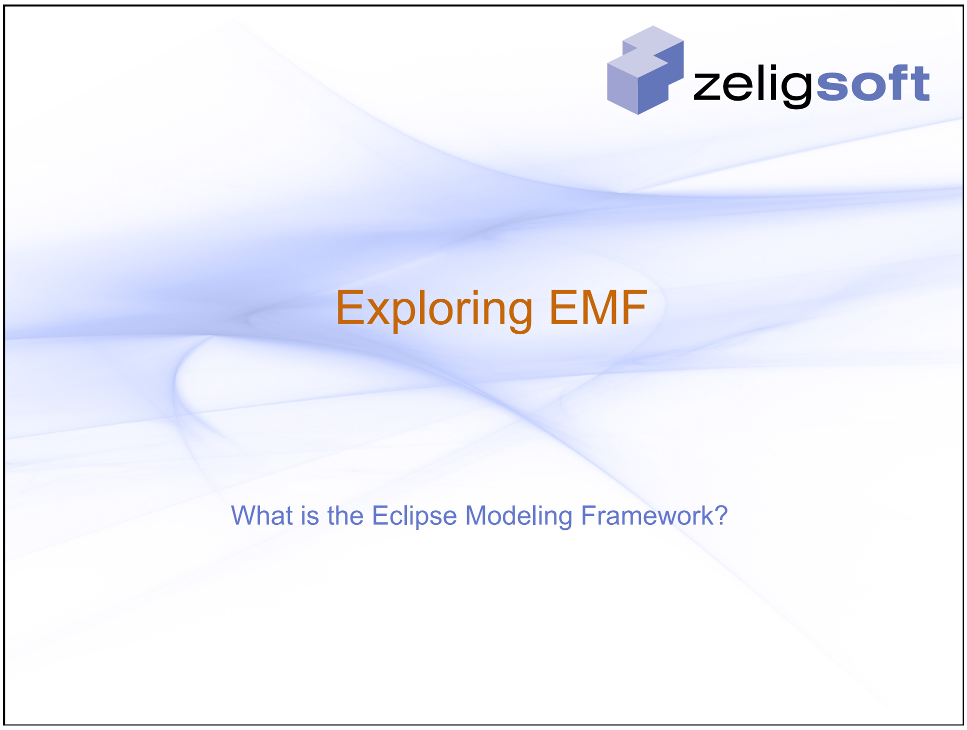
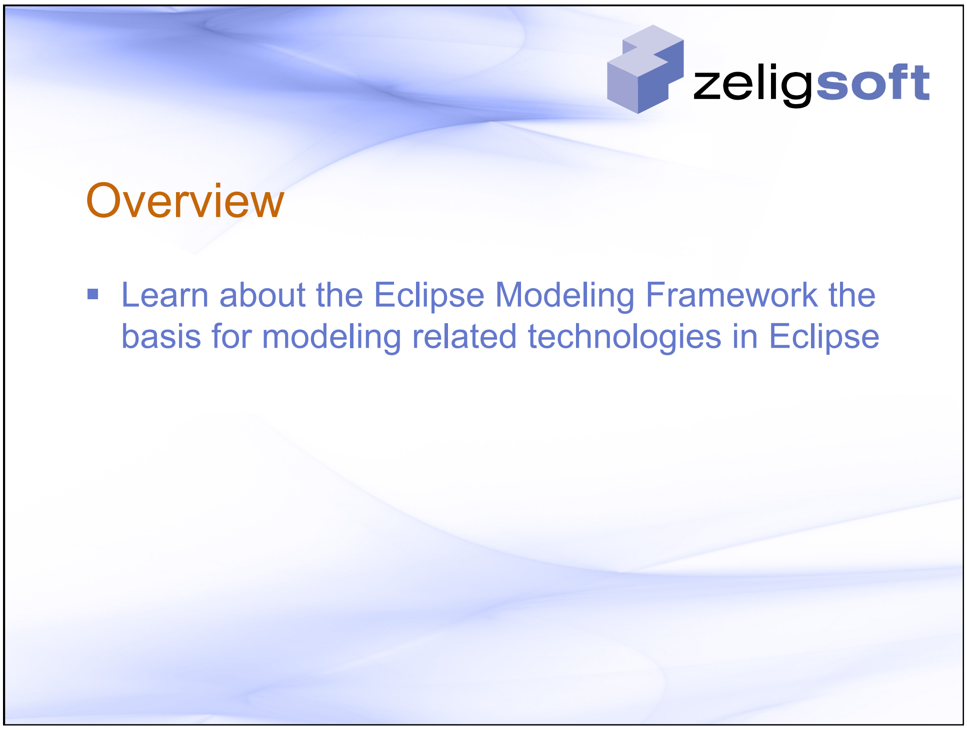


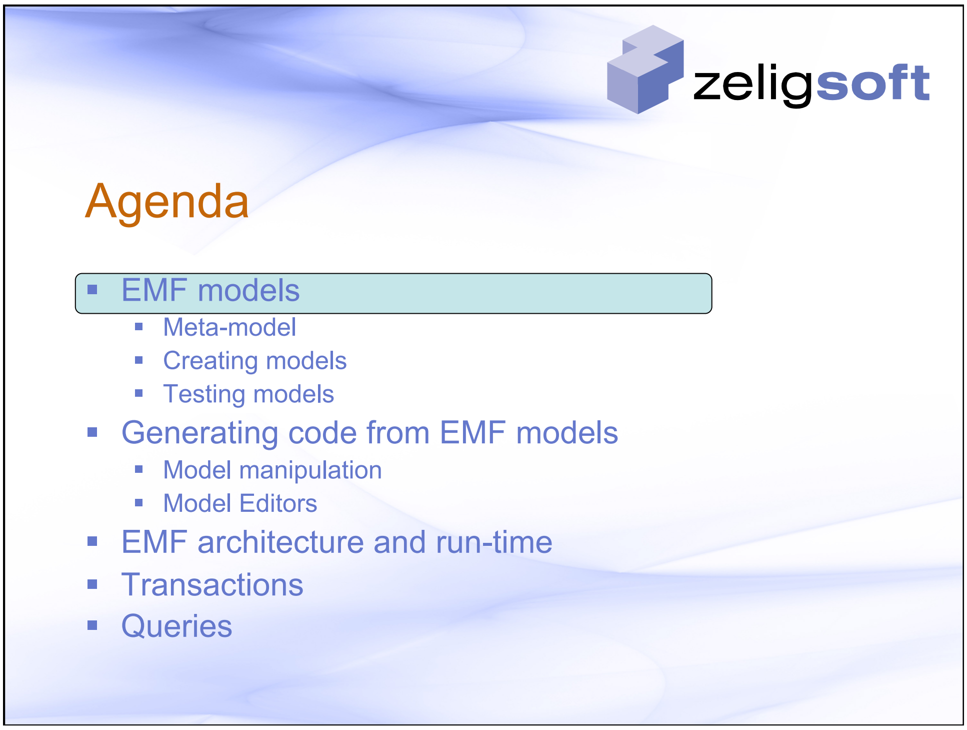
Exploring EMF

What is the Eclipse Modeling Framework?



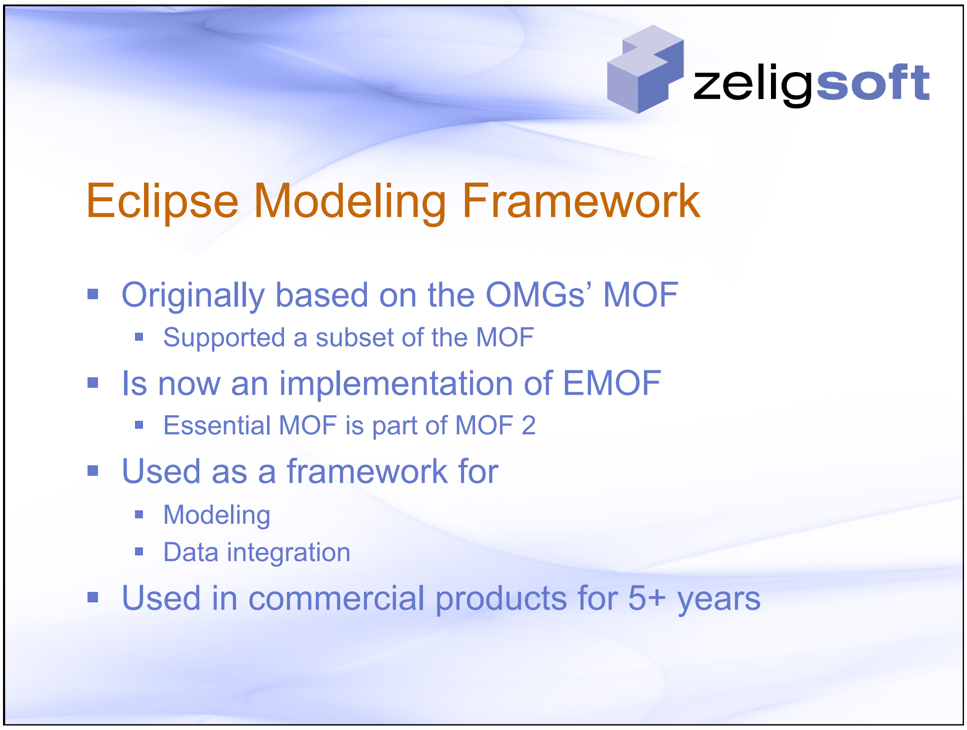
Overview

- Learn about the Eclipse Modeling Framework the basis for modeling related technologies in Eclipse



Agenda

- EMF models
  - Meta-model
  - Creating models
  - Testing models
- Generating code from EMF models
  - Model manipulation
  - Model Editors
- EMF architecture and run-time
- Transactions
- Queries



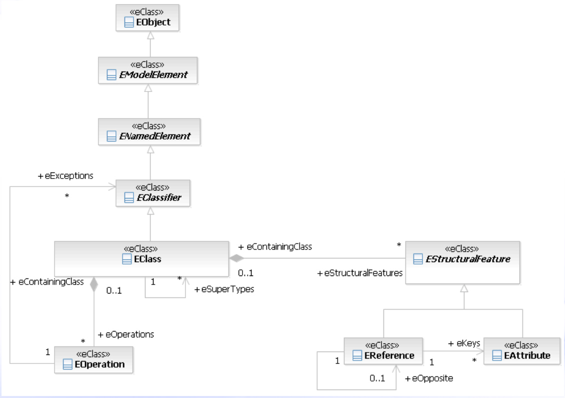
Eclipse Modeling Framework

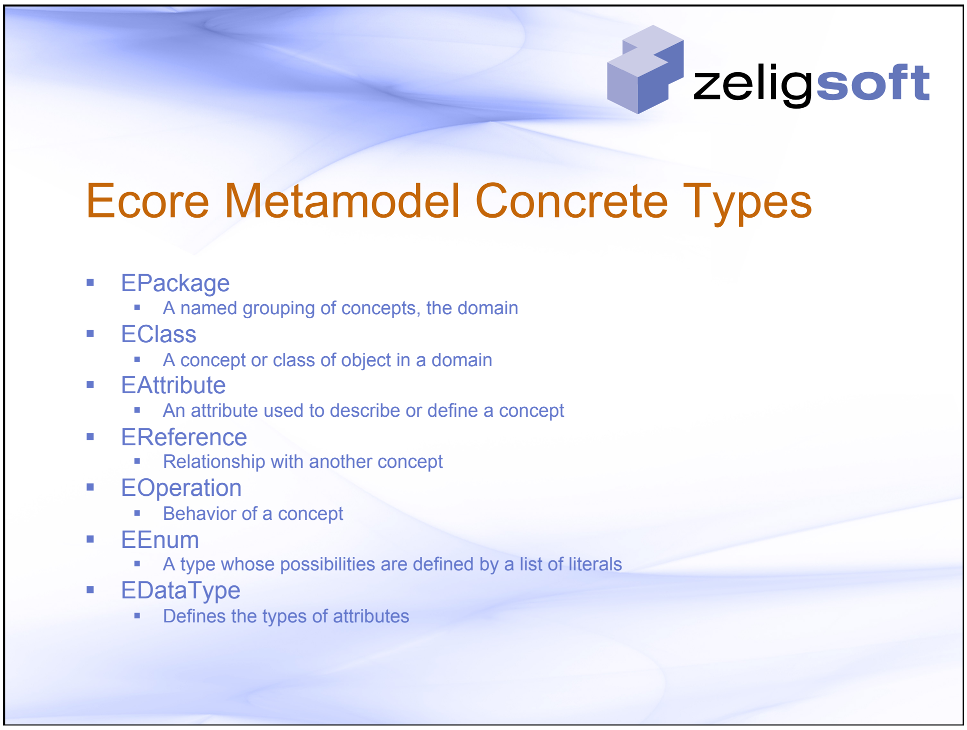
- Originally based on the OMGs' MOF
  - Supported a subset of the MOF
- Is now an implementation of EMOF
  - Essential MOF is part of MOF 2
- Used as a framework for
  - Modeling
  - Data integration
- Used in commercial products for 5+ years

What is an EMF Model?

- Description of data for a domain/application
  - The attributes and capabilities of domain concepts
  - Relationships between the domain concepts
  - Cardinalities of attributes and relationships
- It defines the *metamodel* or abstract syntax for your domain
- Defined in the Ecore modeling language

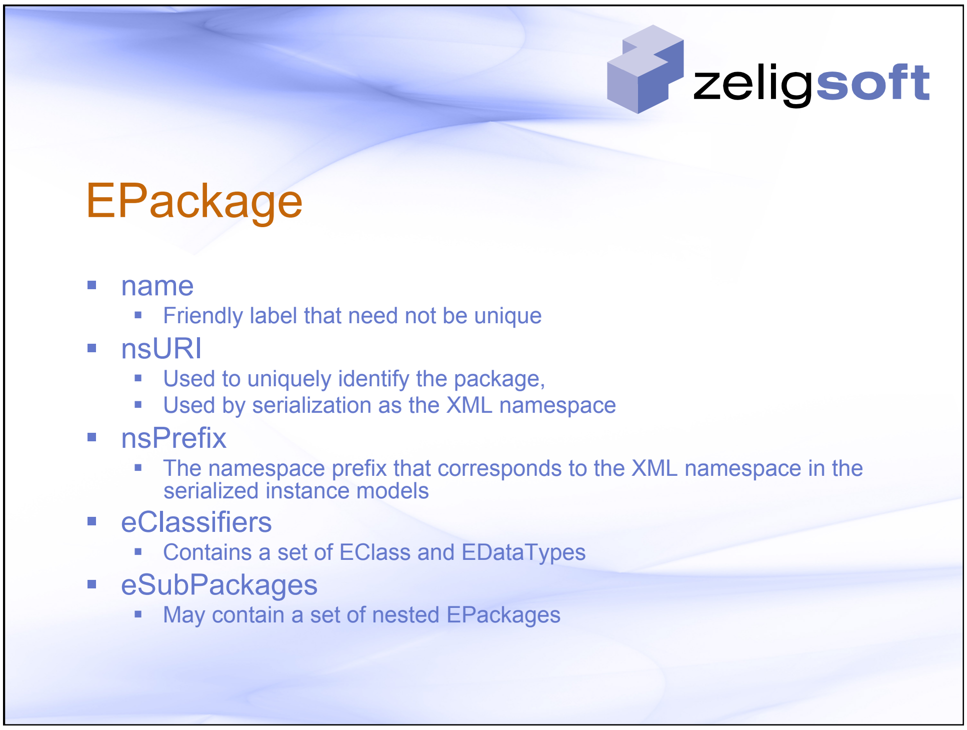
The Ecore Metamodel





Ecore Metamodel Concrete Types

- EPackage
  - A named grouping of concepts, the domain
- EClass
  - A concept or class of object in a domain
- EAttribute
  - An attribute used to describe or define a concept
- EReference
  - Relationship with another concept
- EOperation
  - Behavior of a concept
- EEnum
  - A type whose possibilities are defined by a list of literals
- EDataType
  - Defines the types of attributes

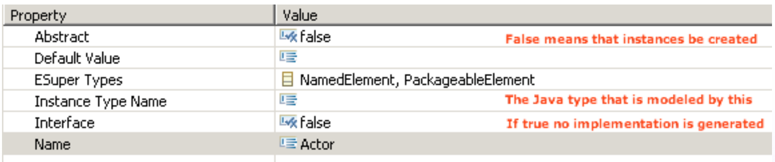


EPackage

- name
  - Friendly label that need not be unique
- nsURI
  - Used to uniquely identify the package,
  - Used by serialization as the XML namespace
- nsPrefix
  - The namespace prefix that corresponds to the XML namespace in the serialized instance models
- eClassifiers
  - Contains a set of EClass and EDataTypes
- eSubPackages
  - May contain a set of nested EPackages

zeligsoft

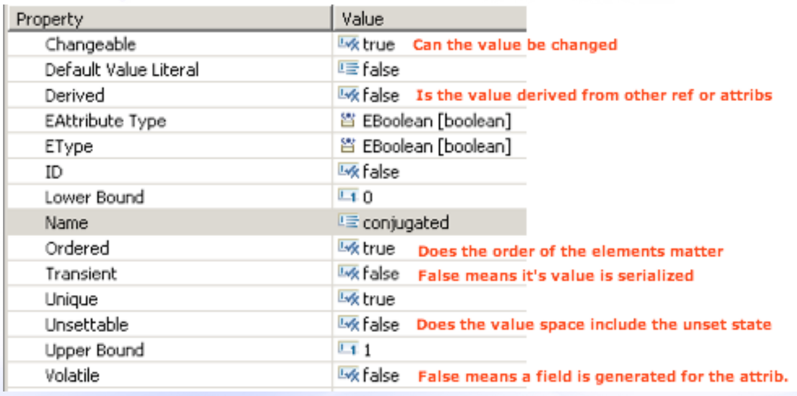
EClass



Property	Value	
Abstract	false	False means that instances be created
Default Value		
ESuper Types	NamedElement, PackageableElement	
Instance Type Name		The Java type that is modeled by this
Interface	false	If true no implementation is generated
Name	Actor	

zeligsoft

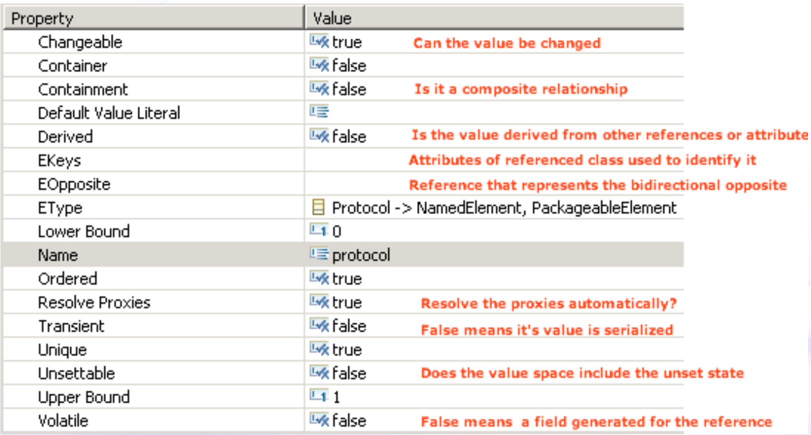
EAttribute



Property	Value	
Changeable	true	Can the value be changed
Default Value Literal	false	
Derived	false	Is the value derived from other ref or attribs
EAttribute Type	EBoolean [boolean]	
EType	EBoolean [boolean]	
ID	false	
Lower Bound	0	
Name	conjugated	
Ordered	true	Does the order of the elements matter
Transient	false	False means it's value is serialized
Unique	true	
Unsettable	false	Does the value space include the unset state
Upper Bound	1	
Volatile	false	False means a field is generated for the attrib.

zeligsoft

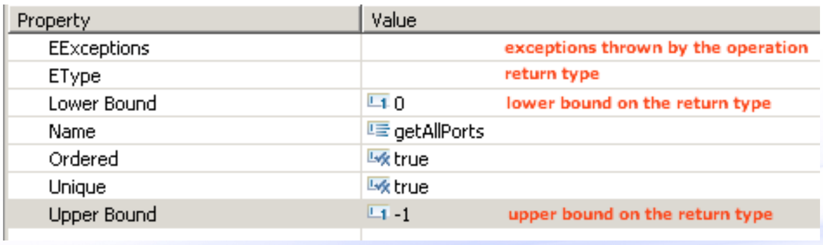
EReference



Property	Value	
Changeable	true	Can the value be changed
Container	false	
Containment	false	Is it a composite relationship
Default Value Literal		
Derived	false	Is the value derived from other references or attribute
EKeys		Attributes of referenced class used to identify it
EOpposite		Reference that represents the bidirectional opposite
EType	Protocol -> NamedElement, PackageableElement	
Lower Bound	0	
Name	protocol	
Ordered	true	
Resolve Proxies	true	Resolve the proxies automatically?
Transient	false	False means it's value is serialized
Unique	true	
Unsettable	false	Does the value space include the unset state
Upper Bound	1	
Volatile	false	False means a field generated for the reference

zeligsoft

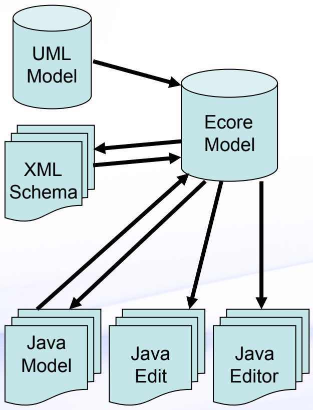
EOperation



Property	Value	
EExceptions		exceptions thrown by the operation
EType		return type
Lower Bound	0	lower bound on the return type
Name	getAllPorts	
Ordered	true	
Unique	true	
Upper Bound	-1	upper bound on the return type

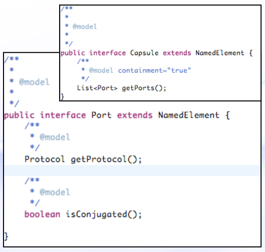
Creating an EMF Model

- EMF models can be created from
  - Java Interfaces
  - UML Class diagram
  - XML Schema
  - Ecore Diagram Editor
- With EMF if you have one of the above then you can generate the others
- Supports both
  - EMFizing a legacy data model
  - Greenfields development



