zeligsoft

Eclipse Transformation Training


zeligsoft
Introductions


zeligsoft
Introduction - Instructors
Workshop leader
Toby McClean
Principal Technology Specialist
Modeling, Domain Specific Modeling and Languages
Model transformations and code generation
Component based development and standards
Facilitators
Mark Hermeling (week 1)
Director, Product Technology
Tim McGuire (week 2)
Senior Software Engineering Specialist
zeligsoft


zeligsoft
Introductions - Participants
Briefly, a little about you
Name
Department
What is your interest in Eclipse and the Rational Modeling Platform
Experience
With Eclipse
With RSA, RSD or RSM
How do you expect to apply what you learn this week
zeligsoft

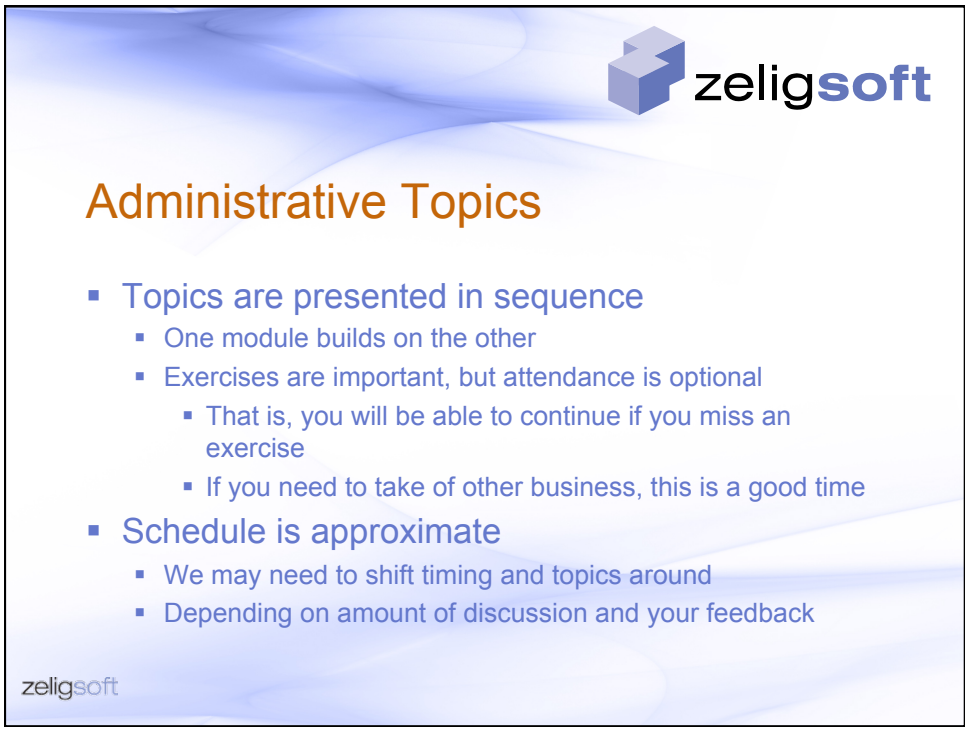

zeligsoft
Workshop Goals
Workshop contents
Provide a high-level overview of Eclipse technologies
Related to modeling, domain specialization and transformation
Give you the chance to gain hands-on experience
With programmatically creating and transforming models
The second bullet requires significant detail
What you will be able to do after the workshop
You will understand the possibilities of Eclipse and RSA-RTE
And have a good idea how to extend this environment for your day-to-day activities
zeligsoft


zeligsoft
Workshop Agenda
Monday
Architecture overview
Eclipse modeling overview
Tuesday
EMF in depth
UML in depth
Wednesday
Transformation 1
Thursday
Transformation 2
Validation
zeligsoft


zeligsoft
Schedule
9:00-10:30
10:45-12:00
13:00-15:00
15:15-17:00
zeligsoft


zeligsoft
Workshop Make-Up
Half of this workshop will be slide material
Half of it will be exercises and discussions
Typical flow
Presentation of module
Discussion on how to apply to Ericsson's environment
Exercise introduction
Exercise
Wrap-up of topic
This is a custom-built course for Ericsson
We encourage discussion and exploration
zeligsoft


zeligsoft
Workshop Materials
Workbook with slides
Workbook with exercises
VMWare image with Eclipse pre-installed
Solution files
zeligsoft


zeligsoft
Administrative Topics
Topics are presented in sequence
One module builds on the other
Exercises are important, but attendance is optional
That is, you will be able to continue if you miss an exercise
If you need to take of other business, this is a good time
Schedule is approximate
We may need to shift timing and topics around
Depending on amount of discussion and your feedback
zeligsoft


zeligsoft
The Eclipse Project
An introduction to the Eclipse Project


zeligsoft
Overview
In this module we will explore the Eclipse Project including the different aspects of the Eclipse Workbench.
We will also explore how to develop and deploy plug-ins to extend the Eclipse Workbench.
Some content taken from
Developing Plug-ins for Eclipse (http://www.eclipse.org/resources/)
by Dwight Deugo and Nesa Matic


zeligsoft
Goals
Following the completion of this module and its exercises you will
Have an understanding of the Eclipse Workbench
Know some of the terminology used within the Eclipse Workbench
Have an understanding of the Eclipse architecture
Have an understanding of how Eclipse is extended
Be able to develop and deploy a basic plug-in


zeligsoft
Agenda
What is Eclipse?
The Eclipse Workbench
The Eclipse Architecture
Extending the Workbench


zeligsoft
What is Eclipse?
Eclipse is an open source project
http://www.eclipse.org
Consortium of companies, including IBM
Launched in November, 2001
Designed to help developers with specific development tasks

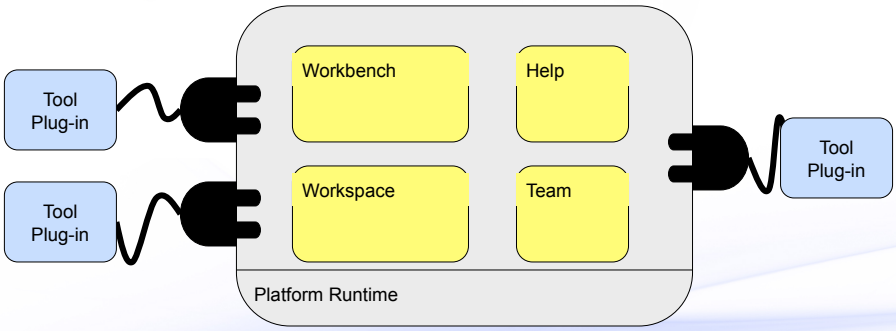

zeligsoft
Projects
On topic
Business Intelligence and Reporting Tools (BIRT)
Device Software Development Platform
Eclipse Project
Eclipse Modeling Project
Off topic
Data Tools Platform
Eclipse RT
SOA Tools
Eclipse Technology Project
Tools Project
Test and Performance Tools Platform Project
Eclipse Web Tools Platform Project
http://www.eclipse.org/projects/listofprojects.php


zeligsoft
Brief History of Eclipse
1994
VisualAge for Smalltalk
1996
VisualAge for Java
1996-2001
VisualAge Micro Edition
2001
Eclipse Project


zeligsoft
The Motivation Behind Eclipse
Support for the construction of application development tools
Support for the development of GUI and non-GUI application development
Support for multiple content types
Java, HTML, C, XML
Facility the integration of tools
Cross-platform support

zeligsoft

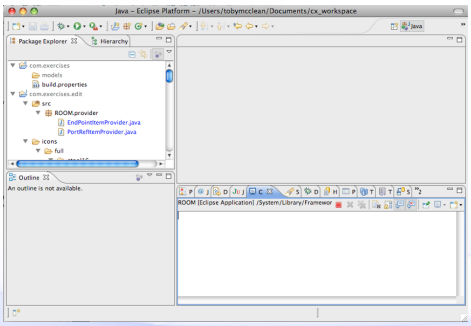
Eclipse Architecture



zeligsoft

Workbench

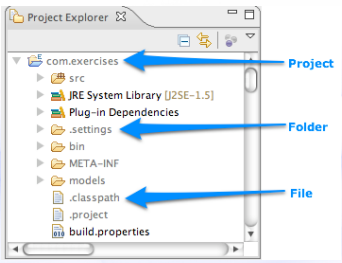
- The desktop development environment
  - Tools for resource management
  - Common of navigating through resources
- Multiple workbenches can be opened at the same time
  - Different workspaces



zeligsoft

Workspace

- Represents the user data
- A set of resources
  - Projects
    - Collection of files and folders
  - Folders
    - Contain other folders or files
  - Files



zeligsoft

Help

- For the creation and publishing of documentation
- Supports both
  - User guides
  - Programmer guides
- Created using
  - HTML for content
  - XML for navigation
- Support for context sensitive help

Toby McClean, Zeligsoft, 2008


zeligsoft
Team
Provides support for
Versioning
Configuration management
Integration with team repository
Allows team repository provider to hook into environment
Team repository providers specify how to intervene with resources
Has optimistic and pessimistic locking support


zeligsoft
How is Eclipse Used?
As an Integrated Development Environment
Supports the manipulation of multiple content types
Used for specifying and designing applications
Requirements, models, documentation, etc.
Used for writing code
Java, C, C++, C#, Python, ...
As a product base
Supported through plug-in architecture and customizations


zeligsoft
Eclipse as an IDE
Java Development Tooling
Editors, compiler integration, ant integration, debugger integration, etc.
C/C++ Development Tooling
Editors, compiler integration, make integration, debugger integration, etc.
Dynamic languages
Editors, interpreter and compiler integration, debugger integration, etc.
Modeling projects


zeligsoft
Eclipse as a Product Base
Eclipse can be used a Java product base
Its flexible architecture can be used as product framework
Reuse plug-in architecture
Create new plug-ins
Customize the environment
Support for branding
Rich client platform support


zeligsoft
Rich Client Platform
A minimal set of Eclipse plug-ins necessary for building a product
Control over resource model
Control over look and feel
But still capable of leveraging existing plug-ins


zeligsoft
Summary
In this module we explored
Eclipse, its background and the components that form its foundation
Eclipse use cases

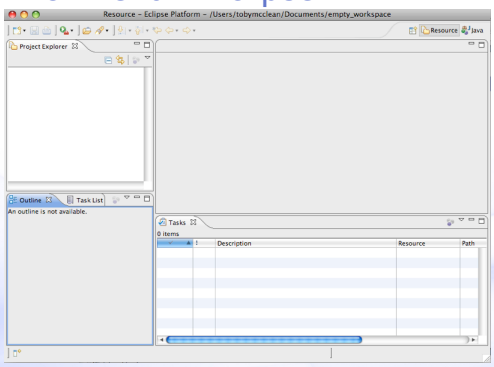

zeligsoft
Agenda
What is Eclipse?
The Eclipse Workbench
The Eclipse Architecture
Extending the Workbench


zeligsoft
The Eclipse Project
The Eclipse Workbench

zeligsoft

What is the Workbench?

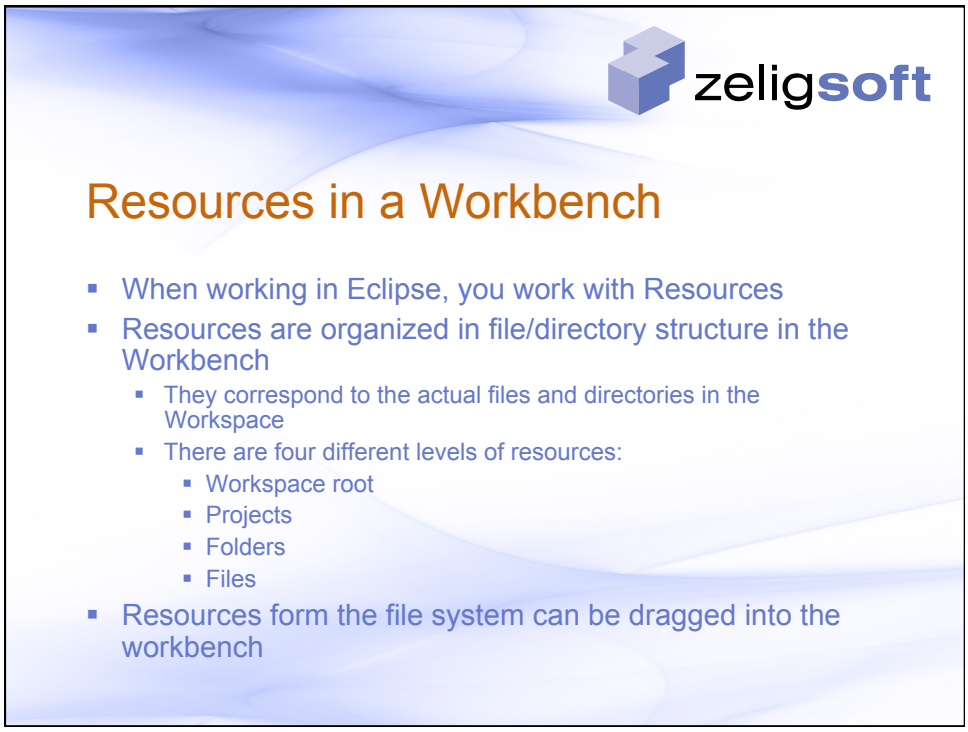
- The working environment in Eclipse



zeligsoft

Multiple Workbench Instances

- Instance of Workbench comes up when Eclipse is launched
- It is possible to open another window for the Workbench
  - Window → New Window
  - This opens up a new Workbench window
  - Can have different perspectives open in the different windows
  - Same result is not achieved by launching Eclipse twice

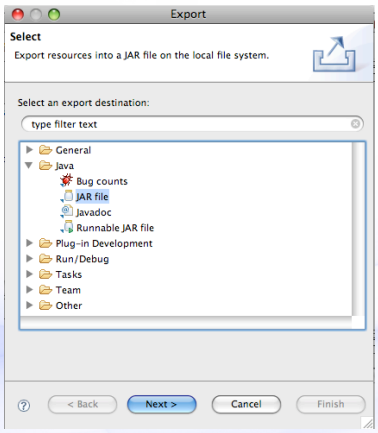


Toby McClean, Zeligsoft, 2008

zeligsoft

Exporting Resources

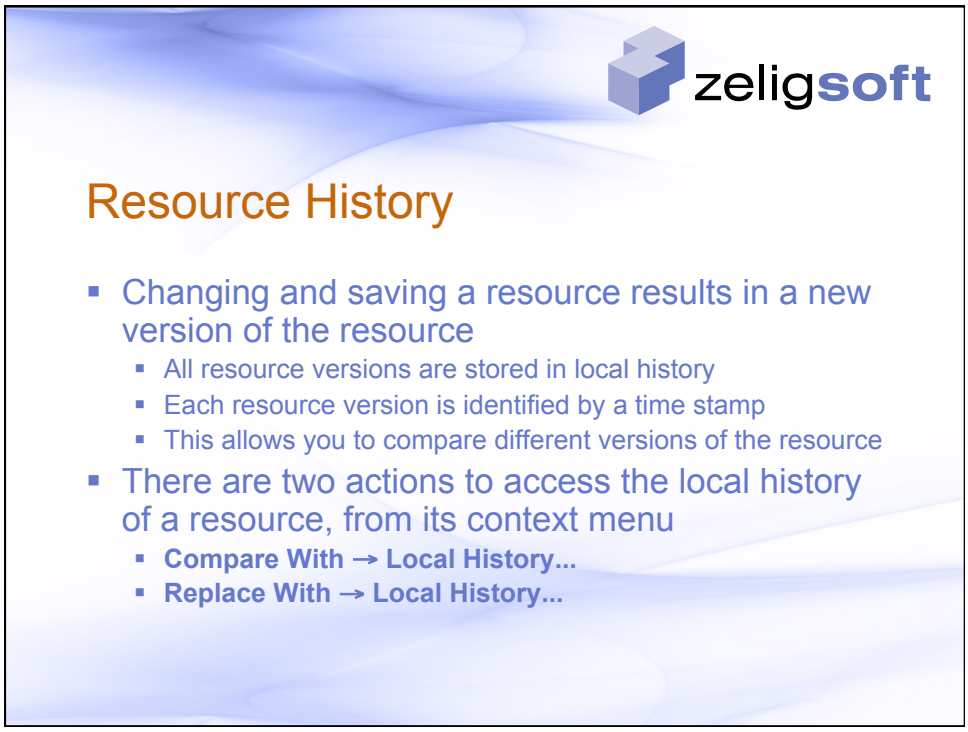
- Available through
  - Menu option **File → Export...**
  - Right-click menu **Export...** option on a Resource in the Workbench

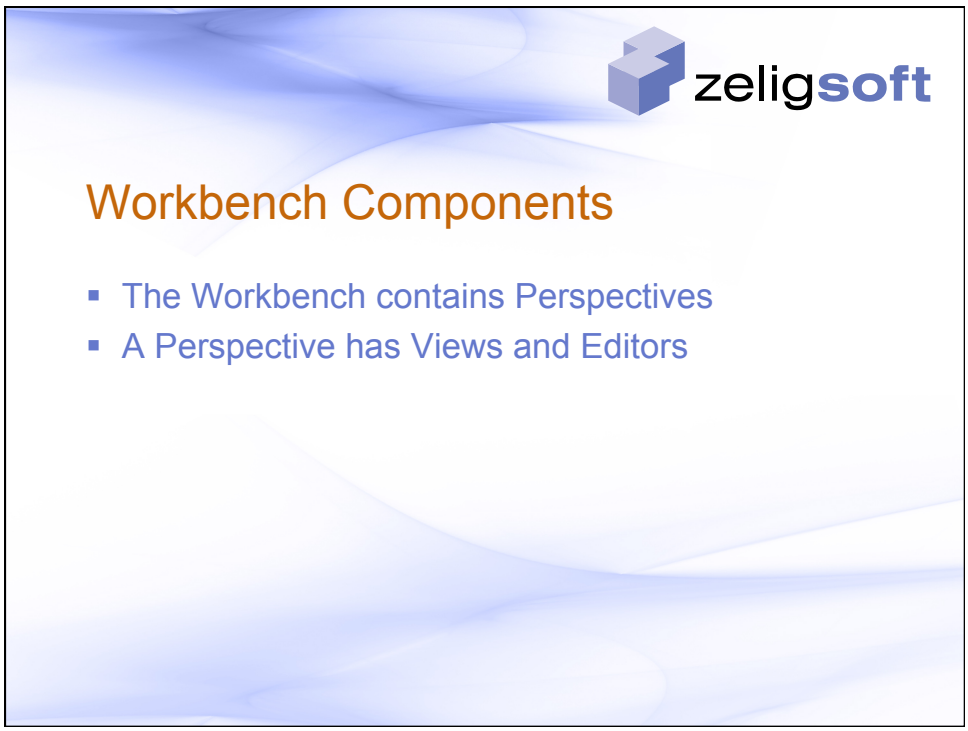


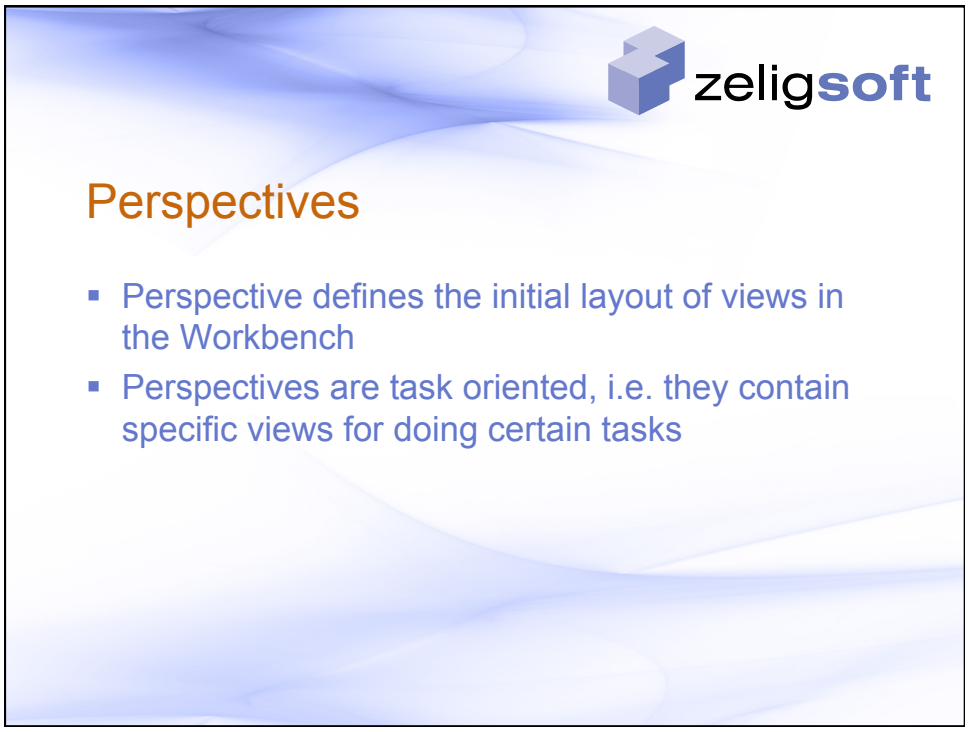
zeligsoft

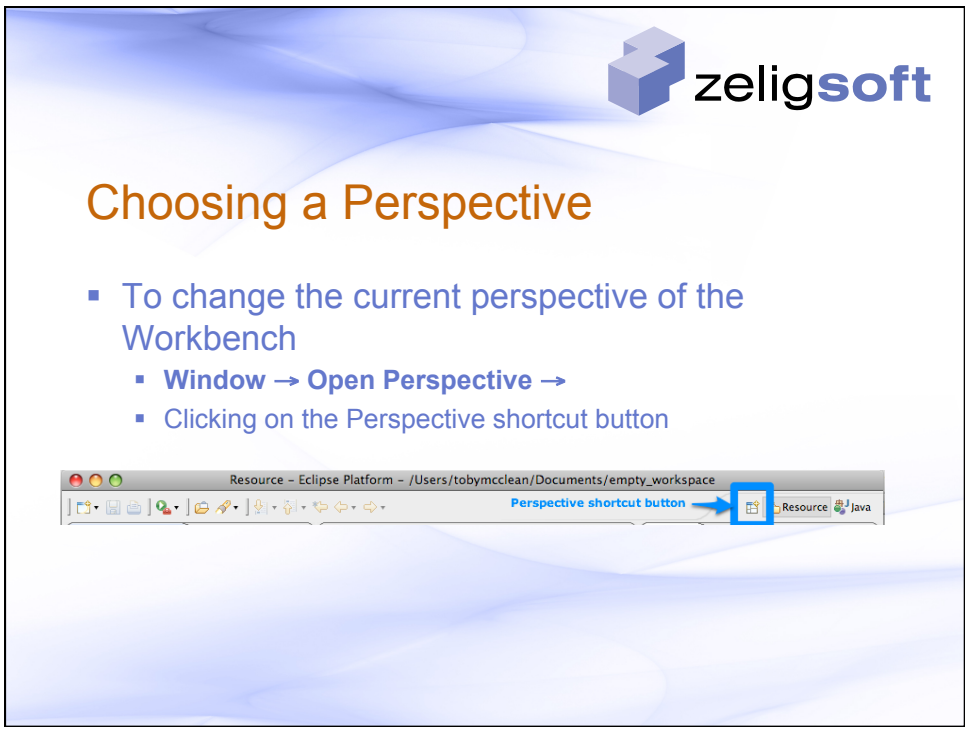
Refreshing Workbench

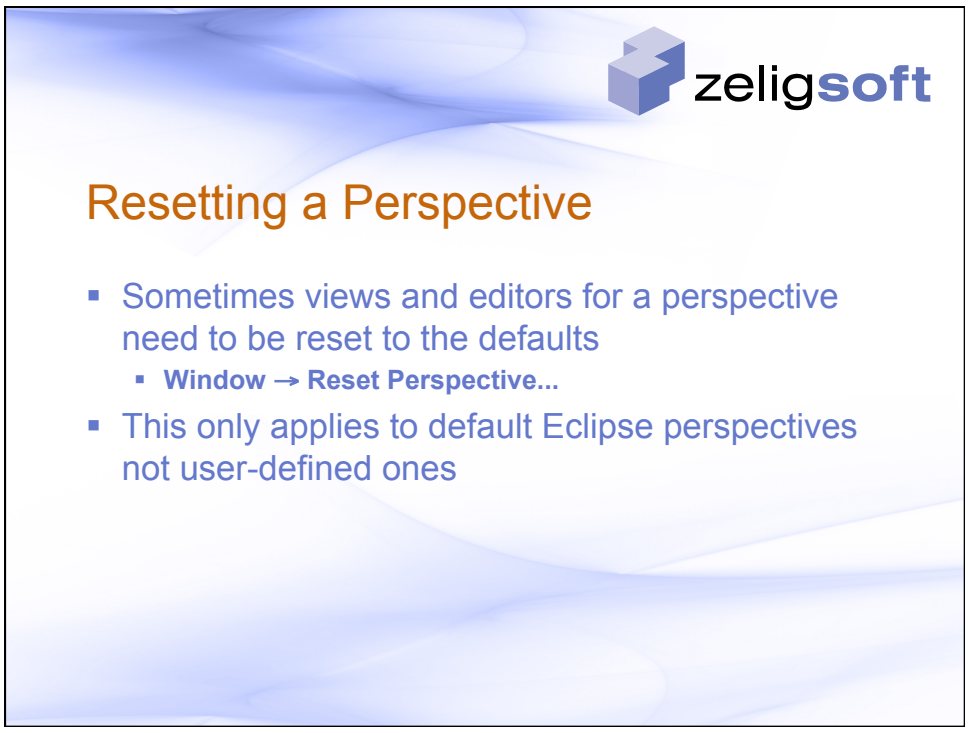
- Used for refreshing resources that change in the Workspace outside the workbench
- For example, if a file added to a directory in the Workspace
  - Select the project or directory in the workbench
  - Choose **Refresh** from its context menu
  - Alternatively you can press the **F5 button** on your keyboard
- Best practice
  - Work with the resources from within the Workbench when possible


zeligsoft
Resource History
Changing and saving a resource results in a new version of the resource
All resource versions are stored in local history
Each resource version is identified by a time stamp
This allows you to compare different versions of the resource
There are two actions to access the local history of a resource, from its context menu
Compare With → Local History...
Replace With → Local History...

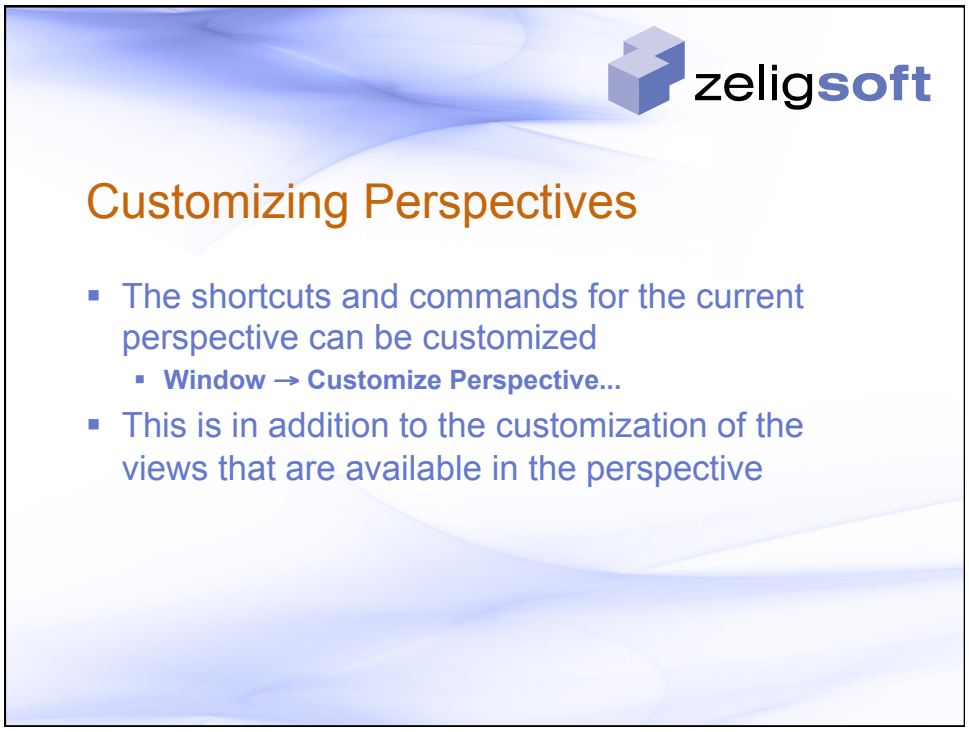

zeligsoft
Workbench Components
The Workbench contains Perspectives
A Perspective has Views and Editors

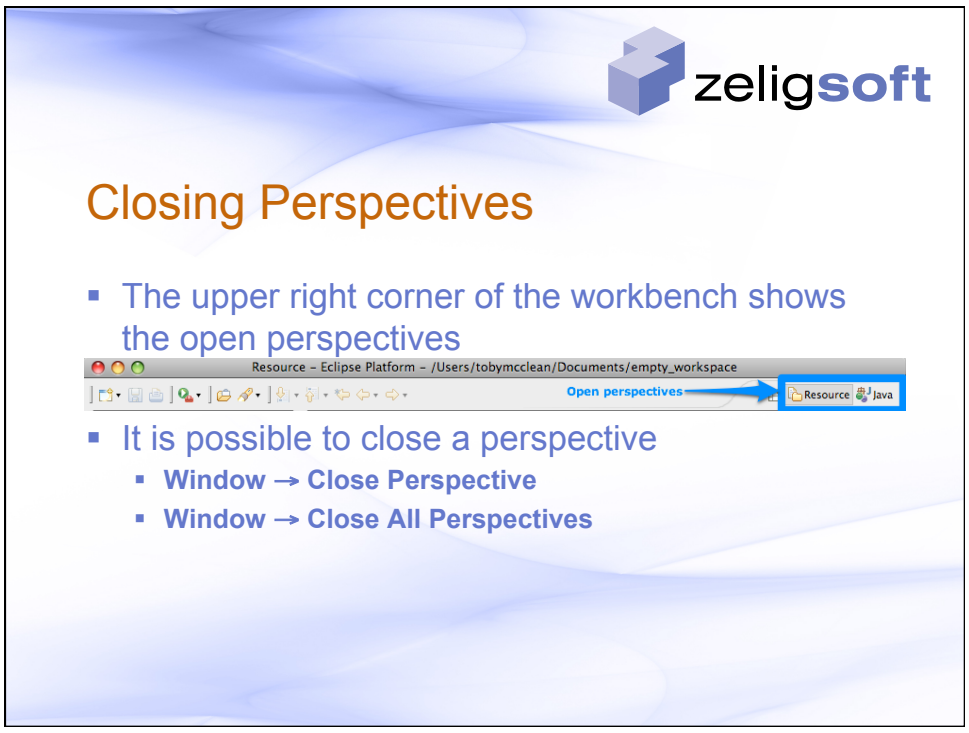

zeligsoft
Perspectives
Perspective defines the initial layout of views in the Workbench
Perspectives are task oriented, i.e. they contain specific views for doing certain tasks

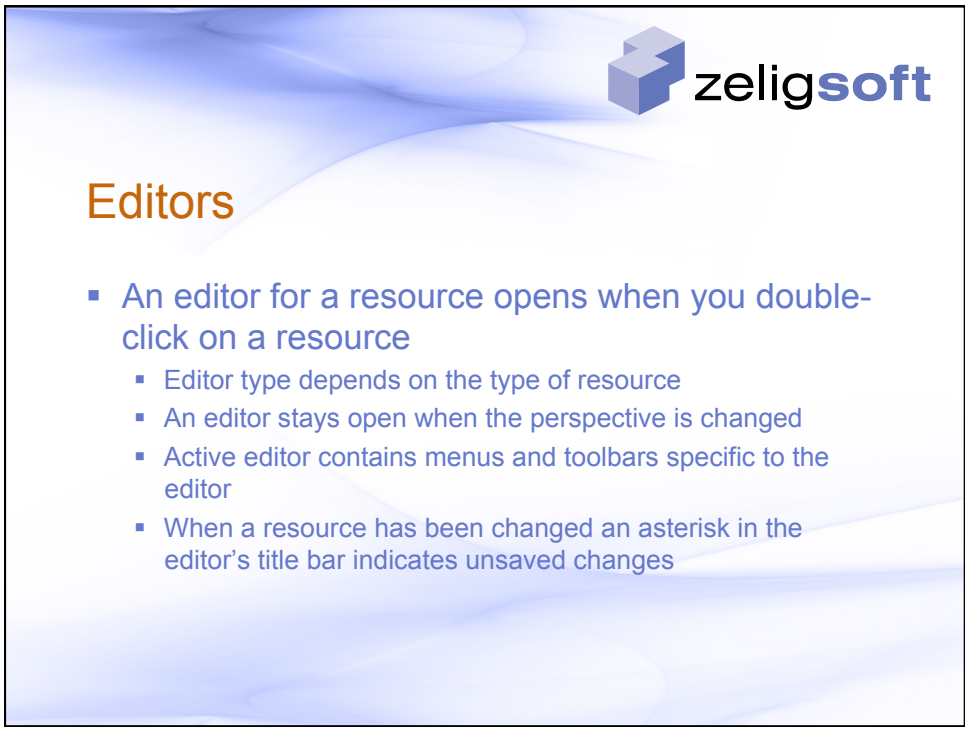

zeligsoft
Choosing a Perspective
To change the current perspective of the Workbench
Window → Open Perspective →
Clicking on the Perspective shortcut button
Resource – Eclipse Platform – /Users/tobymcclean/Documents/empty_workspace
Perspective shortcut button
Resource
Java

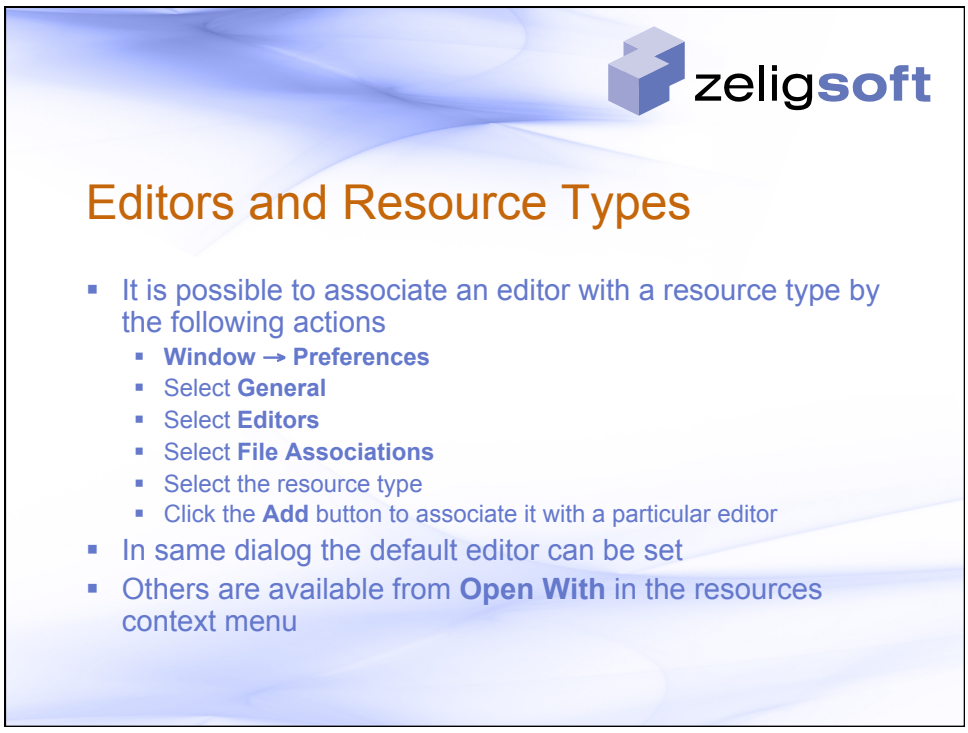


Toby McClean, Zeligsoft, 2008


zeligsoft
Customizing Perspectives
The shortcuts and commands for the current perspective can be customized
Window → Customize Perspective...
This is in addition to the customization of the views that are available in the perspective

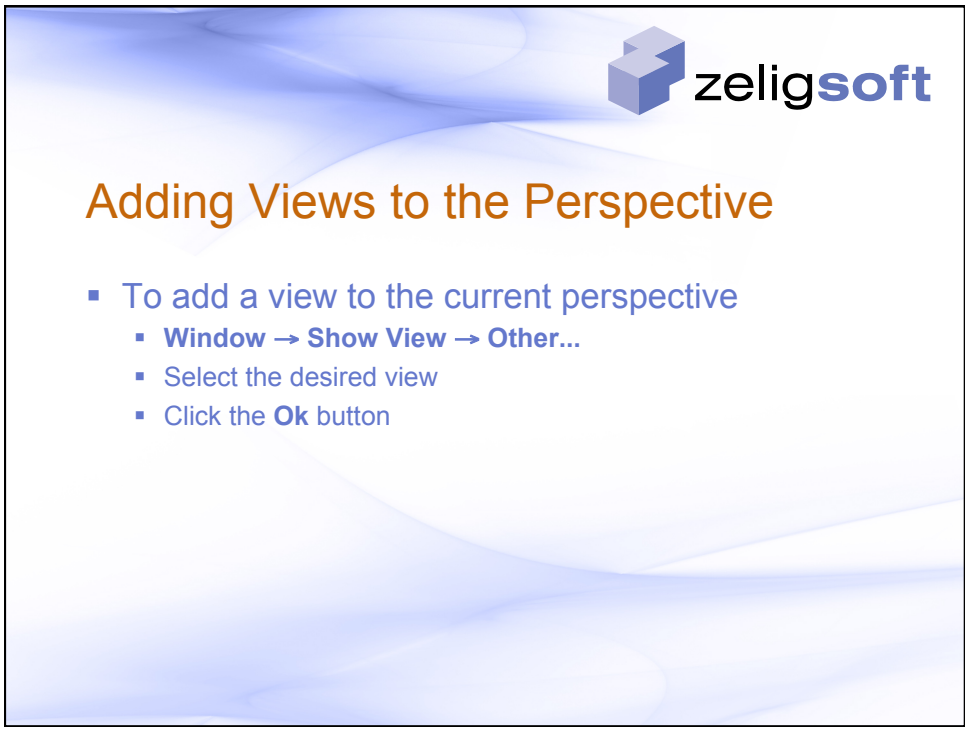

zeligsoft
Closing Perspectives
The upper right corner of the workbench shows the open perspectives
Resource – Eclipse Platform – /Users/tobymcclean/Documents/empty_workspace
Open perspectives
Resource
Java
It is possible to close a perspective
Window → Close Perspective
Window → Close All Perspectives


zeligsoft
Editors
An editor for a resource opens when you double-click on a resource
Editor type depends on the type of resource
An editor stays open when the perspective is changed
Active editor contains menus and toolbars specific to the editor
When a resource has been changed an asterisk in the editor's title bar indicates unsaved changes


zeligsoft
Editors and Resource Types
It is possible to associate an editor with a resource type by the following actions
Window → Preferences
Select General
Select Editors
Select File Associations
Select the resource type
Click the Add button to associate it with a particular editor
In same dialog the default editor can be set
Others are available from Open With in the resources context menu


zeligsoft
Views
The main purpose of a view is
Support editors
Provide alternative presentation and navigation
Views can have their own menus and toolbars
Items available in menus and toolbars are only available in that view


zeligsoft
More Views
Views can
Appear on their own
Appear stacked with other views
Layout of views can be changed by clicking on the title bar and moving views
Single views can be moved together with other views
Stacked views can be moved to be single views


zeligsoft
Adding Views to the Perspective
To add a view to the current perspective
Window → Show View → Other...
Select the desired view
Click the Ok button


zeligsoft
Stacked Views and Stacking Views
Stacked views appear in a notebook form
Each view is a page in the notebook
A view can be added to a stack by dragging into the area of the tab area of the 'notebook'
Similarly a view can be removed from a stack by dragging its tab away from the 'notebook'


zeligsoft
Fast Views
A fast view is hidden and can be quickly opened and closed
Created by
Dragging an open view to the shortcut bar
Selecting Fast View from the view's menu
A fast view is activated by clicking on its Fast View pop-up menu option
Bottom right of the workbench
Deactivates as soon as focus is given to another view or editor


zeligsoft
Summary
In this module we explored
Components of the Eclipse workbench
Perspectives;
Editors; and
Views


zeligsoft
Agenda
What is Eclipse?
The Eclipse Workbench
The Eclipse Architecture
Extending the Workbench


zeligsoft
The Eclipse Project
Eclipse Architecture

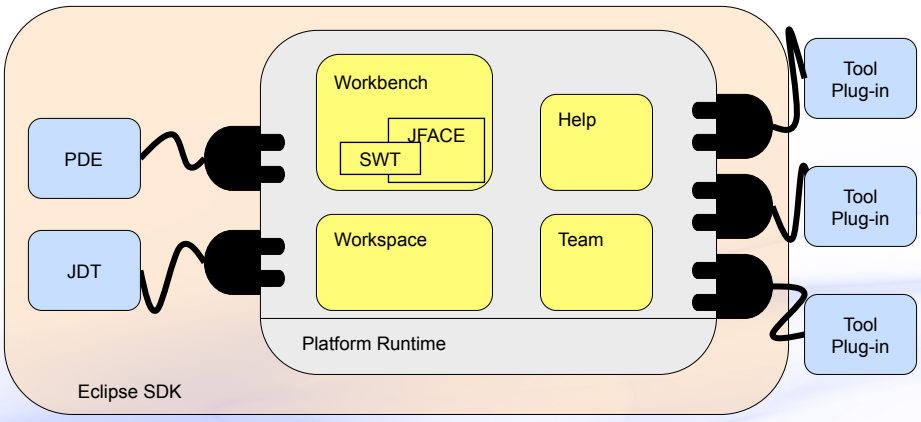
zeligsoft

Eclipse Architecture

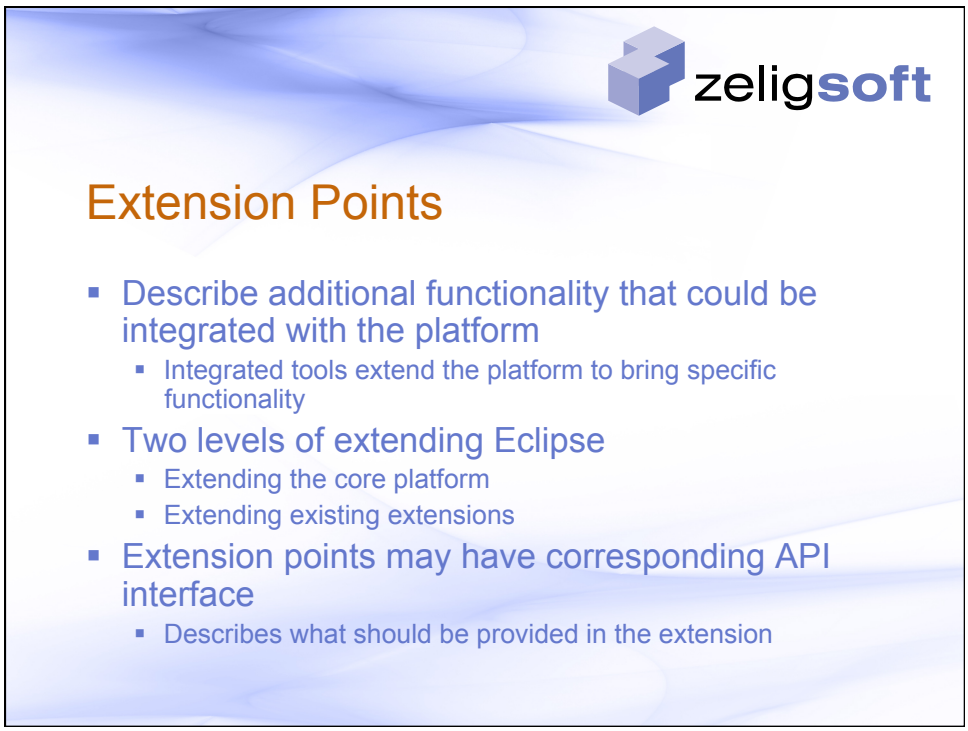
- Flexible, structured around
  - Extension points
  - Plug-ins
- Architecture allows for
  - Other tools to be used within the platform
  - Other tools to be extended
  - Integration between tools and the platform

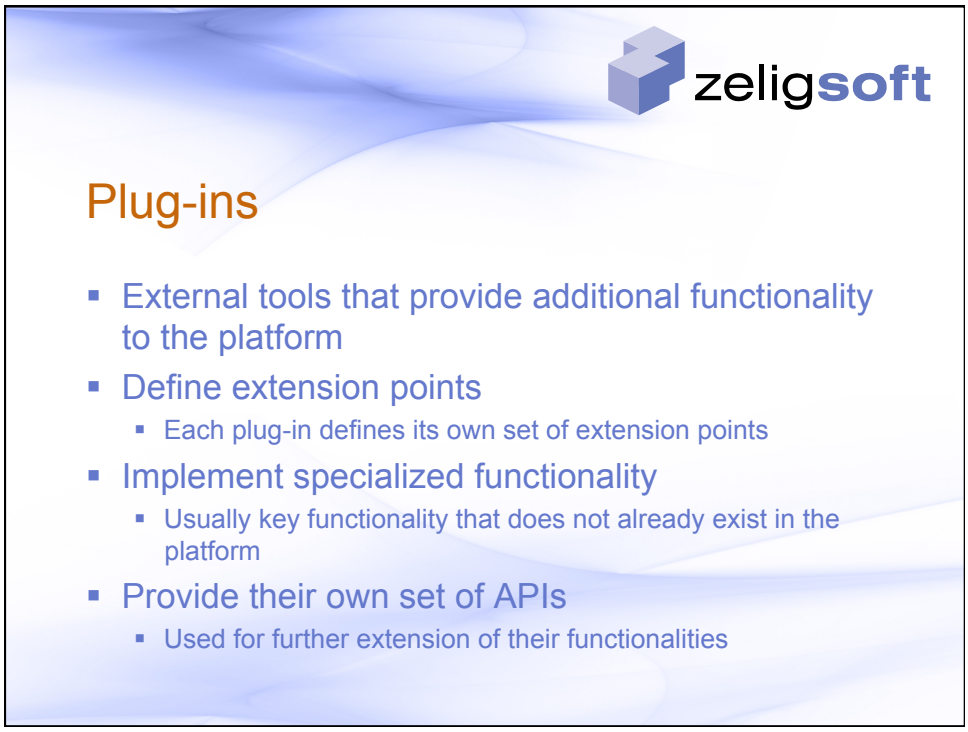
zeligsoft

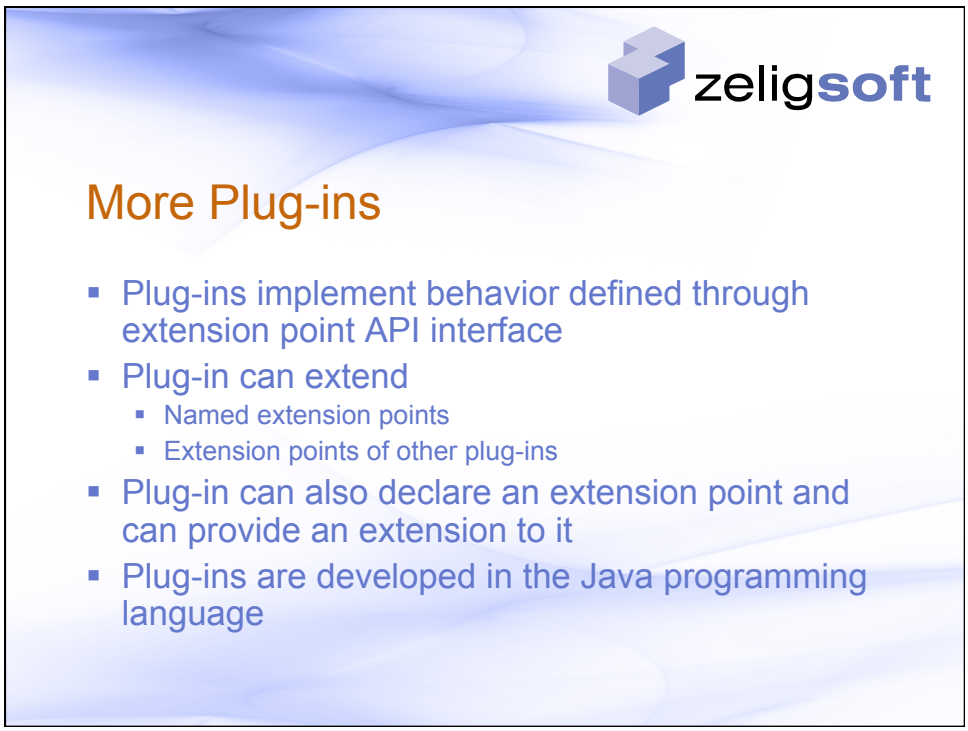
More Eclipse Architecture

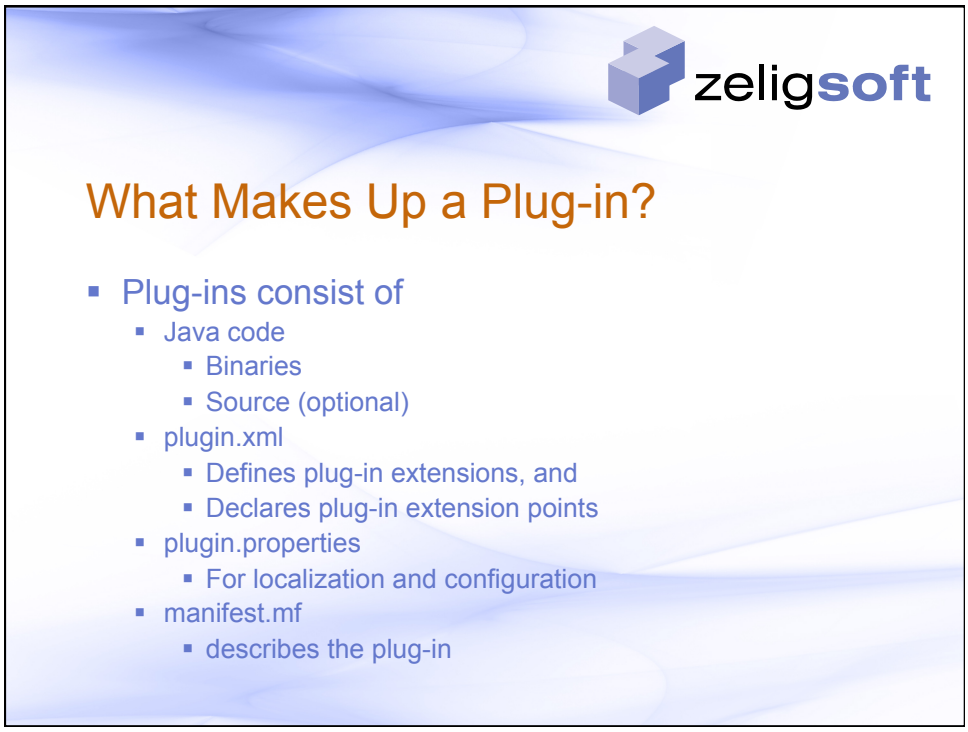


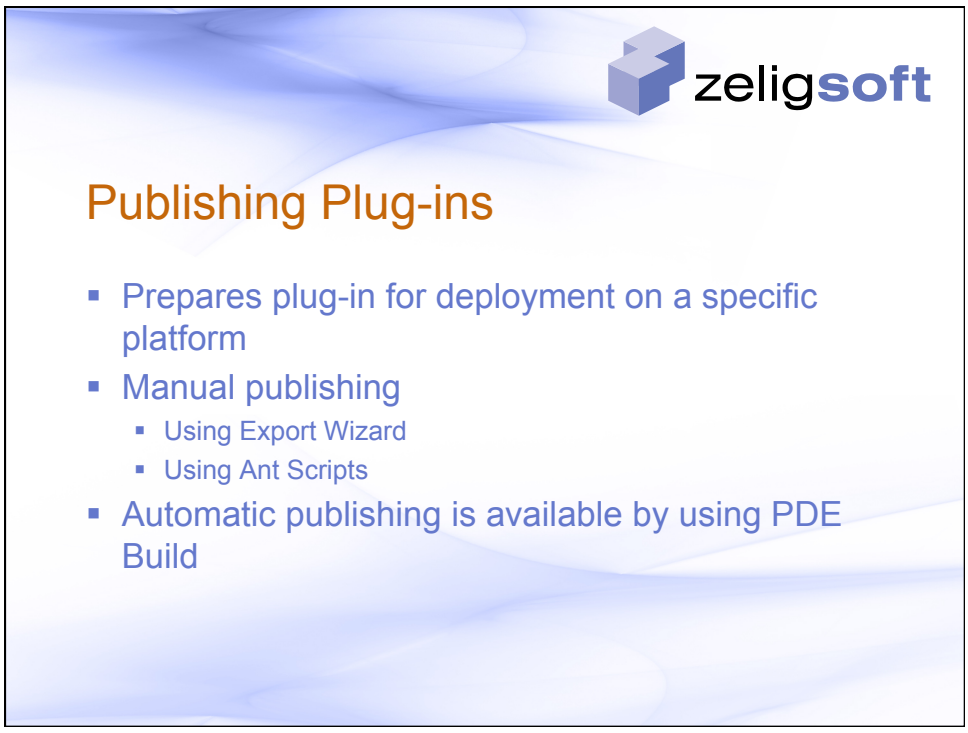

zeligsoft
Platform Runtime
Everything is a plug-in except the Platform Runtime
A small kernel that represents the base of the Platform
All other subsystems build up on the Platform Runtime following the rules of plug-ins
i.e. they are plug-ins themselves


zeligsoft
Extension Points
Describe additional functionality that could be integrated with the platform
Integrated tools extend the platform to bring specific functionality
Two levels of extending Eclipse
Extending the core platform
Extending existing extensions
Extension points may have corresponding API interface
Describes what should be provided in the extension


zeligsoft
Plug-ins
External tools that provide additional functionality to the platform
Define extension points
Each plug-in defines its own set of extension points
Implement specialized functionality
Usually key functionality that does not already exist in the platform
Provide their own set of APIs
Used for further extension of their functionalities


zeligsoft
More Plug-ins
Plug-ins implement behavior defined through extension point API interface
Plug-in can extend
Named extension points
Extension points of other plug-ins
Plug-in can also declare an extension point and can provide an extension to it
Plug-ins are developed in the Java programming language

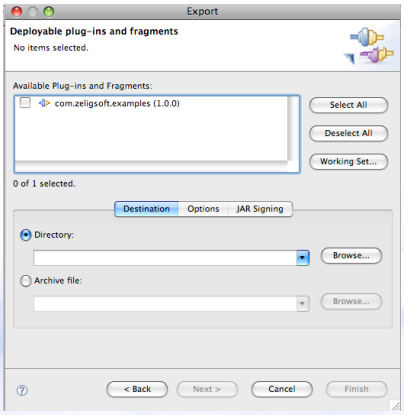

zeligsoft
What Makes Up a Plug-in?
Plug-ins consist of
Java code
Binaries
Source (optional)
plugin.xml
Defines plug-in extensions, and
Declares plug-in extension points
plugin.properties
For localization and configuration
manifest.mf
describes the plug-in


zeligsoft
Publishing Plug-ins
Prepares plug-in for deployment on a specific platform
Manual publishing
Using Export Wizard
Using Ant Scripts
Automatic publishing is available by using PDE Build

zeligsoft

Manually Publishing a Plug-in

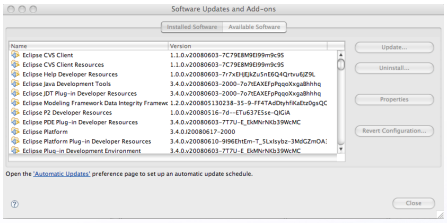
- A plug-in can be published manually by
  - **File → Export...**
  - Select Deployable plug-ins and fragments
  - Click **Next >** button
- Parameters configure how the plug-in published
- Can be saved as an Ant script



zeligsoft

Installing Plug-ins

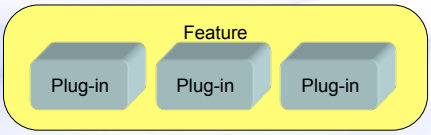
- Plug-ins are installed under \plugins\ directory of Eclipse installation
- Plug-ins can be installed from a update site
  - **Help → Software Updates..**
  - Typically as part of a feature
- Restart workbench



zeligsoft

Features

- An Eclipse feature is a collection of plug-ins
  - Represents smallest unit of separately downloadable and installable functionality
- Installed into \features\ directory of Eclipse installation
- Features can be installed from an update site



zeligsoft

Product

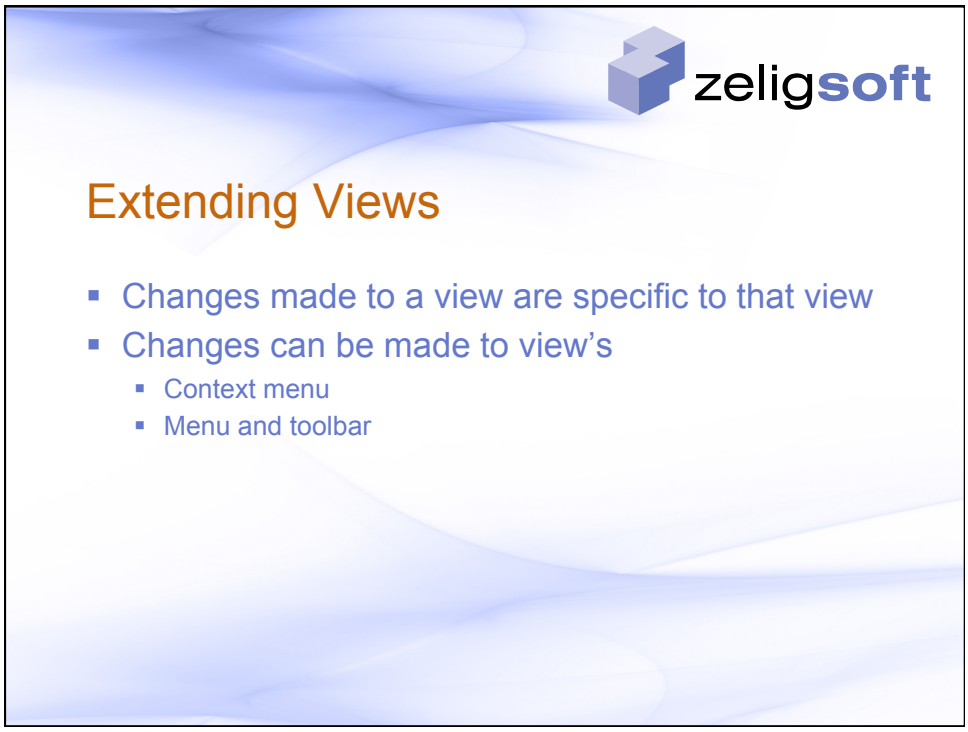
- Stand-alone application built on Eclipse platform
- Typically include the JRE and Eclipse platform
- Defined by using an extension point
- Configurable
  - Splash screen
  - Default preference values
  - Welcome pages
- Customized installation

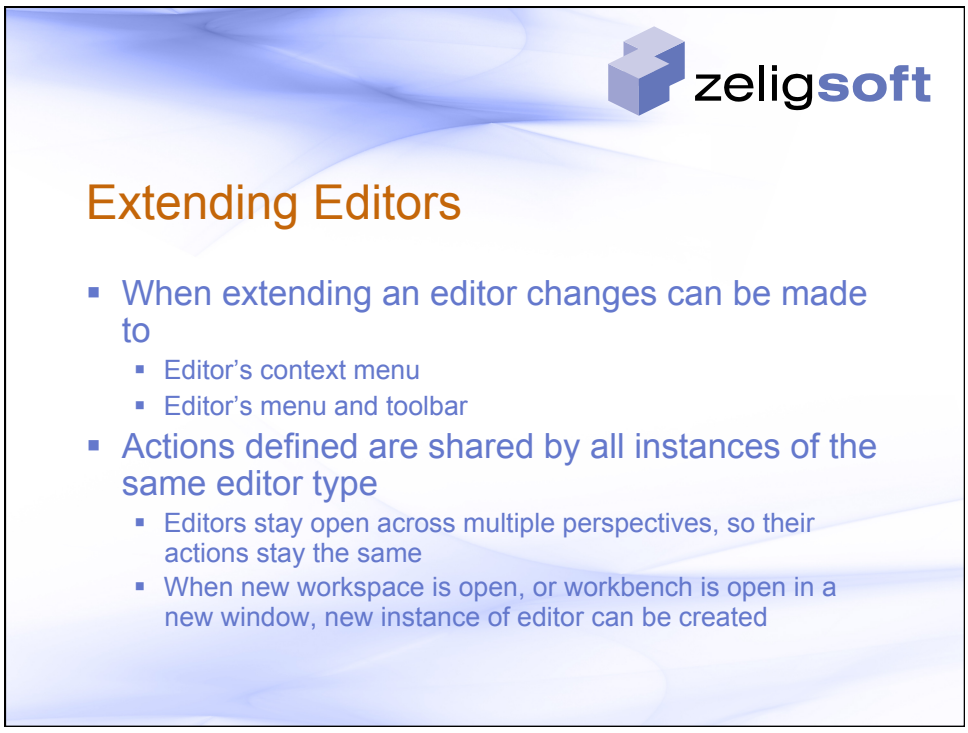

zeligsoft
Summary
In this module we explored
Eclipse architecture
Extension points
Plug-ins
Features
Products

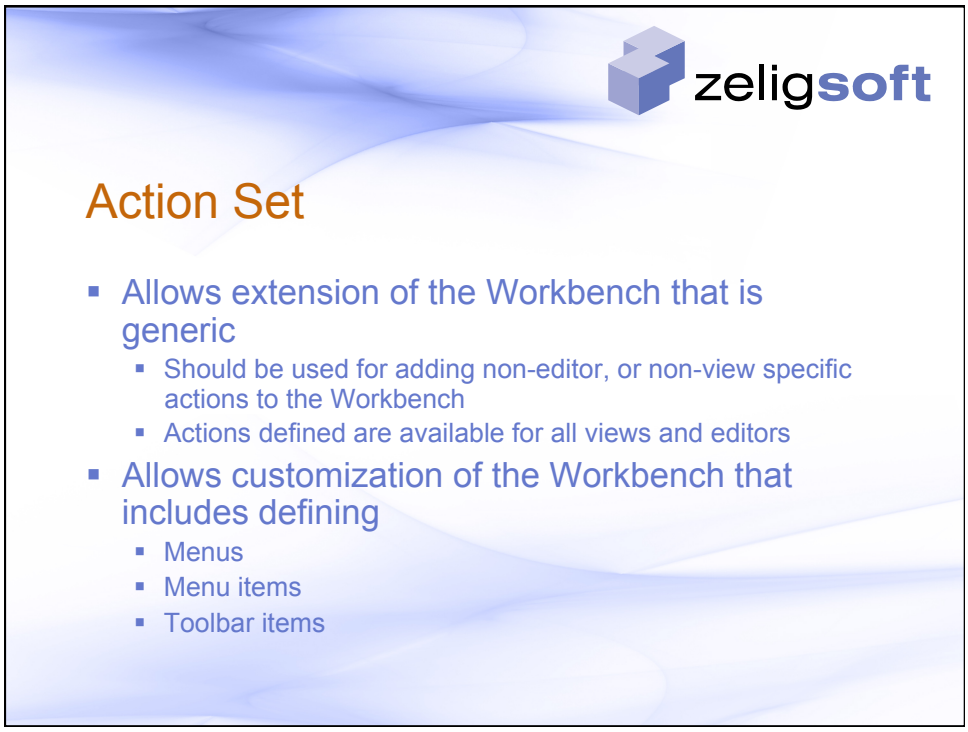

zeligsoft
Agenda
What is Eclipse?
The Eclipse Workbench
The Eclipse Architecture
Extending the Workbench

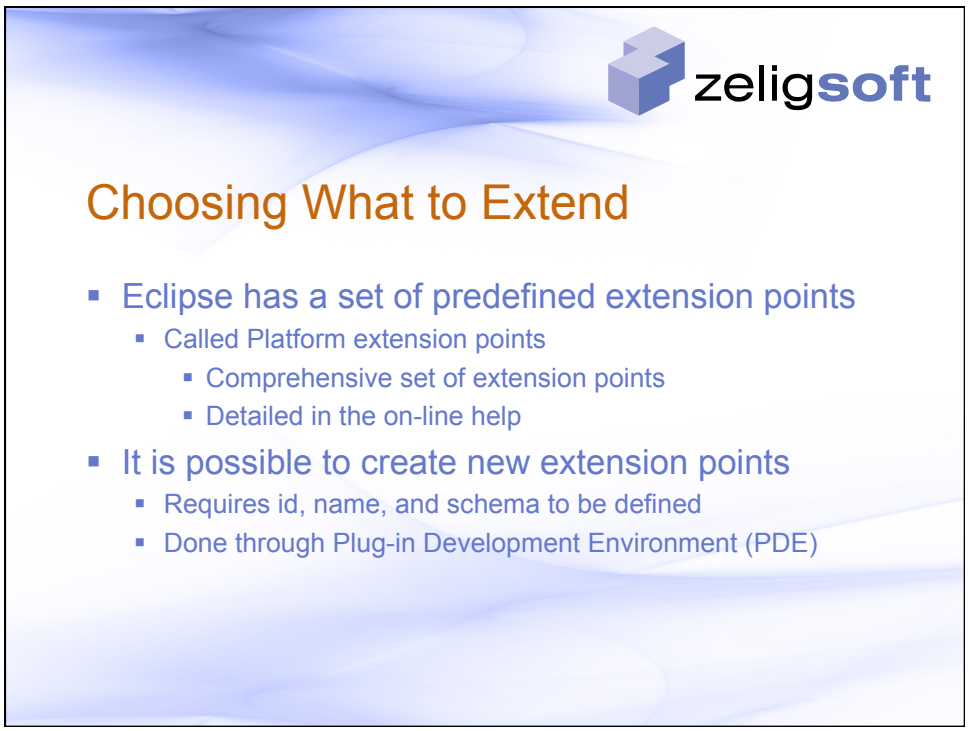

zeligsoft
The Eclipse Project
Plug-ins

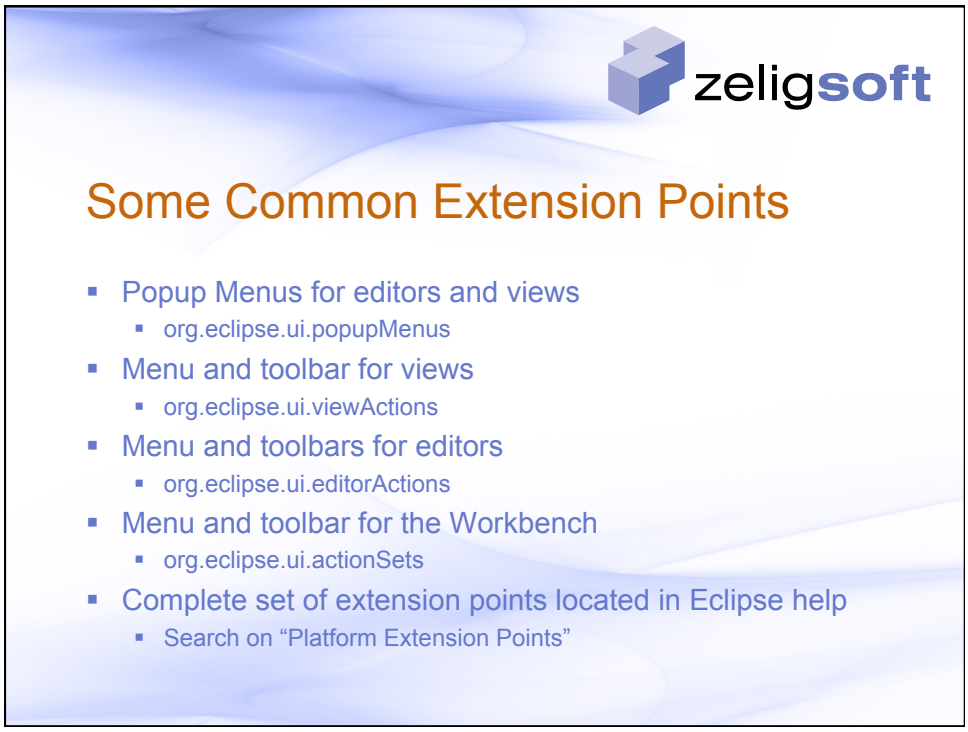

zeligsoft
Integration Between Plug-ins
Supported through the ability to contribute actions to existing plug-ins
New plug-in contributes an action to existing plug-in
Allows for tight integration between plug-ins
There are many areas where actions can be contributed
Context menus and editors
Local toolbar and pull-down menu of a view
Toolbar and menu of an editor - appears on the Workbench when editor opens
Main toolbar and menu for the Workbench

