Model-Driven Software Engineering

MDSE with the Eclipse Modeling Framework II

Dr. Jochen Küster  (jku@zurich.ibm.com)
Contents

- EMF Generation
- Runtime Framework of EMF
  - EMF Persistence
  - EMF Notification and Adapter
- EMF Validation
- EMF Editor Generation
- Setting EMF into MDSD Perspective
- Summary and Literature
EMF Generation
EMF Road Map

Domain Model

EMF Tools

EMF Codegen

generates

EMF Runtime

Java Implementation of Model

EMF Core

Java Implementation of Editor

EMF Edit

Eclipse Platform

Adapted and extended from: Effective Use of the Eclipse Modeling Framework, EclipseCon 2007
EMF as Architecture-Centric MDSD

Application

EMF Model

Infrastructure code

Business Logic Code manually written

Generative Architecture

Ecore

expressed in

Generator Templates

Infrastructure Components

EMF Generator
Generator Template for Classes

- For each EClass, a Java interface and an implementation class is generated
  - If the EClass has the interface property set as true, only an interface will be generated
- All interfaces of EClasses in one EPackage are generated in the same Java package
- Accessor methods for attributes and references
- Implementation classes include implementations of the accessor methods

```java
public interface Book extends EObject {
    String getTitle();
    void setTitle(String value);
}
```

```java
public class BookImpl extends EObjectImpl implements Book {
    ...
}
```
Generator Templates for One-Way References

public class PrivateLibraryImpl extends EObjectImpl implements PrivateLibrary{

    public void setOwner(Owner newOwner) {
        Owner oldOwner = owner;
        owner = newOwner;
        if (eNotificationRequired())
            eNotify(new ENotificationImpl(this, Notification.SET,
                                           PrivateLibraryPackage.PRIVATE_LIBRARY__OWNER, oldOwner, owner));
    } ....
}

- Setting of the owner instance variable
- Send notification to all registered observers (Observer pattern)
Generator Templates for Bidirectional References

```
public void setAuthor(Writer newAuthor) {
    if (newAuthor != author) {
        NotificationChain msgs = null;
        if (author != null)
            msgs = ((InternalEObject) author).eInverseRemove(this,
                BooksPackage.WRITER__BOOK, Writer.class, msgs);
        if (newAuthor != null)
            msgs = ((InternalEObject) newAuthor).eInverseAdd(this,
                BooksPackage.WRITER__BOOK, Writer.class, msgs);
        msgs = basicSetAuthor(newAuthor, msgs);
        if (msgs != null) msgs.dispatch();
    } else if (eNotificationRequired())
        eNotify(new ENotificationImpl(this, Notification.SET,
            BooksPackage.BOOK__AUTHOR, newAuthor, newAuthor));
}
```

- Other end of reference must be set by calling eInverseAdd
- Due to multiplicity 1 an already set author must be removed by eInverseRemove
- basicSetAuthor sets newAuthor reference from Book to Writer
A book can only be part of one library, so EMF takes care of this constraint.

If a book is added to a library, it is automatically removed from another library.
Generator Templates for Factories and Packages

- EMF automatically generates a package and a factory
- Package provides static constants and convenience methods used by other classes
- Factory should be used for instantiation (see Factory pattern by Gamma et al.)

```java
BooksFactory factory = BooksFactory.eINSTANCE;
Book book = factory.createBook();

Writer writer = factory.createWriter();
writer.setName("William Shakespeare");

book.setTitle("King Lear");
book.setAuthor(writer);
```
Runtime Framework of EMF
EMF Road Map

Domain Model

EMF Tools

EMF Codegen

generates

EMF Runtime

EMF Core

EMF Edit

Eclipse Platform

Adapted and extended from: Effective Use of the Eclipse Modeling Framework, EclipseCon 2007
EMF Persistence
Object Persistence in EMF

- EMF provides a persistence framework that can be used to serialize and deserialize objects from any model (not only Ecore models)
- Ecore models are (by default) serialized using XMI
- Ecore models can also be serialized using Essential MOF (EMOF)
- Models defined using an XML schema can be persisted as XML instance documents conforming to the schema
- The persistence framework is extensible, the way of serialization can be determined by providing a new form of serialization
The Concept of a Resource and URIs

- A resource is a container for one or more objects that are to be persisted together, with their contents
- A resource is associated with a URI (Universal Resource Identifier)
- A URI uniquely identifies a resource, references objects within resources and can be used for identifying packages
- The data can be in a file or on the internet or in a database
- A URI consists of a scheme, a scheme-specific part and an optional fragment
- In Eclipse, scheme “platform” is used for URIs to identify resources in e.g. the Eclipse workspace

platform:/resource/data/Example.books
The Concept of a Resource and URIs

- A URI fragment identifies part of the contents of the resource identified, separated by the “#” character
- A URI fragment can be used to identify objects in a resource

\[\text{platform:/resource/data/Example.books#@books.1}\]