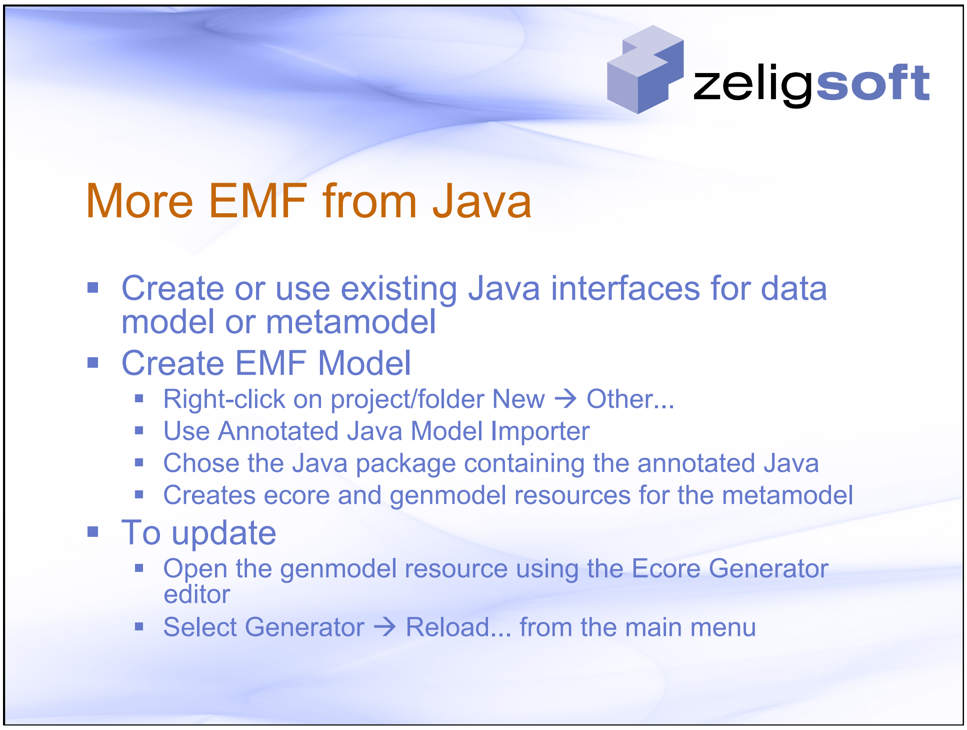
EMF from Java

- EMF models can be created from annotated Java
  - Typically for EMFizing legacy software
- @model annotation indicates parts of the code that correspond to model elements
  - interface - creates a class in EMF
  - method - creates operation in EMF
  - get method - creates attribute in EMF
- Supports reloading
  - i.e. can update the model by editing Java



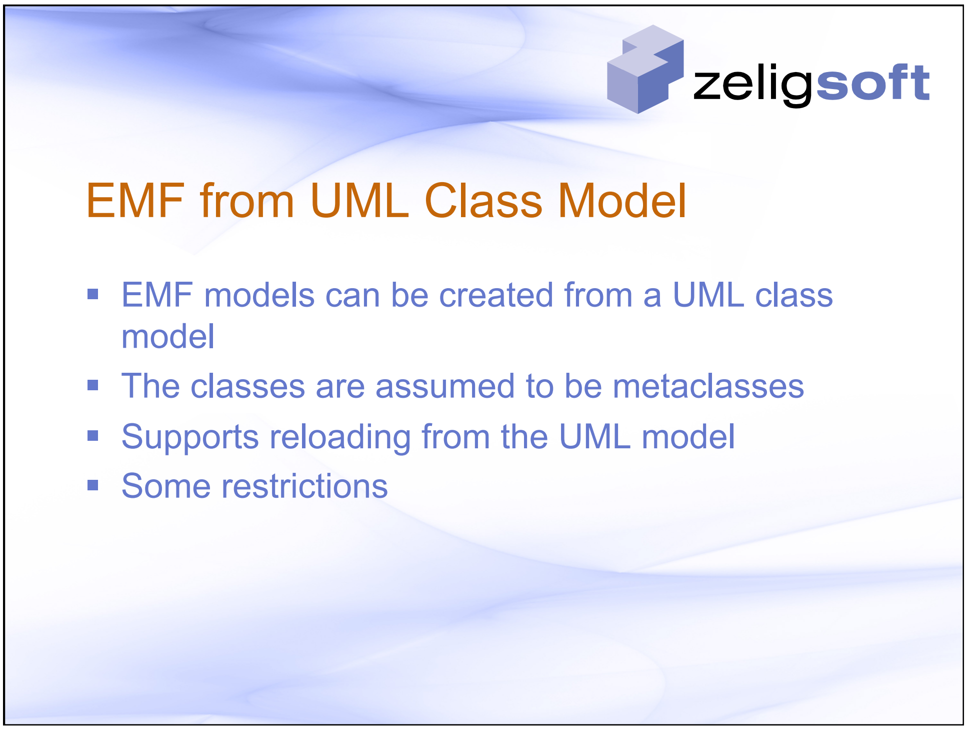
```
/**
 *
 * @model
 *
 */
public interface Capsule extends NamedElement {
    /**
     * @model containment="true"
     */
    List<Port> getPorts();
}
```

```
/**
 *
 * @model
 *
 */
public interface Port extends NamedElement {
    /**
     * @model
     */
    Protocol getProtocol();

    /**
     * @model
     */
    boolean isConjugated();

}
```



More EMF from Java

- Create or use existing Java interfaces for data model or metamodel
- Create EMF Model
  - Right-click on project/folder New → Other...
  - Use Annotated Java Model Importer
  - Chose the Java package containing the annotated Java
  - Creates ecore and genmodel resources for the metamodel
- To update
  - Open the genmodel resource using the Ecore Generator editor
  - Select Generator → Reload... from the main menu



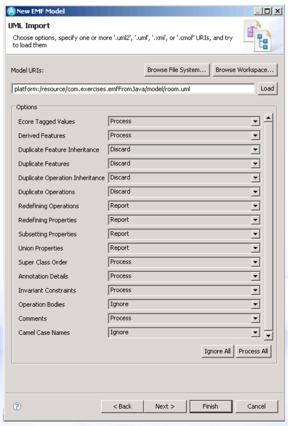
EMF from UML Class Model

- EMF models can be created from a UML class model
- The classes are assumed to be metaclasses
- Supports reloading from the UML model
- Some restrictions

zeligsoft

More EMF from UML Class Model

- Create UML model
- Create EMF model
  - New → Other...
  - Use UML Model Importer
  - Chose the UML model
  - Configure import
  - Creates ecore and genmodel resources for the metamodel
- To update
  - Open the genmodel resource using the Ecore Generator Editor
  - Select Generator → Reload...



UML Import

Choose options, specify one or more '.uml2', '.uml', '.xmi', or '.cmof' URIs, and try to load them

Model URIs:

platform:/resource/com.exercises.emfFromJava/model/room.uml

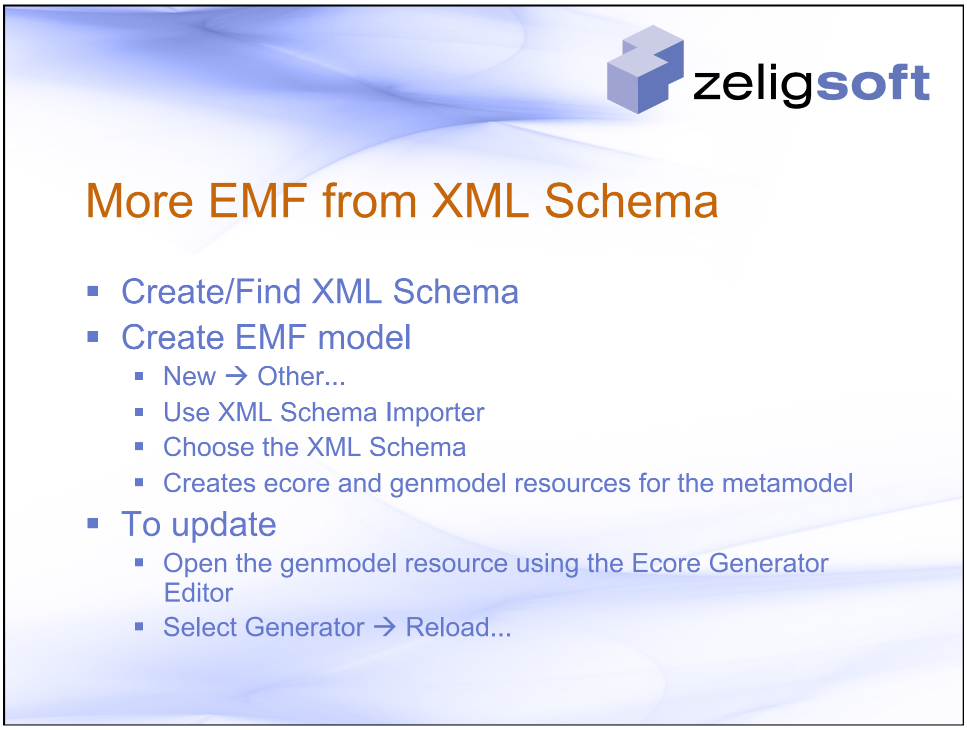
Options

Ecore Tagged Values	Process
Derived Features	Process
Duplicate Feature Inheritance	Discard
Duplicate Features	Discard
Duplicate Operation Inheritance	Discard
Duplicate Operations	Discard
Redefining Operations	Report
Redefining Properties	Report
Subsetting Properties	Report
Union Properties	Report
Super Class Order	Process
Annotation Details	Process
Invariant Constraints	Process
Operation Bodies	Ignore
Comments	Process
Camel Case Names	Ignore

zeligsoft

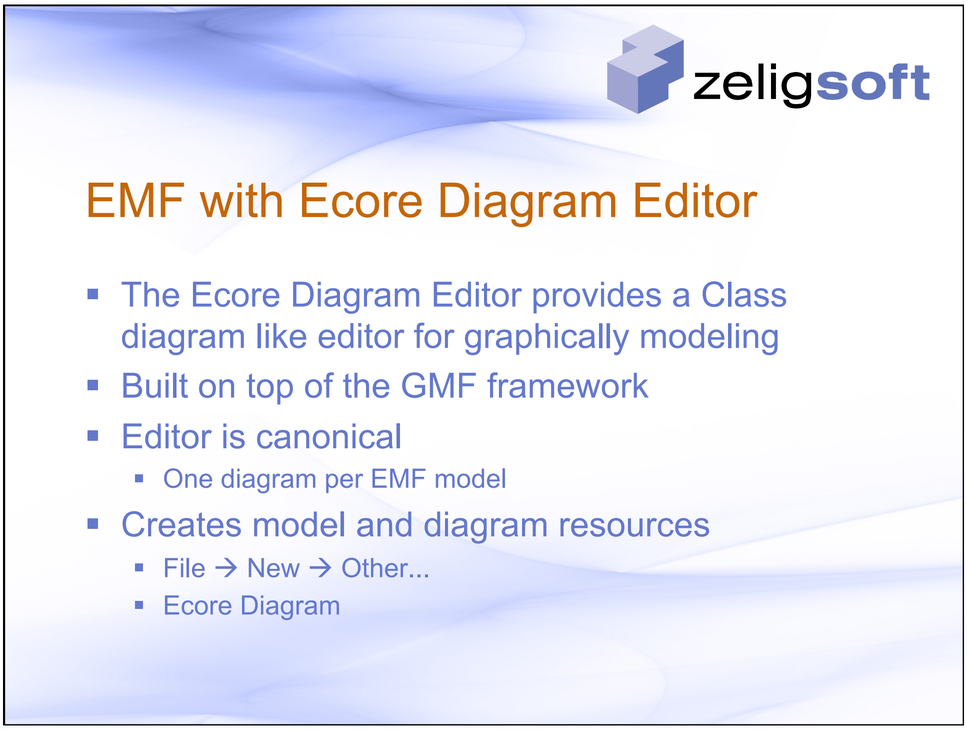
EMF from XML Schema

- Many industry standards produce XML schemas to define their data format
- EMF model can be created from an XML schema
  - Model instances are schema compliant
- Supports reloading from XML schema
- Schema can be regenerated from EMF model



More EMF from XML Schema

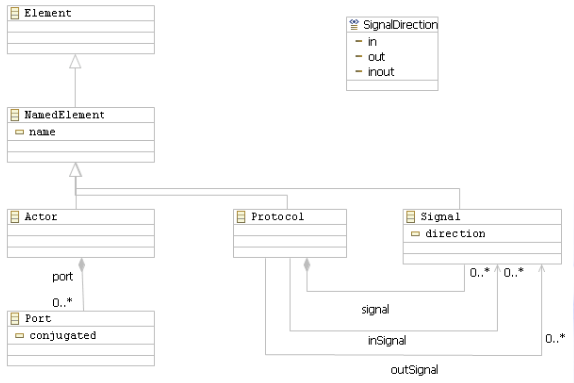
- Create/Find XML Schema
- Create EMF model
  - New → Other...
  - Use XML Schema Importer
  - Choose the XML Schema
  - Creates ecore and genmodel resources for the metamodel
- To update
  - Open the genmodel resource using the Ecore Generator Editor
  - Select Generator → Reload...



EMF with Ecore Diagram Editor

- The Ecore Diagram Editor provides a Class diagram like editor for graphically modeling
- Built on top of the GMF framework
- Editor is canonical
  - One diagram per EMF model
- Creates model and diagram resources
  - File → New → Other...
  - Ecore Diagram

More EMF with Ecore Diagram Editor

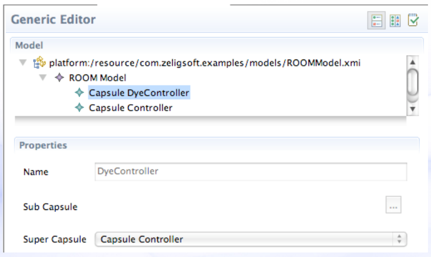


How do I Test my Model?

- Manually review the model and ensure that it looks like it will work!
  - Error prone
- Build the generation model now generate the default model editor
  - Time consuming when frequently changing
- Use the dynamic instance creation capabilities
  - Part of the Sample Ecore Model Editor
- Use the generated test suite from EMF
  - Generates a JUnit test suite, to test the user supplied code in operations and derived features

Dynamically Creating Models

- While developing a model it can be tested by creating dynamic instance
  - Does not require anything to be generated
  - Does not require a runtime workbench
  - Uses reflective capabilities of EMF
- In the Ecore editor
  - Right-click on the EClass
  - Create Dynamic Instance...
- Stored as XMI



Generic Editor

Model

- platform:/resource/com.zeligsoft.examples/models/ROOMModel.xmi
  - ROOM Model
    - Capsule DyeController
    - Capsule Controller

Properties

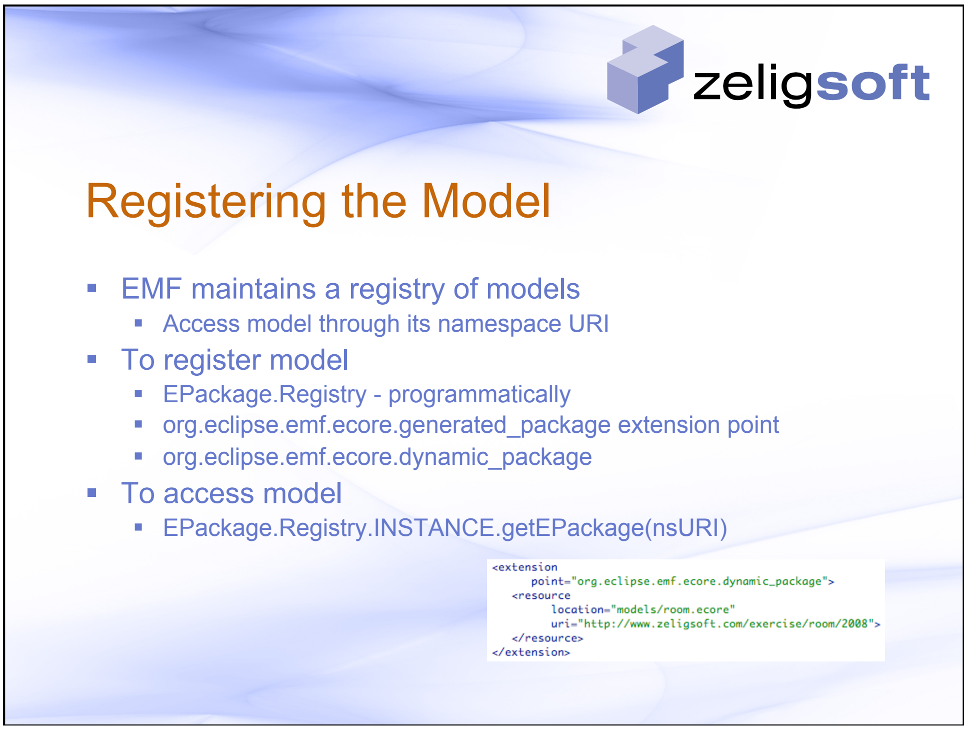
Name: DyeController

Sub Capsule: ...

Super Capsule: Capsule Controller

Do this with Code?

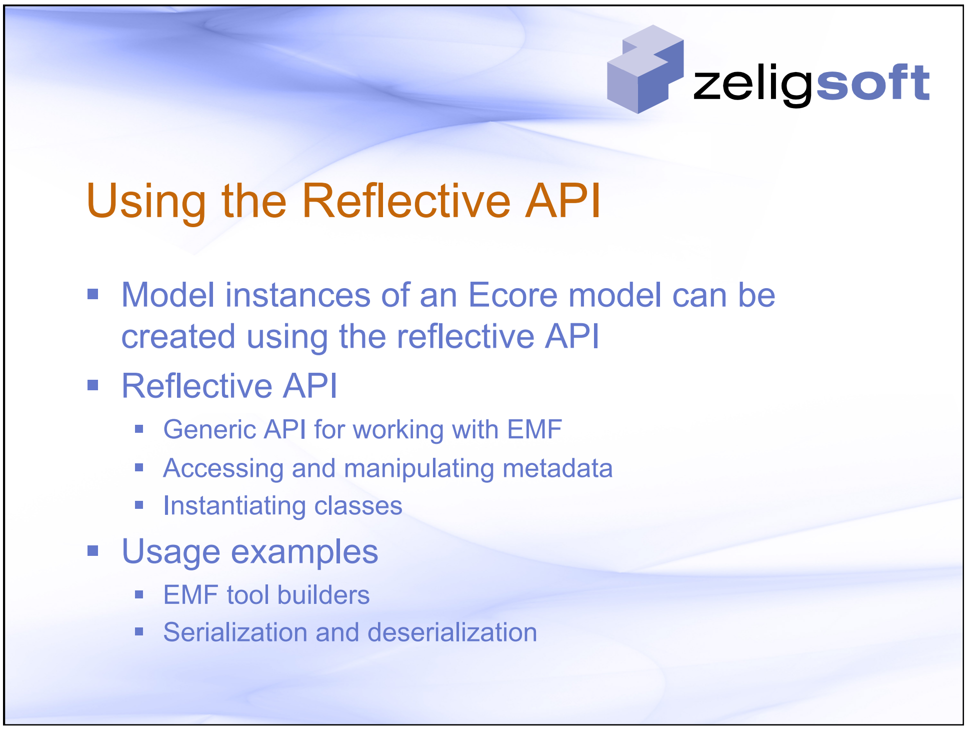
- Register the ecore model as a dynamic package
- Use the reflective API of EMF
  - Can create instances and persist them



Registering the Model

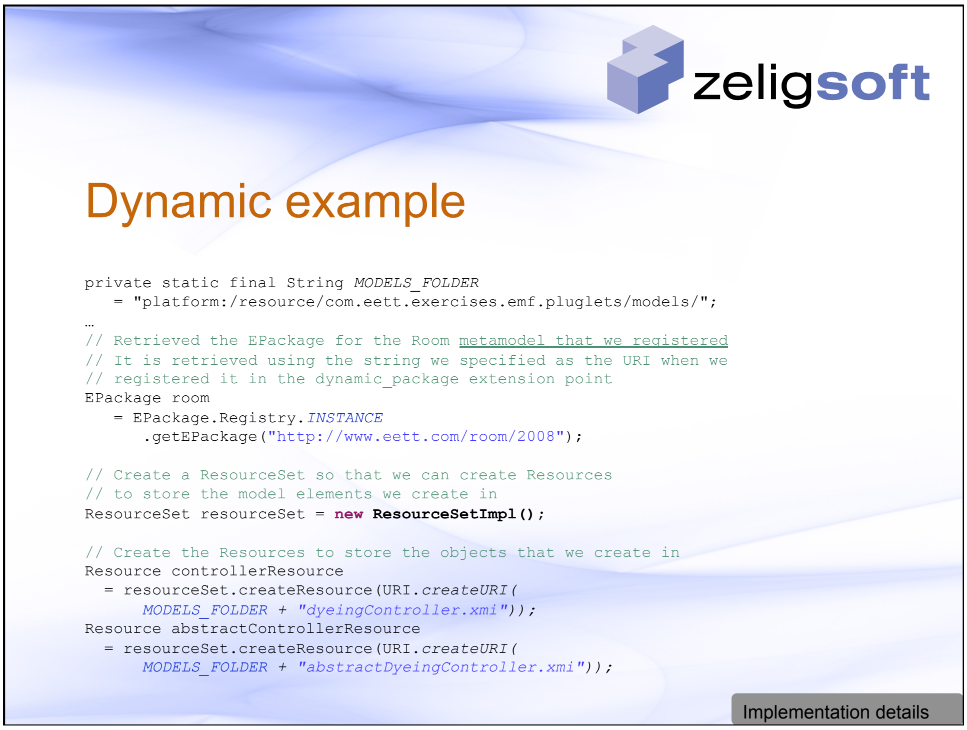
- EMF maintains a registry of models
  - Access model through its namespace URI
- To register model
  - EPackage.Registry - programmatically
  - org.eclipse.emf.ecore.generated_package extension point
  - org.eclipse.emf.ecore.dynamic_package
- To access model
  - EPackage.Registry.INSTANCE.getEPackage(nsURI)

```
<extension
      point="org.eclipse.emf.ecore.dynamic_package">
   <resource
         location="models/room.ecore"
         uri="http://www.zeligsoft.com/exercise/room/2008">
   </resource>
</extension>
```



