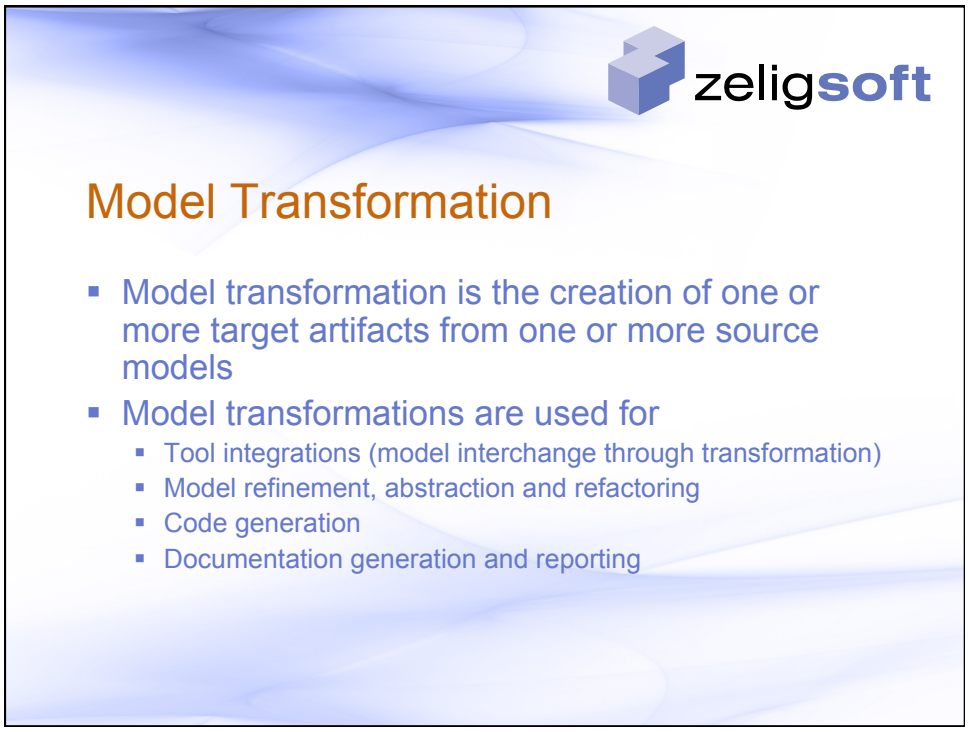
Toby McClean, Zeligsoft, 2008


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

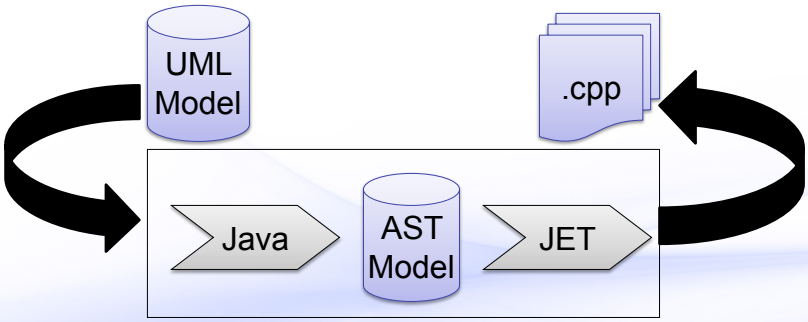
Model to Text Transformations

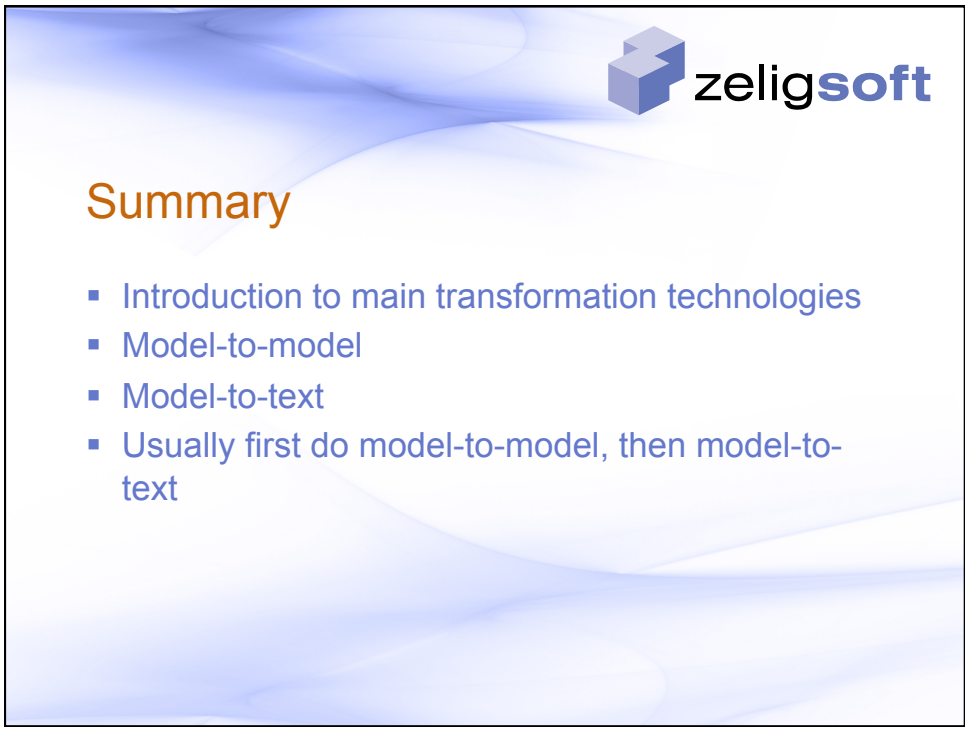
- A M2T transformation creates one or more text based artifacts from one or more source model
- Typically template based
  - Create a template from an example of the desired artifact
- M2T consideration
  - Often it is best to use M2T with a source model that is dedicated to the transformation
- Eclipse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

