

Exploring EMF

What is the Eclipse Modeling Framework?

Overview

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



Agenda

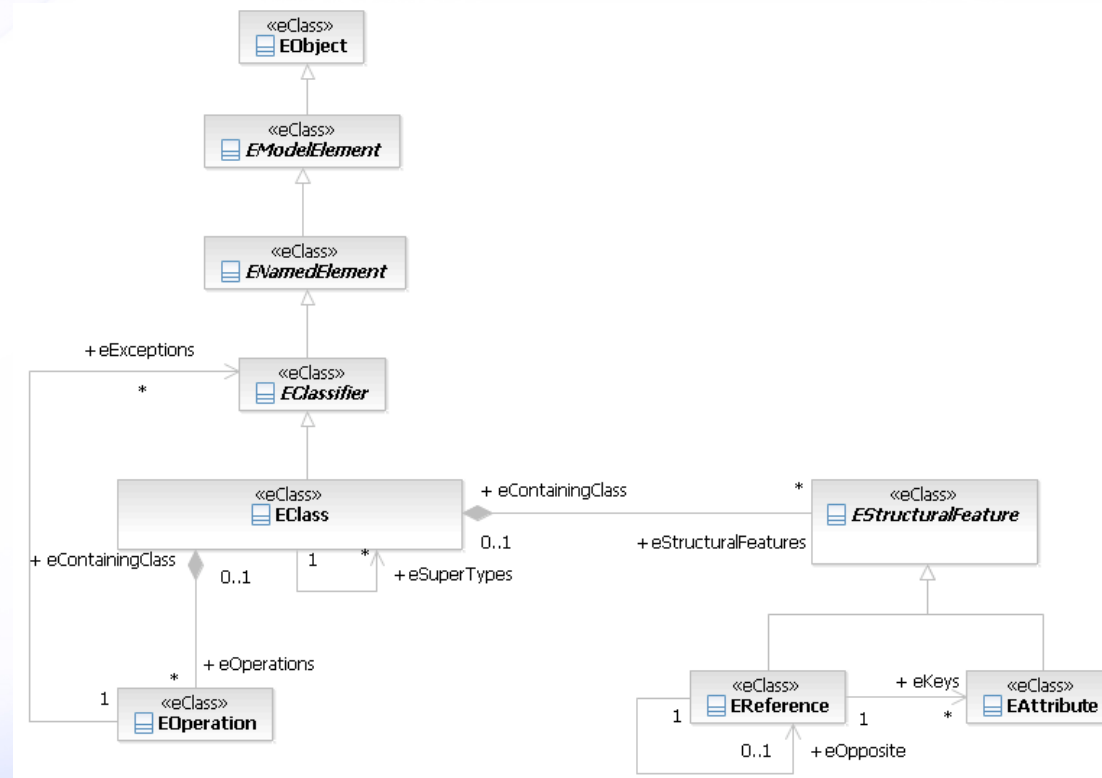
Eclipse Modeling Framework

- 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?

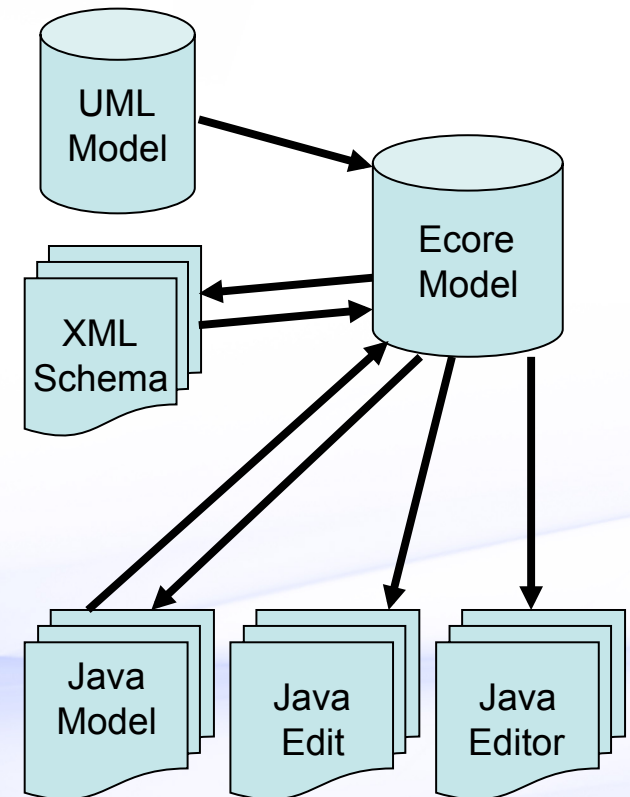
- Description of data for a domain/application
 - The attributes and capabilities of domain concepts
 - Relationships between the domain concepts
 - Cardinalities of attributes and relationships
- It defines the *metamodel* or abstract syntax for your domain
- Defined in the Ecore modeling language

The Ecore Metamodel



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



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();
}

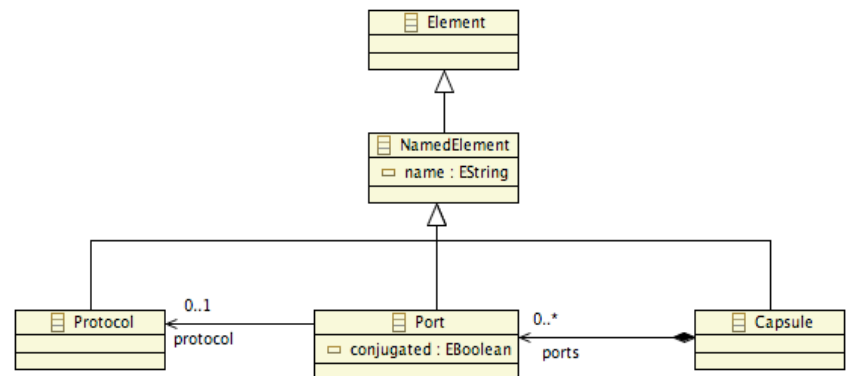
/**
 * @model
 */
boolean isConjugated();
}
```


More EMF from Java

- Create Java interfaces for 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

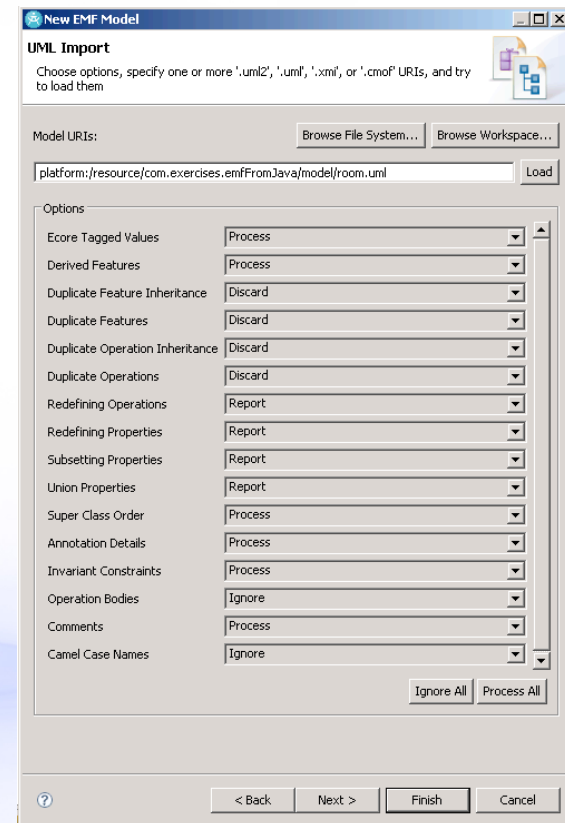
EMF from UML Class Model

- 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



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...



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

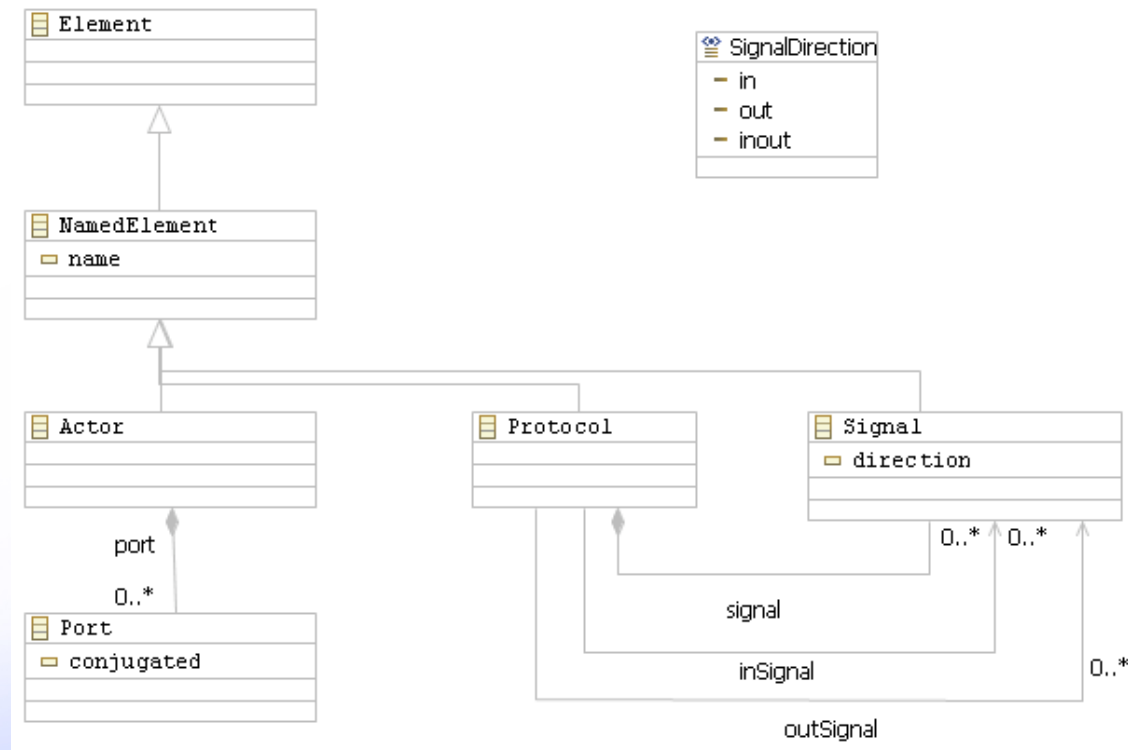
More EMF from XML Schema

- 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

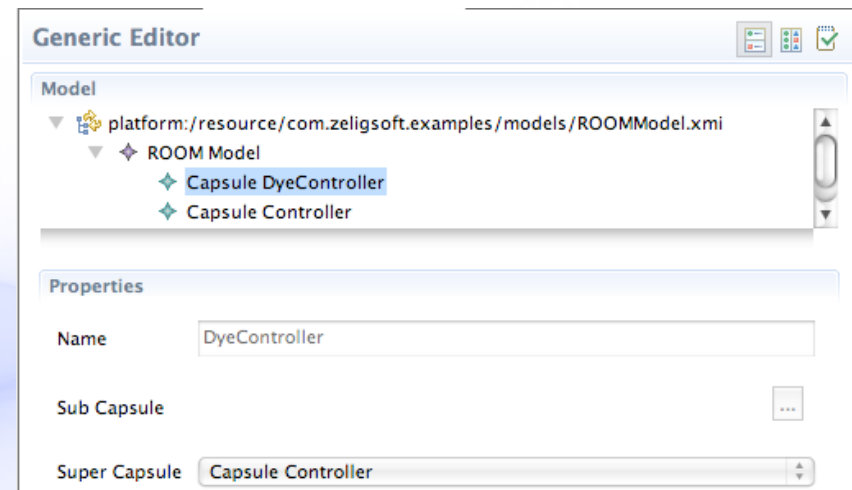
- The Ecore Diagram Editor provides a Class diagram like editor for graphically modeling
- Built on top of the GMF framework
- Editor is canonical
 - One diagram per EMF model
- Creates model and diagram resources
 - File --> New --> Other...
 - Ecore Diagram