Using the Reflective API

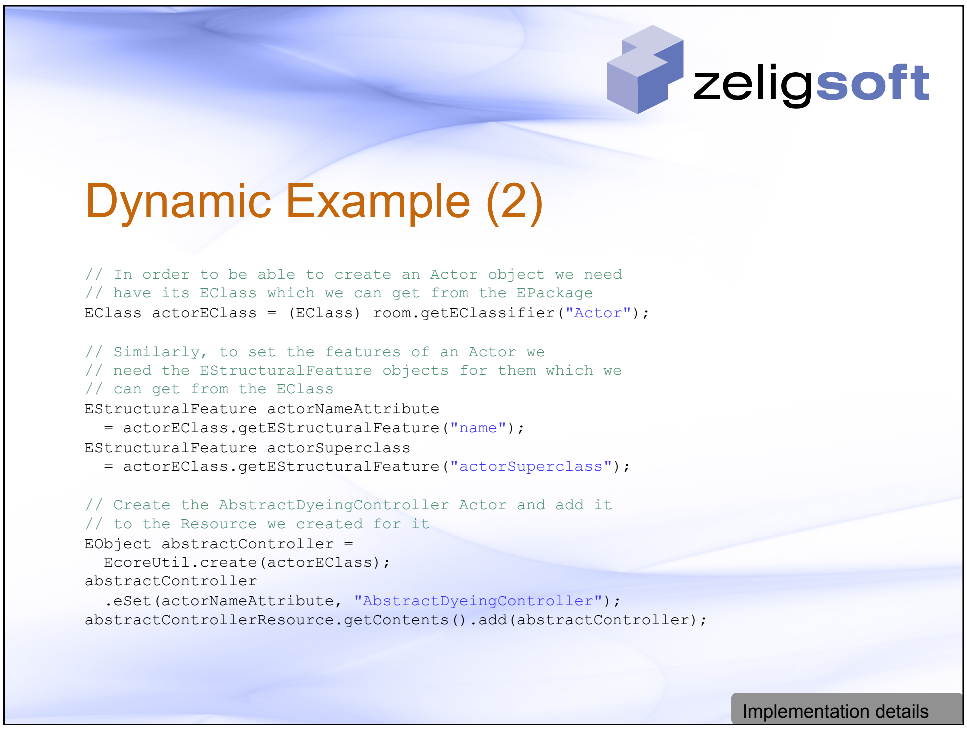
- Model instances of an Ecore model can be created using the reflective API
- Reflective API
  - Generic API for working with EMF
  - Accessing and manipulating metadata
  - Instantiating classes
- Usage examples
  - EMF tool builders
  - Serialization and deserialization



Dynamic example

```
private static final String MODELS_FOLDER
   = "platform:/resource/com.eett.exercises.emf.pluglets/models/";
…
// Retrieved the EPackage for the Room metamodel that we registered
// It is retrieved using the string we specified as the URI when we
// registered it in the dynamic_package extension point
EPackage room
   = EPackage.Registry.INSTANCE
      .getEPackage("http://www.eett.com/room/2008");

// Create a ResourceSet so that we can create Resources
// to store the model elements we create in
ResourceSet resourceSet = new ResourceSetImpl();

// Create the Resources to store the objects that we create in
Resource controllerResource
  = resourceSet.createResource(URI.createURI(
      MODELS_FOLDER + "dyeingController.xmi"));
Resource abstractControllerResource
  = resourceSet.createResource(URI.createURI(
      MODELS_FOLDER + "abstractDyeingController.xmi"));
```

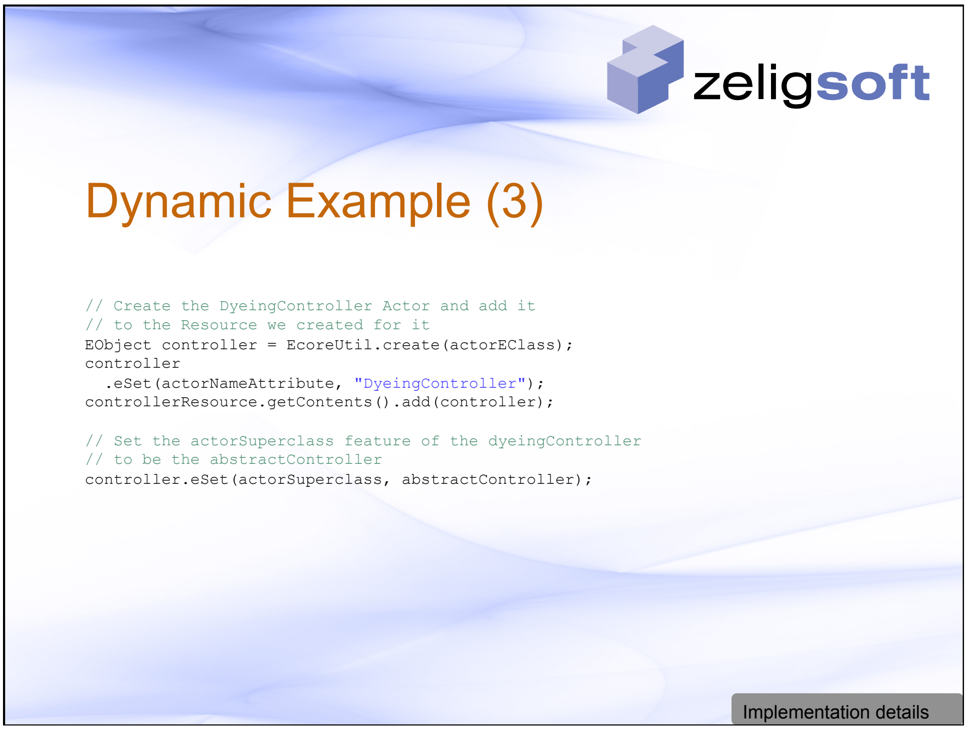
Implementation details



Dynamic Example (2)

```
// In order to be able to create an Actor object we need
// have its EClass which we can get from the EPackage
EClass actorEClass = (EClass) room.getEClassifier("Actor");

// Similarly, to set the features of an Actor we
// need the EStructuralFeature objects for them which we
// can get from the EClass
EStructuralFeature actorNameAttribute
  = actorEClass.getEStructuralFeature("name");
EStructuralFeature actorSuperclass
  = actorEClass.getEStructuralFeature("actorSuperclass");

// Create the AbstractDyeingController Actor and add it
// to the Resource we created for it
EObject abstractController =
  EcoreUtil.create(actorEClass);
abstractController
  .eSet(actorNameAttribute, "AbstractDyeingController");
abstractControllerResource.getContents().add(abstractController);
```

Implementation details

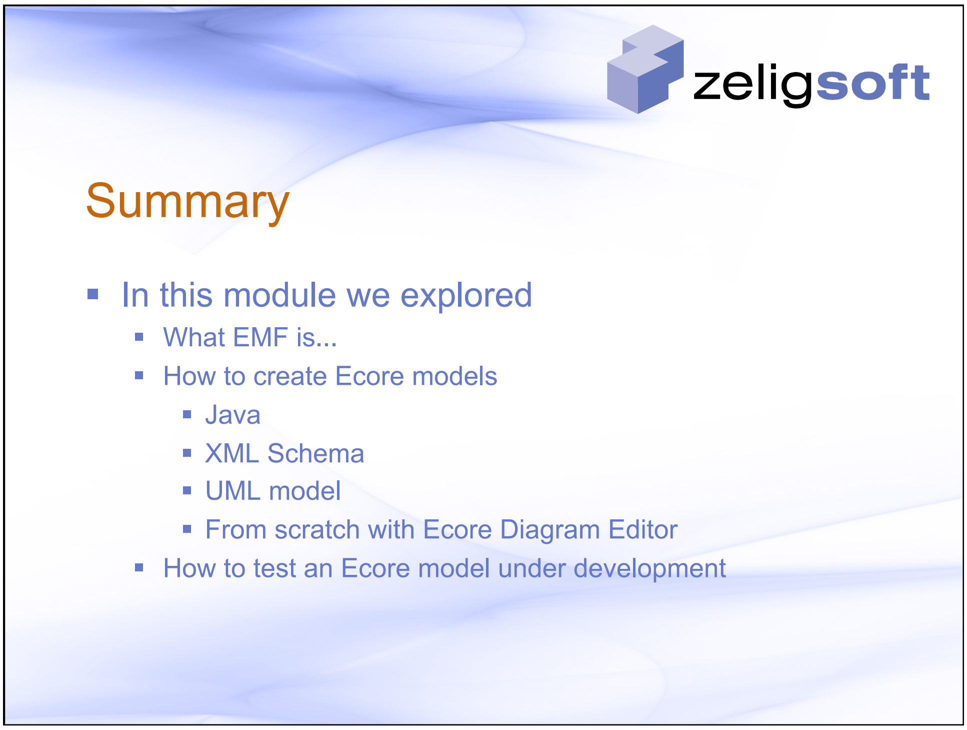


zeligsoft

Dynamic Example (3)

```
// Create the DyeingController Actor and add it
// to the Resource we created for it
EObject controller = EcoreUtil.create(actorEClass);
controller
  .eSet(actorNameAttribute, "DyeingController");
controllerResource.getContents().add(controller);

// Set the actorSuperclass feature of the dyeingController
// to be the abstractController
controller.eSet(actorSuperclass, abstractController);
```

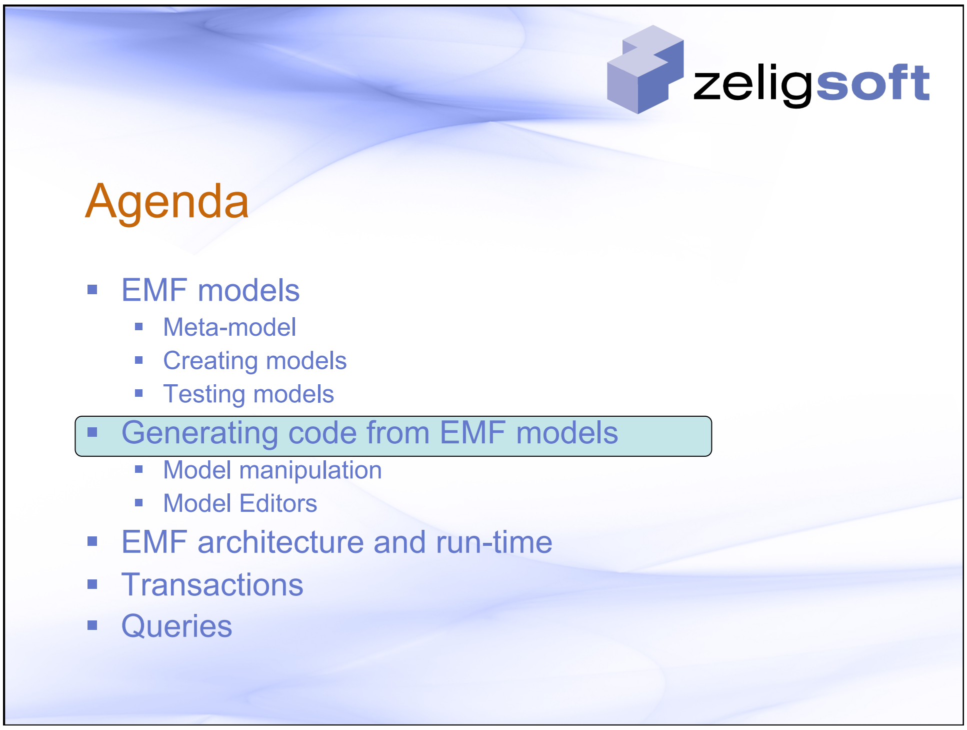
Implementation details



zeligsoft

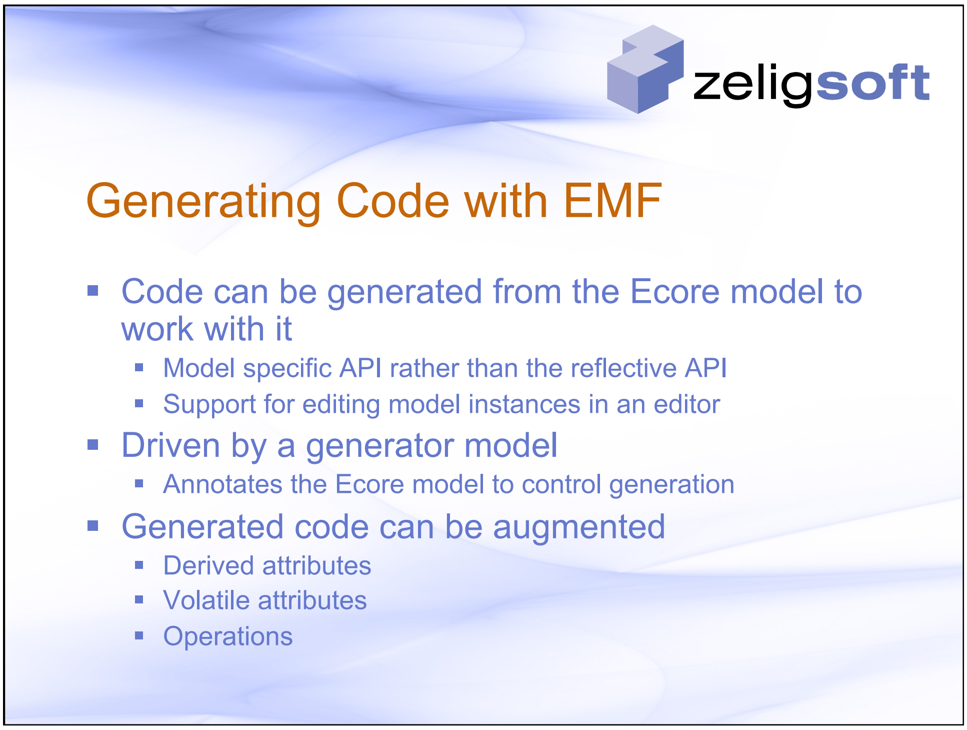
Summary

- In this module we explored
  - What EMF is...
  - How to create Ecore models
    - Java
    - XML Schema
    - UML model
    - From scratch with Ecore Diagram Editor
  - How to test an Ecore model under development



Agenda

- EMF models
  - Meta-model
  - Creating models
  - Testing models
- Generating code from EMF models
  - Model manipulation
  - Model Editors
- EMF architecture and run-time
- Transactions
- Queries



Generating Code with EMF

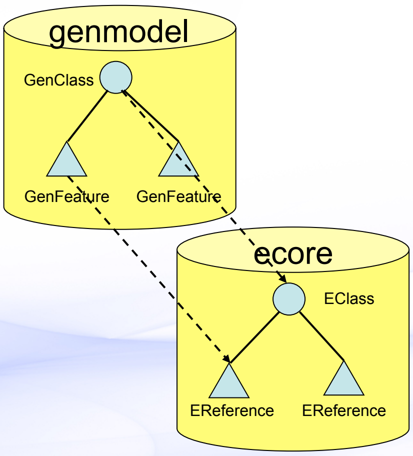
- Code can be generated from the Ecore model to work with it
  - Model specific API rather than the reflective API
  - Support for editing model instances in an editor
- Driven by a generator model
  - Annotates the Ecore model to control generation
- Generated code can be augmented
  - Derived attributes
  - Volatile attributes
  - Operations

More Generating Code with EMF

- Full support for regeneration and merge
- Code can be customized
  - Configuration parameters
  - Custom templates
- Code can be generated from
  - Workbench
  - Ant script
  - Command line
  - *NOTE THAT BOTH ANT AND COMMAND LINE STILL REQUIRE ECLIPSE*

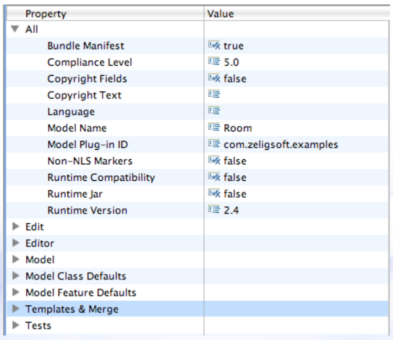
The genmodel

- Wraps the Ecore model and automatically kept in sync
  - Only true for Ecore
  - Others like Java require explicit reload
- Decorates the Ecore model
  - Target location for generated code
  - Prefix for some of the generated classes



More genmodel

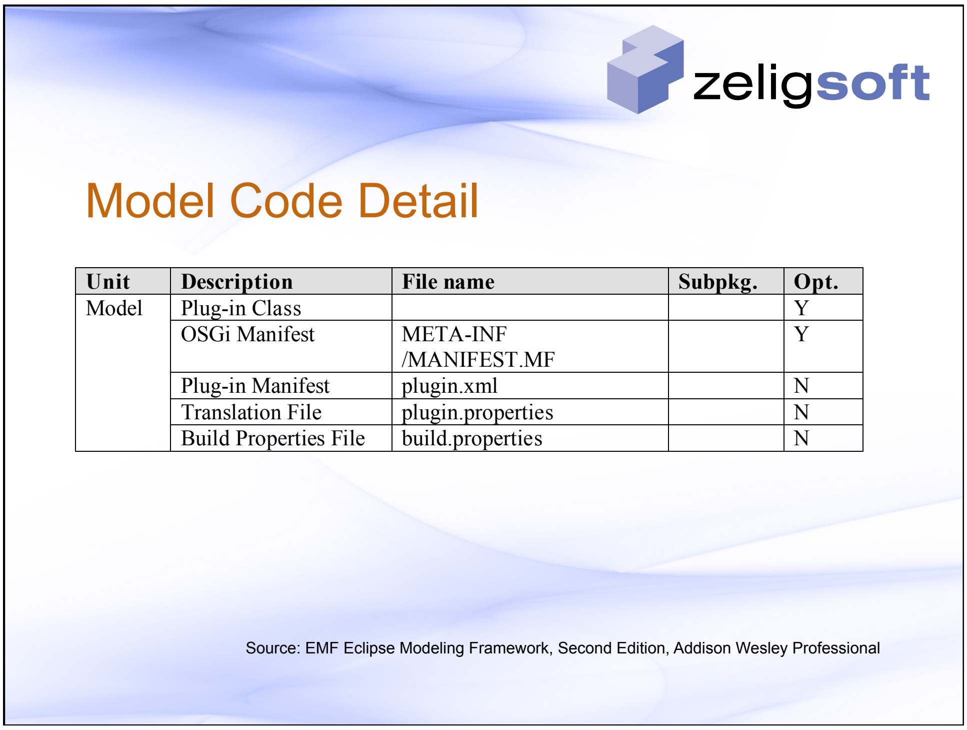
- Global generator configuration
- To generate
  - Context menu in the editor
  - Main menu in the editor
- Generates for
  - GenModel, GenPackage, GenClass and GenEnum
- What is generated
  - Model
  - Edit
  - Editor
  - Tests



Property	Value
▼ All	
Bundle Manifest	true
Compliance Level	5.0
Copyright Fields	false
Copyright Text	
Language	
Model Name	Room
Model Plug-in ID	com.zeligsoft.examples
Non-NLS Markers	false
Runtime Compatibility	false
Runtime Jar	false
Runtime Version	2.4
▶ Edit	
▶ Editor	
▶ Model	
▶ Model Class Defaults	
▶ Model Feature Defaults	
▶ Templates & Merge	
▶ Tests	

Model Code

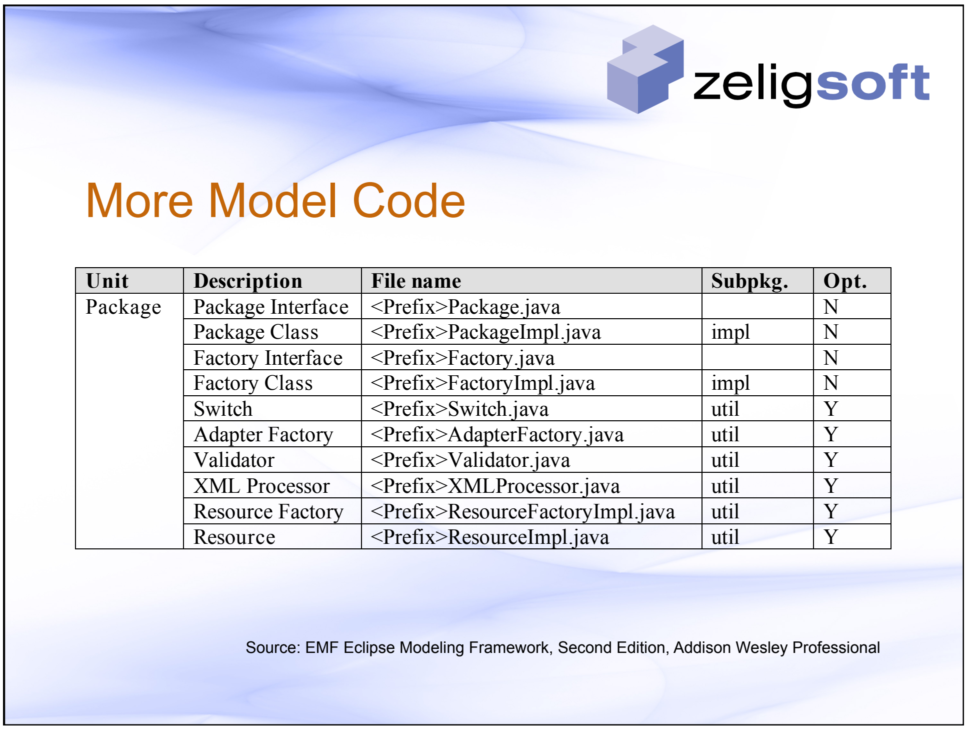
- Java code for working with and manipulating model instances
- Interfaces, classes and enumerations
- Metadata
  - Package
- API for creating instances of classes
  - Factory
- Persistence
  - Resource and XML processor
- Utilities
  - E.g. Switch for visiting model elements



Model Code Detail

Unit	Description	File name	Subpkg.	Opt.
Model	Plug-in Class			Y
	OSGi Manifest	META-INF /MANIFEST.MF		Y
	Plug-in Manifest	plugin.xml		N
	Translation File	plugin.properties		N
	Build Properties File	build.properties		N

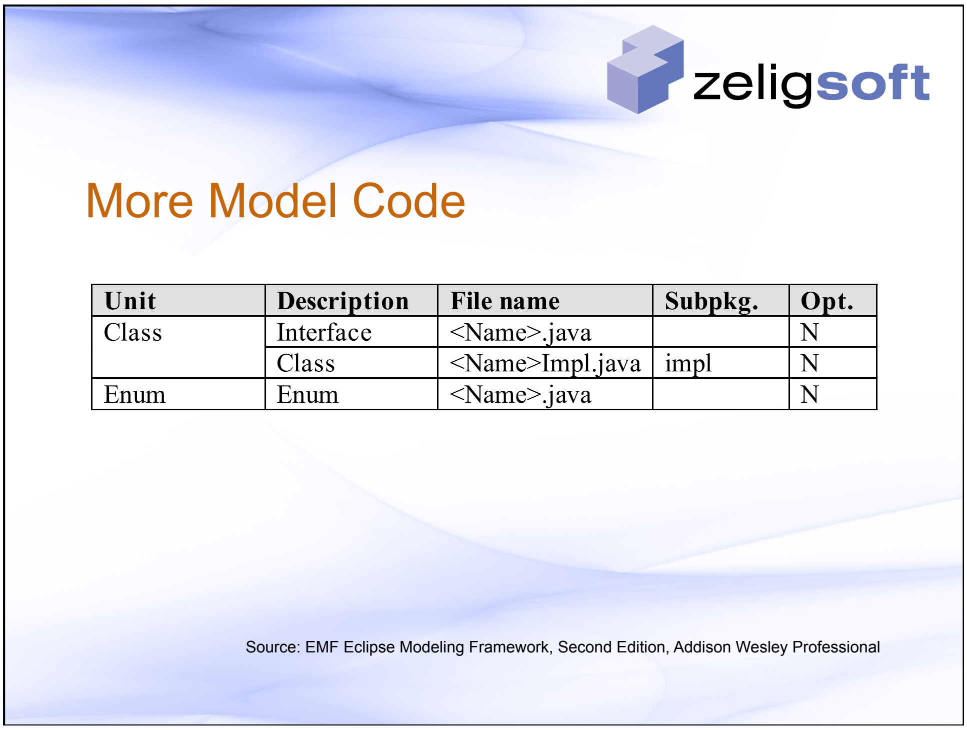
Source: EMF Eclipse Modeling Framework, Second Edition, Addison Wesley Professional