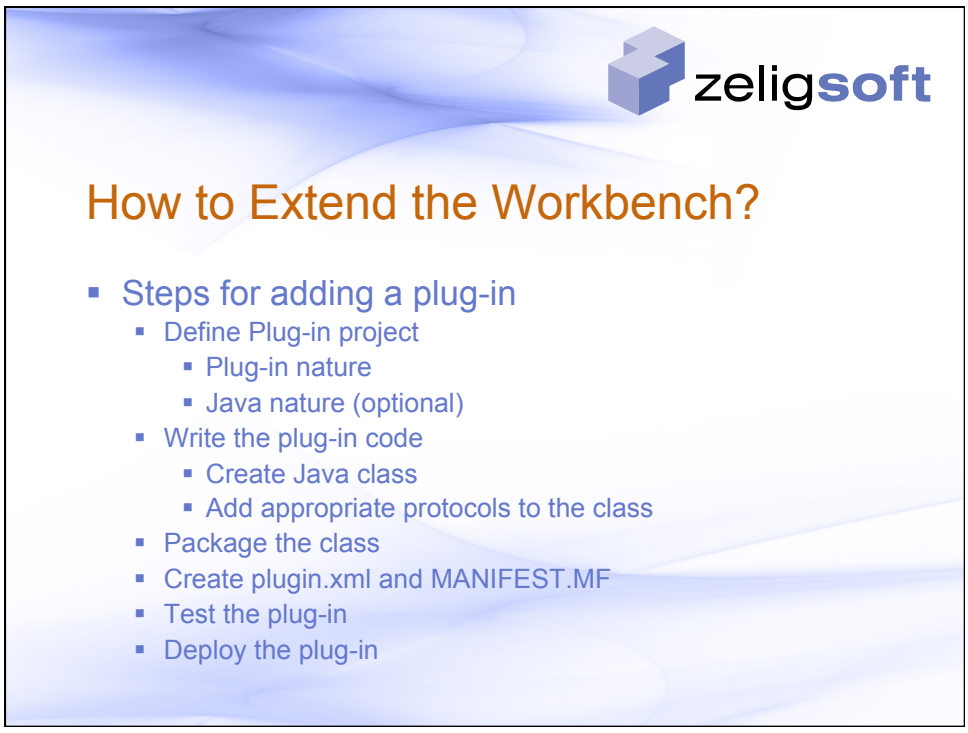

zeligsoft
Extending Views
Changes made to a view are specific to that view
Changes can be made to view's
Context menu
Menu and toolbar


zeligsoft
Extending Editors
When extending an editor changes can be made to
Editor's context menu
Editor's menu and toolbar
Actions defined are shared by all instances of the same editor type
Editors stay open across multiple perspectives, so their actions stay the same
When new workspace is open, or workbench is open in a new window, new instance of editor can be created


zeligsoft
Action Set
Allows extension of the Workbench that is generic
Should be used for adding non-editor, or non-view specific actions to the Workbench
Actions defined are available for all views and editors
Allows customization of the Workbench that includes defining
Menus
Menu items
Toolbar items


zeligsoft
Choosing What to Extend
Eclipse has a set of predefined extension points
Called Platform extension points
Comprehensive set of extension points
Detailed in the on-line help
It is possible to create new extension points
Requires id, name, and schema to be defined
Done through Plug-in Development Environment (PDE)

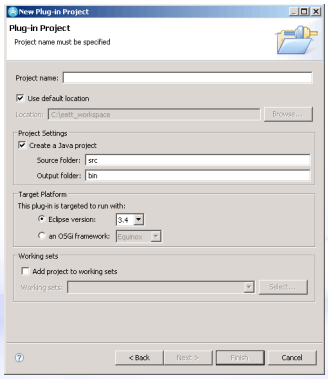

zeligsoft
Some Common Extension Points
Popup Menus for editors and views
org.eclipse.ui.popupMenus
Menu and toolbar for views
org.eclipse.ui.viewActions
Menu and toolbars for editors
org.eclipse.ui.editorActions
Menu and toolbar for the Workbench
org.eclipse.ui.actionSets
Complete set of extension points located in Eclipse help
Search on “Platform Extension Points”


zeligsoft
How to Extend the Workbench?
Steps for adding a plug-in
Define Plug-in project
Plug-in nature
Java nature (optional)
Write the plug-in code
Create Java class
Add appropriate protocols to the class
Package the class
Create plugin.xml and MANIFEST.MF
Test the plug-in
Deploy the plug-in

zeligsoft

Creating Plug-in Project

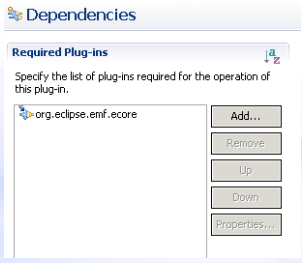
- Plug-in project are created with the New Project wizard
  - File → New → Project…
- Project will contain
  - source code for the plug-in
  - manifest and property files

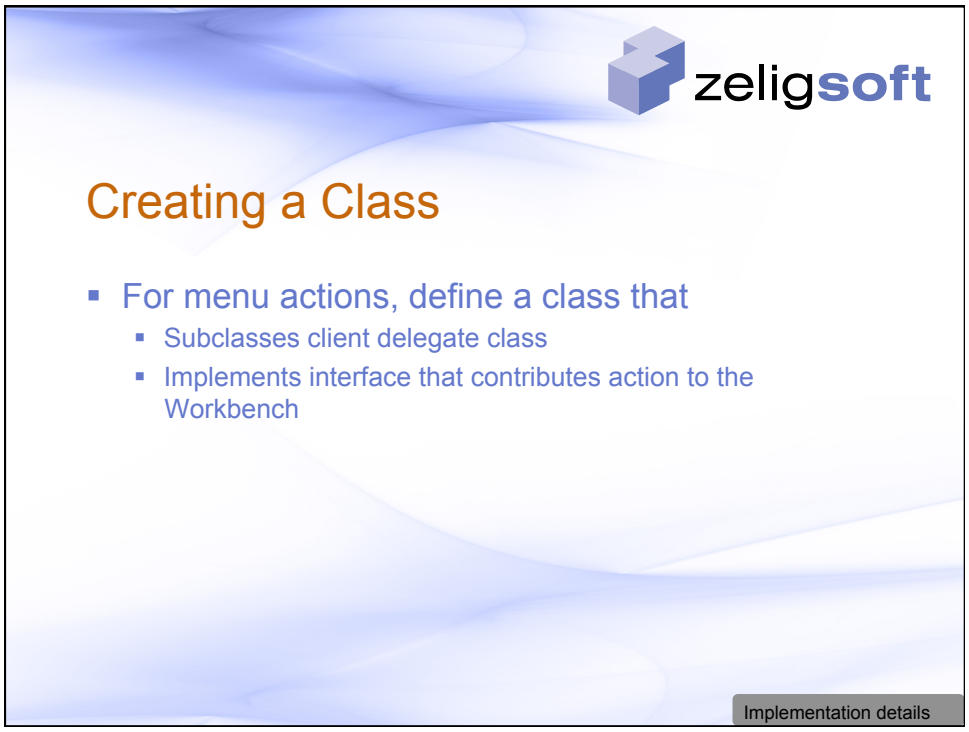


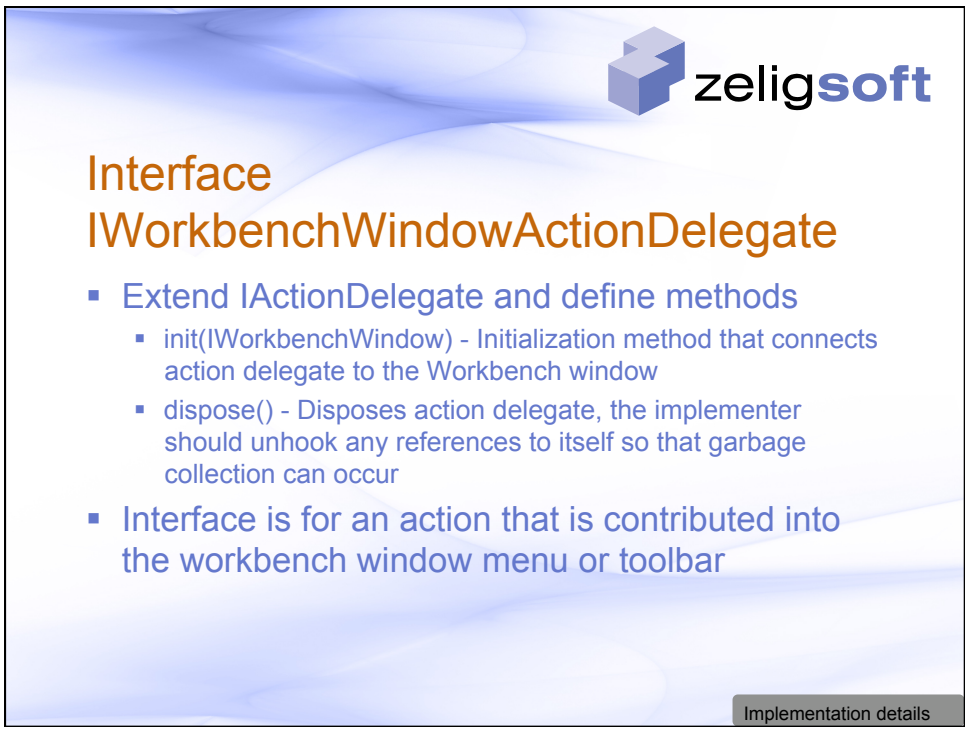
zeligsoft

Updating Project's Dependencies

- To make classes required for the plug-in visible for the project, update its dependencies
  - You add a dependency on another plug-in
  - You can add a jar that is not part of a plug-in to the build path




zeligsoft
Creating a Class
For menu actions, define a class that
Subclasses client delegate class
Implements interface that contributes action to the Workbench
Implementation details


zeligsoft
Interface IWorkbenchWindowActionDelegate
Extend IActionDelegate and define methods
init(IWorkbenchWindow) - Initialization method that connects action delegate to the Workbench window
dispose() - Disposes action delegate, the implementer should unhook any references to itself so that garbage collection can occur
Interface is for an action that is contributed into the workbench window menu or toolbar
Implementation details

zeligsoft

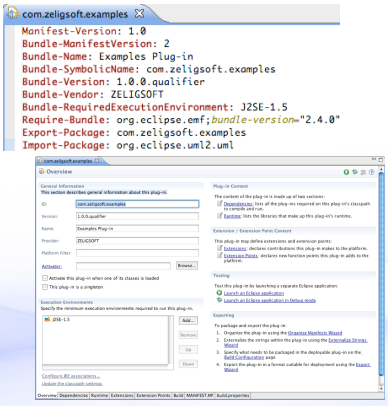
Class ActionDelegate

- Abstract base implementation for client delegate action (defines same methods as interface)
- In addition it also defines
  - runWithEvent(IAction, Event)
    - Does the actual work when action is triggered
    - Parameters represent the action proxy that handles the presentation portion of the action and the SWT event which triggered the action being run
    - Default implementation redirects to the run() method
  - run(IAction)
    - Inherited method, does the actual work as it's called when action is triggered
  - selectionChanged(IAction, ISelection)
    - Inherited method, notifies action delegate that the creation in the Workbench has changed

zeligsoft

Defining Manifest

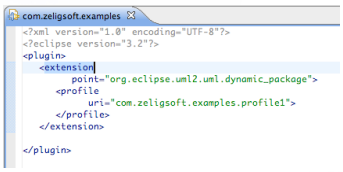
- MANIFEST.MF file
  - In the META-INF directory
- Defines
  - Plug-in name
  - Plug-in version
  - Plug-in vendor/provider
  - Plug-in dependencies
  - Packages exported
- Manifest editor available

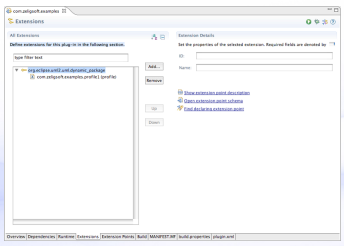


zeligsoft

Defining plugin.xml

- In the root of the project
- Defines
  - Extensions
  - Extension points
- Extension editor
  - Includes wizards
- Extension point editor



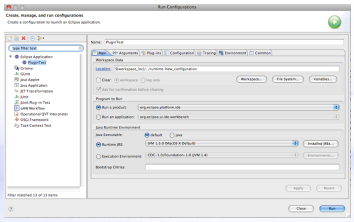


Implementation details

zeligsoft

Testing Plug-in from Workbench

- Available by opening new Workbench instance
  - Run → Run As → Eclipse Application
- Used when developing plug-in within the platform
- Can be configured
  - Run → Run Configurations...
  - Eclipse Application
- Debug available
  - Run → Debug As → Eclipse Application



Implementation details

zeligsoft

Getting Ready to Export Plug-in

- Create a build.properties file in your project

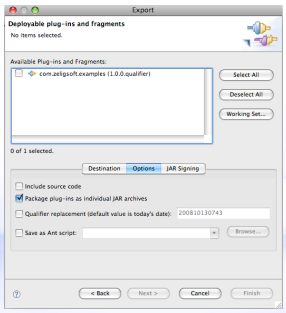
```
source.. = src/
output.. = bin/
bin.includes = META-INF, \
               .,\
               plugin.xml, \
               plugin.properties
```

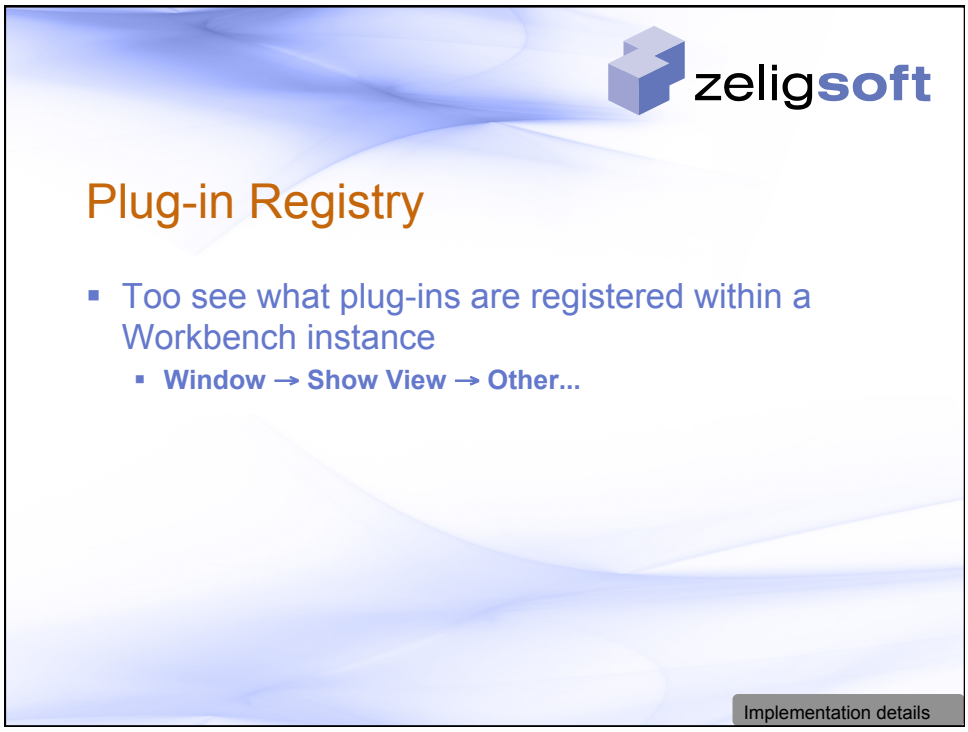
- Build properties editor available
  - Open with → Build Properties Editor

zeligsoft

Exporting a Plug-in

- **File → Export...**
  - Deployable Plug-ins and Fragments
- Configure export
  - Archive vs. Directory
  - Packaging
  - Include source?
- Can save as ant script
- Ready to install

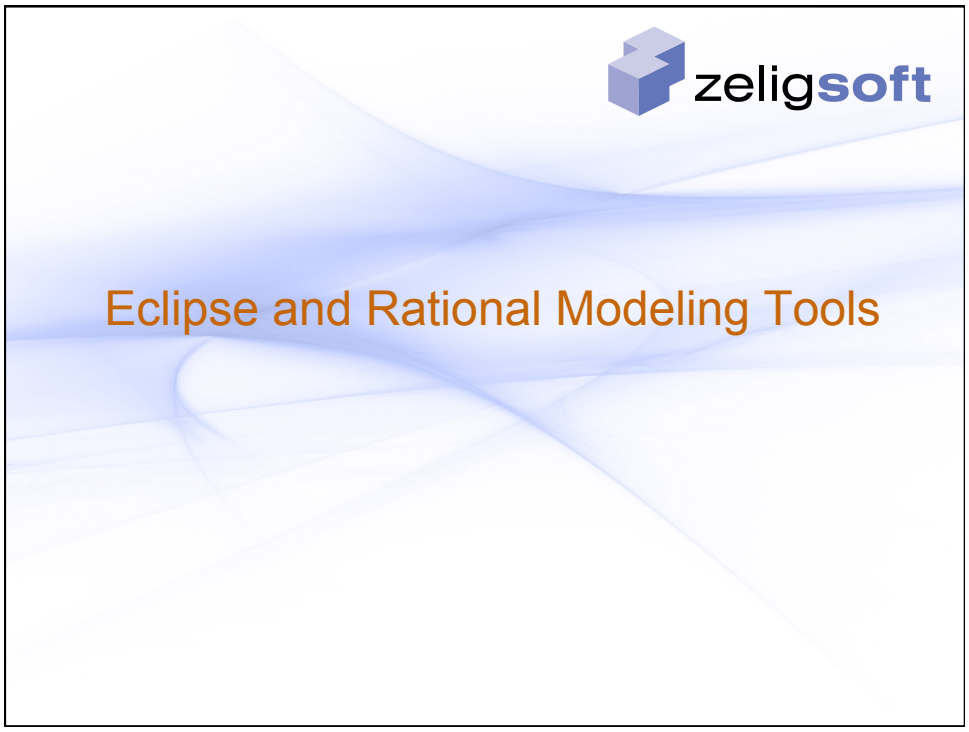



zeligsoft
Plug-in Registry
Too see what plug-ins are registered within a Workbench instance
Window → Show View → Other...
Implementation details


zeligsoft
Summary
In this module we explored
Plug-ins
MANIFEST
Plug-in descriptors
Testing plug-ins
Exporting plug-ins

zeligsoft

Questions


zeligsoft
Eclipse and Rational Modeling Tools


zeligsoft
Overview
In this module we explore the eclipse modeling tools in more detail
Look at the different projects and how they relate


zeligsoft
Goals
After this module you will
Have an understanding of Eclipse modeling projects
How they extend each-other
How they can be transformed into models and text
You will also have a roadmap for the rest of the training


zeligsoft
Agenda
Modeling in Eclipse
EMF, GMF, TMF, TCS
Tooling in Eclipse
MDT, EMFT,
Transformations
M2M, M2T

zeligsoft

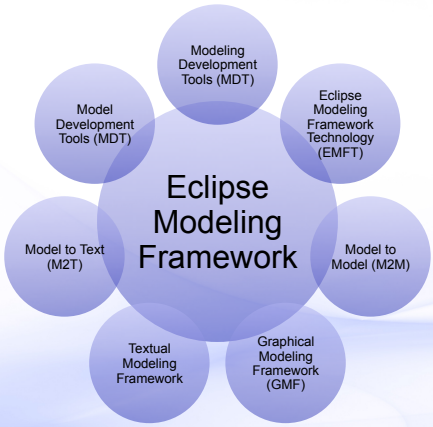
Eclipse Modeling Project

- Model-based development technologies
  - Frameworks
  - Tooling
  - Implementations of standards

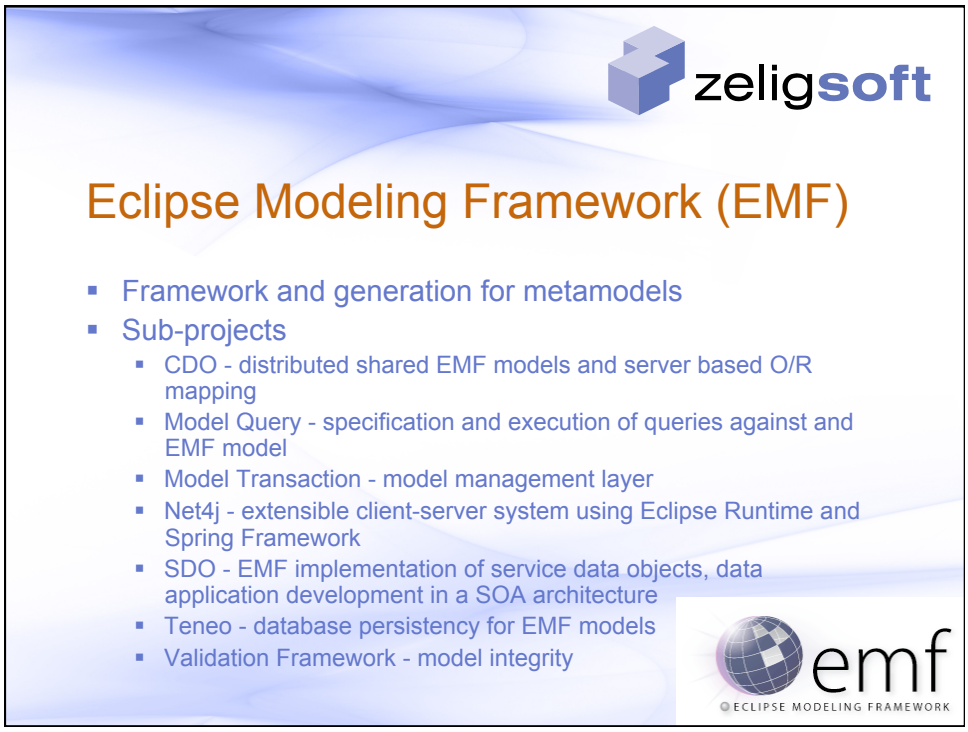
eclipse modeling PROJECT

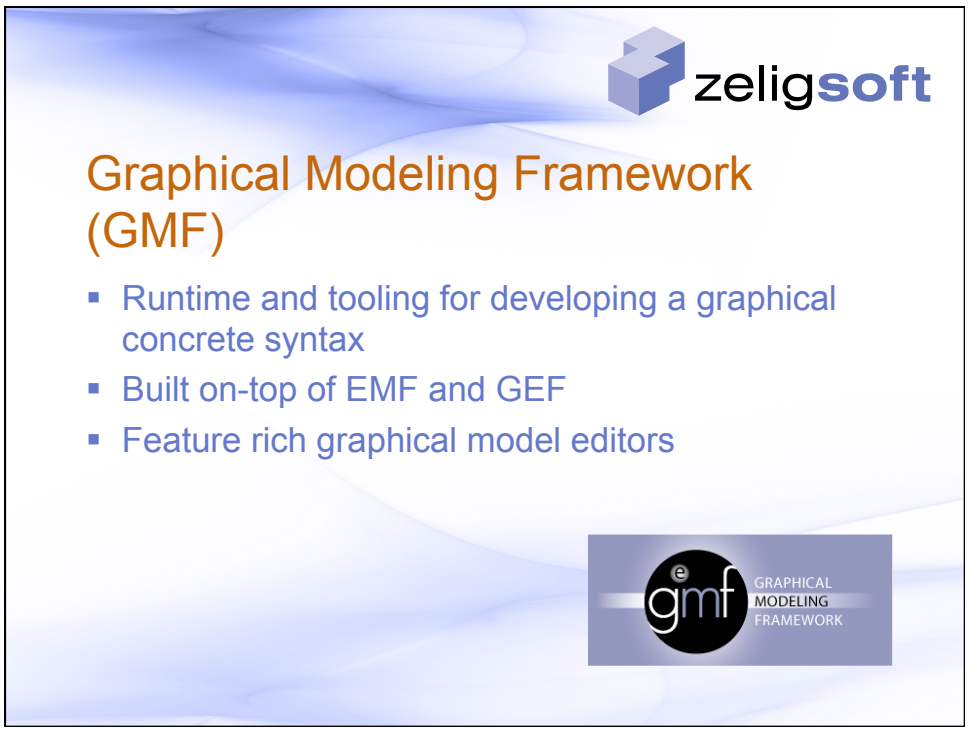
zeligsoft

Ecosystem of the Modeling Project



Toby McClean, Zeligsoft, 2008





Toby McClean, Zeligsoft, 2008

zeligsoft

Textual Modeling Framework (TMF)

- Runtime and tooling for developing textual concrete syntax
- Generation of editors and other tooling from a grammar
- Sub-projects
  - TCS
  - xtext

zeligsoft

TMF - TCS

- TCS - Textual Concrete Syntax
- Built on top of EMF
- Non-standards based language for annotating an abstract syntax model with textual syntax
- Able to generate
  - Grammar
  - Editor
  - Model to text transformation with traceability

Toby McClean, Zeligsoft, 2008

zeligsoft

TMF - xtext

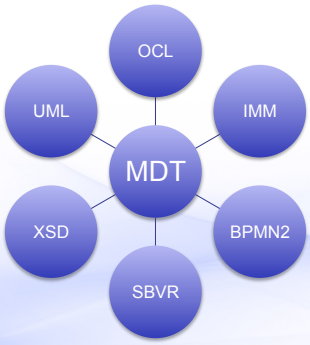
- Developed as part of the openArchitectureWare
- Built on-top of EMF and ANTLr
- From proprietary grammar language
  - Generate editor - syntax highlighting and code completion
  - Generate EMF abstract syntax model
  - Generate integration with generation framework
  - Generate integration with semantic validation framework

xtext

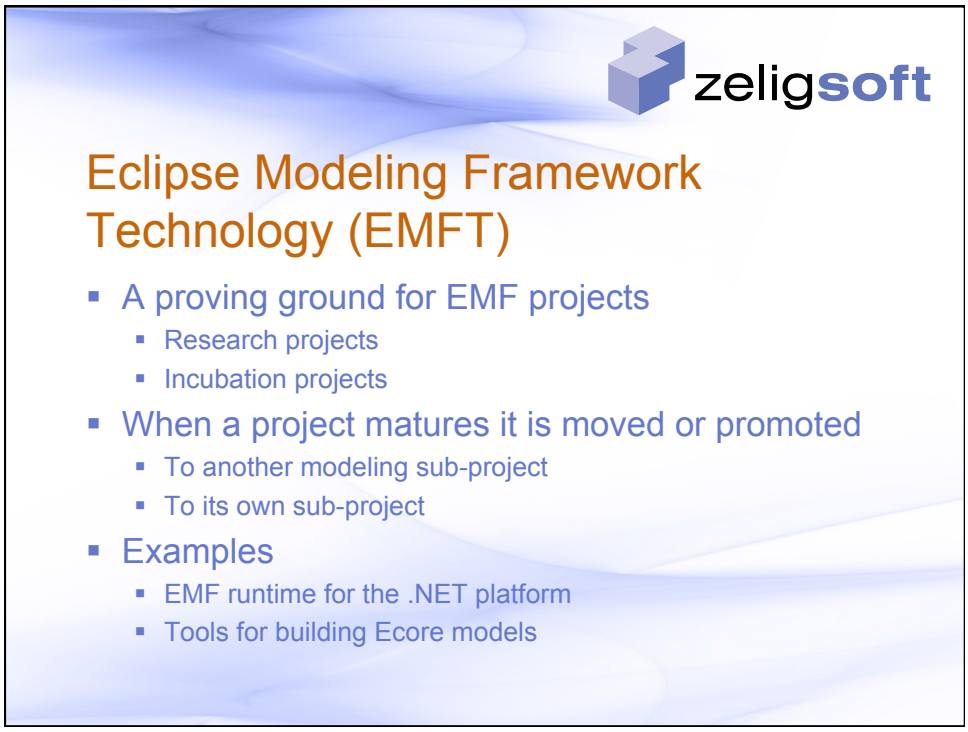
zeligsoft

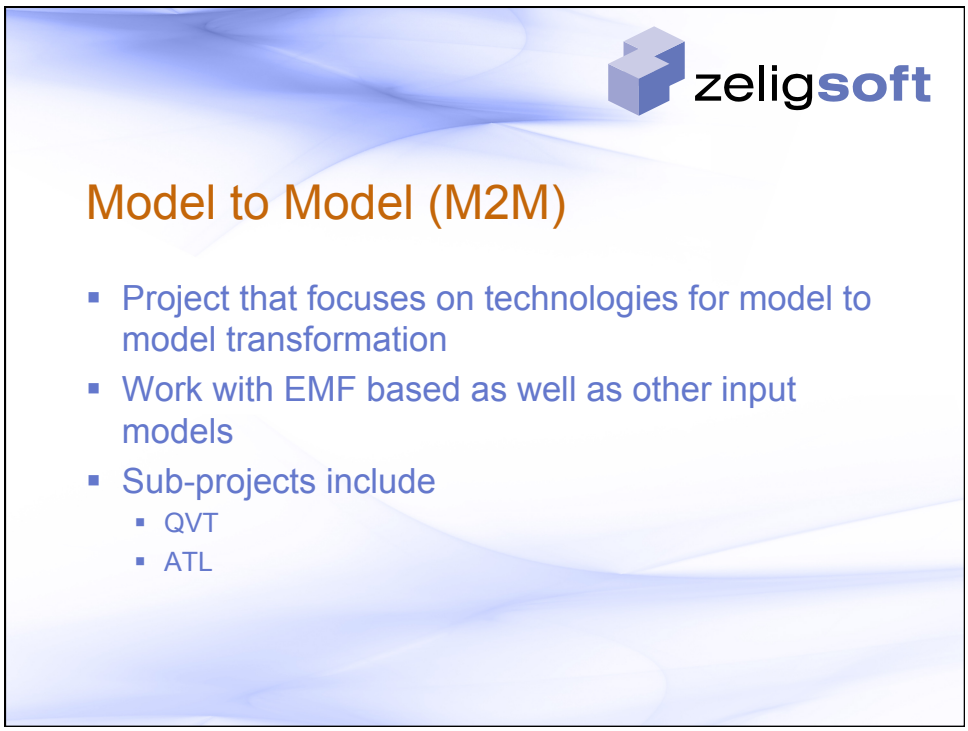
Model Development Tools (MDT)

- Project that focuses on industry stand models

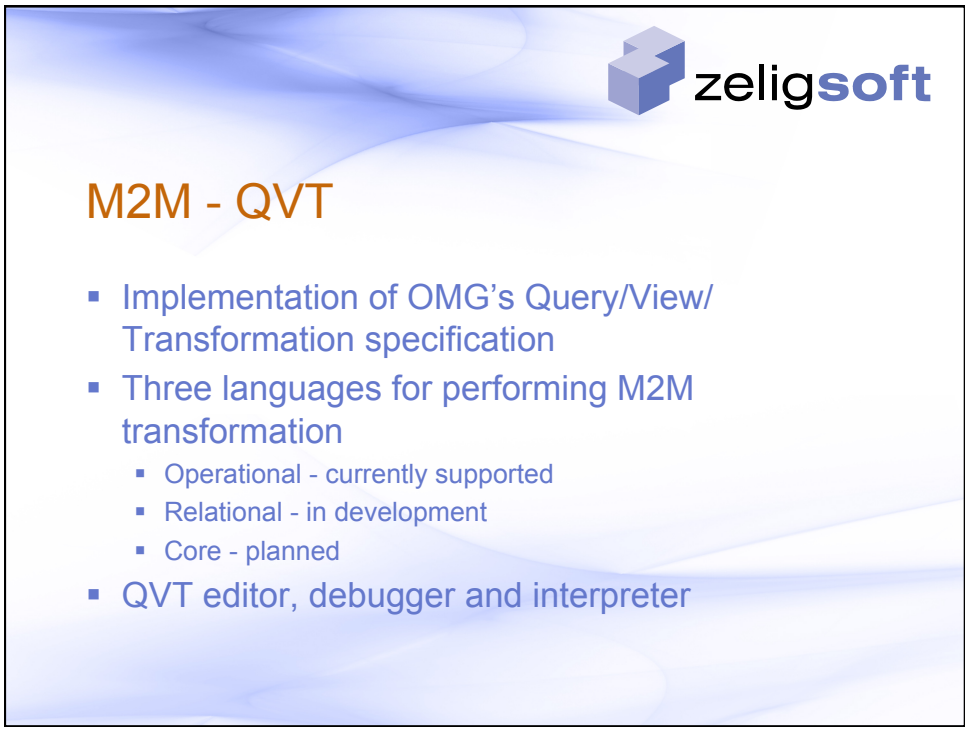


Toby McClean, Zeligsoft, 2008

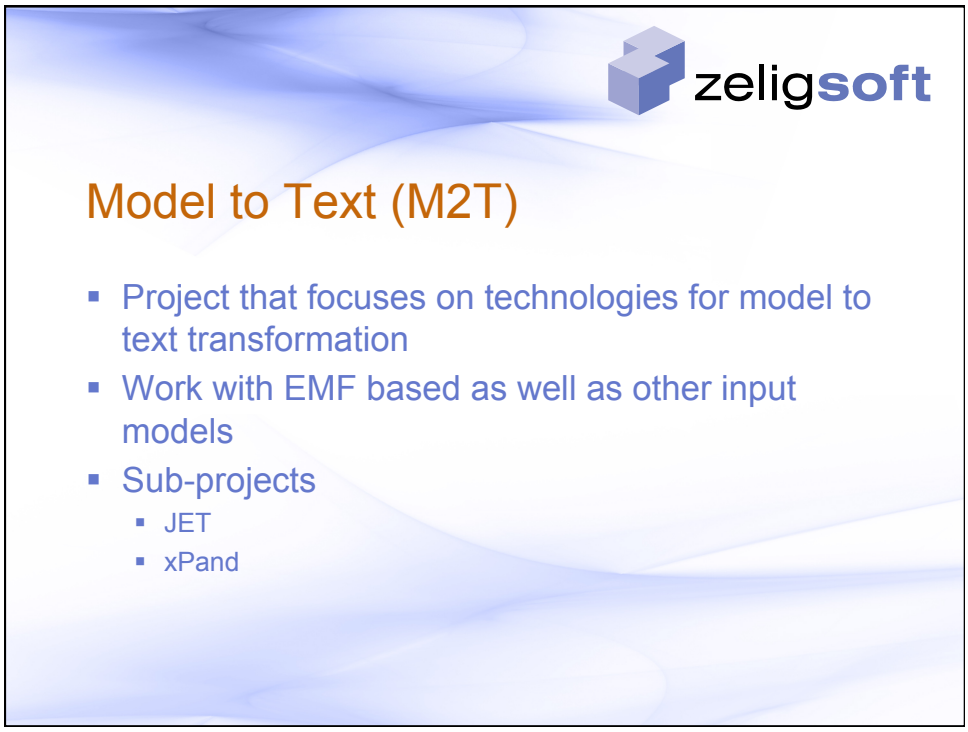




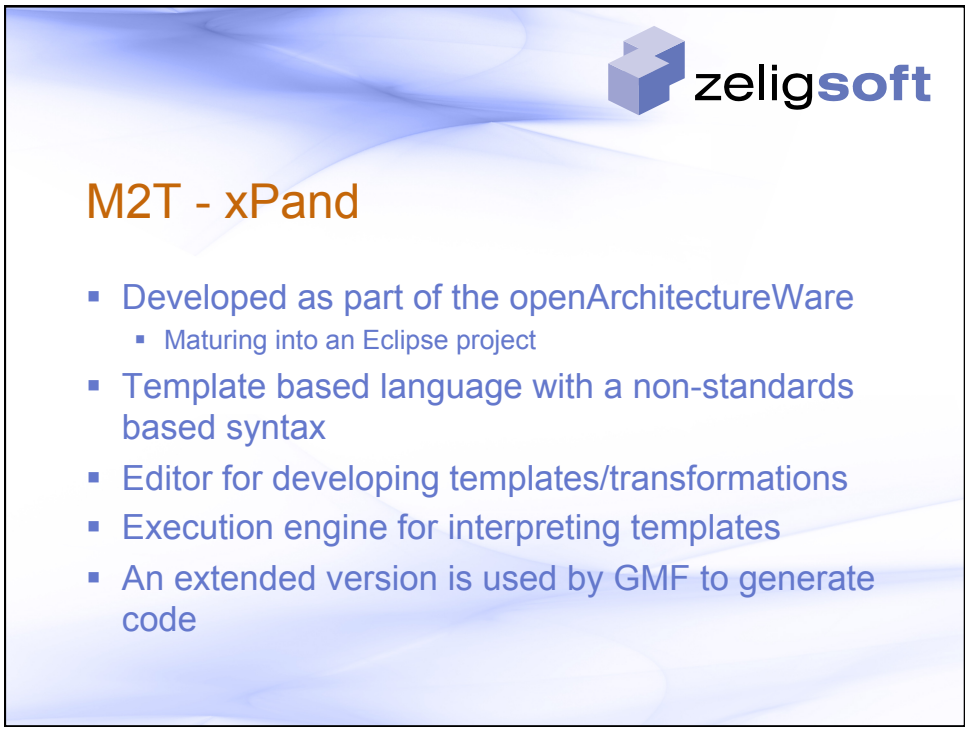
Toby McClean, Zeligsoft, 2008



Toby McClean, Zeligsoft, 2008


zeligsoft
Model to Text (M2T)
Project that focuses on technologies for model to text transformation
Work with EMF based as well as other input models
Sub-projects
JET
xPand

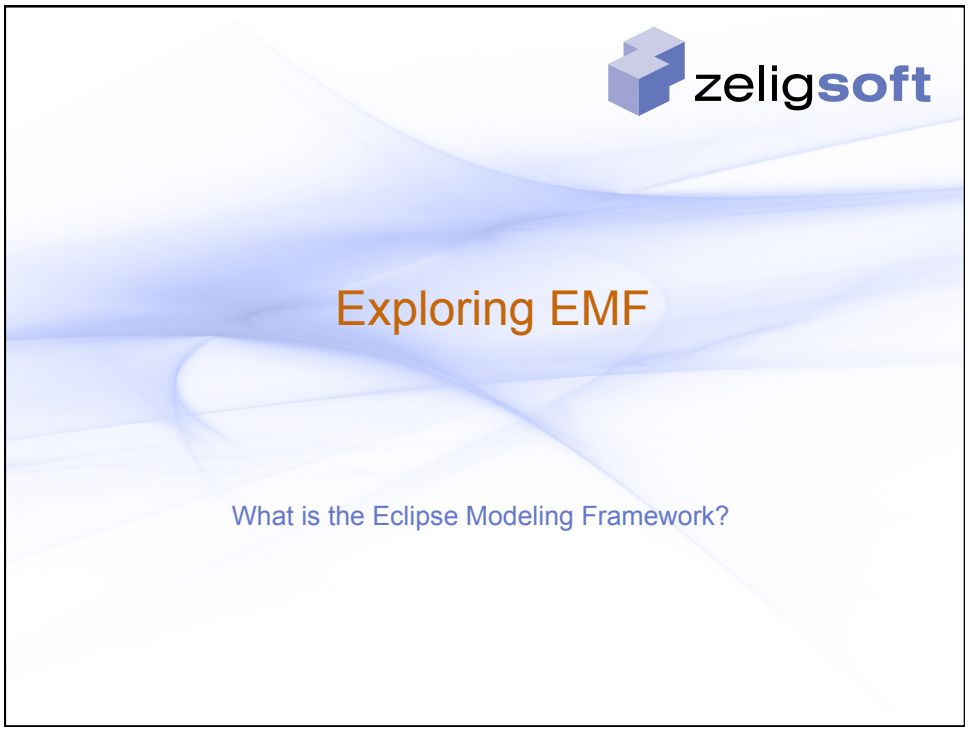

zeligsoft
M2T- JET
JET - Java Emitter Templates
Template based language with an JSP like syntax
Editor for developing templates/transformations
Execution engine for executing transformation
Used by EMF to generate code

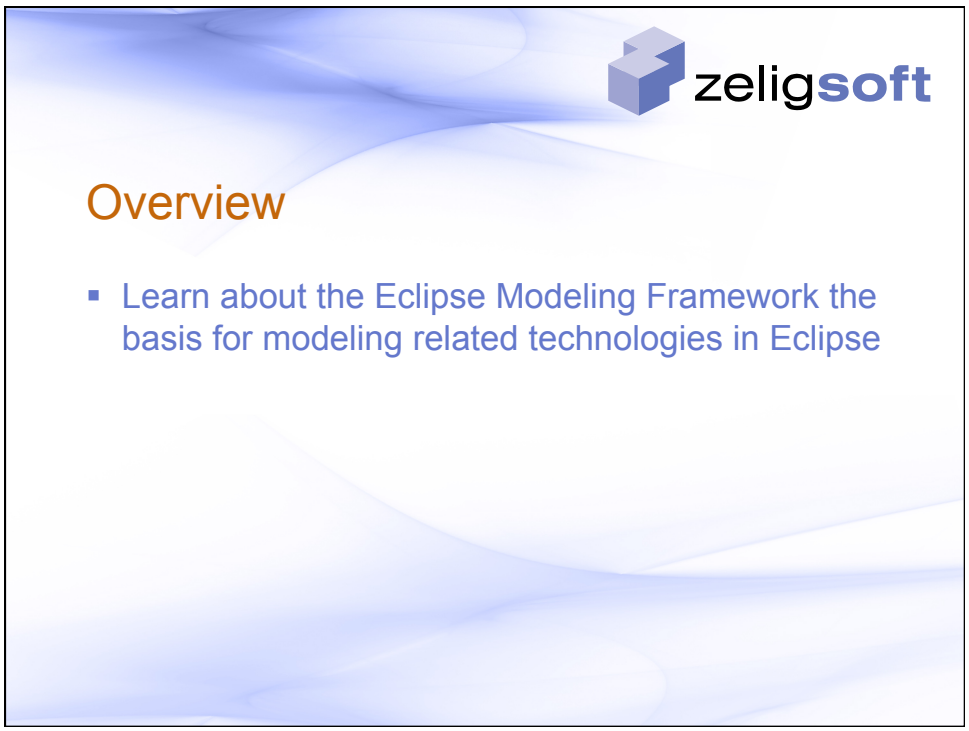

zeligsoft
M2T - xPand
Developed as part of the openArchitectureWare
Maturing into an Eclipse project
Template based language with a non-standards based syntax
Editor for developing templates/transformations
Execution engine for interpreting templates
An extended version is used by GMF to generate code

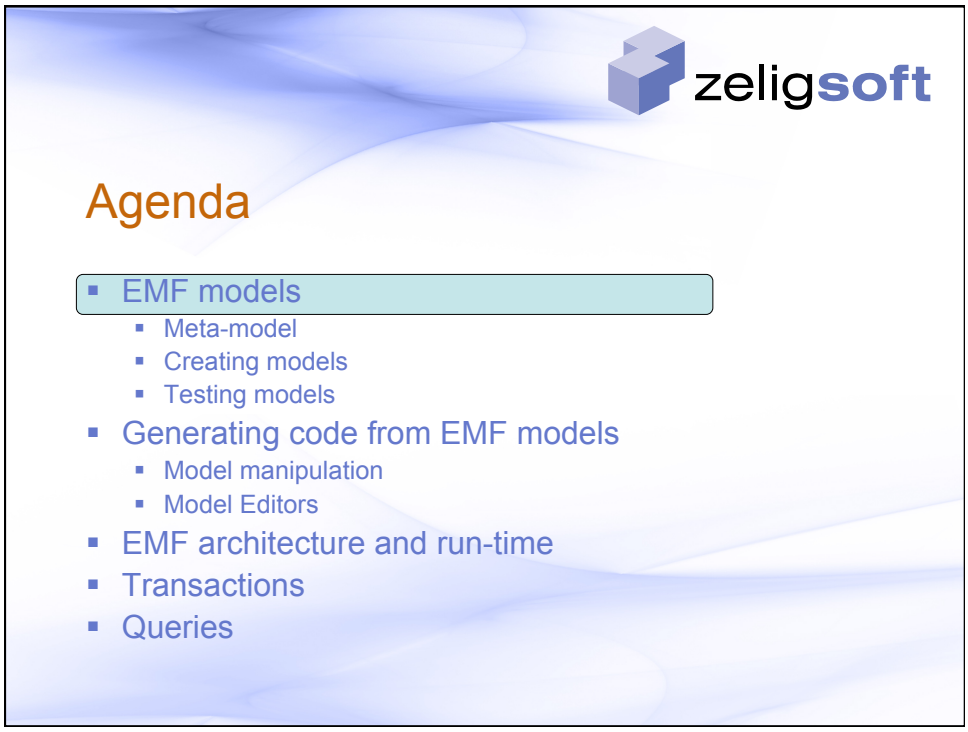

zeligsoft
Summary
The projects are nicely layered
Good effort to keep things separated
Ability for users to pick and choose what they want
Lots of content to go through this week

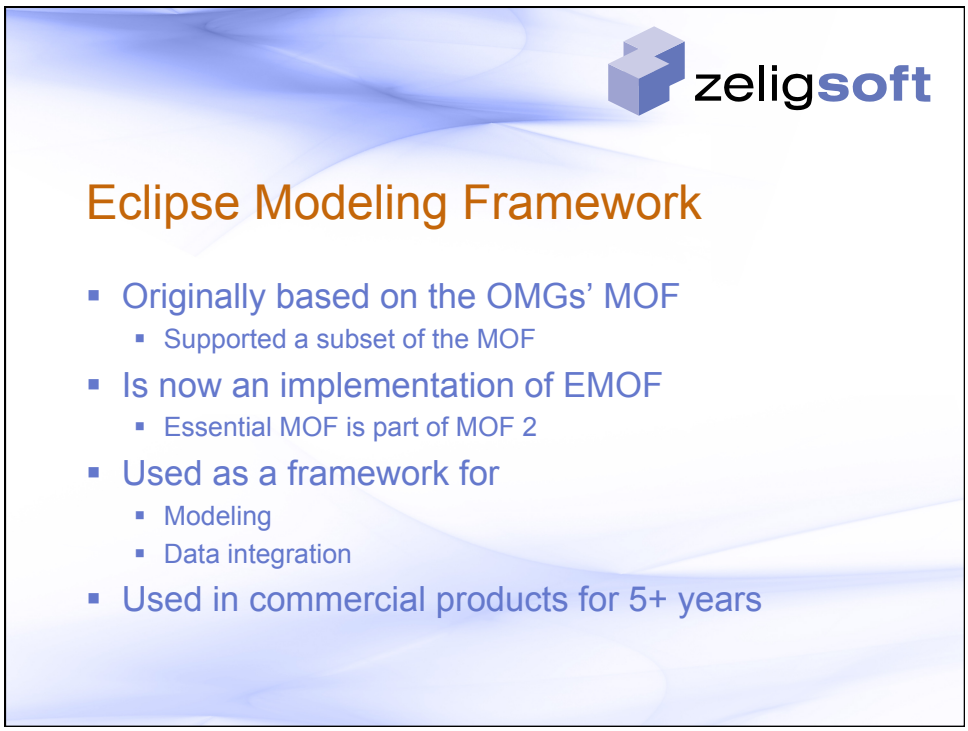
zeligsoft

Questions


zeligsoft
Exploring EMF
What is the Eclipse Modeling Framework?


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


zeligsoft
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


zeligsoft
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

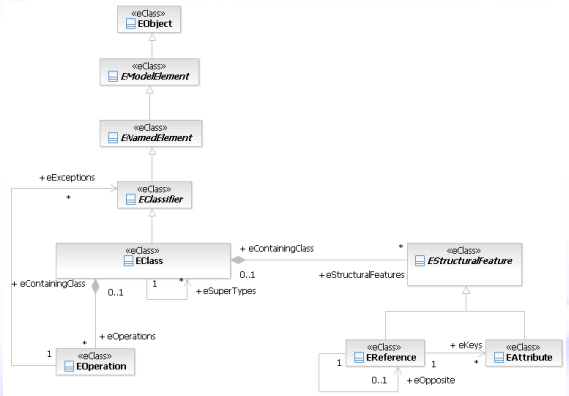
zeligsoft

What is an EMF Model?

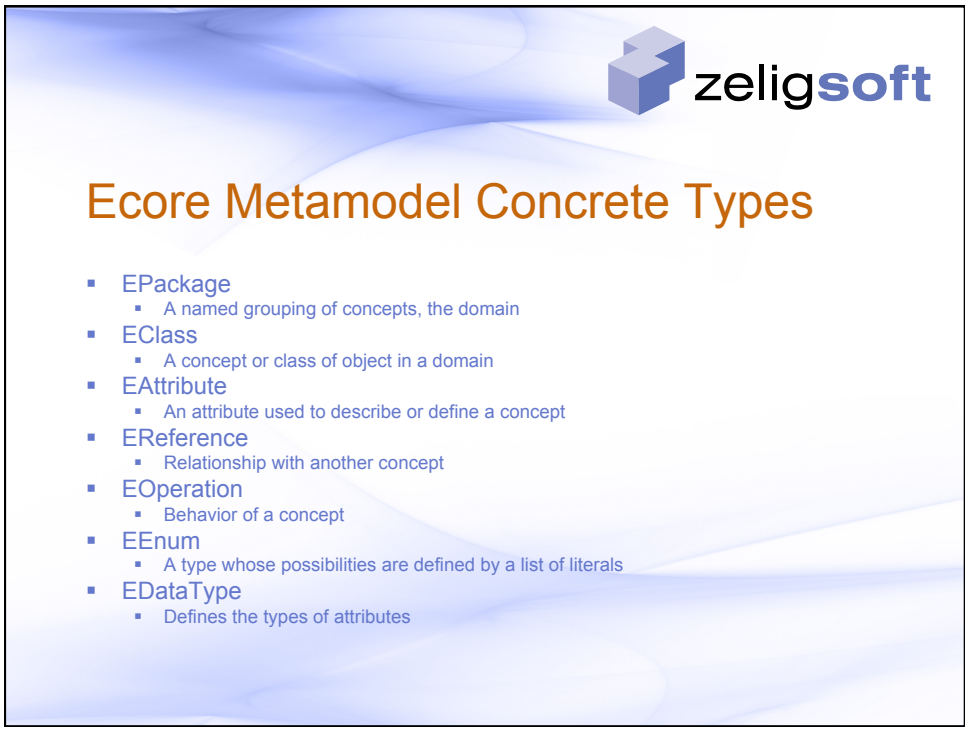
- Description of data for a domain/application
  - The attributes and capabilities of domain concepts
  - Relationships between the domain concepts
  - Cardinalities of attributes and relationships
- It defines the *metamodel* or abstract syntax for your domain
- Defined in the Ecore modeling language

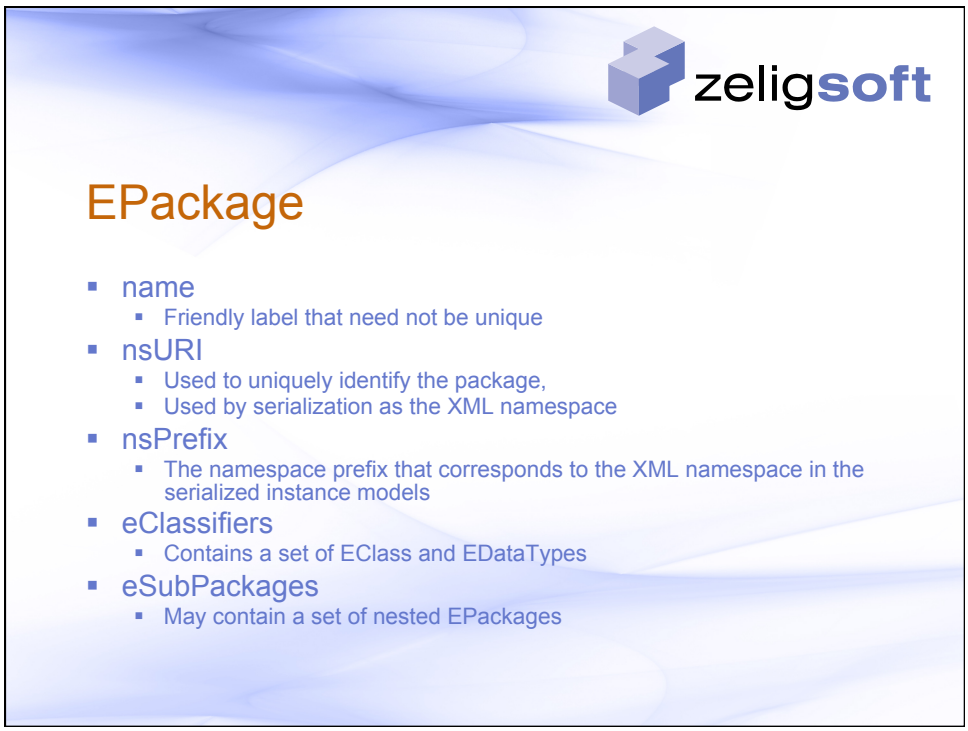
zeligsoft

The Ecore Metamodel

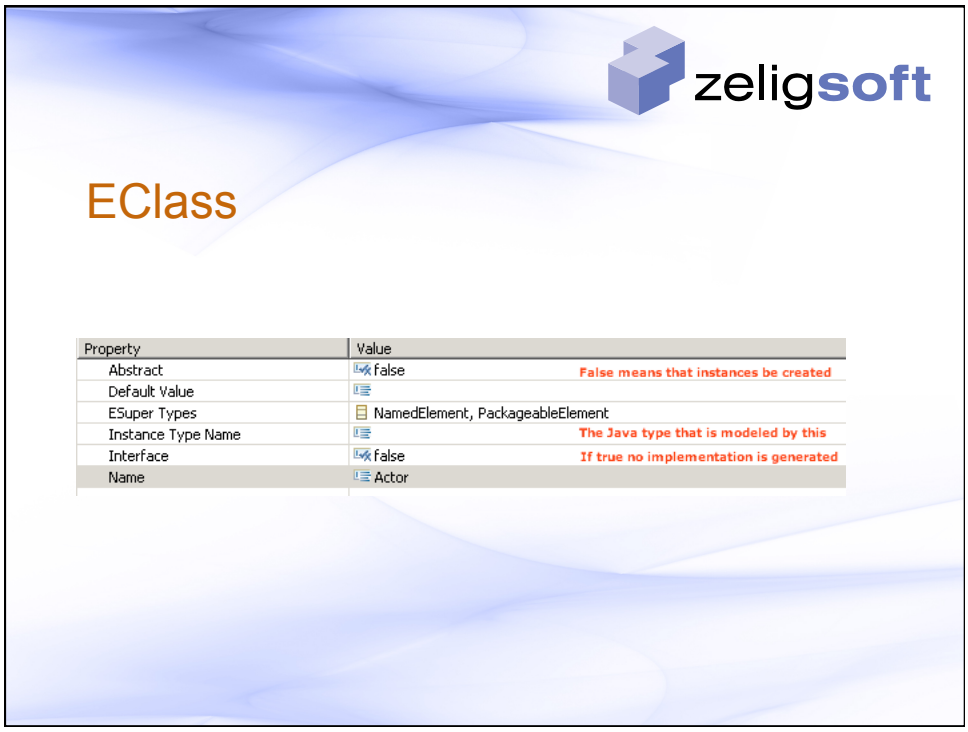


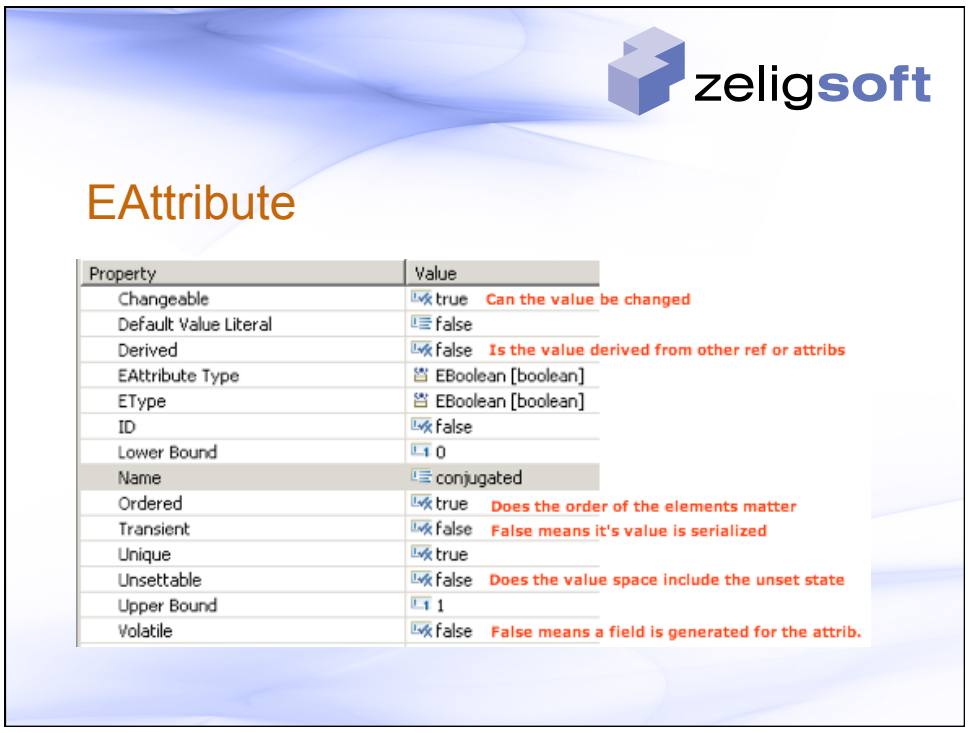
Toby McClean, Zeligsoft, 2008

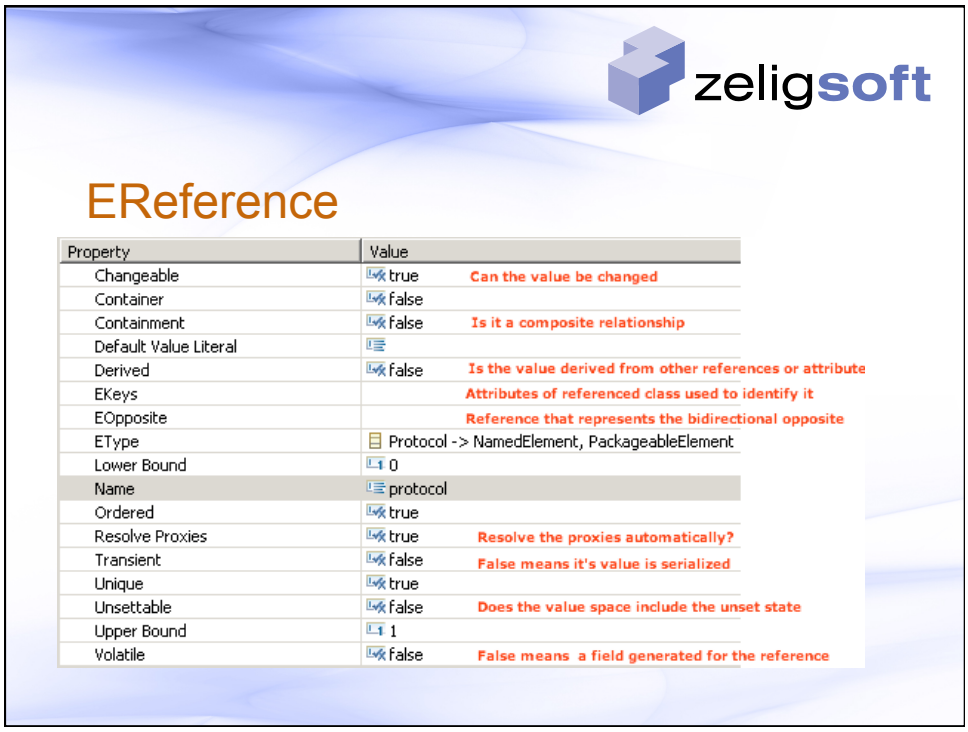




Toby McClean, Zeligsoft, 2008


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.