More EMF with Ecore Diagram Editor



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



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(ns)

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

Using 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

More Reflective API

```
//get the EPackage for the model we are working with using the
//namespace URI that it is registered against
EPackage ePackage = EPackage.Registry.INSTANCE.getEPackage("java.xml");
//using the package get an EClass that we want to instantiate
EClass eClass = (EClass)ePackage.getEClassifier("JavaClass");
//get a EStructuralFeature that we want to set on an instance of the EClass
EStructuralFeature nameFeature = eClass.getEStructuralFeature("name");
//instantiate the EClass
EObject javaClass = ePackage.getEFactoryInstance().create(eClass);
//set the name of the instantiate EClass
javaClass.eSet(nameFeature, "DyeController");
```

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

Generating Code with EMF

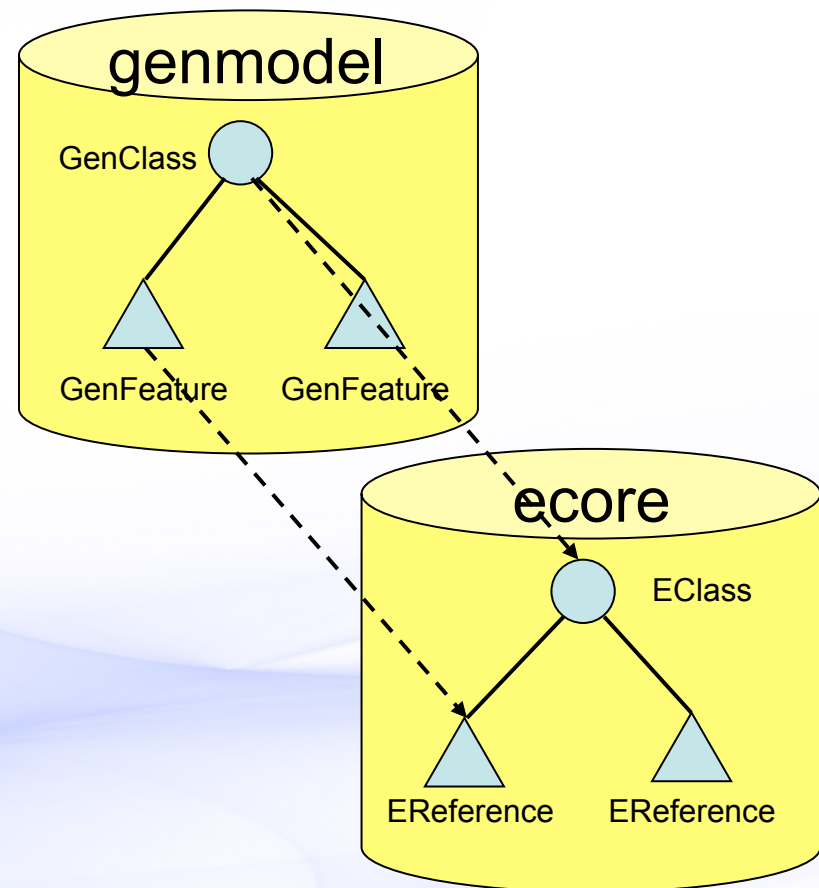
- Code can be generated from the Ecore model to work with it
 - Model specific API rather than the reflective API
 - Support for editing model instances in an editor
- Driven by a generator model
 - Annotates the Ecore model to control generation
- Generated code can be augmented
 - Derived attributes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












The genmodel

- Wraps the Ecore model and automatically kept in sync
- 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 to generate
 - 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