More Model Code

Unit	Description	File name	Subpkg.	Opt.
Package	Package Interface	<Prefix>Package.java		N
	Package Class	<Prefix>PackageImpl.java	impl	N
	Factory Interface	<Prefix>Factory.java		N
	Factory Class	<Prefix>FactoryImpl.java	impl	N
	Switch	<Prefix>Switch.java	util	Y
	Adapter Factory	<Prefix>AdapterFactory.java	util	Y
	Validator	<Prefix>Validator.java	util	Y
	XML Processor	<Prefix>XMLProcessor.java	util	Y
	Resource Factory	<Prefix>ResourceFactoryImpl.java	util	Y
	Resource	<Prefix>ResourceImpl.java	util	Y

Source: EMF Eclipse Modeling Framework, Second Edition, Addison Wesley Professional

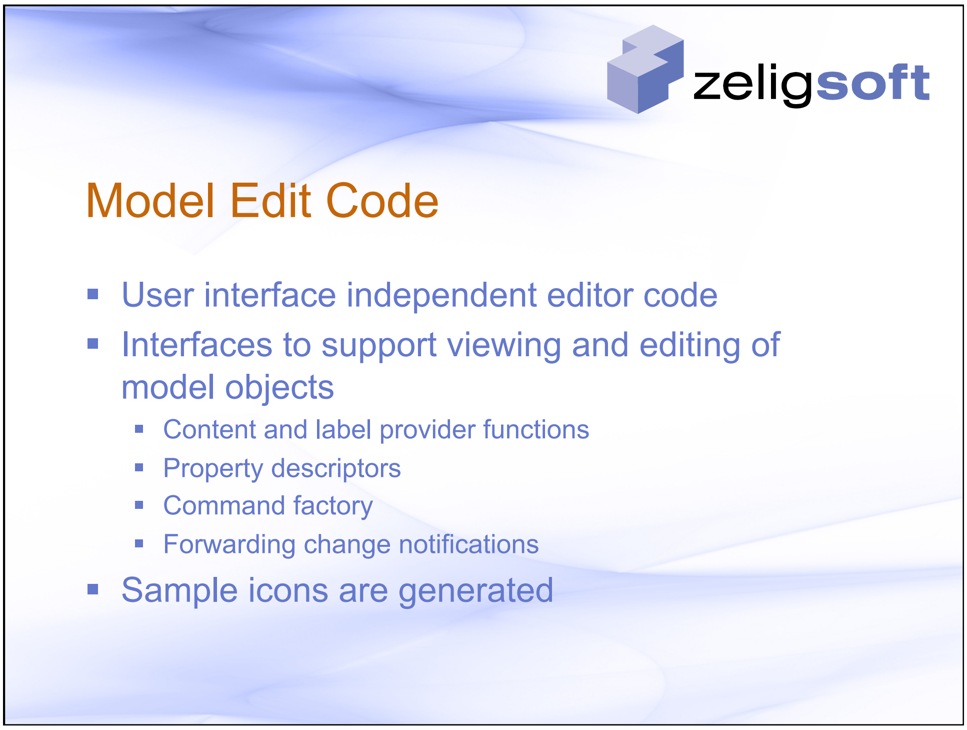


zeligsoft

More Model Code

Unit	Description	File name	Subpkg.	Opt.
Class	Interface	<Name>.java		N
	Class	<Name>Impl.java	impl	N
Enum	Enum	<Name>.java		N

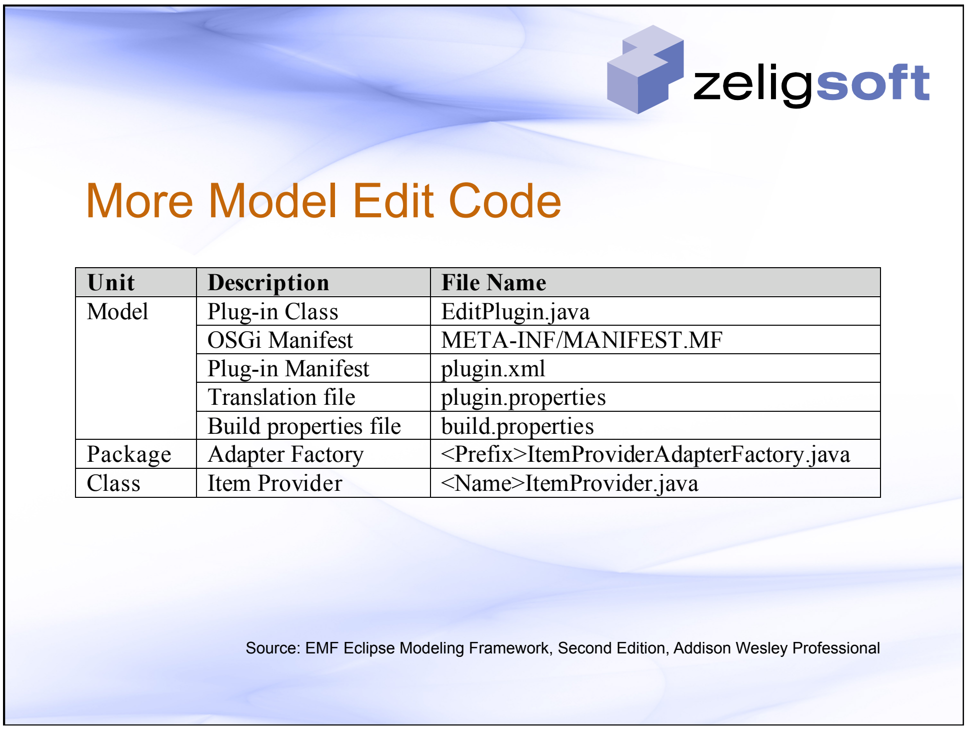
Source: EMF Eclipse Modeling Framework, Second Edition, Addison Wesley Professional



zeligsoft

Model Edit Code

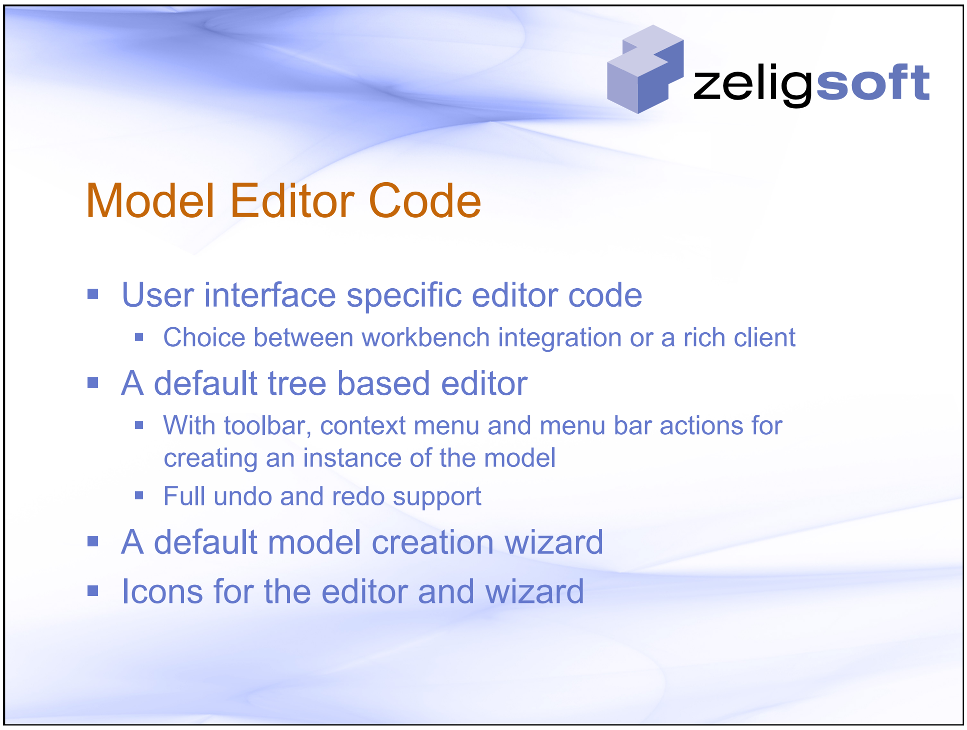
- User interface independent editor code
- Interfaces to support viewing and editing of model objects
  - Content and label provider functions
  - Property descriptors
  - Command factory
  - Forwarding change notifications
- Sample icons are generated



More Model Edit Code

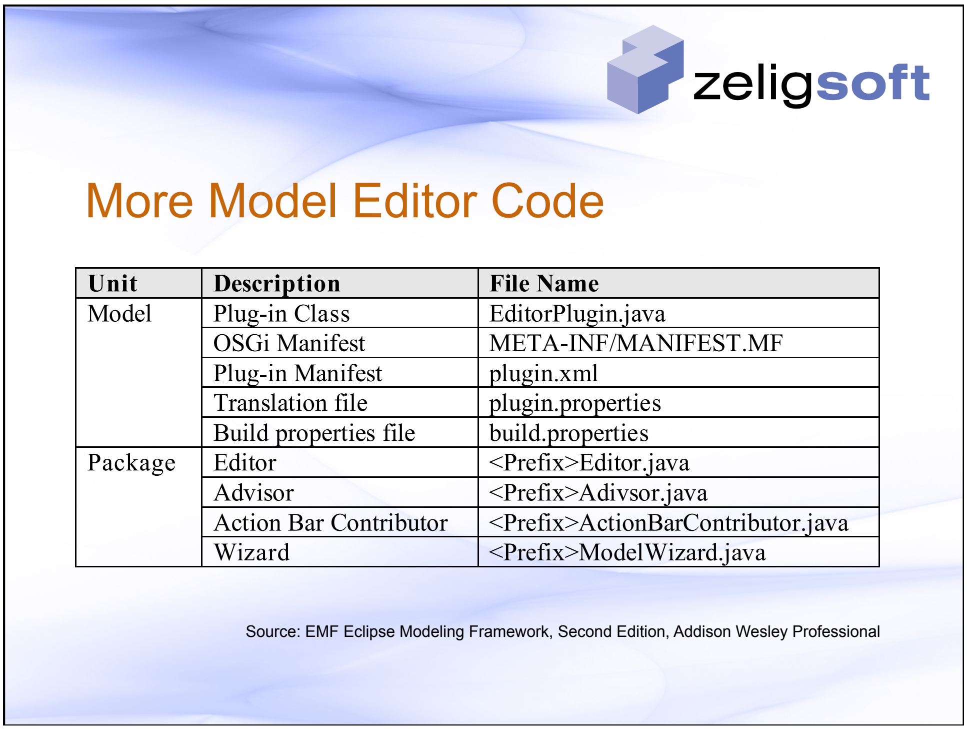
Unit	Description	File Name
Model	Plug-in Class	EditPlugin.java
	OSGi Manifest	META-INF/MANIFEST.MF
	Plug-in Manifest	plugin.xml
	Translation file	plugin.properties
	Build properties file	build.properties
Package	Adapter Factory	<Prefix>ItemProviderAdapterFactory.java
Class	Item Provider	<Name>ItemProvider.java

Source: EMF Eclipse Modeling Framework, Second Edition, Addison Wesley Professional



Model Editor Code

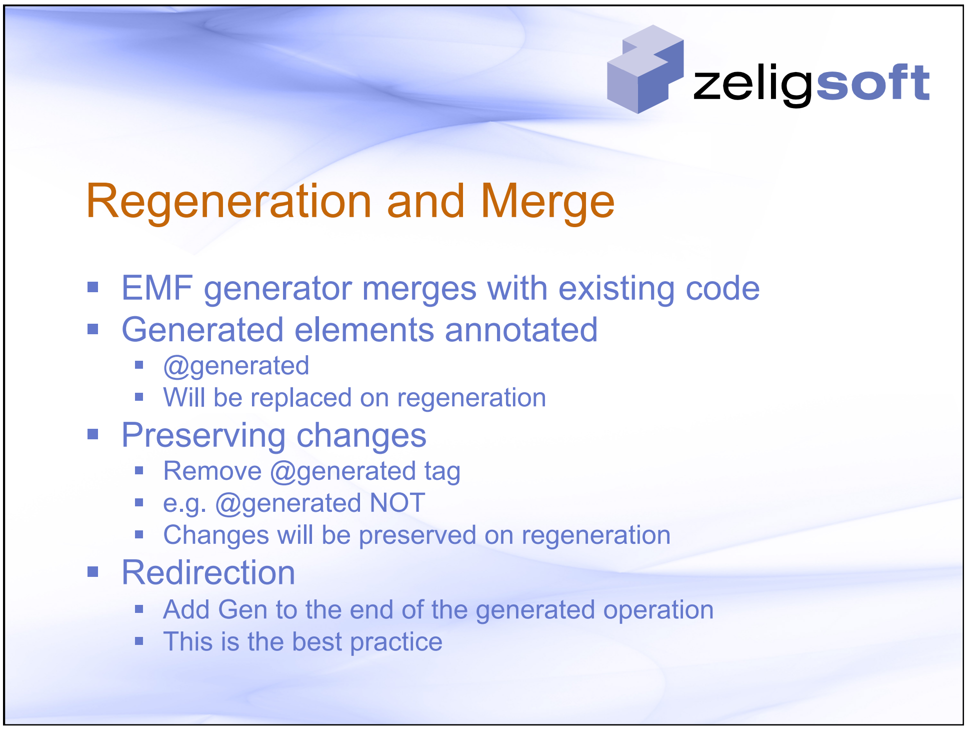
- User interface specific editor code
  - Choice between workbench integration or a rich client
- A default tree based editor
  - With toolbar, context menu and menu bar actions for creating an instance of the model
  - Full undo and redo support
- A default model creation wizard
- Icons for the editor and wizard



More Model Editor Code

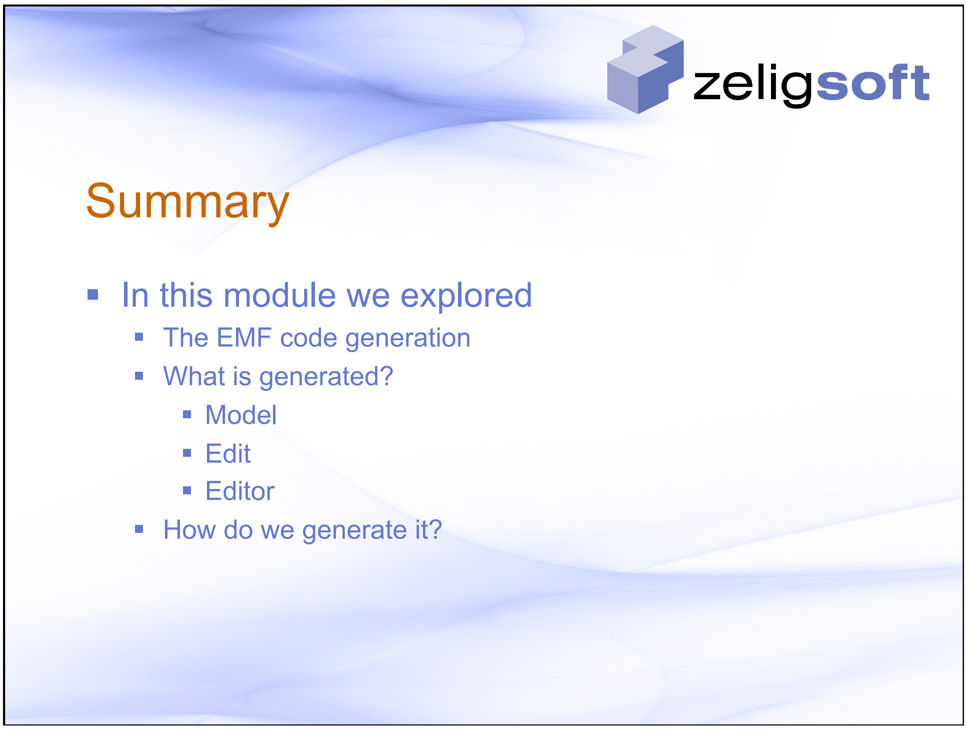
Unit	Description	File Name
Model	Plug-in Class	EditorPlugin.java
	OSGi Manifest	META-INF/MANIFEST.MF
	Plug-in Manifest	plugin.xml
	Translation file	plugin.properties
	Build properties file	build.properties
Package	Editor	<Prefix>Editor.java
	Advisor	<Prefix>Adivsor.java
	Action Bar Contributor	<Prefix>ActionBarContributor.java
	Wizard	<Prefix>ModelWizard.java

Source: EMF Eclipse Modeling Framework, Second Edition, Addison Wesley Professional



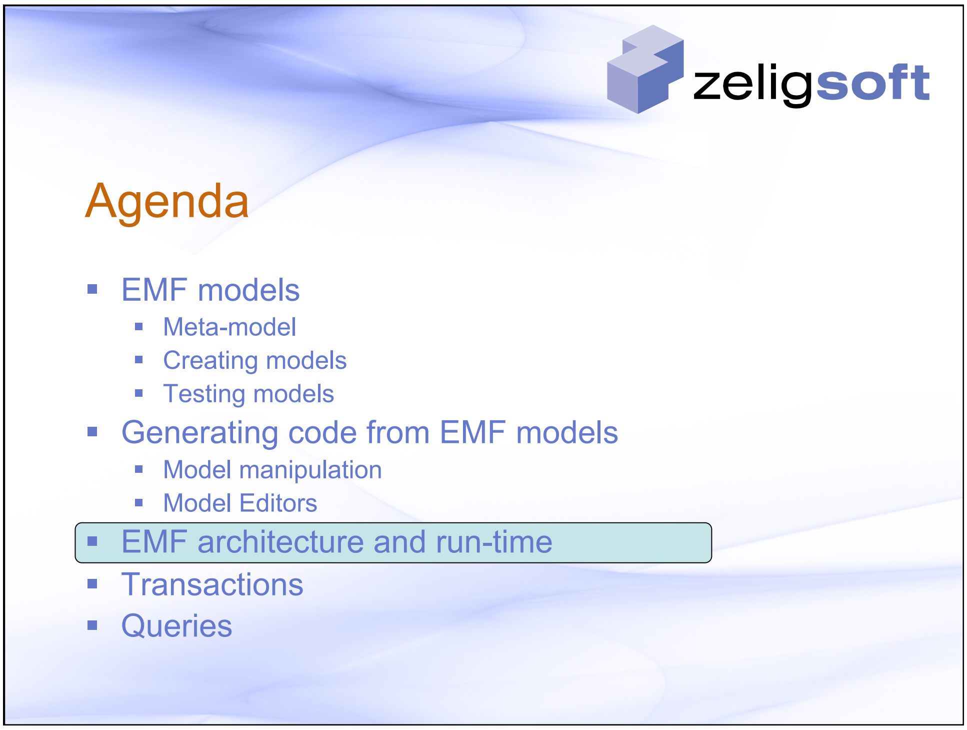
Regeneration and Merge

- EMF generator merges with existing code
- Generated elements annotated
  - @generated
  - Will be replaced on regeneration
- Preserving changes
  - Remove @generated tag
  - e.g. @generated NOT
  - Changes will be preserved on regeneration
- Redirection
  - Add Gen to the end of the generated operation
  - This is the best practice



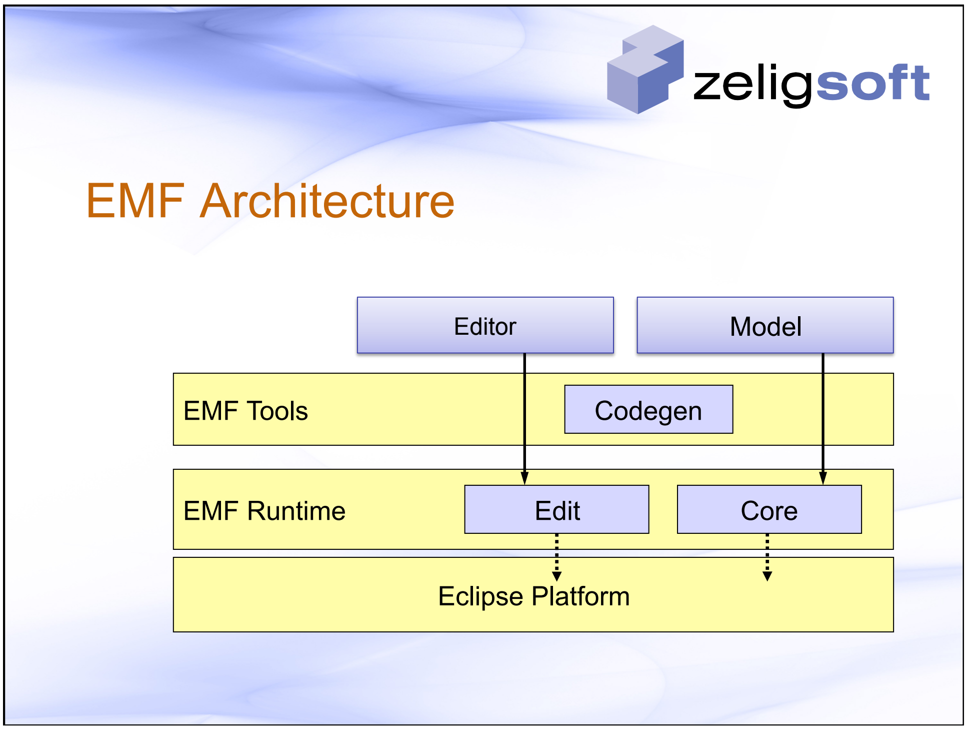
Summary

- In this module we explored
  - The EMF code generation
  - What is generated?
    - Model
    - Edit
    - Editor
  - How do we generate it?