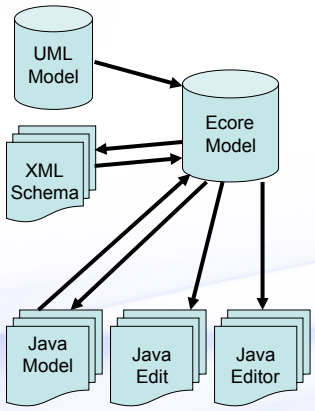


Toby McClean, Zeligsoft, 2008

zeligsoft

Creating an EMF Model

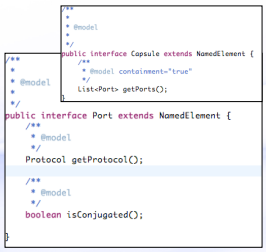
- EMF models can be created from
  - Java Interfaces
  - UML Class diagram
  - XML Schema
  - Ecore Diagram Editor
- With EMF if you have one of the above then you can generate the others
- Supports both
  - EMFizing a legacy data model
  - Greenfields development

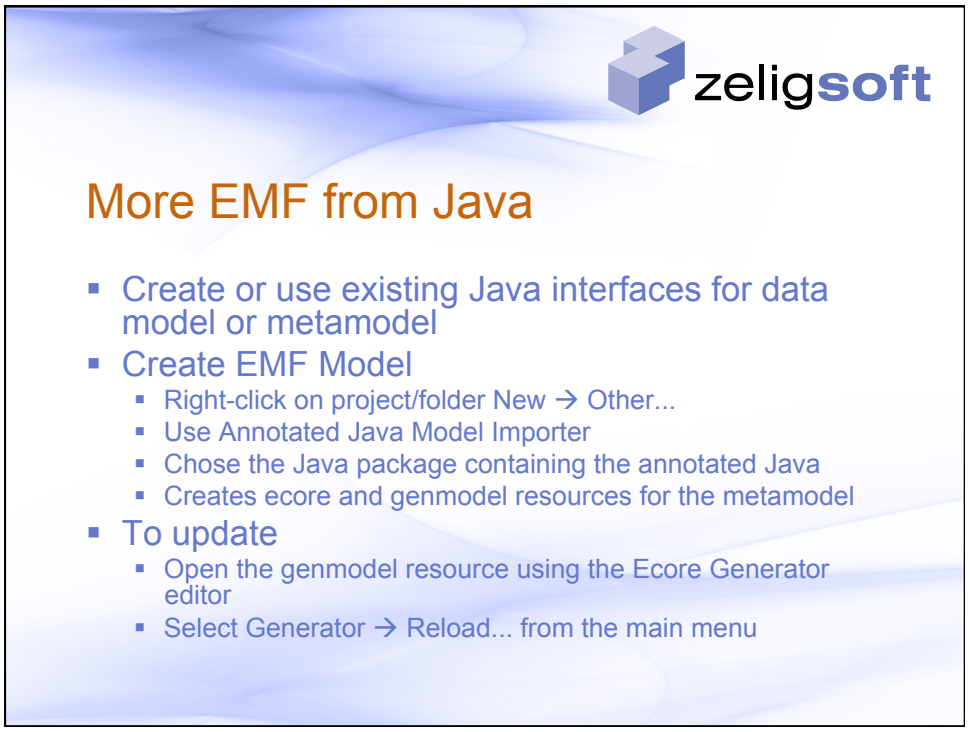


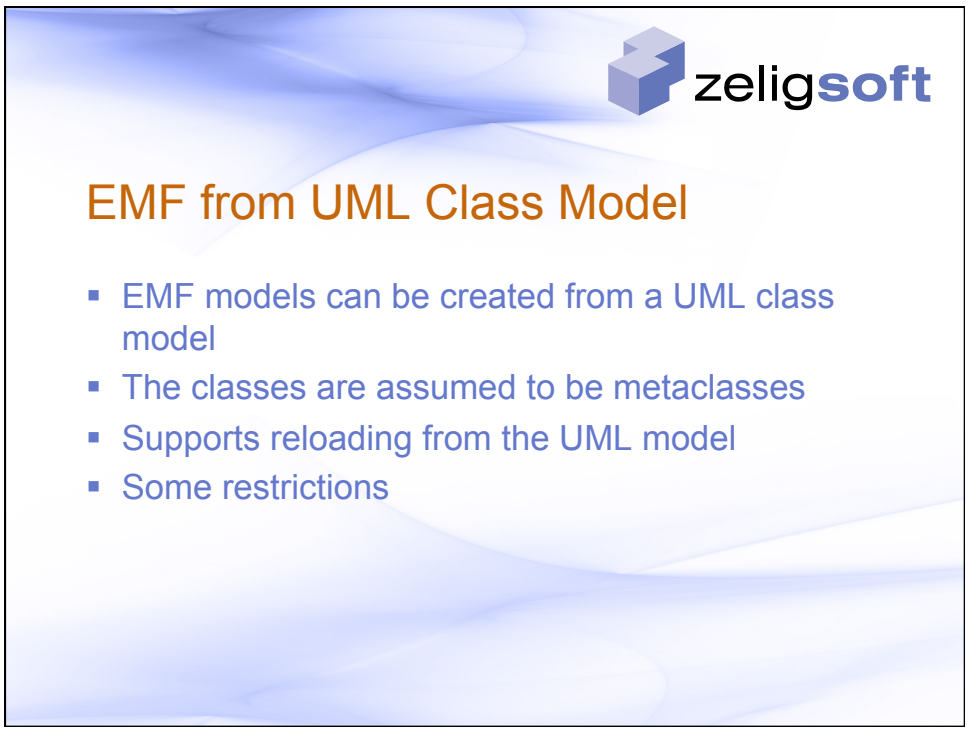
zeligsoft

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



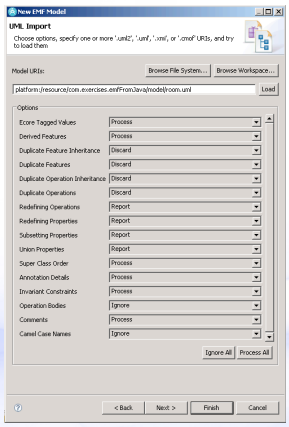

zeligsoft
More EMF from Java
Create or use existing Java interfaces for data model or metamodel
Create EMF Model
Right-click on project/folder New → Other...
Use Annotated Java Model Importer
Chose the Java package containing the annotated Java
Creates ecore and genmodel resources for the metamodel
To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


zeligsoft
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

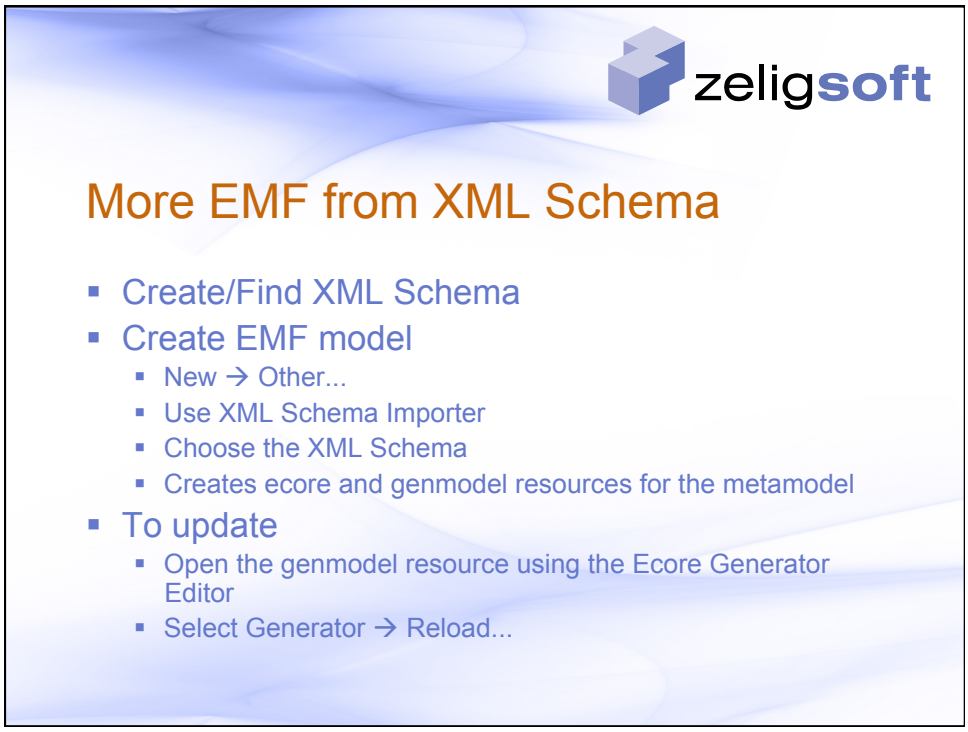
- Create UML model
- Create EMF model
  - New → Other...
  - Use UML Model Importer
  - Chose the UML model
  - Configure import
  - Creates ecore and genmodel resources for the metamodel
- To update
  - Open the genmodel resource using the Ecore Generator Editor
  - Select Generator → Reload...



zeligsoft

EMF from XML Schema

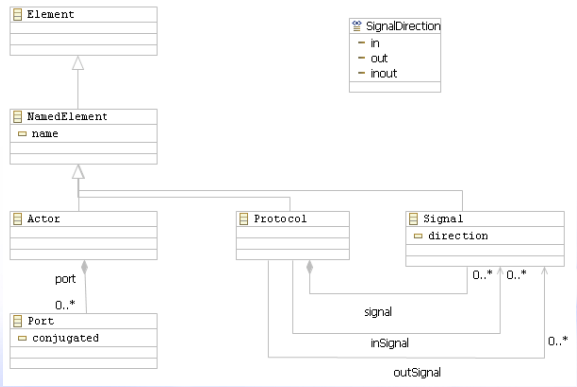
- Many industry standards produce XML schemas to define their data format
- EMF model can be created from an XML schema
  - Model instances are schema compliant
- Supports reloading from XML schema
- Schema can be regenerated from EMF model


zeligsoft
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...


zeligsoft
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

zeligsoft

More EMF with Ecore Diagram Editor



zeligsoft

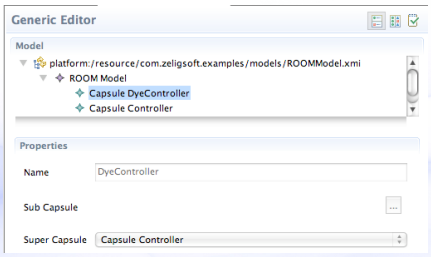
How do I Test my Model?

- 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

zeligsoft

Dynamically Creating Models

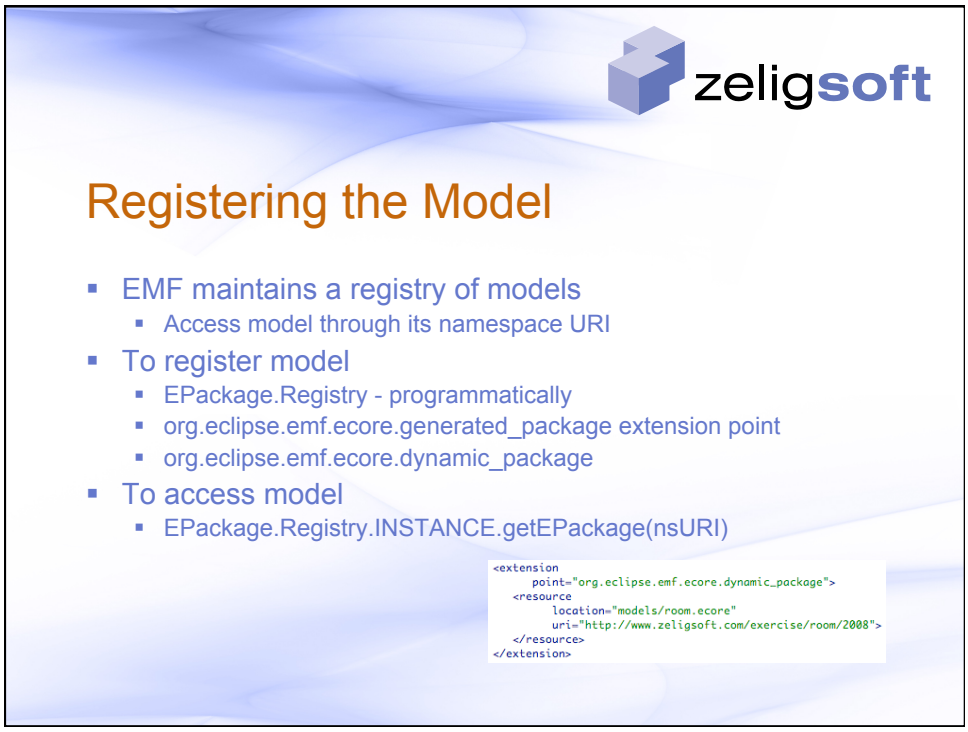
- While developing a model it can be tested by creating dynamic instance
  - Does not require anything to be generated
  - Does not require a runtime workbench
  - Uses reflective capabilities of EMF
- In the Ecore editor
  - Right-click on the EClass
  - Create Dynamic Instance...
- Stored as XMI

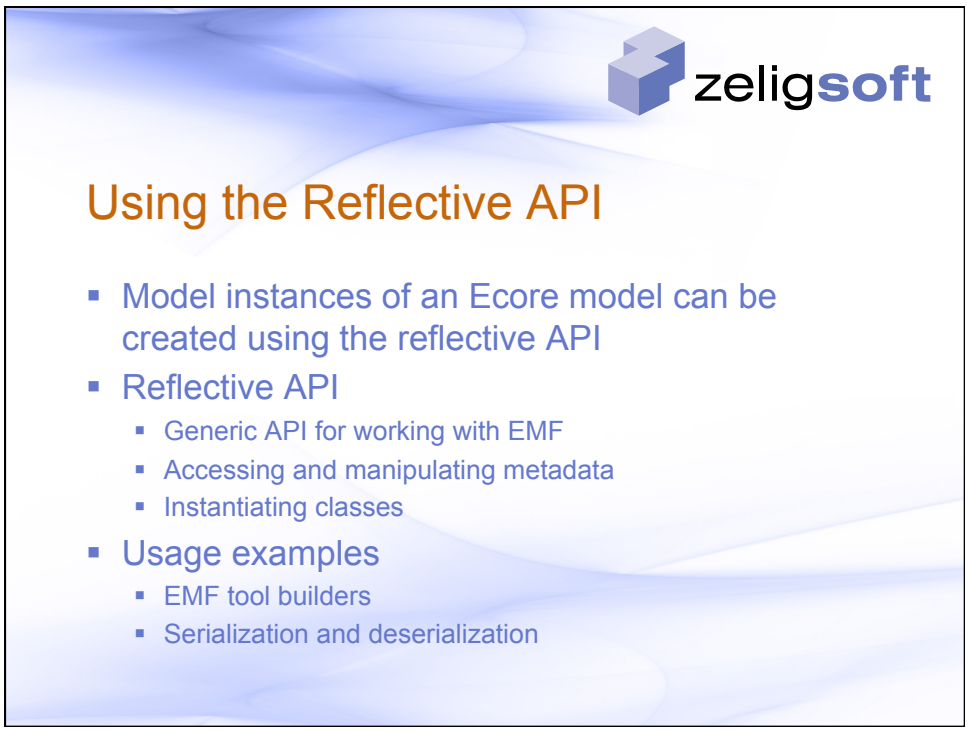


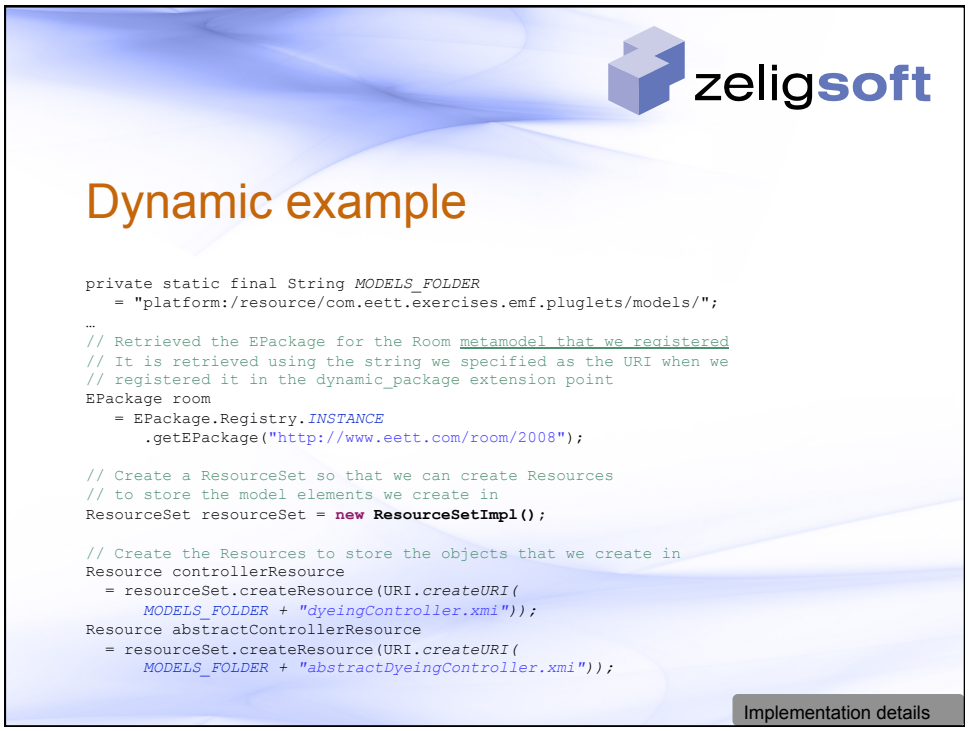
zeligsoft

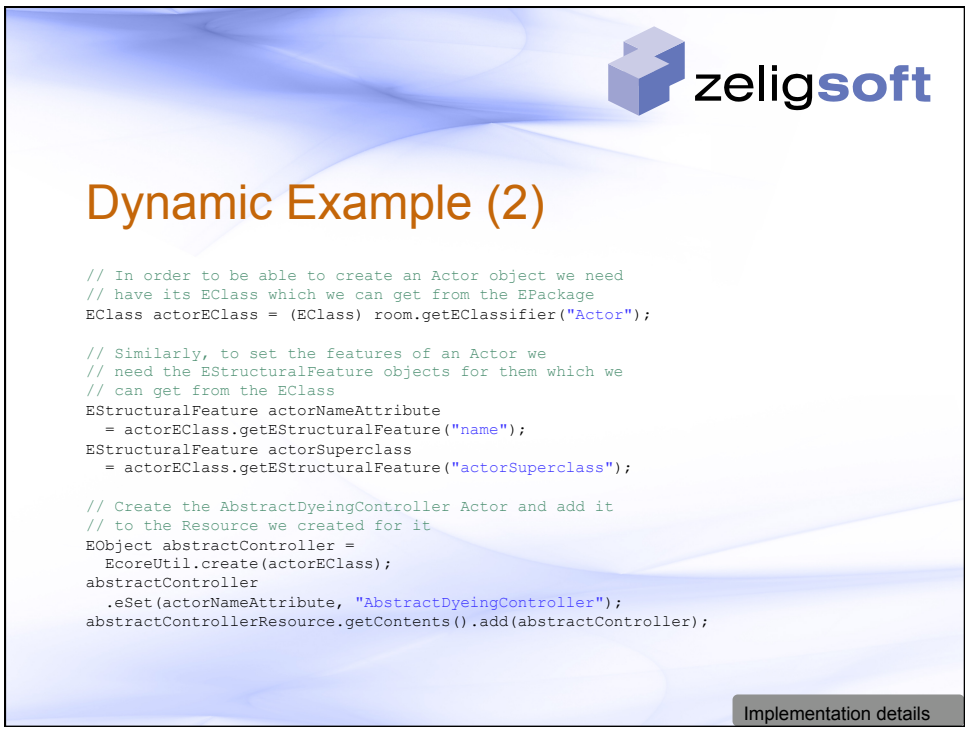
Do this with Code?

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

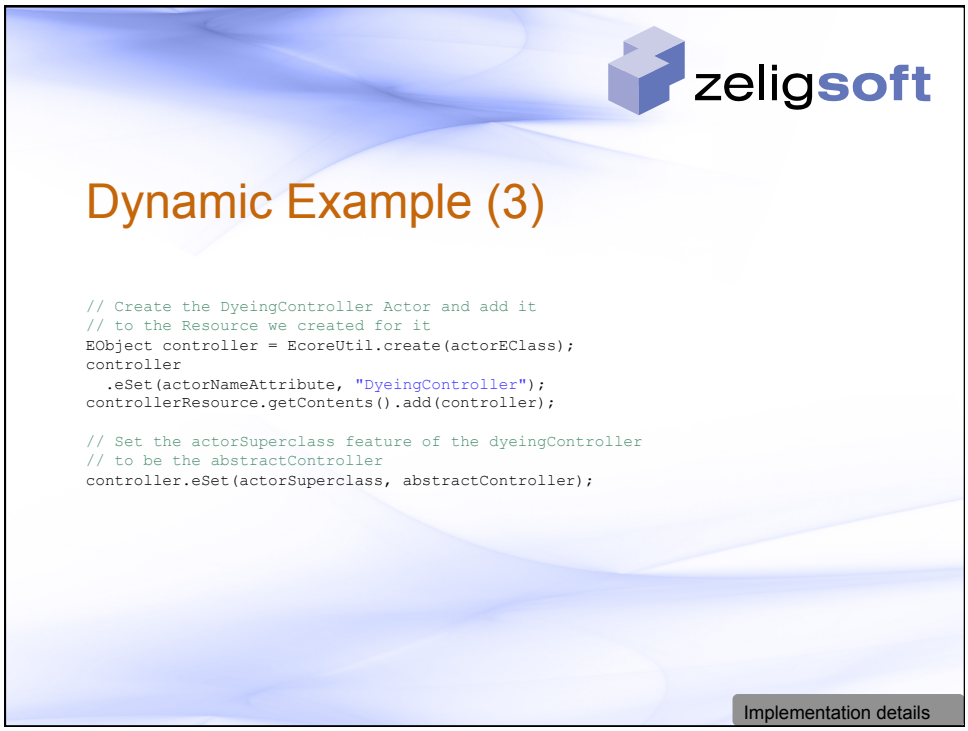

zeligsoft
Registering the Model
EMF maintains a registry of models
Access model through its namespace URI
To register model
EPackage.Registry - programmatically
org.eclipse.emf.ecore.generated_package extension point
org.eclipse.emf.ecore.dynamic_package
To access model
EPackage.Registry.INSTANCE.getEPackage(nsURI)
<extension
point="org.eclipse.emf.ecore.dynamic_package">
<resource
location="models/room.ecore"
uri="http://www.zeligsoft.com/exercise/room/2008">
</resource>
</extension>

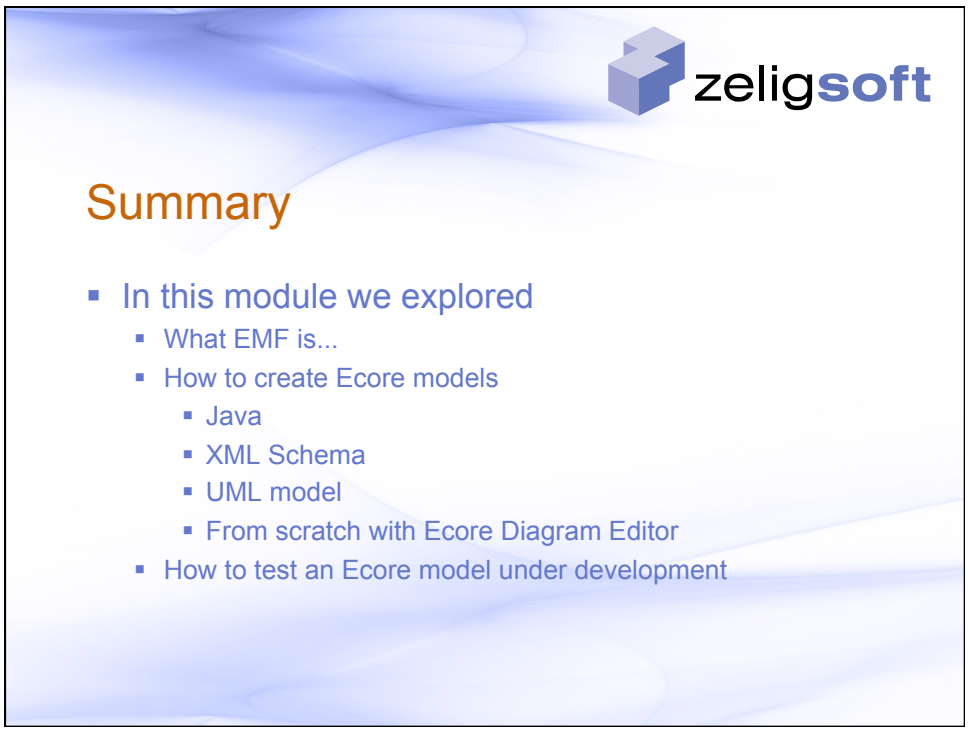

zeligsoft
Using the Reflective API
Model instances of an Ecore model can be created using the reflective API
Reflective API
Generic API for working with EMF
Accessing and manipulating metadata
Instantiating classes
Usage examples
EMF tool builders
Serialization and deserialization

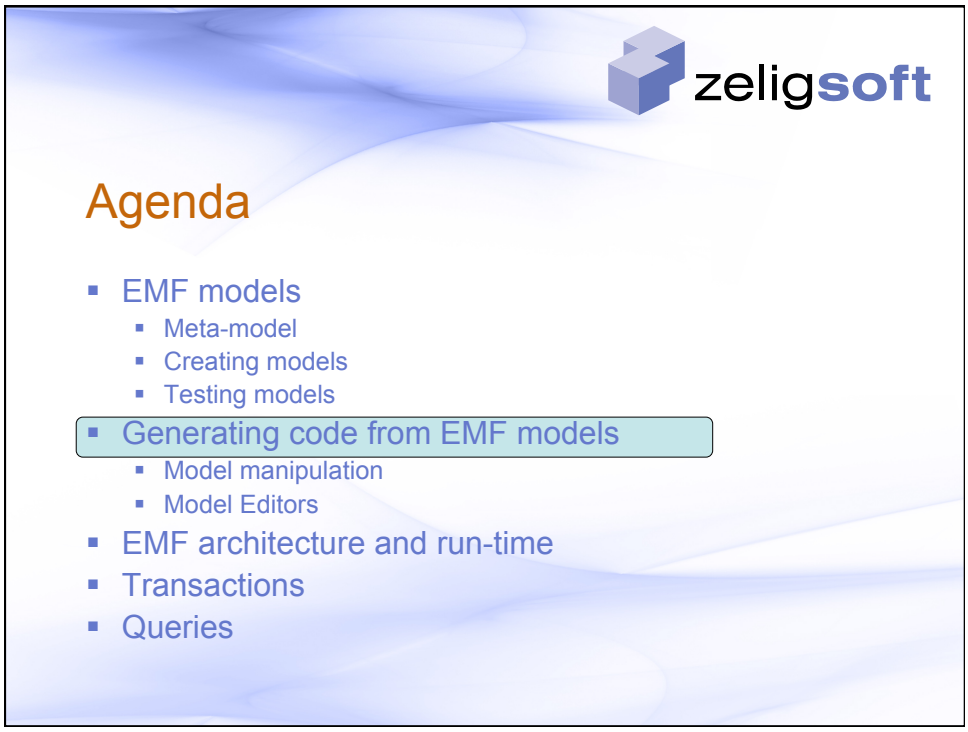


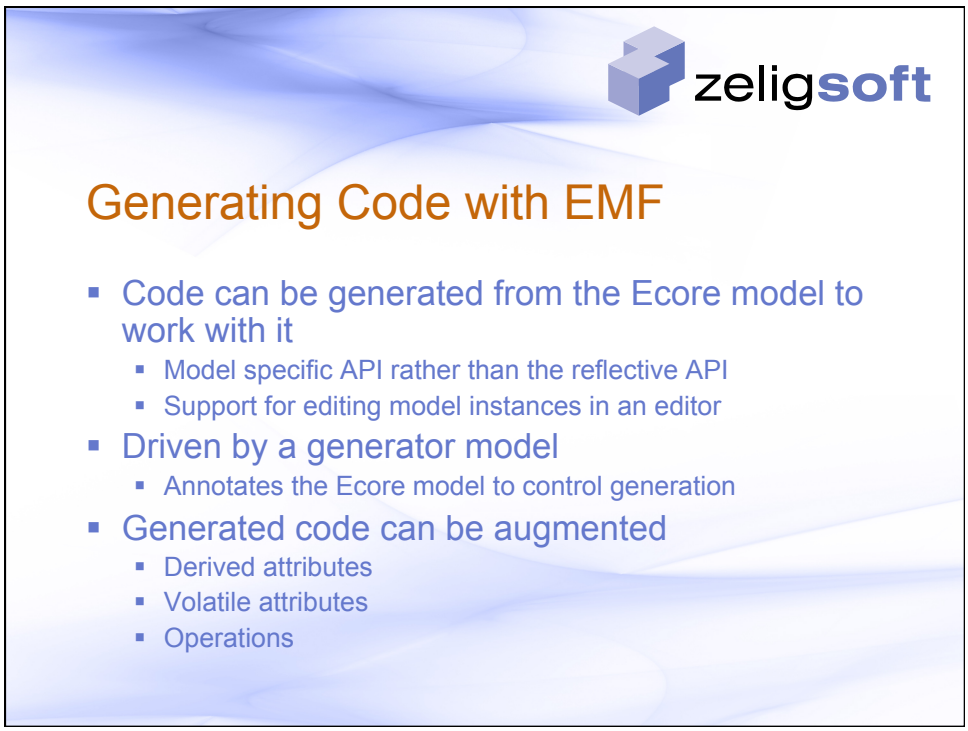


Toby McClean, Zeligsoft, 2008


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


zeligsoft
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


zeligsoft
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

zeligsoft

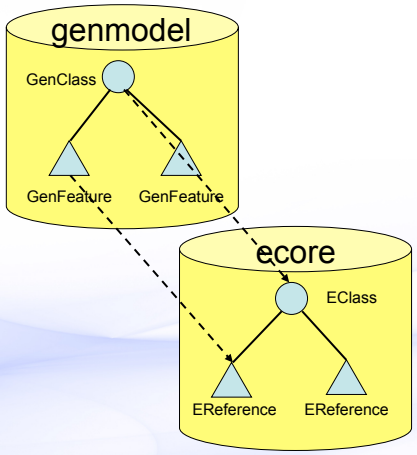
More Generating Code with EMF

- 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*

zeligsoft

The genmodel

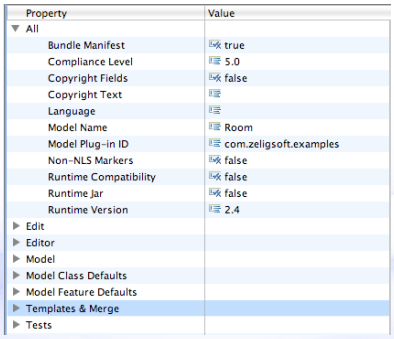
- Wraps the Ecore model and automatically kept in sync
  - Only true for Ecore
  - Others like Java require explicit reload
- Decorates the Ecore model
  - Target location for generated code
  - Prefix for some of the generated classes



zeligsoft

More genmodel

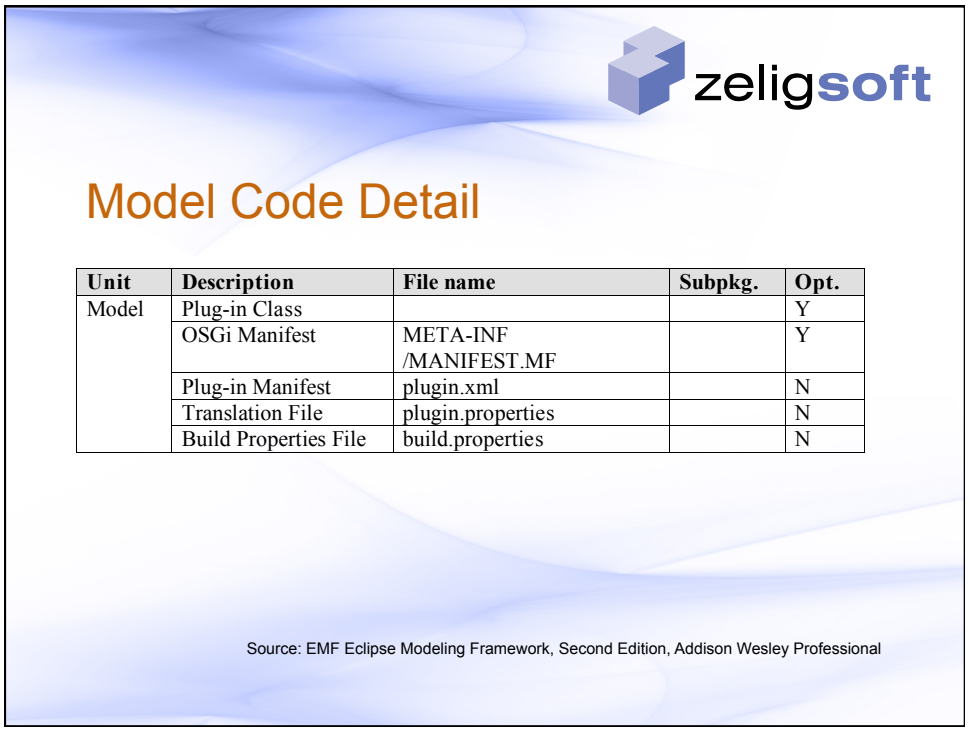
- Global generator configuration
- To generate
  - Context menu in the editor
  - Main menu in the editor
- Generates for
  - GenModel, GenPackage, GenClass and GenEnum
- What is generated
  - Model
  - Edit
  - Editor
  - Tests



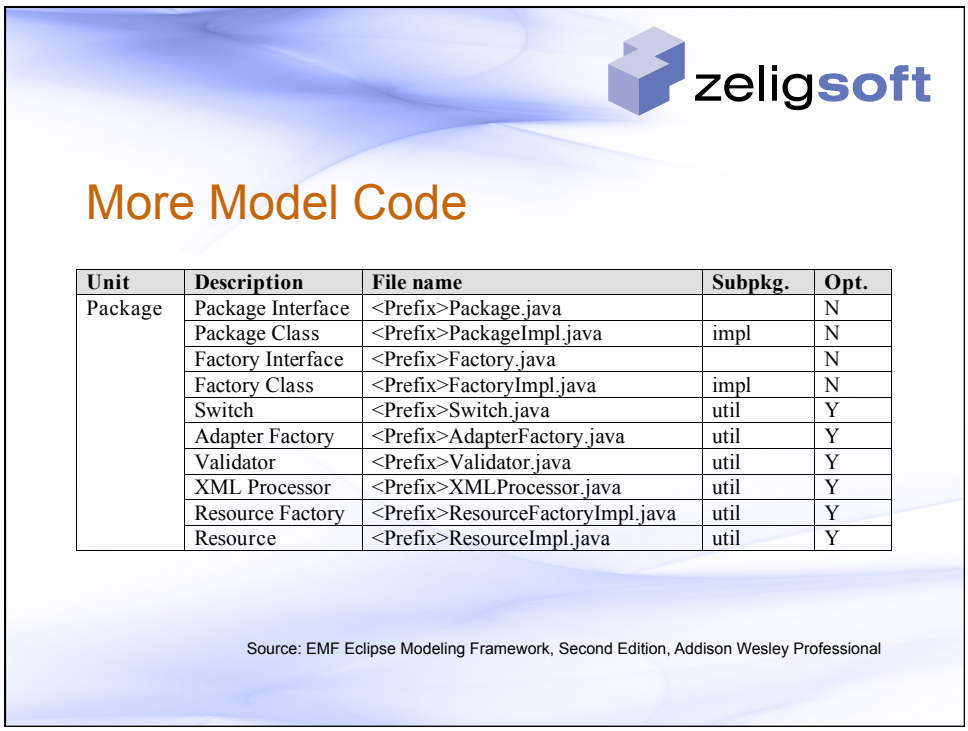
zeligsoft

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



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



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

zeligsoft

More Model Code

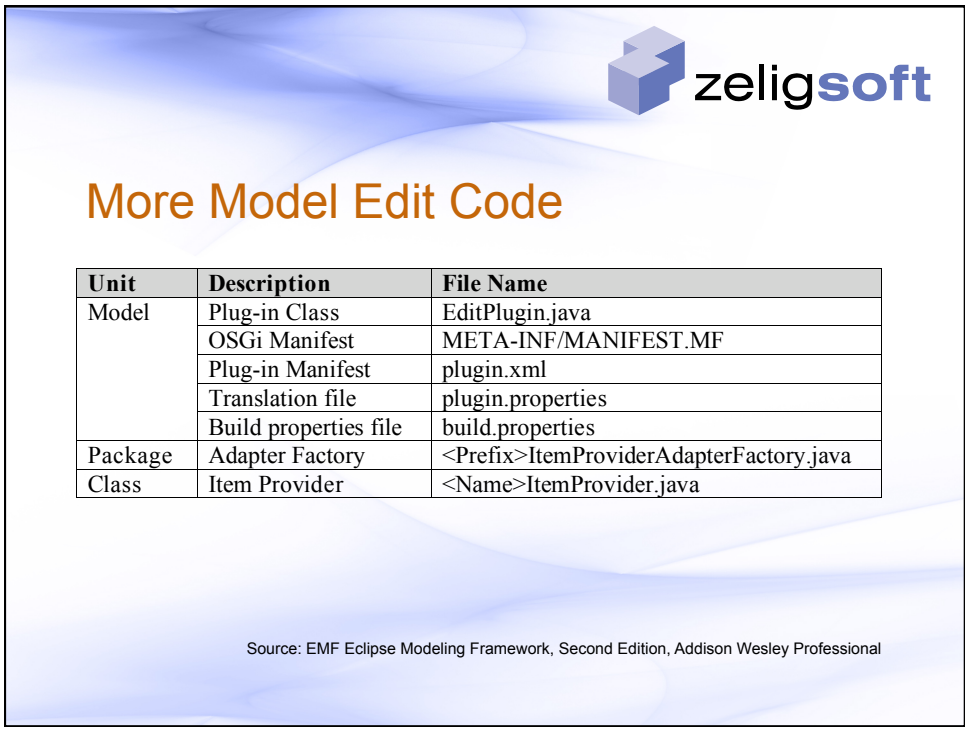
Unit	Description	File name	Subpkg.	Opt.
Class	Interface	<Name>.java		N
	Class	<Name>Impl.java	impl	N
Enum	Enum	<Name>.java		N

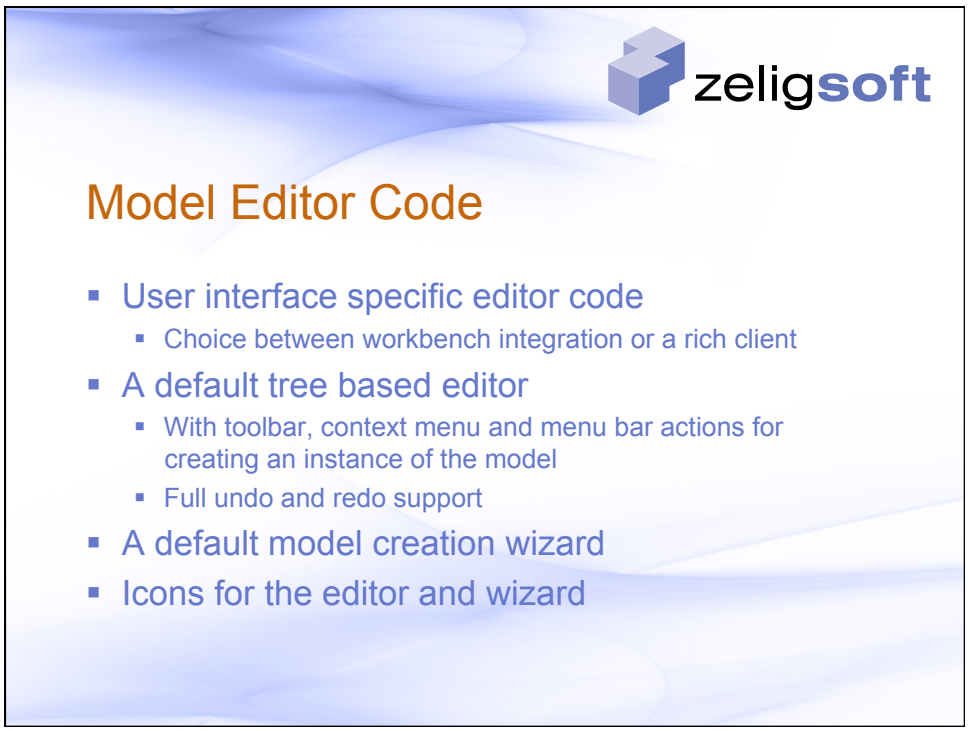
Source: EMF Eclipse Modeling Framework, Second Edition, Addison Wesley Professional

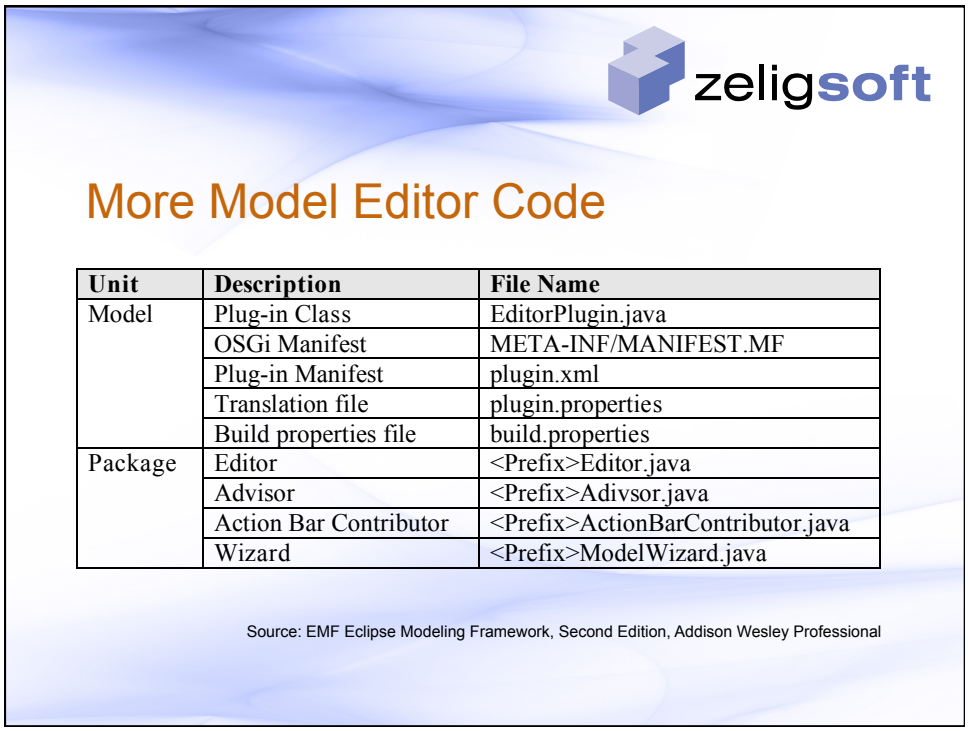
zeligsoft

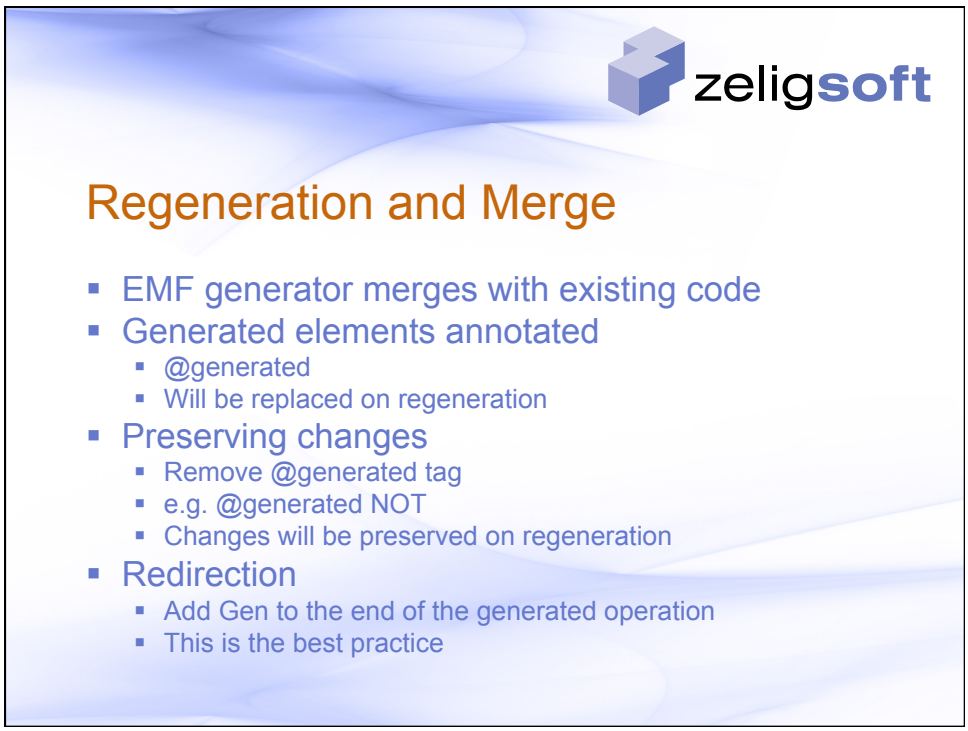
Model Edit Code

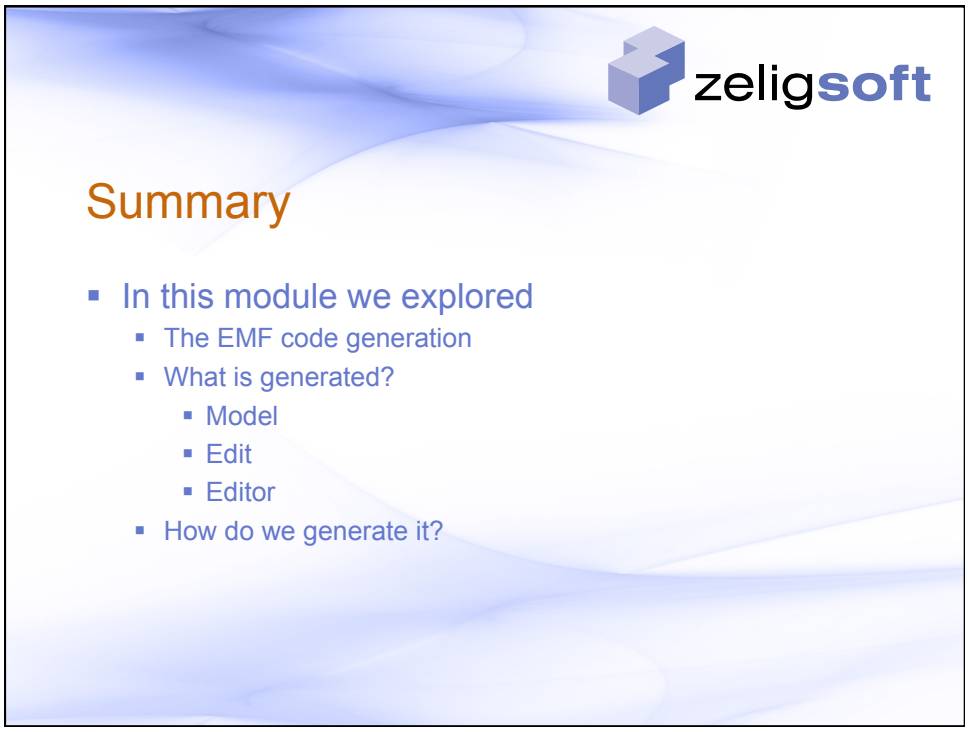
- User interface independent editor code
- Interfaces to support viewing and editing of model objects
  - Content and label provider functions
  - Property descriptors
  - Command factory
  - Forwarding change notifications
- Sample icons are generated

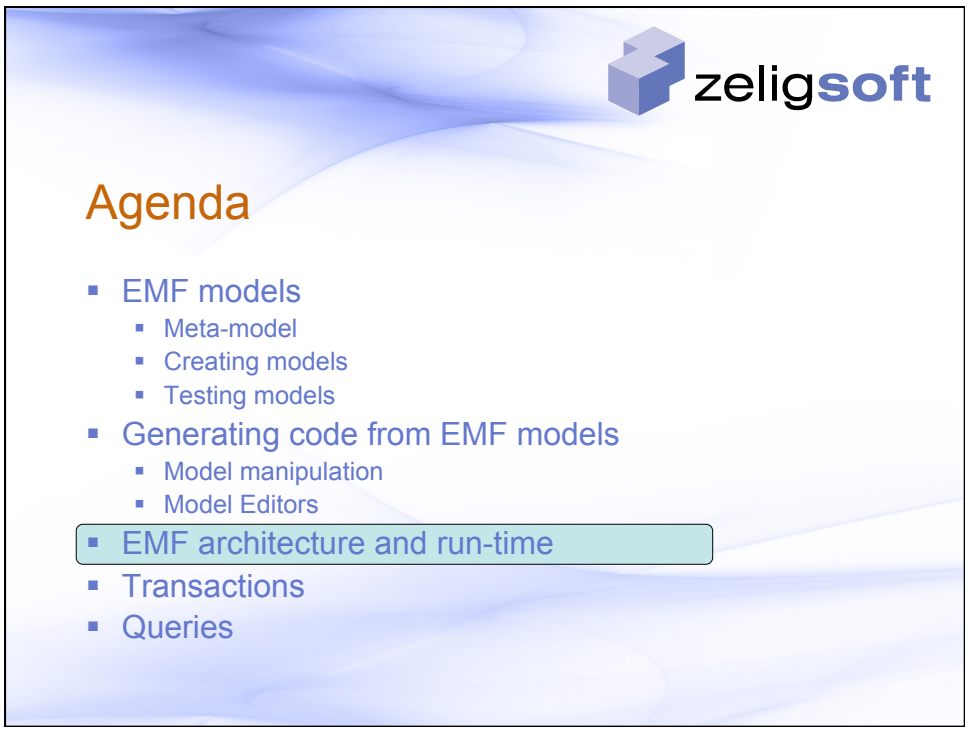

zeligsoft
More Model Edit Code
Unit | Description | File Name
Model | Plug-in Class | EditPlugin.java
OSGi Manifest | META-INF/MANIFEST.MF
Plug-in Manifest | plugin.xml
Translation file | plugin.properties
Build properties file | build.properties
Package | Adapter Factory | <Prefix>ItemProviderAdapterFactory.java
Class | Item Provider | <Name>ItemProvider.java
Source: EMF Eclipse Modeling Framework, Second Edition, Addison Wesley Professional


zeligsoft
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


zeligsoft
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

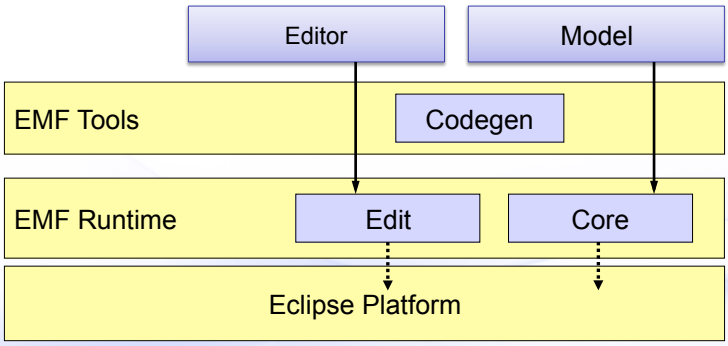

zeligsoft
Regeneration and Merge
EMF generator merges with existing code
Generated elements annotated
@generated
Will be replaced on regeneration
Preserving changes
Remove @generated tag
e.g. @generated NOT
Changes will be preserved on regeneration
Redirection
Add Gen to the end of the generated operation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


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


zeligsoft
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

zeligsoft

EMF Architecture

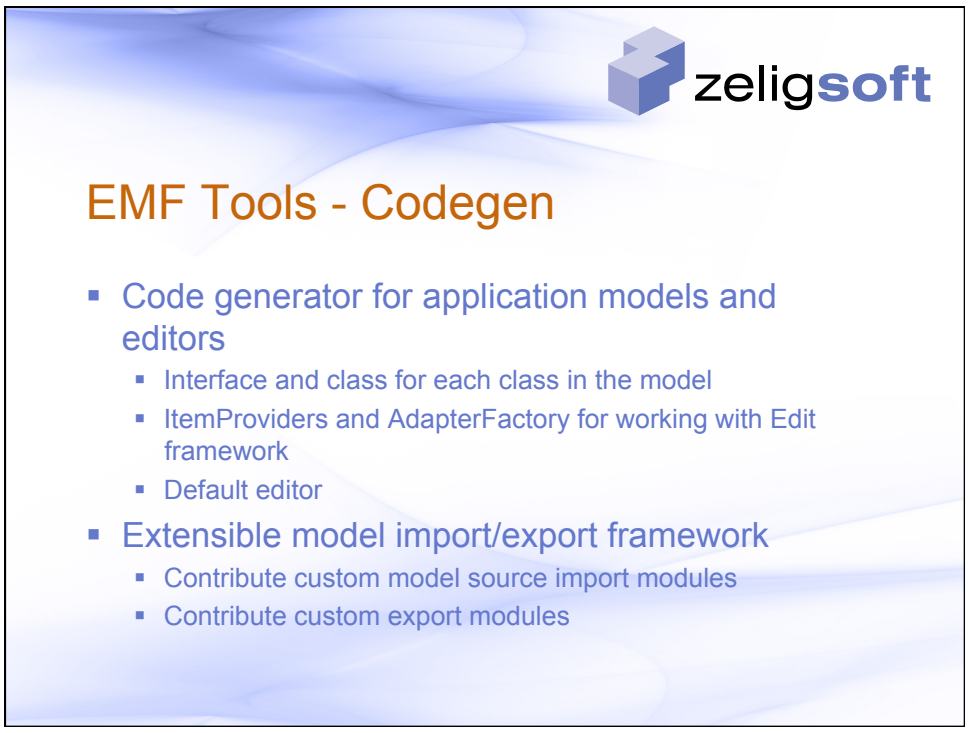


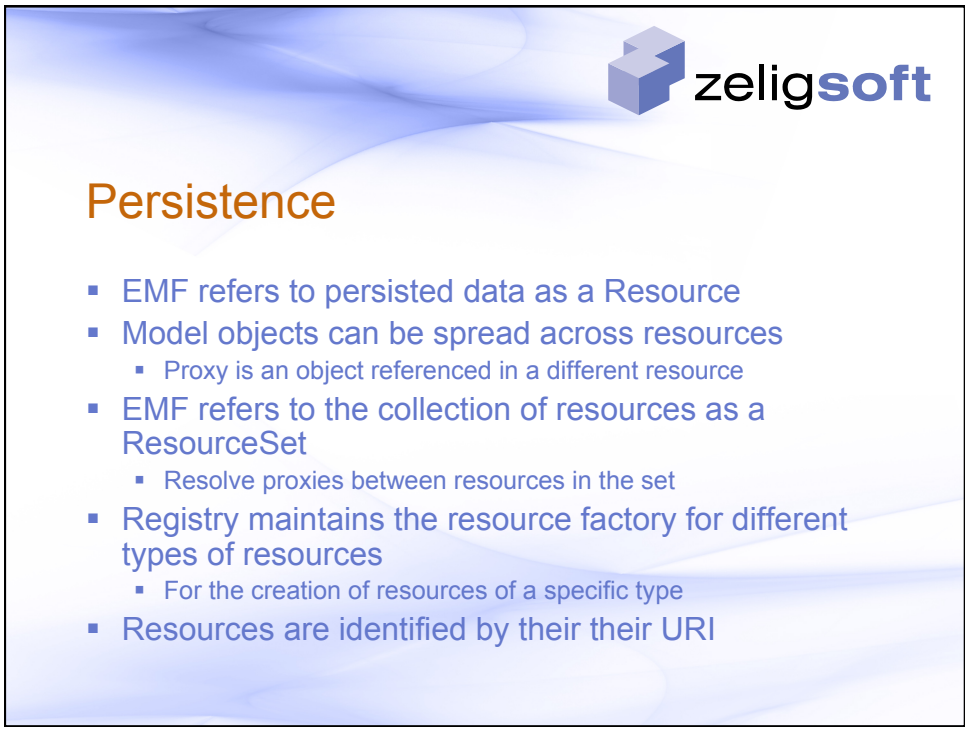
zeligsoft

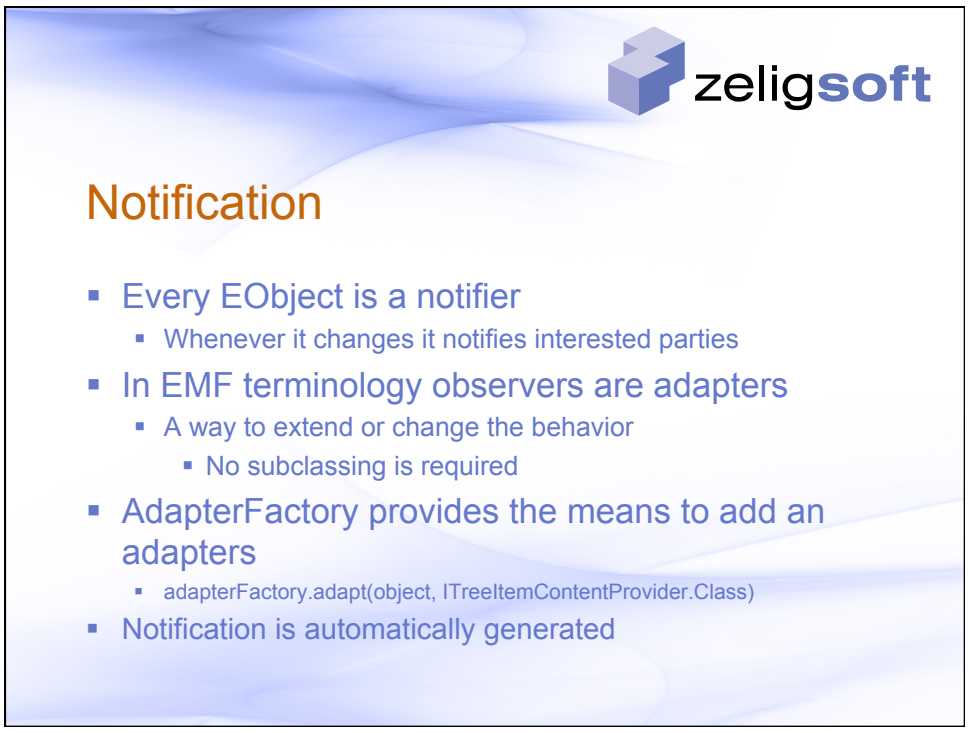
EMF Runtime - Core

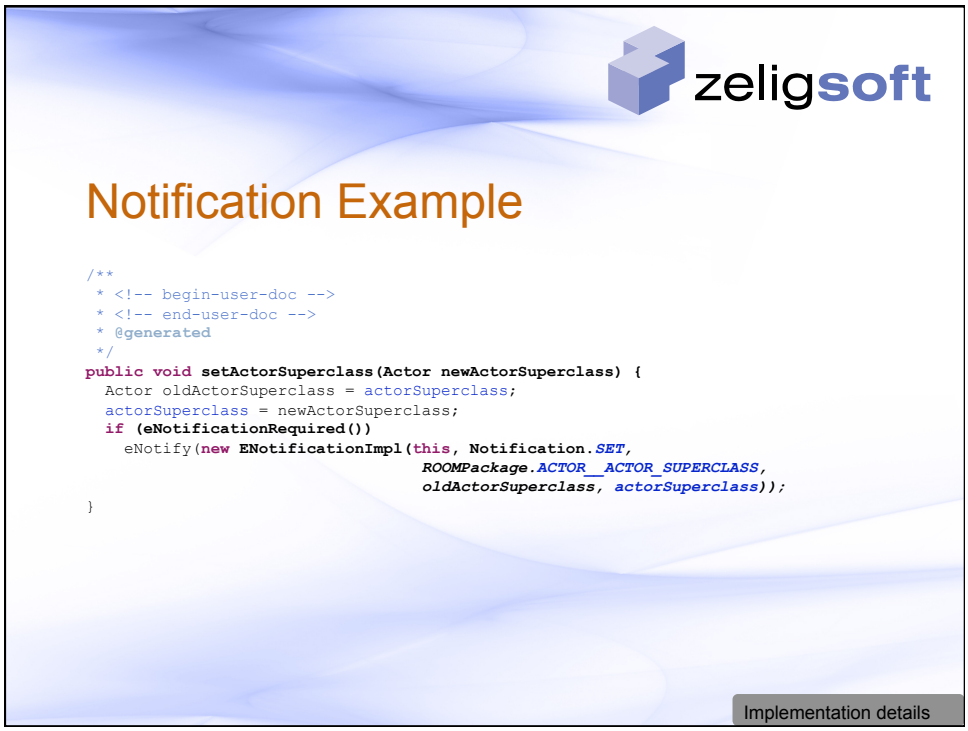
- Notification framework
- Ecore metamodel
- Persistence
- Validation
- Change model

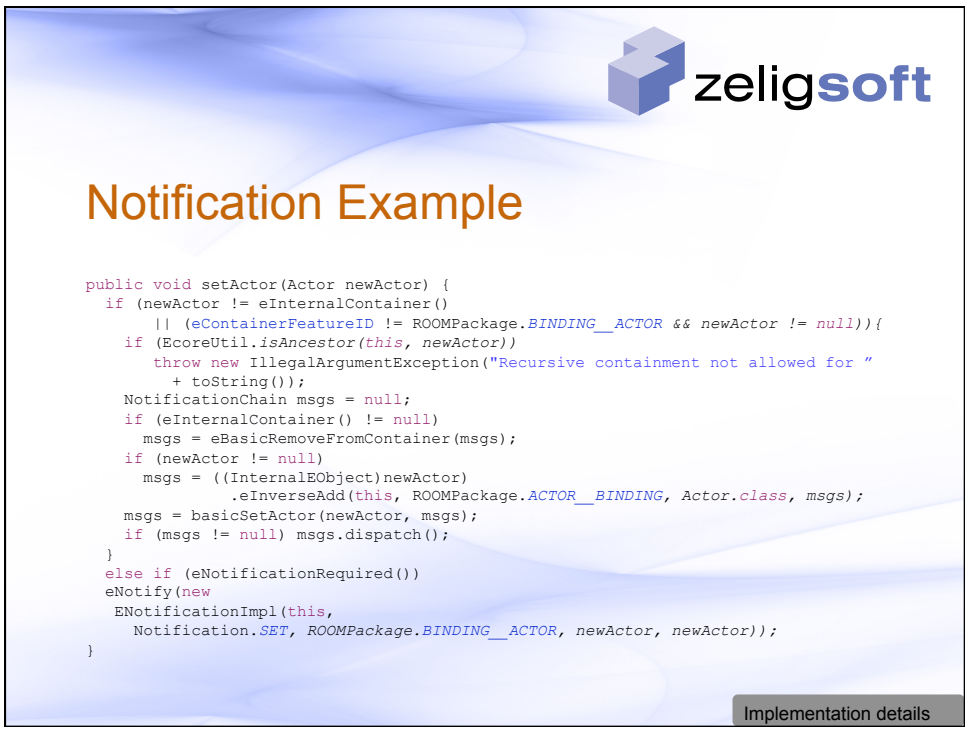

zeligsoft
EMF Runtime - Edit
Support for model-based editors and viewers
Notification framework
Sends out notification whenever attribute or reference is changed
Observers (also an adapter) receive notification and can act
Default reflective editor


zeligsoft
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

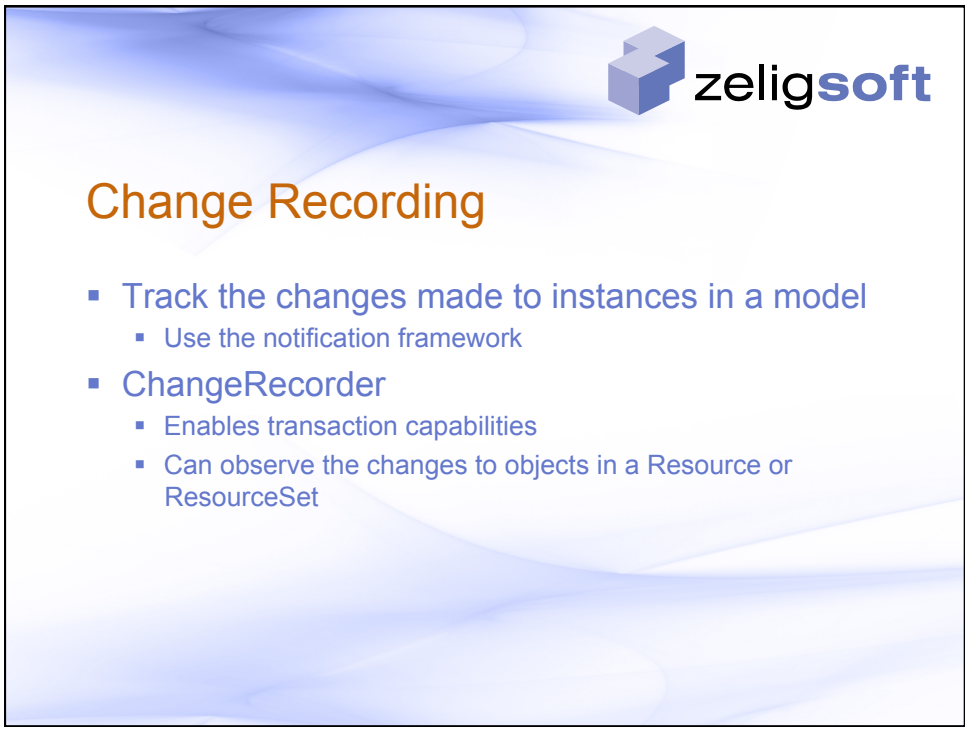

zeligsoft
Persistence
EMF refers to persisted data as a Resource
Model objects can be spread across resources
Proxy is an object referenced in a different resource
EMF refers to the collection of resources as a ResourceSet
Resolve proxies between resources in the set
Registry maintains the resource factory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

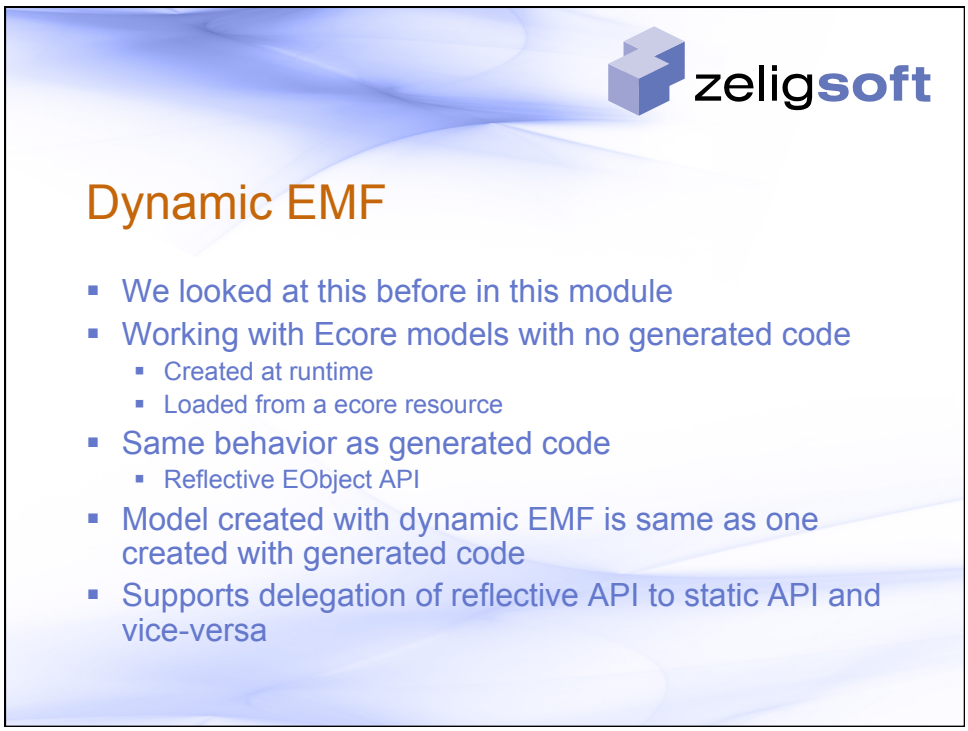

zeligsoft
Notification
Every EObject is a notifier
Whenever it changes it notifies interested parties
In EMF terminology observers are adapters
A way to extend or change the behavior
No subclassing is required
AdapterFactory provides the means to add an adapters
adapterFactory.adapt(object, ITreeItemContentProvider.Class)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