- Resources can be assembled into ResourceSets
- Resources are persisted using load()/save()
- Each object in a resource is uniquely identified by a URI fragment

\[\text{platform:/resource/data/Example.books#@books.0}\]
Example of Object Persistence

// Create a resource set.
ResourceSet resourceSet = new ResourceSetImpl();

// Register the default resource factory -- only needed for stand-alone!
resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap().put(
    Resource.Factory.Registry.DEFAULT_EXTENSION, new
    XMIResourceFactoryImpl());

// Get the URI of the model file.
URI fileURI = URI.createFileURI(new File("mylibrary.xmi").getAbsolutePath());

// Create a resource for this file.
Resource resource = resourceSet.createResource(fileURI);

// Add the book and writer objects to the contents.
resource.getContents().add(book);
resource.getContents().add(writer);

// Save the contents of the resource to the file system.
try
{
    resource.save(Collections.EMPTY_MAP);
}
catch (IOException e) {}
Example of Object Persistence

<?xml version="1.0" encoding="UTF-8"?>
<books:Library xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:books="http://books/1.0">
  <writers author="//@books.0" title="EMF Modeling Framework"/>
  <writers author="//@books.1" title="Model-Driven Software Development"/>
  <books book="//@writers.0" name="Dave Steinberg"/>
  <books book="//@writers.1" name="Markus Voelter"/>
</books:Library>

- Fully automatic persistence of object model
- Takes care of all model elements such as:
  - multiplicity-1-attributes: are serialized as an XML attribute
  - multiplicity-many attributes: are serialized as nested XML elements
  - non-containment references: are mapped using URI fragment paths
  - containment references: contained elements are nested within the element corresponding to the container
  - cross-document references: are persisted using XML elements with href
Model Deserialization

- The EMF persistence framework uses SAX XML parser for loading documents
- Namespace URIs are read and used for locating Ecore packages
- Using a package, the loader uses the factory in the package to create EMF objects
- For each XML attribute and element encountered, the corresponding Ecore construct is located by name in the package

Examples:
- For an EAttribute, it uses the corresponding factory to convert the string value into an object and sets the value of the attribute
- For an EReference, it locates the corresponding EObject using the XML attribute value
EMF Notification and Adapter
Notification and Adapter in EMF

- Notification and adapter framework of EMF is basis for model change notifications
- Every EMF class is a Notifier
  - Can send notifications whenever attribute or reference is changed
- EMF objects can be observed in order to update views or other forms of dependent objects
- In EMF such observers are called “adapters”
- Adapters can observe but they can also implement additional behavior (extend the observed class)
- EContentAdapter is attached to the root object and will then attach itself to all the contents
- Adapters are used extensively in EMF and are foundation of UI and command support
public class ChangeCounterAdapter extends AdapterImpl {
    public static int bookCount;

    public void notifyChanged(Notification notification) {
        if (notification.getNotifier() instanceof Book) {
            ++bookCount;
        }
    }
}

ChangeCounterAdapter adapter = new ChangeCounterAdapter();

BooksFactory factory = BooksFactory.eINSTANCE;
Book book = factory.createBook();
book.eAdapters().add(adapter);

book.setTitle("Design Patterns");
Writer writer = factory.createWriter();
book.setAuthor(writer);
System.out.println("Book changes: " + adapter.bookCount);
EMF Validation
EMF Validation Framework

- The validation framework allows to define and validate that instances of an EMF model are valid
- The validation framework supports constraints and invariants
- The validation framework supports explicitly calling or implicitly calling of a validation
- Code generation of EMF generates, based on constraints and invariants, suitable code for this purpose
Constraints and Invariants

- **Invariant** is a statement that must be always valid
  - Example: The attribute pages of a class Book is always nonnegative.

- **Constraint** is a statement that must be valid at some point in time
  - Example: Method precondition or method postcondition.