Agenda

- EMF models
  - Meta-model
  - Creating models
  - Testing models
- Generating code from EMF models
  - Model manipulation
  - Model Editors
- EMF architecture and run-time
- Transactions
- Queries



EMF Architecture

Editor

Model

EMF Tools

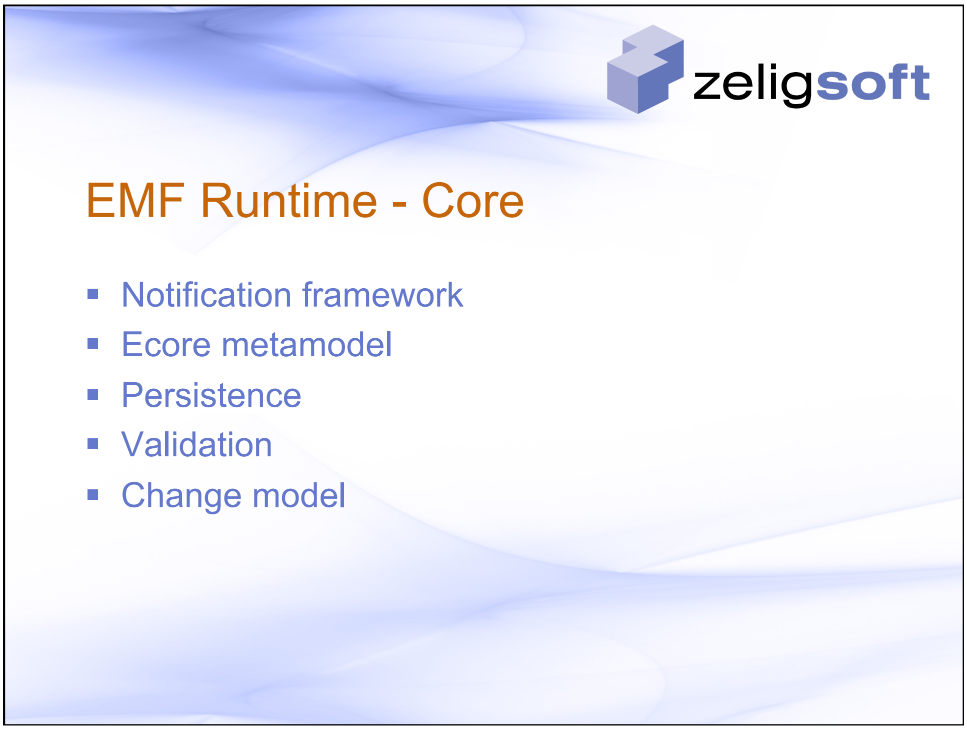
Codegen

EMF Runtime

Edit

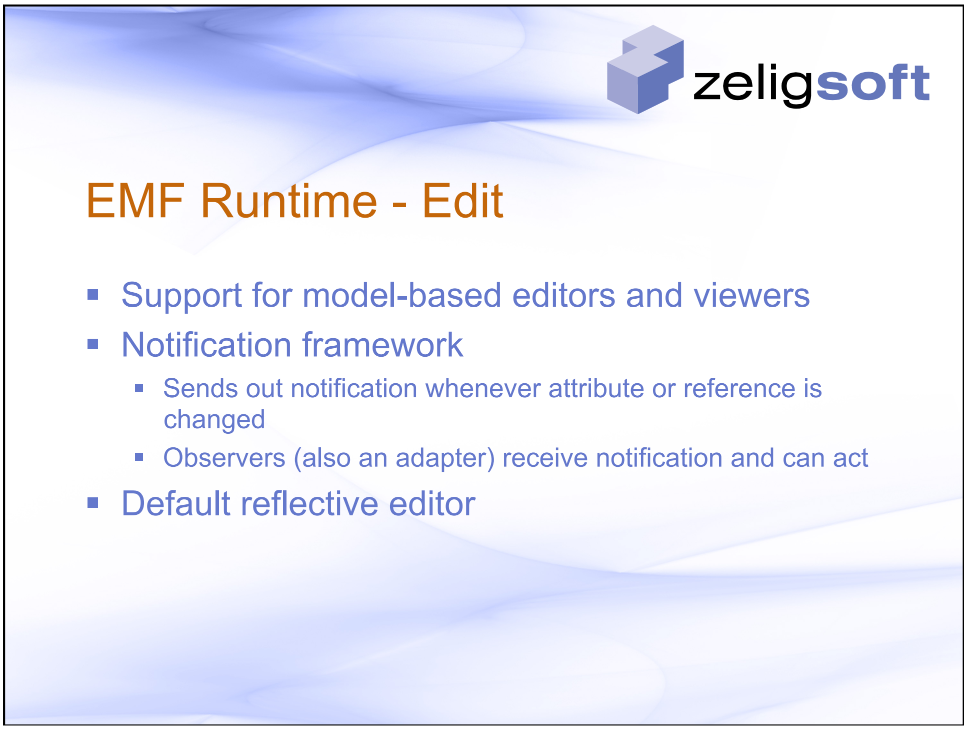
Core

Eclipse Platform



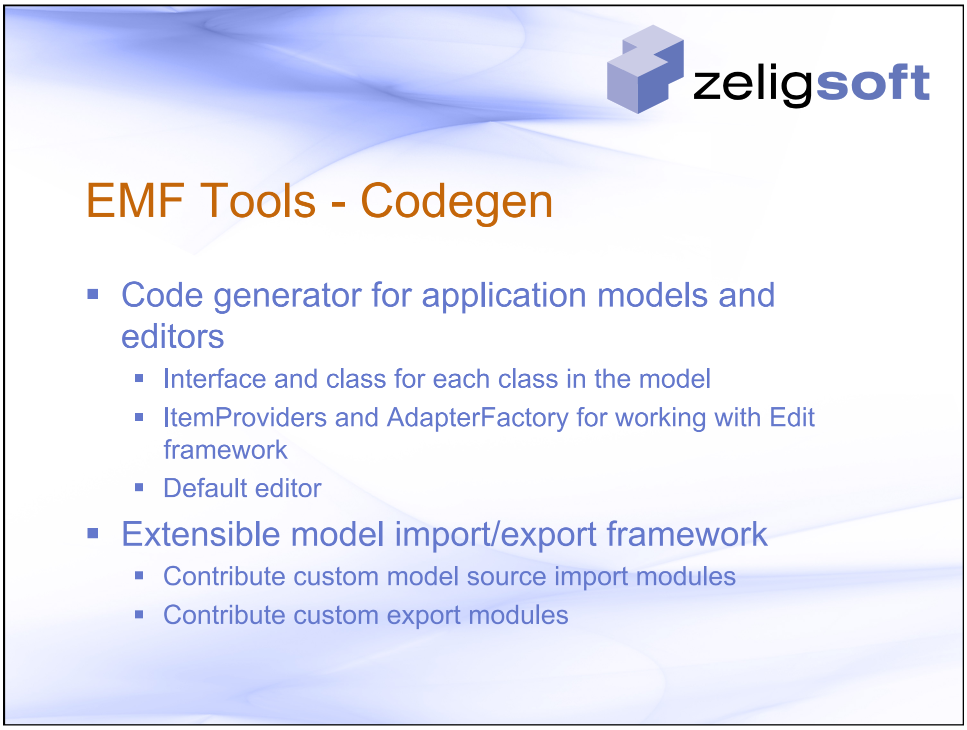
EMF Runtime - Core

- Notification framework
- Ecore metamodel
- Persistence
- Validation
- Change model



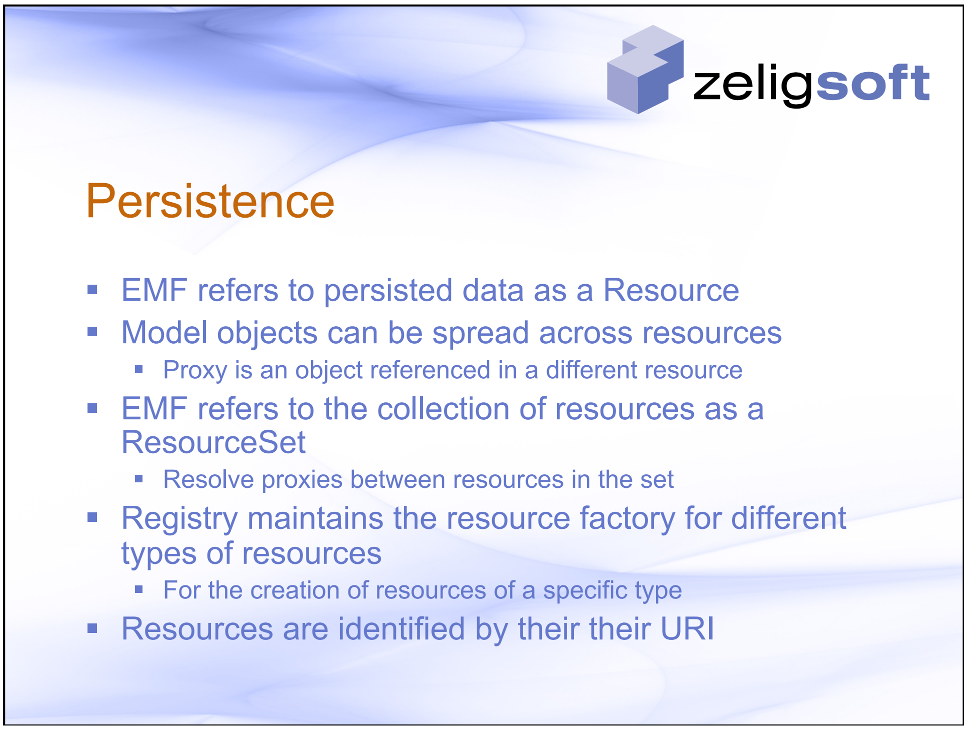
EMF Runtime - Edit

- Support for model-based editors and viewers
- Notification framework
  - Sends out notification whenever attribute or reference is changed
  - Observers (also an adapter) receive notification and can act
- Default reflective editor



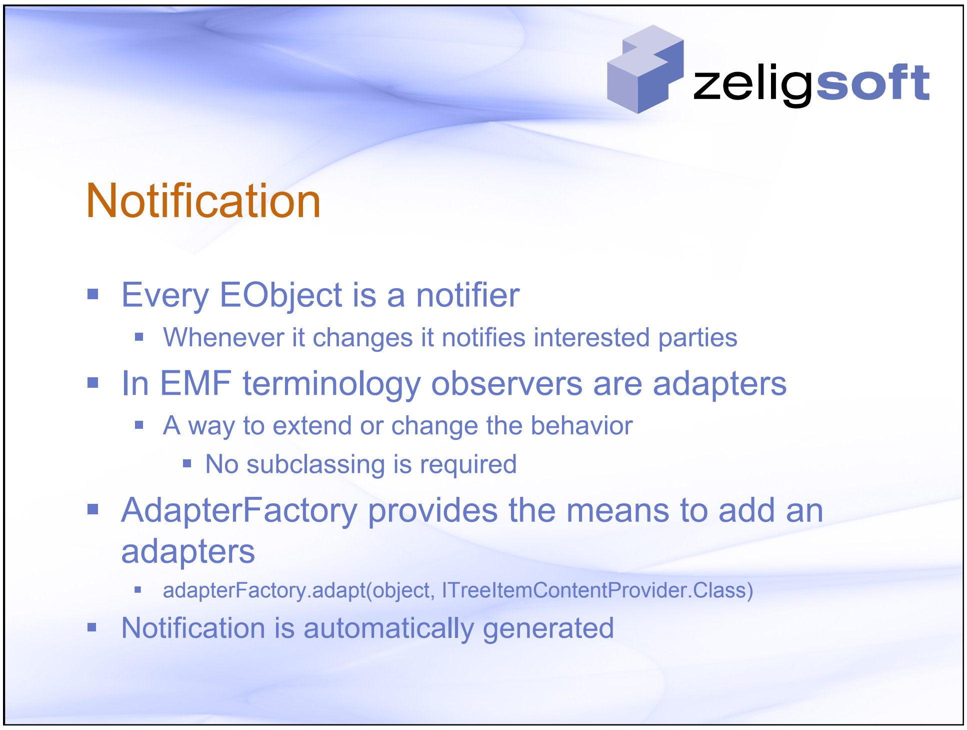
EMF Tools - Codegen

- Code generator for application models and editors
  - Interface and class for each class in the model
  - ItemProviders and AdapterFactory for working with Edit framework
  - Default editor
- Extensible model import/export framework
  - Contribute custom model source import modules
  - Contribute custom export modules



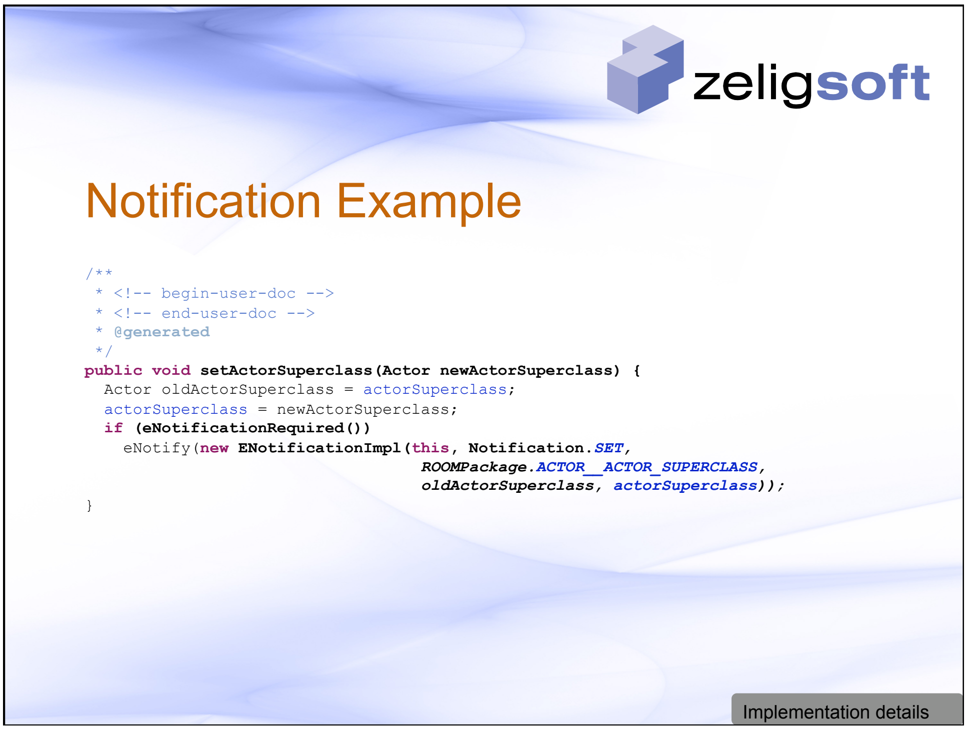
Persistence

- EMF refers to persisted data as a Resource
- Model objects can be spread across resources
  - Proxy is an object referenced in a different resource
- EMF refers to the collection of resources as a ResourceSet
  - Resolve proxies between resources in the set
- Registry maintains the resource factory for different types of resources
  - For the creation of resources of a specific type
- Resources are identified by their their URI



Notification

- Every EObject is a notifier
  - Whenever it changes it notifies interested parties
- In EMF terminology observers are adapters
  - A way to extend or change the behavior
    - No subclassing is required
- AdapterFactory provides the means to add an adapters
  - adapterFactory.adapt(object, ITreeItemContentProvider.Class)
- Notification is automatically generated

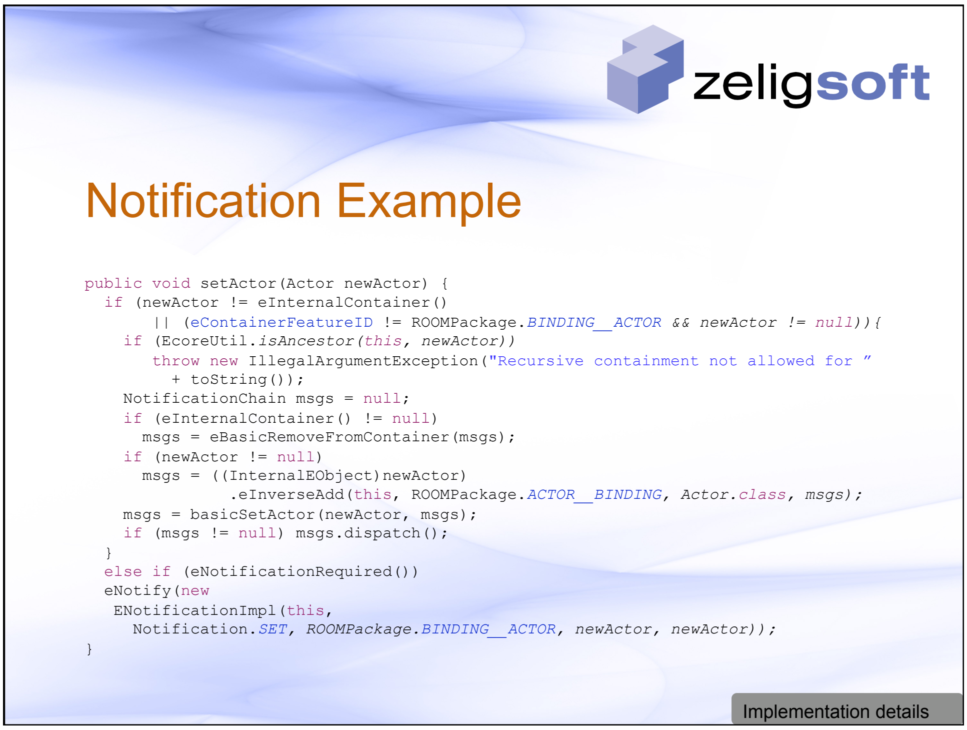


zeligsoft

Notification Example

```
/**
 * <!-- begin-user-doc -->
 * <!-- end-user-doc -->
 * @generated
 */
public void setActorSuperclass(Actor newActorSuperclass) {
  Actor oldActorSuperclass = actorSuperclass;
  actorSuperclass = newActorSuperclass;
  if (eNotificationRequired())
    eNotify(new ENotificationImpl(this, Notification.SET,
                                   ROOMPackage.ACTOR__ACTOR_SUPERCLASS,
                                   oldActorSuperclass, actorSuperclass));
}
```

Implementation details

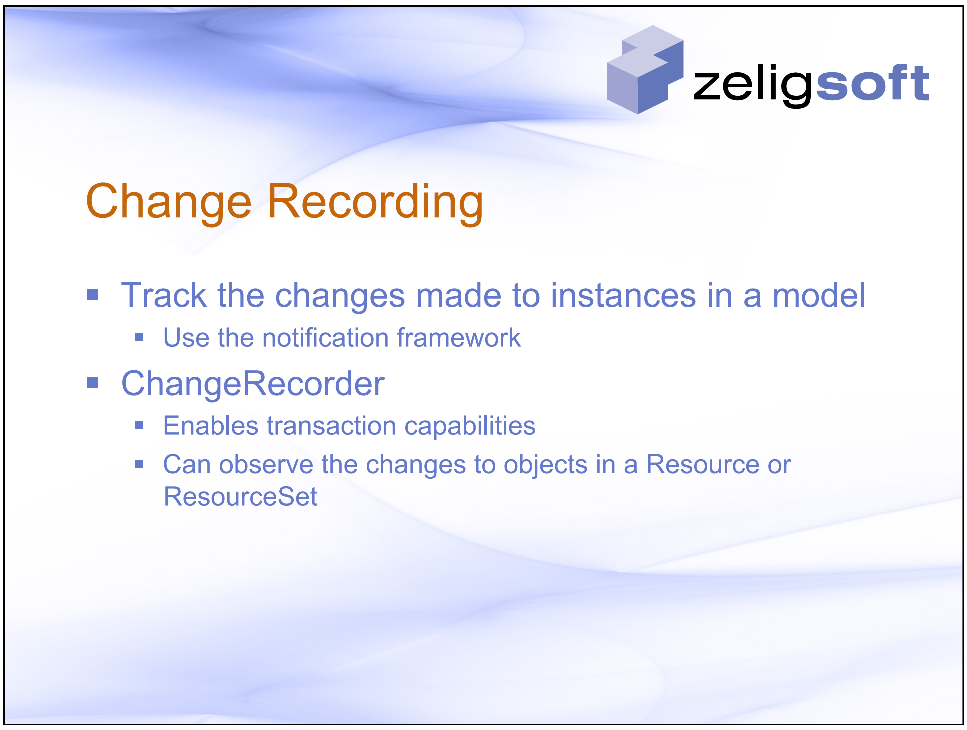


zeligsoft

Notification Example

```
public void setActor(Actor newActor) {
  if (newActor != eInternalContainer()
       || (eContainerFeatureID != ROOMPackage.BINDING__ACTOR && newActor != null)){
    if (EcoreUtil.isAncestor(this, newActor))
       throw new IllegalArgumentException("Recursive containment not allowed for ”
         + toString());
    NotificationChain msgs = null;
    if (eInternalContainer() != null)
      msgs = eBasicRemoveFromContainer(msgs);
    if (newActor != null)
      msgs = ((InternalEObject)newActor)
               .eInverseAdd(this, ROOMPackage.ACTOR__BINDING, Actor.class, msgs);
    msgs = basicSetActor(newActor, msgs);
    if (msgs != null) msgs.dispatch();
  }
  else if (eNotificationRequired())
  eNotify(new
   ENotificationImpl(this,
     Notification.SET, ROOMPackage.BINDING__ACTOR, newActor, newActor));
}
```