More Model Code

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

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

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

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

Model Editor Code

- User interface specific editor code
- 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>Advisor.java
	Action Bar Contributor	<Prefix>ActionBarContributor.java
	Wizard	<Prefix>ModelWizard.java

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

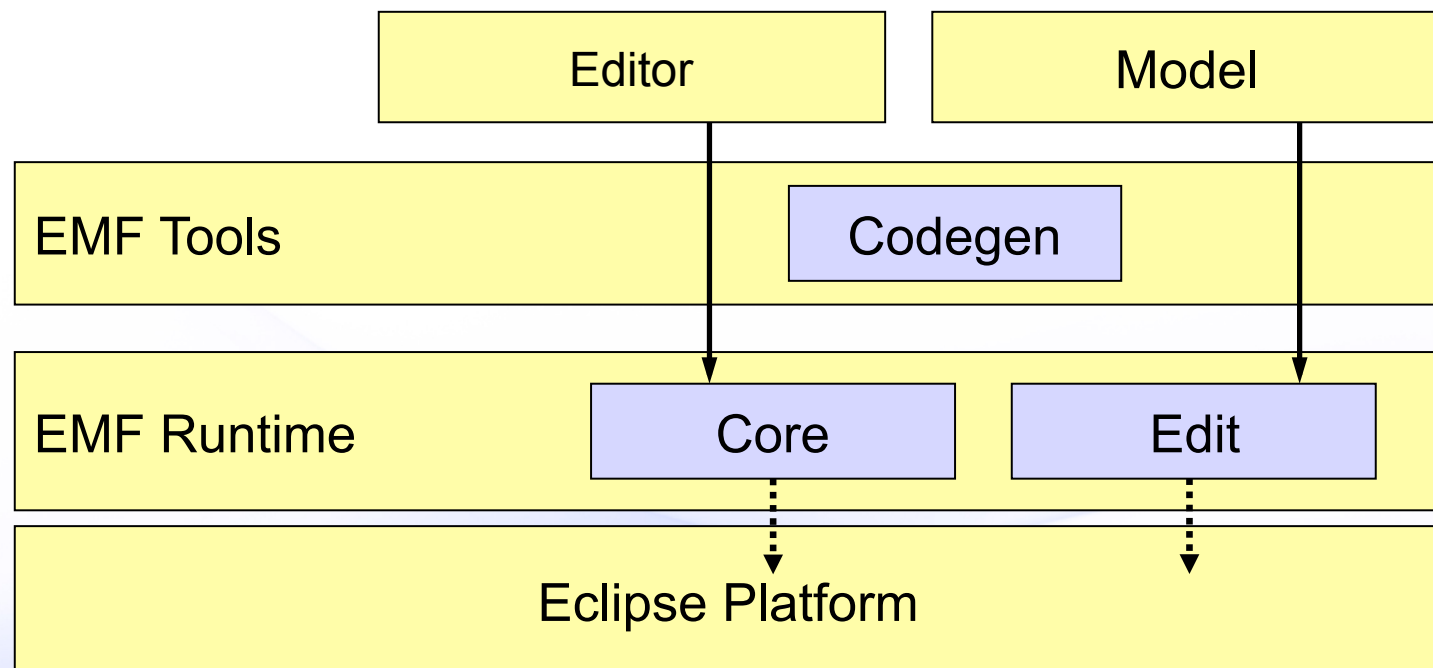
Regeneration and Merge

- 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

Summary

- In this module we explored

EMF Architecture



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

Creating a Resource

- [TM TODO]
 - ResourceSet
 - Resource.Factory
 - Saving Resource
 - Loading Resource

Notification

- [TM TODO]

Dynamic EMF

- 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

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

Automatically implemented Constraints

- 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

Validator

- Invokes the invariants and constraints
- Invariants are defined by <<inv>> operations on a class
- Constraints are defined using a Validator

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 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 ResourceSet
- Prevent other threads from observing incomplete changes

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



Workbench Integration

Obtaining the Editing Domain

