Eclipse Modeling Framework – What, why and how
Eclipse Modeling Framework

• **What** – framework for developing three-tier applications
  – User interface
  – Business objects and logic
  – Persistence

• **Why** – a lot of the code in an application is trivial use of non-trivial code templates and patterns
  – get and set methods
  – event notification
  – data-driven GUI
  – XML-based storage

• **How** – modeling, code generation and customization
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)
What Have People Said About EMF?

- EMF represents the **core subset** that's left when the non-essentials are eliminated. It represents a **rock solid foundation** upon which the more ambitious extensions of UML and MDA can be built.
  
  - Vlad Varnica, OMONDO Business Development Director, 2002

- EMF **provides the glue between the modeling and programming worlds**, offering an infrastructure to use models effectively in code by integrating UML, XML and Java. EMF thus fits well into [the] Model-Driven Development approach, and is **critically important for Model-Driven Architecture**, which underpins service-oriented architectures [SOA].
  
  - Jason Bloomberg, Senior analyst for XML & Web services, ZapThink, 2003

- EMF was chosen because it (a) provides a **lightweight, pragmatic approach** to modeling with **very low entry cost** and is thus suitable for rapid prototyping, (b) unifies key technologies such as Java and XML, and (c) integrates well into Eclipse.
  
  - Bruch, Bockisch, Schäefer, Mezini, Darmstadt Univ. of Technology, 2005

- [As] a consultant with fiduciary responsibility to my customers, [...] given the **enormous traction** that Eclipse has gathered, we have to view the EMF metadata management framework as the **de facto standard**.
  
  - David Frankel, as seen in Business Process Trends, March 2005
EMF – Why?

• Implementation of the data structures in the business (middle) tier is both *tricky* and *repetitive*
  – *tricky*, because it’s not easy to ensure consistency and generate appropriate change notifications
  – *repetitive*, because the code has the same structure for all domains and the differences are systematic

• The logic and implementation of the data structures can to a large extent be controlled by a *model of the data*

• Use your brain for what’s unique, rather than what’s the same
EMF – Why?

• The implementation is mostly trivial use of non-trivial code templates and patterns
  - When you know the patterns, using them is just a matter of being systematic
  - The knowledge is the pattern itself, which is non-trivial

• Example:
  - get and set methods for JavaBeans, w/generation of change notifications

    ```java
    private <type> <propertyName>;
    public <type> get<PropertyName>() {
        return <propertyName>;
    }
    public void set<PropertyName>(<type> newValue) {
        if (newValue != <propertyName>) {
            <type> oldValue = <propertyName>;
            <propertyName> = newValue;
            pcs.firePropertyChange("<propertyName>", oldValue, newValue);
        }
    }
    ```

• However, this is a lot more complicated for associations with inverses, like parent-child relationships!
EMF – Why?

• Industrial knowledge and practice has given us
  – conventions for naming and typing
  – various implementation techniques
  – high-level coding patterns
  – large and fairly complete libraries and frameworks

• EMF puts all this together, so you can focus on what’s unique in the specific domain that you’re working on

• Compared to other frameworks, EMF gives a fairly large increase in productivity relative to complexity
EMF – What?

• First and foremost a practical tool for programmers
  – a framework with complete functionality
    that may be customized for specific needs
  – tools for authoring models, generating code and managing data

• Tre-tier applikasjon

• Does not depend on Eclipse platform, can be used standalone!
**EMF – Three-tier application**

- A typical business layer includes
  - methods for navigating domain data, e.g. using get methods
  - build and update data structures and ensure validity and consistence
  - generate change notifications for listeners

- EMF supports this by means of
  - generic methods in EObject superclass and template-based code generation
  - automatic handling of inverse associations and generation of change notifications
EMF – Advanced features

• Validation
  – validation is separated from core API
  – consistency rules can be formulated in Java, OCL, ...
  – code templates can be customized to support other languages

• Transactions
  – data structures modifications can be managed as transactions
  – global consistency can be ensured

• Client-server architecture by means of CDO project
  – shared (and often large) data structure is managed by server
  – each client sees and manages part of shared data
  – the server notifies relevant clients when data changes
**EMF - Three-tier application**

- **Persistence should support**
  - safe storage of graph of objects
  - well-defined and open formats
  - various types of storage, e.g. XML files og databases

- **EMF supports this by means of**
  - resource concept
  - XML serialization and database support (in Teneo project)
**EMF - Three-tier application**

- **User interface**
  - access to data in the business layer
  - updates when data changes
  - use of standard GUI components and application structure

- **EMF supports this by means of**
  - classes for accessing data and reacting to changes
  - integration with "data binding" framework
  - commands and generalized undo/redo
EMF Architecture

EMF Tools

EMF Runtime

Eclipse Platform

Application

Model

Editor

CodeGen

Core

Edit

Generates

Eclipse Platform
EMF – How?