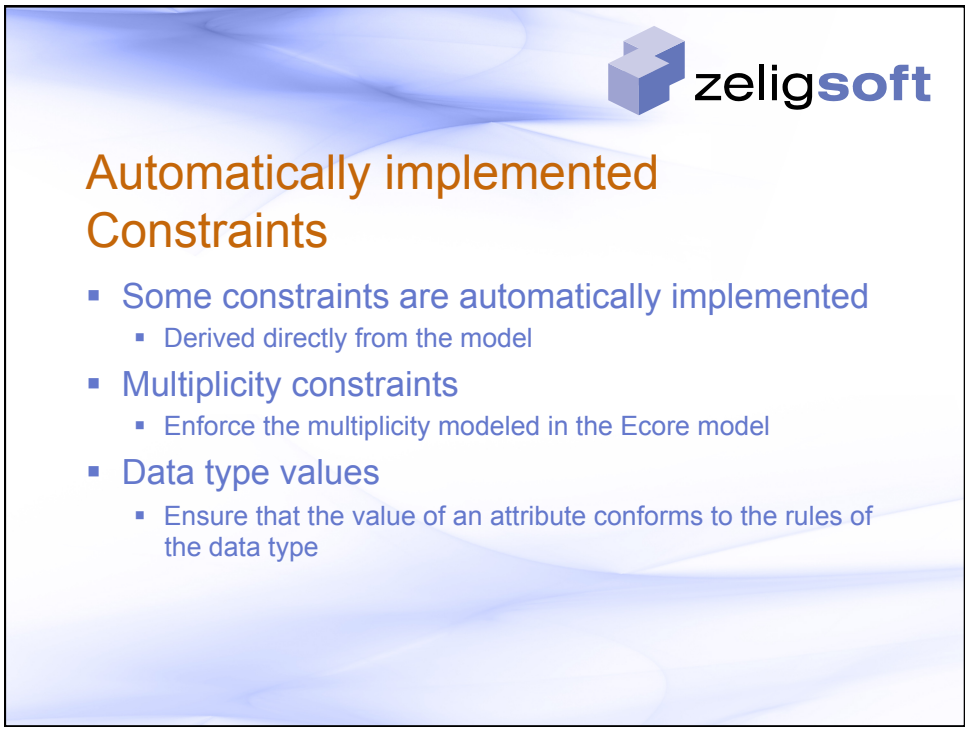




Toby McClean, Zeligsoft, 2008


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

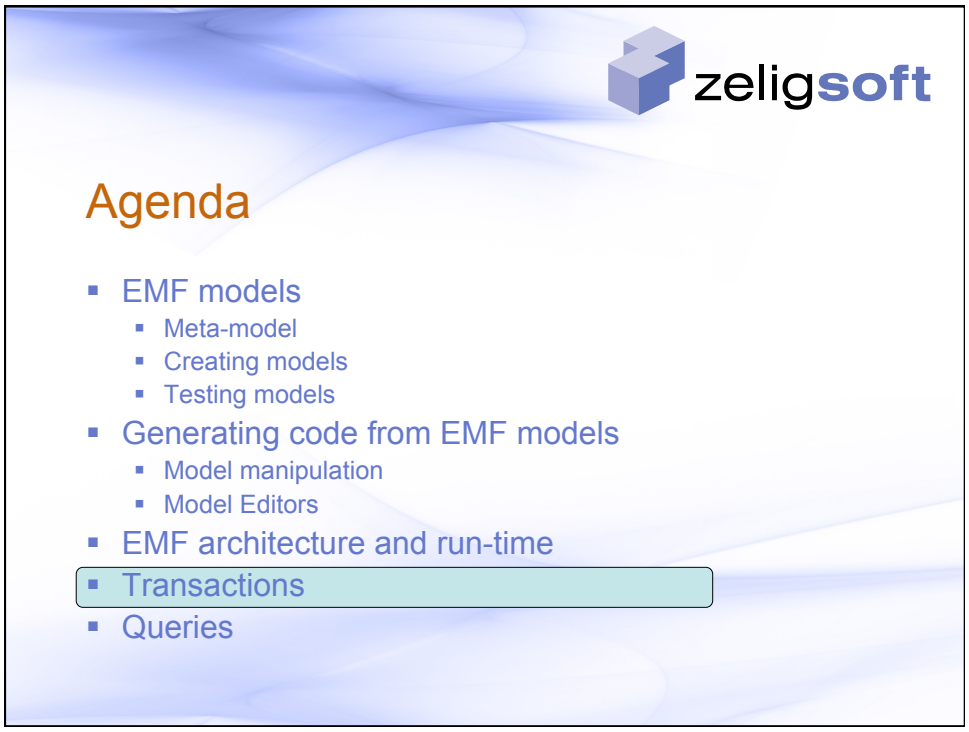

zeligsoft
Dynamic EMF
We looked at this before in this module
Working with Ecore models with no generated code
Created at runtime
Loaded from a ecore resource
Same behavior as generated code
Reflective EObject API
Model created with dynamic EMF is same as one created with generated code
Supports delegation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


zeligsoft
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

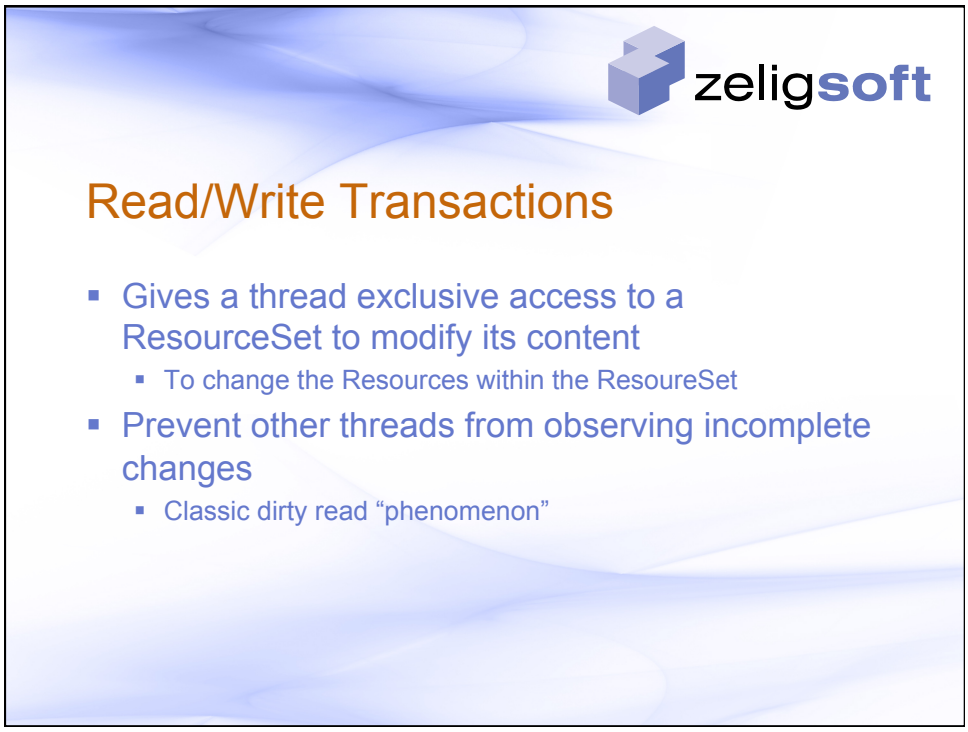

zeligsoft
Validation
Infrastructure for providing rich invariants and constraints on the model
Will invoke the invariants and constraints
Requires the user to write the invariants and constraints
Invariants
Defined by operations on a class with signature
(EDiagnosticChain, EMap) : EBoolean
Constraints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

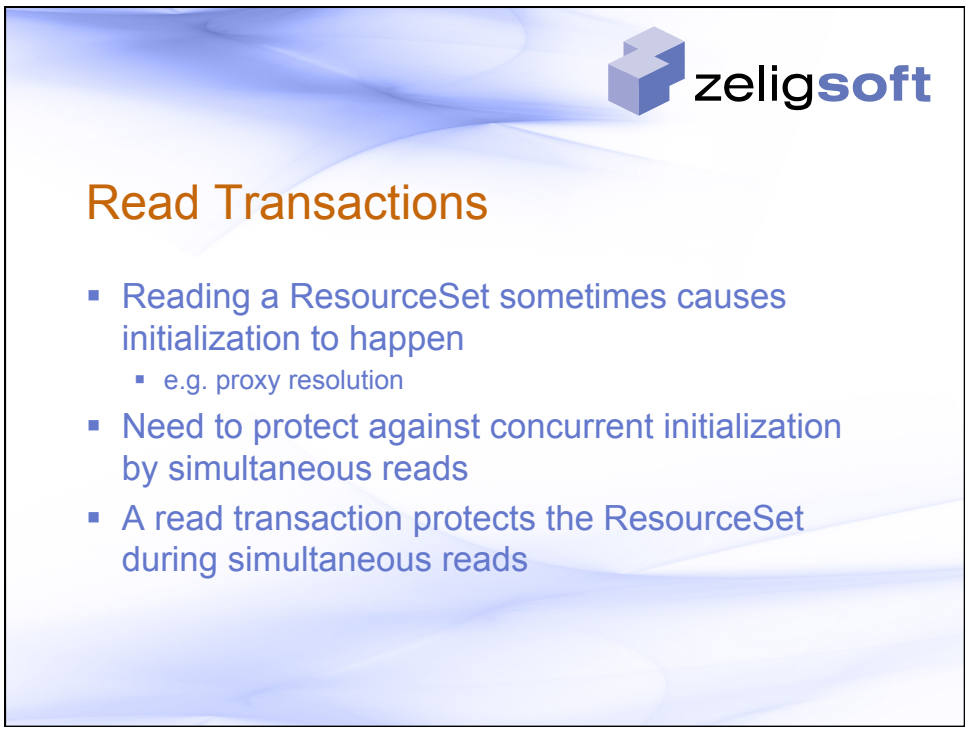

zeligsoft
EMF Utilities
Copying
EcoreUtil.copy
Equality
EcoreUtil.equal
Cross-referencing, contents/container navigation, annotation, proxy resolution, adapter selection, …


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


zeligsoft
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

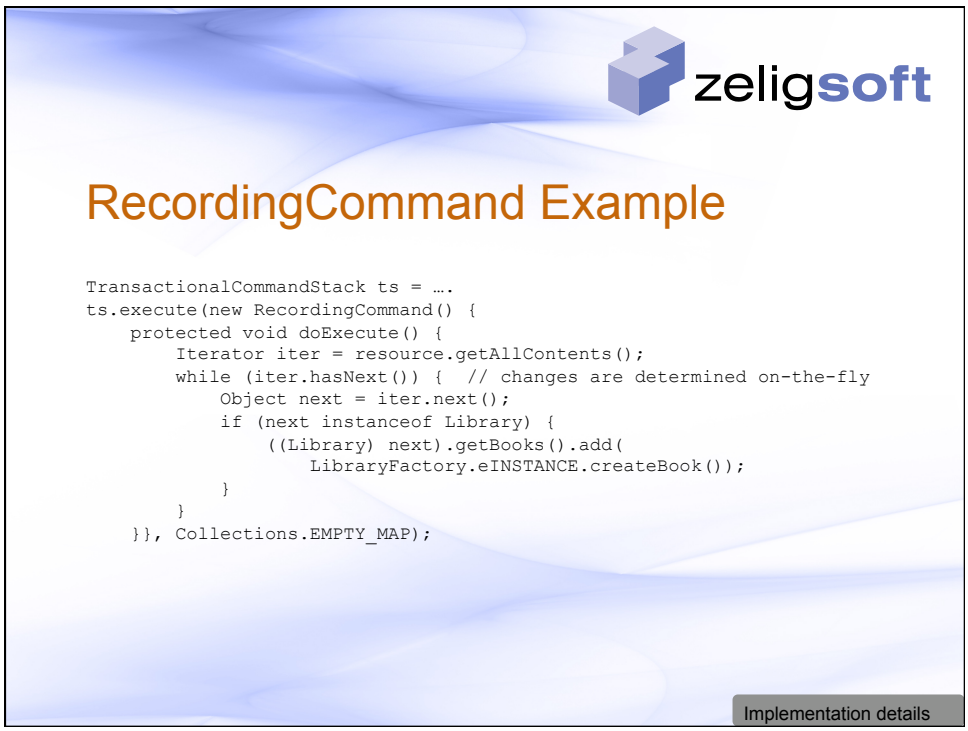

zeligsoft
EMF Transaction
A framework for managing multiple readers and writers to EMF resources or sets of resources
An editing domain object manages access to resources
Can be shared amongst applications
A transaction object is the unit of work
Support for rolling back changes
Support for workbench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


zeligsoft
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”

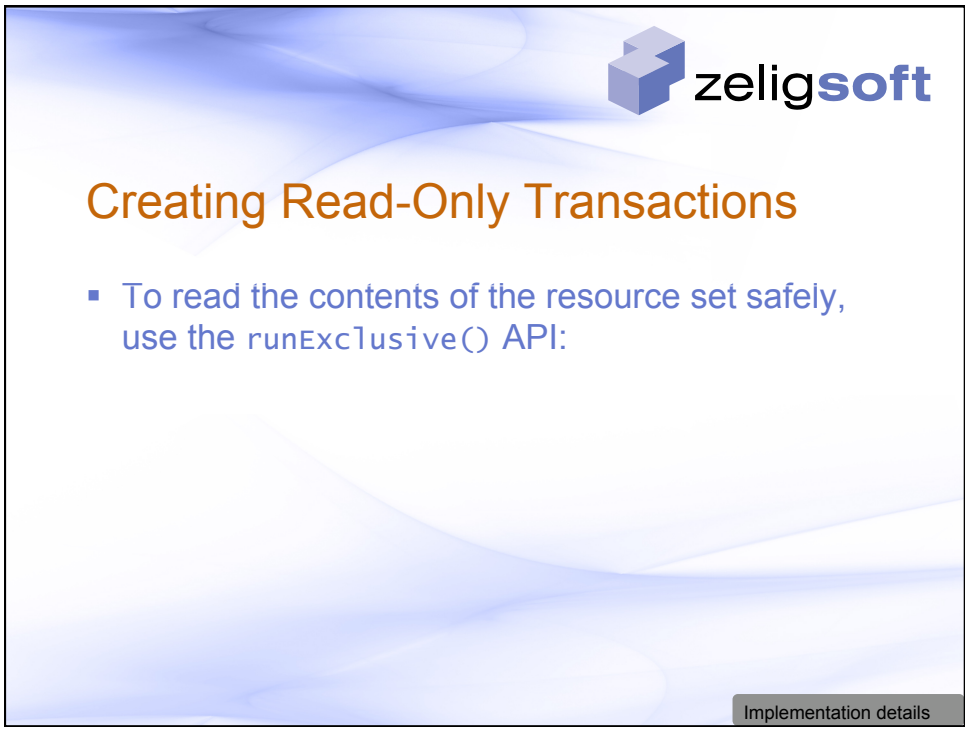

zeligsoft
Read Transactions
Reading a ResourceSet sometimes causes initialization to happen
e.g. proxy resolution
Need to protect against concurrent initialization by simultaneous reads
A read transaction protects the ResourceSet during simultaneous reads

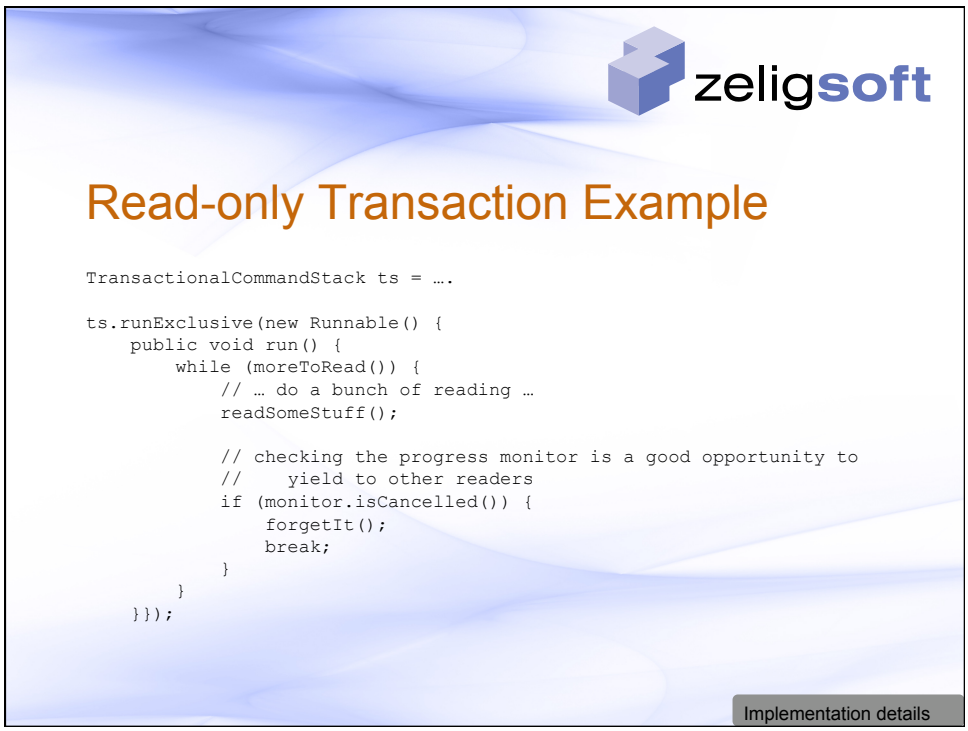

zeligsoft
Change Events
When a transaction is committed change notifications are sent to registered listeners
Includes a summary of changes
Successful commits send notifications as a batch
Prevents listeners from being overwhelmed
Has its quirks, too. Most notable is its asynchronous nature: the object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


zeligsoft
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

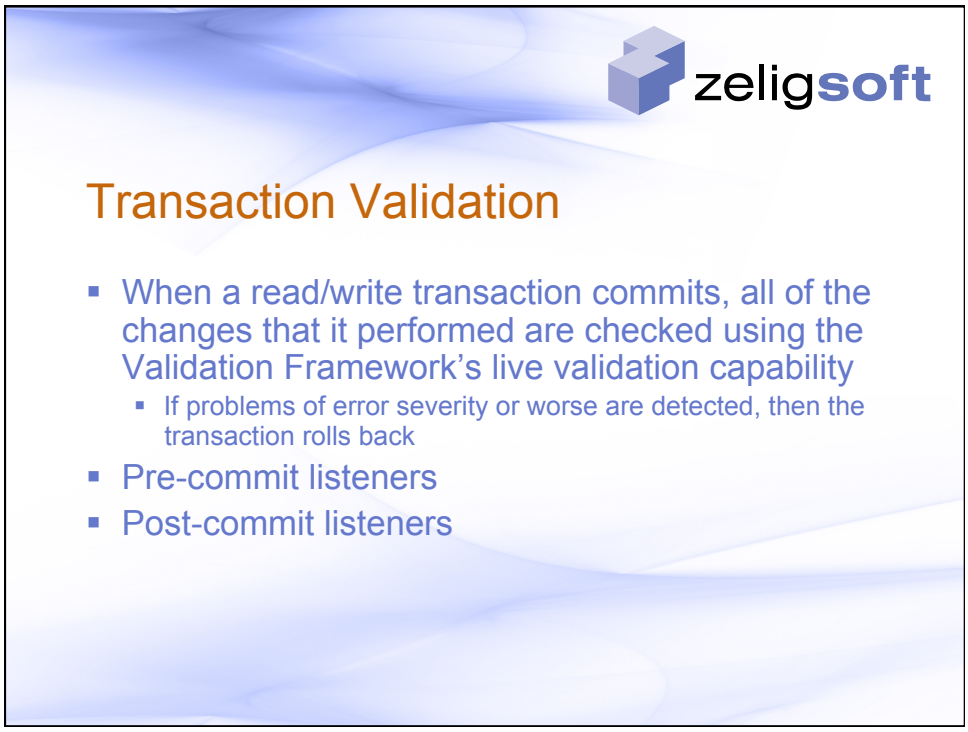


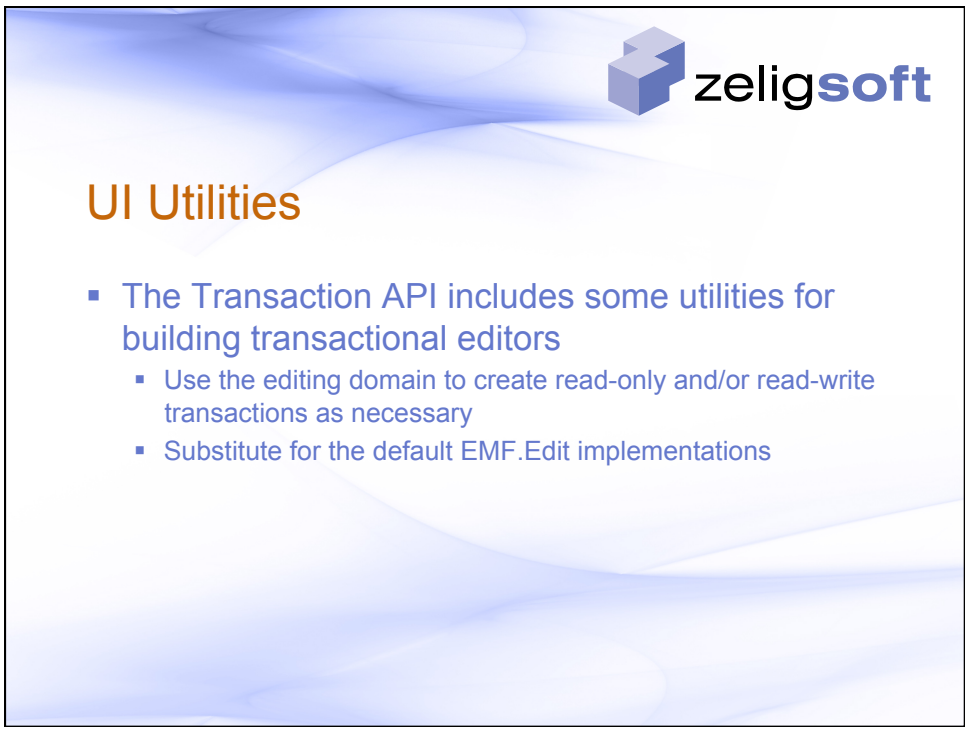
Toby McClean, Zeligsoft, 2008





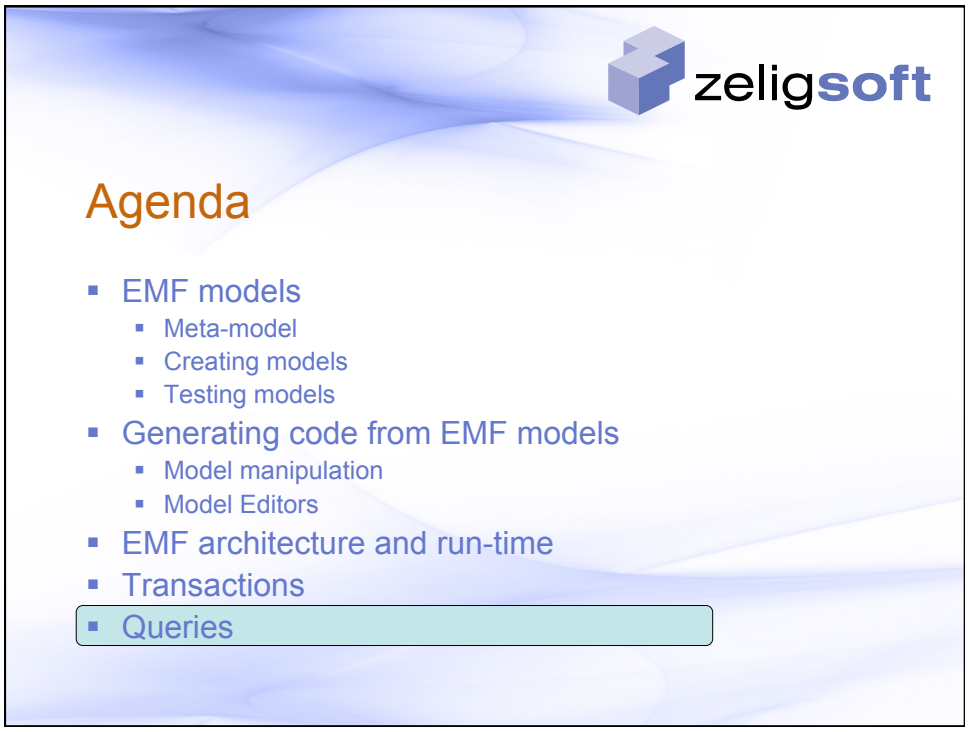
Toby McClean, Zeligsoft, 2008

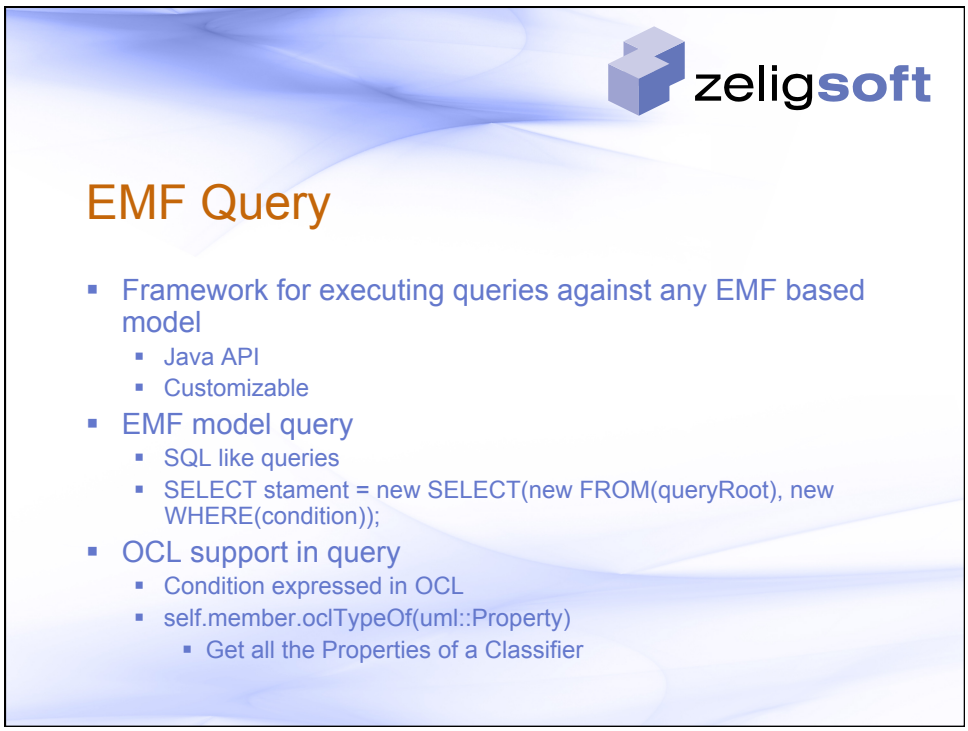

zeligsoft
Transaction Validation
When a read/write transaction commits, all of the changes that it performed are checked using the Validation Framework's live validation capability
If problems of error severity or worse are detected, then the transaction rolls back
Pre-commit listeners
Post-commit listeners

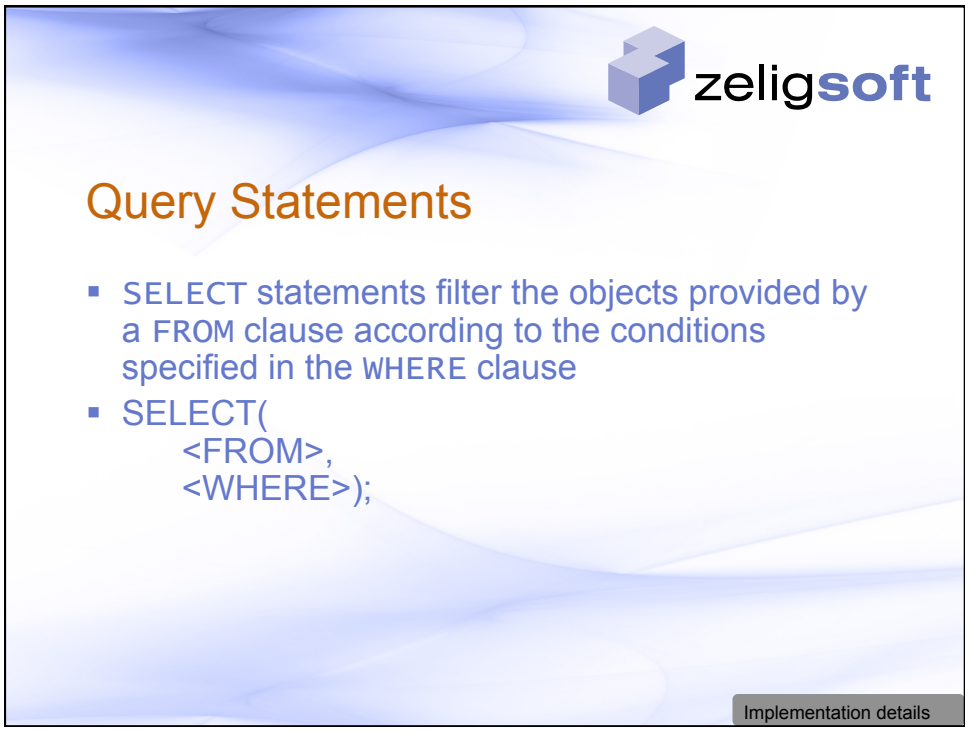

zeligsoft
UI Utilities
The Transaction API includes some utilities for building transactional editors
Use the editing domain to create read-only and/or read-write transactions as necessary
Substitute for the default EMF.Edit implementations

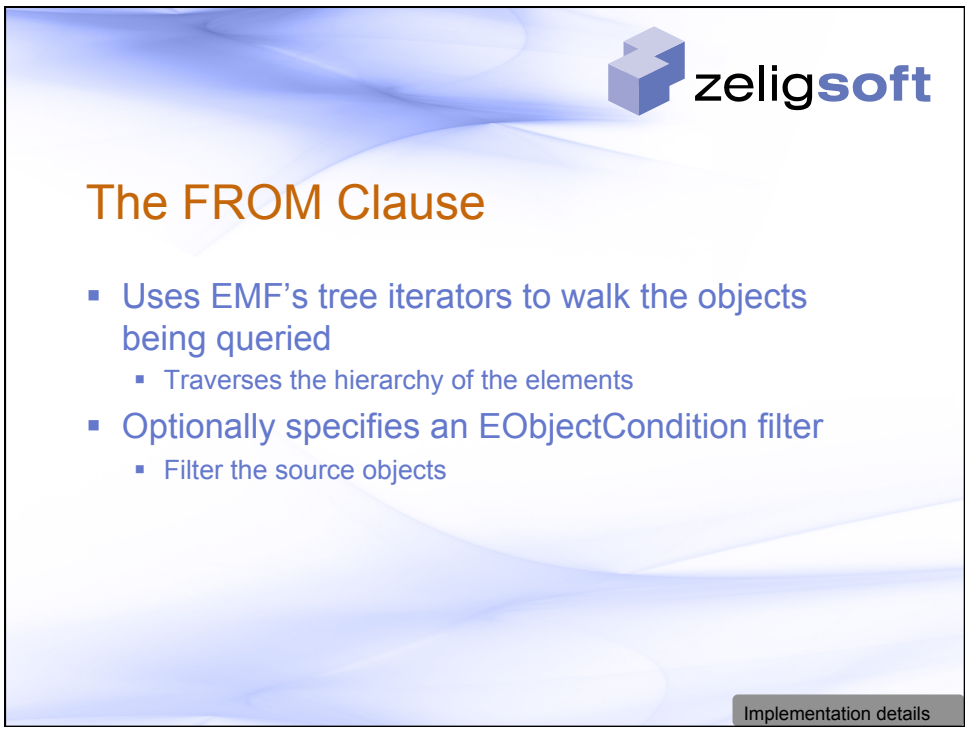

zeligsoft
ResourceSetListener
Avoid reacting to changes before they have been committed
If a transaction rolls back, no event is sent
Except for changes that are not undoable, such as proxy resolution and resource loading. In general, these are the changes that are compatible with a read-only transaction

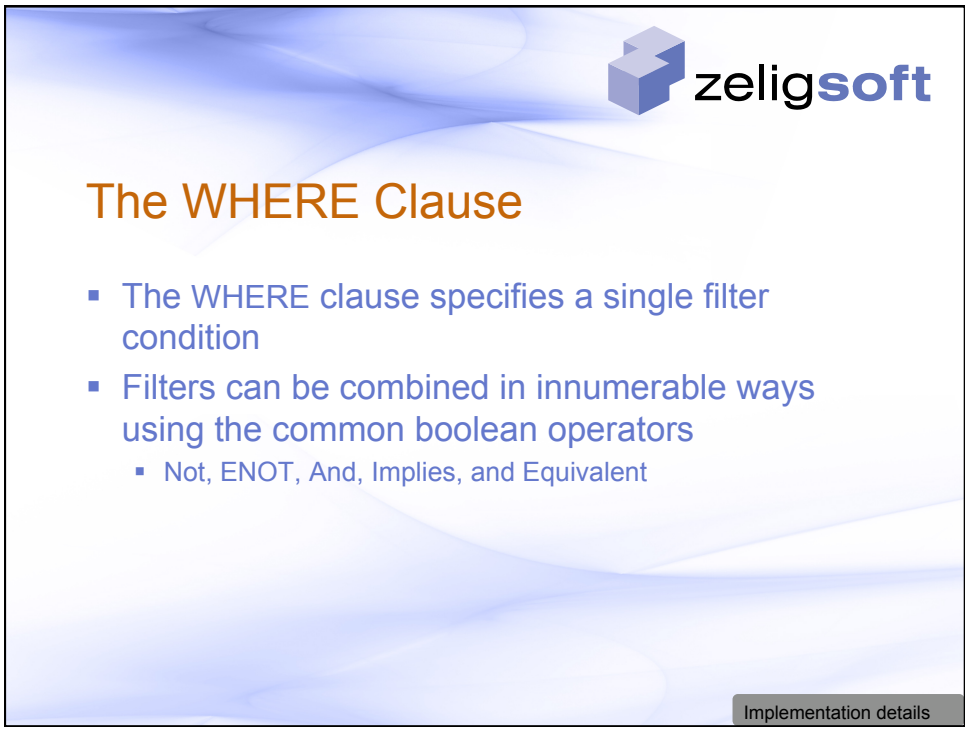

zeligsoft
Summary
In this module we have discussed
EMF transactions and how to respond to their changes

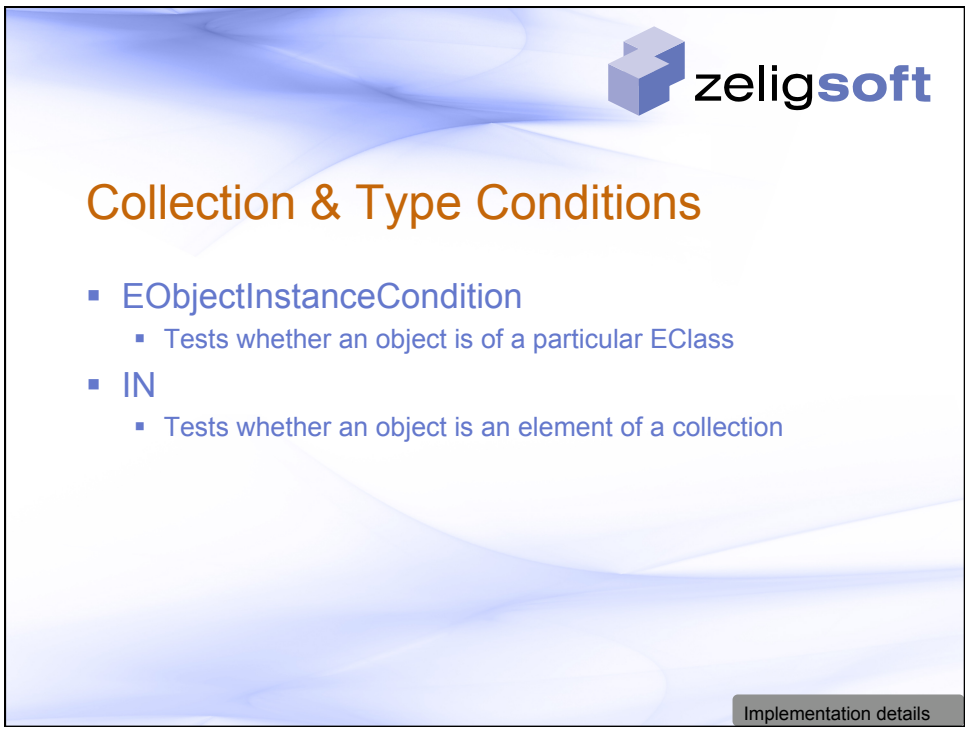

zeligsoft
Agenda
EMF models
Meta-model
Creating models
Testing models
Generating code from EMF models
Model manipulation
Model Editors
EMF architecture and run-time
Transactions
Queries


zeligsoft
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

zeligsoft

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

zeligsoft

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

zeligsoft

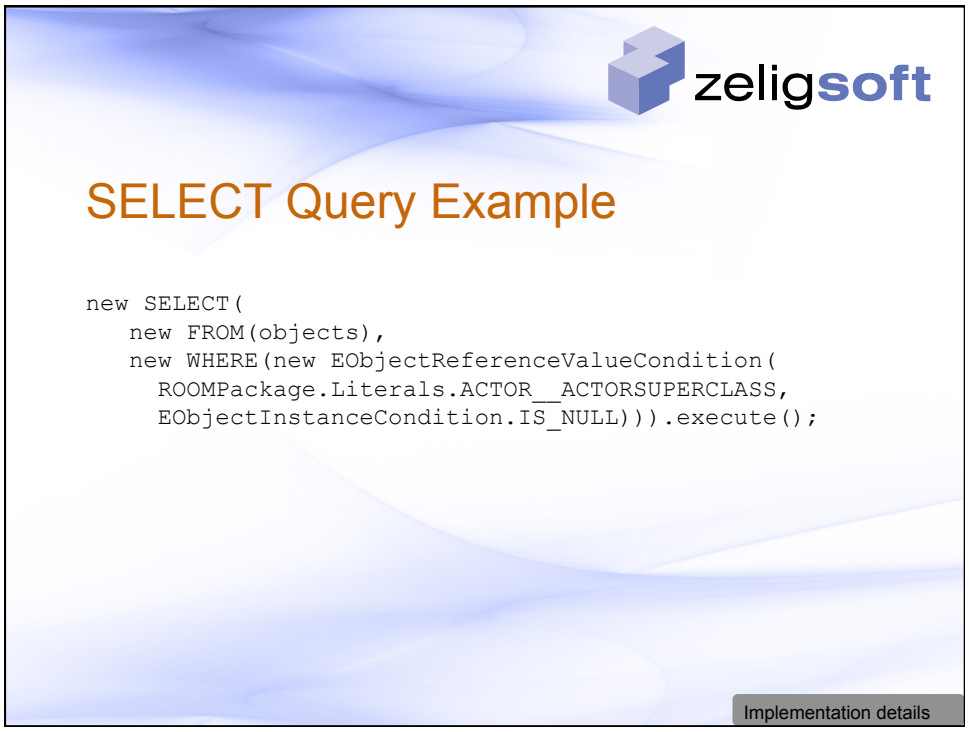
OCL Conditions

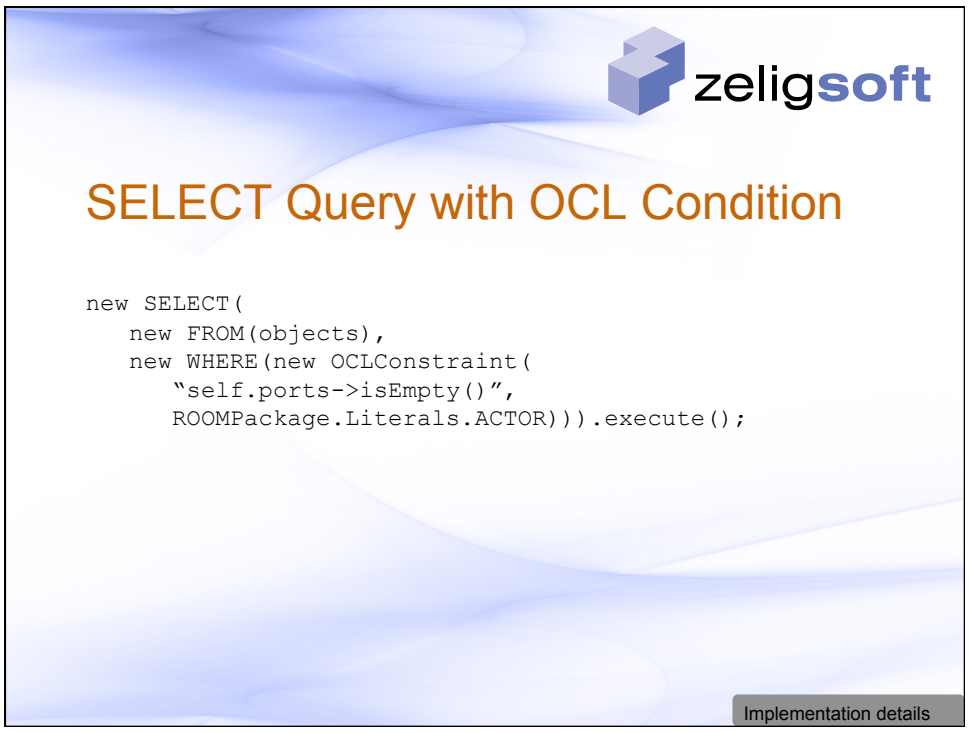
- OCL can be used to specify `WHERE` clause conditions
  - OCLConstraintCondition
- Specifies a boolean-valued expression (i.e., a constraint) that selects those elements for which the expression is satisfied
- Requires that the MDT OCL component

zeligsoft

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





Toby McClean, Zeligsoft, 2008

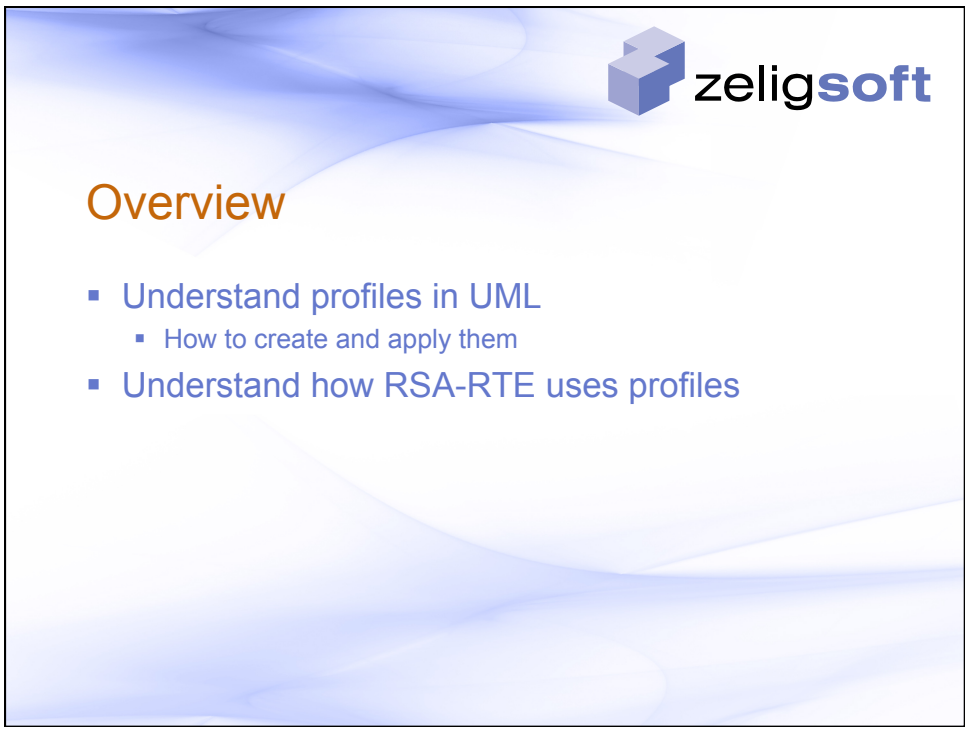
zeligsoft

Summary

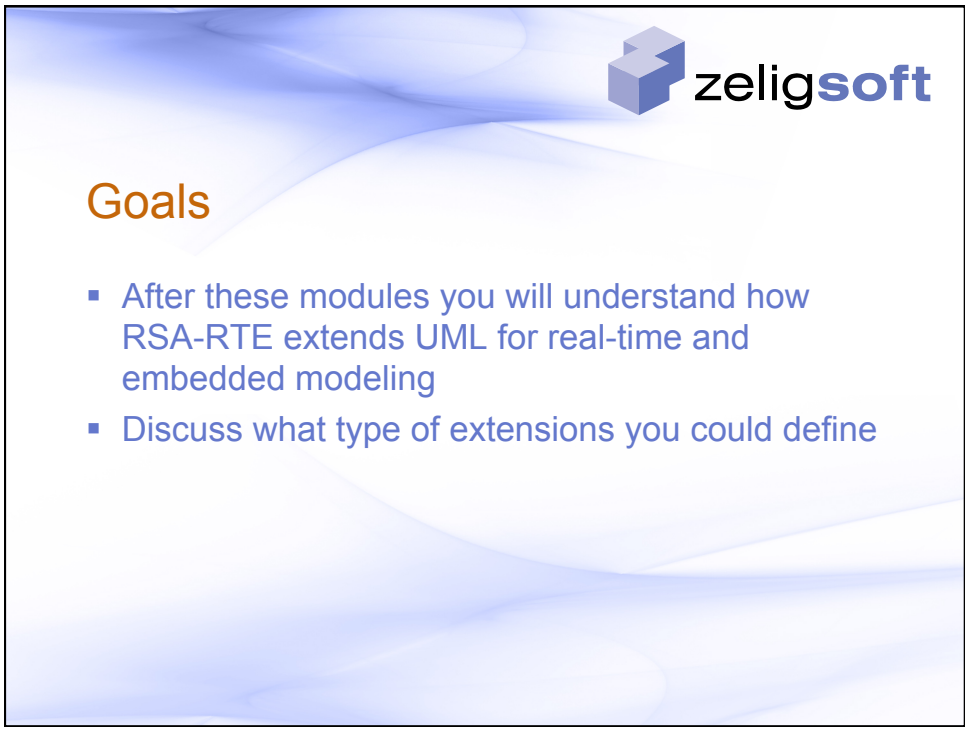
- In this module we explored
  - How to query EMF models

zeligsoft

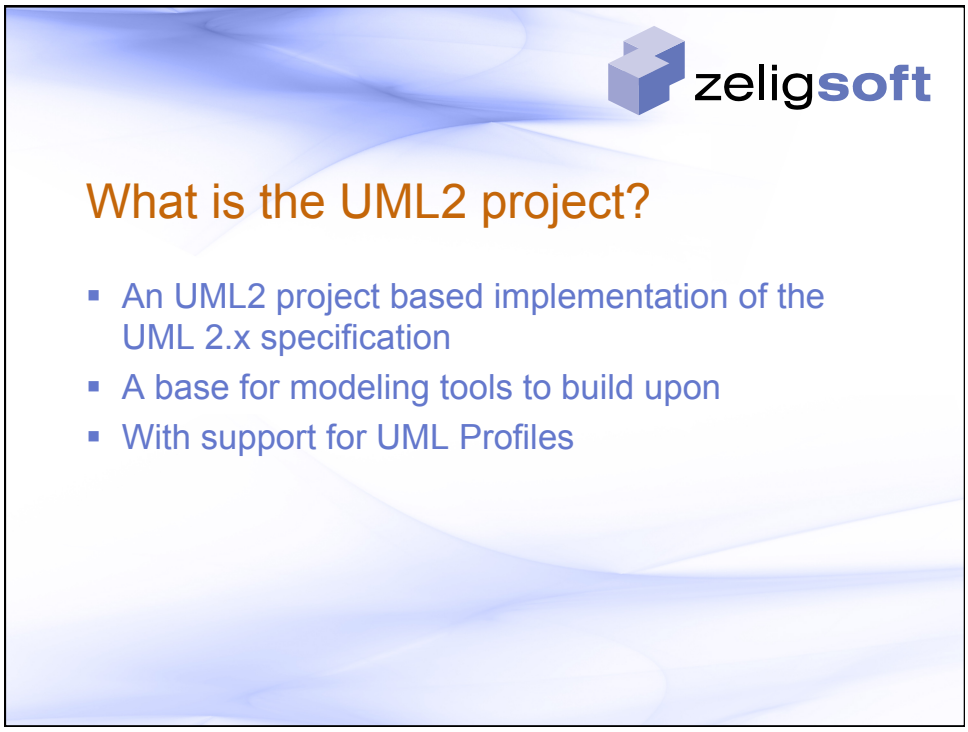
Questions

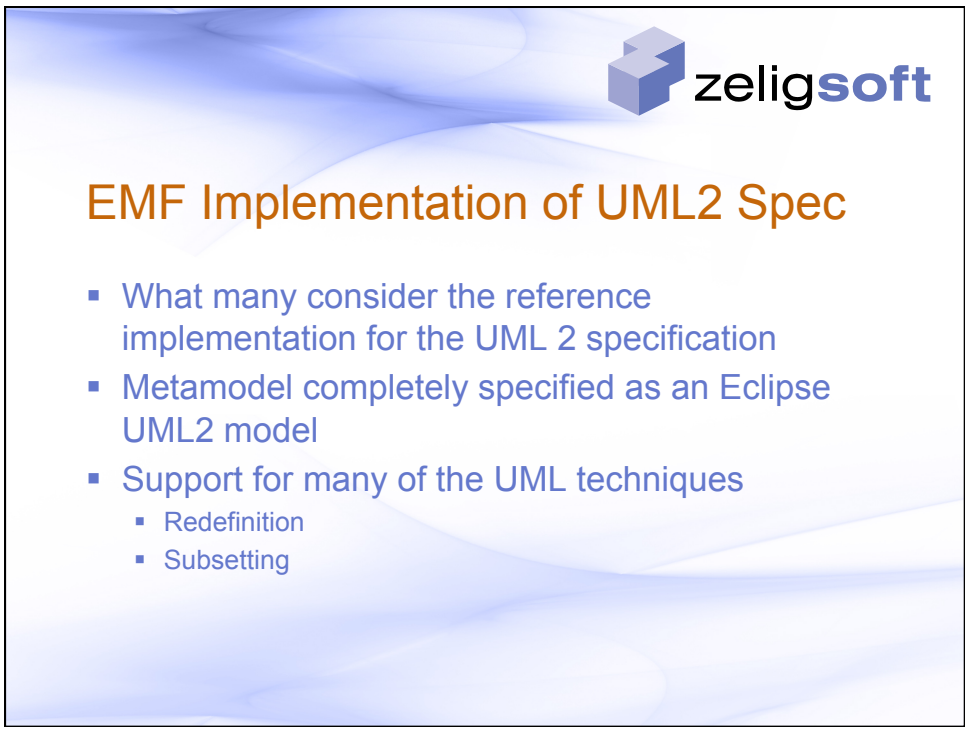


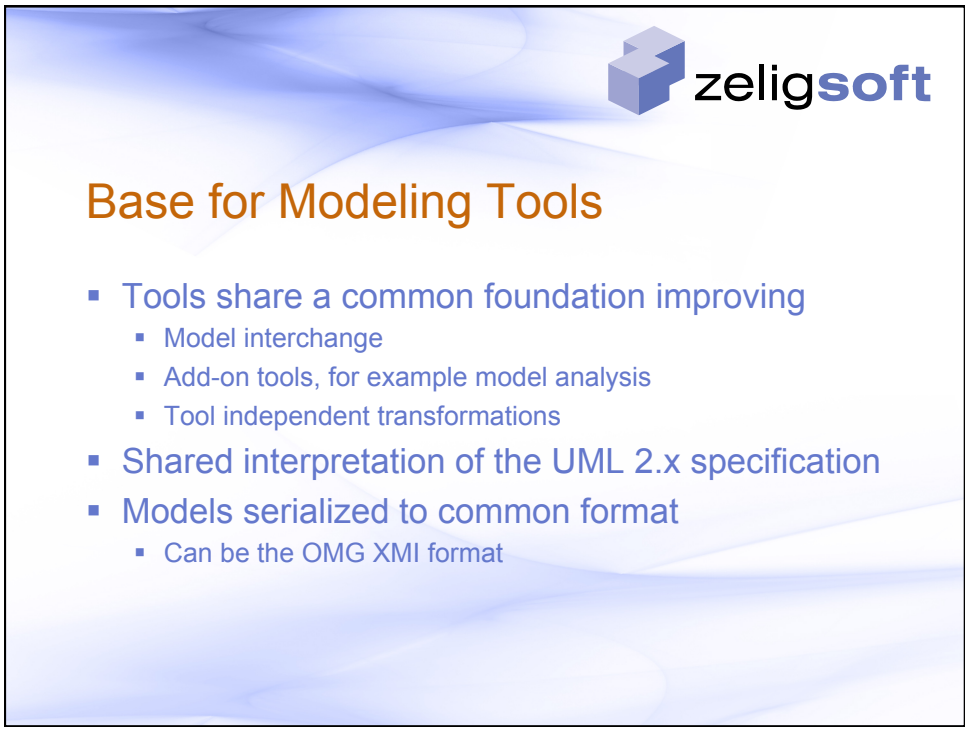
Toby McClean, Zeligsoft, 2008

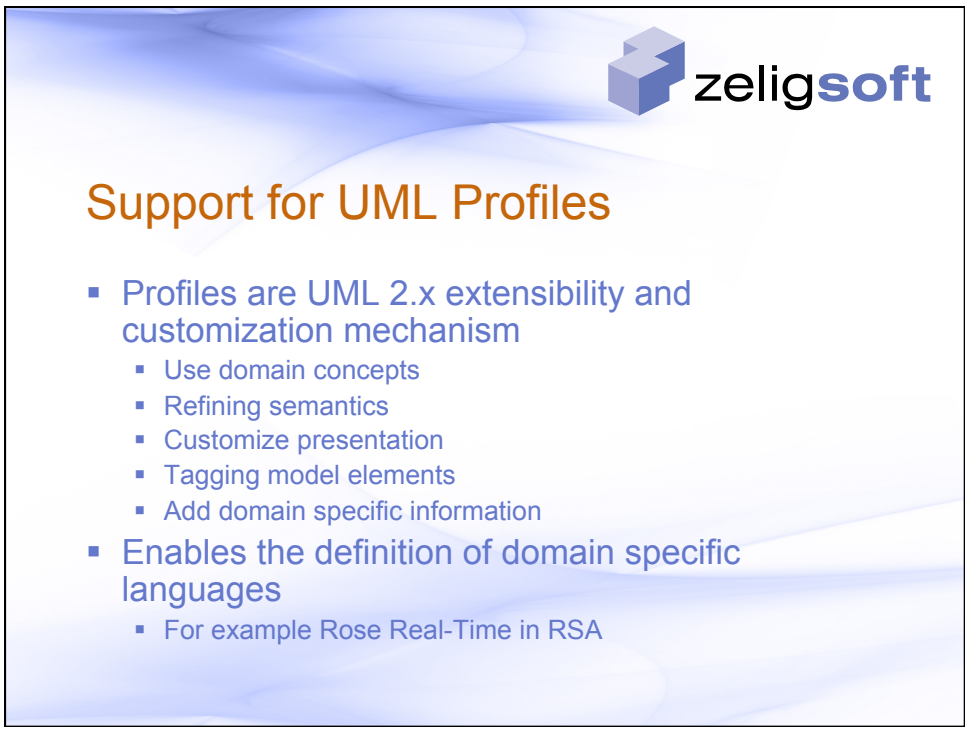

zeligsoft
Goals
After these modules you will understand how RSA-RTE extends UML for real-time and embedded modeling
Discuss what type of extensions you could define


zeligsoft
Agenda
UML2 models
Extending UML2 models
Keywords, profiles, stereotypes
RSA-RTE and profiles


zeligsoft
What is the UML2 project?
An UML2 project based implementation of the UML 2.x specification
A base for modeling tools to build upon
With support for UML Profiles


zeligsoft
EMF Implementation of UML2 Spec
What many consider the reference implementation for the UML 2 specification
Metamodel completely specified as an Eclipse UML2 model
Support for many of the UML techniques
Redefinition
Subsetting


zeligsoft
Base for Modeling Tools
Tools share a common foundation improving
Model interchange
Add-on tools, for example model analysis
Tool independent transformations
Shared interpretation of the UML 2.x specification
Models serialized to common format
Can be the OMG XMI format


zeligsoft
Support for UML Profiles
Profiles are UML 2.x extensibility and customization mechanism
Use domain concepts
Refining semantics
Customize presentation
Tagging model elements
Add domain specific information
Enables the definition of domain specific languages
For example Rose Real-Time in RSA

zeligsoft

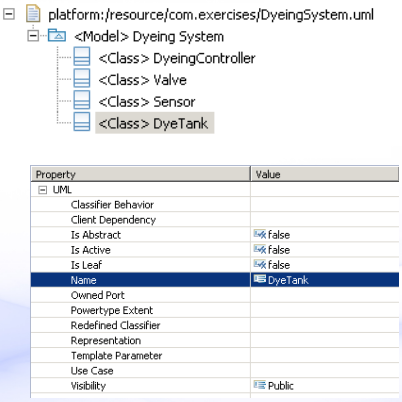
Creating a UML model

- Default editor
  - Tree based editor
- Rational Modeling Platform
  - Visual modeling
- Programmatically
  - Use the Factory for the UML 2 model

zeligsoft

UML Model Editor

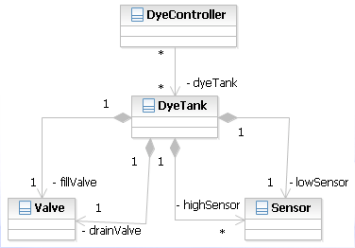
- **File → New...** from the main menu
- Select UML Model
  - Under Example EMF Model Creation Wizards
- Provide a name
- Set Model Object to Model
- Open with UML Model Editor
- Use Create Child in the context menu to create model elements



zeligsoft

Rational Modeling Platform

- Create a UML 2 model graphically
- How to
  - File → New... from the main menu
  - UML Model
    - Under Modeling
  - Set the model name
- Create elements using diagrams and project explorer
- Created model can be opened in the UML Model Editor

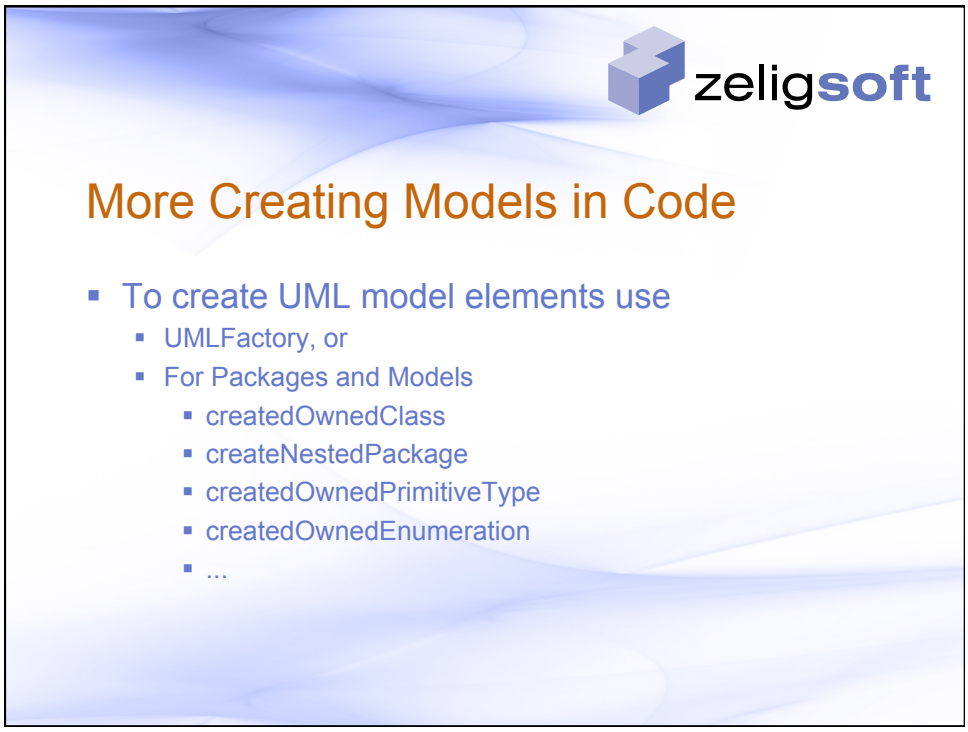


zeligsoft

Creating Models in Code

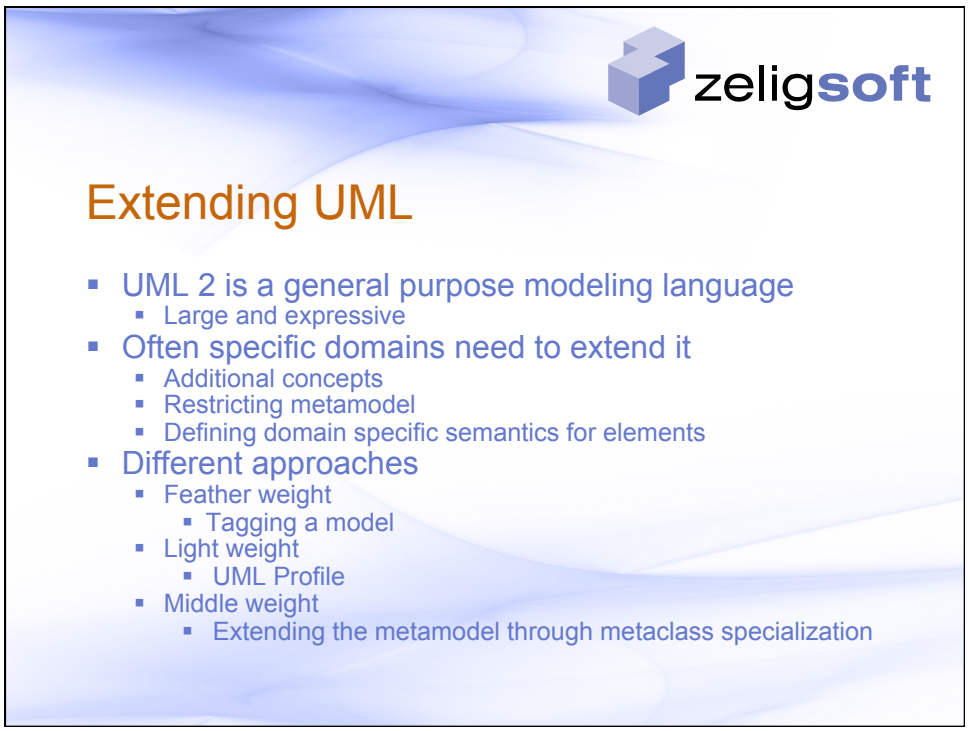
- Programmatically create UML models and elements
- Use of the standard EMF generated API
- EMF reflective capabilities available

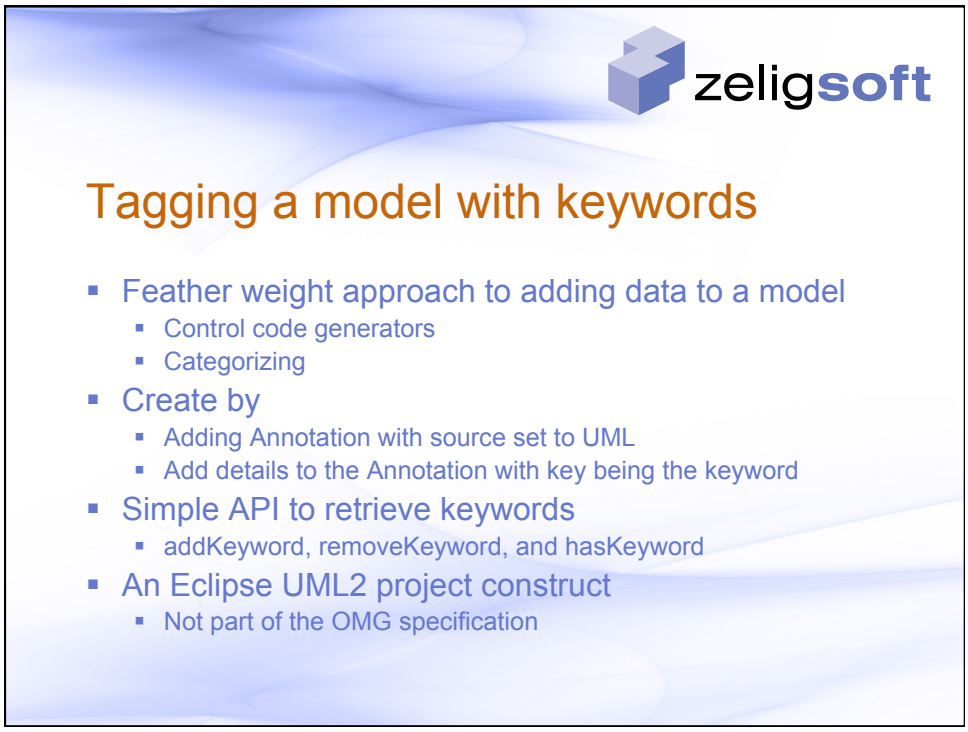
```
// Create resource to add model to
ResourceSet resourceSet = new ResourceSetImpl();
Resource modelResource
    = resourceSet.createResource(URI.createURI("dyeingSystem.uml", true));

// Create UML model and set name
Model dyeingSystemModel = UMLFactory.eINSTANCE.createModel();
dyeingSystemModel.setName("DyeingSystem");

// Add model to the resource
modelResource.getContents().add(dyeingSystemModel);
```

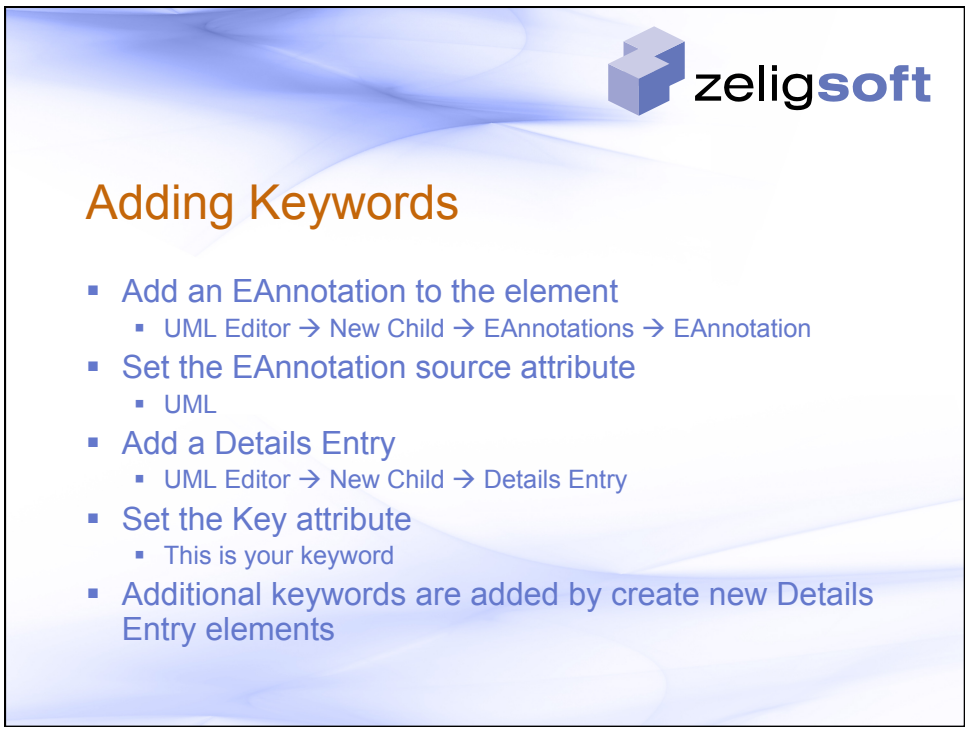

zeligsoft
More Creating Models in Code
To create UML model elements use
UMLFactory, or
For Packages and Models
createdOwnedClass
createNestedPackage
createdOwnedPrimitiveType
createdOwnedEnumeration
...