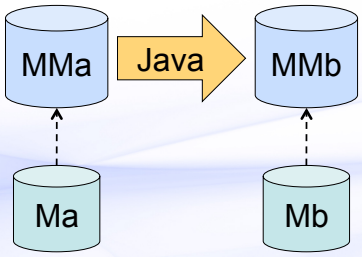
Overview

- 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

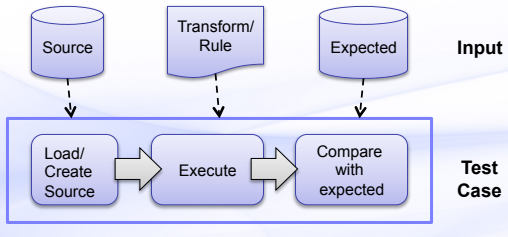
Developing a Transformation

- 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

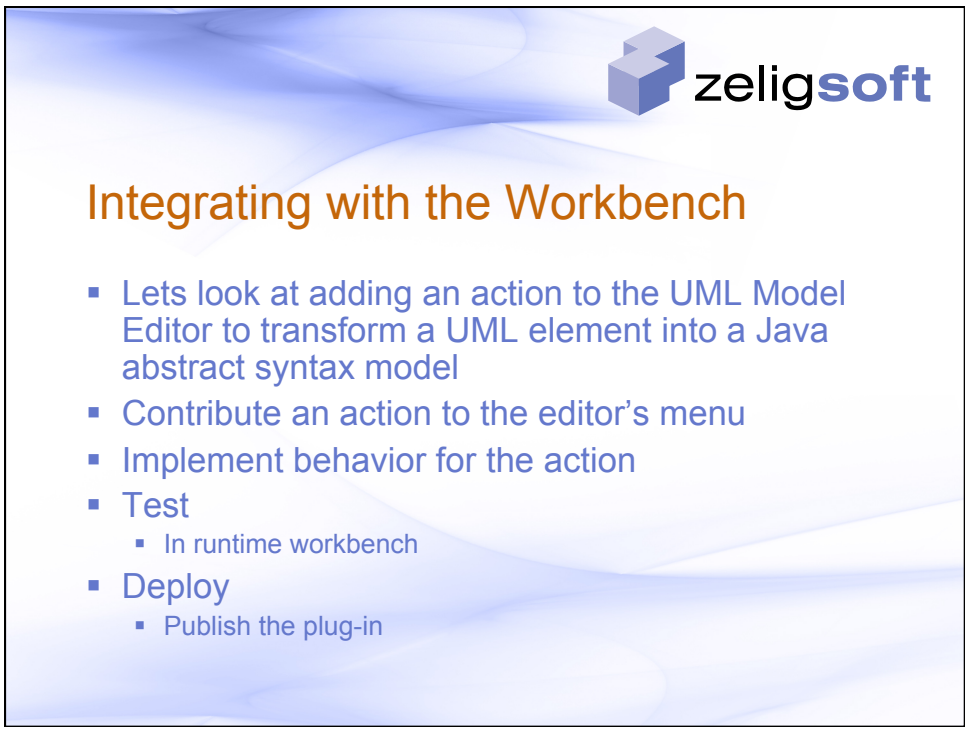
Testing Transformations

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



Toby McClean, Zeligsoft, 2008

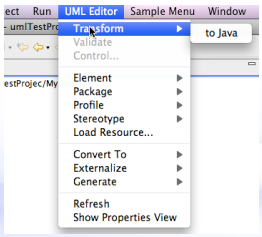

zeligsoft
Executing the Transformation
Use workbench Java execution capabilities