- Compare OCL constraints in earlier lecture
Defining Constraints and Invariants in EMF

- Constraints are defined as annotations
- Invariants are defined as operations with EBoolean return type
- Code generation is then adapted automatically
- Manual implementation of constraints and invariants required
Invoking Validation

- Validation can be invoked manually using a UI
  - After generating model, edit, and editor, use the Validate menu option
- Validation can be or automatically in the code
- Validation also includes conformance to intrinsic constraints
  - Multiplicities
  - Cross-referenced objects
  - UID validity
EMF Editor Generation
EMF Road Map

EMF Tools

Domain Model

EMF Codegen

generates

EMF Runtime

Java Implementation of Model

EMF Core

Java Implementation of Editor

EMF Edit

Eclipse Platform

Adapted and extended from: Effective Use of the Eclipse Modeling Framework, EclipseCon 2007
Motivation for Editor Generation

- Many times a data object of a software system has to be visualized and also edited

- Programming of such functionality involves many repetitious tasks:
  - Visualizing elements of the data objects
  - Providing editing capabilities
  - Updating visualizations
  - Saving new input from the user back into the data object

- All of these functionalities can be generated by EMF Editor Generation based on the domain model

- Customization is possible and often required
Editor Generation for the Example
Behind EMF Editor Generation: Model-View-Controller Pattern

- Separation of data, presentation and interaction into different classes
- Many applications over the years
Model-View-Controller Pattern (Details)

- presents data for the user
- takes care of the graphical UI

- encapsulates data and offers methods for data manipulation
- notifies views and controller if data has changed
- processes user input
- translates events and delegates to model or view
## Model-View-Controller Architecture in EMF

<table>
<thead>
<tr>
<th>EMF View</th>
<th>Content Provider &amp; Label Provider</th>
<th>Domain Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>presents data for the user</td>
<td>provides content to view</td>
<td>encapsulates data and offers methods for data manipulation</td>
</tr>
<tr>
<td>takes care of the graphical UI</td>
<td>provides labels for the content to view</td>
<td>notifies views and controller if data has changed</td>
</tr>
<tr>
<td>needs to know which data to present</td>
<td></td>
<td></td>
</tr>
<tr>
<td>needs to know how to represent the data</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example for Viewers in EMF

Tree Viewer

Property Viewer
Item Providers

- Item providers are EMF adapters (see previous explanations)
- Item providers “provide” functionality for individual editable “items”
- Item providers perform the following major roles
  - Implement content and label provider functions
  - Provide a property source for EMF objects
  - Forward EMF change notifications to viewers
  - Act as a command factory (not further covered)
- Item providers usually subclass the EMF ItemProviderAdapter class
Content and Label Item Providers

- EMF.Edit connect EMF core with Eclipse UI Framework (JFace)
- EMF.Edit provides generic content and label provider implementation classes
  - AdapterFactoryContentProvider and AdapterFactoryLabelProvider
- They delegate their implementation to specific Content and Label providers

- Delegation example: ITreeContentProvider.getChildren()
  - AdapterFactoryContentProvider calls adapt on the ItemProviderAdapterFactory
  - ItemProviderAdapterFactory returns ItemProvider for specified object
  - AdapterFactoryContentProvider delegates to the getChildren() method of the ItemProvider

- Similar approach is used for label providers
EMF Edit Generation

- EMF Edit generation generates UI independent part of editing support classes
- EMF Edit generation generates
  - A set of typed item provider classes, one for each class in the model
  - An item provider adapter factory class that creates the generated item providers
  - A plug-in class that includes methods for locating resources
  - A directory of icons, one for each model class
  - Further supporting files
- Most important are the item provider classes which implement (amongst others) the behavior for content and label providers
EMF Editor Generation

- EMF Editor generation generates a fully functional editor plug-in
- Editor allows to view instances of the model, add, remove, cut and paste objects, modify objects using property sheets
- Editor can be customized easily
- Editor can be run when starting a runtime workbench of Eclipse
Editor Generation for the Example
Setting EMF in MDSD perspective
EMF as Architecture-Centric MDSD

Application

- EMF Model
- Infrastructure code
  - Business Logic Code
    manually written

Generative Architecture

- Ecore
- Generator Templates
- Infrastructure Components

EMF Generator

directed by

expressed in
EMF as Architecture-Centric MDSD

- Domain Model created as EMF model
- Generator Templates
  - for generating code
- Architectural patterns and design patterns used
  - Model-view-controller
  - Observer pattern
  - Factory pattern
- Infrastructure generation
  - persistence framework
  - notification and adapter mechanism
  - common command framework
  - validation framework
Summary of Lecture and References

- EMF is one example of architecture centric MDSD
- EMF generation allows to quickly generate Java implementation for a domain model
- EMF generation applies well-known architectural patterns
- EMF persistence framework is used to serialize and deserialize models
- EMF validation framework ensures that instances are valid
- EMF editor generation can be used to generate a simple editor for EMF models

References:
- Eclipse EMF tutorials online.