: list of EStructuralFeature.Setting with all the source object-feature pairs referencing the keyed target



Going Deeper: EStructuralFeature.Setting

- Interface that represents the value held by a feature of an EObject
- Exposes all the feature-related methods of EObject
  - eGet, eSet, eIsSet, and eUnset



Going Deeper: Types of Reference

- Containment Reference
  - Has an “implicit” opposite (eObject.eContainer())
- Container Reference
  - An explicit reference defined as the opposite of a containment reference
- Cross-Reference
  - May or may not have an opposite
- Any reference can refer to an EObject located in either the same or different resource

EcoreUtil.ProxyCrossReferencer

```
public static void printProxiesDetails(ResourceSet resourceSet)
{
  int counter = 0;
  Map map = ProxyCrossReferencer.find(resourceSet);
  for (Iterator i = map.entrySet().iterator(); i.hasNext();)
  {
    Map.Entry entry = (Map.Entry)i.next();
    EObject proxyEObject = (EObject)entry.getKey();
    System.out.println("" + ++counter + ". " + EcoreUtil.getURI(proxyEObject));
    System.out.println("Is Proxy: " + proxyEObject.eIsProxy());

    List settings = (List)entry.getValue();
    for (Iterator j = settings.iterator(); j.hasNext();)
    {
      EStructuralFeature.Setting setting = (EStructuralFeature.Setting)j.next();
      System.out.println("\tFeature: " + setting.getEStructuralFeature().getName());

      EObject eObject = setting.getEObject();
      EStructuralFeature nameFeature = eObject.eClass().getEStructuralFeature("name");
      if (nameFeature != null)
      {
        System.out.println("\tEObject.getName(): " + setting.getEObject().eGet(nameFeature));
      }
    }
  }
}
```

```
1. uri3.xmi#/0
Is Proxy: true
   Feature: home
   EObject.getName(): john
```


Validating Proxies

- The validation framework can be used to validate proxies
- The EObjectValidator checks if all the referenced objects of a given EObject are resolved
  - The proxy references are resolved during the validation
- If a proxy is broken, the validation produces a Diagnostic with EObjectValidator.*EOBJECT__EVERY_PROXY_RESOLVES* as its error code

Validating Proxies

```
public static void validateProxies(EObject eObject)
{
  Diagnostic diagnostic = Diagnostician.INSTANCE.validate(eObject);
  if (diagnostic.getSeverity() == Diagnostic.ERROR)
  {
    if (hasEveryProxyResolvesCode(diagnostic))
    {
      System.out.println("Broken proxies found!");
      return;
    }
  }
  System.out.println("Proxies are fine.");
}

public static boolean hasEveryProxyResolvesCode(Diagnostic diagnostic)
{
  if (diagnostic.getCode() == EObjectValidator.EOBJECT__EVERY_PROXY_RESOLVES)
    return true;

  for (Iterator i = diagnostic.getChildren().iterator(); i.hasNext();)
  {
    Diagnostic childDiagnostic = (Diagnostic)i.next();
    return hasEveryProxyResolvesCode(childDiagnostic);
  }
  return false;
}
```


Handling Broken Proxies

- Your application can provide a mechanism to handle broken proxies by
  - Reporting the error to the user
  - Avoiding use of the object(s) that could not be resolved
  - Recovering the missing object(s)
- EMF doesn't provide such a mechanism but provides hooks to help you implement one
  - Fragment query capability is one of these hooks

Fragment Queries

- The idea is to “decorate” the fragment portion of a URI with details that further describe the referenced object
  - The description can be used by a service to locate the object or to enrich an error message to be presented to the user

- Query data
  - Added to the fragment portion of the object’s URI
  - Must be at the end of the URI fragment portion
  - Is delimited by “?”
  - Example:
    file:/c:/dir/library.xmi#//@books.0?Book.ISBN.0131425420?

Fragment Query

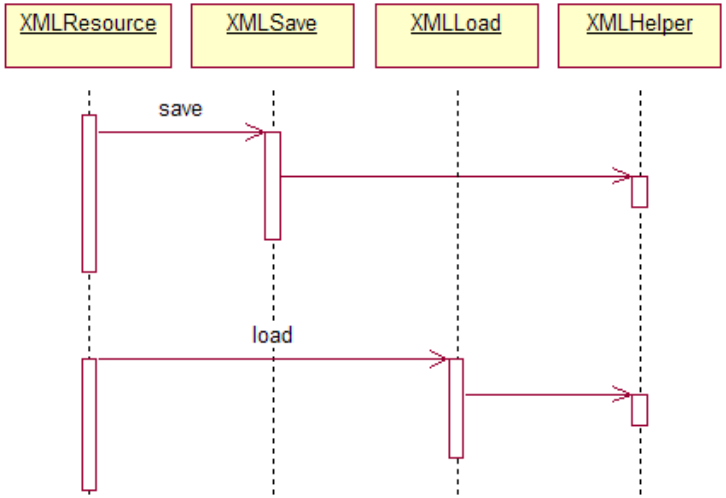
URI Fragment

Saving Fragment Queries

- If you are using a XMLResource to serialize your objects
  - Override the getURIFragmentQuery(Resource, EObject) method of org.eclipse.emf.ecore.xmi.impl.XMLHelperImpl to return the query that should be serialized for the given EObject
  - Override the *createXMLHelper()* method of *org.eclipse.emf.ecore.xmi.impl.XMLResourceImpl* to return an instance of your XMLHelper
- The *getURIFragmentQuery(…)* method returns the string that is used as the “query”
  - The string should not contain the delimiter character (*?*)
  - A null string indicates that there is no query
  - The characters must be valid URI fragment characters (as defined by the specification)

Going Deeper: org.eclipse.emf.ecore.xmi.XMLHelper

- Interface used by the default resource implementations
  - Not used by clients
- Single place where the methods used to save and load objects are located
  - Simplifies customizations



Using Fragment Queries

- EMF doesn't enforce the use of fragment queries
- Reasonable approach:
  - Load an object
  - Try to resolve its proxies
  - Identify the remaining (broken) proxies
  - Use the fragment query of the broken proxies to either report the problem to the user or recover the missing objects