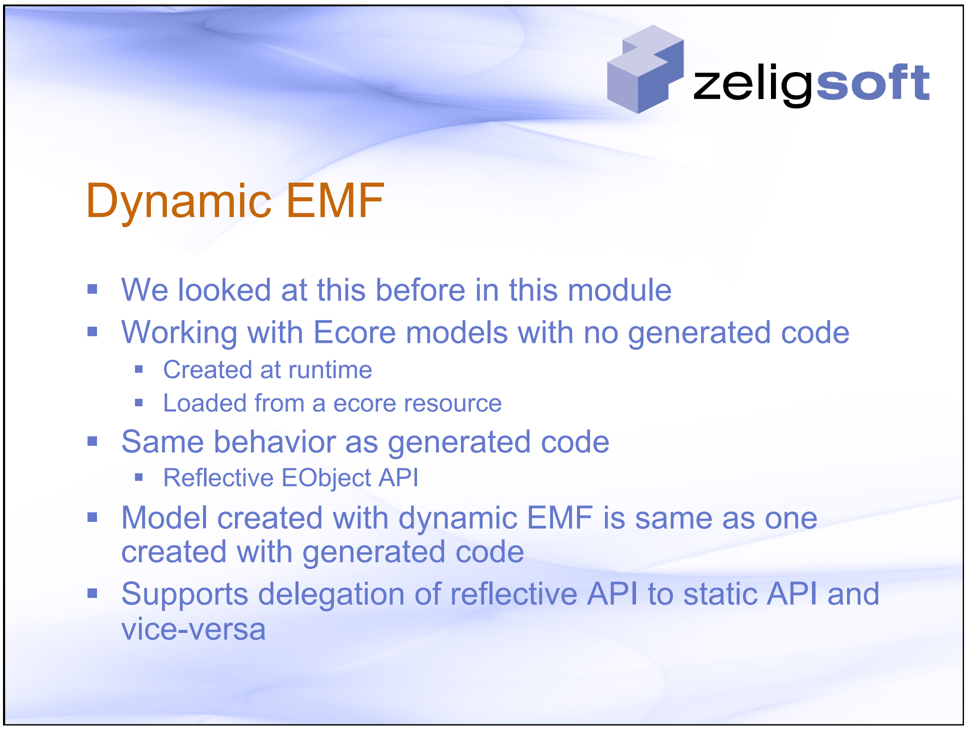
Implementation details



zeligsoft

Change Recording

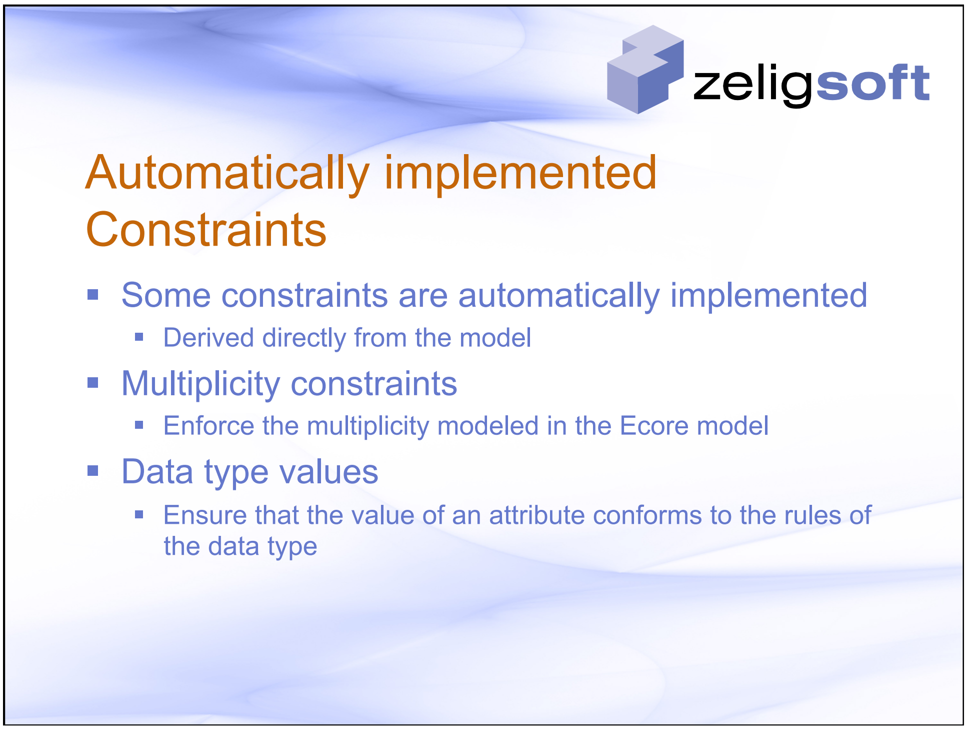
- Track the changes made to instances in a model
  - Use the notification framework
- ChangeRecorder
  - Enables transaction capabilities
  - Can observe the changes to objects in a Resource or ResourceSet



zeligsoft

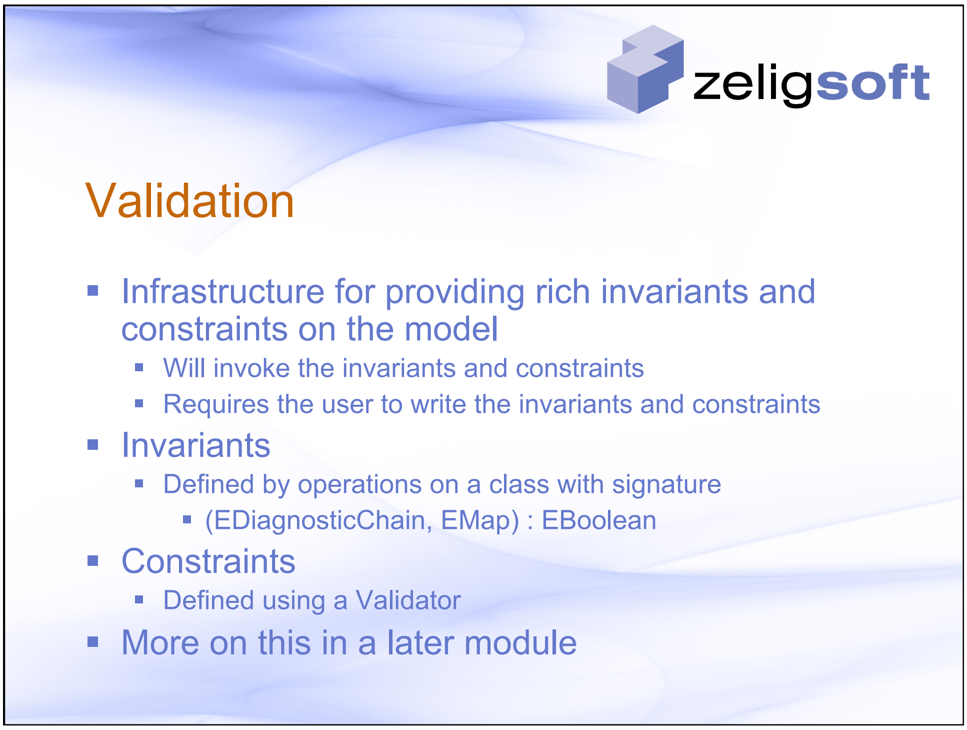
Dynamic EMF

- We looked at this before in this module
- Working with Ecore models with no generated code
  - Created at runtime
  - Loaded from a ecore resource
- Same behavior as generated code
  - Reflective EObject API
- Model created with dynamic EMF is same as one created with generated code
- Supports delegation of reflective API to static API and vice-versa



Automatically implemented Constraints

- Some constraints are automatically implemented
  - Derived directly from the model
- Multiplicity constraints
  - Enforce the multiplicity modeled in the Ecore model
- Data type values
  - Ensure that the value of an attribute conforms to the rules of the data type



Validation

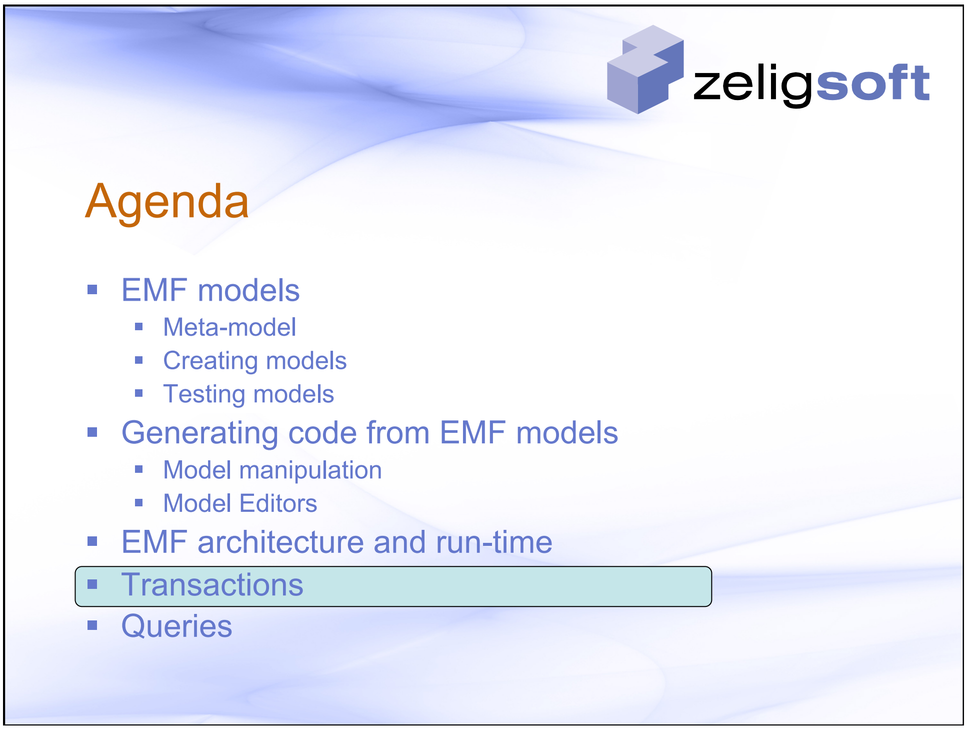
- Infrastructure for providing rich invariants and constraints on the model
  - Will invoke the invariants and constraints
  - Requires the user to write the invariants and constraints
- Invariants
  - Defined by operations on a class with signature
    - (EDiagnosticChain, EMap) : EBoolean
- Constraints
  - Defined using a Validator
- More on this in a later module

EMF Utilities

- Copying
  - EcoreUtil.copy
- Equality
  - EcoreUtil.equal
- Cross-referencing, contents/container navigation, annotation, proxy resolution, adapter selection, …

Summary

- In this module we explored
  - EMF architecture
  - EMF runtime aspects
    - Notification
    - Persistance
    - Change recording
    - Validation and constraints

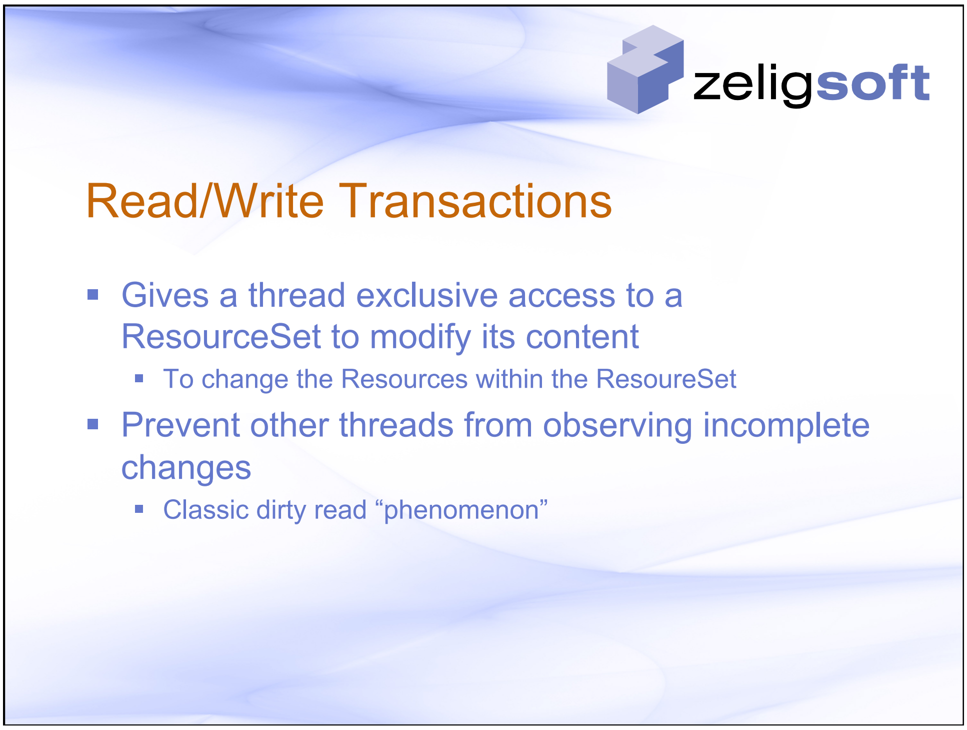


Agenda

- EMF models
  - Meta-model
  - Creating models
  - Testing models
- Generating code from EMF models
  - Model manipulation
  - Model Editors
- EMF architecture and run-time
- Transactions
- Queries

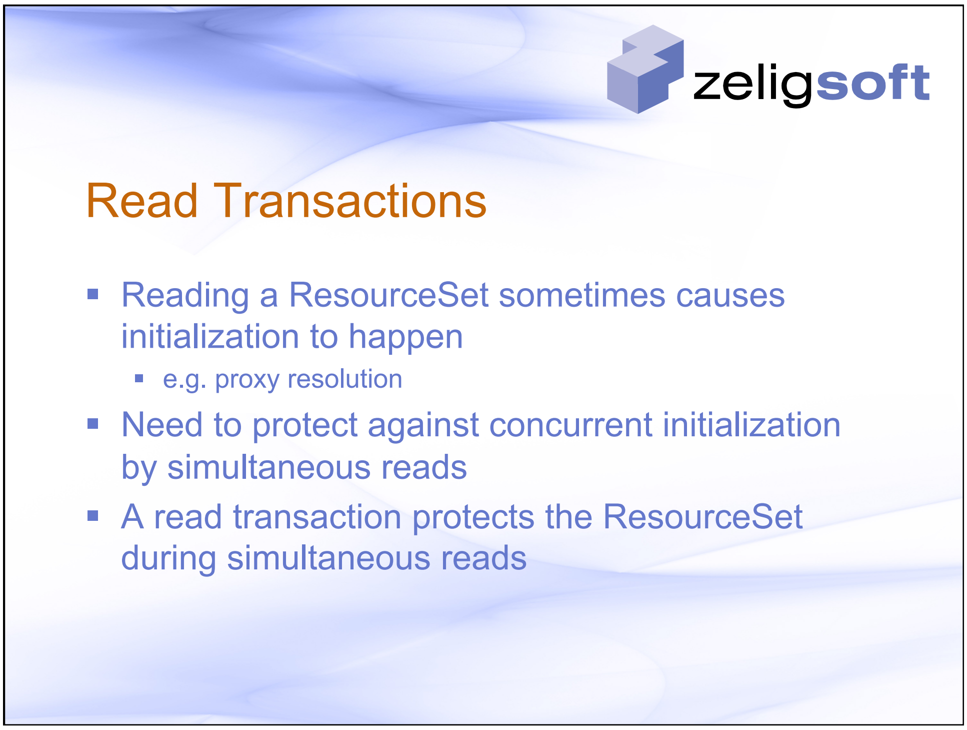
EMF Transaction

- A framework for managing multiple readers and writers to EMF resources or sets of resources
- An editing domain object manages access to resources
  - Can be shared amongst applications
- A transaction object is the unit of work
- Support for rolling back changes
  - Support for workbench undo/redo infrastructure



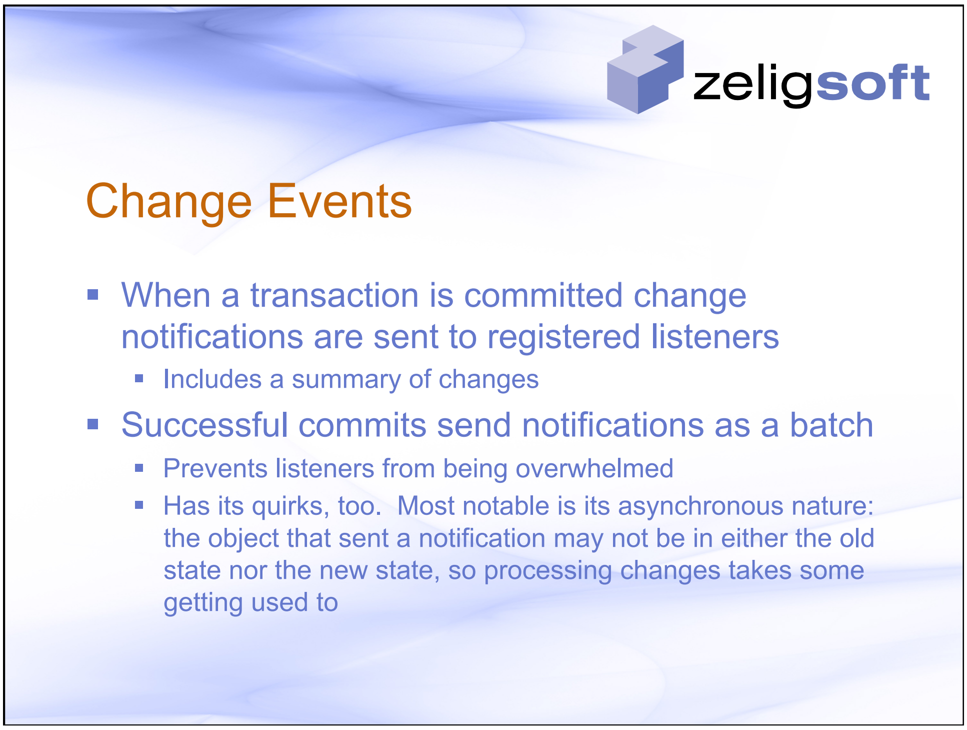
Read/Write Transactions

- Gives a thread exclusive access to a ResourceSet to modify its content
  - To change the Resources within the ResoureSet
- Prevent other threads from observing incomplete changes
  - Classic dirty read “phenomenon”



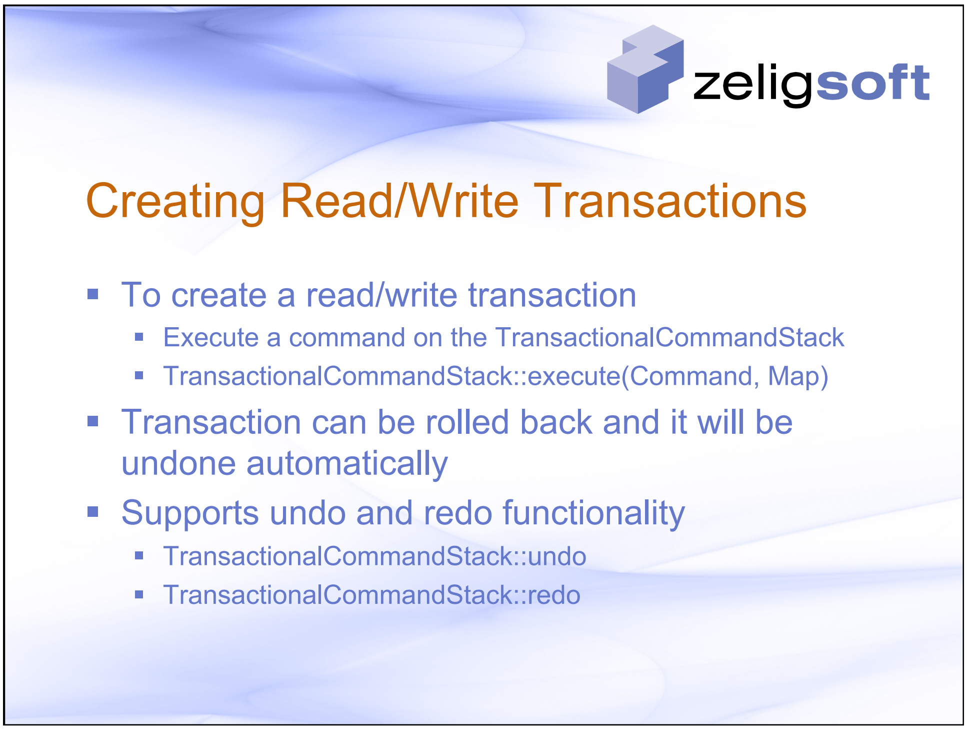
Read Transactions

- Reading a ResourceSet sometimes causes initialization to happen
  - e.g. proxy resolution
- Need to protect against concurrent initialization by simultaneous reads
- A read transaction protects the ResourceSet during simultaneous reads



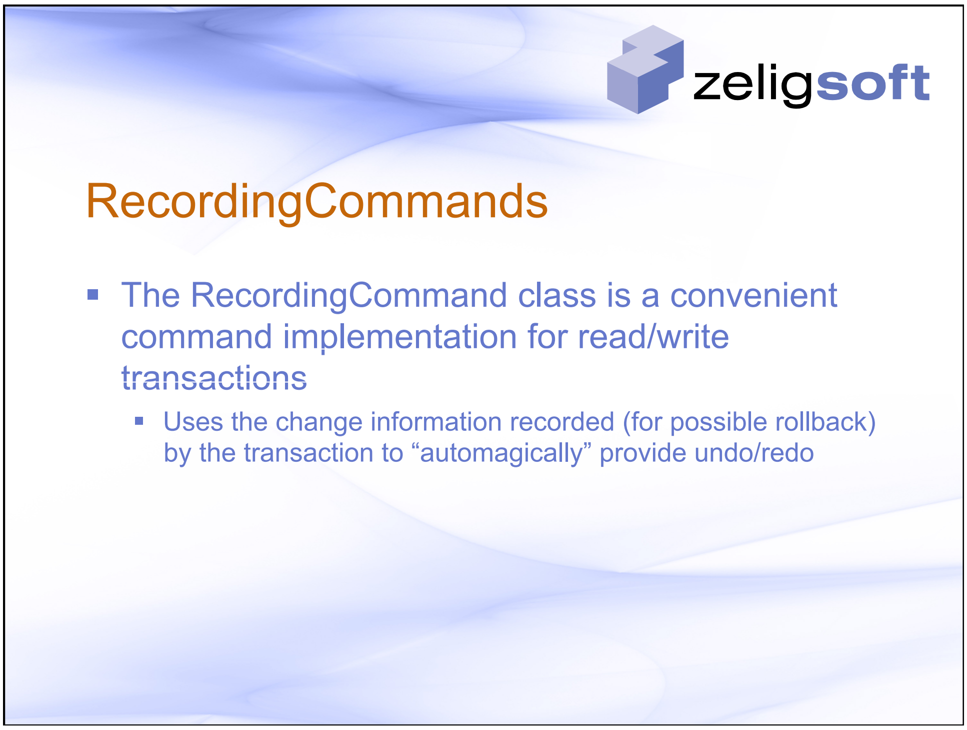
Change Events

- When a transaction is committed change notifications are sent to registered listeners
  - Includes a summary of changes
- Successful commits send notifications as a batch
  - Prevents listeners from being overwhelmed
  - Has its quirks, too. Most notable is its asynchronous nature: the object that sent a notification may not be in either the old state nor the new state, so processing changes takes some getting used to