- Any editor that needs to access the registered editing domain simply gets it from the registry
 - It is lazily created, using the associated factory, on first access

Creating Read/Write Transactions

- To do work in a read/write transaction, simply execute a `Command` on the `TransactionalCommandStack`
 - Just like using a regular EMF Editing Domain
- If the transaction needs to roll back, it will be undone automatically and will not be appended to the stack
 - In order to find out when rollback occurs, use the `TransactionalCommandStack::execute(Command, Map)` method, which throws `RollbackException`, instead of using `CommandStack::execute(Command)`
 - This method also accepts a map of options to configure the transaction that will be created (more on options, later)

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

Transaction Options

- The `TransactionalCommandStack::execute()` method accepts a map of options defined by the Transaction interface that determine how changes occurring during the transaction are handled:
 - `OPTION_NO_NOTIFICATIONS`: changes are not included in post-commit change events
 - `OPTION_NO_TRIGGERS`: changes are not included in pre-commit change events
 - `OPTION_NO_VALIDATION`: changes are not validated
 - `OPTION_NO_UNDO`: changes are not recorded for undo/redo and rollback. Use with extreme caution!

Transaction Options

- (continued from previous slide)
 - `OPTION_UNPROTECTED`: implies `OPTION_NO_UNDO`, `OPTION_NO_VALIDATION`, and `OPTION_NO_TRIGGERS`. In addition, permits writing to the resource set even in an otherwise read-only context. Use with even more extreme caution!