Cross-Resource Containment

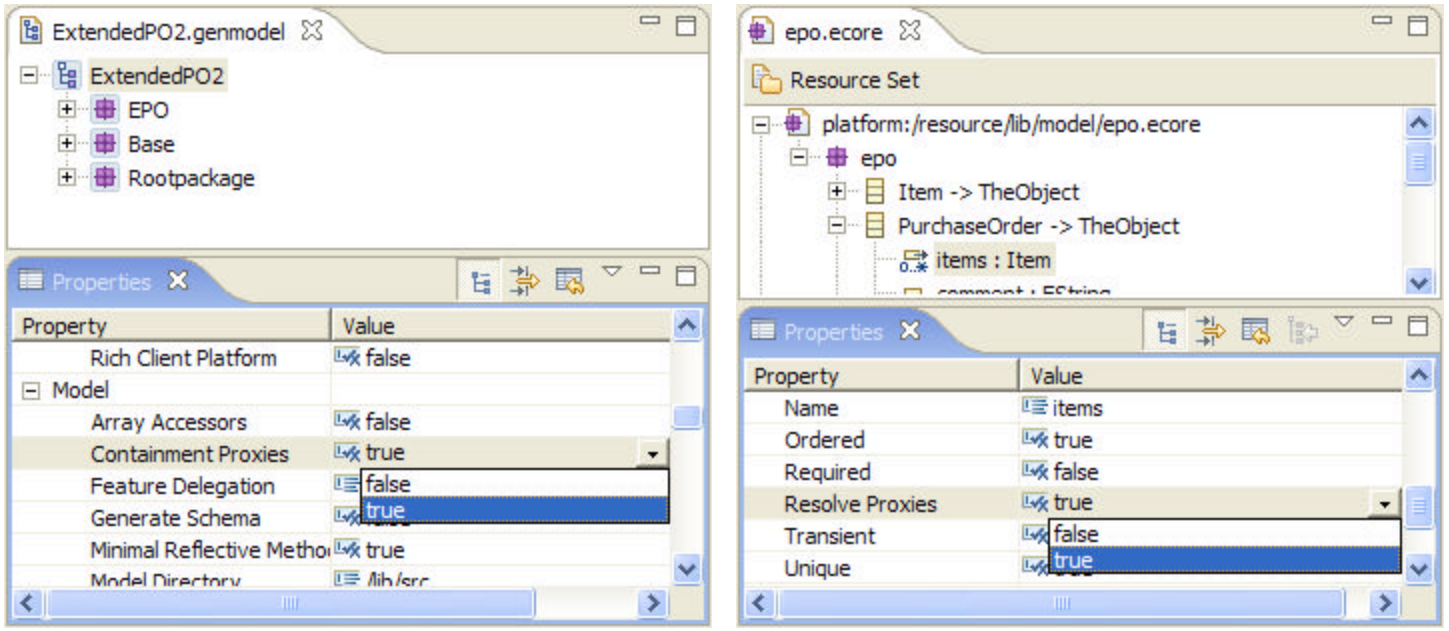
- Allows an object hierarchy to be persisted across multiple resources
  - eObject.eResource() may be different from eObject.eContainer().eResource()
- Must be explicitly enabled
  - The containment reference has to be set to resolve proxies
  - Also, on generated models, the value of the "Containment Proxies" generator model property has to be set to 'true'

Enabling Cross-Resource Containment

Dynamic Model

```
EReference houses = EcoreFactory.eINSTANCE.createEReference();
houses.setName("houses");
houses.setEType(house);
houses.setUpperBound(ETypedElement.UNBOUNDED_MULTIPLICITY);
houses.setContainment(true);
houses.setResolveProxies(true);
person.getEStructuralFeatures().add(houses);
```

Generated Model



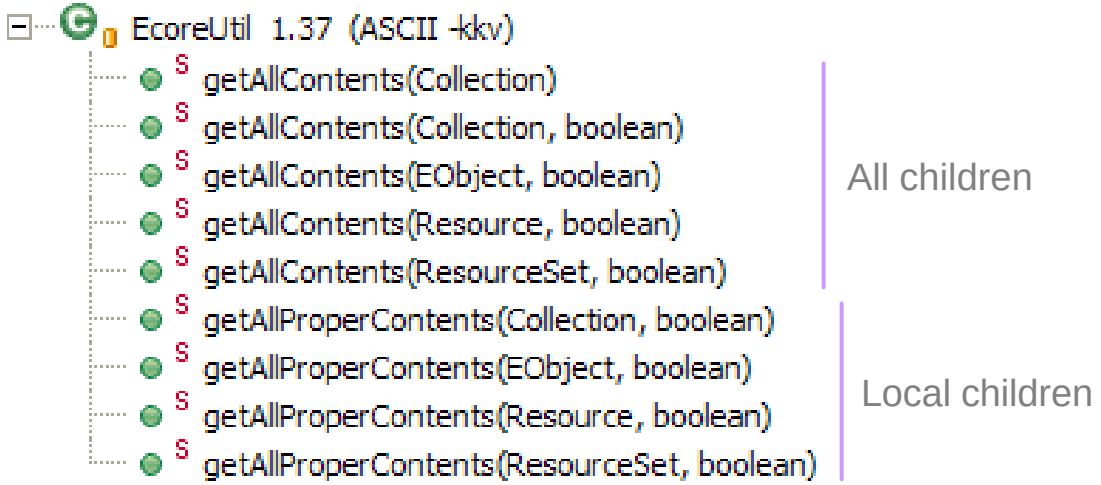
Cross-Resource Containment

```
{
  Library library = LibFactory.eINSTANCE.createLibrary();
  Book book = LibFactory.eINSTANCE.createBook();
  library.getBooks().add(book);

  System.out.println(library.eResource() + " - " + book.eResource());

  Resource libraryResource = new ResourceImpl(URI.createURI("lib"));
  libraryResource.getContents().add(library);

  System.out.println(library.eResource().getURI() + " - “ +
                     book.eResource().getURI());

  Resource bookResource = new ResourceImpl(URI.createURI("book"));
  bookResource.getContents().add(book);

  System.out.println(library.eResource().getURI() + " - " +
                     book.eResource().getURI());

  bookResource.getContents().remove(book);

  System.out.println(library.eResource().getURI() + " - " +
                     book.eResource().getURI());
}
```

```
null - null

lib - lib

lib - book

lib - lib
```


EcoreUtil Methods for Cross-Resource Containment

- When the containment tree is spread across multiple files, one may be interested in differentiating the “local” children of an object from children in other resources
- The **Proper** methods available in EcoreUtil only deal with the “local” children of a containment tree