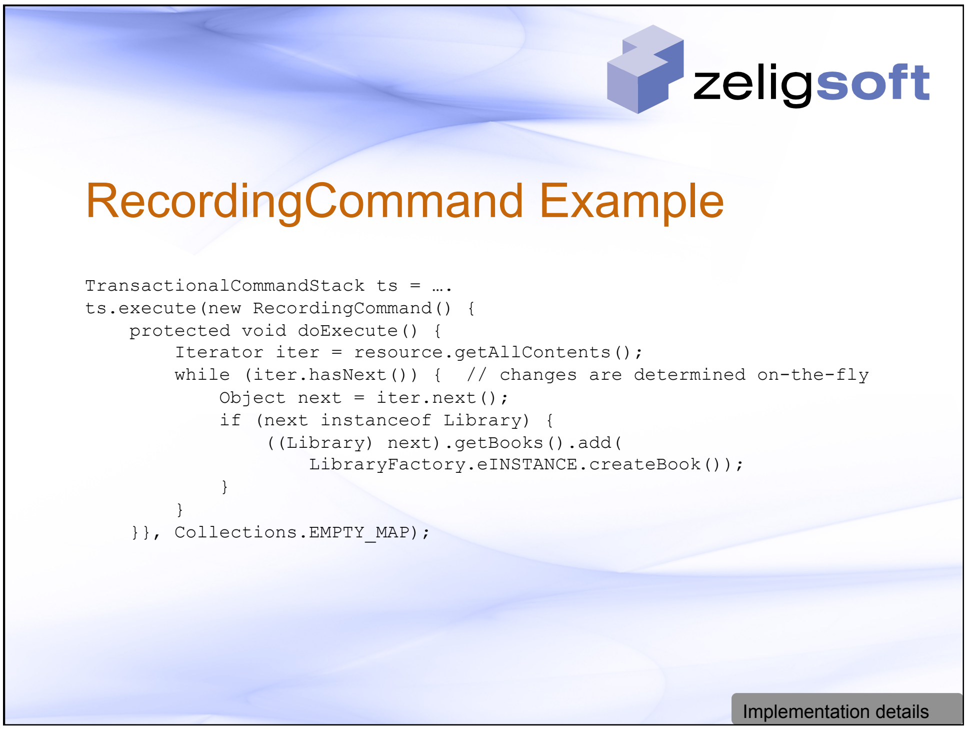
Creating Read/Write Transactions

- To create a read/write transaction
  - Execute a command on the TransactionalCommandStack
  - TransactionalCommandStack::execute(Command, Map)
- Transaction can be rolled back and it will be undone automatically
- Supports undo and redo functionality
  - TransactionalCommandStack::undo
  - TransactionalCommandStack::redo



RecordingCommands

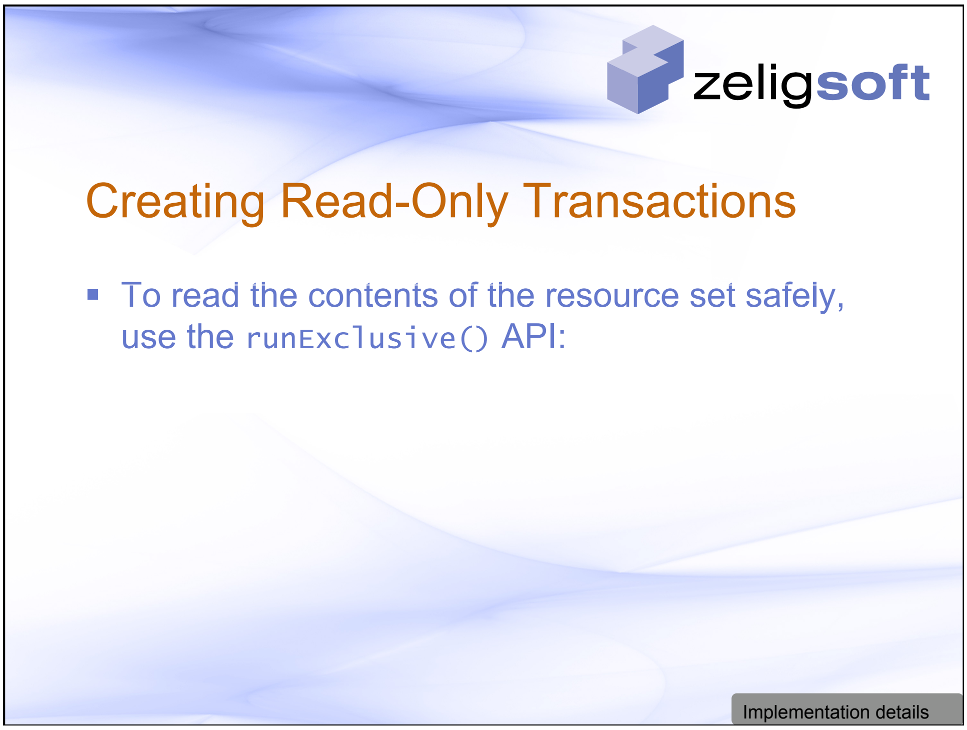
- The RecordingCommand class is a convenient command implementation for read/write transactions
  - Uses the change information recorded (for possible rollback) by the transaction to “automagically” provide undo/redo



RecordingCommand Example

```
TransactionalCommandStack ts = ….
ts.execute(new RecordingCommand() {
    protected void doExecute() {
        Iterator iter = resource.getAllContents();
        while (iter.hasNext()) {  // changes are determined on-the-fly
            Object next = iter.next();
            if (next instanceof Library) {
                ((Library) next).getBooks().add(
                    LibraryFactory.eINSTANCE.createBook());
            }
        }
    }}, Collections.EMPTY_MAP);
```

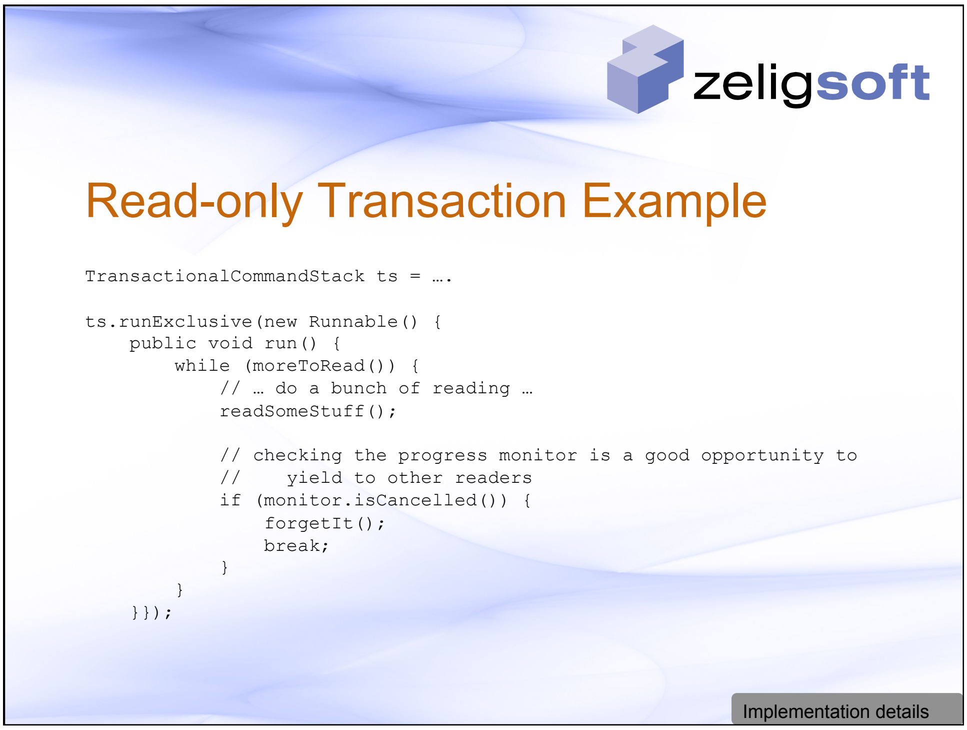
Implementation details



zeligsoft

Creating Read-Only Transactions

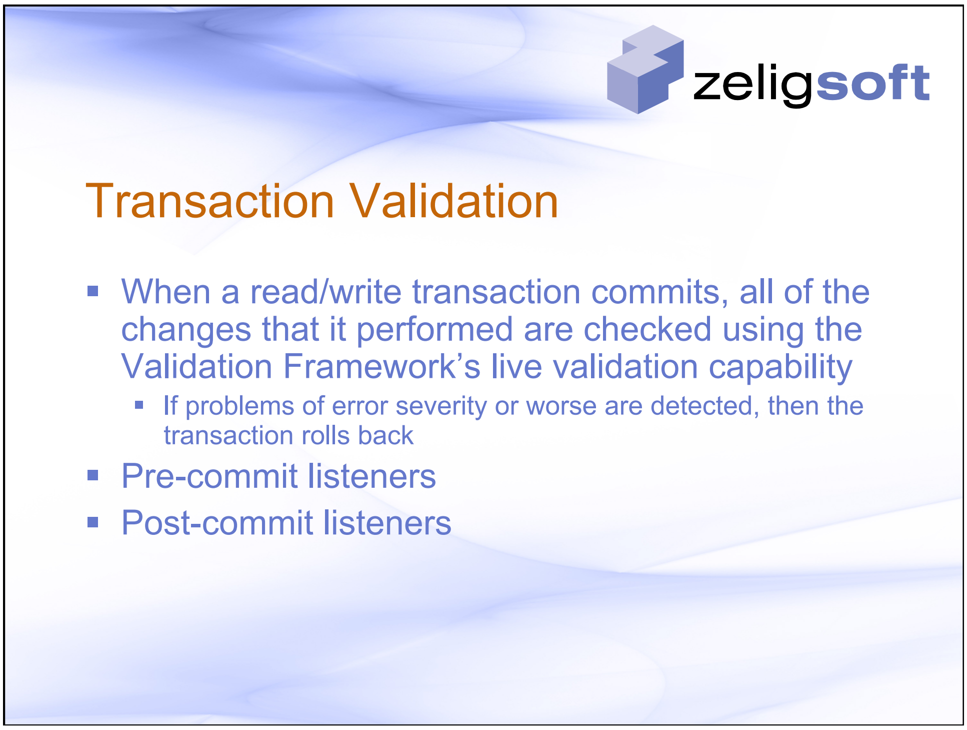
- To read the contents of the resource set safely, use the `runExclusive()` API:



zeligsoft

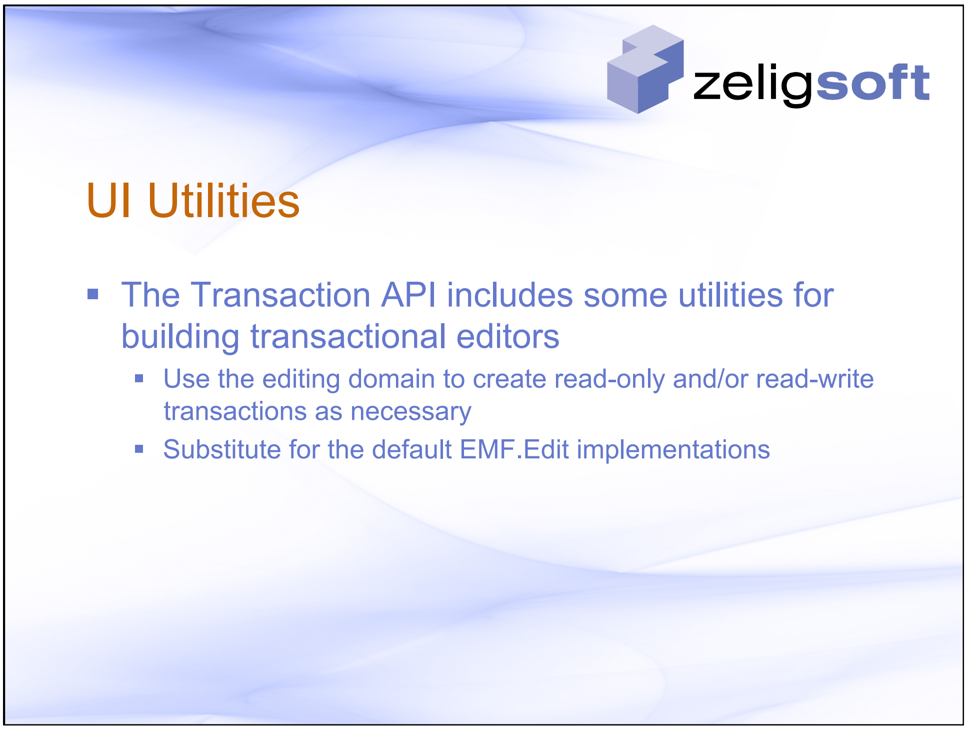
Read-only Transaction Example

```
TransactionalCommandStack ts = ….

ts.runExclusive(new Runnable() {
    public void run() {
        while (moreToRead()) {
            // … do a bunch of reading …
            readSomeStuff();

            // checking the progress monitor is a good opportunity to
            //    yield to other readers
            if (monitor.isCancelled()) {
                forgetIt();
                break;
            }
        }
    }});
```



Transaction Validation

- When a read/write transaction commits, all of the changes that it performed are checked using the Validation Framework’s live validation capability
  - If problems of error severity or worse are detected, then the transaction rolls back
- Pre-commit listeners
- Post-commit listeners



UI Utilities

- The Transaction API includes some utilities for building transactional editors
  - Use the editing domain to create read-only and/or read-write transactions as necessary
  - Substitute for the default EMF.Edit implementations

zeligsoft

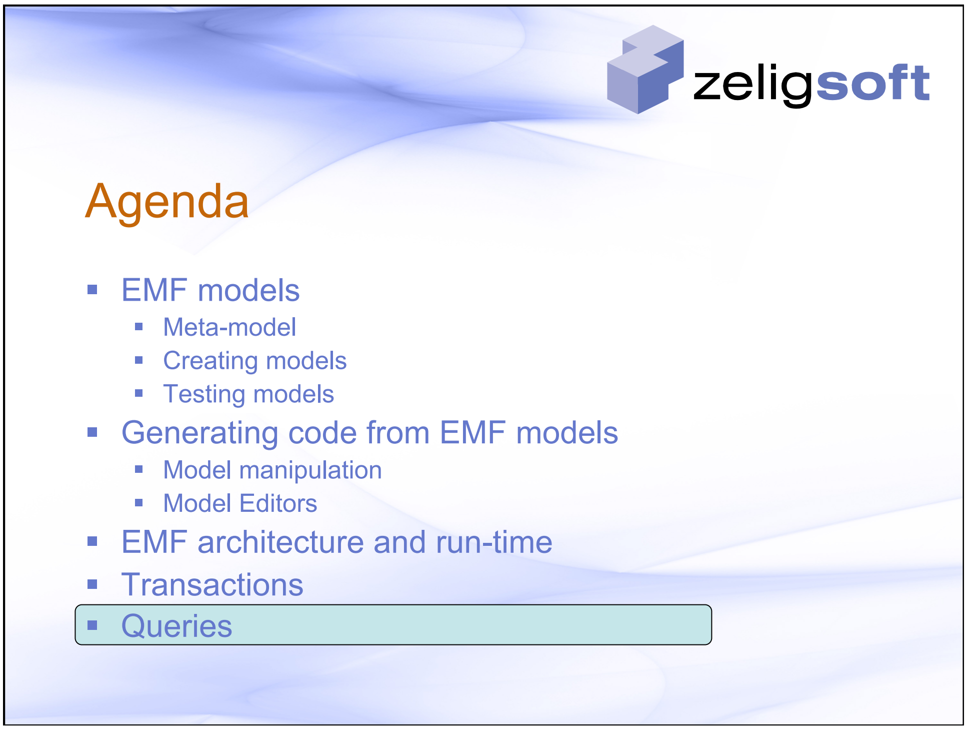
ResourceSetListener

- Avoid reacting to changes before they have been committed
- If a transaction rolls back, no event is sent
  - Except for changes that are not undoable, such as proxy resolution and resource loading. In general, these are the changes that are compatible with a read-only transaction

zeligsoft

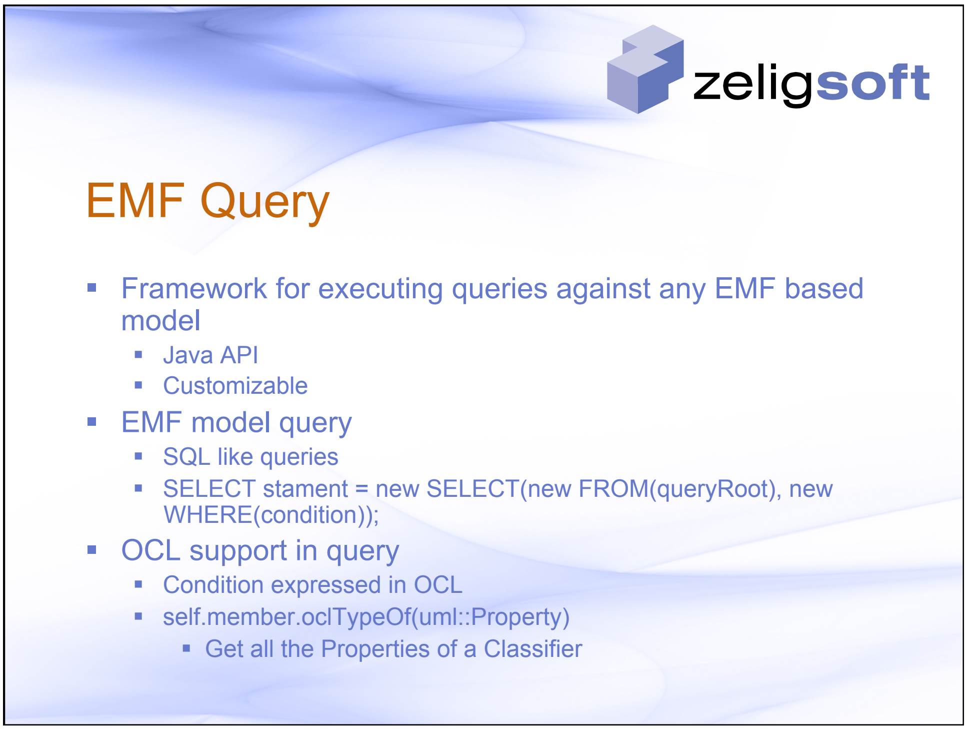
Summary

- In this module we have discussed
  - EMF transactions and how to respond to their changes



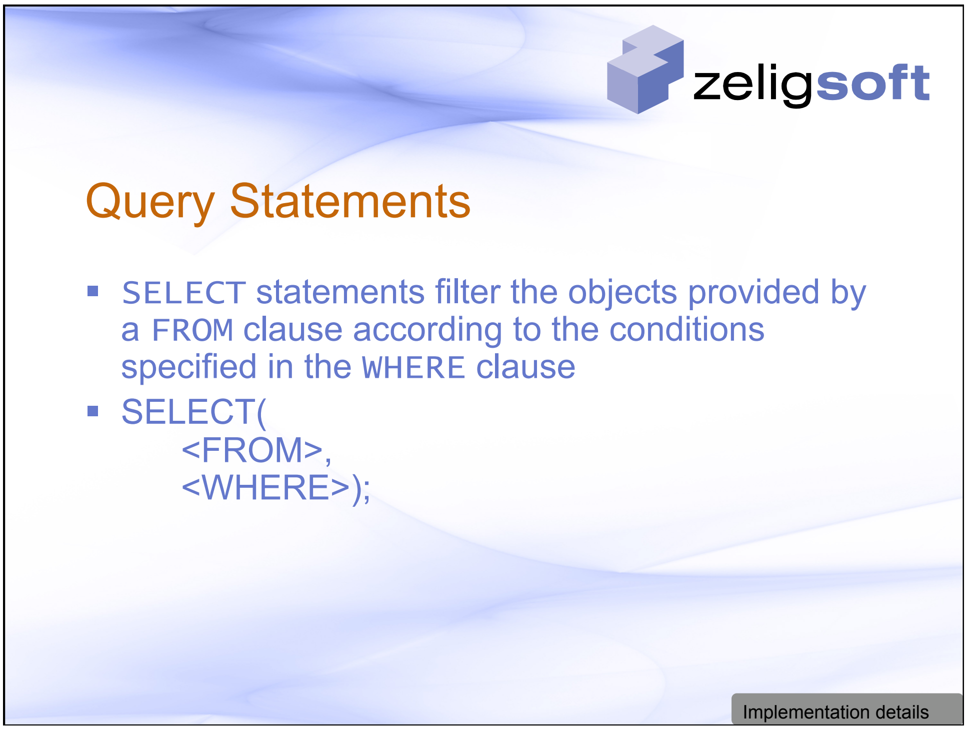
Agenda

- EMF models
  - Meta-model
  - Creating models
  - Testing models
- Generating code from EMF models
  - Model manipulation
  - Model Editors
- EMF architecture and run-time
- Transactions
- Queries



EMF Query

- Framework for executing queries against any EMF based model
  - Java API
  - Customizable
- EMF model query
  - SQL like queries
  - SELECT stament = new SELECT(new FROM(queryRoot), new WHERE(condition));
- OCL support in query
  - Condition expressed in OCL
  - self.member.oclTypeOf(uml::Property)
    - Get all the Properties of a Classifier



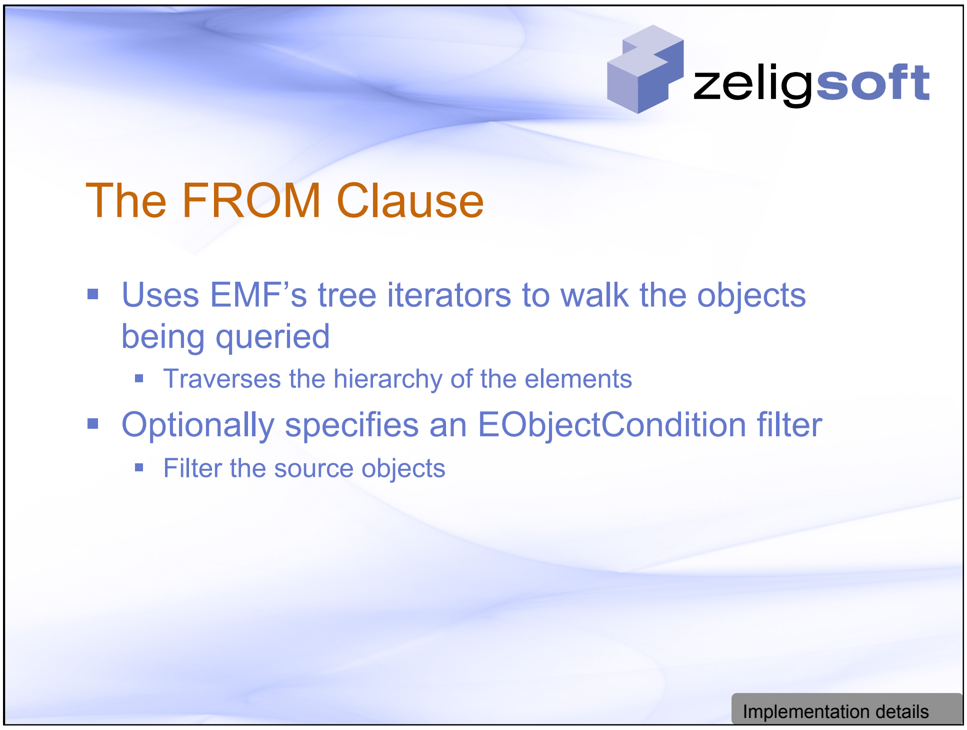
zeligsoft

Query Statements

- `SELECT` statements filter the objects provided by a `FROM` clause according to the conditions specified in the `WHERE` clause
- SELECT(

```
    <FROM>,
    <WHERE>);
```