We can use the Run As → Java Application
We can debug using Debug As → Java Application
To pass transformation parameter values
Use command line
Build a user interface
Integrate with workbench
Provide an action to invoke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

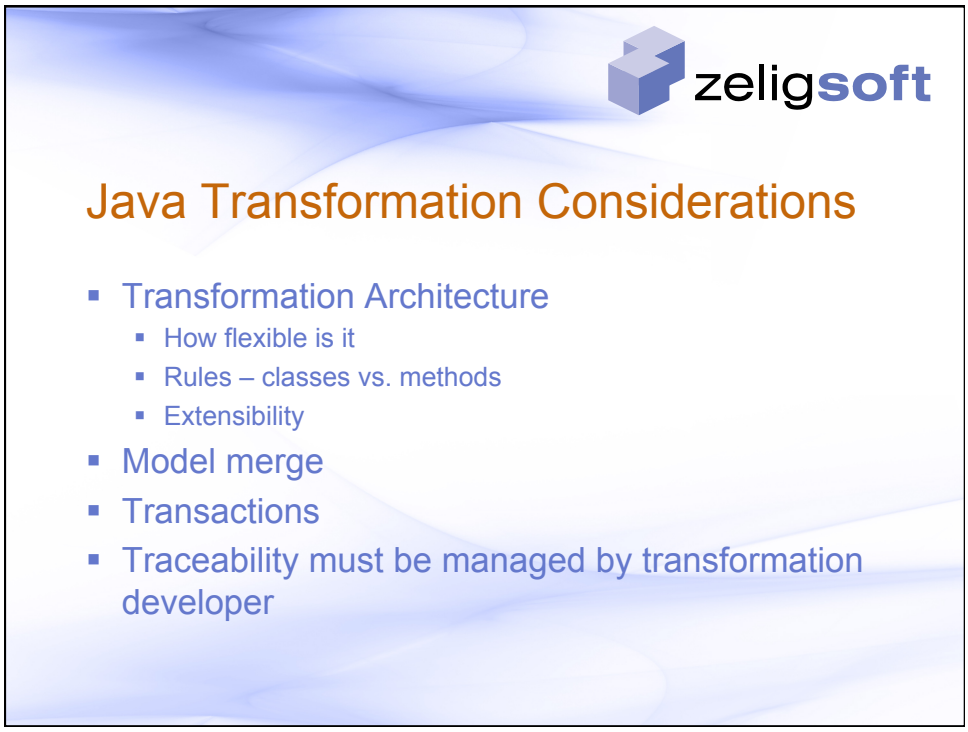
- UML Model Editor ID
  - org.eclipse.uml2.uml.editor.presentation.UMLEditorID
- Menu bar path
  - org.eclipse.uml2.umlMenuID
    - settings
    - actions
    - additions
    - additions-end