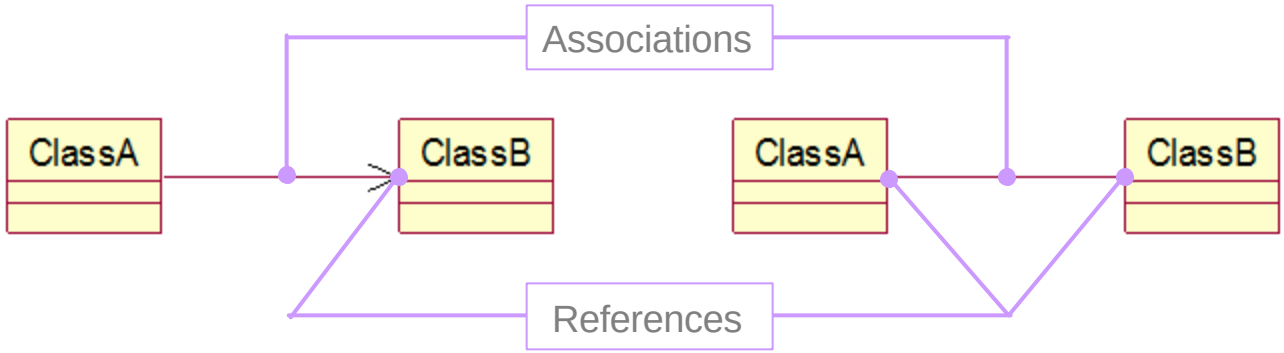


Inverse References

- Sometimes it may be handy to retrieve the opposite of a unidirectional association end
  - The use case you are implementing requires a navigation that was not anticipated when the model was defined
- EMF provides two mechanisms
  - CrossReferencer
  - ECrossReferenceAdapter

Going Deeper: Associations and References

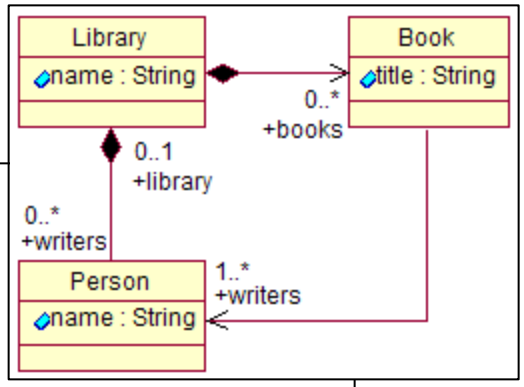
- An association between two classes allows their instances to "communicate" with each other
- Each navigable end of an association is represented by an EReference in Ecore
  - If the association is bidirectional, it will be represented by two references, r1 and r2 where r1.getEOpposite() == r2 && r2.getEOpposite() == r1



Using a CrossReferencer

- Dynamically computes a map with the target object and the list of settings that “refer” to it
  - This computation may be a long operation depending on the number of objects
- The map represents a snapshot of the state of the objects
  - As the objects change, the settings’ values may become inconsistent with the map
- Requires a context to compute the map
  - The context can be a collection of EObjects, a resource, or a resource set, or a combination of these
  - Objects that are not in the scope of the context won’t be available in the map

Using a CrossReferencer



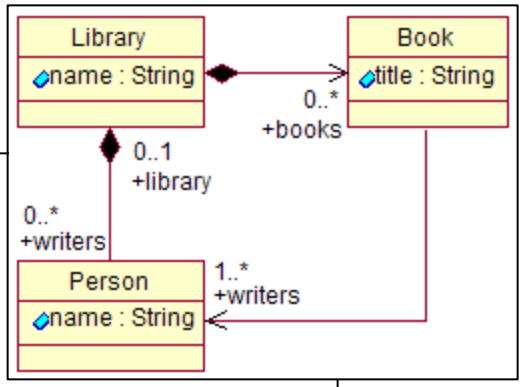

```
{
  Library library = LibFactory.eINSTANCE.createLibrary();
  Book book = LibFactory.eINSTANCE.createBook();
  book.setTitle("EMF");
  Person dave = LibFactory.eINSTANCE.createPerson();

  library.getBooks().add(book);
  library.getWriters().add(dave);
  book.getWriters().add(dave);

  Map map = CrossReferencer.find(Collections.singleton(library));
  List settings = (List)map.get(dave);
  if (settings != null)
  {
    for (Iterator i = settings.iterator(); i.hasNext();)
    {
      EStructuralFeature.Setting setting = (EStructuralFeature.Setting)i.next();
      if (setting.getEObject() == book &&
          setting.getEStructuralFeature() == LibPackage.Literals.BOOK__WRITERS)
      {
        System.out.println("Found it.  The book is \"" + book.getTitle() + "\"");
      }
    }
  }
}
```

```
Found it.  The book is "EMF"
```


Using the ECrossReferenceAdapter

- A special adapter that
  - Attaches itself to all objects in a containment hierarchy
    - May pose a memory footprint problem depending on the number of objects
  - Keeps a CrossReferencer in sync with the state of the objects
- The application can determine and change the context used when computing the map

Using the ECrossReferenceAdapter




```
{
  Library library = LibFactory.eINSTANCE.createLibrary();

  ECrossReferenceAdapter adapter = new ECrossReferenceAdapter();
  library.eAdapters().add(adapter);

  Book book = LibFactory.eINSTANCE.createBook();
  book.setTitle("EMF");
  Person dave = LibFactory.eINSTANCE.createPerson();

  library.getBooks().add(book);
  library.getWriters().add(dave);
  book.getWriters().add(dave);

  Collection settings = adapter.getNonNavigableInverseReferences(dave);
  for (Iterator i = settings.iterator(); i.hasNext();)
  {
    EStructuralFeature.Setting setting = (EStructuralFeature.Setting)i.next();
    if (setting.getEObject() == book &&
        setting.getEStructuralFeature() == LibPackage.Literals.BOOK__WRITERS)
    {
      System.out.println("Found it.  The book is \"" + book.getTitle() + "\"");
    }
  }
}
```

```
Found it.  The book is "EMF"
```

Resource Tips & Tricks

- Unloading a resource
  - Resource.unload()
  - EMF's default implementation performs the following steps:
    - Sets the *loaded* attribute to false;
    - Caches an iterator with all the proper contents of all objects held by the resource
    - Clears the resource's content, error and warning lists
    - Turns each object returned by the iterator into a proxy and clears its adapter list
    - Fires a notification
      - FeatureID = Resource.*RESOURCE__IS_LOADED*
  - You may also remove the resource from the resource set

Resource Tips & Tricks

- Tracking modification
  - Resource.setTrackingModification(boolean)
  - When activated, EMF's default implementation performs the following steps:
    - Instantiates an Adapter and registers it on all proper objects
      - The adapter calls Resource.setModified(boolean) after receiving a notification
    - Registers the adapter on any object added to the resource and deregisters it from objects that are removed
  - When deactivated, the default implementation removes the adapter from the objects
  - You can manually define what is the isModified state of a resource

Resource Tips & Tricks

- Am I loading something?
  - Resource.Internal.isLoading()
    - Every Resource is supposed to implement the Resource.Internal interface
  - When a resource is being loaded the isLoading() method returns true