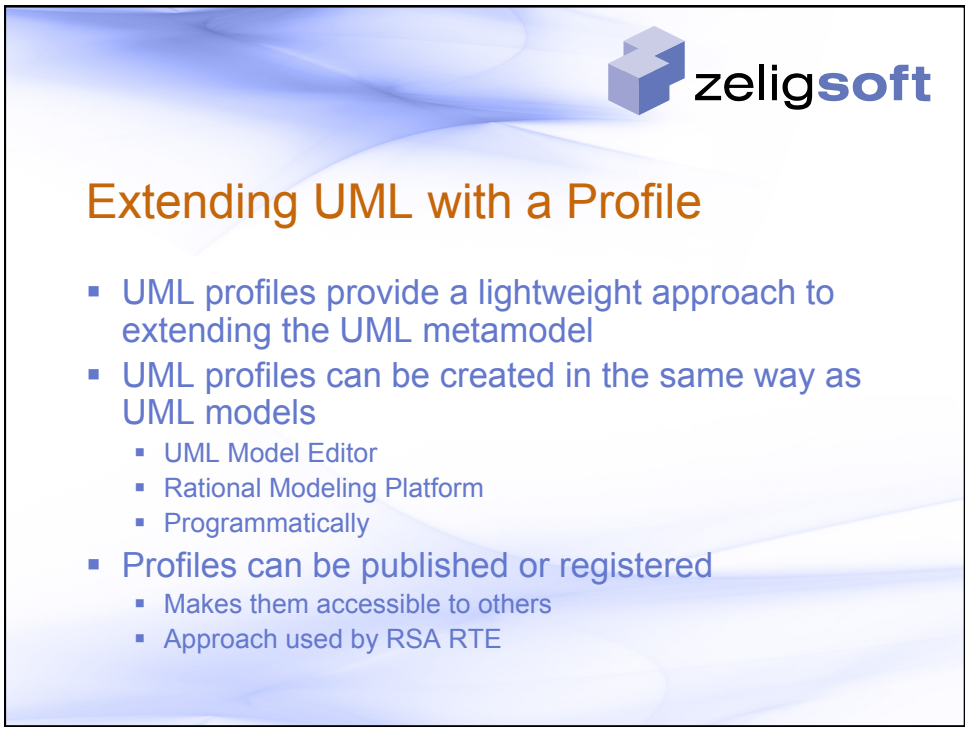

zeligsoft
Persistence and Serialization
Models are persisted in an XMI compliant format
Tools built on top of UML2 project should be able to load the model
Likely won't maintain diagrams
<?xml version="1.0" encoding="UTF-8"?>
<uml:Model xmi:version="2.1" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
xmlns:uml="http://www.eclipse.org/uml2/2.1.0/UML" xmi:id="_AjSRsJutEd2e__u-cIXQ8Q" name="Dyeing System">
<packagedElement xmi:type="uml:Class" xmi:id="_P7Ds8JutEd2e__u-cIXQ8Q" name="DyeingController"/>
<packagedElement xmi:type="uml:Class" xmi:id="_UaWiwJutEd2e__u-cIXQ8Q" name="Valve"/>
<packagedElement xmi:type="uml:Class" xmi:id="_XswusJutEd2e__u-cIXQ8Q" name="Sensor"/>
<packagedElement xmi:type="uml:Class" xmi:id="_cTgQUJutEd2e__u-cIXQ8Q" name="DyeTank"/>
</uml:Model>

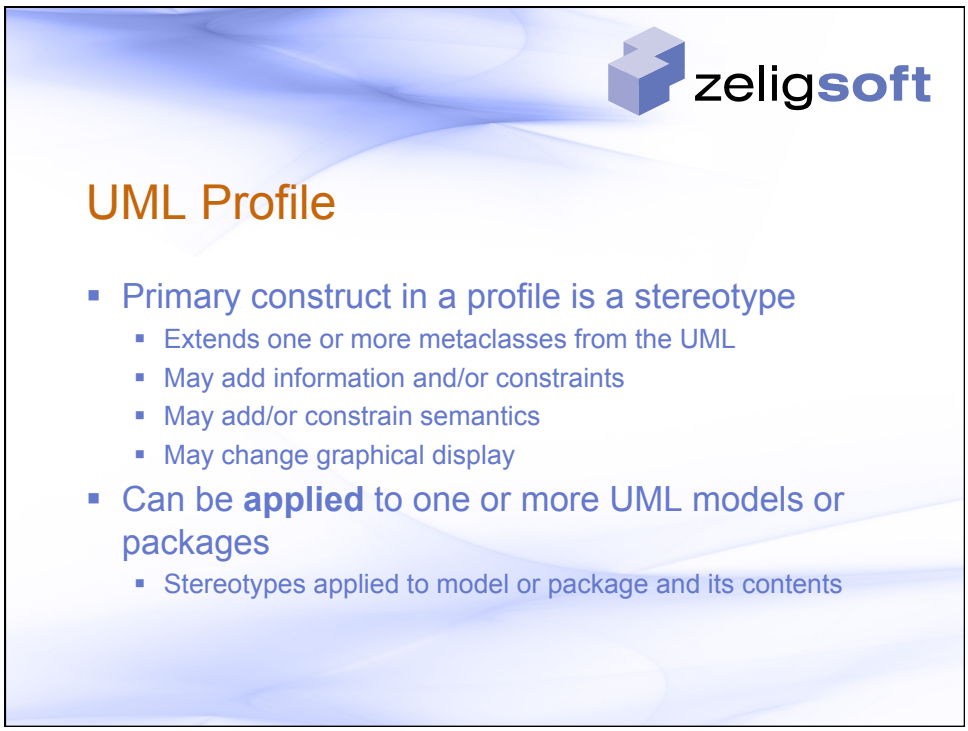

zeligsoft
Summary
In this module we explored
What the UML2 project is
How to create UML2 models
UML Model Editor
Rational Modeling Platform
Through code
Persistence


zeligsoft
Extending UML
UML 2 is a general purpose modeling language
Large and expressive
Often specific domains need to extend it
Additional concepts
Restricting metamodel
Defining domain specific semantics for elements
Different approaches
Feather weight
Tagging a model
Light weight
UML Profile
Middle weight
Extending the metamodel through metaclass specialization

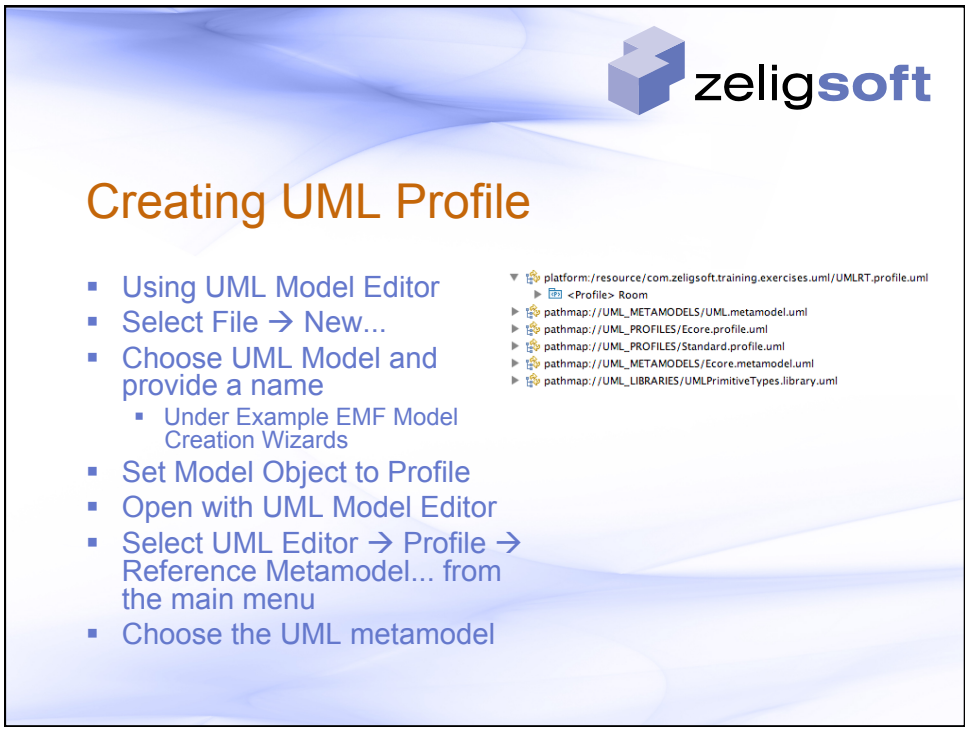

zeligsoft
Tagging a model with keywords
Feather weight approach to adding data to a model
Control code generators
Categorizing
Create by
Adding Annotation with source set to UML
Add details to the Annotation with key being the keyword
Simple API to retrieve keywords
addKeyword, removeKeyword, and hasKeyword
An Eclipse UML2 project construct
Not part of the OMG specification

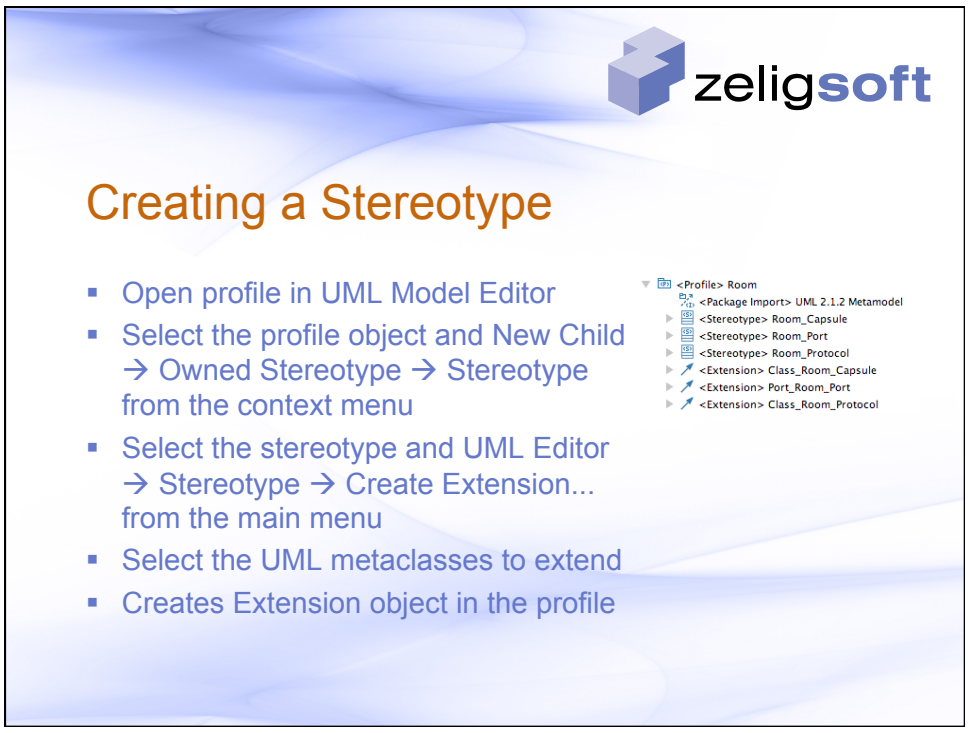

zeligsoft
Adding Keywords
Add an EAnnotation to the element
UML Editor → New Child → EAnnotations → EAnnotation
Set the EAnnotation source attribute
UML
Add a Details Entry
UML Editor → New Child → Details Entry
Set the Key attribute
This is your keyword
Additional keywords are added by create new Details Entry elements

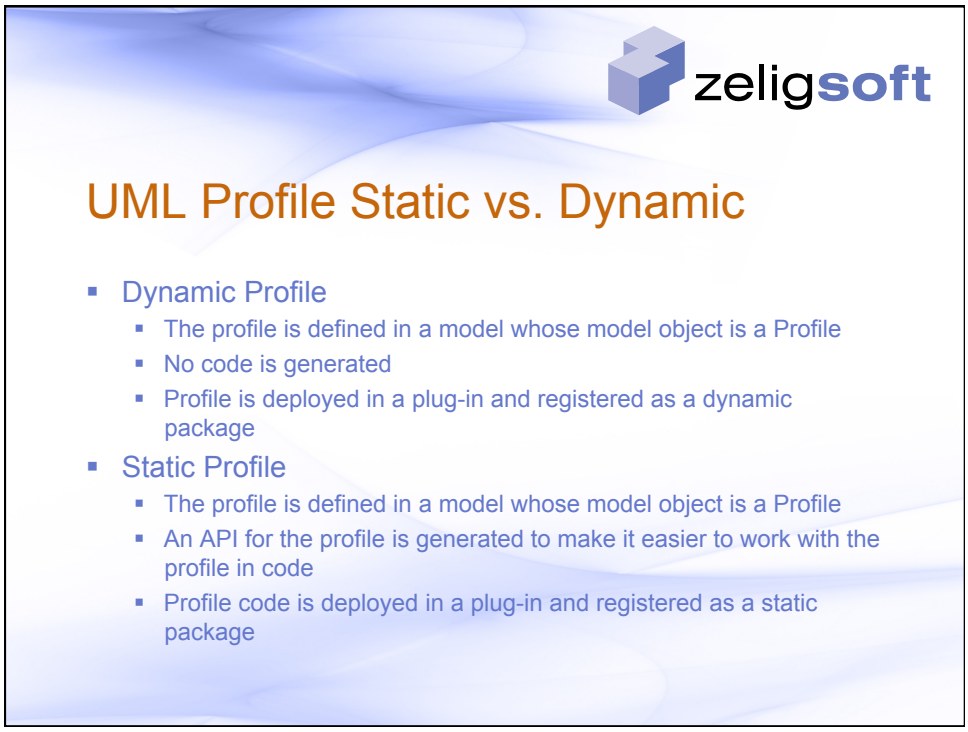


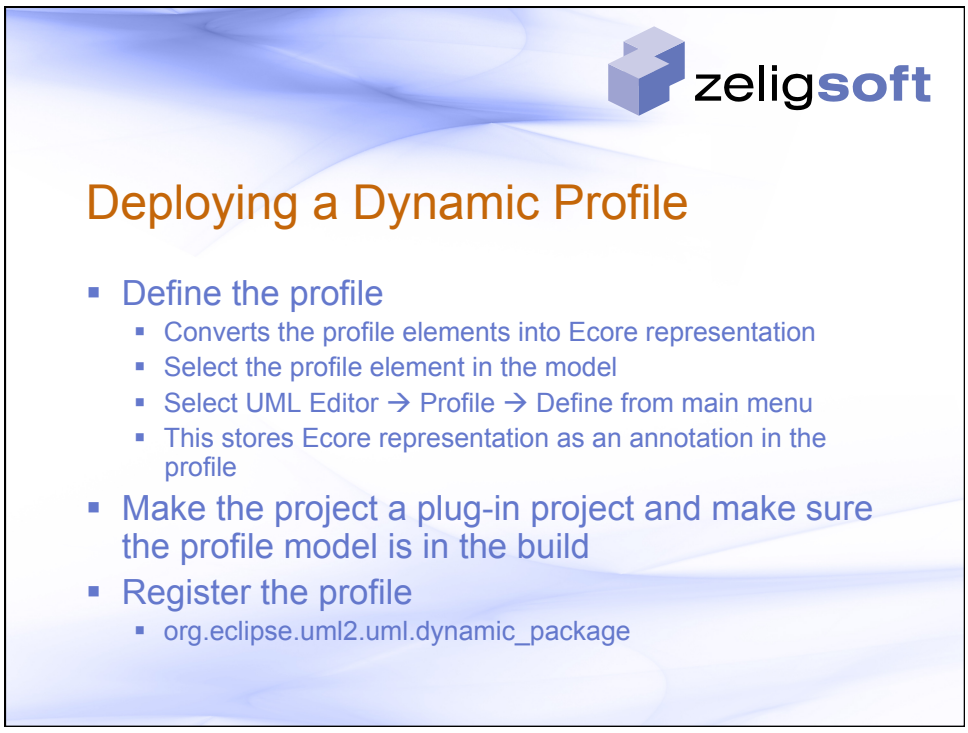


Toby McClean, Zeligsoft, 2008

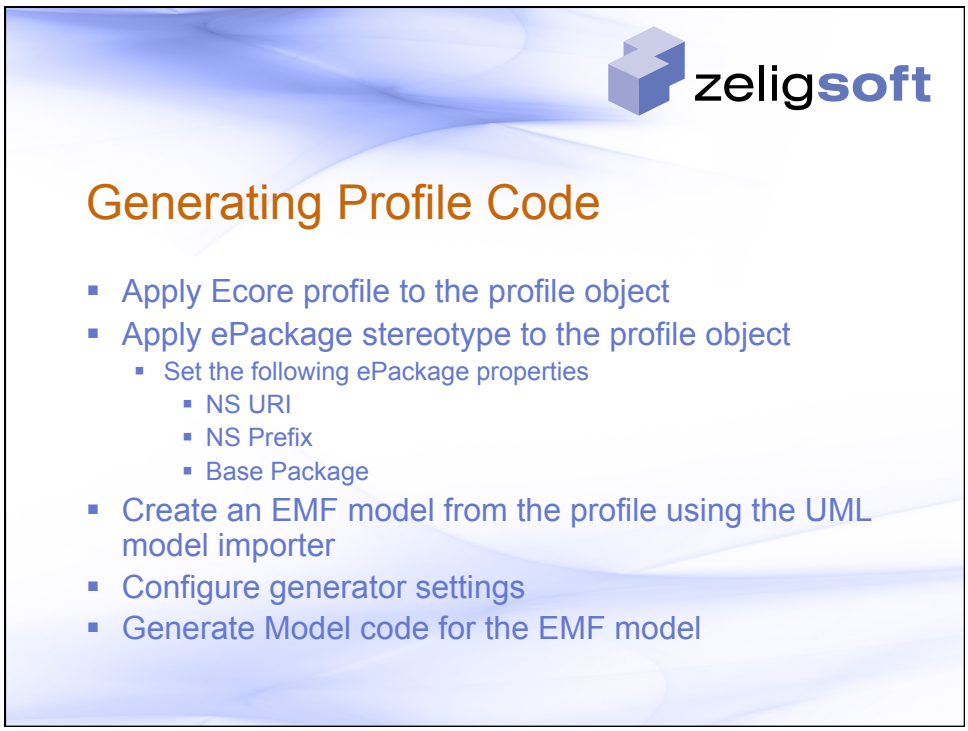

zeligsoft
Creating UML Profile
Using UML Model Editor
Select File → New...
Choose UML Model and provide a name
Under Example EMF Model Creation Wizards
Set Model Object to Profile
Open with UML Model Editor
Select UML Editor → Profile → Reference Metamodel... from the main menu
Choose the UML metamodel
platform:/resource/com.zeligsoft.training.exercises.uml/UMLRT.profile.uml
<Profile> Room
pathmap://UML_METAMODELS/UML.metamodel.uml
pathmap://UML_PROFILES/Ecore.profile.uml
pathmap://UML_PROFILES/Standard.profile.uml
pathmap://UML_METAMODELS/Ecore.metamodel.uml
pathmap://UML_LIBRARIES/UMLPrimitiveTypes.library.uml


zeligsoft
Creating a Stereotype
Open profile in UML Model Editor
Select the profile object and New Child → Owned Stereotype → Stereotype from the context menu
Select the stereotype and UML Editor → Stereotype → Create Extension... from the main menu
Select the UML metaclasses to extend
Creates Extension object in the profile
<Profile> Room
<Package Import> UML 2.1.2 Metamodel
<Stereotype> Room_Capsule
<Stereotype> Room_Port
<Stereotype> Room_Protocol
<Extension> Class_Room_Capsule
<Extension> Port_Room_Port
<Extension> Class_Room_Protocol


zeligsoft
UML Profile Static vs. Dynamic
Dynamic Profile
The profile is defined in a model whose model object is a Profile
No code is generated
Profile is deployed in a plug-in and registered as a dynamic package
Static Profile
The profile is defined in a model whose model object is a Profile
An API for the profile is generated to make it easier to work with the profile in code
Profile code is deployed in a plug-in and registered as a static package


zeligsoft
Deploying a Dynamic Profile
Define the profile
Converts the profile elements into Ecore representation
Select the profile element in the model
Select UML Editor → Profile → Define from main menu
This stores Ecore representation as an annotation in the profile
Make the project a plug-in project and make sure the profile model is in the build
Register the profile
org.eclipse.uml2.uml.dynamic_package


zeligsoft
Deploying a Static Profile
Generate the profile code
Requires creating an EMF representation of the profile
Make the project a plug-in project
Include the generated code and the profile model
Register the profile
org.eclipse.uml2.uml.generated_package


zeligsoft
Generating Profile Code
Apply Ecore profile to the profile object
Apply ePackage stereotype to the profile object
Set the following ePackage properties
NS URI
NS Prefix
Base Package
Create an EMF model from the profile using the UML model importer
Configure generator settings
Generate Model code for the EMF model

zeligsoft

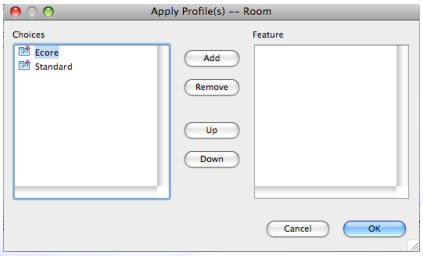
Discussion

- Examples from your experience?
  - Additional information?
  - Additional concepts?
- Does this impact modeling, validation or generation?

zeligsoft

Applying a UML profile

- A Profile is applied to a Model, Profile or Package
- Open model in UML Model Editor
- Select model or package object and UML Editor → Package → Apply Profile... from the main menu
- Choose the profiles to apply
- Creates a Profile Application object in the package



zeligsoft

Applying a UML Profile - Code

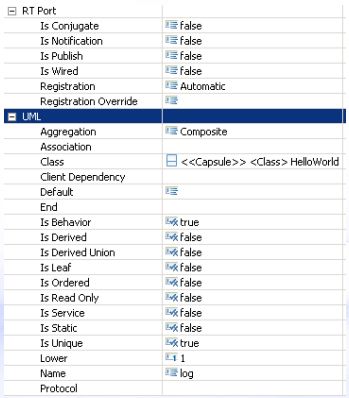
```
// Load the resource containing the profile
Resource roomProfileResource
    = resourceSet.getResource(ROOM_PROFILE_URI, true);
Profile roomProfile =
    (Profile) roomProfileResource.getContents().get(0);

// Apply the profile to a model
dyeingSystemModel.applyProfile(roomProfile);
```

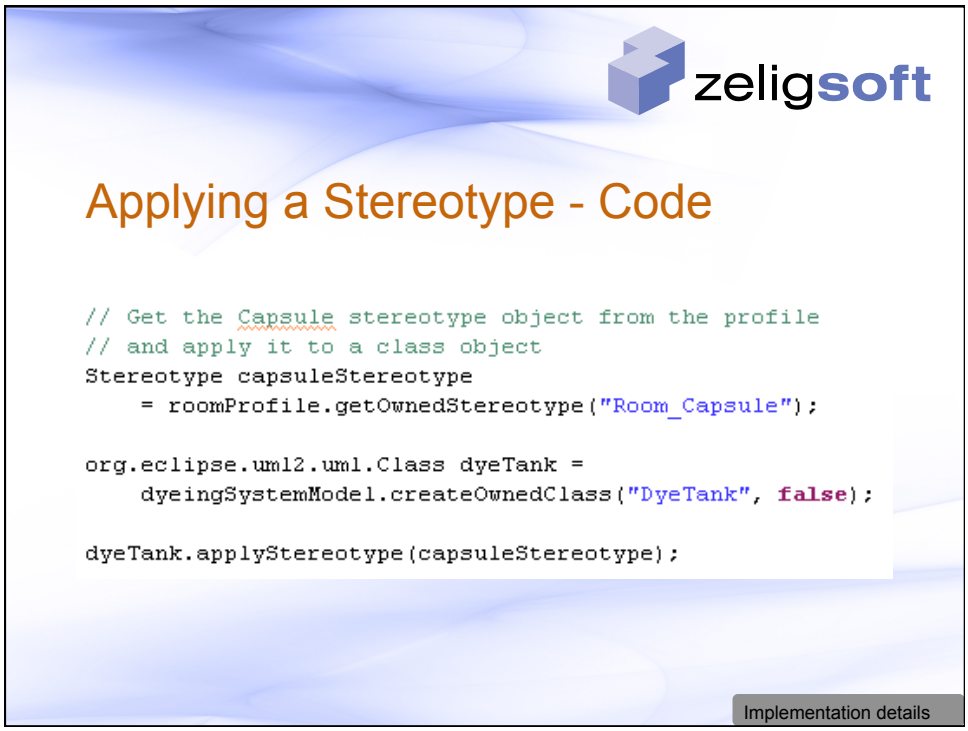
Implementation details

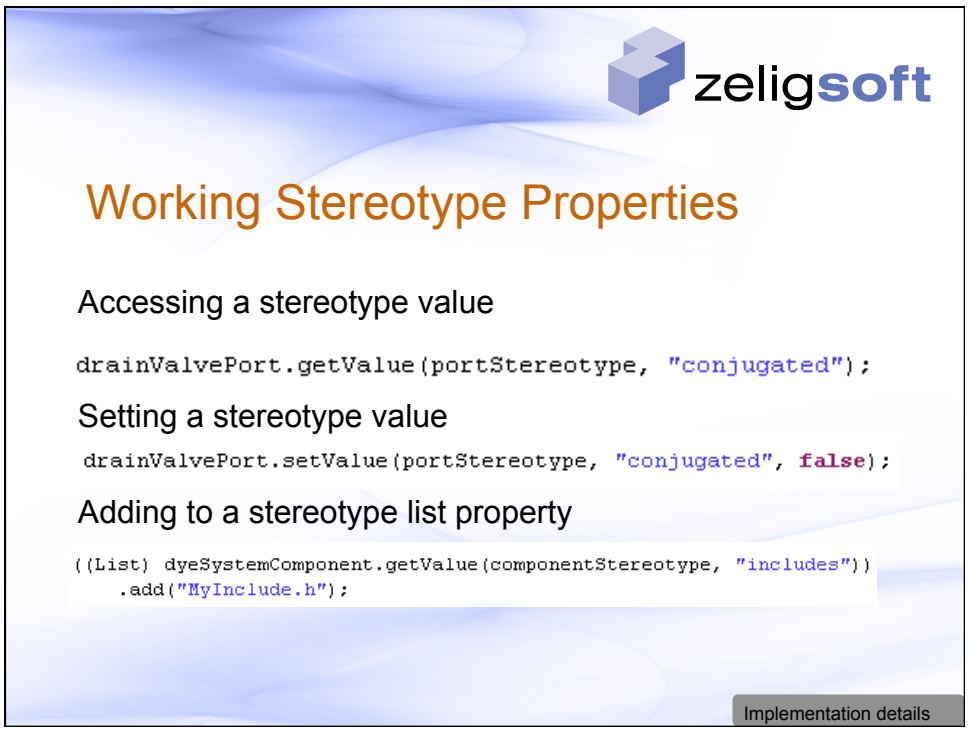
zeligsoft

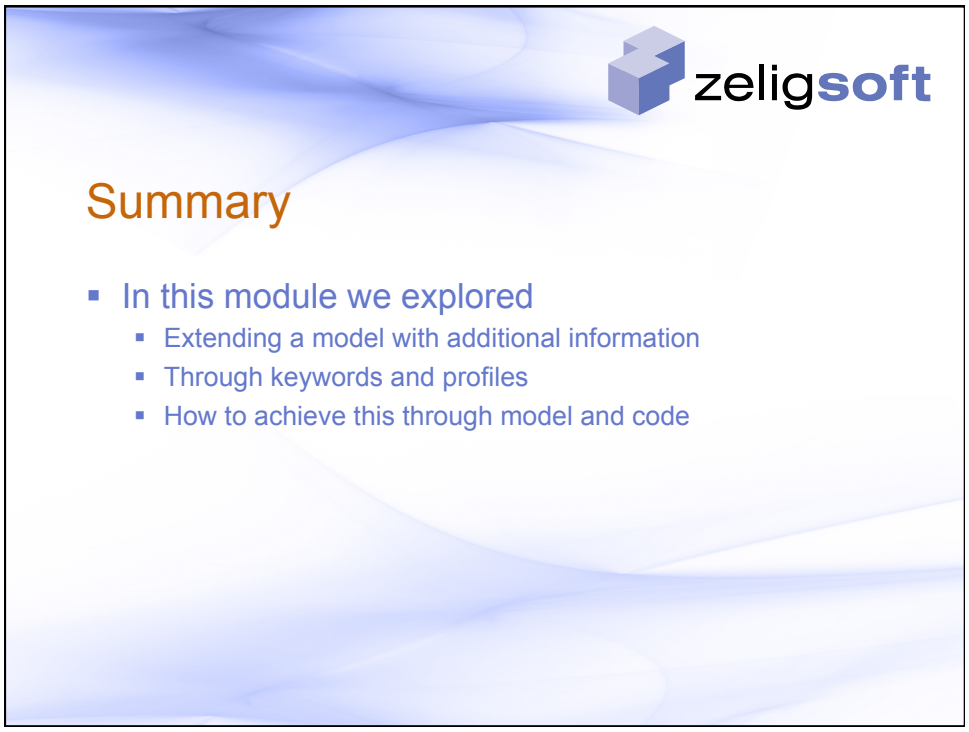
Applying a Stereotype

- Stereotypes are applied to elements in a model
  - More than one can be applied
  - Applicable elements defined by Stereotypes metaclass extensions
  - Attributes of Stereotypes can be set in the properties view of the UML Model Editor
- Select the element in the editor and UML Editor → Element → Apply Stereotype... from the main menu




zeligsoft
Applying a Stereotype - Code
// Get the Capsule stereotype object from the profile
// and apply it to a class object
Stereotype capsuleStereotype
= roomProfile.getOwnedStereotype("Room_Capsule");
org.eclipse.uml2.uml.Class dyeTank =
dyeingSystemModel.createOwnedClass("DyeTank", false);
dyeTank.applyStereotype(capsuleStereotype);
Implementation details


zeligsoft
Working Stereotype Properties
Accessing a stereotype value
drainValvePort.getValue(portStereotype, "conjugated");
Setting a stereotype value
drainValvePort.setValue(portStereotype, "conjugated", false);
Adding to a stereotype list property
((List) dyeSystemComponent.getValue(componentStereotype, "includes"))
.add("MyInclude.h");
Implementation details

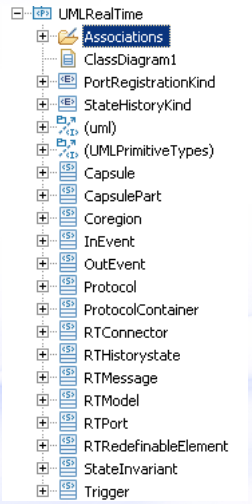

zeligsoft
Summary
In this module we explored
Extending a model with additional information
Through keywords and profiles
How to achieve this through model and code


zeligsoft
Keywords in RSA-RTE
General
Profiles
Stereotypes
Documentation
Constraints
Capabilities
Relationships
Keywords:
Applied Stereotypes:
Stereotype
Profile
Required
modelLibr...
Standard
False
RTModel
UMLRealTime
True
Apply Stereotypes...
Unapply Stereotypes

zeligsoft

Profiles in RSA-RTE

- Profiles are used extensively in RSA-RTE
  - Extending/constraining the UML to UML-RT semantics and notation
    - UMLRealTime profile
    - UMLRealTime model libraries for things like Frame, and Log
  - Adding information to model elements to generate code
    - C++ support through CPPPropertySets profile
    - C++ model libraries for
- When a RSA-RTE model is created
  - UMLRealTime profile is applied
  - CPPPropertySets profile is applied
  - RTClasses, RTComponents and CPPPrimitiveDatatypes libraries



zeligsoft

Creating Profiles in RSA-RTE

- Model the domain
  - Concepts, attributes and relationships in a Profile
  - Complimentary model libraries
  - Constraints - OCL or Java
- Publish the Profile
  - Helps RSA-RTE with versioning and migration
- Register the Profile in a plug-in
  - Make it available for RSA-RTE to apply it
- Can export to open source UML 2

zeligsoft

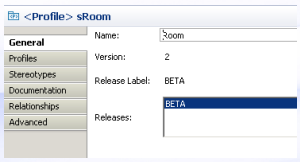
Creating Profiles in RSA-RTE

- File → New → Other…
- Select UML Profile or UML Profile Project…
  - Under Modeling → UML Extensibility
- Set name and import the UML Primitive Types library
- Creates a Profile object that already imports the UML metamodel
- Add stereotypes, classes and relationships

zeligsoft

Releasing a Profile

- RSA-RTE provides a release capability for profiles
  - A released profile is to be additive otherwise backwards compatibility may not work
- To release a profile
  - Select Release in the context menu of the profile object
  - Associate a label with the release
  - Be careful how often a profile is released



zeligsoft

Publishing a Profile

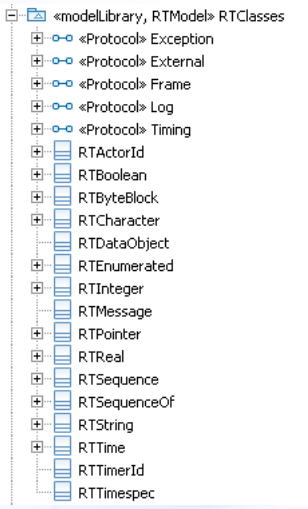
- Publishing a profile makes it available for others to install and use in their RSA-RTE models
- To publish
  - Make the project a plug-in project
  - Add the profile file to the build
  - Extend “com.ibm.xtools.uml.msl.UMLProfiles”
    - Enables RSM to find the profile
  - Publish the plug-in

```
<extension point="com.ibm.xtools.uml.msl.UMLProfiles">
    <UMLProfile id="com.zeligsoft.exercises.profiles.room"
        name="ROOM"
        path="pathmap://EXERCISE_PROFILES/Room.epx" required="false" visible="true" />
</extension>
```

zeligsoft

Model Libraries

- Model libraries provide a mechanism for publishing a set of model elements for reuse
  - e.g. Data types, reusable components
- It is a model with the modelLibrary stereotype applied
  - modelLibrary is part of the Standard profile
- The elements will be read-only in the model that imports the library



zeligsoft

Publishing a Model Library

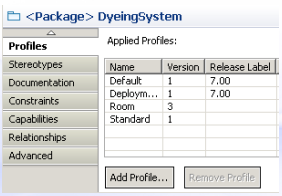
- Publishing a model library makes it available for others to import and use in their RSA-RTE models
- To publish
  - Make the project a plug-in project
  - Add the model file to the build
  - Extends com.ibm.xtools.uml.msl.UMLLibraries
    - Enables RSM to find the profile
  - Publish the plug-in

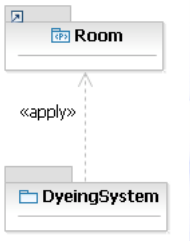
```
<extension point="com.ibm.xtools.uml.msl.UMLLibraries">
   <UMLLibrary
         name="RoomServices"
         path="pathmap://EXERCISE_MODEL_LIBRARIES/RoomServices.emx">
   </UMLLibrary>
</extension>
```

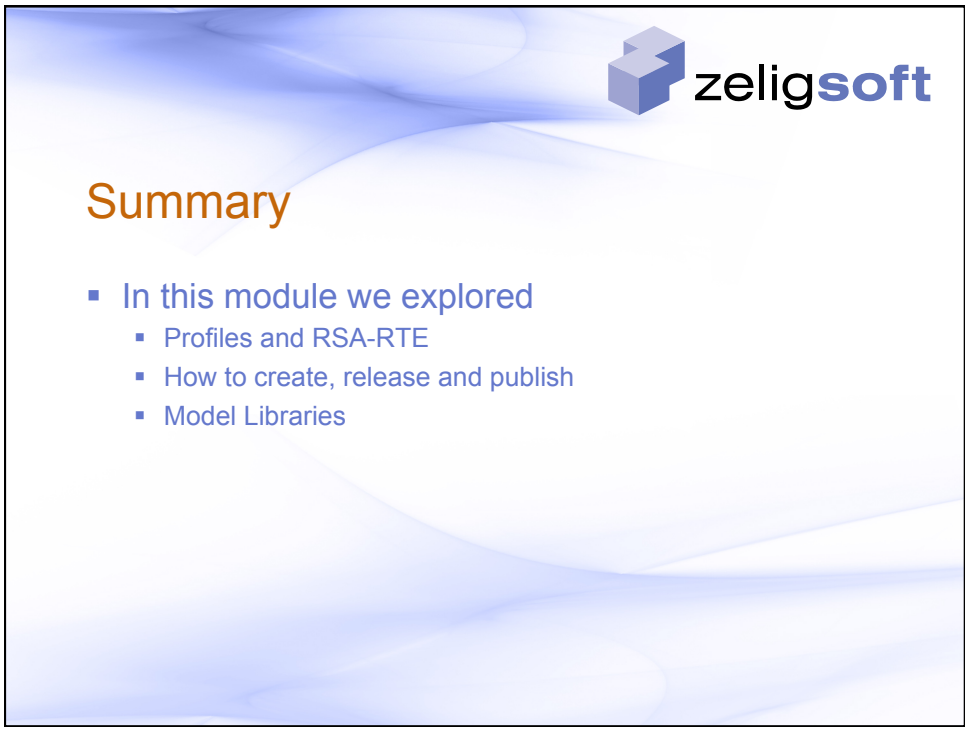
zeligsoft

Applying Profiles in RSA-RTE

- On a Model or Package “Add Profile…”
- Apply Stereotype to Elements in the model
- Set attributes of the Stereotype
  - Advanced tab
  - Stereotype tab






zeligsoft
Summary
In this module we explored
Profiles and RSA-RTE
How to create, release and publish
Model Libraries


zeligsoft
Discussion
Any RSA-RTE concepts you would add/remove?

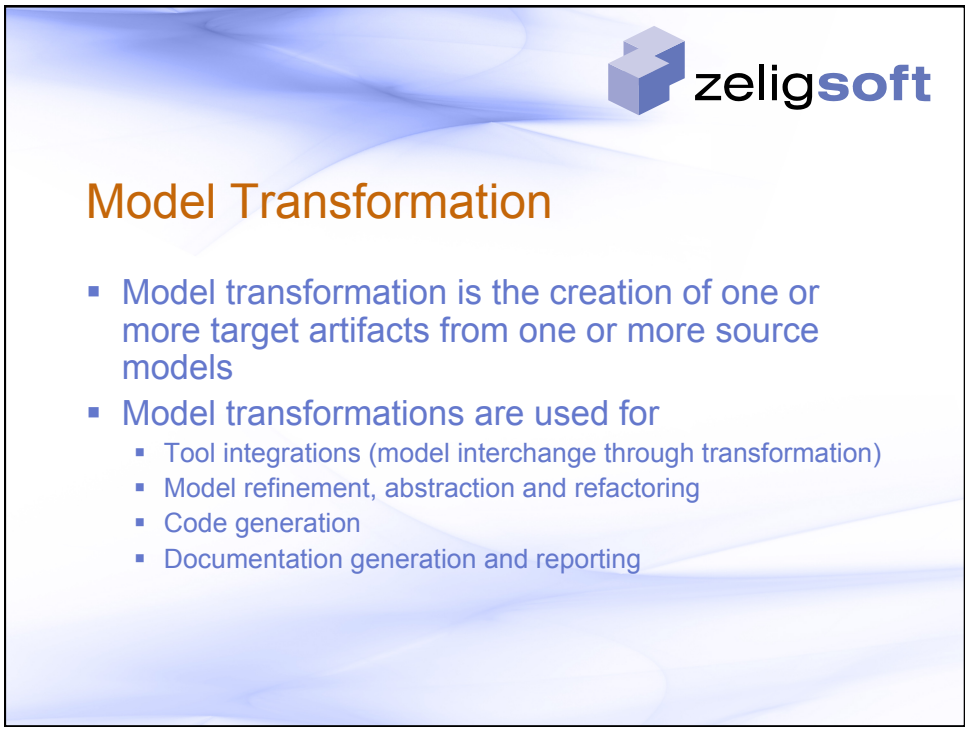
zeligsoft

Questions

Toby McClean, Zeligsoft, 2008


zeligsoft
Goals
After this module you will have
An understanding of the main transformation technologies
Understand their applicability
Be able to put simple transformations together


zeligsoft
Agenda
Model transformation introduction
Model to model
Java
QVT
Model to text
xPand
xTend
JET
oAW



Toby McClean, Zeligsoft, 2008

Toby McClean, Zeligsoft, 2008

zeligsoft

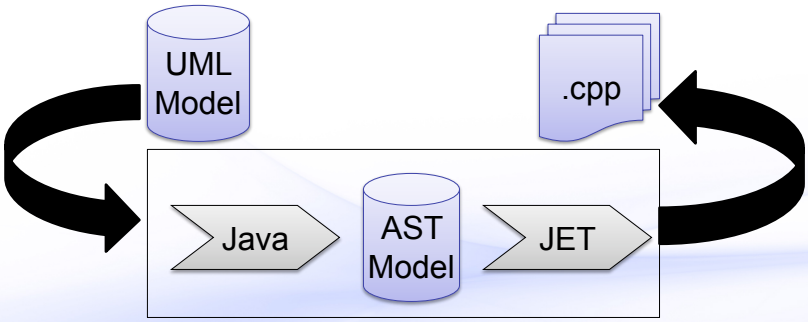
Model to Text Transformations

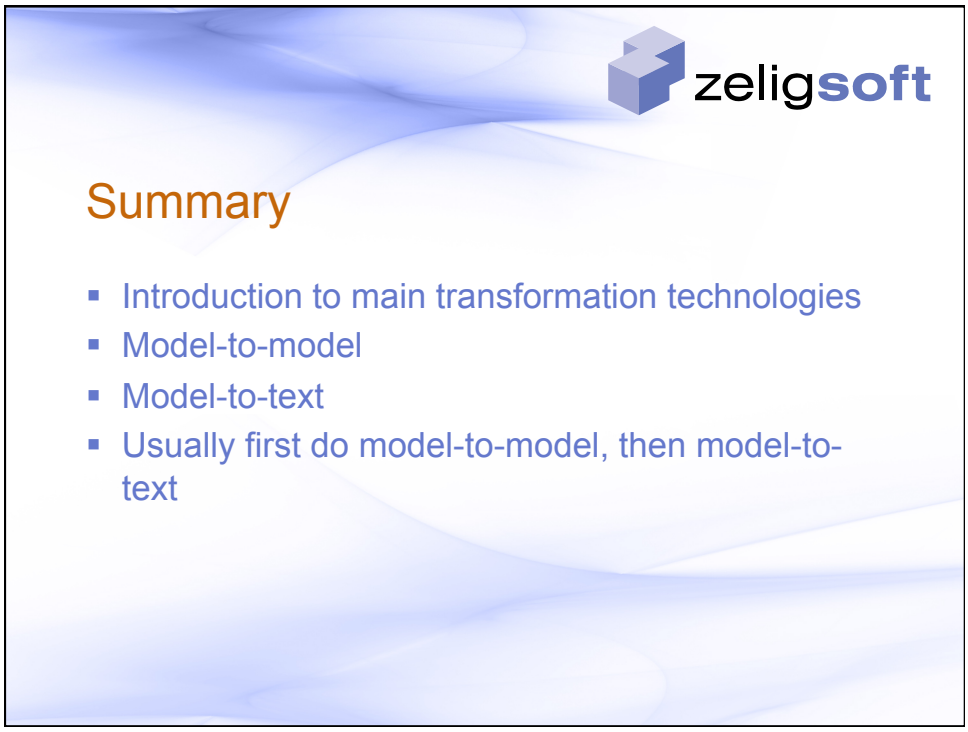
- A M2T transformation creates one or more text based artifacts from one or more source model
- Typically template based
  - Create a template from an example of the desired artifact
- M2T consideration
  - Often it is best to use M2T with a source model that is dedicated to the transformation
- Eclipse technologies
  - M2T Project - **JET2** and **xPand**

zeligsoft

RSA RTE Transformation Workflow

Lets put this into context, RSA-RTE uses a M2M then M2T to generate code




zeligsoft
Summary
Introduction to main transformation technologies
Model-to-model
Model-to-text
Usually first do model-to-model, then model-to-text


zeligsoft
Discussion
What type of transformations could you imagine?
Some examples
Structural, in place
Capsule-to-class
Proxies
Bi-directional
Linking between system and design
Analysis
Walk the model and generate reports
Dependency, packaging violations


zeligsoft
Agenda
Model transformation introduction
Model to model
Java
QVT
Model to text
xPand
xTend
JET
oAW


zeligsoft
Eclipse Transformation Technologies
Model to Model Transformations

zeligsoft

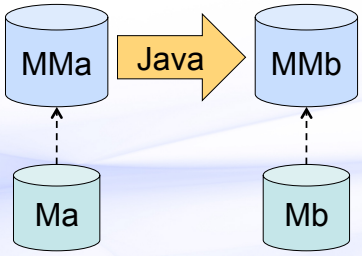
Overview

- In this section we will explore the M2M technologies available in Eclipse
- The specific technologies are
  - Java
  - QVT

zeligsoft

M2M with Java

- The most basic approach to M2M is to use straight Java with the EMF generated API
- Transformation writer has full power or a 3GL
  - Development tools
  - Debugging facilities
- Transformation written against meta-model
  - Executed against instance model



zeligsoft

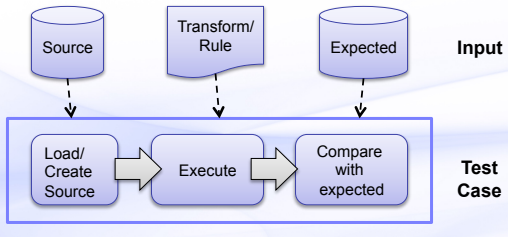
Developing a Transformation

- Develop the transformation as if you were writing a Java application
- Model navigation
  - EMF Reflective API
  - Metamodel specific API generated by EMF
- Transformation rules
  - Rule constraints
  - Target element construction

zeligsoft

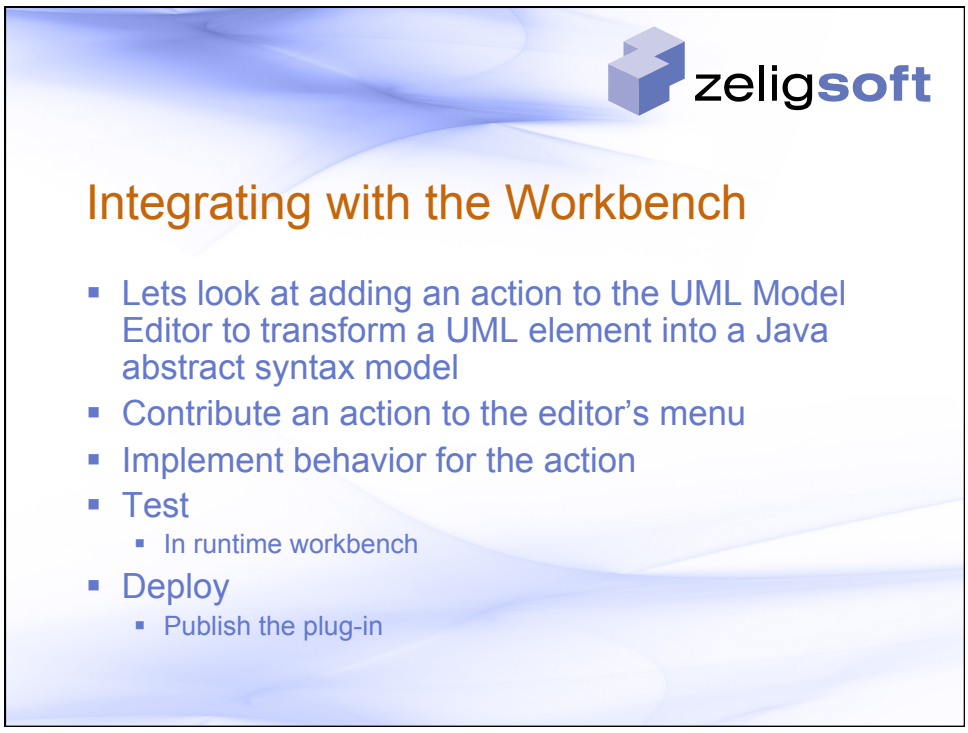
Testing Transformations

- Traditional Java test techniques
  - Hand crafted test framework
  - JUnit
- Using JUnit
  - Automatable
  - Repeatable



Toby McClean, Zeligsoft, 2008

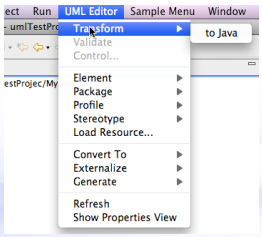

zeligsoft
Executing the Transformation
Use workbench Java execution capabilities
We can use the Run As → Java Application
We can debug using Debug As → Java Application
To pass transformation parameter values
Use command line
Build a user interface
Integrate with workbench
Provide an action to invoke from editor and/or view
Provide a transformation resource with action to execute


zeligsoft
Integrating with the Workbench
Lets look at adding an action to the UML Model Editor to transform a UML element into a Java abstract syntax model
Contribute an action to the editor's menu
Implement behavior for the action
Test
In runtime workbench
Deploy
Publish the plug-in

zeligsoft

Contribute Action to the Editor

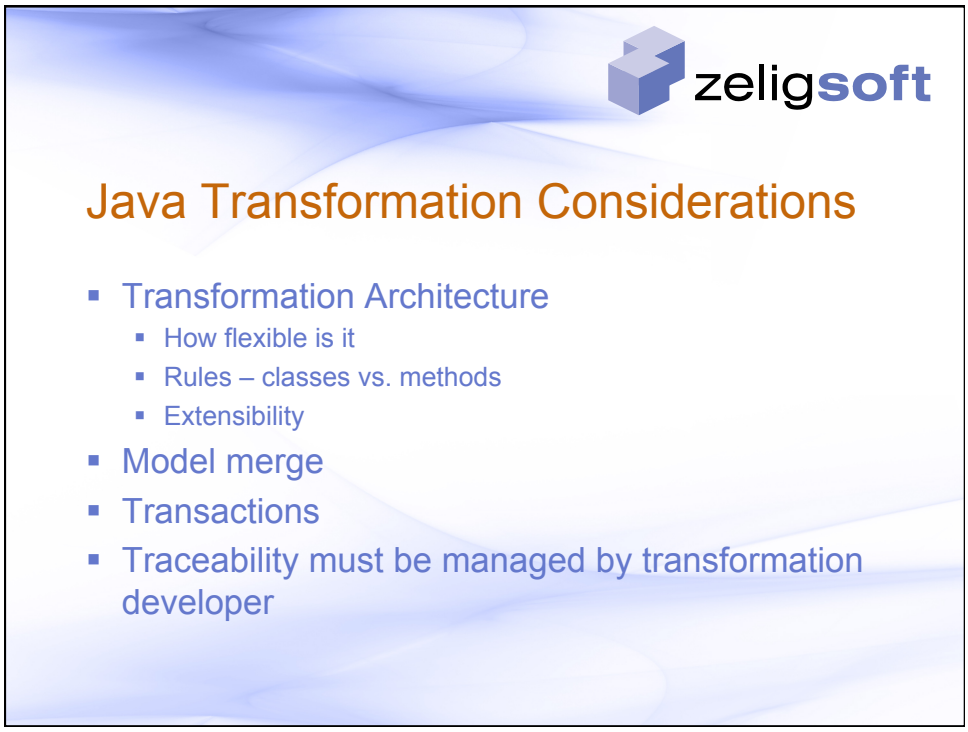
- UML Model Editor ID
  - org.eclipse.uml2.uml.editor.presentation.UMLEditorID
- Menu bar path
  - org.eclipse.uml2.umlMenuID
    - settings
    - actions
    - additions
    - additions-end

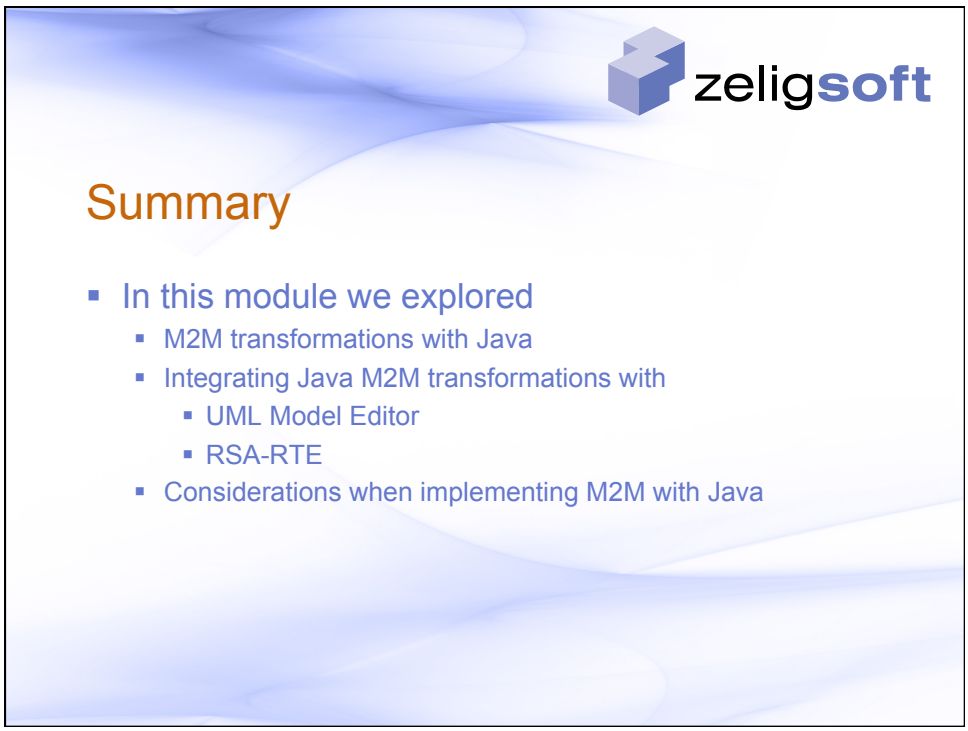


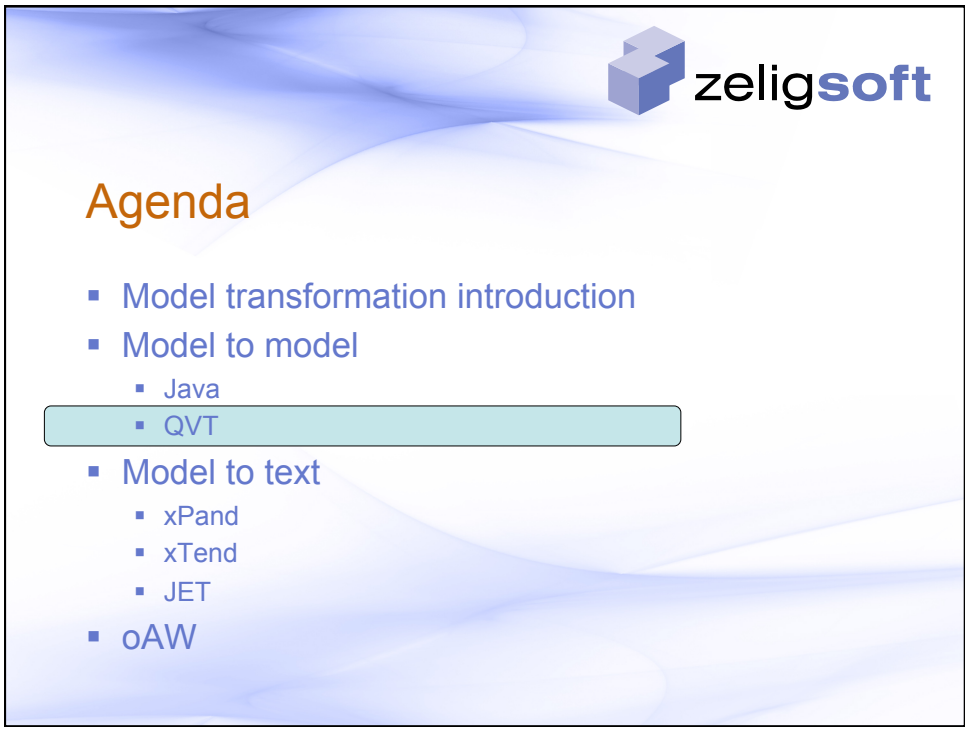
zeligsoft

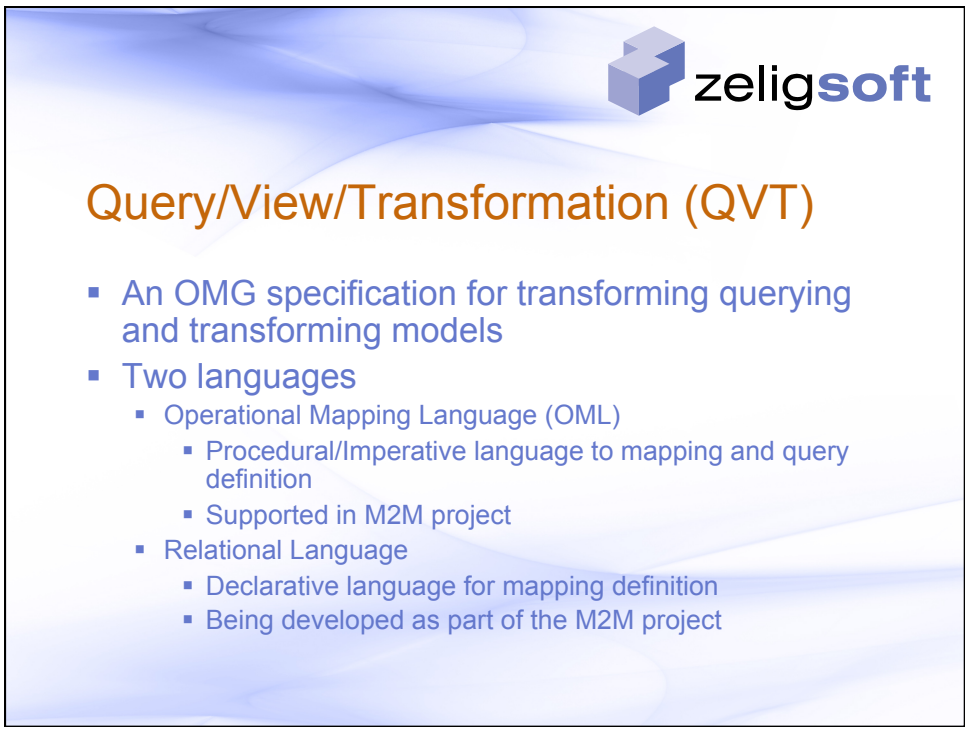
Integrating with RSA-RTE

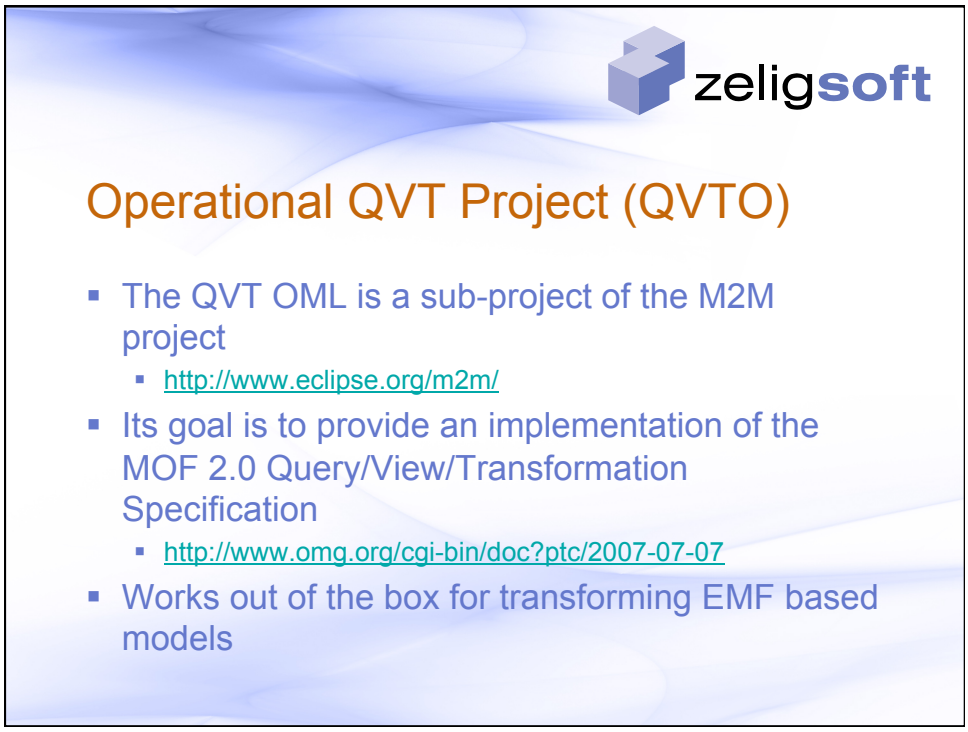
- Contribute an action
  - to the Project Explorer context menu
  - to the Diagram Editor context menu
- Implement action handler
  - Implement IEditorActionDelegate
- Test
  - In runtime workbench
- Deploy
  - Publish the plug-in

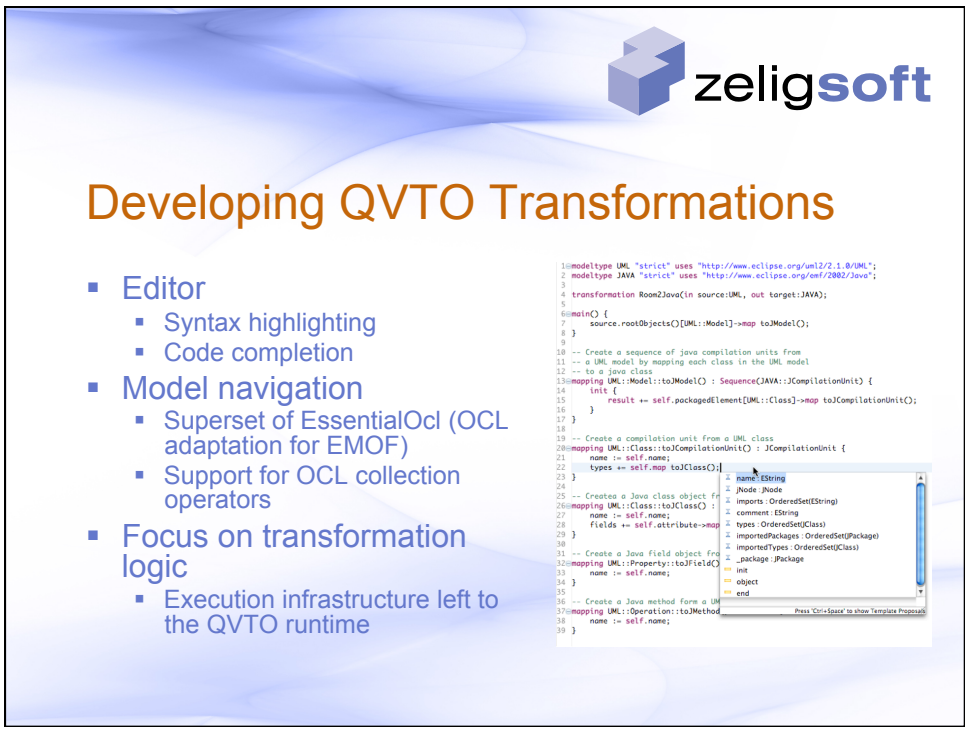

zeligsoft
Java Transformation Considerations
Transformation Architecture
How flexible is it
Rules – classes vs. methods
Extensibility
Model merge
Transactions
Traceability must be managed by transformation developer


zeligsoft
Summary
In this module we explored
M2M transformations with Java
Integrating Java M2M transformations with
UML Model Editor
RSA-RTE
Considerations when implementing M2M with Java


zeligsoft
Agenda
Model transformation introduction
Model to model
Java
QVT
Model to text
xPand
xTend
JET
oAW


zeligsoft
Query/View/Transformation (QVT)
An OMG specification for transforming querying and transforming models
Two languages
Operational Mapping Language (OML)
Procedural/Imperative language to mapping and query definition
Supported in M2M project
Relational Language
Declarative language for mapping definition
Being developed as part of the M2M project

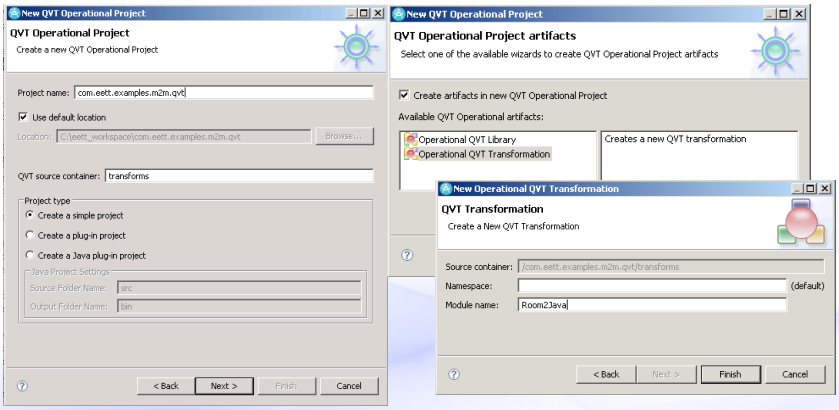




Toby McClean, Zeligsoft, 2008

zeligsoft

Developing QVTO Transformations



zeligsoft

Testing QVTO Transformations

- Same approach as Java transforamtions
- Traditional Java test techniques
  - Hand crafted test framework
  - JUnit
- Using JUnit
  - Automatable
  - Repeatable

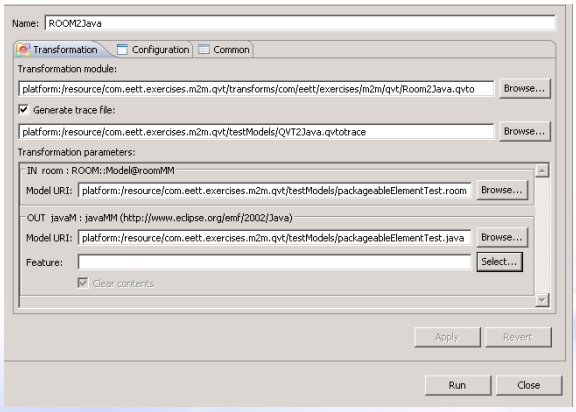
zeligsoft

Executing QVTO Transformations

- The QVTO project integrates with the workbench Run framework
- Select the transformation resource in the project explorer
- Choose Run As → Run Configurations… from the context menu
- Create a new configuration under Operational QVT Interpreter
  - The transformation parameters is populated from the transformation module that is specified
  - The option to generate a trace file for debugging
  - The option to save the configuration for sharing