- The `CommandStack::undo()` and `::redo()` methods use the following options for the undo/redo transaction:
 - `OPTION_NO_UNDO`: because we are undoing or redoing a previous recording, there is no need to record anew
 - `OPTION_NO_TRIGGERS`: triggers performed during execution were recorded and are automatically undone; any additional changes would be inappropriate
 - `OPTION_NO_VALIDATION`: there is no need to validate a reversion to a previous state of the data

Transaction Options

- The pre-defined options only apply to read/write transactions
 - Permitted on read-only transactions but have no effect
- Extensions of the transaction API can define custom options

Creating Read-Only Transactions

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

Transaction Sharing

- Read-only transactions on multiple threads can be interleaved by cooperatively yielding their read lock
 - Recommended for long-running read operations
 - Call `TransactionalEditingDomain::yield()` to yield read access to other threads waiting for read-only transactions
 - Yielding thread waits until other threads return read access to it either by finishing their transactions or by yielding
 - Yielding is fair: read access is always passed to the thread that has waited longest
- Write access is not shared in this way
 - Readers cannot yield to writers
 - Writers cannot yield to any others



Transaction Sharing

Transaction Sharing

- A thread that owns a transaction can lend it to another thread using a `PrivilegedRunnable`
 - The privileged runnable takes over the transaction for the duration of its `run()` method
 - Can be used to share read-only *and* read-write transactions
- Ideal for runnables that need to access the resource set and the UI thread at the same time:
 - Pass the privileged runnable to `Display.syncExec(Runnable)` API to run on the UI thread
- Must only be used with synchronous inter-thread communication such as `syncExec`
 - The originator thread loses the transaction during the runnable



Transaction Sharing

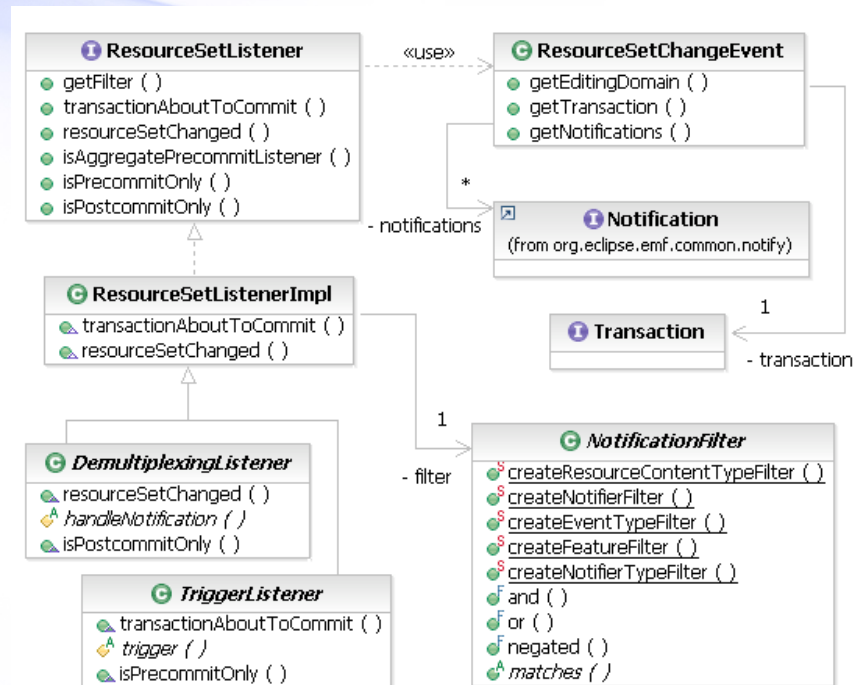
Change Listeners

- EMF provides an Adapter mechanism to notify listeners when objects change
 - In a transactional environment, though, we can end up reacting to changes only to find that they are reverted when a transaction rolls back
- Enter the ResourceSetListener
 - Post-commit event notifies a listener of all of the changes, in a single batch, that were committed by a transaction
 - If a transaction rolls back, no event is sent because there were no changes
 - There are exceptions for changes that are not (and need not be) undone, such as resource loading and proxy resolution



Change Listeners

- `ResourceSetChangeEvent` provides the changes that occurred
 - Transaction additionally has a `ChangeDescription` summarizing the changes
- Listeners can declare filters to receive only events of interest to them
- `ResourceSetListenerImpl` is a convenient base class providing no-ops for the listener call-backs



Change Listeners

- Resource set listeners are added to the transactional editing domain
- Listeners can be registered statically against an editing domain ID
 - Ensures that the listener is attached as soon as the domain comes into being
 - Resolves the problem of timing the addition of listeners

Post-Commit Listeners

- A post-commit listener just overrides `ResourceSetListenerImpl.resourceSetChanged()`