Agenda

- EMF in a Nutshell

- Working with URIs
- Working with Resources and ResourceSets
- ***Customizing the Code Generator***
- Tuning for Performance and/or Memory Footprint
- Importers and Exporters
- Supporting Backward and Forward Compatibility

- Summary

Generating Customized Code

- The EMF code generator can be customized in various ways to meet the needs of your application
- There are different levels of customization
  - Generator Options
  - Dynamic Templates
  - Generator Specialization
- As the generator is made more flexible and extensible, code customization will become even easier

Generator Options

- The EMF code generator provides a number of options which, when enabled, can have a significant effect on the kind of code that is generated using the default templates
  - Array Accessors
  - Suppress Containment
  - Suppress Interfaces
  - Suppress Notification
  - Suppress Unsettable
  - Suppress EMF Meta Data
  - Suppress EMF Model Tags
  - Suppress EMF Types

Array Accessors

- Generates “bean-like”, array-based getters and setters for multi-valued features
- Set the value of the ‘Array Accessors’ generator model property to ‘true’
- Since arrays are strongly typed, provides faster item access (no double index range checking and no casting)
- Direct modification of the array will not produce notification

Suppress Containment

- Generates code that does not automatically update container references when containment features are modified
- Set the value of the ‘Suppress Containment’ generator model property to ‘true’
- Improves performance by avoiding the need to maintain referential integrity
- Many things in the framework will work poorly, so use only if you don't need full integration with the rest of EMF

Suppress Interfaces

- Generates implementation classes without separate interface declarations
- Set the value of the ‘Suppress Interfaces’ generator model property to ‘true’
- Improves performance since dispatch through an interface is slower and less likely to be in-lined by a JIT compiler
- Reduces byte code footprint somewhat
- Can only be used for models with no multiple inheritance

Suppress Notification

- Generates code (setters and lists) that does not provide support for change notification
- Set the value of the ‘Suppress Notification’ generator model property to ‘true’
- Saves the cost of checking whether notification is needed, and reduces footprint by not generating notification code

Suppress Unsettable

- Generates code without support (isSet/unset methods) for unsettable features
- Set the value of the ‘Suppress Unsettable’ generator model property to ‘true’
- Saves on instance footprint (only slightly if using Boolean flags) and on update costs, since there is less state to update
- The same can be achieved by avoiding this in the model in the first place (i.e. no unsettable features)

Suppress EMF Meta Data

- Generates code without a package interface declaring metadata accessors and constants
- Set the value of the ‘Suppress EMF Meta Data’ generator model property to ‘true’
- Helps produce an API with no visible dependencies on EMF, i.e. a "pure" API

Suppress EMF Model Tags

- Generates code without @model tags in the Javadoc
- Set the value of the ‘Suppress EMF Model Tags’ generator model property to ‘true’
- Helps produce an API with no visible dependencies on EMF, i.e. a "pure" API

Suppress EMF Types

- Generates methods for features and operations which use standard Java collection and object types instead of those from EMF (e.g. EList, EMap, EObject)
- Set the value of the ‘Suppress EMF Types’ generator model property to ‘true’
- Helps produce an API with no visible dependencies on EMF, i.e. a "pure" API

Dynamic Templates

- The content of the templates used by the EMF code generator can be changed or replaced by enabling dynamic templates
- Set the value of the ‘Dynamic Templates’ generator model property to ‘true’
- Set the value of the ‘Template Directory’ generator model property to the location of the custom templates
- Dynamic templates can be customized in three ways
  - Replacement
  - Insertions
  - Overrides

Template Replacement

- Copy default template to the dynamic template directory (with the same name and relative location) and modify as desired
- Modified template is compiled dynamically and used in place of the default template
- Must keep in sync with changes in default templates (“clone and own”)…

Template Insertions

- Can insert content into default templates via insertion points identified using JET @include directive (fail="silent")
- Dynamic template includes default template; insertions are included and compiled dynamically, result is used in place of the default template
- By convention, insertions are placed in a folder named after the including template and end with a .insert.*jet extension
- A number of insertion points have been defined for some commonly customized EMF templates (i.e. Class.javajet, ItemProvider.javajet, TestCase.javajet)

Template Overrides

- Can override content of default templates via override points identified using JET @include directive (fail="alternative")
- Dynamic template includes default template; overrides are included and compiled dynamically, result is used in place of the default template
- By convention, overrides are placed in a folder named after the including template and end with a .override.*jet extension
- A number of override points have been defined for some commonly customized EMF templates (i.e. Class.javajet, ResourceFactoryClass.javajet, TestCase.javajet)

Generator Specialization

- Complete control over code generation, including which templates are used and for which model elements, can be obtained via a specialized generator model implementation
- New generator properties/options can be introduced and exposed in the default editor by registering specialized adapter factory via the org.eclipse.emf.edit.itemProviderAdapterFactories extension point
- Must keep up with changes in default generator model implementation
- Can use Generator Annotations as an alternative to introducing new properties

Agenda

- EMF in a Nutshell

- Working with URIs
- Working with Resources and ResourceSets
- Customizing the Code Generator
- ***Tuning for Performance and/or Memory Footprint***
- Importers and Exporters
- Supporting Backward and Forward Compatibility

- Summary

Performance/Memory Overhead

- EMF provides a number of mechanisms that can be used to tune the performance and/or memory overhead of your model implementation
- Code Generator Options
  - Boolean Flags
  - Virtual Feature Delegation
  - Minimal Reflective Methods
  - Literals Interface
- XMLResource Options

Boolean Flags

- Generates code that uses (a) consolidated bit field(s) instead of separate fields to represent the values of Boolean attributes and whether unsettable features are set
- Set the value of the ‘Boolean Flags Field’ generator model property to the name of the bit field(s) to be generated or reused
- Set the value of the ‘Boolean Flags Reserved Bits’ generator model property to the number of bits in the field(s) that are reserved (e.g. used by parent classes)
- Helps reduce memory overhead for models with a large number of Boolean attributes or unsettable features

Virtual Feature Delegation

- Generates code that uses a single, dynamically-allocated array field and associated index field(s) instead of separate fields to represent the values of non-primitive features
- Set the value of the ‘Feature Delegation’ generator model property to ‘Virtual’
- Helps reduce memory overhead for models with a large number of features that are often “sparsely” used

Minimal Reflective Methods

- Generates reflective methods that delegate to *super*, i.e. they delegate switching for any non-matching features to the superclass implementation for dispatching
- Set the value of the 'Minimal Reflective Methods' generator model property to 'true' (default)
- Helps reduce memory (byte code) overhead for models with a large number of inherited features
- May increase performance overhead for models with deep class hierarchies