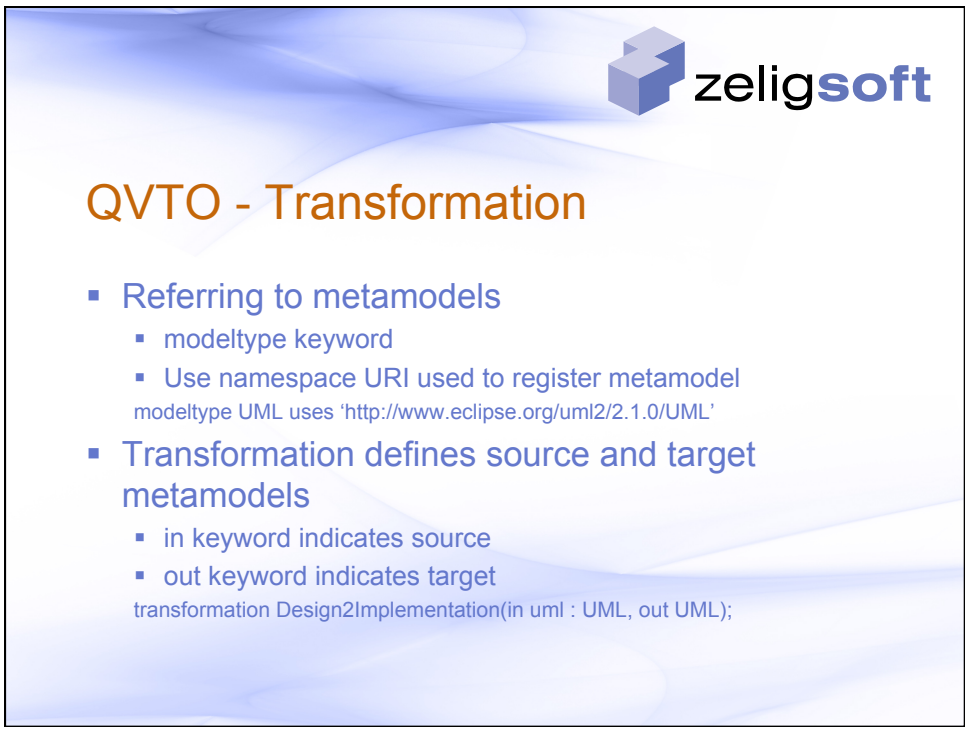
zeligsoft

Executing QVTO Transformations



Toby McClean, Zeligsoft, 2008

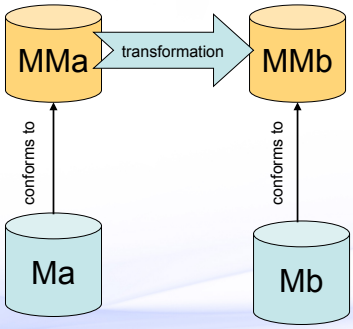

zeligsoft
Integrating with the Workbench
Add an action to the workbench
Create a context
Use an interpreted QVT transformation
Grab the input model element from a Resource or editor
Execute the transformation
Display or persist results and trace


zeligsoft
QVTO - Transformation
Referring to metamodels
modeltype keyword
Use namespace URI used to register metamodel
modeltype UML uses 'http://www.eclipse.org/uml2/2.1.0/UML'
Transformation defines source and target metamodels
in keyword indicates source
out keyword indicates target
transformation Design2Implementation(in uml : UML, out UML);

zeligsoft

QVTO - Transformation

- Source model Ma that conforms to metamodel MMa
- Target model Mb that conforms to metamodel MMb
- Possible to have multiple sources and targets



transformation mMa2MMb(in ma : Mma, out mb : MMb);

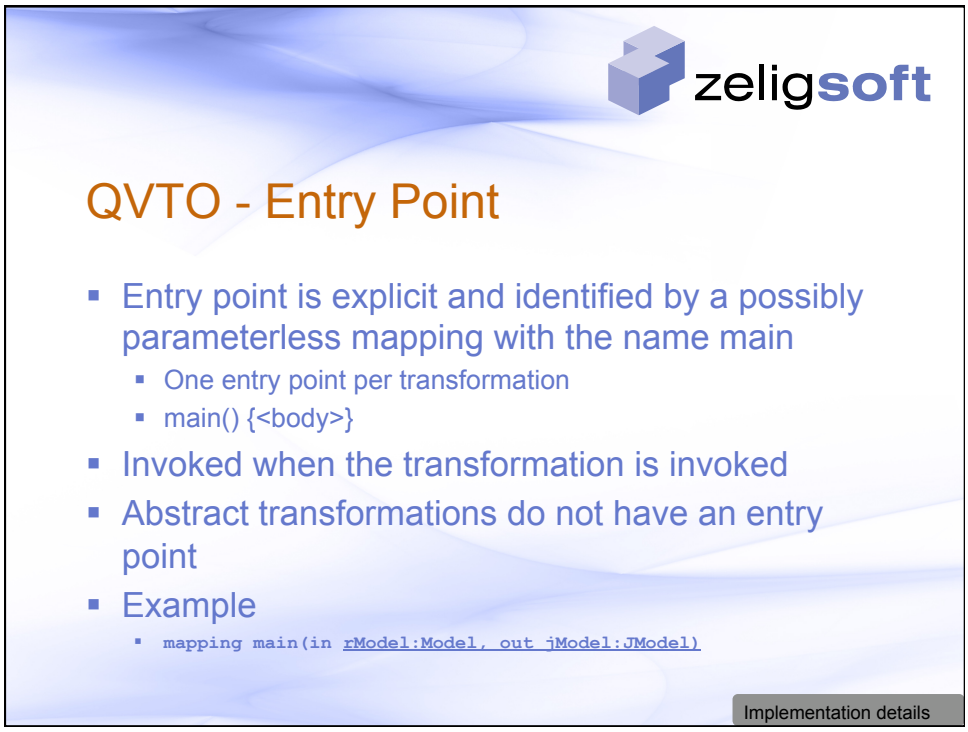
zeligsoft

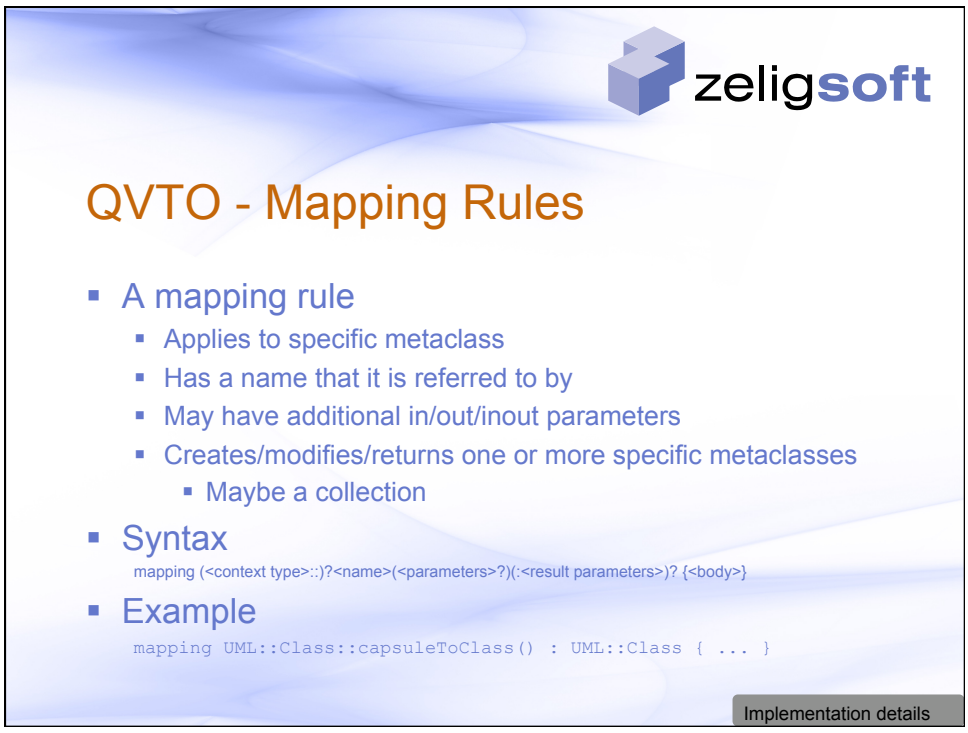
QVTO - Metamodels

- Define the parameter types for transformations
- Can be explictly referenced by their namespace URI
  - UML - http://www.eclipse.org/uml2/2.1.0/UML
  - Ecore - http://www.eclipse.org/emf/2002/Ecore
- Use Metamodel Explorer view to find URI's
  - Window → Show View → Metamodel Explorer
- Syntax

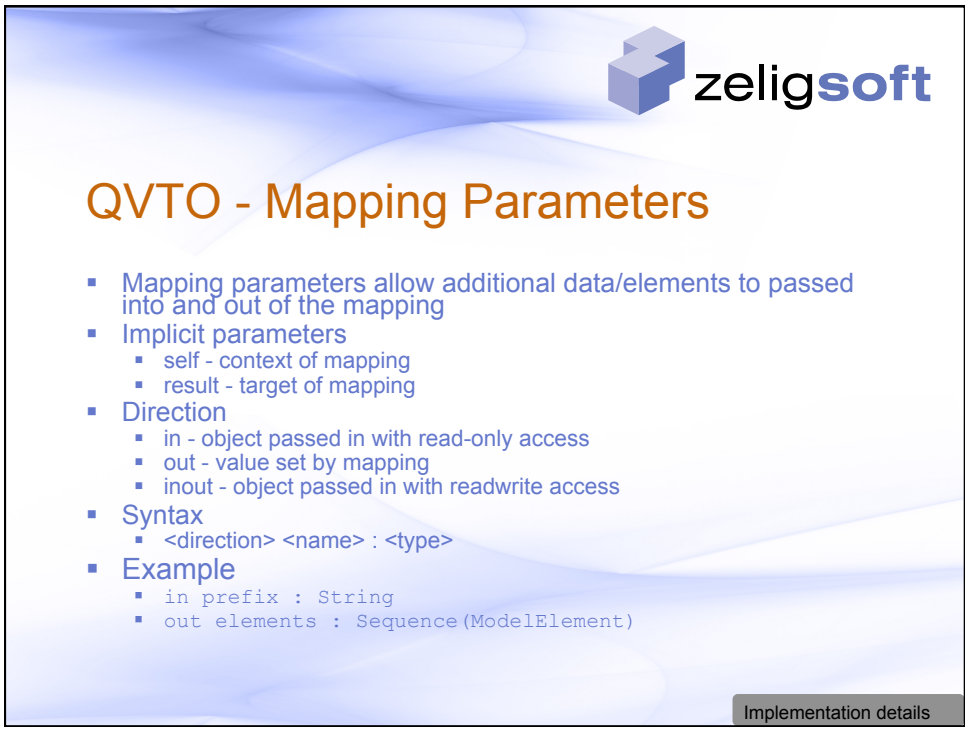
  modeltype <local name> **uses** '<ns uri>';

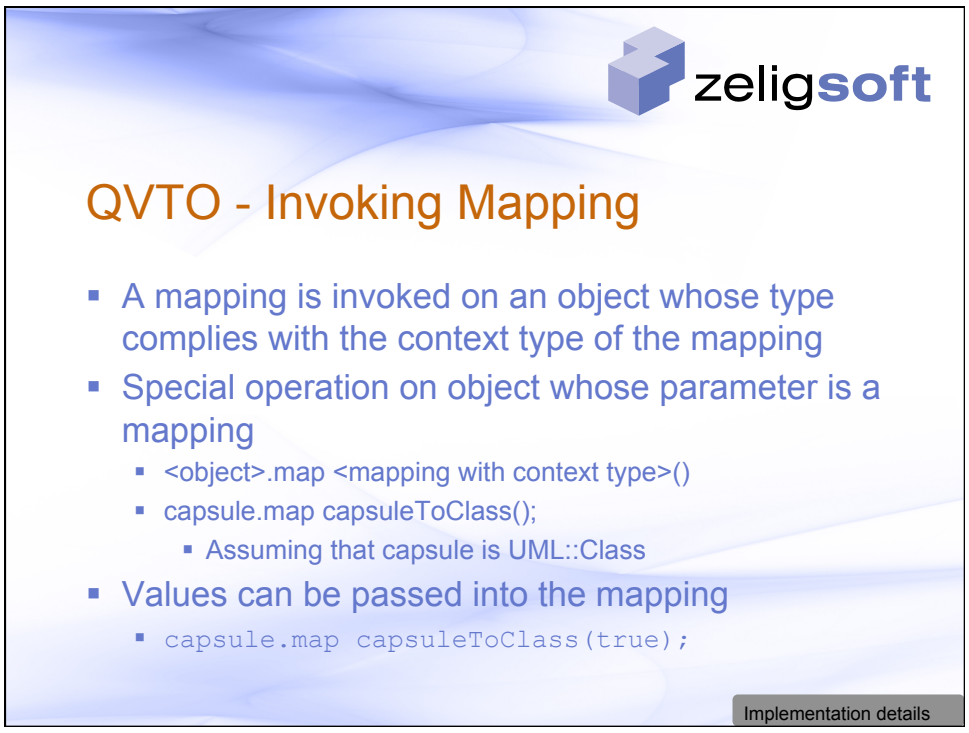
  modeltype UML **uses** 'http://www.eclipse.org/uml2/2.1.0/UML';

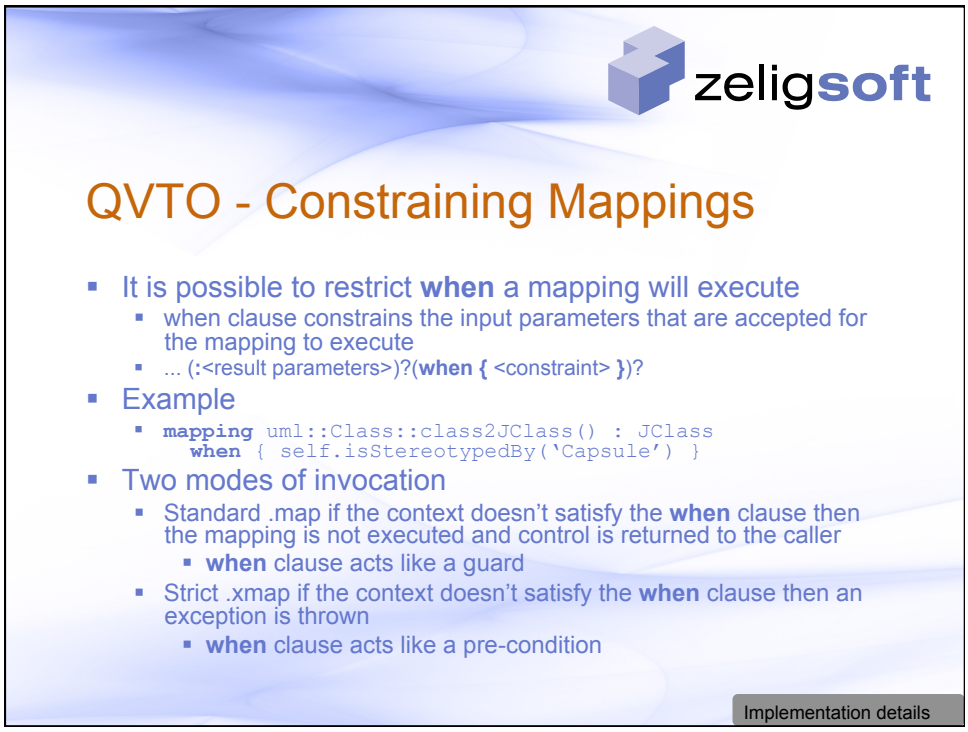


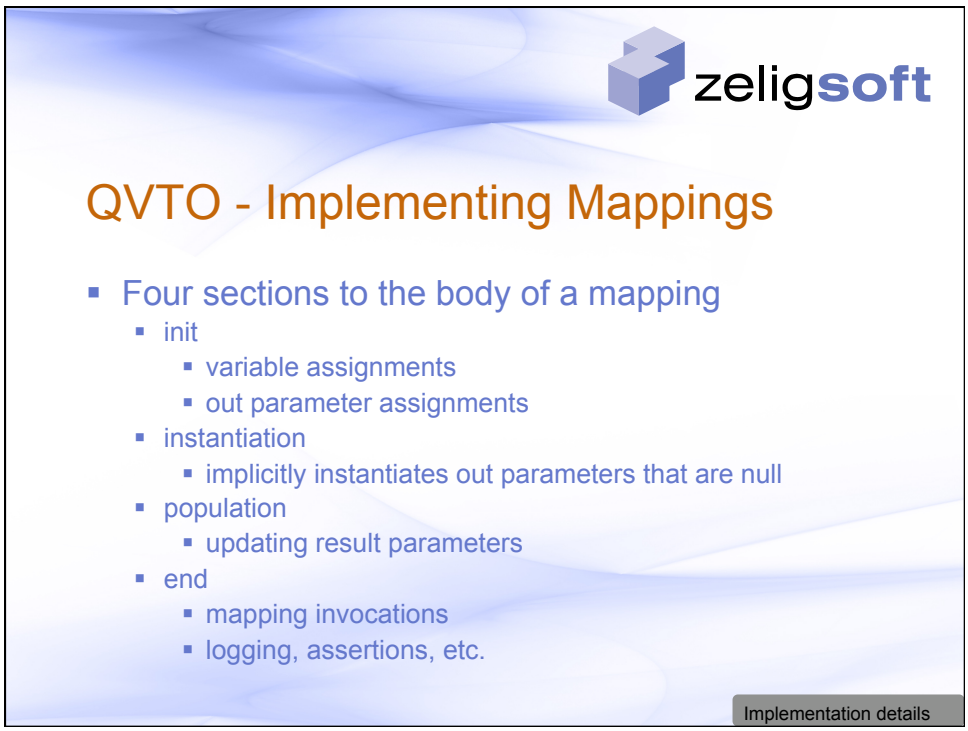


Toby McClean, Zeligsoft, 2008

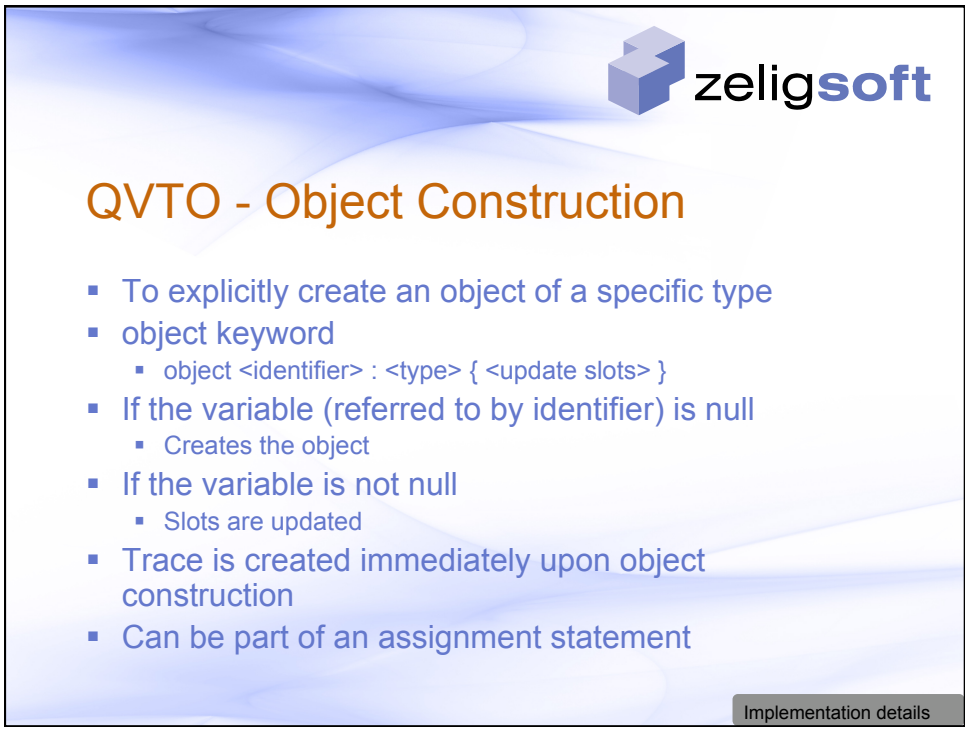

zeligsoft
QVTO - Mapping Parameters
Mapping parameters allow additional data/elements to passed into and out of the mapping
Implicit parameters
self - context of mapping
result - target of mapping
Direction
in - object passed in with read-only access
out - value set by mapping
inout - object passed in with readwrite access
Syntax
<direction> <name> : <type>
Example
in prefix : String
out elements : Sequence(ModelElement)
Implementation details

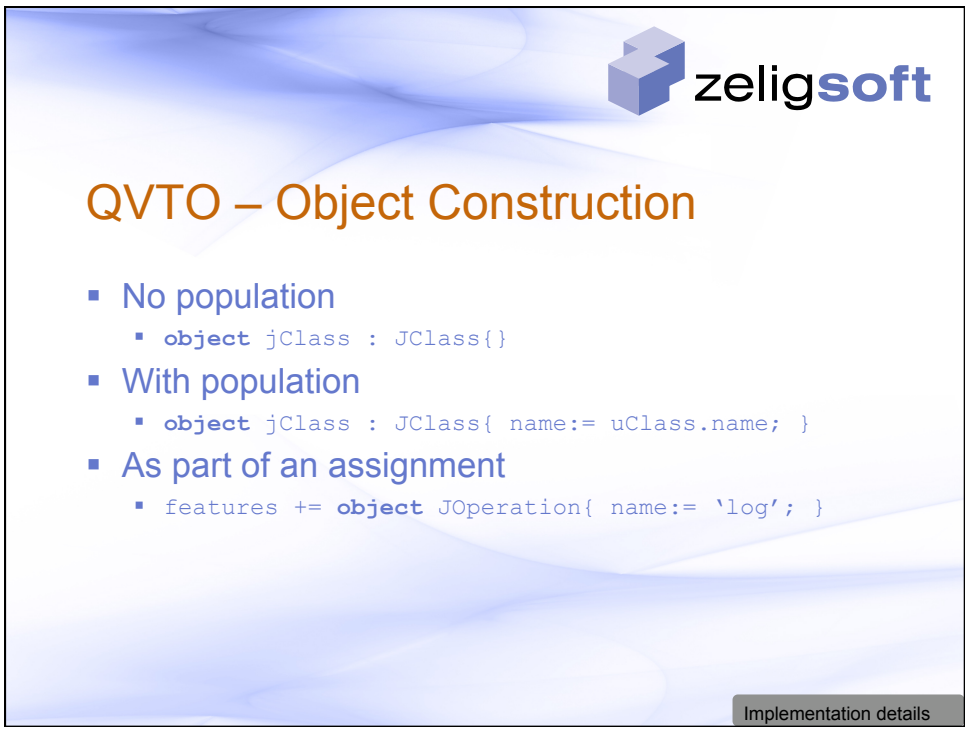

zeligsoft
QVTO - Invoking Mapping
A mapping is invoked on an object whose type complies with the context type of the mapping
Special operation on object whose parameter is a mapping
<object>.map <mapping with context type>()
capsule.map capsuleToClass();
Assuming that capsule is UML::Class
Values can be passed into the mapping
capsule.map capsuleToClass(true);
Implementation details





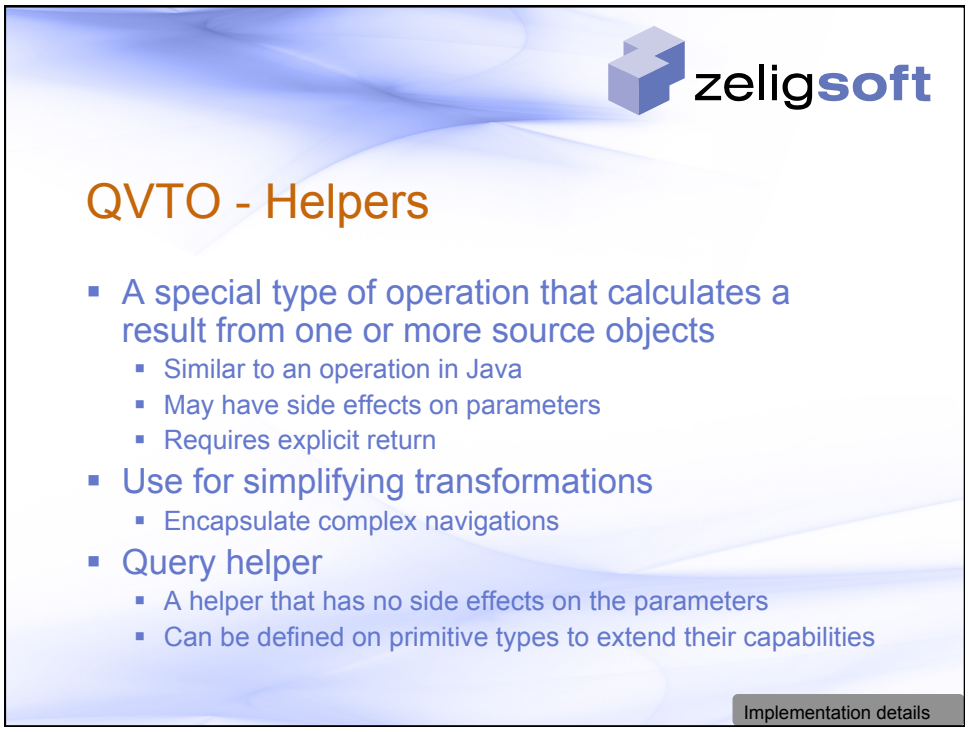
Toby McClean, Zeligsoft, 2008

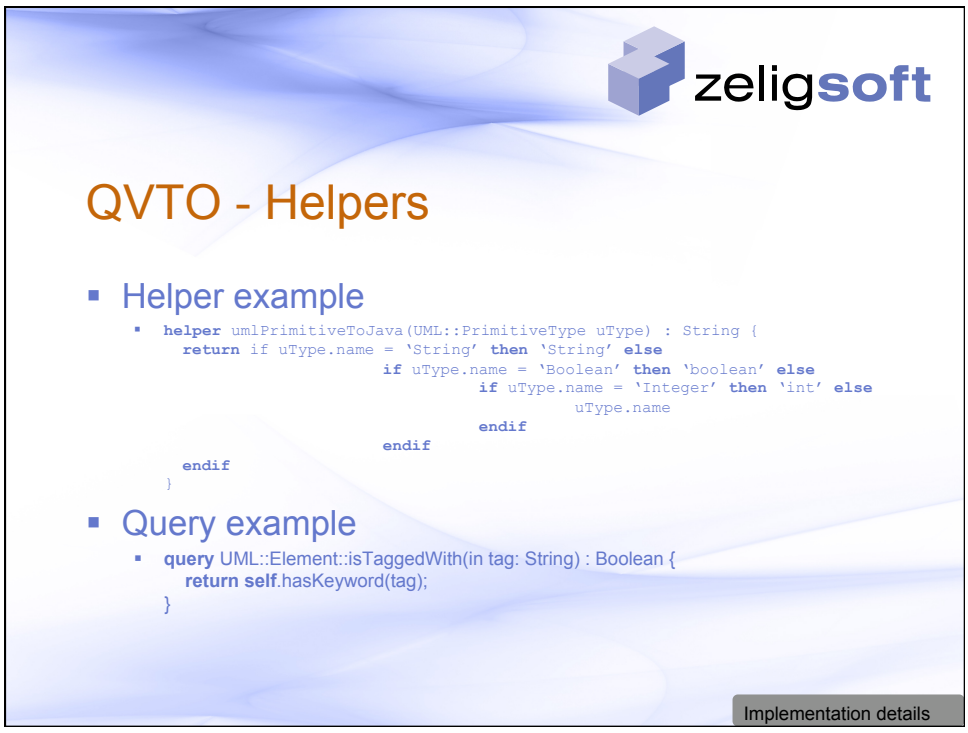




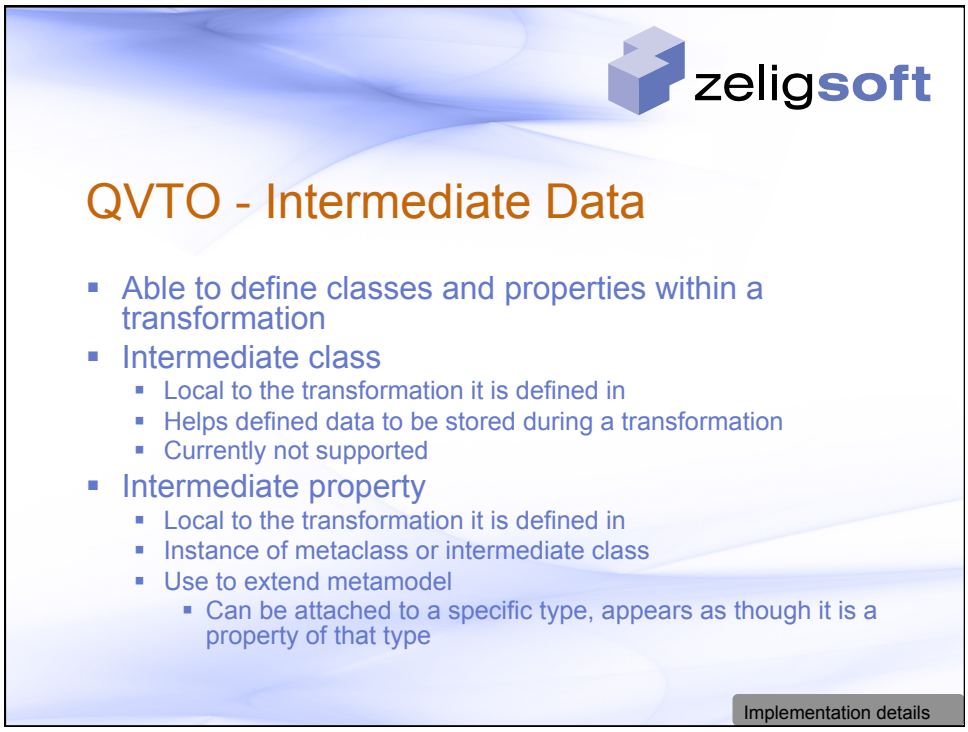
Toby McClean, Zeligsoft, 2008

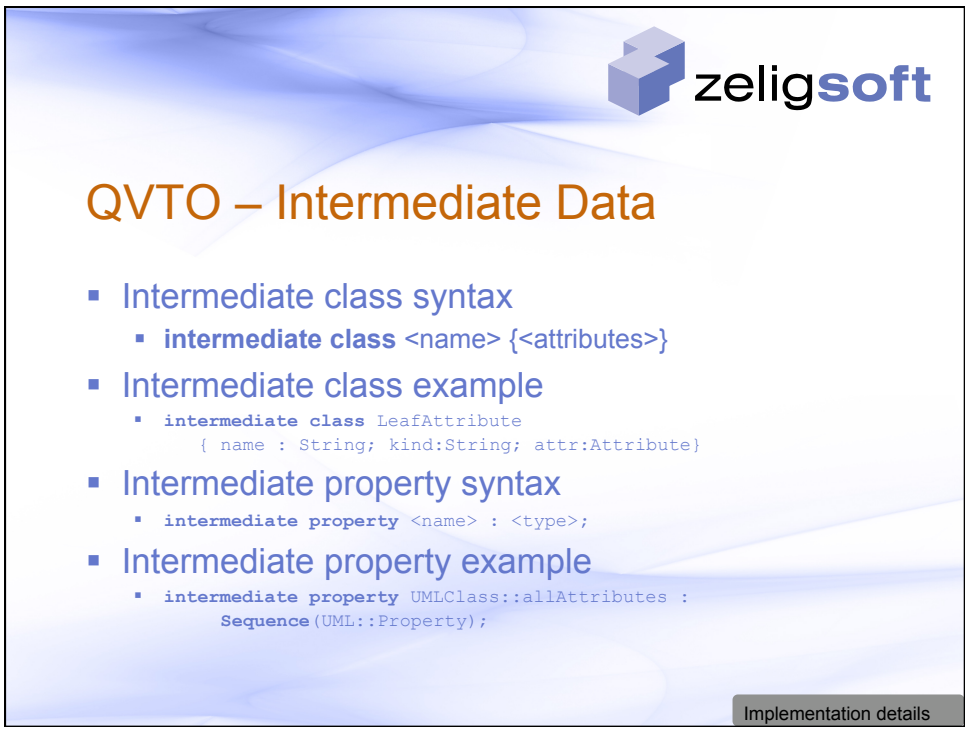
Toby McClean, Zeligsoft, 2008



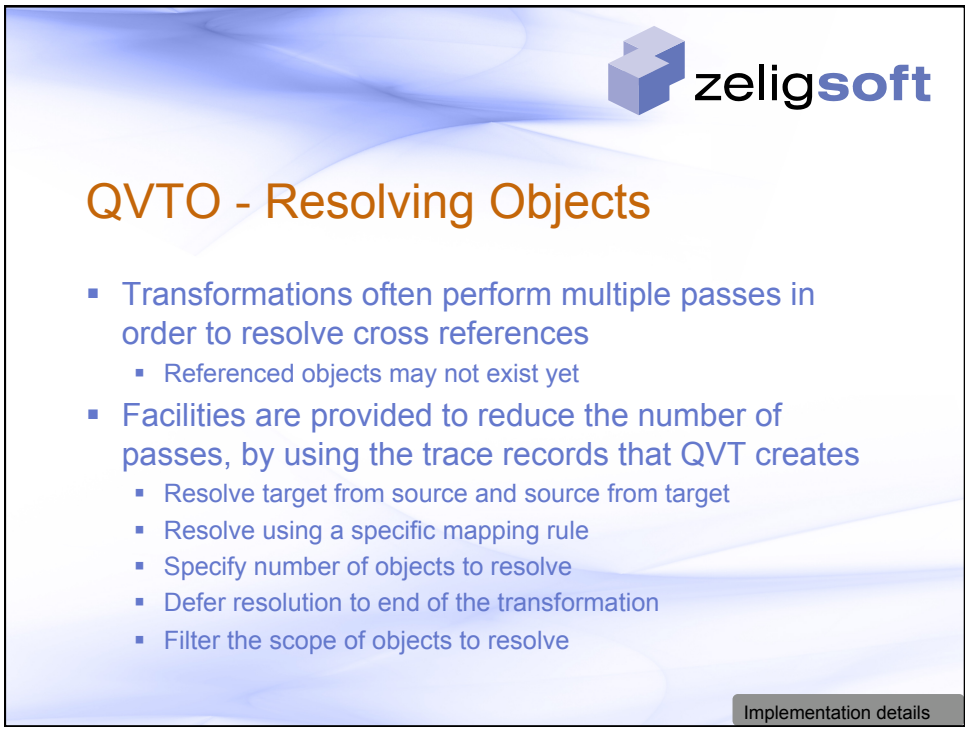


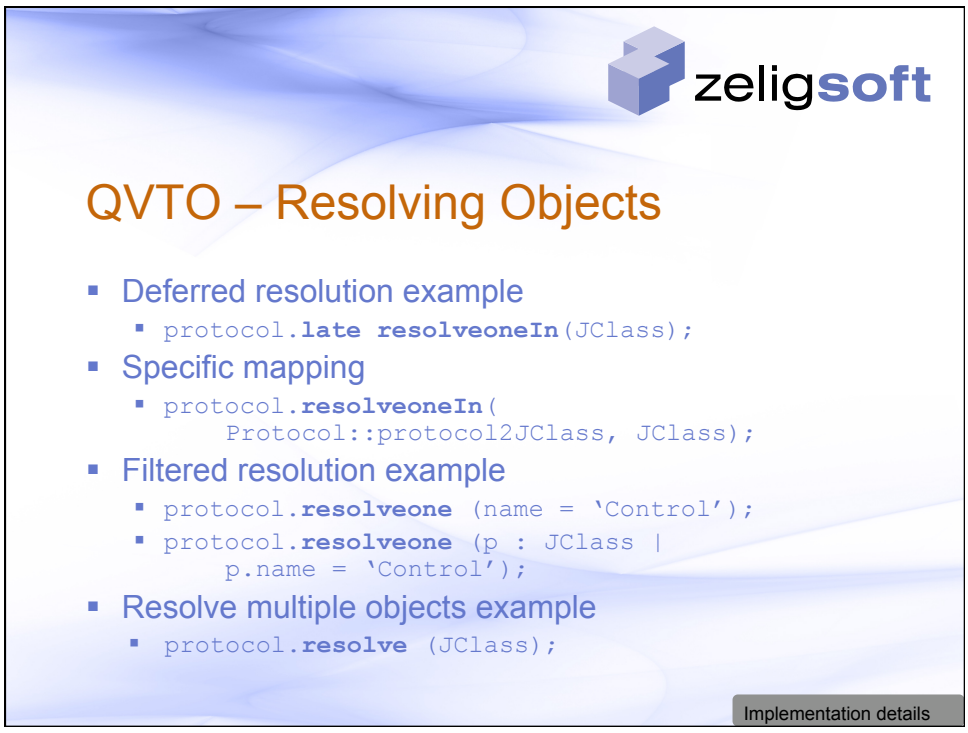
Toby McClean, Zeligsoft, 2008





Toby McClean, Zeligsoft, 2008





Toby McClean, Zeligsoft, 2008

zeligsoft

QVTO – Transformation reuse

- Composition
  - Explicit instantiation and invocation
  - **transformation** ROOM2JavaExt(in room : ROOM, out java : JAVA)
    access transformation ROOM2Java(in ROOM, out JAVA)

```
main() {
  var base :=  new ROOM2Java(room, java);
  base.transform();
}
```

- Extension
  - Implicit instantiation
  - Ability to override a mapping in the extended transformation, which will be used in place of the mapping int he extended transformation
  - **transformation** ROOM2JavaExt(in room : ROOM, out java : JAVA)
    extends transformation ROOM2Java(in ROOM, out JAVA)

Implementation details

zeligsoft

QVTO – Mapping Reuse

- Inheritance
  - Inherited mapping is executed after the init section
  - mapping A::AtoSubB() : SubTypeofB inherits A::AtoB {…}
    - Executes init of AtoSubB then AtoB then the rest of AtoSubB
- Merge
  - List of mappings executed in sequence after end section
  - mapping A::AtoB() : B
    merges A::toSuperB1, A::toSuperB2 {}
    - Executes AtoB then toSuperB1, then toSuperB2
- Parameters of inherited/merged mappings must match

Implementation details

Toby McClean, Zeligsoft, 2008

zeligsoft

QVTO - Disjuncts

- An ordered list of mappings
  - First mapping in the list whose guard (type and when clause) is satisfied is executed
  - Null is returned if no mapping in the list is executed
- Example

```
mapping PackageableElement::roomElement2JCompilationUnit()
: JCompilationUnit
   disjuncts Actor::actor2JCompilationUnit,
             Protocol::protocol2JCompilationUnit {
}
```

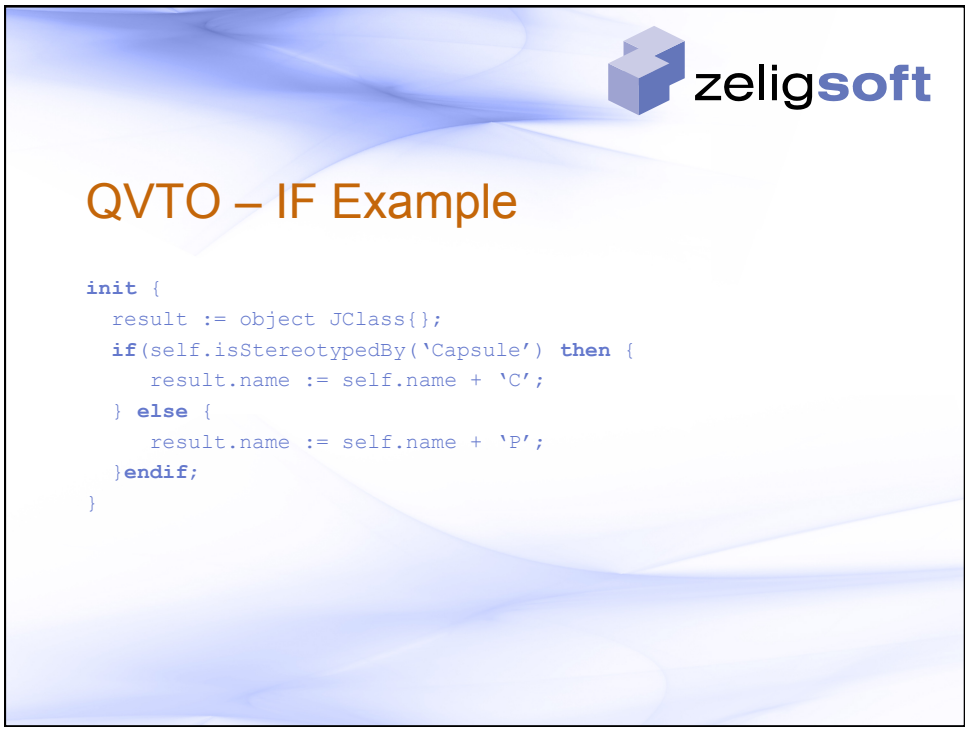
Implementation details

zeligsoft

QVTO - Control

- while loop
  - execute a block until a condition is false
- foreach
  - iterate over a block
  - can add filter to the iterator
- while and foreach support
  - break
  - continue
- if-then-else
  - if(<cond>){<block>}
    elif(<cond>){<block>}
    else{<block>}

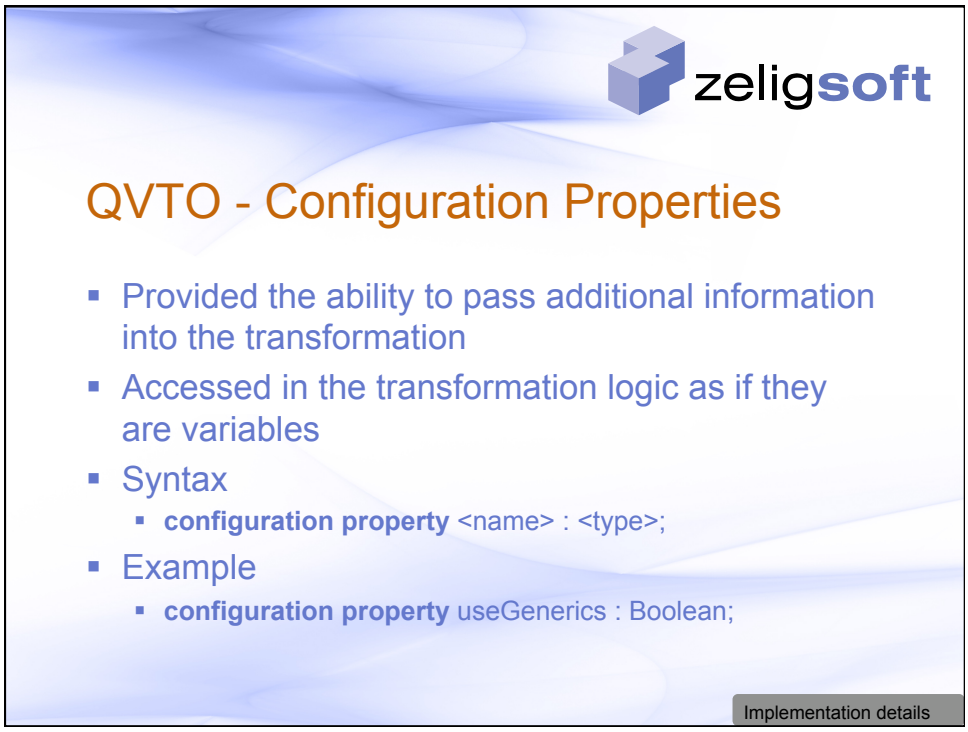
Implementation details

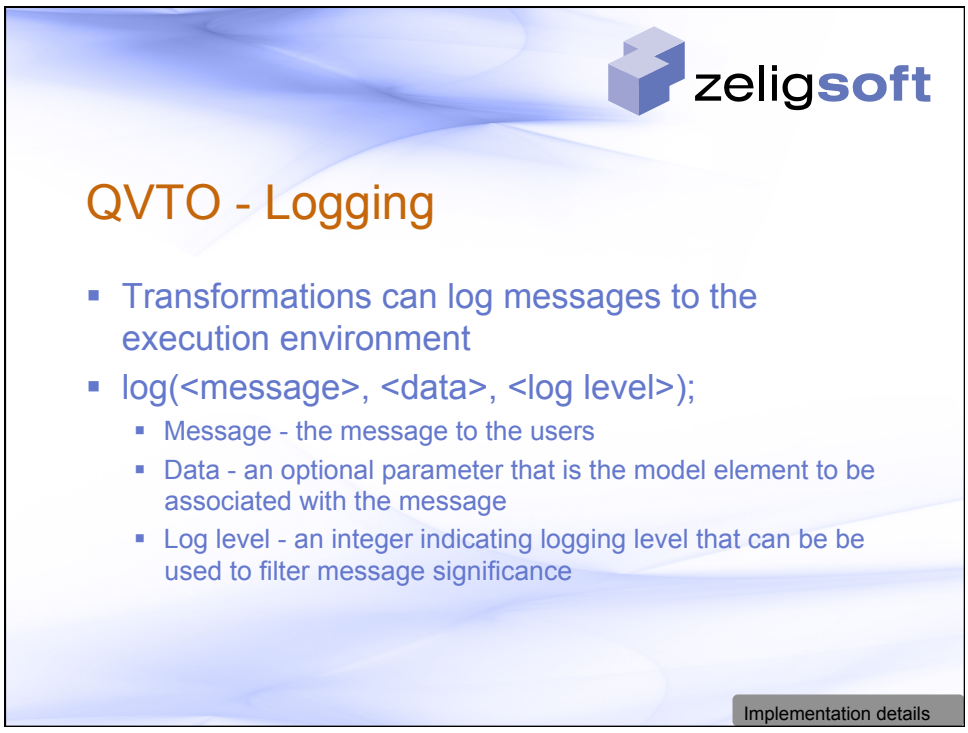


Toby McClean, Zeligsoft, 2008


zeligsoft
QVTO - More than 1 Target Model
When multiple target models are specified, need to be able to indicate which one a model is instantiated in
object <type>@<target model>{}
mapping <type>::<name> : <type>@<target model> {}
Implementation details


zeligsoft
QVTO - Libraries
A QVTO library
contains definitions of specific types
contains queries, constructors, and mappings
A library must explicitly included
By extending
By accessing
Blackboxing
Defining a library in a language other than QVT
Implementation details


zeligsoft
QVTO - Configuration Properties
Provided the ability to pass additional information into the transformation
Accessed in the transformation logic as if they are variables
Syntax
configuration property <name> : <type>;
Example
configuration property useGenerics : Boolean;
Implementation details


zeligsoft
QVTO - Logging
Transformations can log messages to the execution environment
log(<message>, <data>, <log level>);
Message - the message to the users
Data - an optional parameter that is the model element to be associated with the message
Log level - an integer indicating logging level that can be be used to filter message significance
Implementation details

zeligsoft

QVTO - Working with Profiles

- QVTO provides no special operators for working with UML2 profiles
- Transformations and mappings use the API defined in the UML2 metamodel
- A library can be built to simplify the UML2 stereotype API
- Example
  - query UML::Element::isStereotypedBy(in qualifiedName : String) : Boolean {
    return self.getAppliedStereotype(qualifiedName) <> null;
    }

zeligsoft

QVTO - Extensions

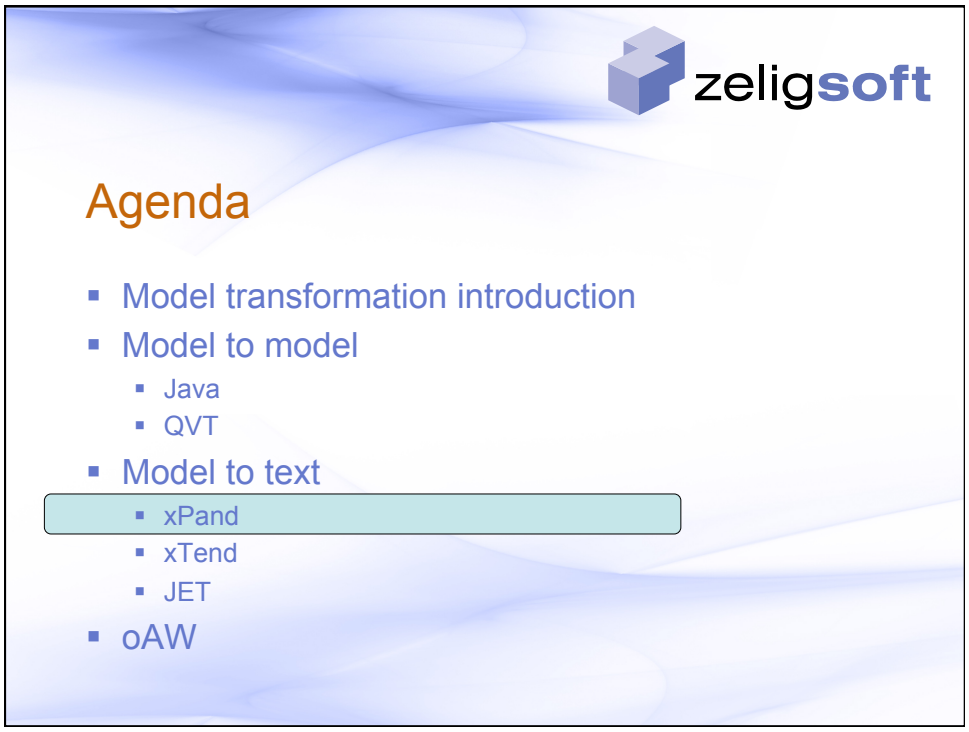
- Blackbox libraries are defined through the org.eclipse.m2m.qvt.oml.ocl.libraries extension point
- Must have a static class Metainfo
  - specifies the parameters of the transformation
- Enables QVTO to leverage the power of a 3GL like Java

QVTO - Programatically Invoking

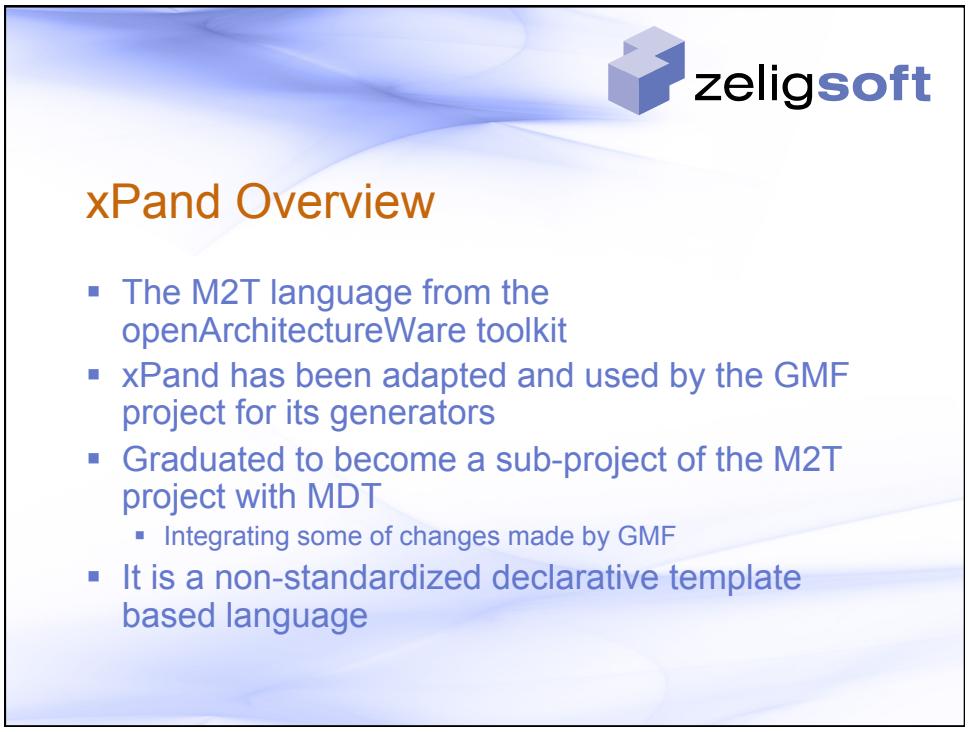
```
URI transformation =
  URI.createURI("platform:/resource/com.eett.exercises.m2m.qvt/transforms/
Room2Java.qvto");
IFile qvtFile = getIFile(transformation);
IContext qvtContext = new Context();
QvtInterpretedTransformation trans =
   new QvtInterpretedTransformation(qvtFile);
EObject inputModel =
   getInput(
     URI.createURI("platform:/resource/com.eett.exercises.m2m.qvt/testModels/
packageableElementTest.room"));
EObject[] inputs = {inputModel};
TransformationRunner.In input =
   new TransformationRunner.In(inputs, qvtContext);
TransformationRunner.Out output = null;

try {
   output = trans.run(input);
} catch (MdaException e) {
   out.println("Error running transformation!");
   out.println(e.getMessage());
}
```

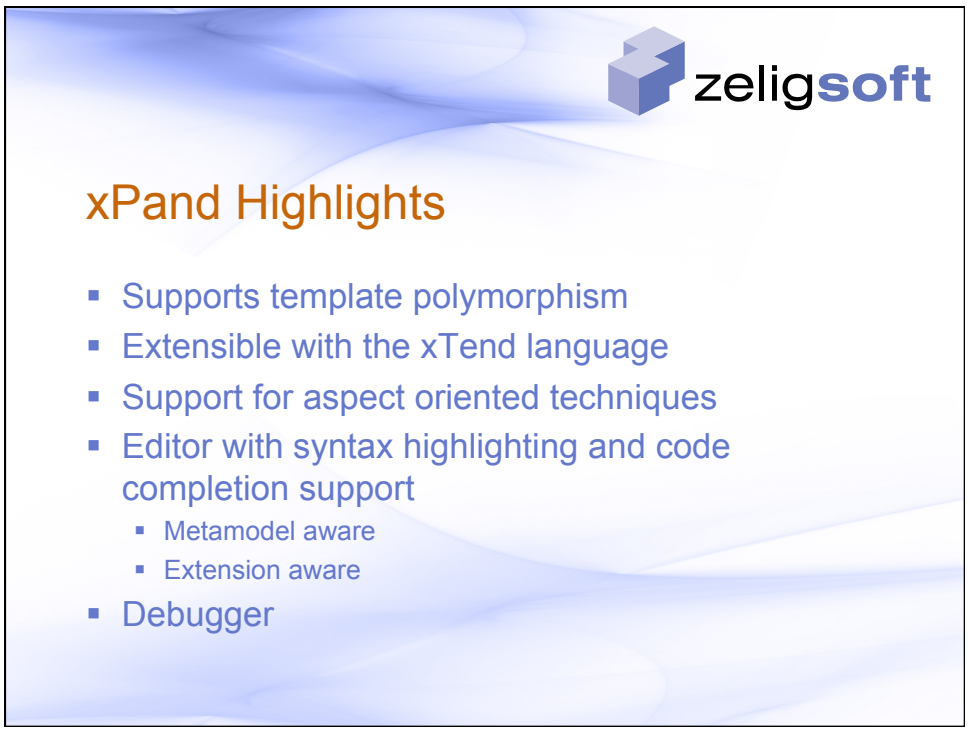
Summary

- In this module we explored
  - QVT mappings
  - Invoking QVT
  - Details on QVT usage


zeligsoft
Agenda
Model transformation introduction
Model to model
Java
QVT
Model to text
xPand
xTend
JET
oAW


zeligsoft
Eclipse Transformation Technologies
M2T with xPand

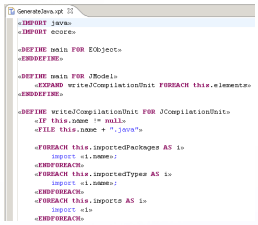




Toby McClean, Zeligsoft, 2008

zeligsoft

Developing xPand Transformations



- Editor
  - Syntax highlighting
  - Code completion
- Model navigation
  - Java based syntax for model navigation
  - Has operators for working with collections
- Focus on transformation logic
  - Execution infrastructure left to the xPand runtime

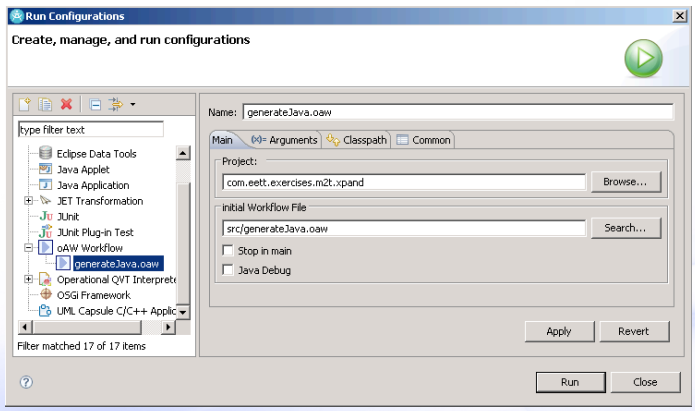
zeligsoft

Executing an xPand Transformation

- Using an oAW workflow
  - More about workflows later
  - Can use integration with the Run as... or Debug as... oAW workflow
- Write a Java application
  - Call the transformation explicitly in code
    - Use the XPandFacade class
  - Use the Run as... or Debug as... Java Application

zeligsoft

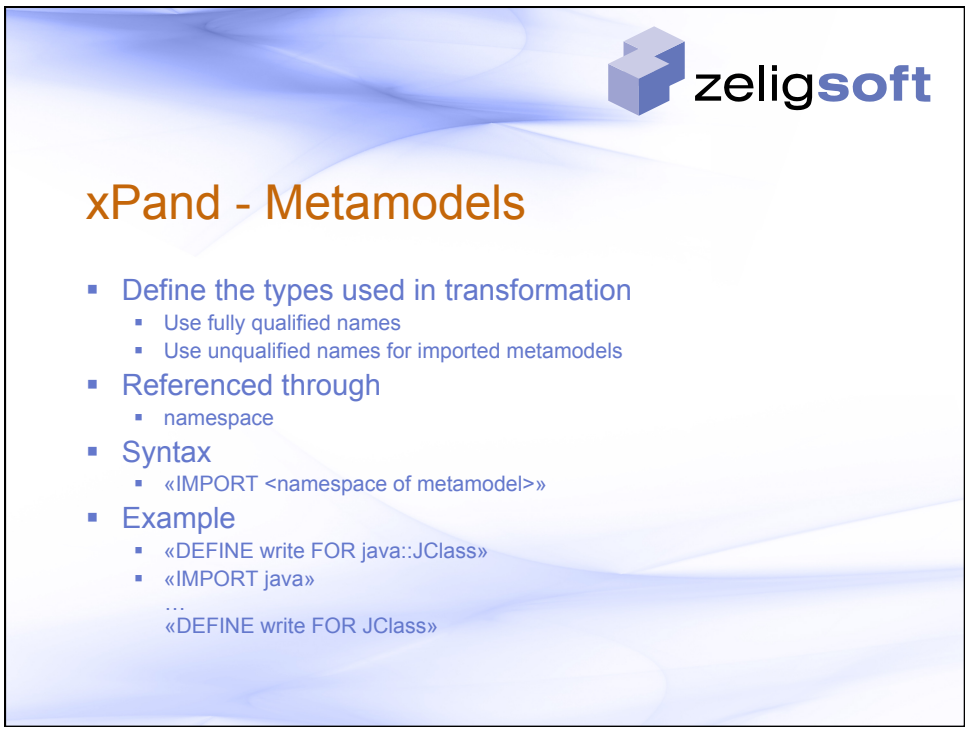
Executing an xPand Transformation



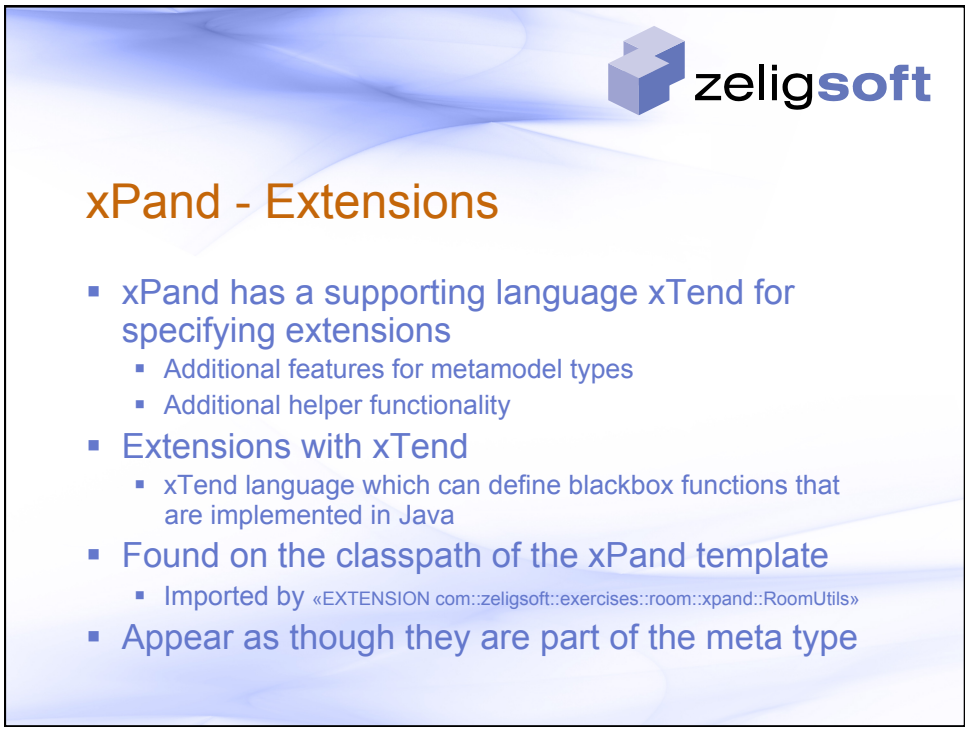
zeligsoft

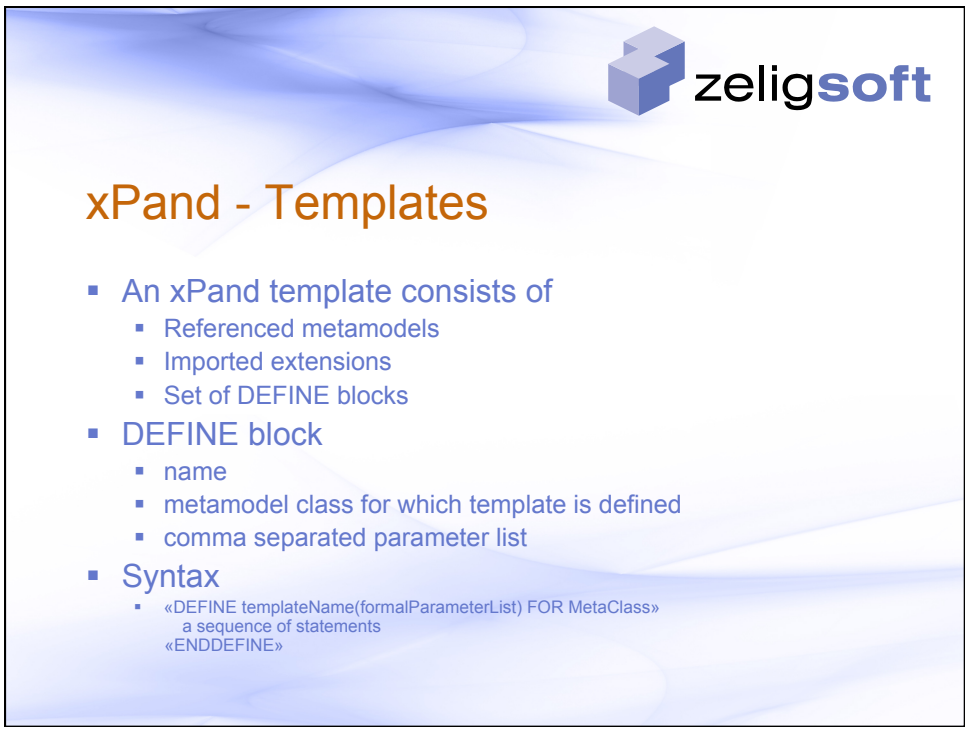
Integrating with Workbench

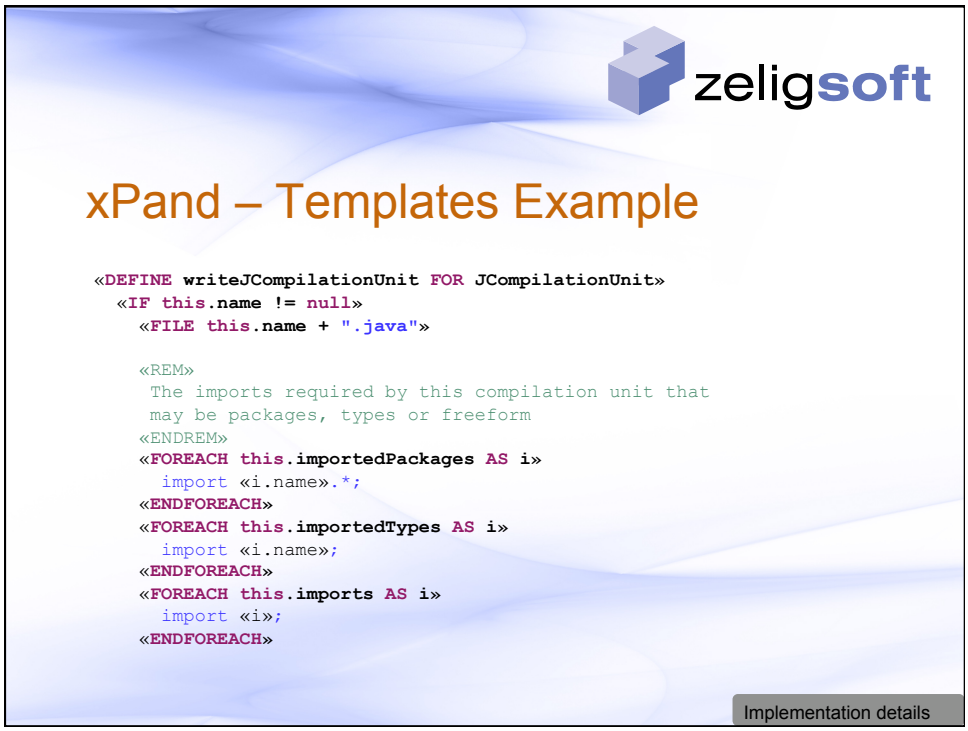
- Contribute an action to the editor's menu
  - Implement the action handler
    - Call the workflow passing in the selected element as the source
- Alternative is append to an existing action
  - e.g. the ROOM to Java transformation
- Test
  - In runtime workbench
- Deploy
  - Publish the plug-in

