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
• 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)
MDA Principles

• Current engineering approaches
  – Models not part of engineering processes (documentation)

• MDA approach
  – Models as first class entities
  – Need for dedicated tools

• MDA basic concepts
  – System – real world situation
  – Model – abstraction of a system
    • Describe a given aspect of the system
  – Metamodel – rules to define an abstraction

MDA?

• Doubts exist about the ability of MDA to deliver on its promises:
  – Ambitiousness of vision
  – Model expressiveness vs. complexity
  – Availability of implementations (“vaporware”)
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 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 the runtime information)
  - Improve extensibility
  - Provide a UI layer

- What is an 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)
- 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 specification; UML2 based
  – EMF is approximately the same functionality
    • Significant contributor to the spec; adapting to it
Who is Using EMF Today?

• Eclipse projects
  – Foundation for the Modeling Project: Graphical Modeling Framework (GMF), EMF Ontology Definition Metamodel (EODM), UML2…
  – Other uses: Web Tools Platform (WTP), Test and Performance Tools Platform (TPTP), Business Intelligence and Reporting Tools (BIRT), Data Tools Platform (DTP), Visual Editor (VE)…
• Commercial offerings
  – IBM, Borland, Oracle, …
• Applied sciences
  – Darmstadt University of Technology, Mayo Clinic College of Medicine, European Space Agency…
• Large open source community
  – Over 1,000,000 download in 2006

Agenda

• What is Modeling?
• Defining a Model with EMF
• EMF Architecture
• Code Generation
• Programming with EMF
• 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

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

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

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

UML Class Diagram

- Built-in support for Rational Rose®
- UML2 support available with UML2 (from MDT)
XML Schema

```xml
<?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
What is Eclipse?

- Eclipse is a universal platform for integrating development tools
- Open, extensible architecture based on plugins

![Diagram showing Eclipse components]

EMF Framework

- Object oriented framework
  - **Edition** of new EMF models
    - Ecore metamodel
    - XML as canonical representation
  - **Import** of existing models
    - UML model
    - XML schema
    - annotated Java code
  - **Export** of Ecore-based models
    - Java code generation
      - Accessor methods
      - Operations skeletons
Agenda

• What is Modeling?
• Defining a Model with EMF
• EMF Architecture
• Code Generation
• Programming with EMF
• Summary
EMF Components

- Core Runtime
  - Notification framework
  - Ecore metamodel
  - Persistence (XML/XMI), validation, change model
- EMF.Edit
  - Support for model-based editors and viewers
  - Default reflective editor
- Codegen
  - Code generator for application models and editors
  - Extensible model importer/exporter framework

Creating the Ecore Model

- Representing the modeled domain in Ecore is the first step in using EMF
- Ecore can be created
  - Directly using the EMF editors
  - Through a graphical UI provided by external contributions
  - By converting a model specification for which a Model Importer is available
- Model Importers available in EMF
  - Java Interfaces
  - UML models expressed in Rational Rose® files
  - XML Schema
- Choose the one matching your perspective or skills
Ecore

• EMF’s metamodel (model of a model)

![Ecore Diagram]

Ecore

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

![Ecore Application Models Diagram]
Ecore

- Persistent format is XMI (.ecore file)

```
<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 Essential MOF XMI (.emof file)

Partial List of Ecore Data Types

<table>
<thead>
<tr>
<th>Ecore Data Type</th>
<th>Java Primitive Type or Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBoolean</td>
<td>boolean</td>
</tr>
<tr>
<td>EChar</td>
<td>char</td>
</tr>
<tr>
<td>EFloat</td>
<td>float</td>
</tr>
<tr>
<td>EString</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>EByteArray</td>
<td>byte[]</td>
</tr>
<tr>
<td>EBooleanObject</td>
<td>java.lang.Boolean</td>
</tr>
<tr>
<td>EFloatObject</td>
<td>java.lang.Float</td>
</tr>
<tr>
<td>EJavaObject</td>
<td>java.lang.Object</td>
</tr>
</tbody>
</table>

- Ecore data types are serializable and custom data types are supported
EMF Tools: Model Import and Generation

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

Direct Ecore Modeling

- Ecore models can be created directly
  - Sample Ecore editor (in EMF)
  - Ecore Tools graphical editor (from EMFT)
Unifying Java, XML and UML Technologies

• The Model Importers available in EMF were carefully chosen to integrate today’s most important technologies
• All three forms provide the same information
  – Different visualization/representation
  – The application’s “model” of the structure
• From a model definition, EMF can generate
  – Java implementation code, including UI
  – XML Schemas
  – Eclipse projects and plug-in

Typical EMF Usage Scenario

• Create an Ecore model that represents the domain you are working on
  – Import UML (e.g. Rose .mdl file)
  – Import XML Schema
  – Import annotated Java interfaces
  – Create Ecore model directly using EMF’s Ecore editor or a graphical editor
• Generate Java code for model
• Prime the model with instance data using generated EMF model editor
• Iteratively refine model (and regenerate code) and develop Java application
  – You will use the EMF generated code to implement the use cases of your application
• Optionally, use EMF.Edit to build customized user interface
EMF Model Importers

- **UML**
  - Rational Rose .mdl file
  - Eclipse UML2 project provides importer for .uml2
- **Annotated Java**
  - Java interfaces representing modeled classes
  - Javadoc annotations using @model tags to express model properties not captured by method declarations
  - Lowest cost approach
- **XML Schema**
  - Describes the data of the modeled domain
  - Provides richer description of the data, which EMF exploits
- **Ecore model (*.ecore file)**
  - Just creates the generator model (discussed later)
  - Also handles EMOF (*.emof)

Ecore Model Creation

- An Ecore model is created within an Eclipse project via a wizard
- Input: one of the model specifications
- Output:
  - `modelName.ecore`
    - Ecore model file in XMI format
    - Canonical form of the model
  - `modelName.genmodel`
    - A "generator model" for specifying generator options
    - Contains decorators for Ecore model elements, providing details that would otherwise pollute the model (e.g. target directories for code generation)
    - EMF code generator is an EMF .genmodel editor
    - Automatically kept in synch with .ecore file
Ecore Model Editor

• A graphical editor is a better approach
  – GMF Ecore Diagram Example
    (http://www.eclipse.org/gmf/)

Agenda

• What is Modeling?
• Defining a Model with EMF
• EMF Architecture
• Code Generation
• Programming with EMF
• Summary
Generated Model Code

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

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

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

Summary of Generated Artifacts

• Model
  – Interfaces and classes
  – Type-safe enums
  – Package (metadata)
  – Factory
  – Switch utility
  – Adapter factory base
  – Validator
  – Custom resource
  – XML Processor

• Edit (UI Independent)
  – Item providers
  – Item provider adapter factory

  ```javascript
  Editor
  Model Wizard
  ```

• Tests
  – Test cases
  – Test suite
  – Stand-alone example
Agenda

• What is Modeling?
• Defining a Model with EMF
• EMF Architecture
• Code Generation
• Programming with EMF
• Summary

#assignments

• Installation

  – Using Eclipse Install Manager, point your install Manager at this site:
    Help > Software Updates... > Available Software > Add Site...

  – Location:
    http://download.eclipse.org/modeling/emf/emf/
    updates/releases/
#assignments

- Introduction to EMF Tutorial:
  
  http://eclipsesource.com/blogs/tutorials/emf-tutorial/

Agenda

- What is Modeling?
- Defining a Model with EMF
- EMF Architecture
- Code Generation
- Programming with EMF
- 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 integrates 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