Literals Interface

- Generates an interface (nested within the package interface) that defines metadata literals (i.e. types and features)
- Set the value of the ‘Literals Interface’ generator package property to ‘true’ (default)
- Helps reduce performance overhead of access to a model’s metadata, e.g. EcorePackage.Literals.ECLASS vs. EcorePackage.eINSTANCE.getEClass()

XMLResource Options

- XMLResource offers a set of options that can be used to tweak load and save behavior
- The options are passed to the resource's save and load methods as entries in a map
  - The key is the constant that represents the option
  - Each option requires an appropriate value

XMLResource Load Options

- OPTION_DEFER_IDREF_RESOLUTION
  - Option value: Boolean
  - Defers resolving references within a resource until the whole document has been parsed
- OPTION_USE_PARSER_POOL
  - Option value: org.eclipse.emf.ecore.xmi.XMLParserPool
  - Provides a parser pool, from which SAXParser instances are created and reused
- OPTION_USE_XML_NAME_TO_FEATURE_MAP
  - Option value: java.util.Map
  - Enables sharing the cache of mappings between qualified XML names (namespace + local name) and corresponding Ecore features across invocations of load(…), or even among resources

XMLResource Load Options

- OPTION_USE_CACHED_LOOKUP_TABLE
  - Option value: java.util.List
  - Specifies a list as a place holder for caching information during the subsequent saving of XML documents
- OPTION_USE_DEPRECATED_METHODS
  - Option value: Boolean
  - Use methods that were deprecated in EMF
  - The default value is Boolean.TRUE

XMLResource Save Options

- OPTION_FLUSH_THRESHOLD
  - Option value: Integer
  - Specifies a maximum number of characters to allow in the output stream before flushing it
- OPTION_USE_CACHED_LOOKUP_TABLE
  - Option value: java.util.List
  - Provides a placeholder to cache information about structure of the model (using qualified XML names as a key for caching information)
- OPTION_CONFIGURATION_CACHE
  - Option value: Boolean
  - Enables caching and reusing generic data in repeated save operations, avoiding the cost of reinitializing the data

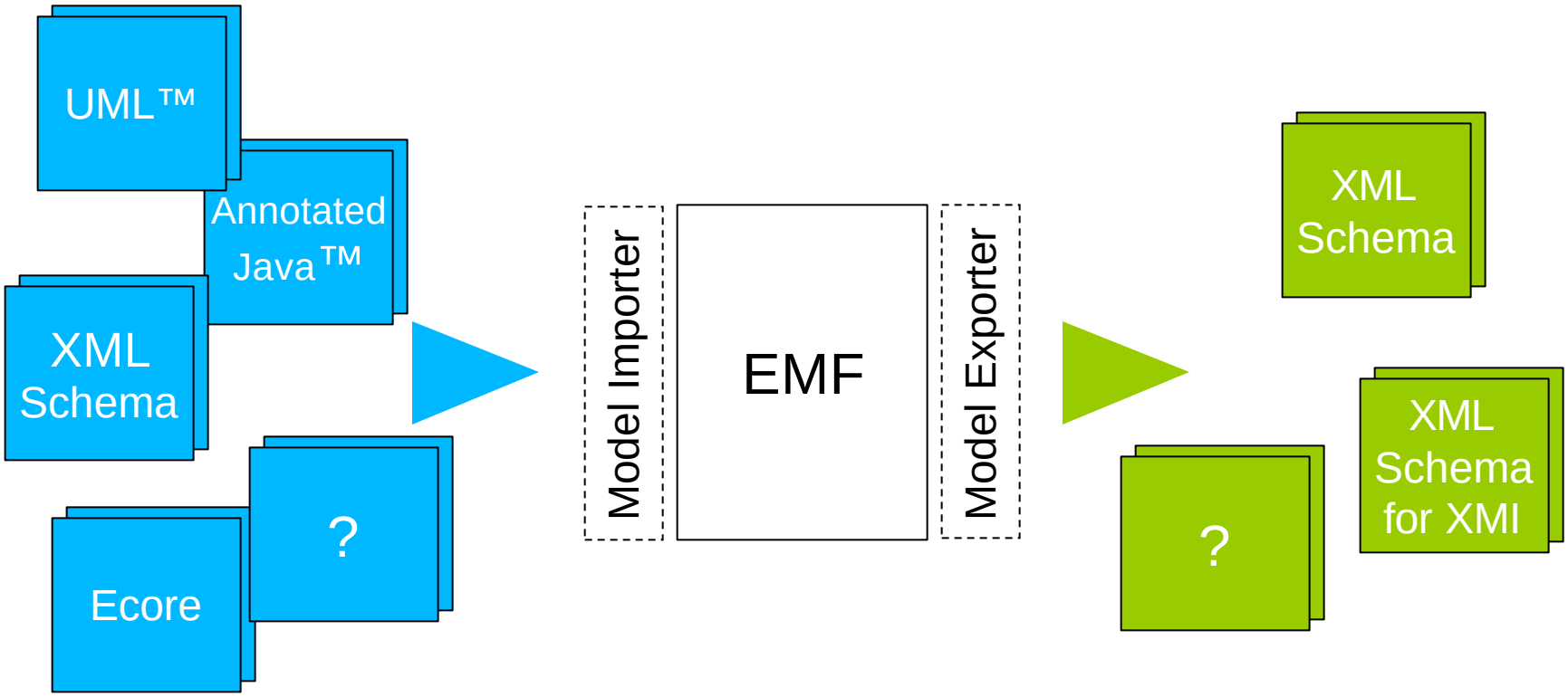
XMLResource Save Options

- OPTION_FORMATTED
  - Option value: Boolean
  - Disables formatting of documents, omitting whitespaces and line brakes, to improve the performance of saving and, subsequently, loading the resulting document
- OPTION_USE_FILE_BUFFER
  - Option value: Boolean
  - Enables accumulating output during serialization in a temporary file, rather than an in-memory buffer

Agenda

- EMF in a Nutshell
- Working with URIs
- Working with Resources and ResourceSets
- Customizing the Code Generator
- Tuning for Performance and/or Memory Footprint
- ***Importers and Exporters***
- Supporting Backward and Forward Compatibility
- Summary

Importers and Exporters



Model Importers

- A Model Importer (aka an importer) is responsible for creating an Ecore model from the description of a modeled domain
  - An importer is also expected to create the Generator Model (.genmodel file) for the Ecore Model (.ecore file)
- Each importer knows how to handle a specific format of description
  - EMF provides importers that handle the following formats
    - IBM® Rational Rose® models
    - XML Schemas
    - Annotated Java Interfaces
    - Ecore and EMOF files

Model Exporters

- A Model Exporter (aka exporter) is able to read an Ecore model and generate another description of the same modeled domain
  - An exporter typically uses the information described in the Generator Model to accomplish its purposes
- EMF provides two exporters
  - XML Schema
  - XML Schema for XMI

Going Deeper: Why is the Generator Model so important?