 - `DemultiplexingListener` implements this by dispatching the notifications one by one to the `handleNotification()` method

Post-Commit Listeners

- Advantages of the ResourceSetChangedEvent include:
 - Listeners know that the changes are permanent
 - Notifications can be processed efficiently as an aggregate
 - Don't need to worry about dependency on "future" changes
 - No further changes can occur while the change event is being dispatched
 - Listeners are invoked in read-only transactions, so that they can safely read the resource set while analyzing the changes
- Listeners need to be aware that notifications are delayed relative to the timing of the changes
 - Notifications are only received after all changes are complete
 - Any given notification may not correspond to the current state of the resource set, depending on subsequent changes

Pre-Commit Listeners

- Before a transaction closes, pre-commit listeners are notified of the changes performed
 - Listeners can provide additional changes, in the form of commands, to be appended to the transaction
 - As with post-commit listeners, the pre-commit listener is invoked in a read-only transaction, so it does not make changes “directly”
 - These commands implement proactive model integrity, as do triggers in RDBMS. Hence the term “trigger command”
- Trigger commands are executed in a nested transaction
 - This procedure is recursive: the nested transaction also invokes pre-commit listeners when it commits



Pre-Commit Listeners

Pre-Commit Listeners

- The `TriggerListener` class is convenient for processing notifications one by one, where appropriate

Transaction Validation

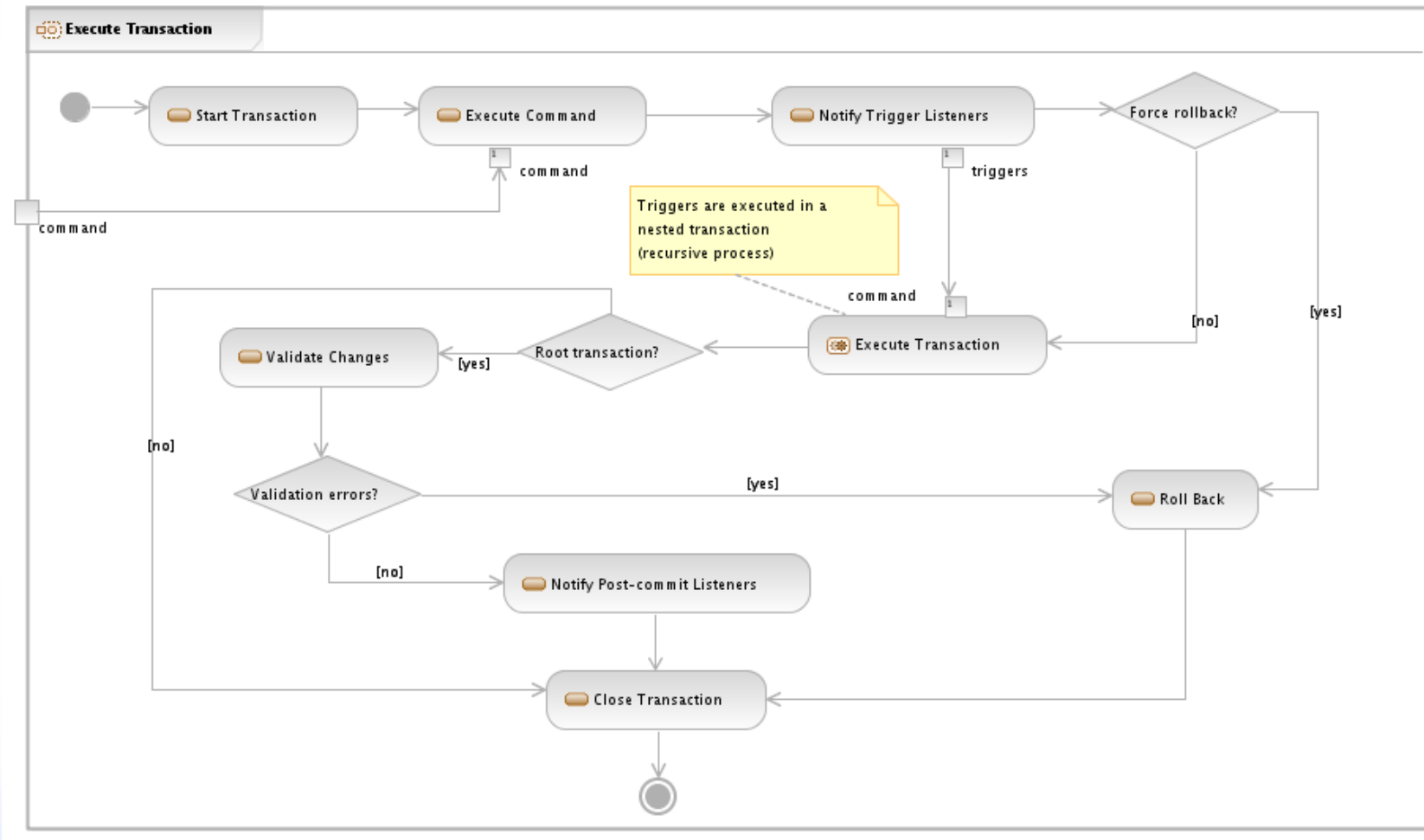
- When a read/write transaction commits, all of the changes that it performed are checked using the Validation Framework's live validation capability
 - If problems of error severity or worse are detected, then the transaction rolls back
- Pre-commit listeners can also force the transaction to roll back by throwing a `RollbackException`
 - If a pre-commit listener cannot construct the command that it requires to maintain integrity, then it should roll back

Transaction Nesting

- Both read-only and read-write transactions can nest to any depth
- Post-commit events are sent only when the root transaction commits
 - Because even after a nested transaction has committed, it can be rolled back if its parent (or some ancestor) rolls back
- Pre-commit events are sent at every level of nesting
 - Because a parent transaction may assume data integrity conditions guaranteed by triggers when it resumes
- Validation is performed on all changes when a root transaction commits
 - Because triggers must be invoked first, in nested transactions

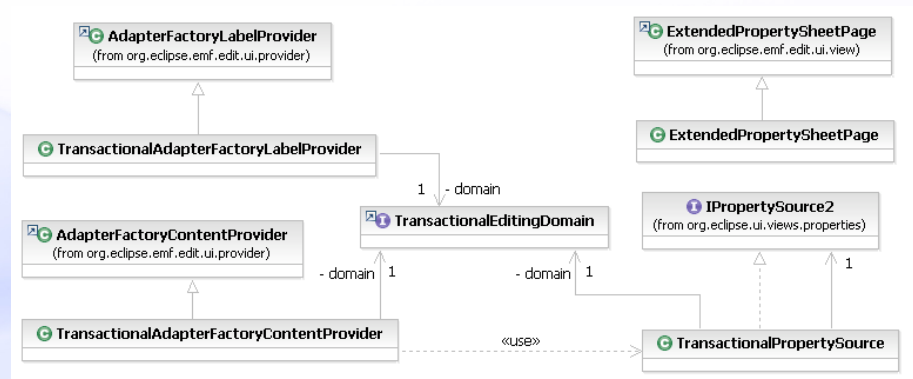
Transaction Nesting

- Nested transactions inherit options from their parents
 - The standard options cannot be disinherited. e.g., if a parent transaction does not send post-commit notifications, then none of its descendents will, either, even if they explicitly specify `Boolean.FALSE` for that option
- Nested transactions can, however, apply more options than their parents
 - e.g., a child transaction can disable notifications. When its parent commits, the changes that it reports will simply exclude any that occurred during the execution of the child
- The inheritance of custom options in an extension of the transaction API is defined by that extension