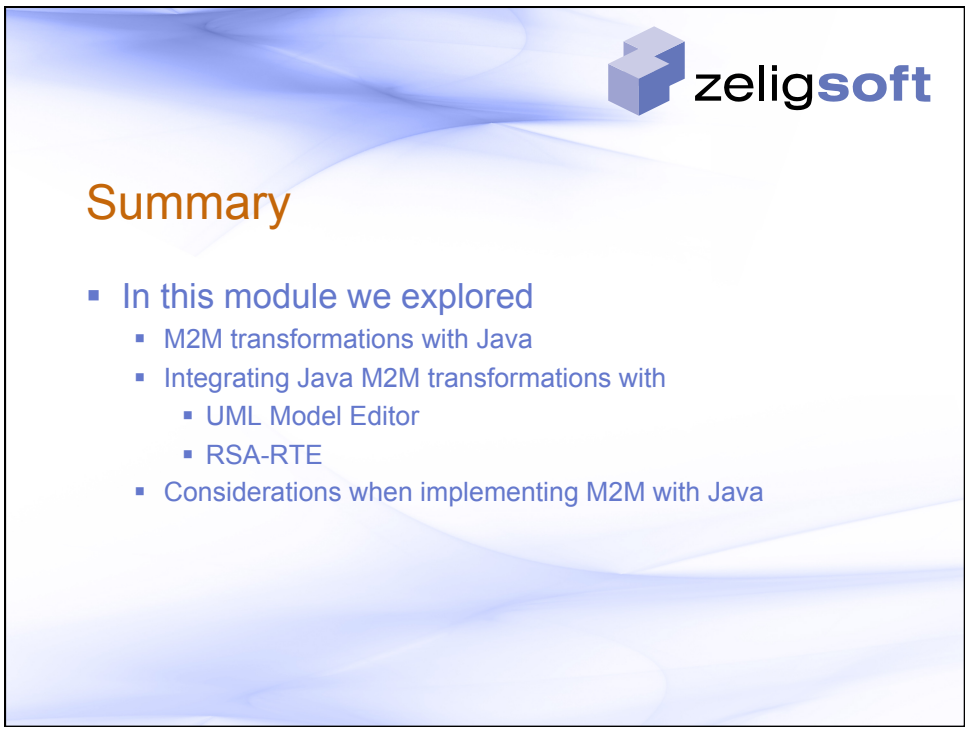


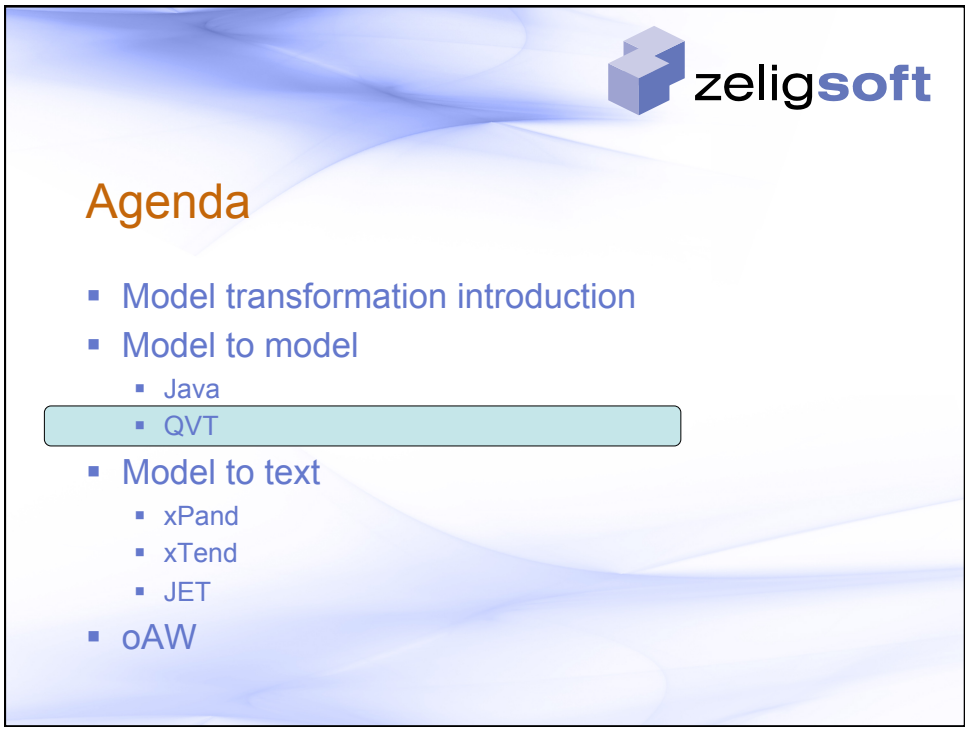
zeligsoft

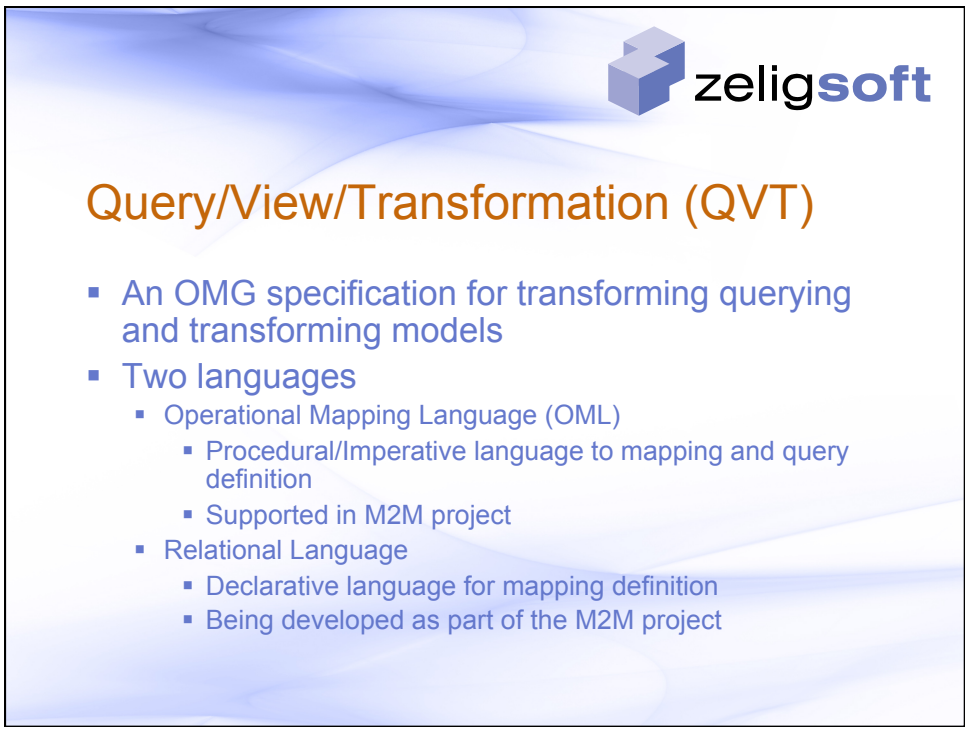
Integrating with RSA-RTE

- 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

zeligsoft

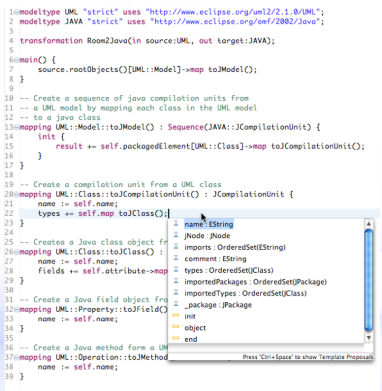
Operational QVT Project (QVTO)