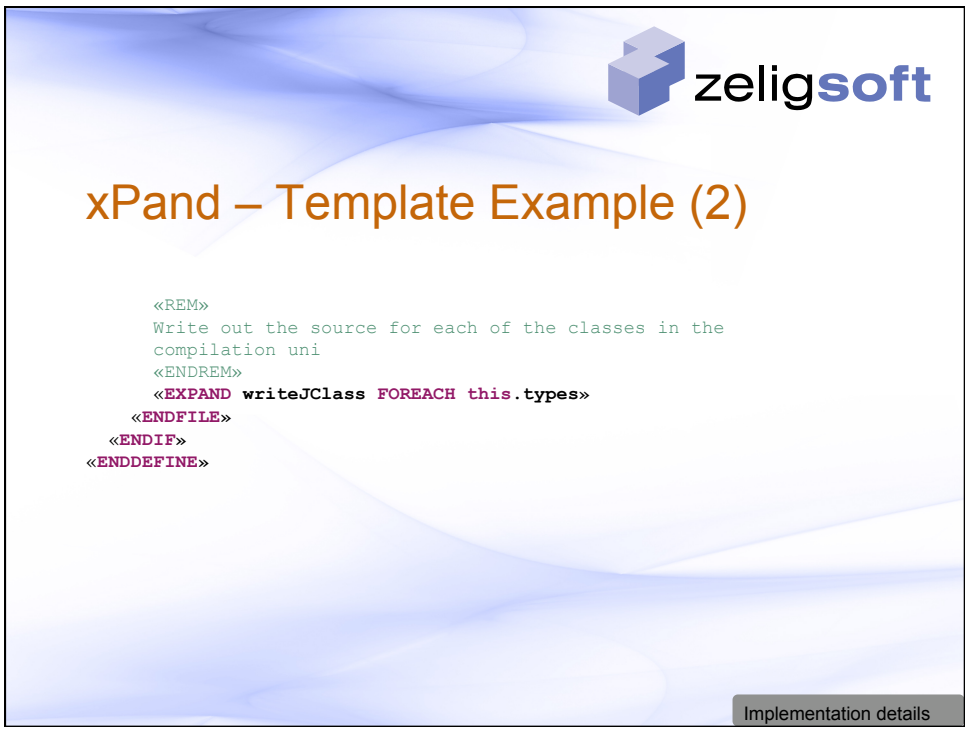


Toby McClean, Zeligsoft, 2008

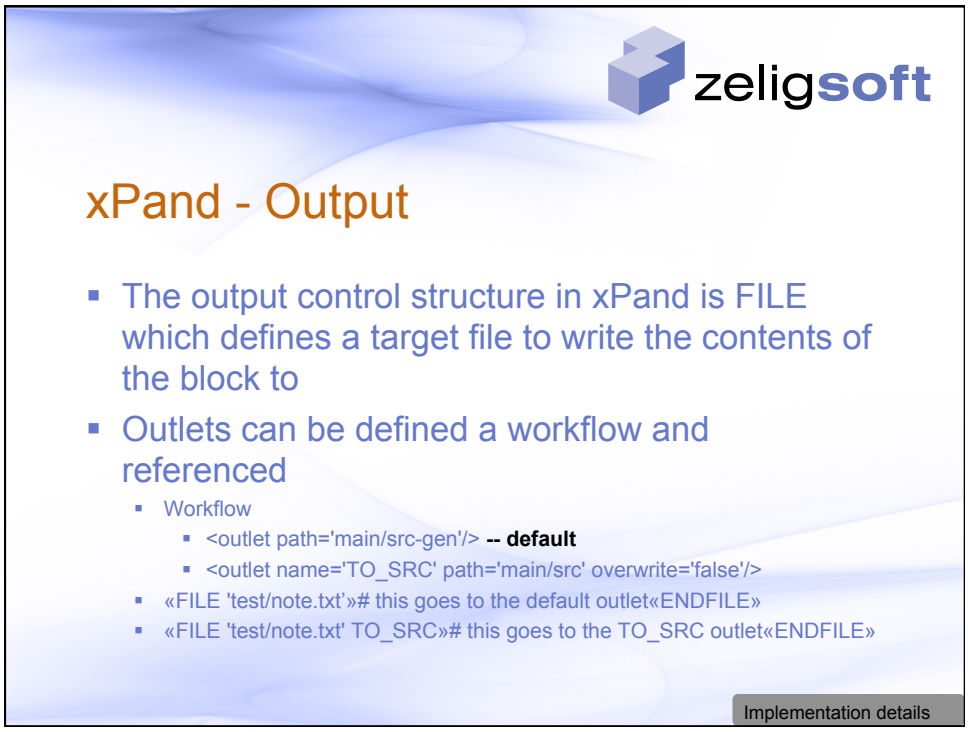

zeligsoft
xPand - Extensions
xPand has a supporting language xTend for specifying extensions
Additional features for metamodel types
Additional helper functionality
Extensions with xTend
xTend language which can define blackbox functions that are implemented in Java
Found on the classpath of the xPand template
Imported by «EXTENSION com::zeligsoft::exercises::room::xpand::RoomUtils»
Appear as though they are part of the meta type


zeligsoft
xPand - Templates
An xPand template consists of
Referenced metamodels
Imported extensions
Set of DEFINE blocks
DEFINE block
name
metamodel class for which template is defined
comma separated parameter list
Syntax
«DEFINE templateName(formalParameterList) FOR MetaClass»
a sequence of statements
«ENDDEFINE»

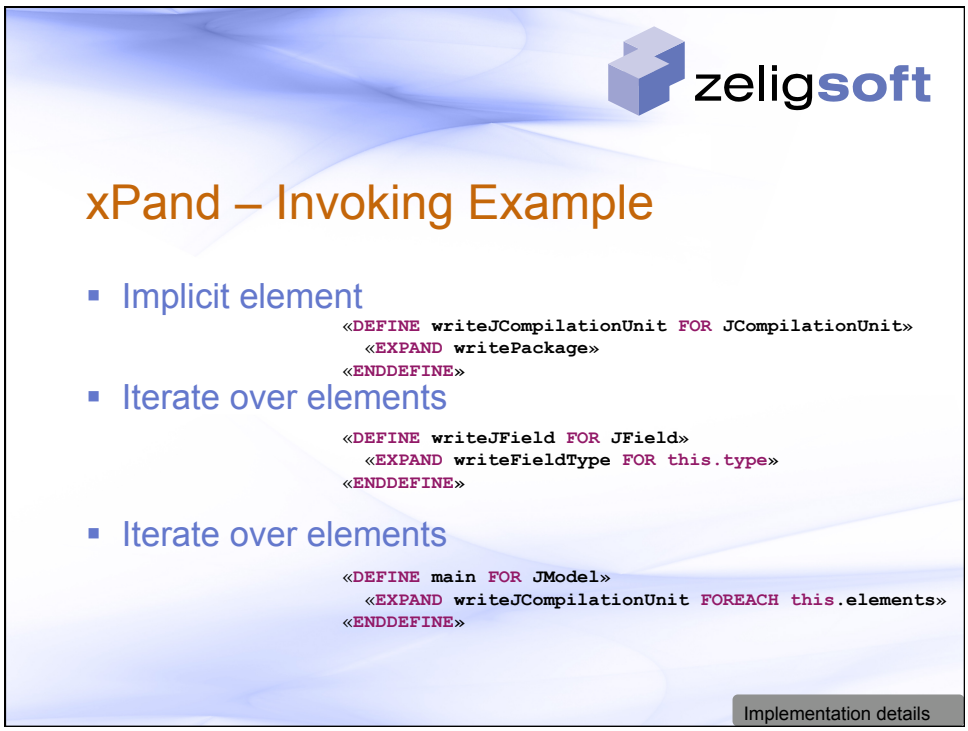




Toby McClean, Zeligsoft, 2008

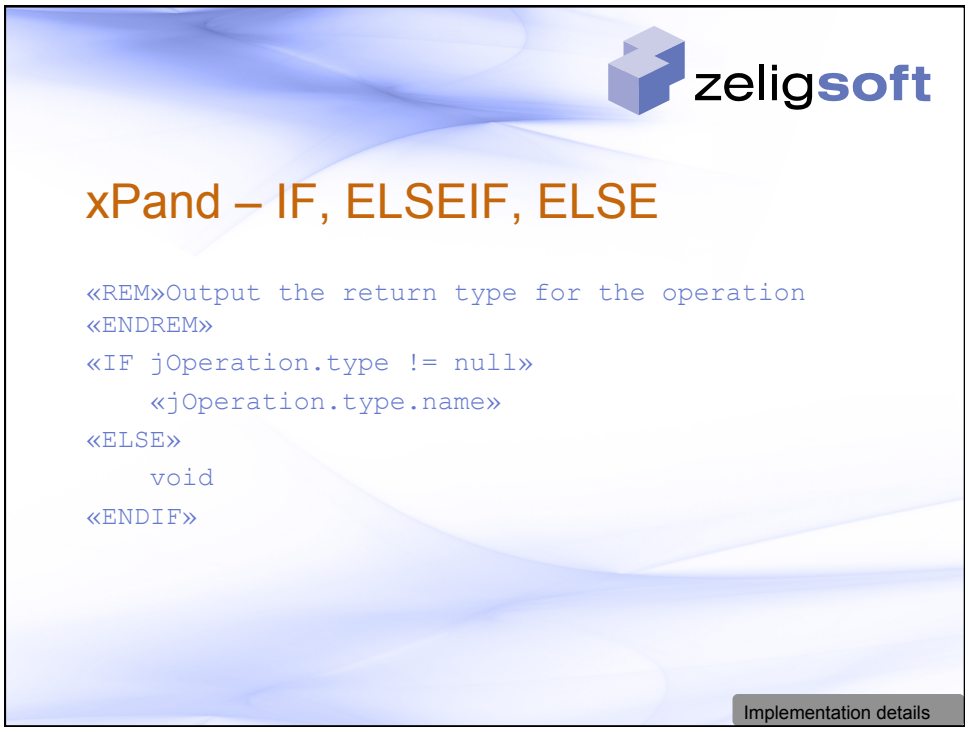

zeligsoft
xPand - Output
The output control structure in xPand is FILE which defines a target file to write the contents of the block to
Outlets can be defined a workflow and referenced
Workflow
<outlet path='main/src-gen'/> -- default
<outlet name='TO_SRC' path='main/src' overwrite='false'/>
«FILE 'test/note.txt'»# this goes to the default outlet«ENDFILE»
«FILE 'test/note.txt' TO_SRC»# this goes to the TO_SRC outlet«ENDFILE»
Implementation details

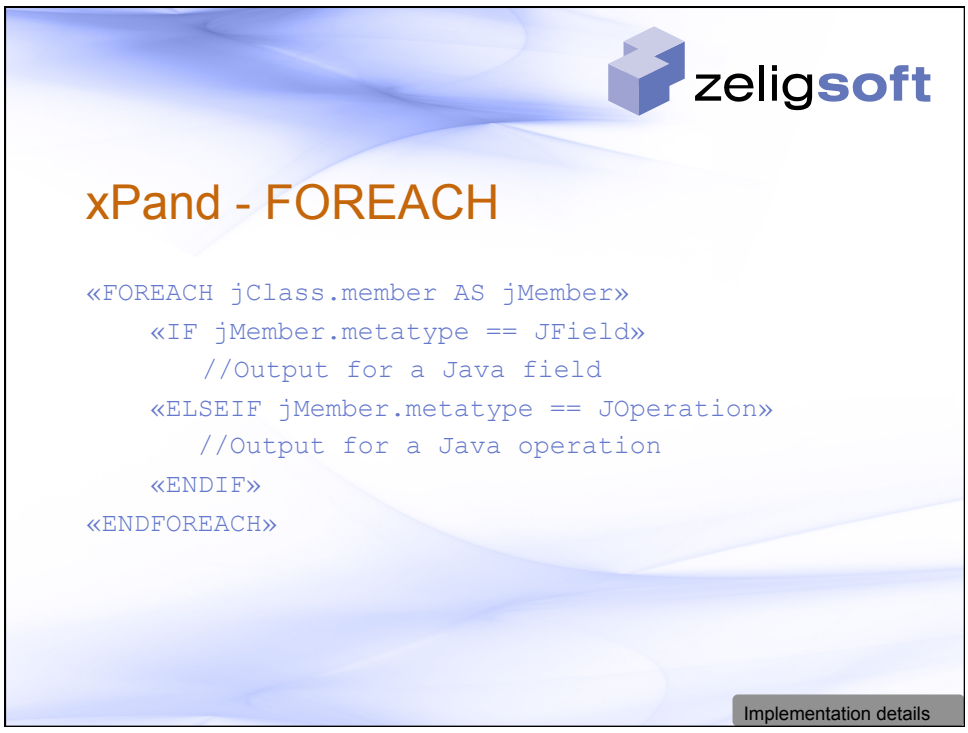

zeligsoft
xPand - Invoking a Template
The xPand language uses EXPAND to invoke a template
«EXPAND definitionName [(parameterList)]
[FOR expression | FOREACH expression [SEPARATOR expression]]»
FOR
Invokes the template on the specified element
FOREACH
Invokes the template on each element in the collection
SEPERATOR
Specifies an optional delimiter output between each invocation
Omitting FOR\FOREACH invokes with FOR this
Implementation details



Toby McClean, Zeligsoft, 2008

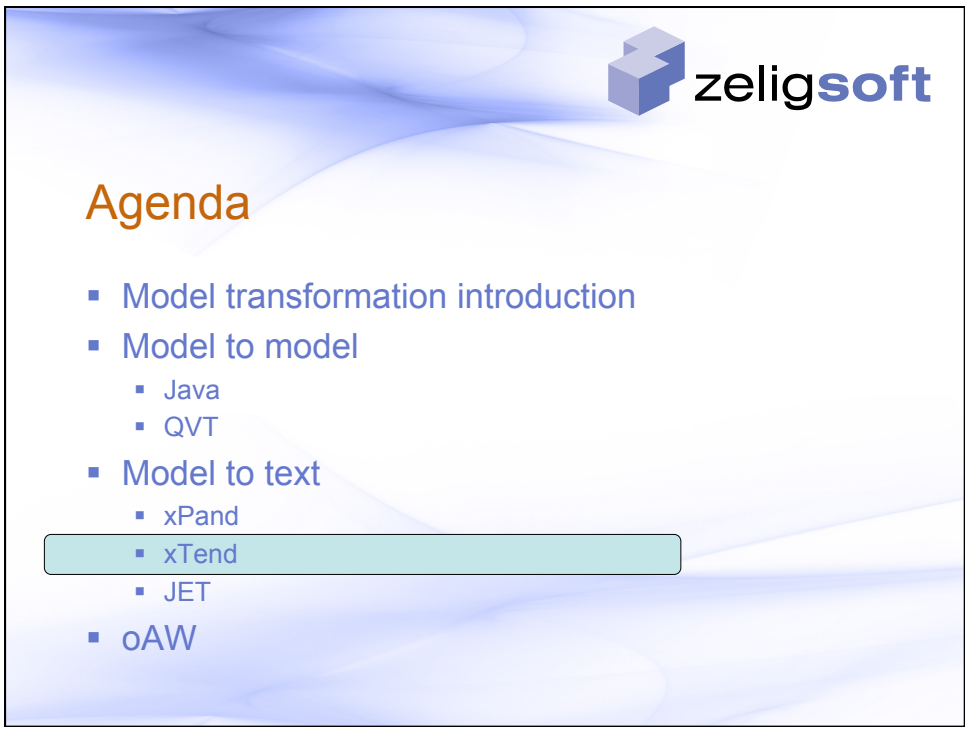
Toby McClean, Zeligsoft, 2008

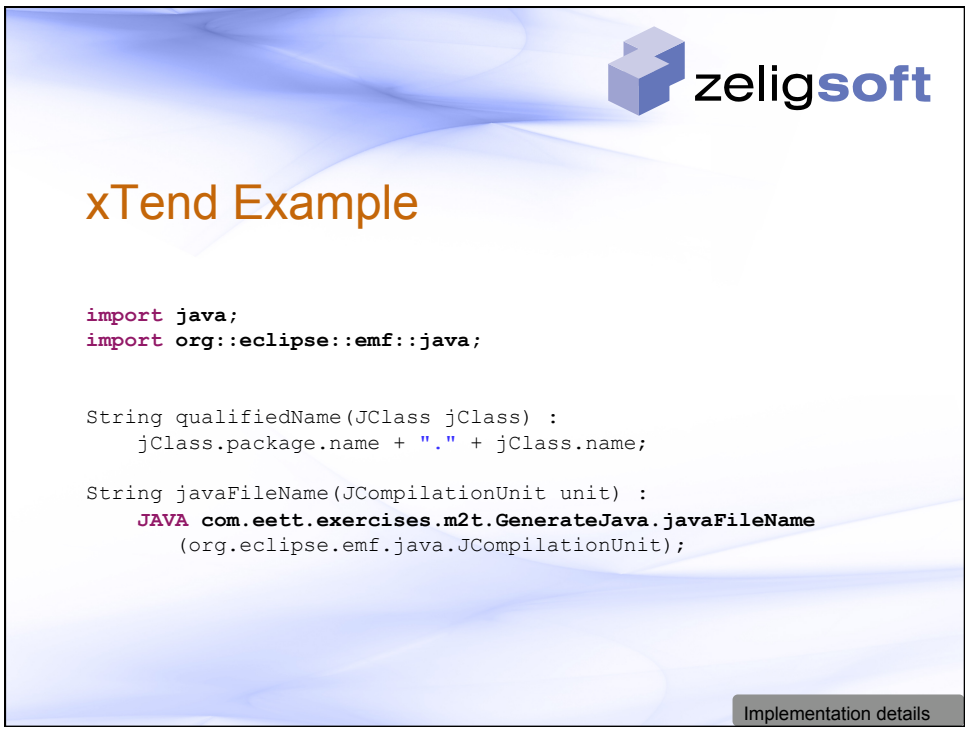




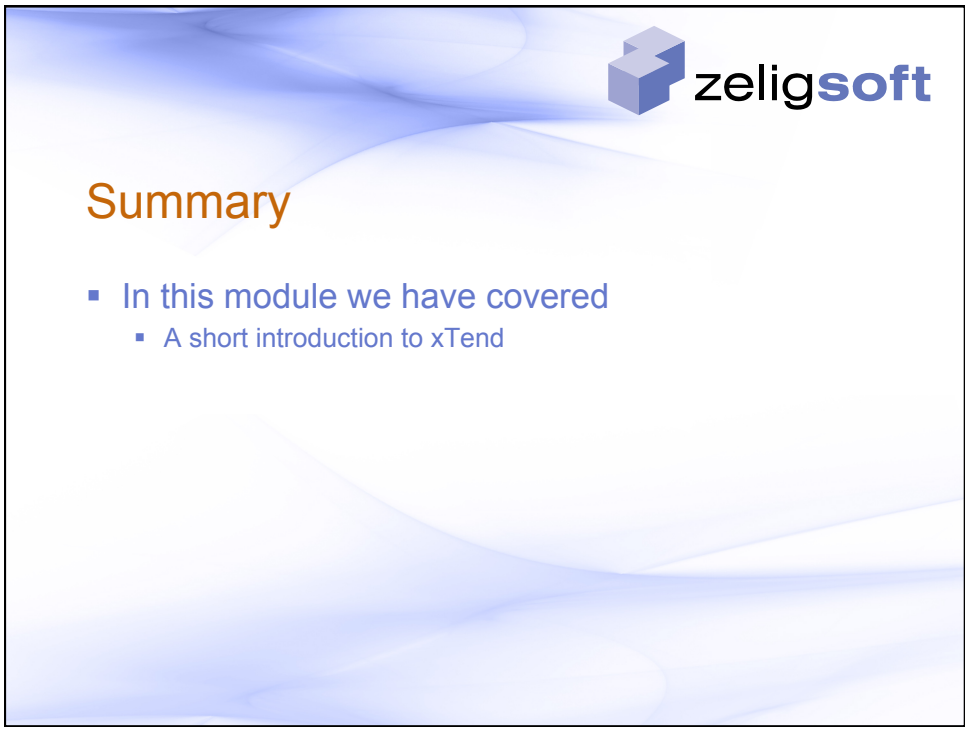
Toby McClean, Zeligsoft, 2008

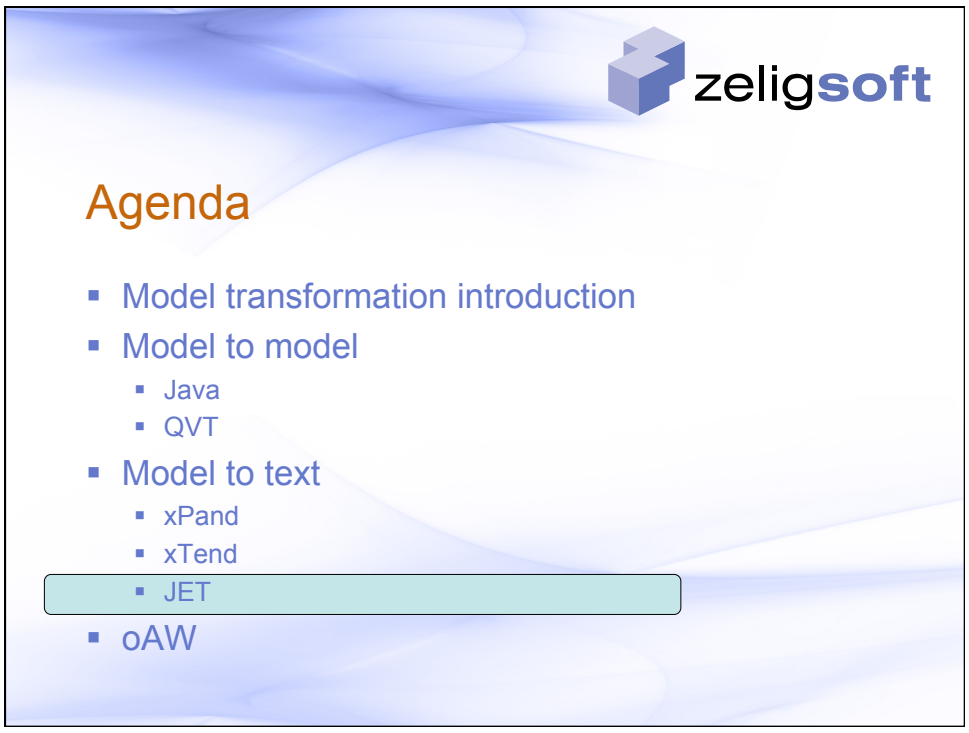

zeligsoft
Summary
In this module we explored
xPand transformations
Invoking xPand
xPandsyntax

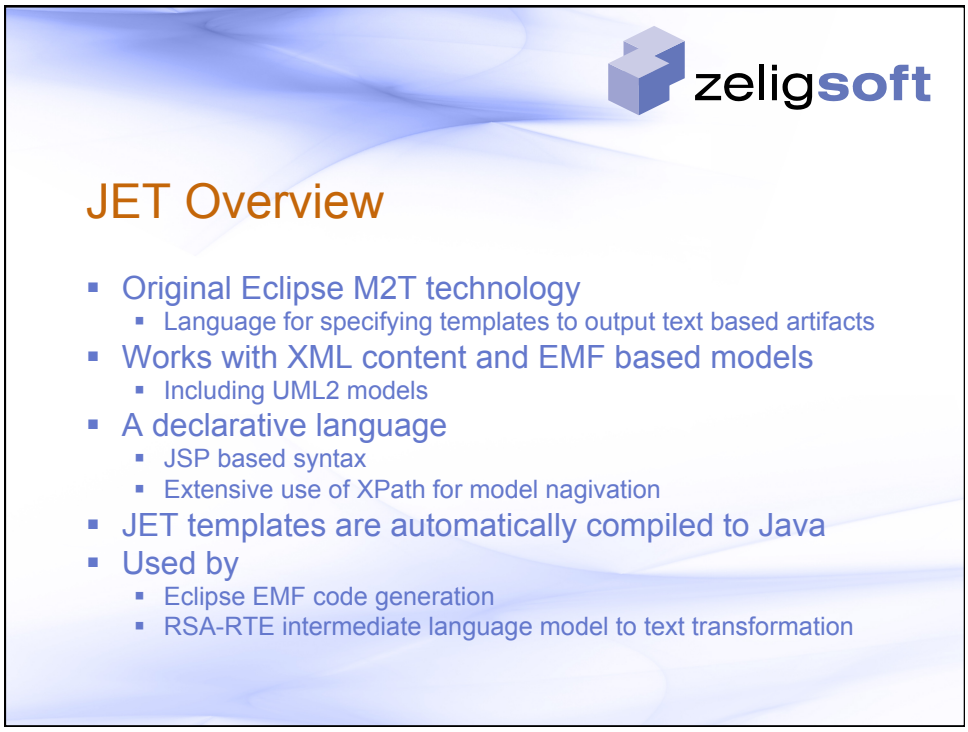

zeligsoft
Agenda
Model transformation introduction
Model to model
Java
QVT
Model to text
xPand
xTend
JET
oAW



Toby McClean, Zeligsoft, 2008


zeligsoft
Summary
In this module we have covered
A short introduction to xTend

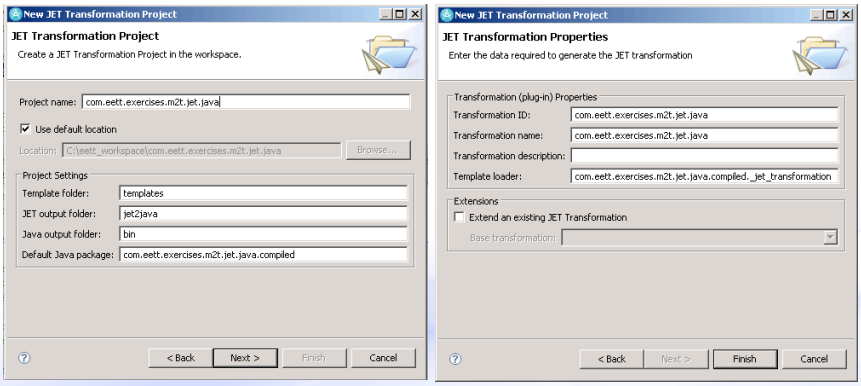

zeligsoft
Agenda
Model transformation introduction
Model to model
Java
QVT
Model to text
xPand
xTend
JET
oAW


zeligsoft
JET Overview
Original Eclipse M2T technology
Language for specifying templates to output text based artifacts
Works with XML content and EMF based models
Including UML2 models
A declarative language
JSP based syntax
Extensive use of XPath for model nagivation
JET templates are automatically compiled to Java
Used by
Eclipse EMF code generation
RSA-RTE intermediate language model to text transformation


zeligsoft
Developing a JET Transformation
JET transformations can be created by
Adding a JET transformation to an existing project
Creating a JET transformation project
Editor
Syntax highlighting
Model navigation
XPath
Focus on transformation logic
Execution infrastructure left to the compiled JET code

zeligsoft

Developing a JET Transformation



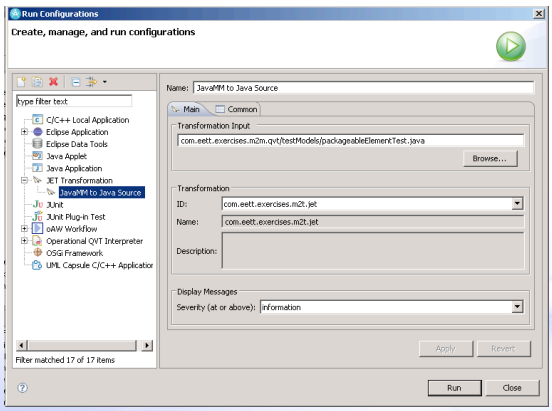
zeligsoft

Executing a JET Transformation

- The JET project integrates with the workbench Run framework
- Select the transformation resource in the project explorer
- Choose Run As → Run Configurations… from the context menu
- Create a new configuration under JET transformation
  - The input model is specified
  - The option to save the configuration for sharing

zeligsoft

Executing a JET Transformation

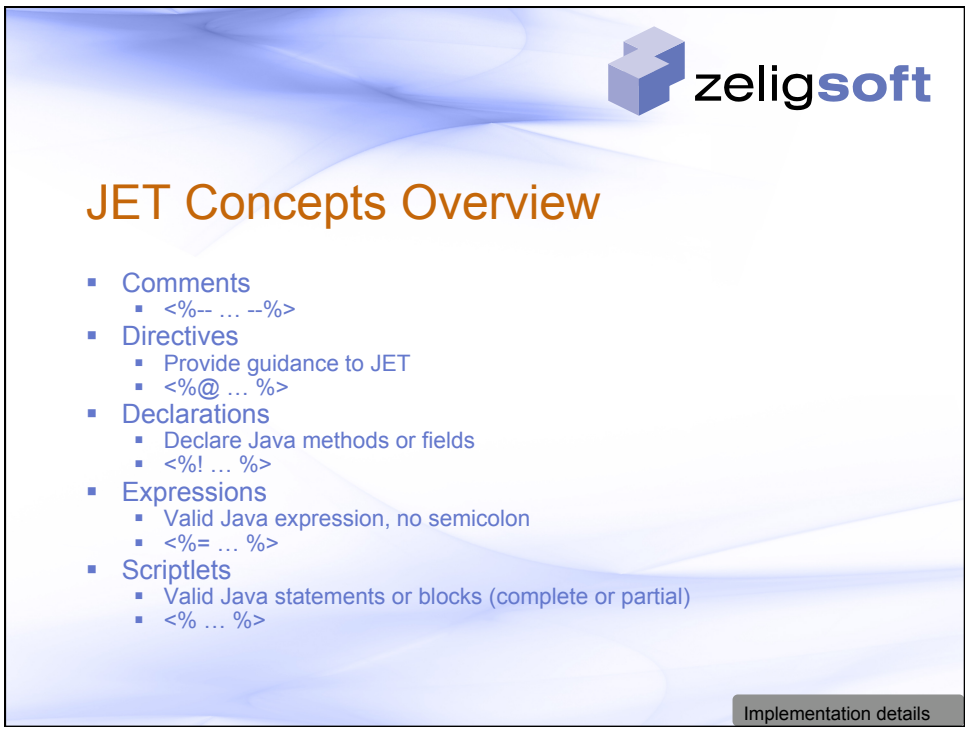


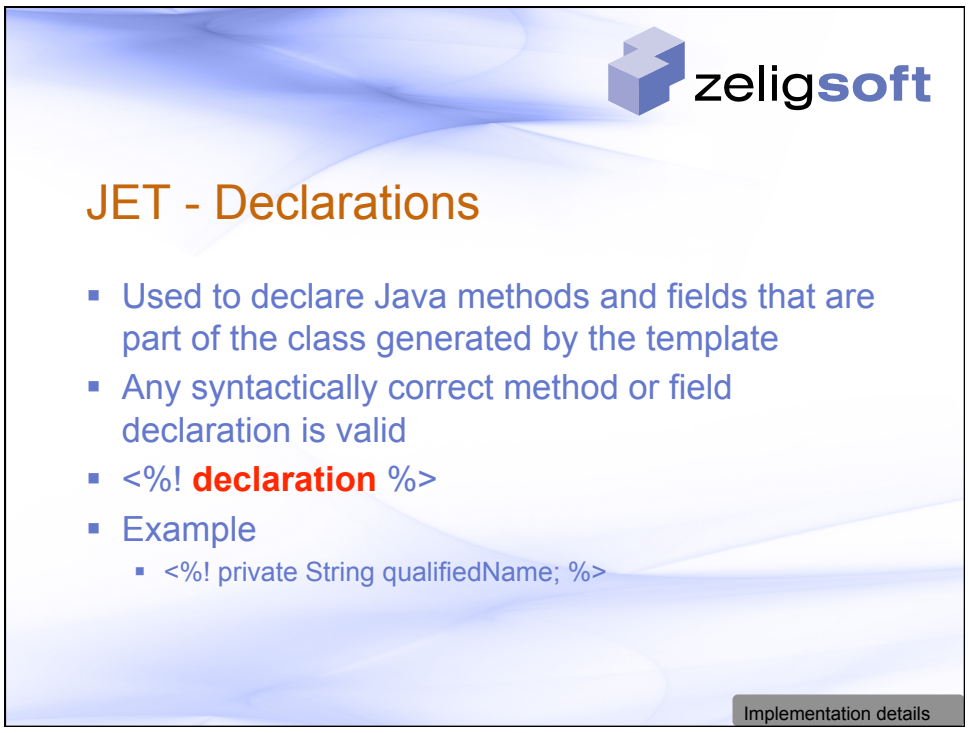
zeligsoft

Integrating with Workbench

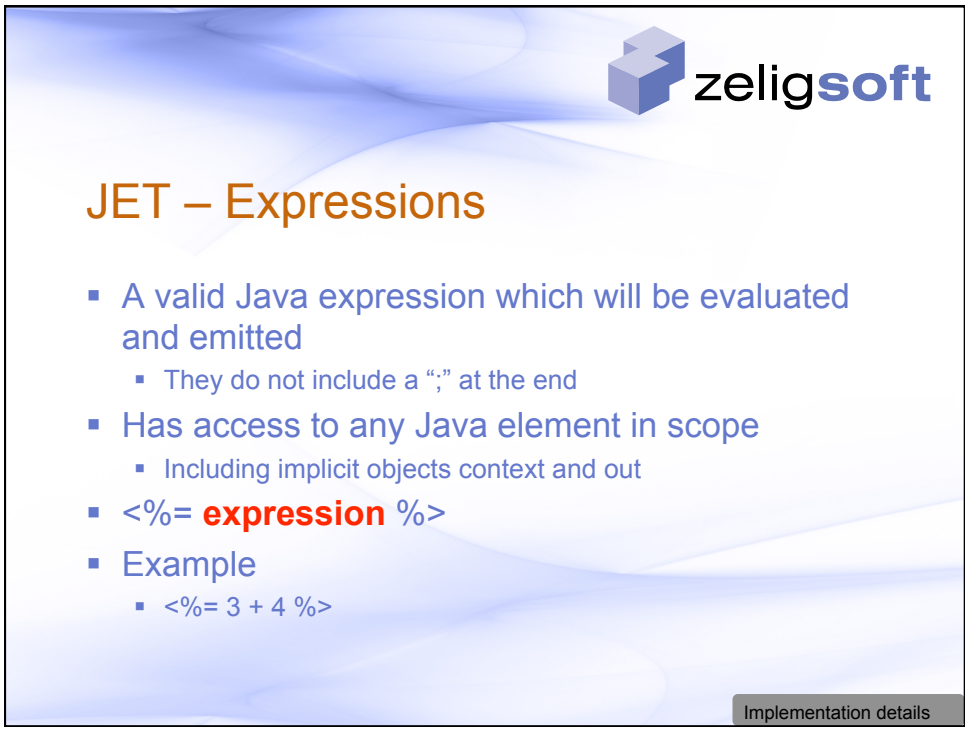
- Contribute an action to the editor's menu
  - Implement the action handler
    - Call the workflow passing in the selected element as the source
- Alternative is append to an existing action
  - e.g. the ROOM to Java transformation
- Test
  - In runtime workbench
- Deploy
  - Publish the plug-in

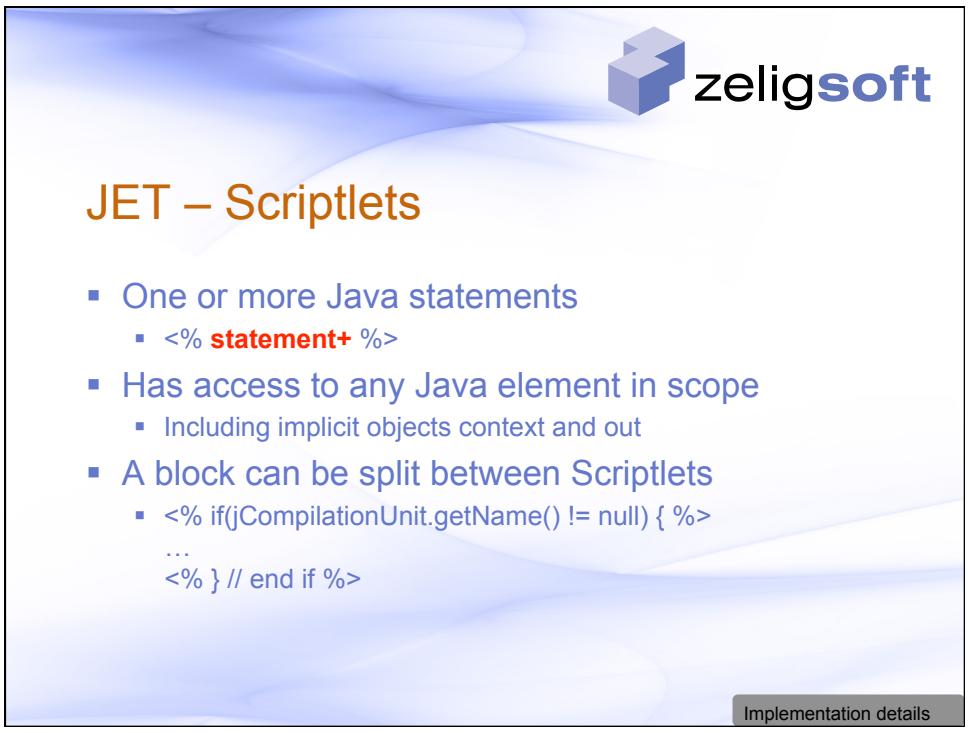

zeligsoft
Integrating with RSA-RTE
Contribute an action
to the Project Explorer context menu
to the Diagram Editor context menu
Implement action handler
Class that implements IEditorActionDelegate
Test
In runtime workbench
Deploy
Publish the plug-in


zeligsoft
JET Concepts Overview
Comments
<%-- … --%>
Directives
Provide guidance to JET
<%@ … %>
Declarations
Declare Java methods or fields
<%! … %>
Expressions
Valid Java expression, no semicolon
<%= … %>
Scriptlets
Valid Java statements or blocks (complete or partial)
<% … %>
Implementation details



Toby McClean, Zeligsoft, 2008


zeligsoft
JET – Expressions
A valid Java expression which will be evaluated and emitted
They do not include a ";" at the end
Has access to any Java element in scope
Including implicit objects context and out
<%= expression %>
Example
<%= 3 + 4 %>
Implementation details


zeligsoft
JET – Scriptlets
One or more Java statements
<% statement+ %>
Has access to any Java element in scope
Including implicit objects context and out
A block can be split between Scriptlets
<% if(jCompilationUnit.getName() != null) { %>
…
<% } // end if %>
Implementation details

zeligsoft

JET - Metamodels

- By default JET is setup to work with XML files
- To work with EMF models
  - In the transform extension specify model loader as org.eclipse.jet.emf.modelLoader
- To work with a specific metamodel
  - Add its Java package to the imports list of the template
    - <%@jet imports="org.eclipse.emf.java.*" %>
  - Can use the schema for an EMF model to control inputs

```
<transform
    modelLoader="org.eclipse.jet.emf"
    modelSchema="emf.java.xsd"
    startTemplate="templates/main.jet"
    templateLoaderClass="com.eett.exercises.m2t.jet.co
  <description></description>
  <tagLibraries>
```

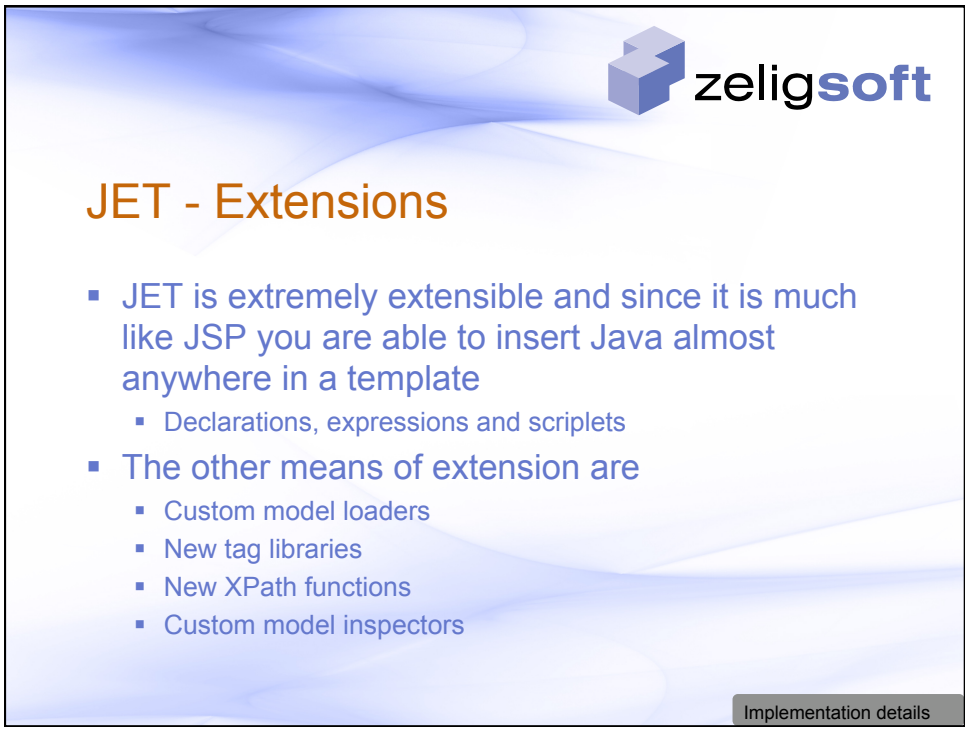
Implementation details

zeligsoft

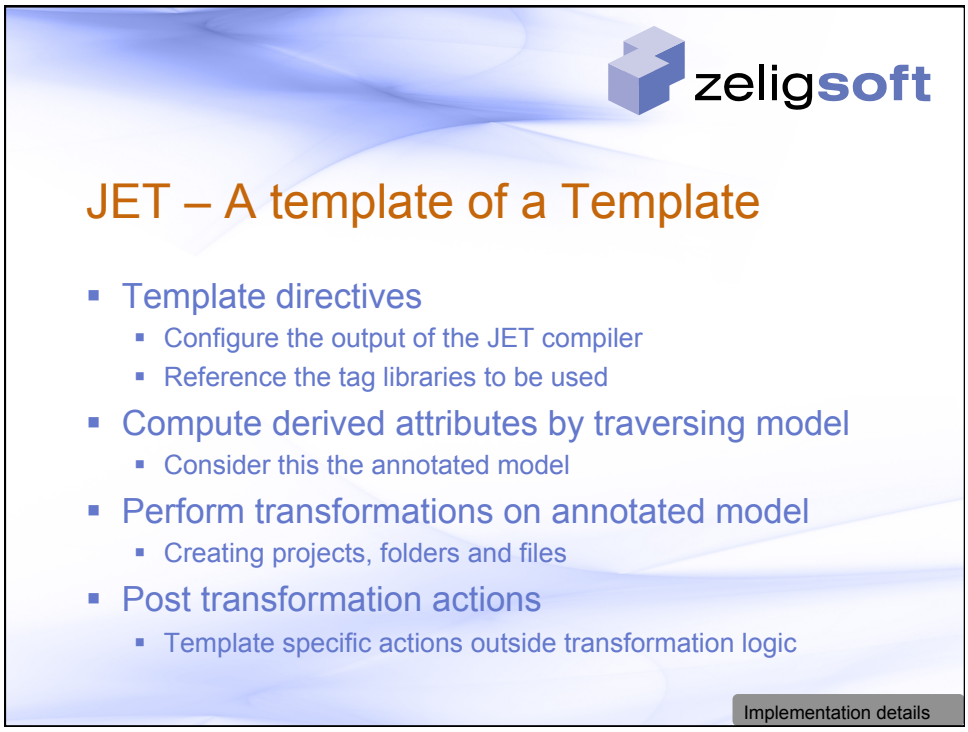
JET – Loading Models

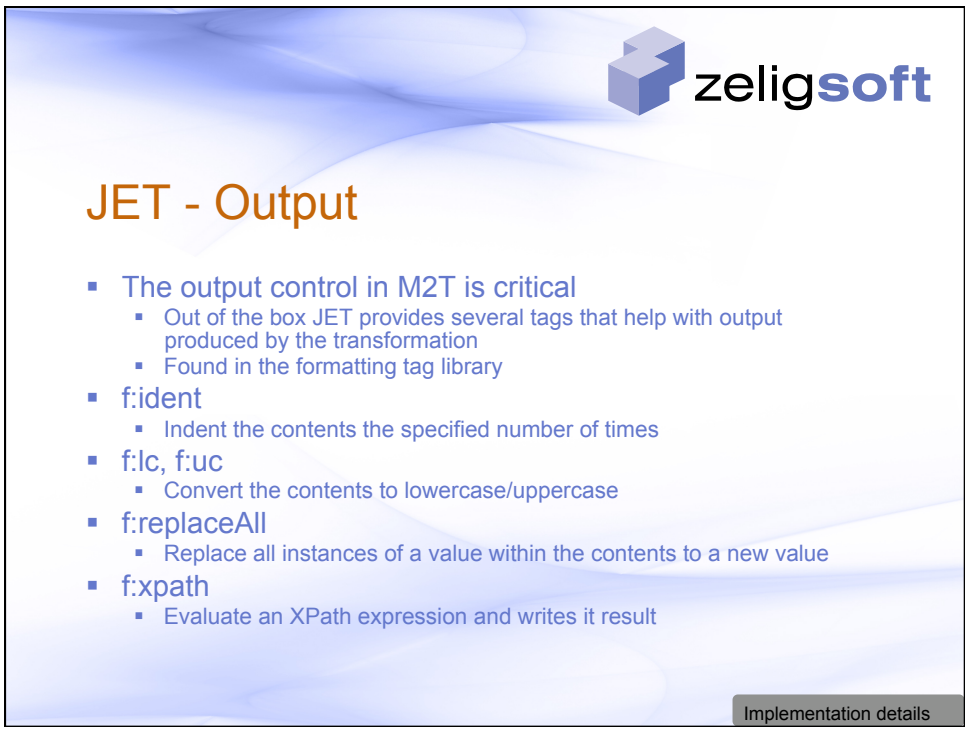
- Tags in JET used to load content
  - Load the content into a variable so it can be referenced
- c:load
  - Loads the referenced model into a specified variable
  - Can be referenced through the variable after that
- c:loadContent
  - Load the content of the tag into the specified variable as XML

Implementation details


zeligsoft
JET - Extensions
JET is extremely extensible and since it is much like JSP you are able to insert Java almost anywhere in a template
Declarations, expressions and scriplets
The other means of extension are
Custom model loaders
New tag libraries
New XPath functions
Custom model inspectors
Implementation details

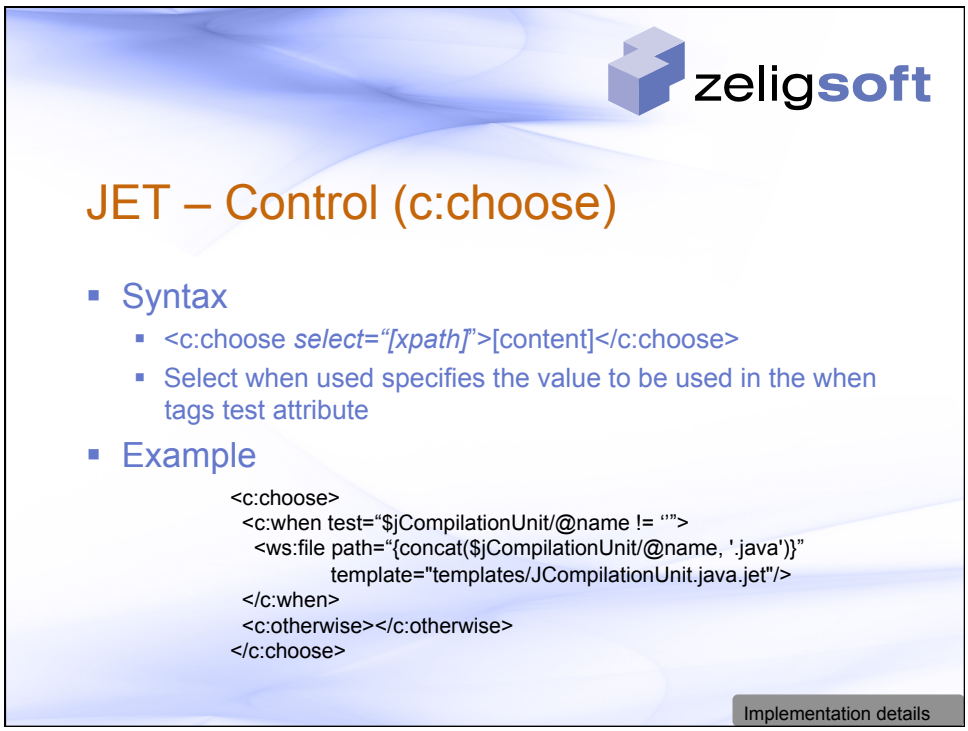

zeligsoft
JET - Templates
A template in JET is defined in a file
There is a 1 to 1 mapping between file and template
Directives configure the template
@jet
affects the code created by the JET compiler
package
class
imports
startTag
endTag
@taglib
imports a tag library for use in the template and assigns it a namespace prefix
Implementation details

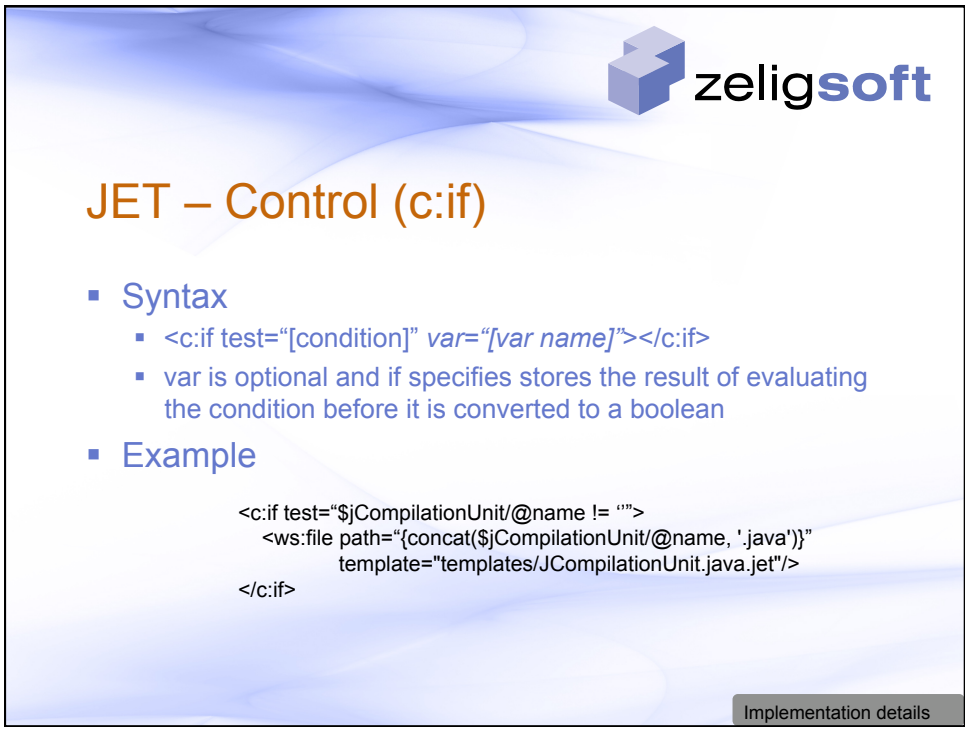


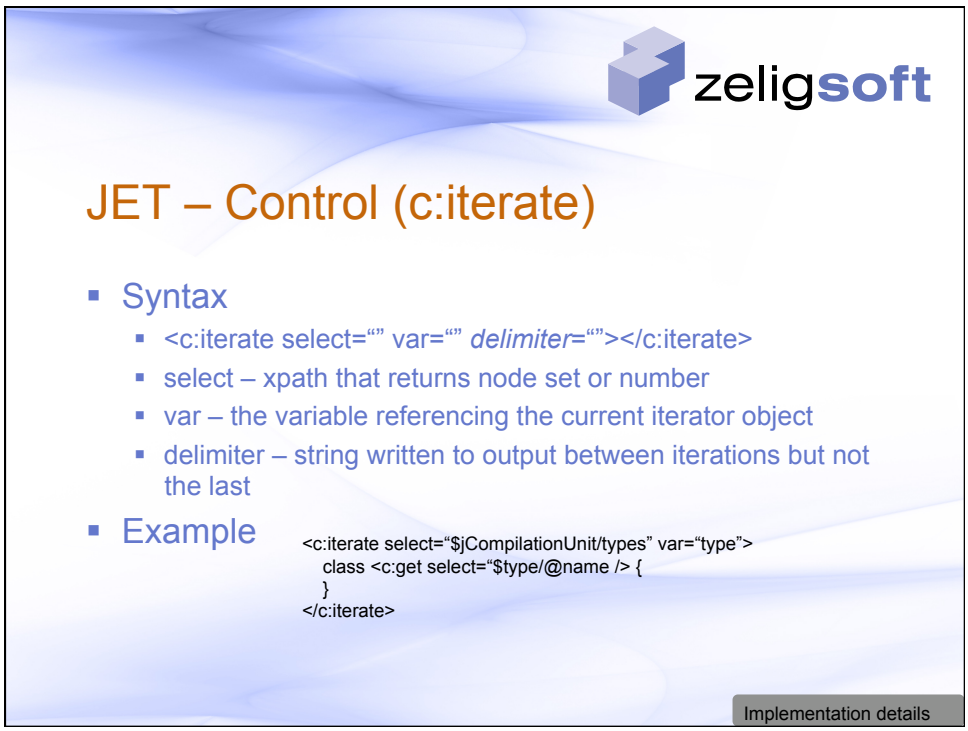


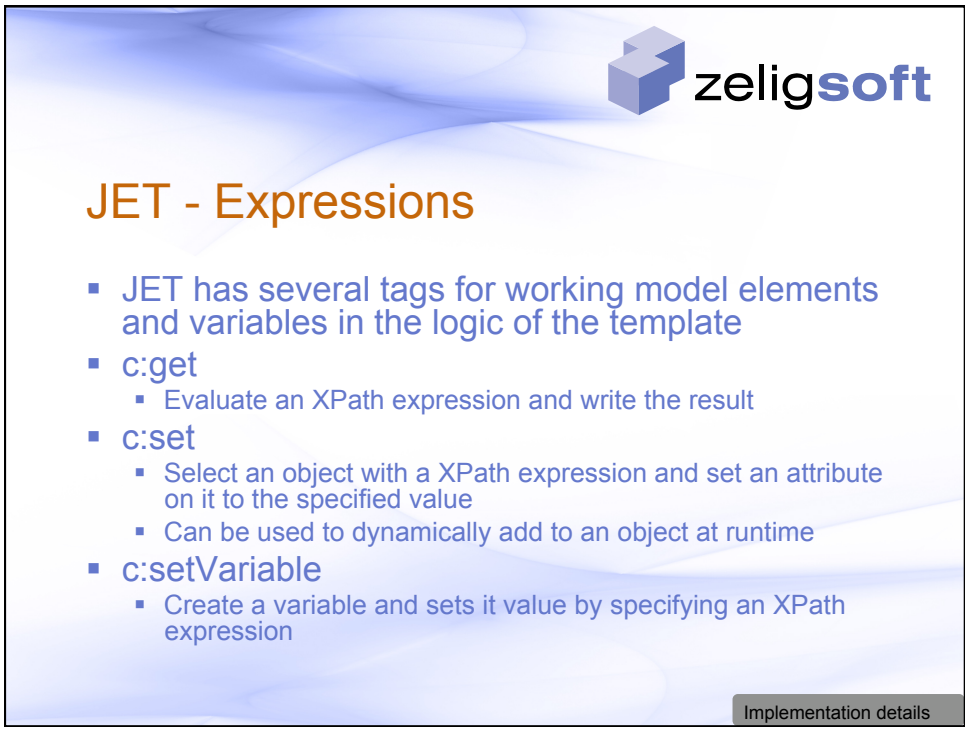
Toby McClean, Zeligsoft, 2008

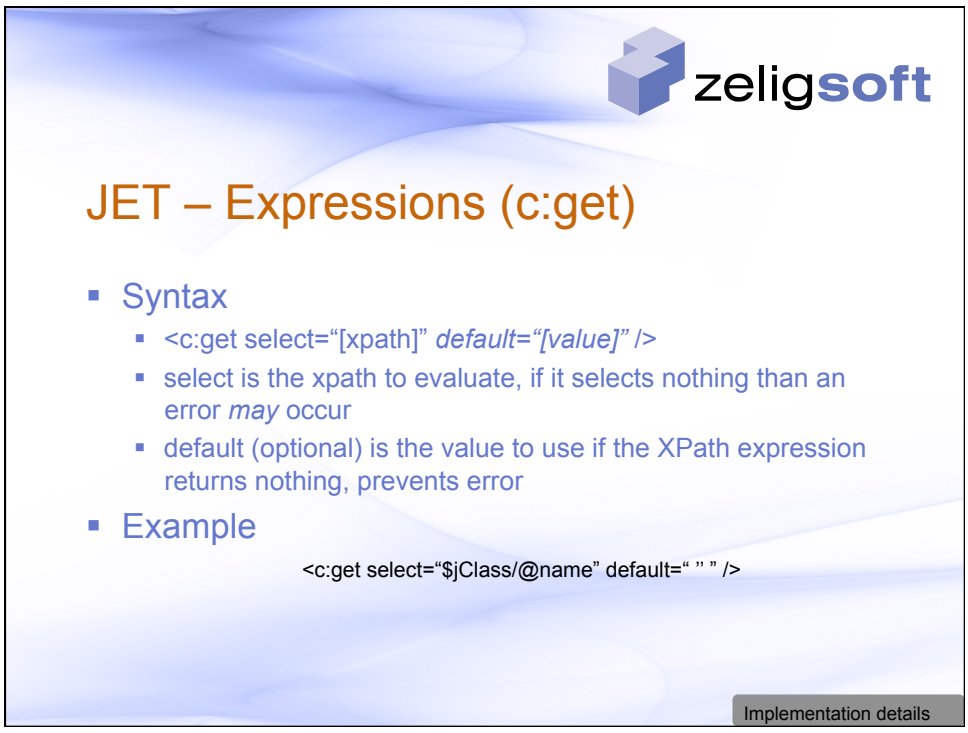

zeligsoft
JET - Control
The control tag library provides capabilities for putting control logic into your template
c:choose
A group of mutually exclusive choices
c:if
Only process the contents if a test condition is satisfied
c:iterate
Process the contents for each element specified by an XPath expression
If the XPath expression evaluates to a number that it iterates that number of times
Implementation details

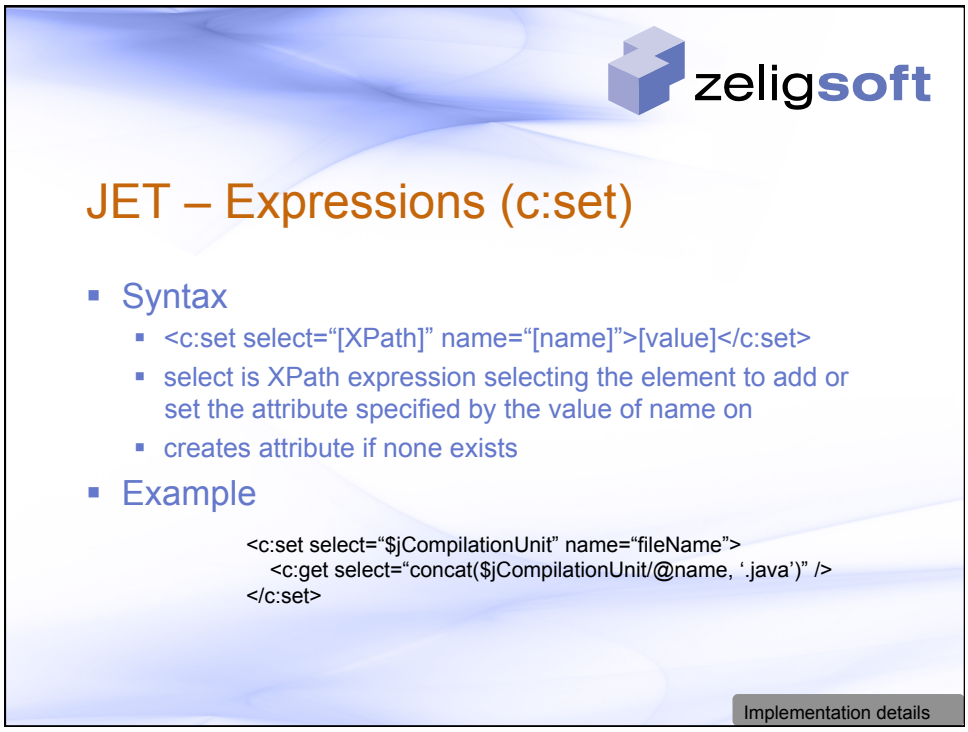

zeligsoft
JET – Control (c:choose)
Syntax
<c:choose select="[xpath]">[content]</c:choose>
Select when used specifies the value to be used in the when tags test attribute
Example
<c:choose>
<c:when test="$jCompilationUnit/@name != ''">
<ws:file path="{concat($jCompilationUnit/@name, '.java')}"
template="templates/JCompilationUnit.java.jet"/>
</c:when>
<c:otherwise></c:otherwise>
</c:choose>
Implementation details

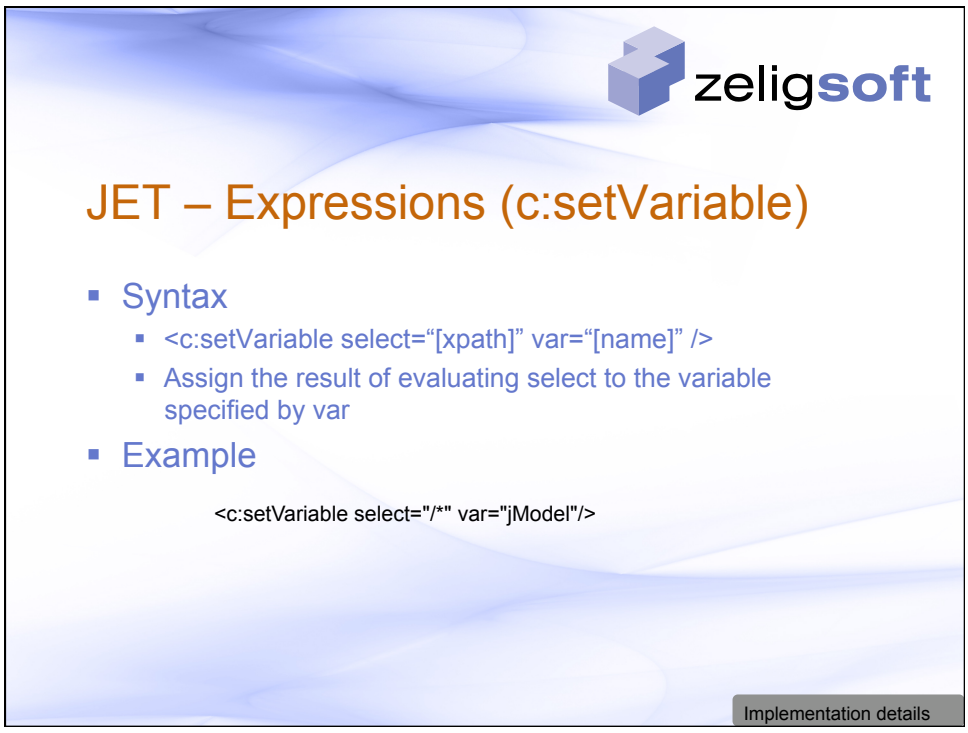

zeligsoft
JET – Control (c:if)
Syntax
<c:if test="[condition]" var="[var name]"></c:if>
var is optional and if specifies stores the result of evaluating the condition before it is converted to a boolean
Example
<c:if test="$jCompilationUnit/@name != ''">
<ws:file path="{concat($jCompilationUnit/@name, '.java')}"
template="templates/JCompilationUnit.java.jet"/>
</c:if>
Implementation details

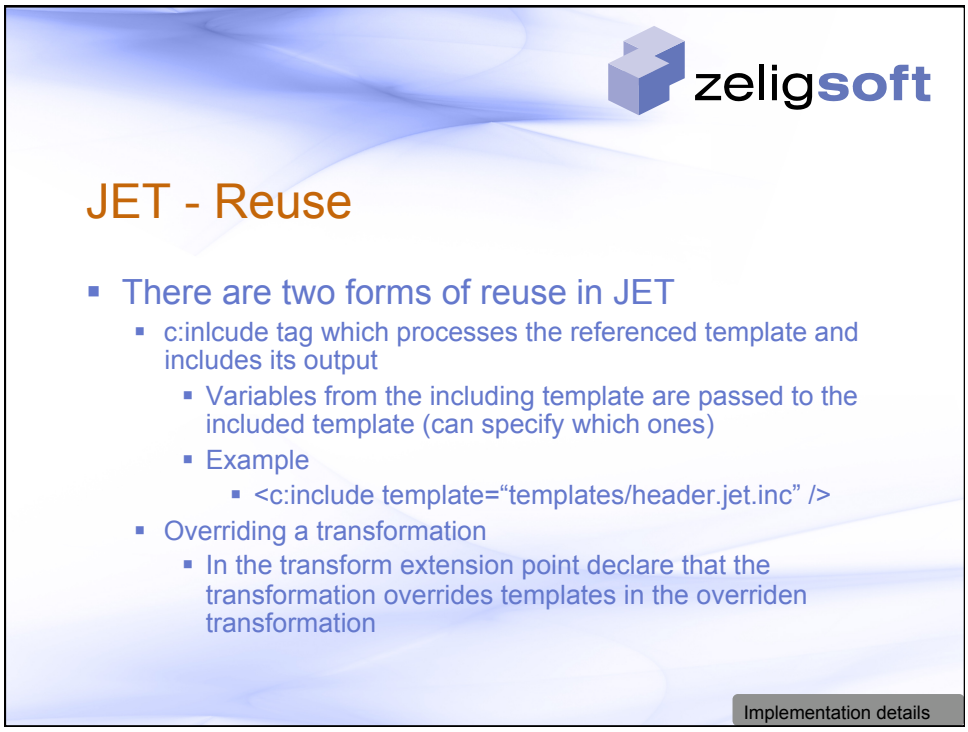

zeligsoft
JET – Control (c:iterate)
Syntax
<c:iterate select="" var="" delimiter=""></c:iterate>
select – xpath that returns node set or number
var – the variable referencing the current iterator object
delimiter – string written to output between iterations but not the last
Example
<c:iterate select="$jCompilationUnit/types" var="type">
class <c:get select="$type/@name /> {
}
</c:iterate>
Implementation details