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



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

Query Statements

- SELECT statements filter the objects provided by a FROM clause according to the conditions specified in the WHERE clause
- SELECTs are IEObjectSources, so they can be used in FROM clauses to nest queries
- UPDATE statements use a SET clause to update the objects provided by the FROM clause (filtered, of course, by the WHERE clause)

The FROM Clause

- Uses EMF's tree iterators to walk the objects being queried
- Optionally specifies an `EObjectCondition` filter
- Search scope is encapsulated in an `IEObjectSource`
 - provides objects via an iterator. The `FROM` descends into the contents of these objects if it is a hierarchical `IteratorKind`

The WHERE Clause

- The WHERE clause specifies a single filter condition
- This filter condition can be arbitrarily complex
- Filters can be combined in innumerable ways using the common boolean operators
- The IN filter detects whether an object is an element of a supplied set (as in SQL)

The WHERE Clause

- The framework provides a condition on the model type of objects
- Can also filter for objects that are identical to some target (`EObjectInstanceCondition`)

The WHERE Clause

- The framework provides conditions that filter objects based on values of their features
- In particular, the `EObjectReferencerCondition` filters for cross-references to a particular object

The WHERE Clause: Condition Policies

- For multi-valued structural features, the ConditionPolicy determines whether a Condition must match all values or just any value
 - Correspond to the relational \forall (for-all) and \exists (exists) quantifiers

The WHERE Clause: Prune Handlers

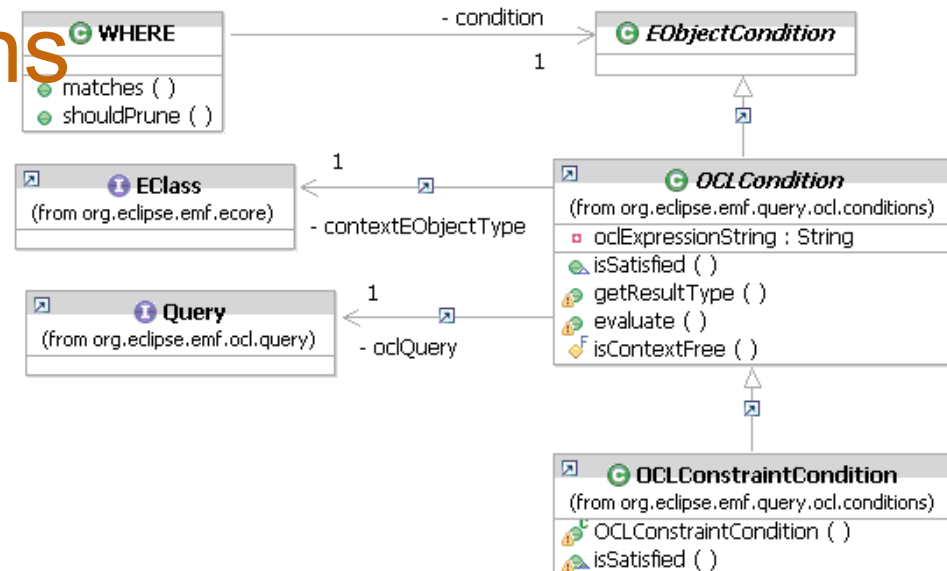
- Just as with EMF's tree iterators, a query's iteration over its scope can be *pruned*
- Any `EObjectCondition` can have a `PruneHandler` that determines whether the search tree can be pruned
 - This allows the query to skip over entire sub-trees when a condition knows that it will never be satisfied for any of the contents of the current object
- The default prune handler in most cases is `NEVER` which, as the name implies, never prunes

The WHERE Clause: Other Conditions

- The framework includes a variety of conditions for working with primitive-valued EAttributes
 - Including strings, booleans, and numbers of all kinds
- Adapters convert inputs to the required data type
 - Default implementations simply cast, assuming that the values already conform
 - Can be customized to convert values by whatever means is appropriate

OCL Conditions

- OCL can be used to specify WHERE clause conditions
- Only available when the OCL component of MDT is installed
- An `OCLConstraintCondition` specifies a boolean-valued expression (i.e., a constraint) that selects those elements for which the expression is true
- OCL expressions can be contextful or context-free



The UPDATE Statement

- The UPDATE statement behaves much like a SELECT, except that it passes its result objects through the client-supplied SET clause
- The result of the UPDATE is the subset of the selected objects for which the SET clause returned true (indicating that they were, in fact, modified)

SELECT Query in Action

- Queries can be used for anything from simple search functions to complex structural analysis
- They can even be used to implement validation constraints!