- The Generator Model acts as a decorator for an Ecore Model
  - It provides details that would pollute the model, such as
    - The qualified name of an EPackage
      - BasePrefix attribute of a GenPackage
    - Actual location of the model, edit, editor, and test source folders
- When a modeled domain is converted into an EMF model, the importer may be able to capture some Generator Model details and store them in a .genmodel file
  - The Java package of the Annotated Java Interfaces
  - Referenced XML Schemas that may or may not be already represented by Ecore models
- The Generator Model is useful to an exporter because
  - It can be used to persist details about the exported artifact
  - Some details may be important to properly describe the modeled domain

Using Importers and Exporters

- The importers and exporters are presented to the user as pages of a standard EMF wizard
  - The registered importers are presented during the execution of the new EMF model or EMF project wizards
  - The list of exporters is shown when the action “Export Model” is invoked on a Generator Model
  - The motivation of this design is to let the importers and exporters decide the input that the user needs to provide
    - Each implementation has total control over the number of pages of the wizard and their content
- The Rose and XML Schema importers are also available as Eclipse applications and Ant tasks

Contributing an Importer via an Extension

```
<extension point="org.eclipse.emf.importer.modelImporterDescriptors">
  <modelImporterDescriptor
     id="com.mycompany.ModelImporterDescriptor.XYZ"
     name="XYZ class model"
     extensions="xyz"
     wizard="com.mycompany.ImportWizard"/>
</extension>
```

- The wizard must be a class that implements EMF's IModelImporterWizard interface
  - Defines the setters that will be used by the new EMF model and project wizards to communicate the details about the model specification being imported

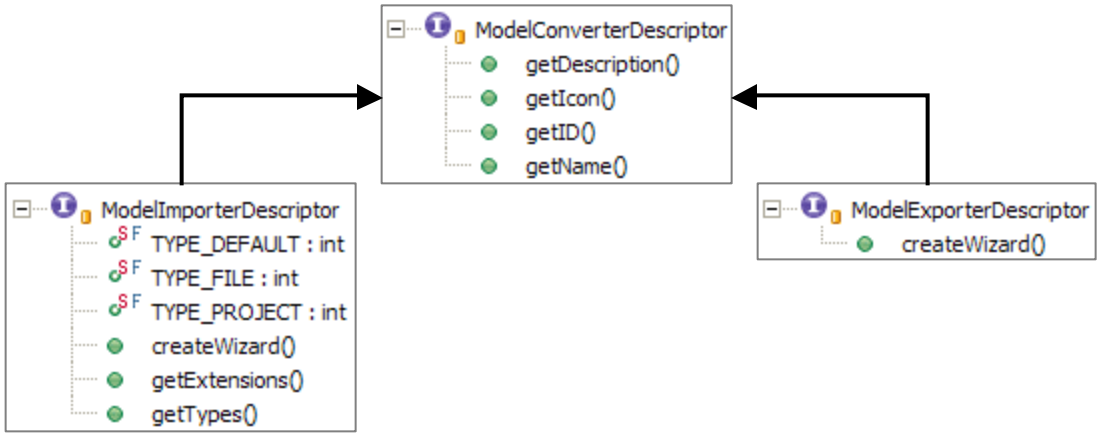
Contributing an Exporter via an Extension

```
<extension point="org.eclipse.emf.importer.modelExporterDescriptors">
  <modelExporterDescriptor
     id="com.mycompany.ModelExporterDescriptor.XYZ"
     name="XYZ class model"
     wizard="com.mycompany.ExportWizard"/>
</extension>
```

- The wizard must be a class that implements Eclipse's IWorkbenchWizard interface
  - The first time the wizard is presented to the user, the method init(IWorkbench, IStructuredSelection) is invoked and the selection argument is either an IFile that contains the Generator Model or an instance of GenModel

Contributing Importers and Exporters via Global Registries

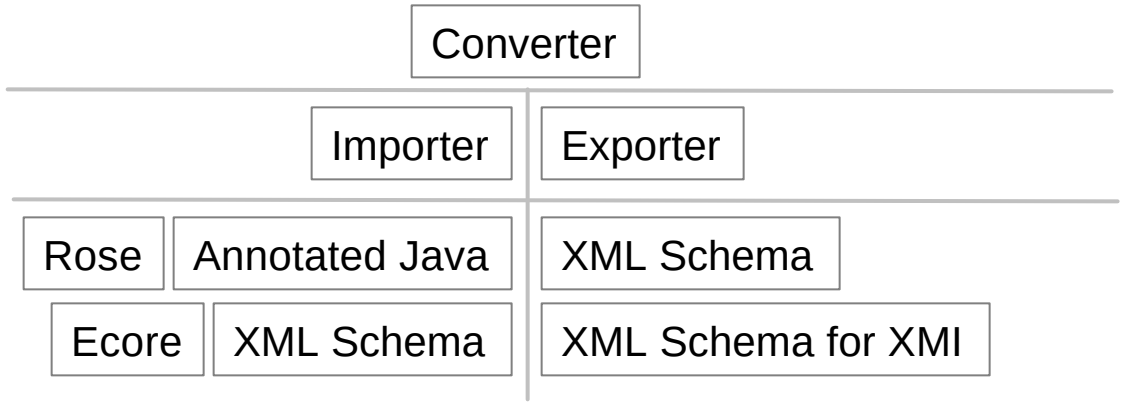
- The idea is to add the importer/exporter description to the appropriate list
  - ModelImporterManager.INSTANCE.getModelConverterDescriptors()
  - ModelExporterManager.INSTANCE.getModelConverterDescriptors()
- The importer and exporter descriptions are quite similar, which led to the creation of a common parent interface



Converter Framework

- EMF contributes four importers and two exporters; it would be really tedious to code each one of them from scratch so we've created a framework
  - Using this framework is completely optional

- The component architecture of the framework is



Creating an Importer Using the Converter Framework

1. Create a class that extends *ModelImporter*
   - This is the most important class of the new importer
   - Is the object responsible for actually converting the domain specification into an Ecore model
   - Should be UI-independent
   - It also
     - Stores the data entered by the user on each page of the wizard
     - Is responsible for performing all the non-UI work of the import
   - Examples:
     - RoseImporter
     - XSDImporter