- The QVT OML is a sub-project of the M2M project
  - http://www.eclipse.org/m2m/
- Its goal is to provide an implementation of the MOF 2.0 Query/View/Transformation Specification
  - http://www.omg.org/cgi-bin/doc?ptc/2007-07-07
- Works out of the box for transforming EMF based models

zeligsoft

Developing QVTO Transformations

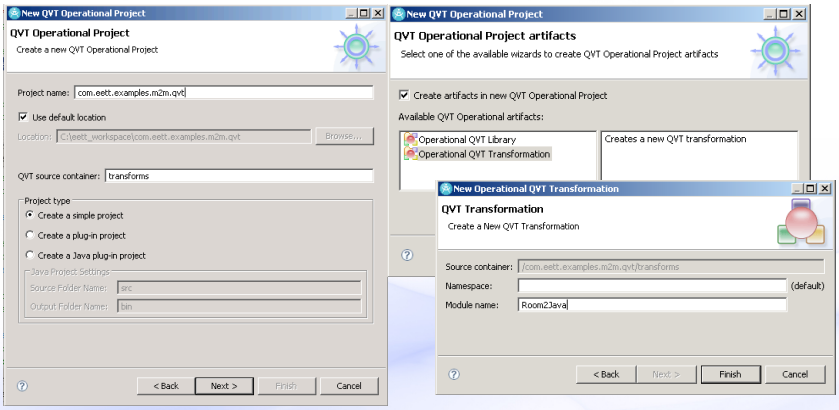
- Editor
  - Syntax highlighting
  - Code completion
- Model navigation
  - Superset of EssentialOcl (OCL adaptation for EMOF)
  - Support for OCL collection operators
- Focus on transformation logic
  - Execution infrastructure left to the QVTO runtime