SELECT Query with OCL Condition

- Context-free OCL conditions are applied to any element on which they can be parsed



UPDATE Query in Action

Summary

- In this module we explored



zeligsoft

Uniform Resource Identifier (URI)

- A formatted string that serves as an identifier for a resource
- Syntax:
 - Generic
`[scheme:]scheme-specific-part[#fragment]`
 - Hierarchical
`[scheme:] [//authority] [path] [?query] [#fragment]`
- Used in EMF to identify a resource, an object in a resource, or an Ecore package (namespace URI)



- 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?`

- 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)

Cross-Resource Containment

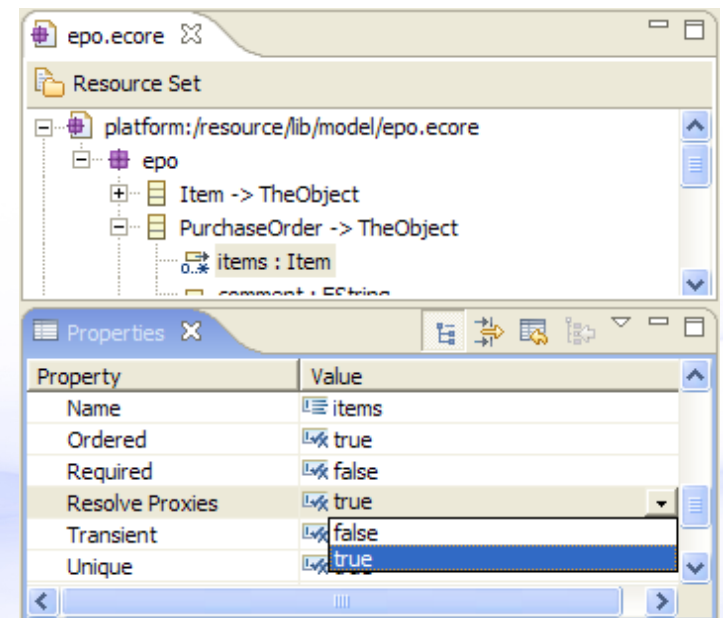
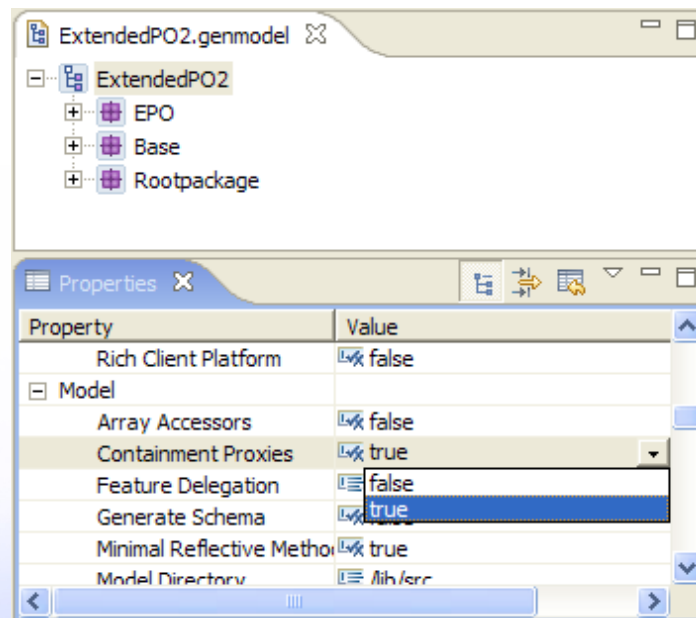
- Allows an object hierarchy to be persisted across multiple resources
 - `eObject.eResource()` may be different from `eObject.eContainer().eResource()`
- Must be explicitly enabled
 - The containment reference has to be set to resolve proxies
 - Also, on generated models, the value of the “Containment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

Enabling Cross-Resource Containment

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

Dynamic Model

Generated Model



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
 - Feature ID = 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

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
 - Prefix for package-related class names
 - 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