Creating an Importer Using the Converter Framework

2. Create the wizard pages for the new importer
   - Each page should implement IModelImporterPage and extend ModelConverterPage
     - Ensures that the page has access to the instance of the ModelConverter created in the first step
   - The pages of the different importers provided by EMF are very similar; if this is the case for the new importer, it is possible to reuse the pages defined in the importer plug-in
     - ModelImporterDetailPage
     - ModelImporterPackagePage

Creating an Importer Using the Converter Framework

3. Create a class that extends *ModelImporterWizard*
   - This class is the entry point of the new importer, since it instantiates the ModelImporter subclass defined in the first step
   - Add the pages created for this importer

4. Register the new importer
   - If you've decided to register the importer via the global registry, you can extend ModelImporterManager.ModelImporterDescriptorImpl instead of creating a new implementation of the ModelImporterDescriptor interface

Creating an Exporter Using the Converter Framework

- The recipe for creating an exporter is similar to the one used for an importer
  - The ingredients are little bit different though
    - ModelExporter instead of ModelImporter
    - IModelExporterPage instead of IModelImporterPage
    - And so on…

Saving the Information of an Export Operation

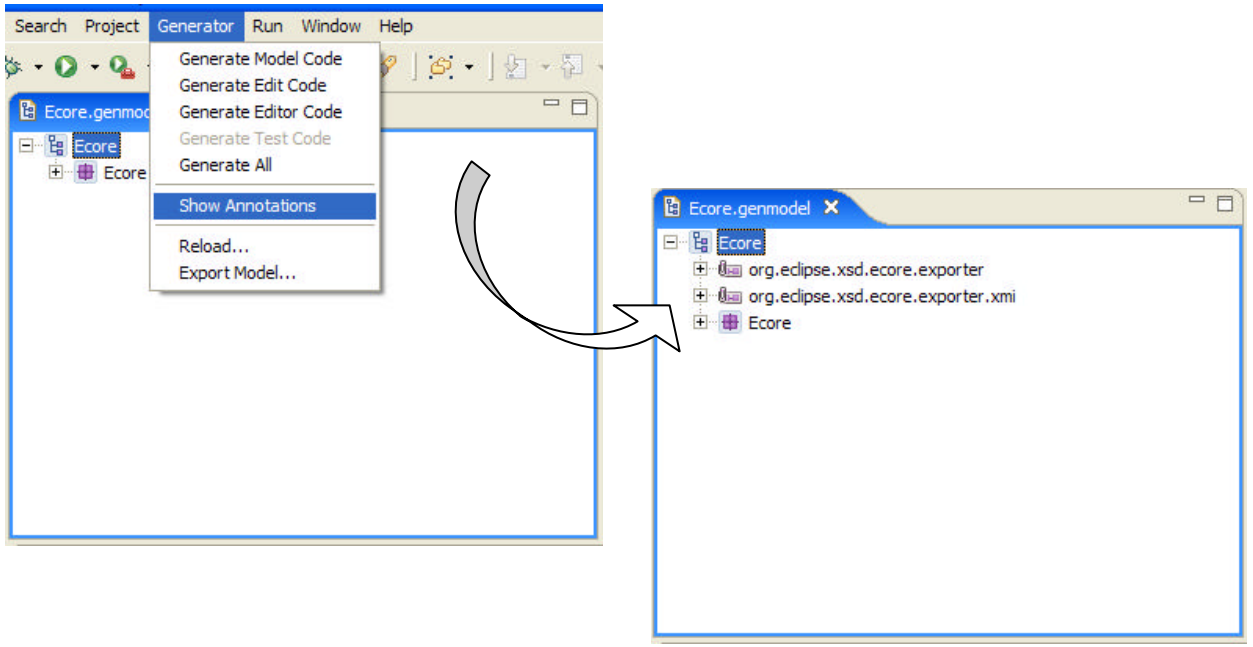
- One interesting feature provided by the Converter Framework is the capability to persist the information related to an execution of an export
- The ModelExporter can save
  - The details used to generate the exported artifacts, i.e., the information provided through the exporter's wizard
    - Selected EPackages
    - Referenced EPackages
    - The directory URI
  - The location of the exported artifacts
- The Converter Framework stores this data using GenAnnotations

Going Deeper: GenAnnotations

- GenAnnotations can be added to any instance of GenBase, i.e. any object in a GenModel containment tree (including GenAnnotations themselves)
  - The GenBase class provides two methods to work with GenAnnotations
    - getGenAnnotations()
    - getGenAnnotation(String)
- A GenAnnotation, like its close relative EAnnotation, defines
  - A source attribute, which is typically used to store a URI representing the type of the annotation
  - A details map attribute to store name-value pairs
  - Containment and non-containment references to EObjects

Going Deeper: GenAnnotations

- The GenAnnotations are usually hidden in the Generator
  - You can unhide them by checking "Show Annotations" in the "Generator" menu



Agenda

- EMF in a Nutshell
- Working with URIs
- Working with Resources and ResourceSets
- Customizing the Code Generator
- Tuning for Performance and/or Memory Footprint
- Importers and Exporters
- ***Supporting Backward and Forward Compatibility***
- Summary

Backward/Forward Compatibility

- EMF provides mechanisms that can be used to support migration of data between different versions of a model (or between two different models, for that matter):
  - Use Ecore2Ecore and Ecore2XML to define a mapping between the different model(s) (versions)
  - Use Ecore2XMLExtendedMetaData with save/load options to handle unrecognized data
  - Use a resource handler to pre/post-process data that cannot be mapped automatically
- Source and target models must have different namespace URIs and XML references must not be based on feature names

Ecore2Ecore Mappings

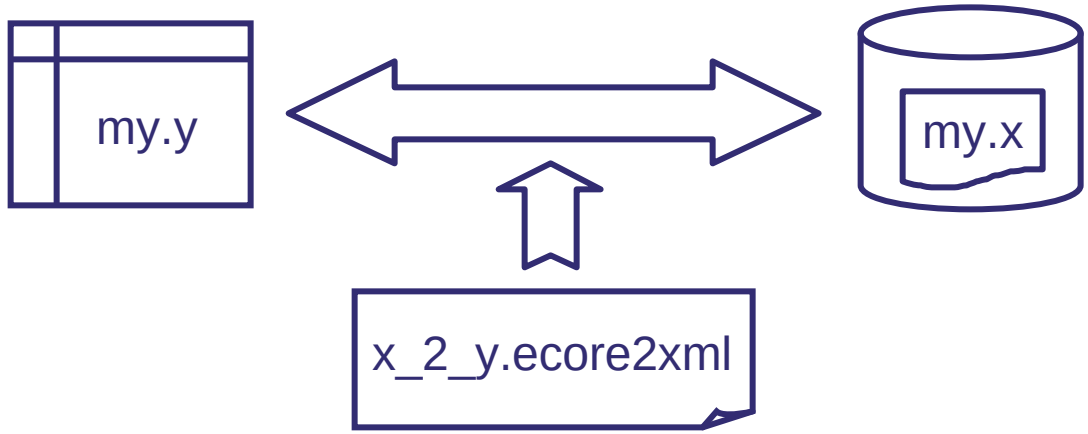
- Describe a mapping between two Ecore models
- Can be created from an Ecore (*.ecore) model via the ‘Map To Ecore…’ context menu item in the Package Explorer or Resource Navigator
- Typically used as a development-time artifact (only)
- Can include one-to-one, one-to-many, many-to-one, many-to-many, and one-sided mappings
- Only one-to-one and one-sided (one-to-none, none-to-one) mappings are useful for data migration