- Typical workflow
  - Model the data
    - from scratch or by importing Java interface declarations, XML Schema eller UML class diagram
    - edit with tree-based or graphical editor
  - Validate with example data
    - check if the model allows with actual data
  - Generate and customize code
    - interface declarations og implementation classes, editor support and tests
    - add custom code to generated methods and classes
  - Implement (the rest of) the applikasjonen
    - utilize generated editor support
    - bind GUI components to EMF data
    - (configure as Eclipse plugin)
Create EMF model

- Eclipse has several wizards
  - Empty EMF Project
  - Ecore Model – base model
  - Ecore Diagram – diagram with base model
Ecore

- EMF’s metamodel (model of a model)
Ecore modeling concepts

- **EClass** – Ecore’s class concept
- **EObject** – Ecore instances, i.e. Eclass
- **EPackage** – a container for Eclasses, at least one for each model
- **EReference** – directed association, models a relation between an EObject and one or more EObjects
- **EAttribute** – attribute, models a relation between an EObject and ”simple” values, i.e. non-EObjects
- **EDataType** – custom data types, based on existing Java classes, for use with attributes
- **EOperation** – method declarations
- **EAnnotation** – application-specific metadata, e.g. the body of generated EOperations
Models (and data) can have many forms

- ecore is base model (stored as XML)
- graphical modelling with ecorediag
- textual modelling with EMFatic and HUTN
EMF Runtime

- Persistence and serialization of model data
  - Proxy resolution and demand load
- Automatic notification of model changes
- Bi-directional reference handshaking
- Dynamic object access through a reflective API
- Runtime environments
  - Eclipse
    - Full IDE
    - RCP
  - Standalone Java
Instance data

• EMF support creation of an EObject graph, using a so-called "reflective editor"
  – only based on the model, does not need generated code
  – provides model-specific editing commands, in addition to generic ones like delete, cut & paste, drag’n drop
  – ensures a valid and consistent EObject graph

• Useful for
  – checking that the model can represent the real world scenarios
  – checking that the model supports necessary navigation in the graph
  – creating sample data for prototypes

• Instance data must always be contained in a root object
  – all EObjects must be directly or indirectly contained in the root EObject
  – can be necessary to introduce an artificial root EClass
    (I typically call it UoD, for Universe of Discourse)
Persistence and Serialization

- 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
Ecore code/runtime concepts

- **Resource**
  - The basic storage unit, typically a file
  - contains hierarchy of EObjects with links across branches
- **Resource URI**
  - an identifier for Resources, typically file-based or workspace-relative
  - with a fragment, you can identify an EObject within a Resource
  - URIs are used to link EObjects in different Resources
- **ResourceSet**
  - contains a set of linked Resources
  - Resources are loaded into ResourceSets manually or automatically
  - when links to other Resources are discovered, the other Resources are automatically loaded into the same ResourceSet
- **EFactory** – manages creation of EObjects
- **EPackage** – provides access to metadata
What can you do with the model?

• **Manipulate**
  – store, validate, manipulate, transform with generic tools

• **Generate Java code**
  – several template-based solutions for generating EMF-dependent code, POJO (plain old Java objects) or even C++

• **Build client-server systems**
  – CDO supports distributing and synchronizing data across a network

• **Build graphical tools**
  – GMF and Eugenia lets you quickly make graphical editor

• **Build textual language**
  – Xtext and EMFTText lets you quickly build textual language
Code generation

• Code generation is controlled by a generator model
  – genmodel-related code-oriented attributes to the base Ecore model
  – includes important aspects of the Java code, e.g. package and class naming
  – code templates can be customized
• Resulting model code
  – interface declaration (<EClass>)
  – implementation classes (<EClassImpl> implements <EClass>)
  – factory class (<ePackage>Factory extends EFactory)
  – metadata class (<ePackage>Package extends EPackage)
• Edit code
  – Eclipse-independent support for GUI
• Editor code
  – Eclipse-dependent and model-specific editor, based on generic editor
• Test code
  – code for testing model code, typically for checking the validity of customization
Code customization

• Many reasons for customizing code
  – implement declared methods
  – extra validation code in set methods
  – convert to/from EDatatype values and String-objekter
  – implement extra interfaces
  – ...

• The general problem of code generation is regeneration
  – what happens with edited code, if it is regenerated?
  – custom code must survive regeneration!

• The EMF solution
  – generated code is marked with @generated comments, and by changing to @generated NOT, the marked code will be spared
  – custom fields and methods are not touced by regeneration
Integration with XML

• Why use XML when you have Ecore?
  – existing XML-based data and protocols may need to be supported
  – both support defining a schema, and the languages are fairly compatible
  – XML is important for BPM, e.g. web services and BPEL

• XML to Ecore
  – an XML schema (xsd file) may be translated to a generator model (genmodel)
  – the mapping between XML and Ecore is stored as Ecore model annotations
  – instance data may be saved as XML Schema compliant XML

• Ecore to XML
  – a generator model may be exported to an XML Schema
  – XML-specific types may be used, to avoid "littering" the schema with Ecore
  – by adding annotations to the Ecore model, the XML can be customized
  – the XML Schema may subsequently be used in appropriate contexts,
    e.g. for annotating BPMN connections
Write the rest of the application

• The EMF runtime does not depend on the Eclipse platform, but utilizes it if present
  – The application logic may be prototyped within Eclipse, but run outside
  – EMF can easily be used standalone, e.g. in web applications

• The editor is partly dependent on Eclipse
  – generic editor support, e.g. commands and undo/redo support can be used outside Eclipse
  – the integration with Eclipse is naturally Eclipse-dependent

• EMF – lots of possibilities...
Related projects

- Ecore Tools – standard graphical Eclipse editor for Ecore models
- Data binding – connects EMF data to GUI components
- Validation, Query og Transaction – advanced techniques for managing Ecore models and data
- Teneo – database-based persistence
- CDO – sharing of server-hosted data among several clients
- GMF – framework for building graphical editors based on Ecore-modeling (used by Ecore Tools)
- Xtext – textual syntax and editor support Ecore models and data
- Javascript – support for javascripting Ecore models
- EMFT – collection of sub-projects related to EMF
- Amalgam – collection of tools for building DSLs
Resources

• Links
  – http://www.eclipse.org/emf - EMF home page
  – http://www.eclipse.org/downloads/ - download page, including Eclipse Modeling Tools distribution, containing lots of modeling-related plugins

• Newsgroup
  – eclipse.tools.emf@new.eclipse.org

• Book