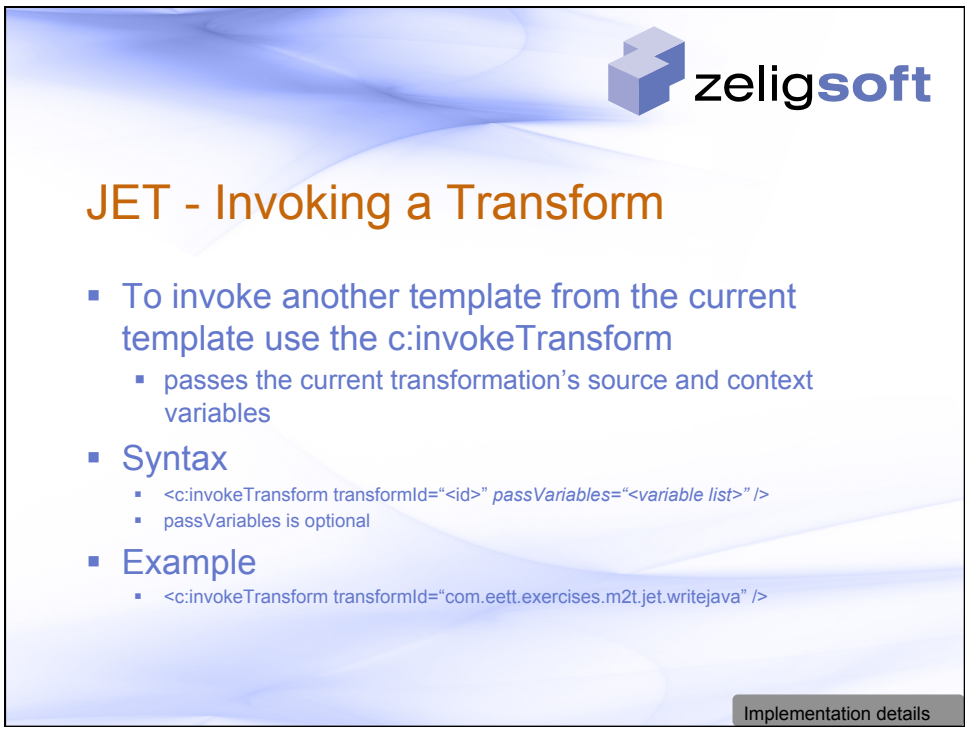

zeligsoft
JET - Expressions
JET has several tags for working model elements and variables in the logic of the template
c:get
Evaluate an XPath expression and write the result
c:set
Select an object with a XPath expression and set an attribute on it to the specified value
Can be used to dynamically add to an object at runtime
c:setVariable
Create a variable and sets it value by specifying an XPath expression
Implementation details

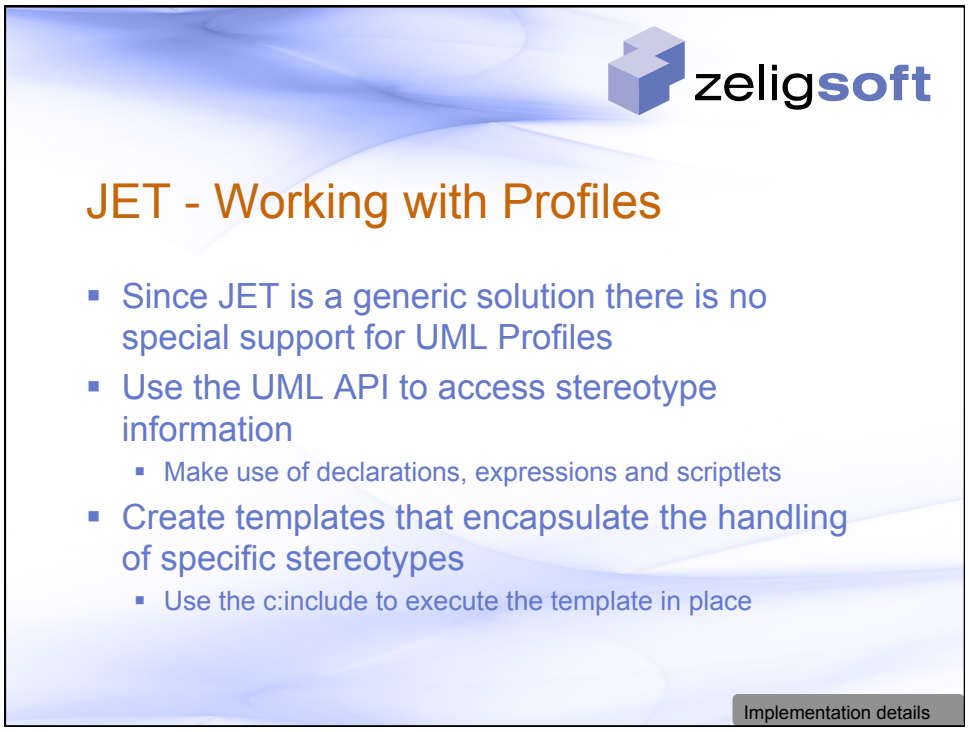

zeligsoft
JET – Expressions (c:get)
Syntax
<c:get select="[xpath]" default="[value]" />
select is the xpath to evaluate, if it selects nothing than an error may occur
default (optional) is the value to use if the XPath expression returns nothing, prevents error
Example
<c:get select="$jClass/@name" default=" " " />
Implementation details

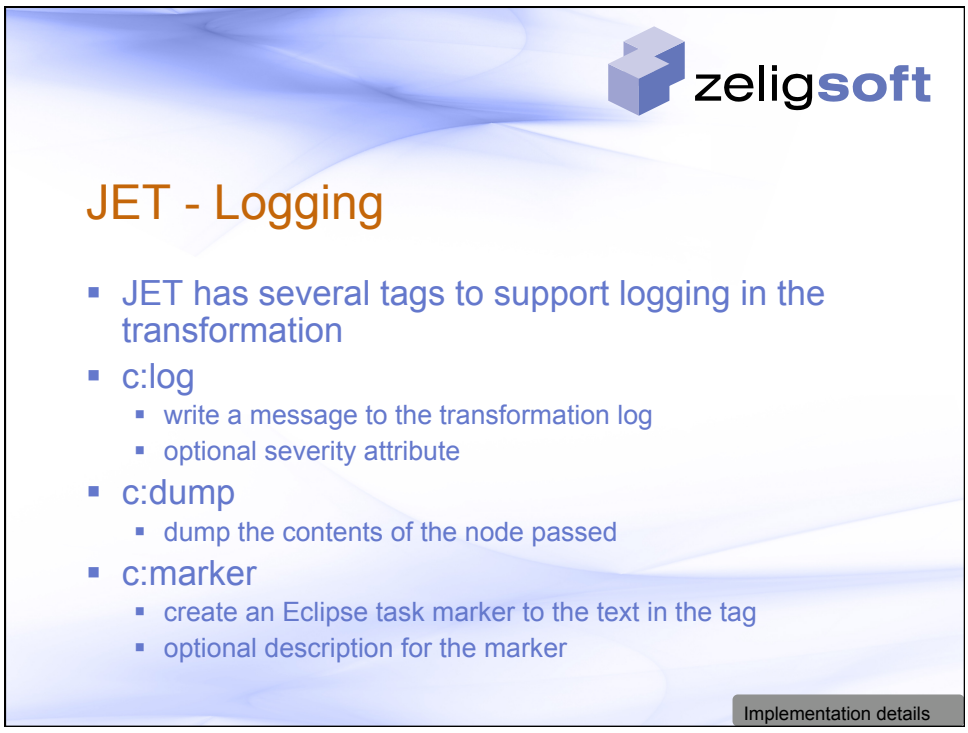

zeligsoft
JET – Expressions (c:set)
Syntax
<c:set select="[XPath]" name="[name]">[value]</c:set>
select is XPath expression selecting the element to add or set the attribute specified by the value of name on
creates attribute if none exists
Example
<c:set select="$jCompilationUnit" name="fileName">
<c:get select="concat($jCompilationUnit/@name, '.java')" />
</c:set>
Implementation details


zeligsoft
JET – Expressions (c:setVariable)
Syntax
<c:setVariable select="[xpath]" var="[name]" />
Assign the result of evaluating select to the variable specified by var
Example
<c:setVariable select="/*" var="jModel"/>
Implementation details

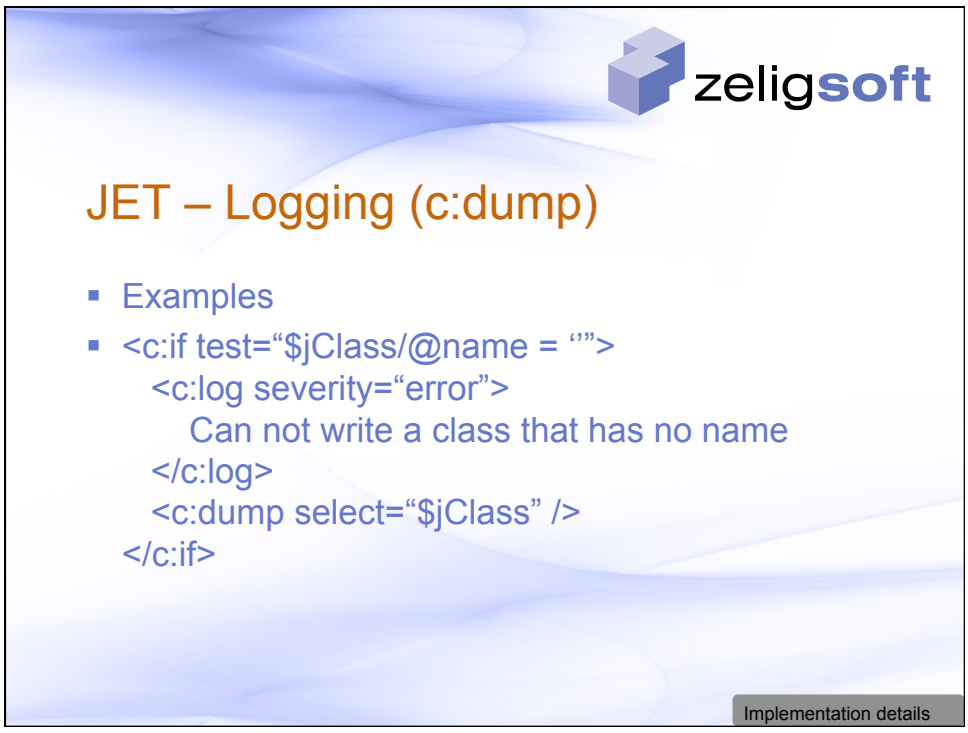

zeligsoft
JET - Reuse
There are two forms of reuse in JET
c:inlcude tag which processes the referenced template and includes its output
Variables from the including template are passed to the included template (can specify which ones)
Example
<c:include template="templates/header.jet.inc" />
Overriding a transformation
In the transform extension point declare that the transformation overrides templates in the overriden transformation
Implementation details

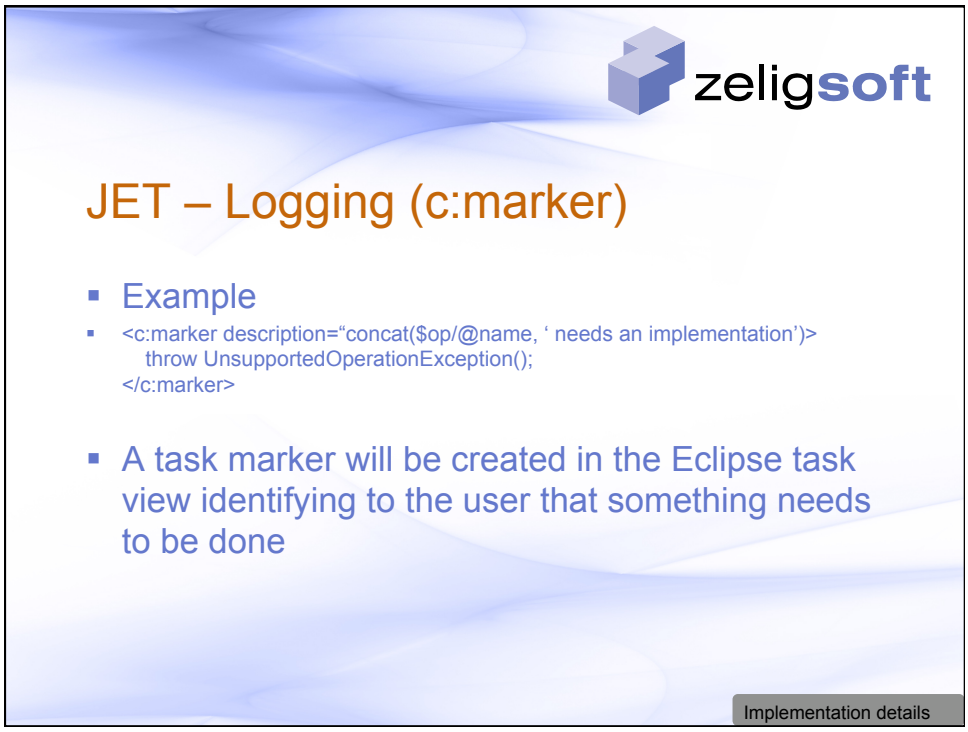

zeligsoft
JET - Invoking a Transform
To invoke another template from the current template use the c:invokeTransform
passes the current transformation's source and context variables
Syntax
<c:invokeTransform transformId="<id>" passVariables="<variable list>" />
passVariables is optional
Example
<c:invokeTransform transformId="com.eett.exercises.m2t.jet.writejava" />
Implementation details


zeligsoft
JET - Working with Profiles
Since JET is a generic solution there is no special support for UML Profiles
Use the UML API to access stereotype information
Make use of declarations, expressions and scriptlets
Create templates that encapsulate the handling of specific stereotypes
Use the c:include to execute the template in place
Implementation details

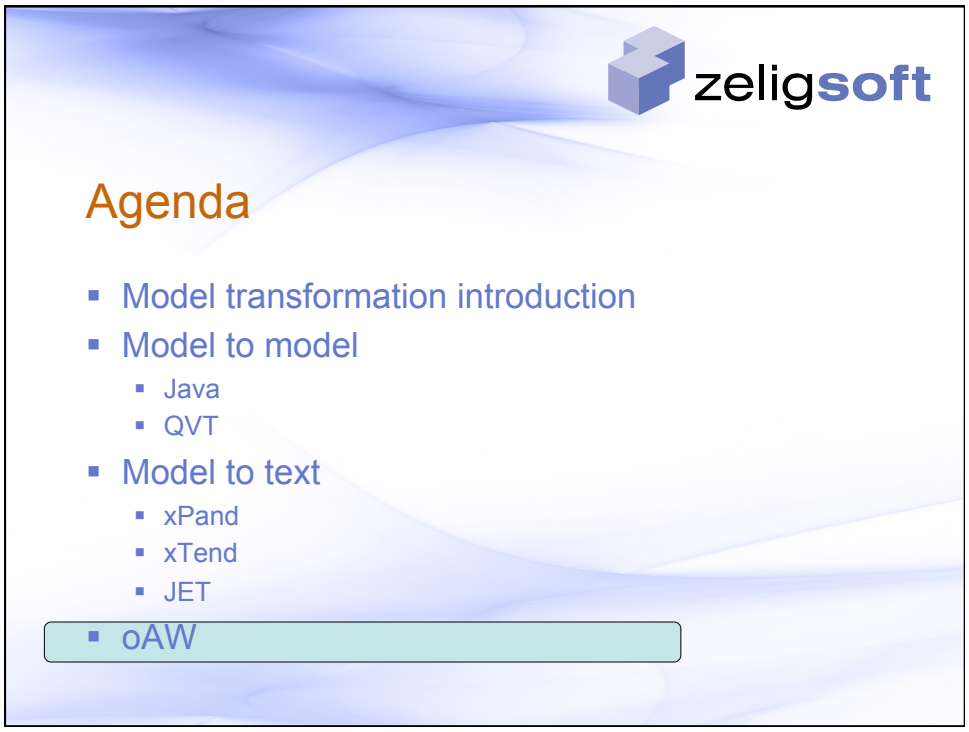

zeligsoft
JET - Logging
JET has several tags to support logging in the transformation
c:log
write a message to the transformation log
optional severity attribute
c:dump
dump the contents of the node passed
c:marker
create an Eclipse task marker to the text in the tag
optional description for the marker
Implementation details

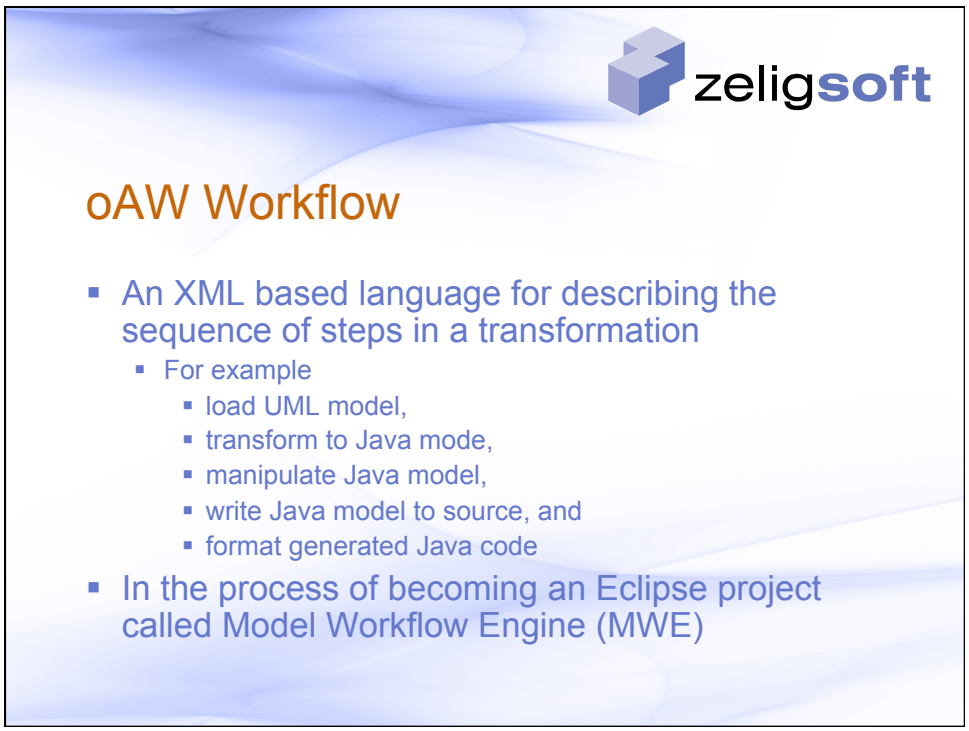

zeligsoft
JET – Logging (c:log)
Examples
<c:if test="$jClass/@name = ''">
<c:log severity="error">
Can not write a class that has no name
</c:log>
</c:if>
<c:log>
Writing <c:get select="$jClass/@name" />
</c:log>
Implementation details

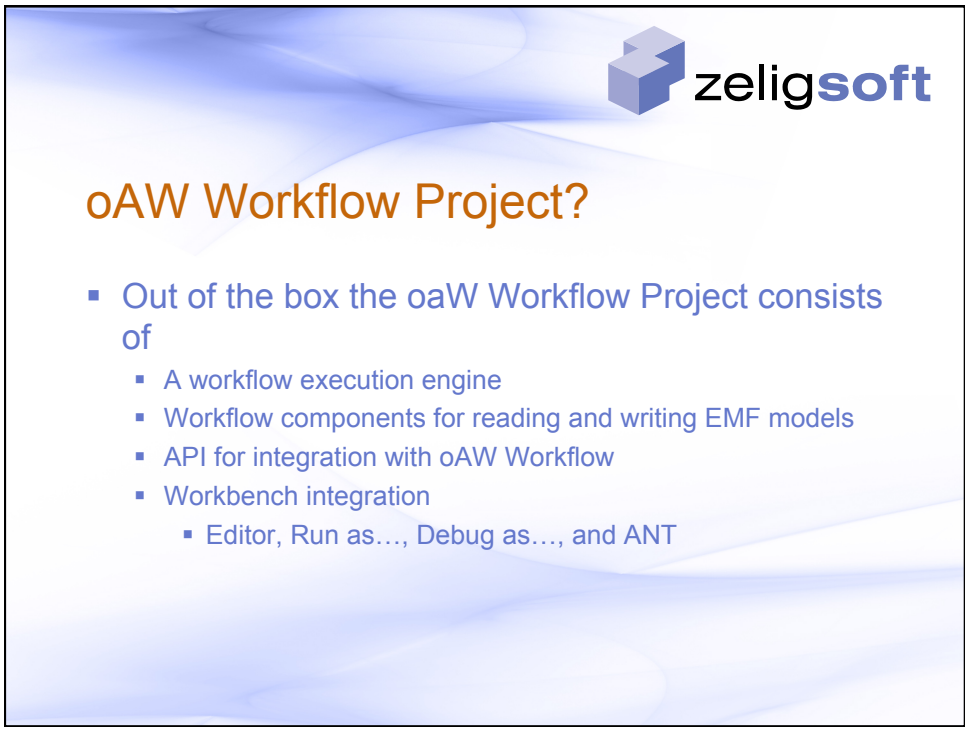

zeligsoft
JET – Logging (c:dump)
Examples
<c:if test="$jClass/@name = ''">
<c:log severity="error">
Can not write a class that has no name
</c:log>
<c:dump select="$jClass" />
</c:if>
Implementation details


zeligsoft
JET – Logging (c:marker)
Example
<c:marker description="concat($op/@name, ' needs an implementation')>
throw UnsupportedOperationException();
</c:marker>
A task marker will be created in the Eclipse task view identifying to the user that something needs to be done
Implementation details


zeligsoft
Summary
We have explored JET
What is JET
How I develop a model to text transformation with JET
How is my transformation integrated with Eclipse
The features and syntax of JET
You should now be able to
Do the JET exercises
Navigate around the JET documentation and other resources


zeligsoft
Agenda
Model transformation introduction
Model to model
Java
QVT
Model to text
xPand
xTend
JET
oAW


zeligsoft
oAW Workflow
An XML based language for describing the sequence of steps in a transformation
For example
load UML model,
transform to Java mode,
manipulate Java model,
write Java model to source, and
format generated Java code
In the process of becoming an Eclipse project called Model Workflow Engine (MWE)

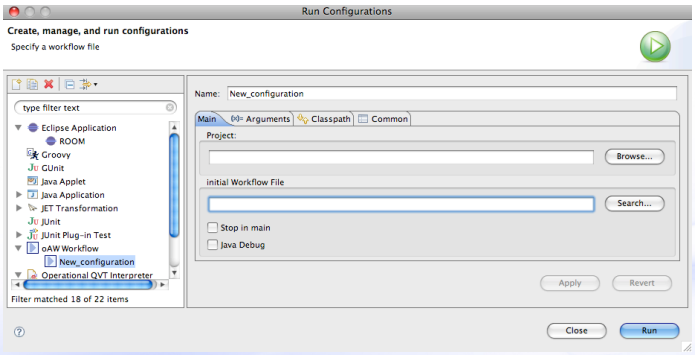

zeligsoft
oAW Workflow Project?
Out of the box the oaW Workflow Project consists of
A workflow execution engine
Workflow components for reading and writing EMF models
API for integration with oAW Workflow
Workbench integration
Editor, Run as…, Debug as…, and ANT


zeligsoft
More oAW Workflow Project?
API allows for custom workflow components
e.g. QVT transformation execution
Configuration properties
Properties passed into the workflow
<property name='targetDir' value='src-gen/'/>
Slots or variables
Simple syntax, variables are referred to by name
No declaration
<component id="generator" class="oaw.xtend.XtendComponent">
…
<outputSlot value="javaModel" />
</component>

zeligsoft

Executing an oAW Workflow

- The oAW Workflow Engine is integrated with Run as…



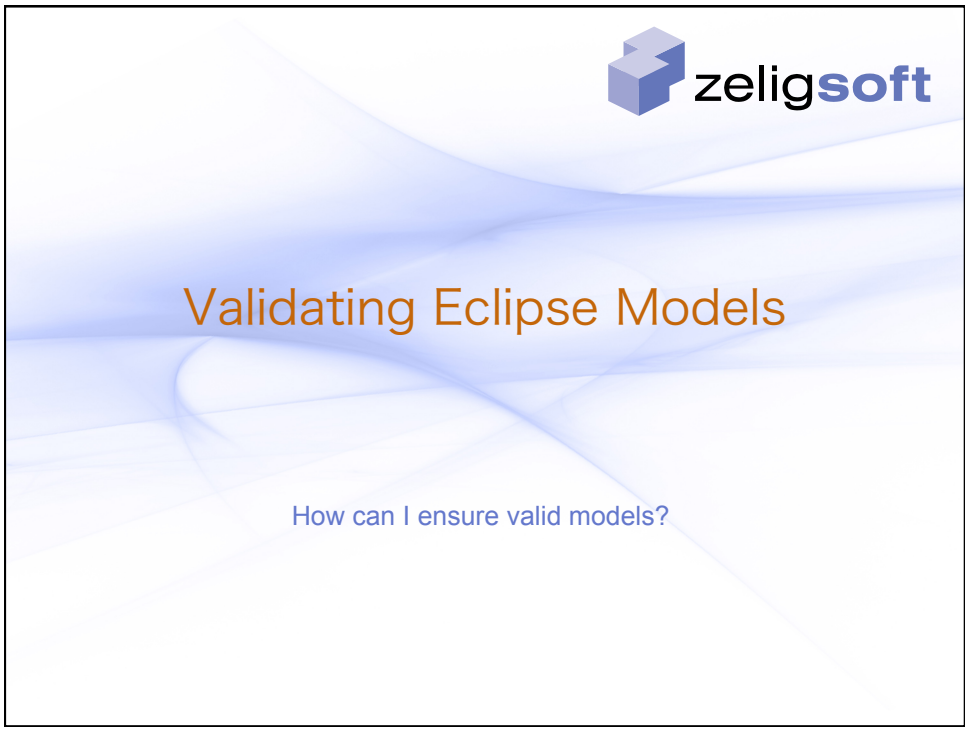
zeligsoft

Discussion

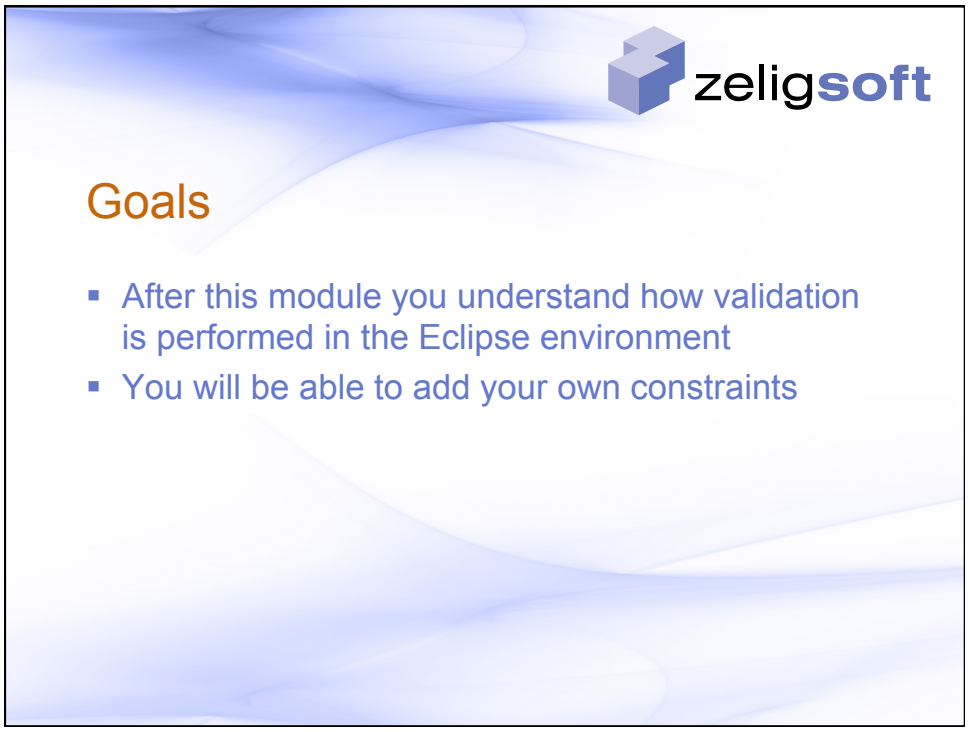
- We can combine the previous module and this module
  - Extend the model with additional information
  - Extend generation to use this information
  - May require RTS customization
- Examples?
  - Deadlines in messages
  - Port patterns

zeligsoft

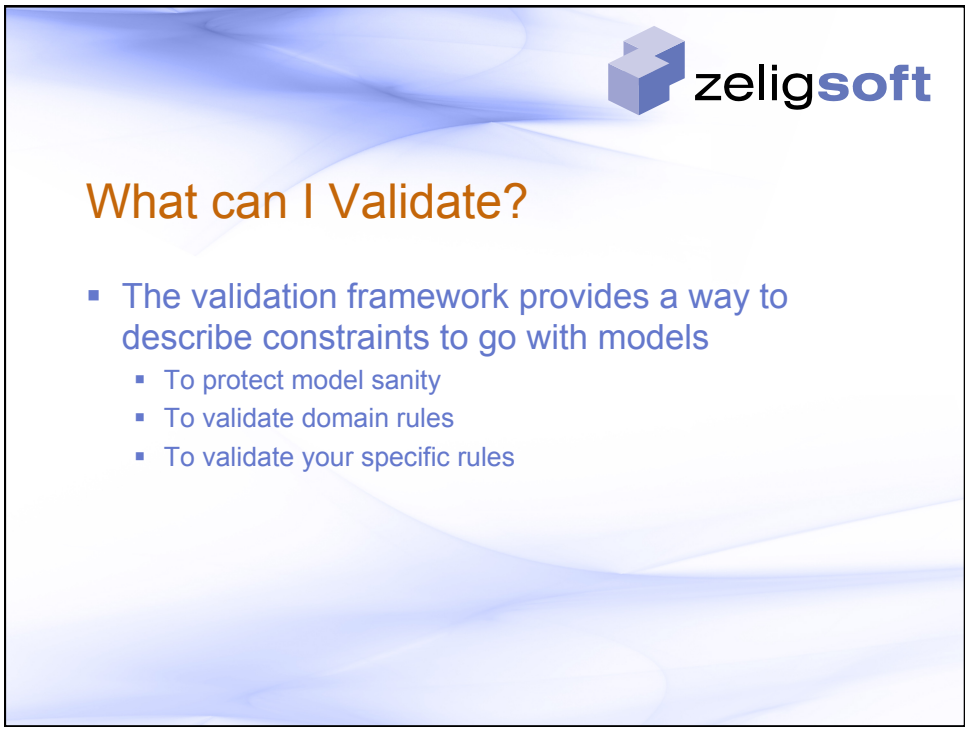
Questions



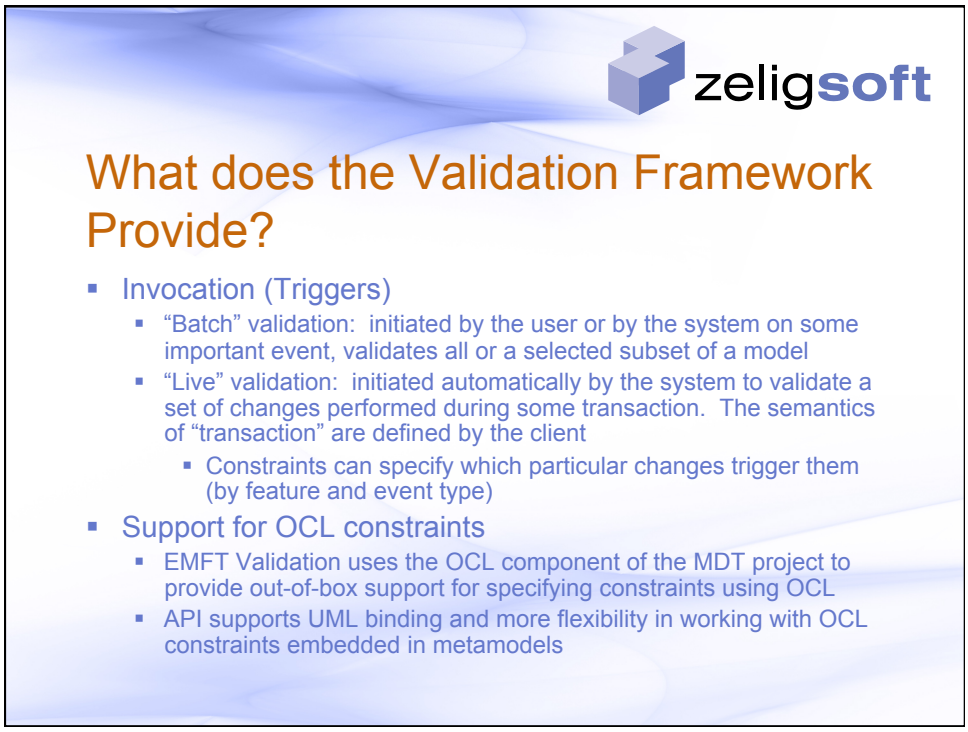
Toby McClean, Zeligsoft, 2008

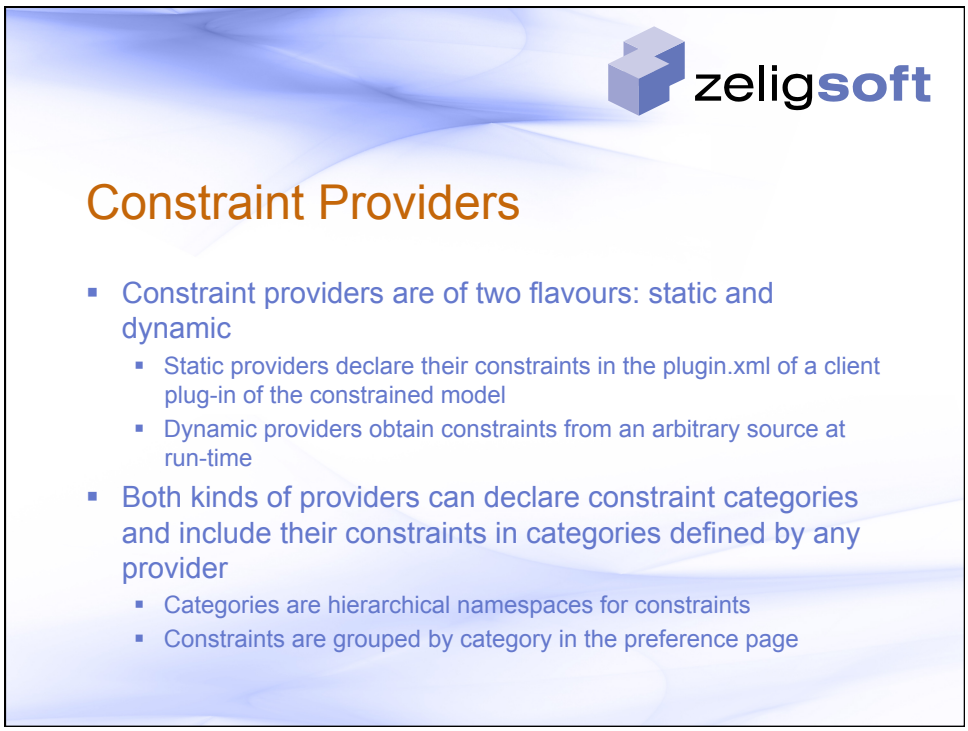

zeligsoft
Goals
After this module you understand how validation is performed in the Eclipse environment
You will be able to add your own constraints

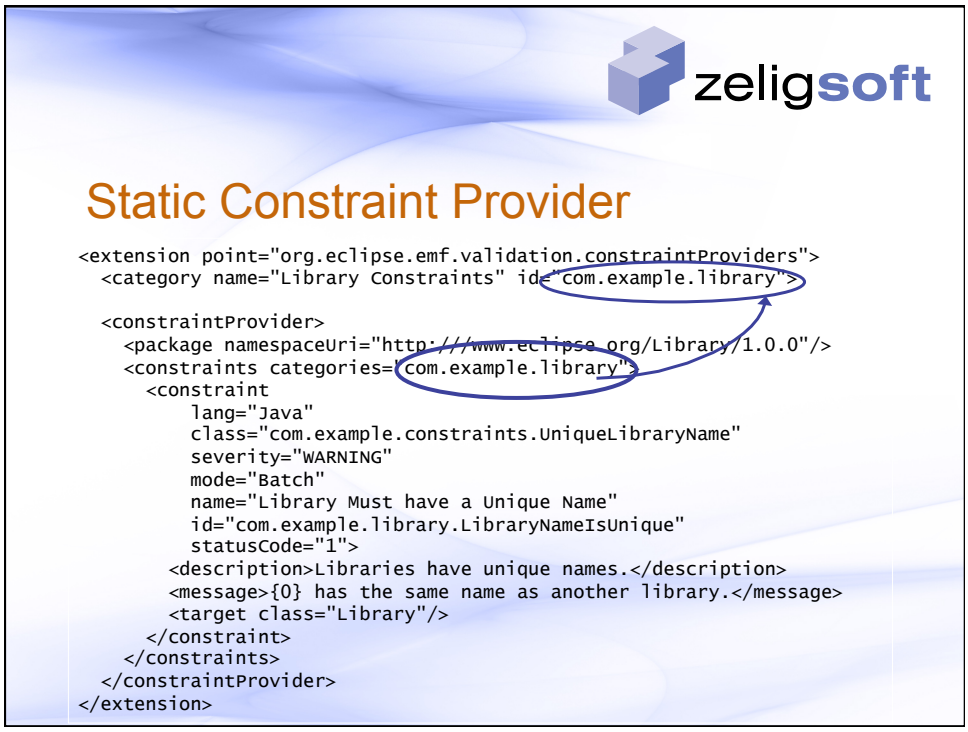

zeligsoft
Agenda
Overview
Static and dynamic constraint providers
Constraints
Validation service
Creating constraints

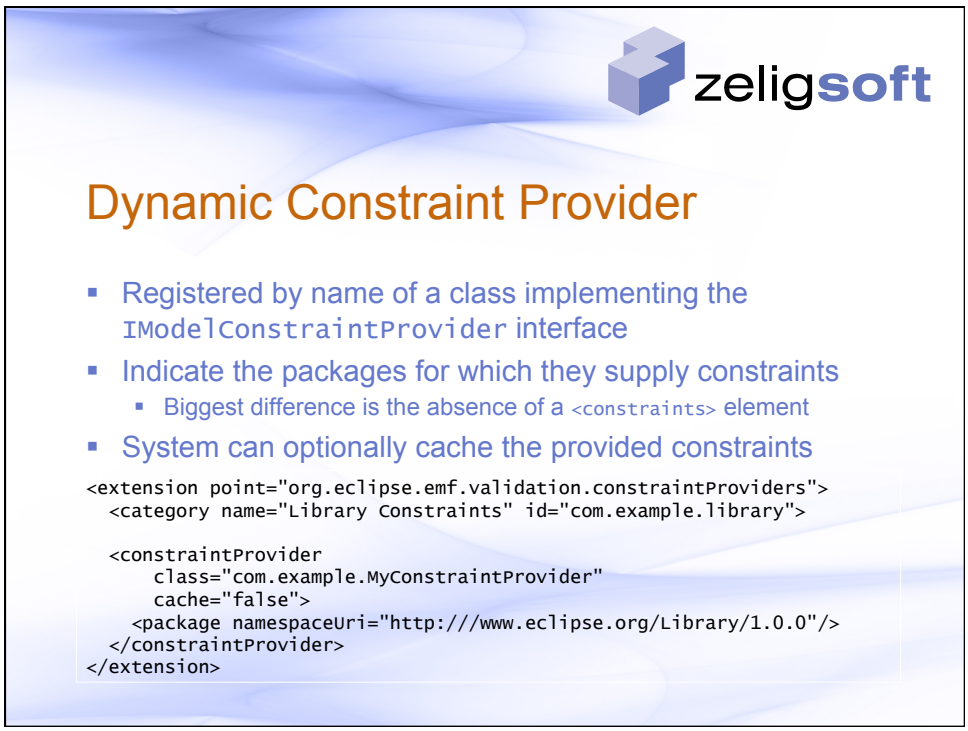

zeligsoft
What can I Validate?
The validation framework provides a way to describe constraints to go with models
To protect model sanity
To validate domain rules
To validate your specific rules


zeligsoft
Validation Framework Overview
Constraints are organized through categories and bindings
Constraint providers contribute constraints
Constraint parsers implement implementation languages
Traversal strategies walk the model
Validation listeners are notified when validation occurs
Notification generators for custom (esp. higher-order) notifications for processing in live validation


zeligsoft
What does the Validation Framework Provide?
Invocation (Triggers)
"Batch" validation: initiated by the user or by the system on some important event, validates all or a selected subset of a model
"Live" validation: initiated automatically by the system to validate a set of changes performed during some transaction. The semantics of "transaction" are defined by the client
Constraints can specify which particular changes trigger them (by feature and event type)
Support for OCL constraints
EMFT Validation uses the OCL component of the MDT project to provide out-of-box support for specifying constraints using OCL
API supports UML binding and more flexibility in working with OCL constraints embedded in metamodels


zeligsoft
Constraint Providers
Constraint providers are of two flavours: static and dynamic
Static providers declare their constraints in the plugin.xml of a client plug-in of the constrained model
Dynamic providers obtain constraints from an arbitrary source at run-time
Both kinds of providers can declare constraint categories and include their constraints in categories defined by any provider
Categories are hierarchical namespaces for constraints
Constraints are grouped by category in the preference page



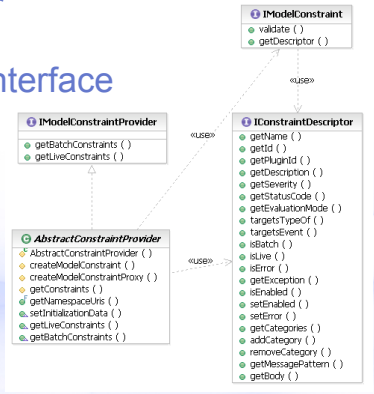


Toby McClean, Zeligsoft, 2008

zeligsoft

Dynamic Constraint Provider

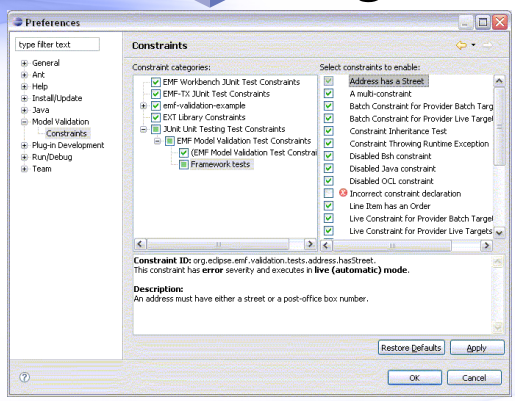
- A dynamic constraint provider implements the `IModelConstraintProvider` interface
- A descriptor supplies meta-data about a constraint
- The validation algorithm is supplied by the constraint object in the `validate()` method



zeligsoft

Constraints

- Descriptor provides:
  - ID and localizable name
  - Evaluation mode
  - Target EClasses
  - Triggering events (live)
  - Severity, error message
  - Categorization
  - Enabled state
- Much of this information appears in the preference page
- Users can choose which constraints to activate



zeligsoft

Evaluation Modes

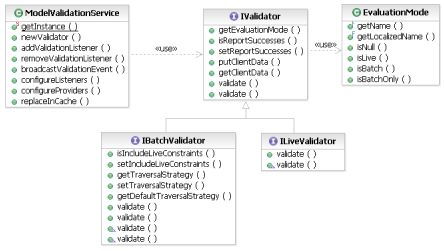
- Batch validation is usually explicitly requested by the user, via a menu action or toolbar button
- Live validation is performed automatically as changes are made to EMF resources
  - Live constraints indicate which specific changes trigger them

```
<constraint
    mode="Live"
    … >
  <description>Libraries have unique names.</description>
  <message>{0} has the same name as: {1}.</message>
  <target class="Library">
     <event name="Set">
        <feature name="name"/>
     </event>
  </target>
</constraint>
```

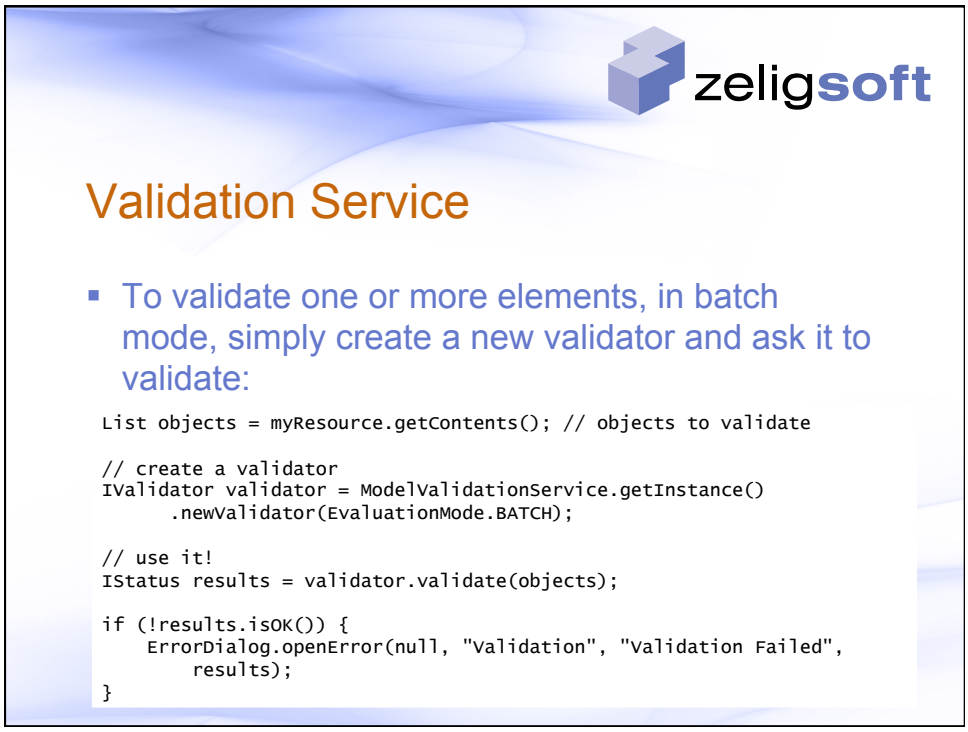
zeligsoft

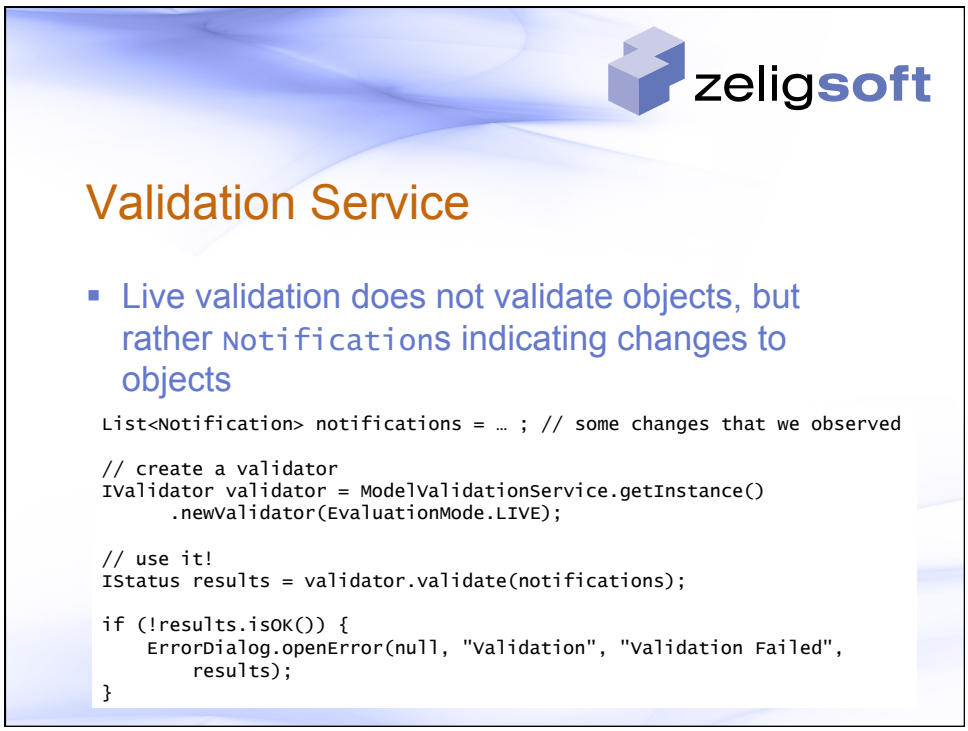
Validation Service

- Evaluation of constraints is performed via the Validation Service
- The service provides validators corresponding to the available evaluation modes
- By default, batch validation includes live constraints, also, for completeness



Toby McClean, Zeligsoft, 2008


zeligsoft
Validation Service
To validate one or more elements, in batch mode, simply create a new validator and ask it to validate:
List objects = myResource.getContents(); // objects to validate
// create a validator
IValidator validator = ModelValidationService.getInstance()
.newValidator(EvaluationMode.BATCH);
// use it!
IStatus results = validator.validate(objects);
if (!results.isOK()) {
ErrorDialog.openError(null, "Validation", "Validation Failed",
results);
}


zeligsoft
Validation Service
Live validation does not validate objects, but rather Notifications indicating changes to objects
List<Notification> notifications = … ; // some changes that we observed
// create a validator
IValidator validator = ModelValidationService.getInstance()
.newValidator(EvaluationMode.LIVE);
// use it!
IStatus results = validator.validate(notifications);
if (!results.isOK()) {
ErrorDialog.openError(null, "Validation", "Validation Failed",
results);
}

zeligsoft

Creating Constraints

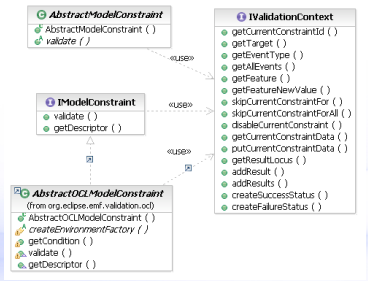
- Specifying a constraint doesn't necessarily require any code
- Requires the OCL component of the MDT project

```
<constraint
    lang="OCL"
    mode="Batch"
    … >
  <description>Libraries have unique names.</description>
  <message>{0} has the same name as: {1}.</message>
  <target class="Library"/>

  <![CDATA[
  Library.allInstances()->forAll(l |
       l <> self implies l.name <> self.name)
  ]]>

</constraint>
```

zeligsoft

Creating Constraints

- Sometimes the easiest or best way to formulate a constraint is in Java
- Java constraints extend the `AbstractModelConstraint` class
- The validation context provides the `validate()` method with information about the validation operation



zeligsoft

Validation Context

- Provides the element being validated (the target)
- Indicates the current evaluation mode (live or batch)
  - In case of live invocation, provides the changed feature and the event type
- Allows constraints to cache arbitrary data for the duration of the current validation operation
- Provides convenient methods for reporting results
- Provides the ID of the constraint that is being invoked

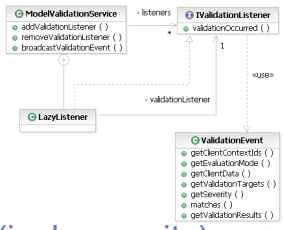
zeligsoft

Creating Constraints

```
public class LibraryNameIsUnique extends AbstractConstraint {
  public IStatus validate(IValidationContext ctx) {
    Library target = (Library) ctx.getTarget(); // object to validate

    // does this library have a unique name?
    Set<Library> libs = findLibrariesWithName(target.getName());
    if (libs.size() > 1) {
       // report this problem against all like-named libraries
       ctx.addResults(libs);

       // don't need to validate these other libraries
       libs.remove(target);
       ctx.skipCurrentConstraintFor(libs);

       return ctx.createFailureStatus(new Object[] {
          target, libs});
    }

    return ctx.createSuccessStatus();
  }
}
```

zeligsoft

Listening for Validation Events

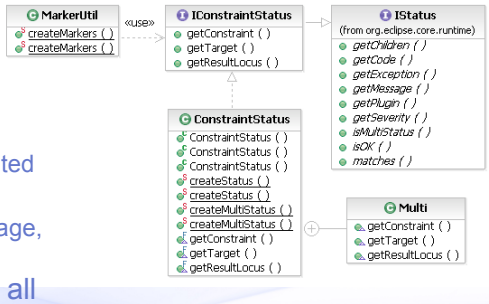


- Every validation operation executed by the Validation Service generates an event
- The event indicates the kind of validation performed, on what objects, and what were the results (incl. severity)
- Listeners registered on the extension point are lazily initialized when their selection criteria are met

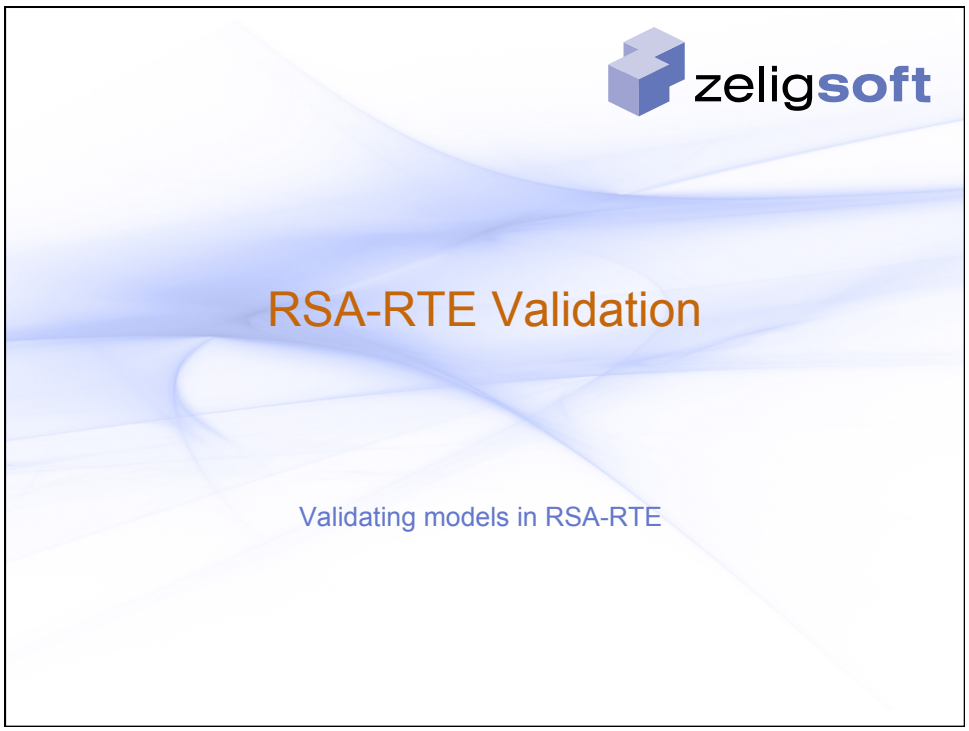
```
<extension point="org.eclipse.emf.validation.validationListeners">
    <listener class="com.example.validation.ProblemsReporter">
        <clientContext id="com.example.MyClientContext"/>
    </listener>
</extension>
```

zeligsoft

Reporting Problems



- Problems are reported as IConstraintStatus objects. A status knows:
  - The constraint that was violated
  - The objects that violated it
  - The severity and error message, as usual for an IStatus
- The marker utility encodes all of this information in a problem marker on the appropriate resource, to show in the Problems view



Toby McClean, Zeligsoft, 2008

zeligsoft

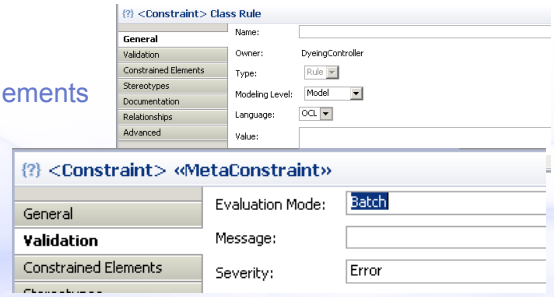
Overview

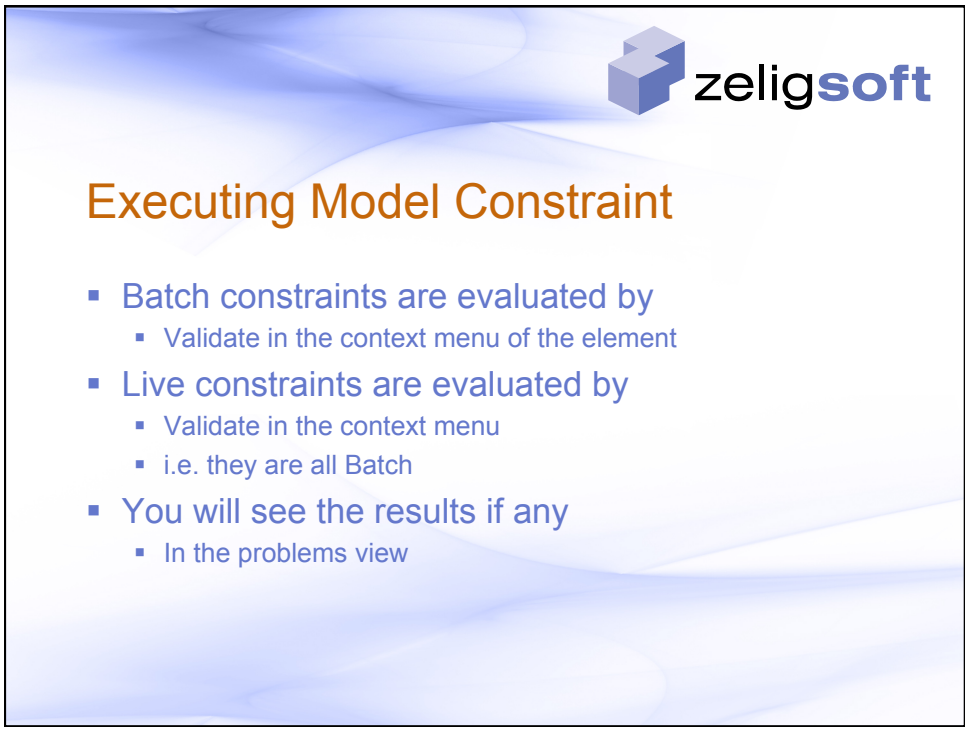
- Adding constraints to models in RSA-RTE
  - Authoring
  - Executing
- Adding constraints to profiles in RSA-RTE
  - Authoring
  - Executing

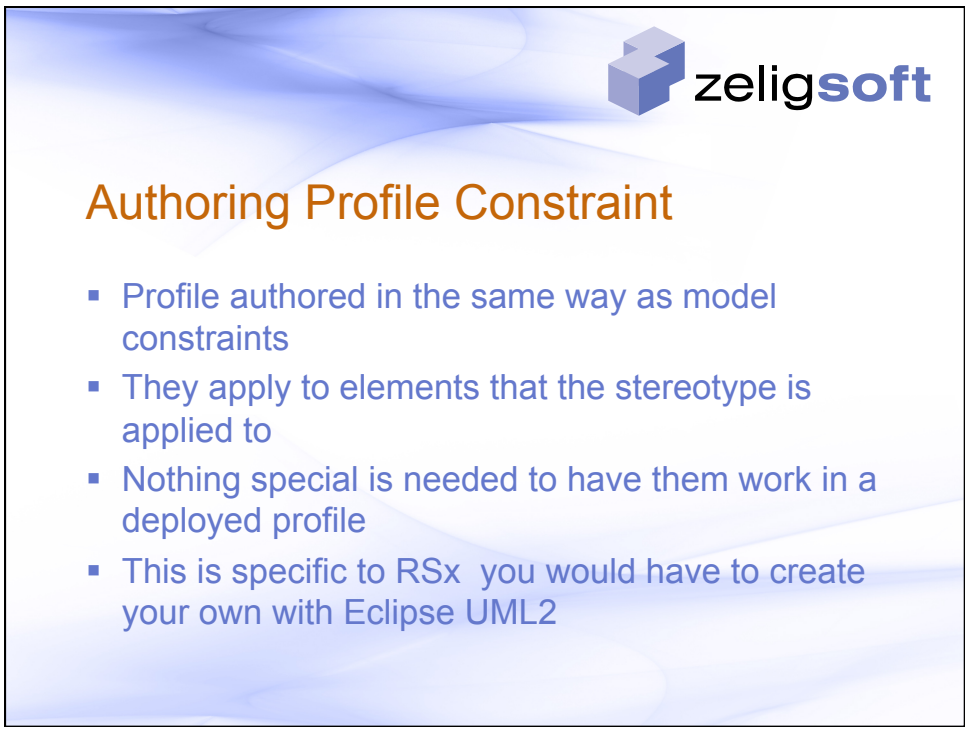
zeligsoft

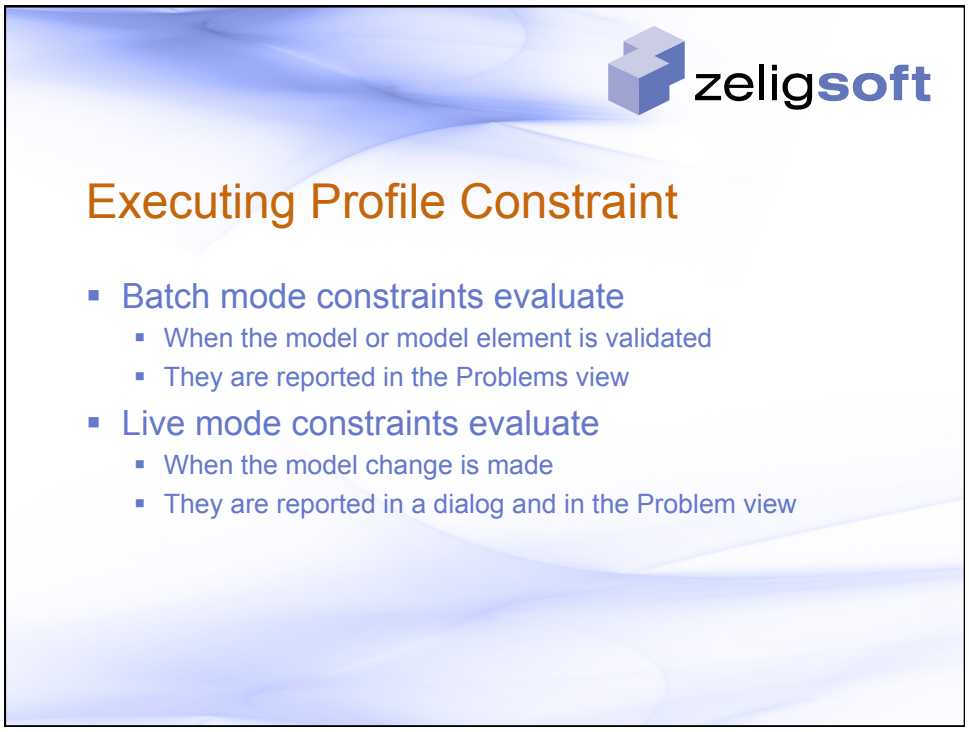
Authoring Model Constraint

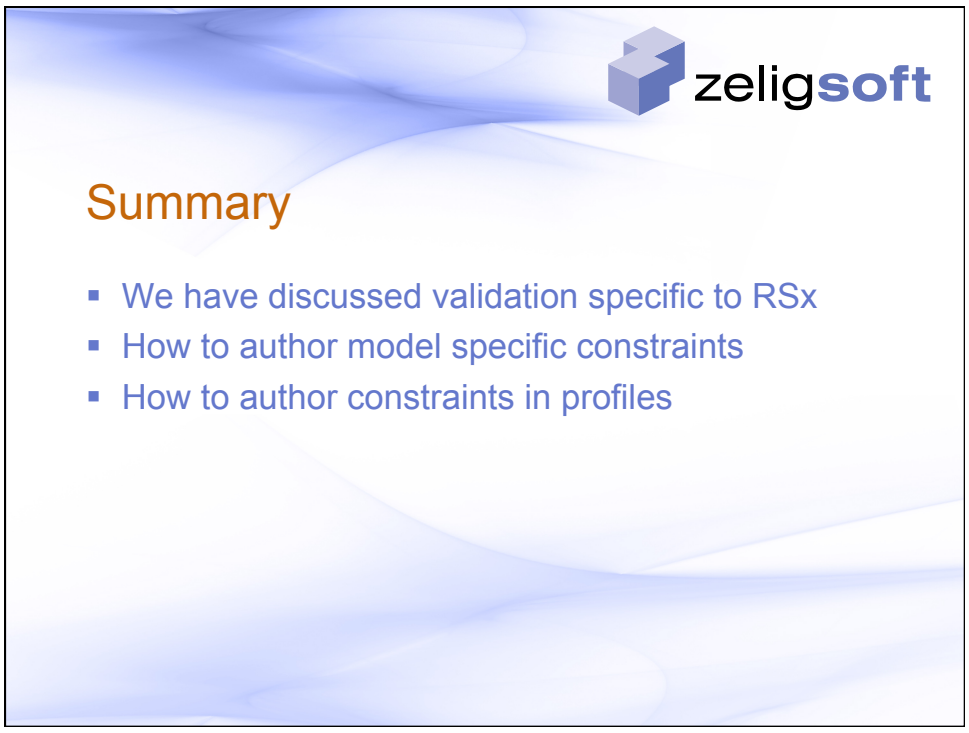
- Contraints can be added to model elements
  - Add UML → Constraint from the elements context menu
  - Defined by
    - Name
    - Constrained elements
    - Modeling level
    - Language
    - Mode
    - Message




zeligsoft
Executing Model Constraint
Batch constraints are evaluated by
Validate in the context menu of the element
Live constraints are evaluated by
Validate in the context menu
i.e. they are all Batch
You will see the results if any
In the problems view


zeligsoft
Authoring Profile Constraint
Profile authored in the same way as model constraints
They apply to elements that the stereotype is applied to
Nothing special is needed to have them work in a deployed profile
This is specific to RSx you would have to create your own with Eclipse UML2


zeligsoft
Executing Profile Constraint
Batch mode constraints evaluate
When the model or model element is validated
They are reported in the Problems view
Live mode constraints evaluate
When the model change is made
They are reported in a dialog and in the Problem view


zeligsoft
Summary
We have discussed validation specific to RSx
How to author model specific constraints
How to author constraints in profiles

zeligsoft

Questions