Implementation details

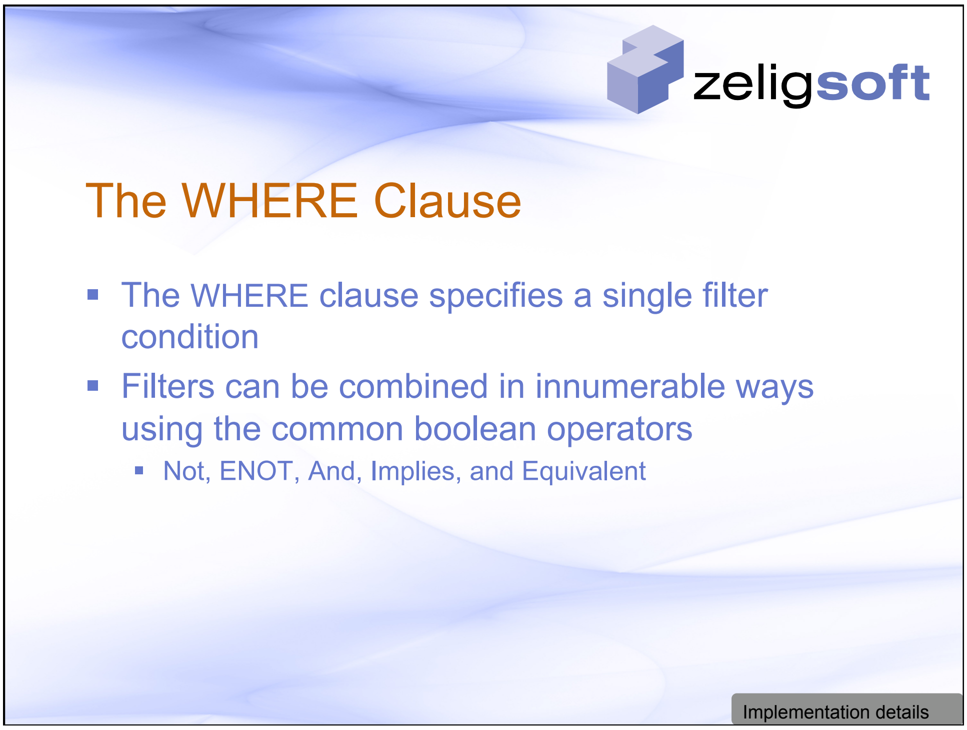


zeligsoft

The FROM Clause

- Uses EMF’s tree iterators to walk the objects being queried
  - Traverses the hierarchy of the elements
- Optionally specifies an EObjectCondition filter
  - Filter the source objects

Implementation details

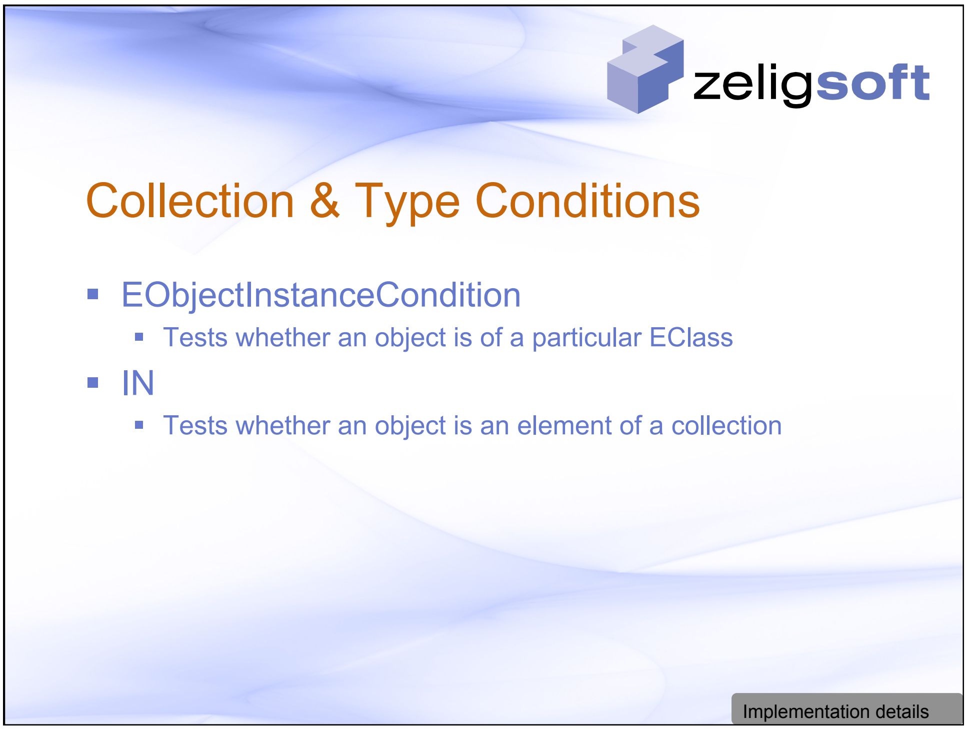


zeligsoft

The WHERE Clause

- The WHERE clause specifies a single filter condition
- Filters can be combined in innumerable ways using the common boolean operators
  - Not, ENOT, And, Implies, and Equivalent

Implementation details

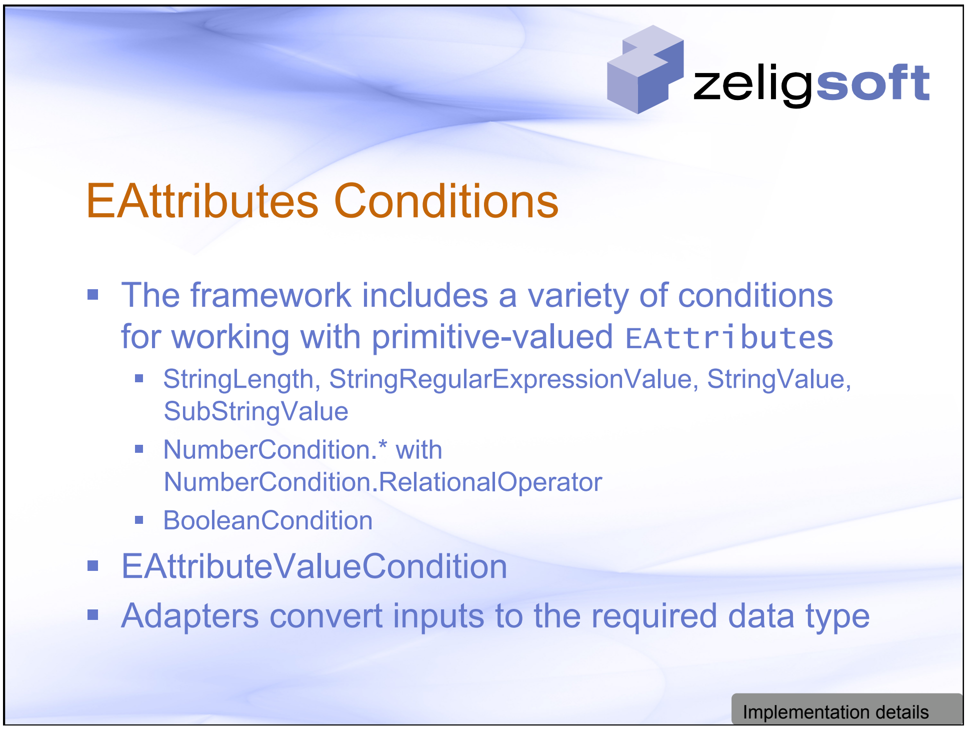


zeligsoft

Collection & Type Conditions

- EObjectInstanceCondition
  - Tests whether an object is of a particular EClass
- IN
  - Tests whether an object is an element of a collection

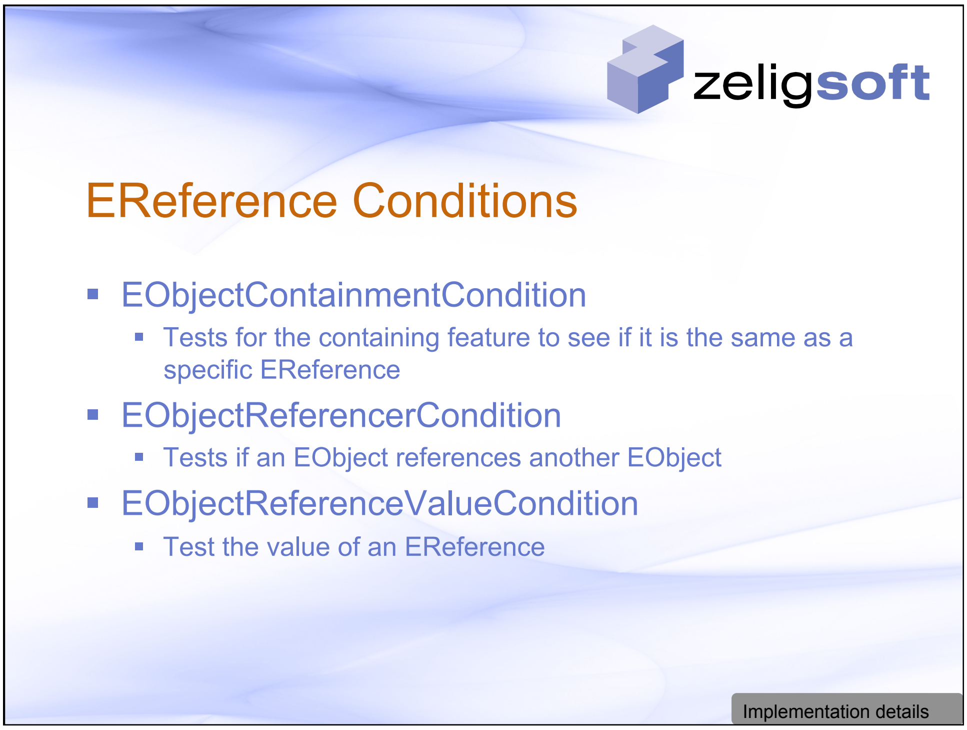
Implementation details



EAttributes Conditions

- The framework includes a variety of conditions for working with primitive-valued `EAttributes`
  - StringLength, StringRegularExpressionValue, StringValue, SubStringValue
  - NumberCondition.* with NumberCondition.RelationalOperator
  - BooleanCondition
- EAttributeValueCondition
- Adapters convert inputs to the required data type

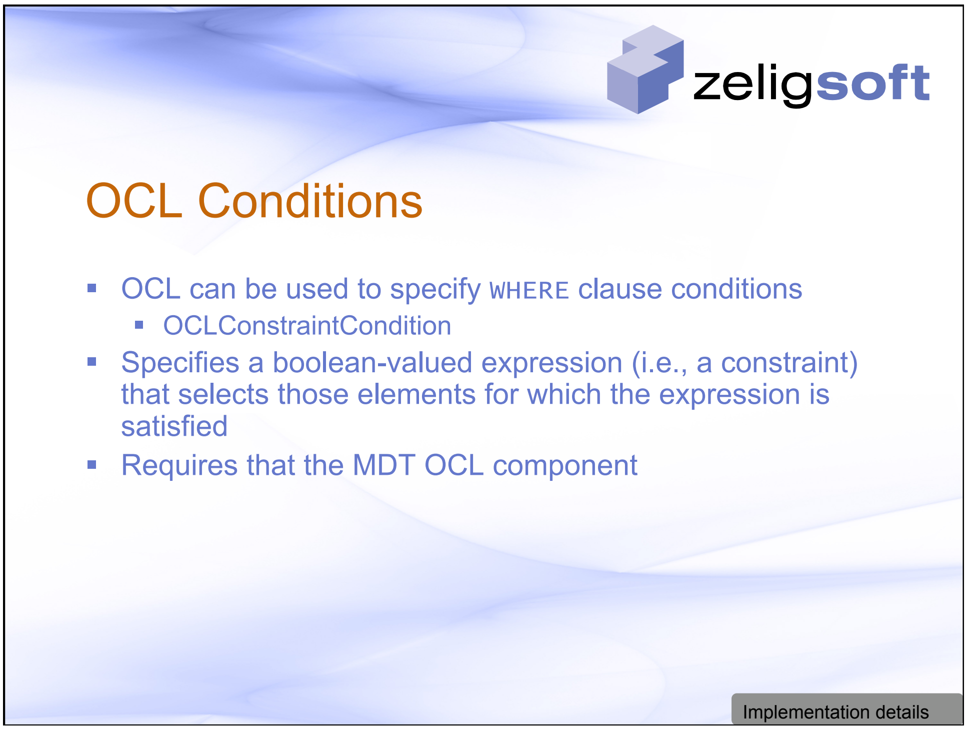
Implementation details



EReference Conditions

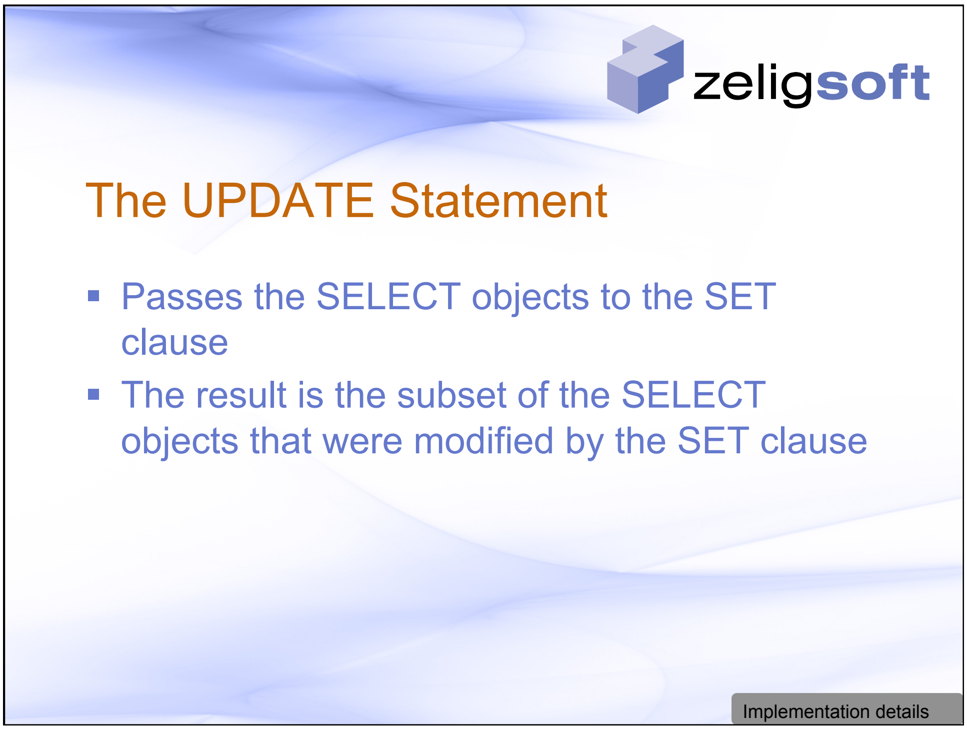
- EObjectContainmentCondition
  - Tests for the containing feature to see if it is the same as a specific EReference
- EObjectReferencerCondition
  - Tests if an EObject references another EObject
- EObjectReferenceValueCondition
  - Test the value of an EReference

Implementation details



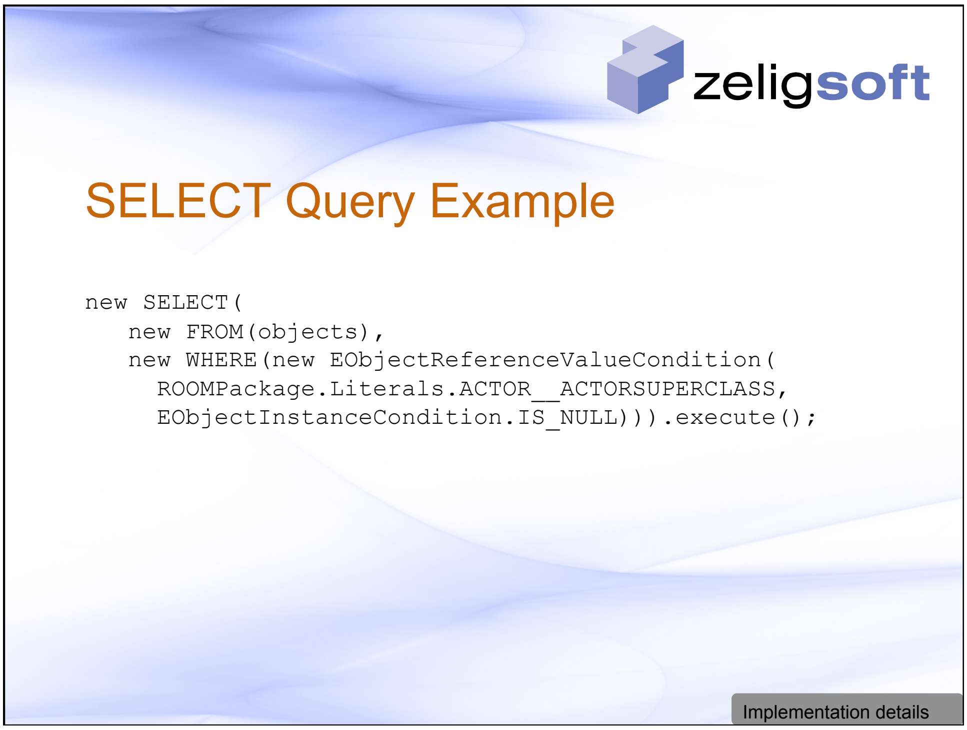
OCL Conditions

- OCL can be used to specify WHERE clause conditions
  - OCLConstraintCondition
- Specifies a boolean-valued expression (i.e., a constraint) that selects those elements for which the expression is satisfied
- Requires that the MDT OCL component



The UPDATE Statement

- Passes the SELECT objects to the SET clause
- The result is the subset of the SELECT objects that were modified by the SET clause

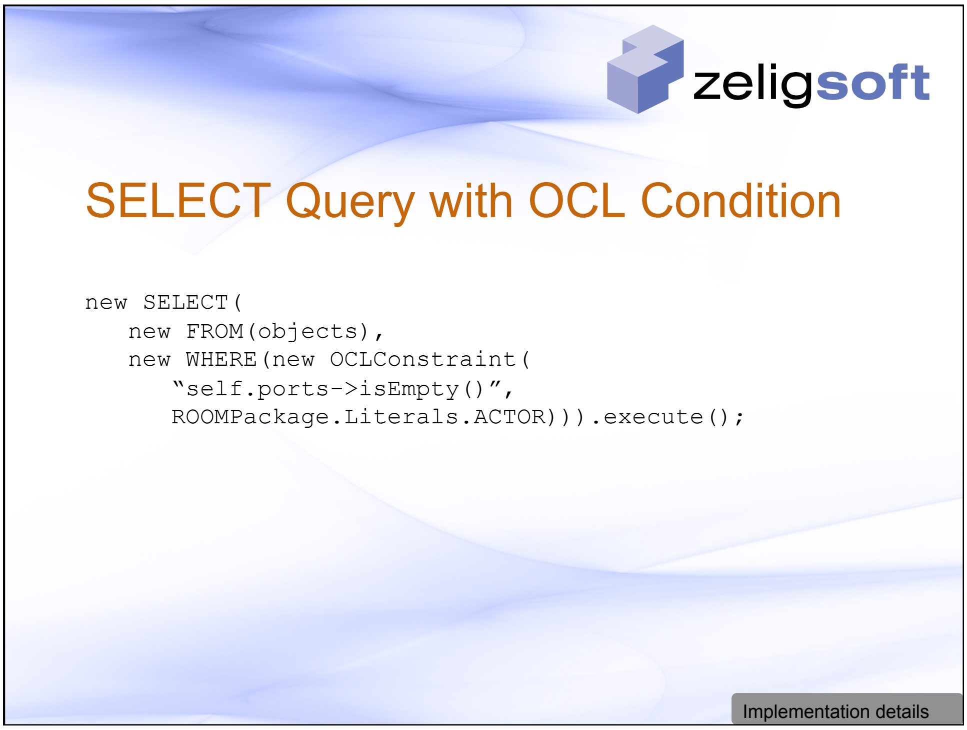


zeligsoft

SELECT Query Example

```
new SELECT(
   new FROM(objects),
   new WHERE(new EObjectReferenceValueCondition(
     ROOMPackage.Literals.ACTOR__ACTORSUPERCLASS,
     EObjectInstanceCondition.IS_NULL))).execute();
```

Implementation details

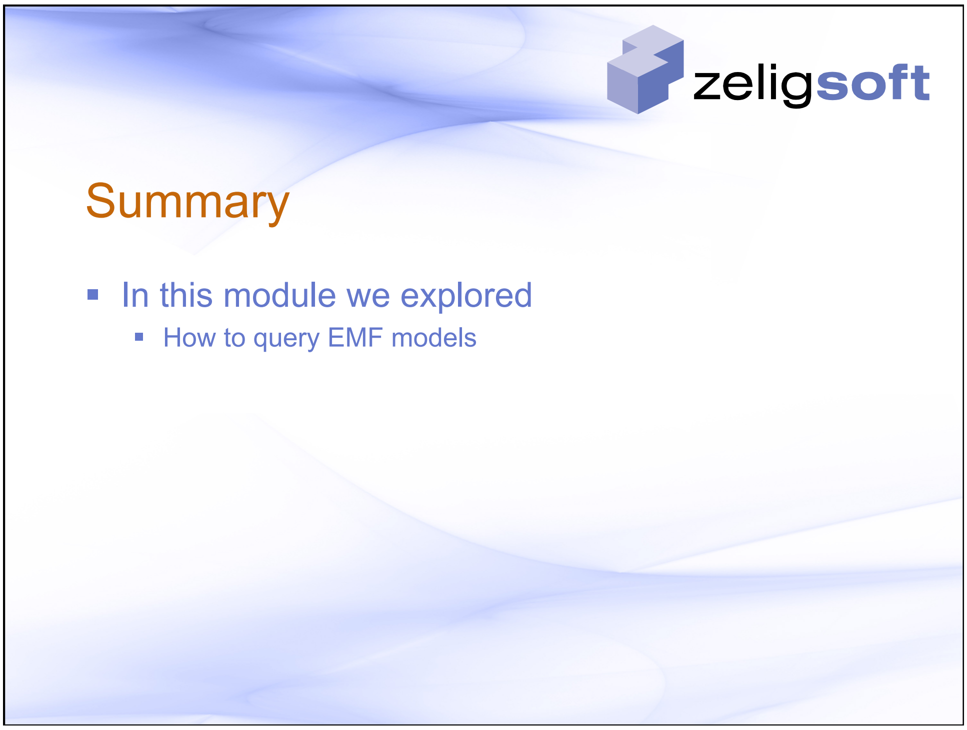


zeligsoft

SELECT Query with OCL Condition

```
new SELECT(
   new FROM(objects),
   new WHERE(new OCLConstraint(
      “self.ports->isEmpty()”,
      ROOMPackage.Literals.ACTOR))).execute();
```

Implementation details



zeligsoft

Summary

- In this module we explored
  - How to query EMF models

zeligsoft

Questions