Ecore2XML Mappings

- Describe a mapping between an Ecore model and its XML representation
- Can be generated from an Ecore2Ecore (*.ecore2ecore) model via the ‘Generate Ecore to XML Mapping…’ context menu item in the Package Explorer or Resource Navigator
- Often used as a run-time artifact (in conjunction with Ecore2XMLExtendedMetaData)
- Can include one-to-one and many-to-one mappings, but only the former are useful for data migration

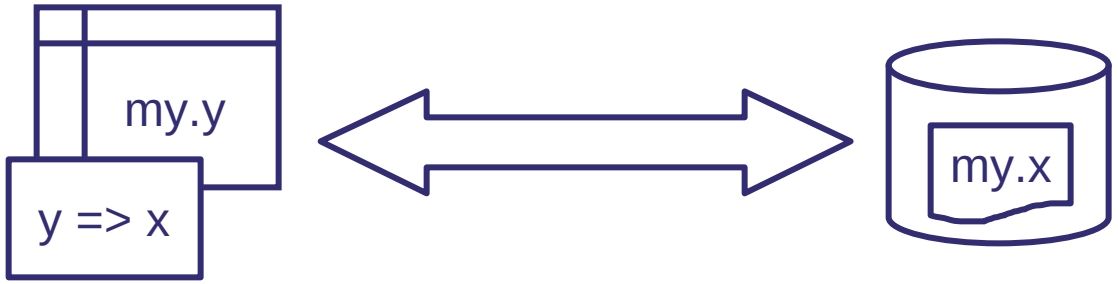
Ecore2XMLExtendedMetaData

- Can be used with the OPTION_EXTENDED_META_DATA save/load option defined on XMLResource to affect how data are serialized/deserialized
- Will consult registered Ecore2XML mappings to determine the XML representation of objects



OPTION_RECORD_UNKNOWN_FEATURE

- Load option defined on XMLResource
- Will record data for unrecognized types and features in an extension map (XMLResource#getEObjectToExtensionMap()) on the resource
- Recorded data will be serialized again when the resource is saved (unless the entries in the map have been cleared)



Resource Handlers

- Interface XMLResource.ResourceHandler defines callbacks that can be implemented to do special processing before/after a resource is loaded/saved
- Used with OPTION_RESOURCE_HANDLER save/load option defined on XMLResource
- Can support backward compatibility by extracting data from a resource's extension map after it is loaded
- Can support forward compatibility by inserting data into a resource's extension map before it is saved

Enabling Data Migration

1. Create a package registry and add entries which map the source namespace URI and target Ecore model location to the target package

2. Create an Ecore2XML registry and add an entry which maps the source namespace URI to the Ecore2XML mapping

3. Create an instance of Ecore2XMLExtendedMetaData based on the package and Ecore2XML registries; pass this extended metadata as the value for the XMLResource.OPTION_EXTENDED_META_DATA load/save option

Enabling Data Migration

4. Pass Boolean.TRUE as the value for the XMLResource.OPTION_RECORD_UNKNOWN_FEATURE load option

5. If required, pass an implementation of XMLResource.ResourceHandler as the value for the XMLResource.OPTION_RESOURCE_HANDLER load/save option

Agenda

- EMF in a Nutshell

- Working with URIs
- Working with Resources and ResourceSets
- Customizing the Code Generator
- Tuning for Performance and/or Memory Footprint
- Importers and Exporters
- Supporting Backward and Forward Compatibility

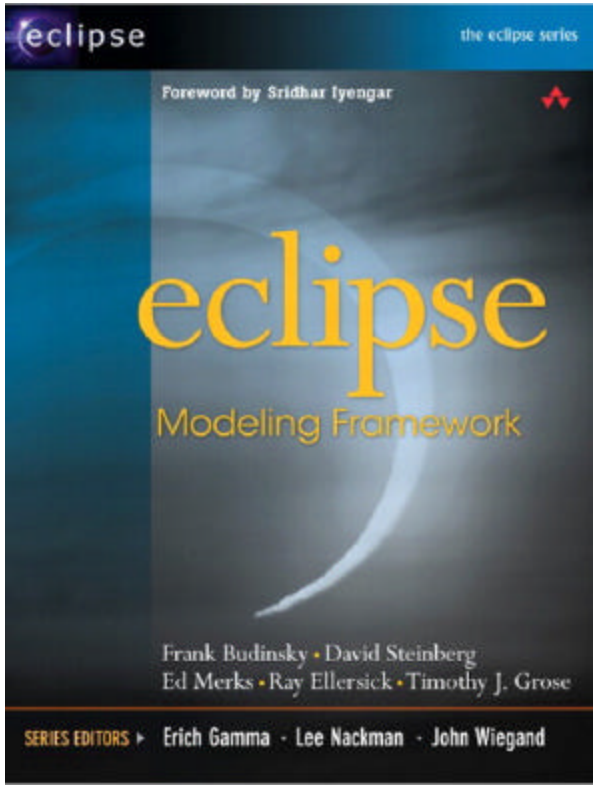
- ***Summary***

Summary

- We have seen examples of how flexible and extensible the Eclipse Modeling Framework is
- Hopefully you have gained greater insight into how some of the more “advanced” features of EMF can be exploited to meet your application’s needs

Resources

- EMF documentation in Eclipse Help
  - Overviews, tutorials, API reference
- EMF Project Web Site
  - http://www.eclipse.org/emf/
  - Overviews, tutorials, newsgroup, Bugzilla
- Eclipse Modeling Framework by Frank Budinsky et al.
  - Addison-Wesley; 1st edition (August 13, 2003)
  - ISBN: 0131425420.



Legal Notices

- IBM, alphaWorks, Lotus, Rational, Rational Rose and WebSphere are registered trademarks of International Business Corp. in the United States and other countries
- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both
- UML is a trademark of the Object Management Group
- Other company, product, or service names may be trademarks or service marks of others