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

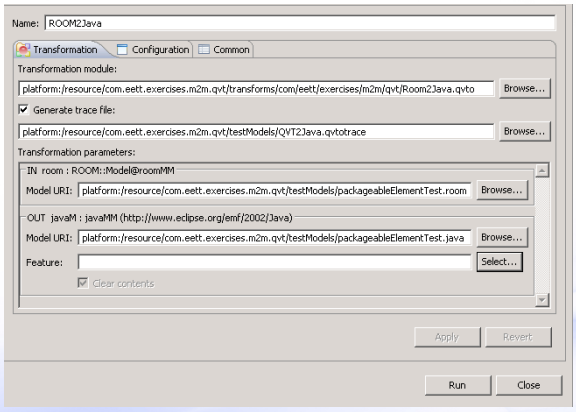
zeligsoft

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

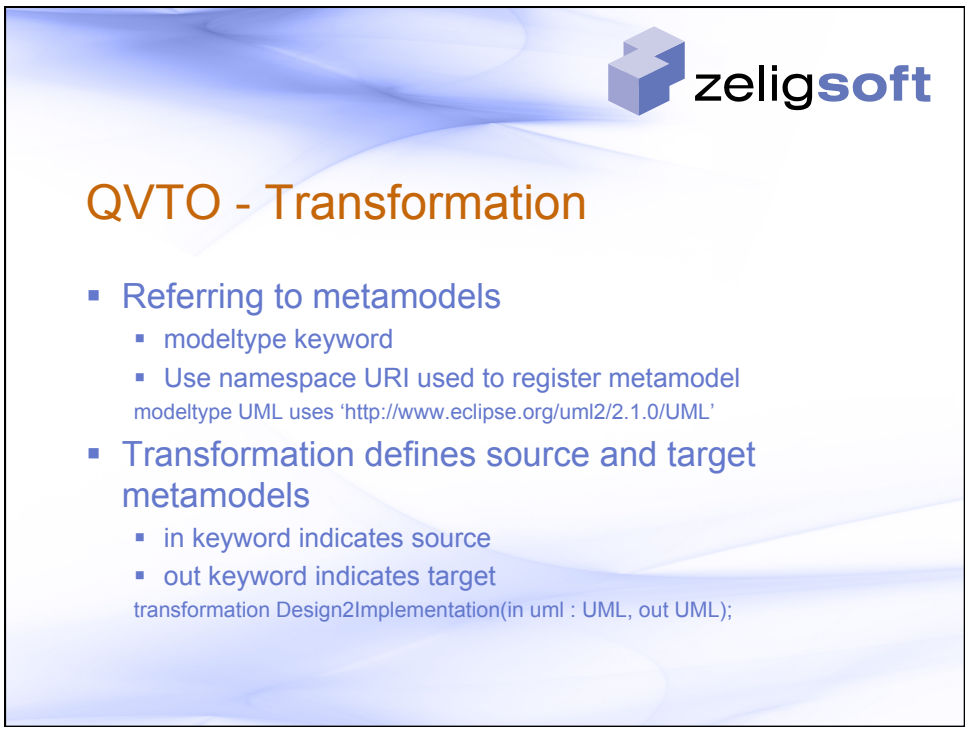
zeligsoft

Executing QVTO Transformations



Toby McClean, Zeligsoft, 2008

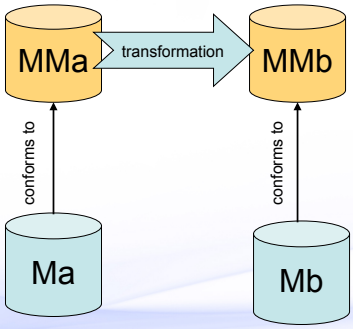

zeligsoft
Integrating with the Workbench
Add an action to the workbench
Create a context
Use an interpreted QVT transformation
Grab the input model element from a Resource or editor
Execute the transformation
Display or persist results and trace


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);

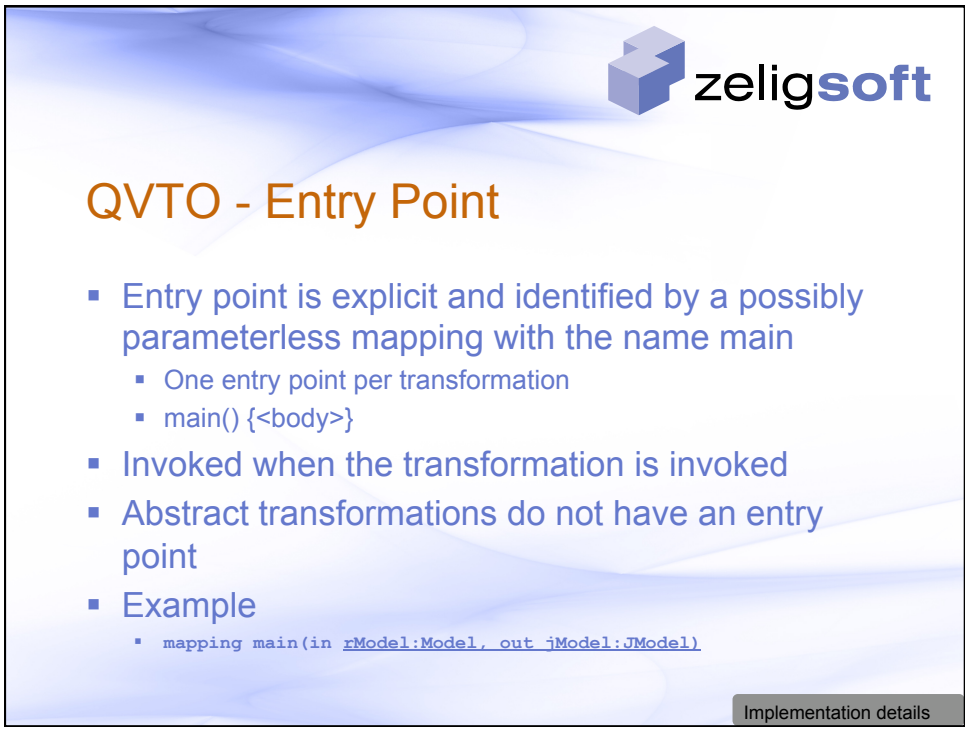
zeligsoft

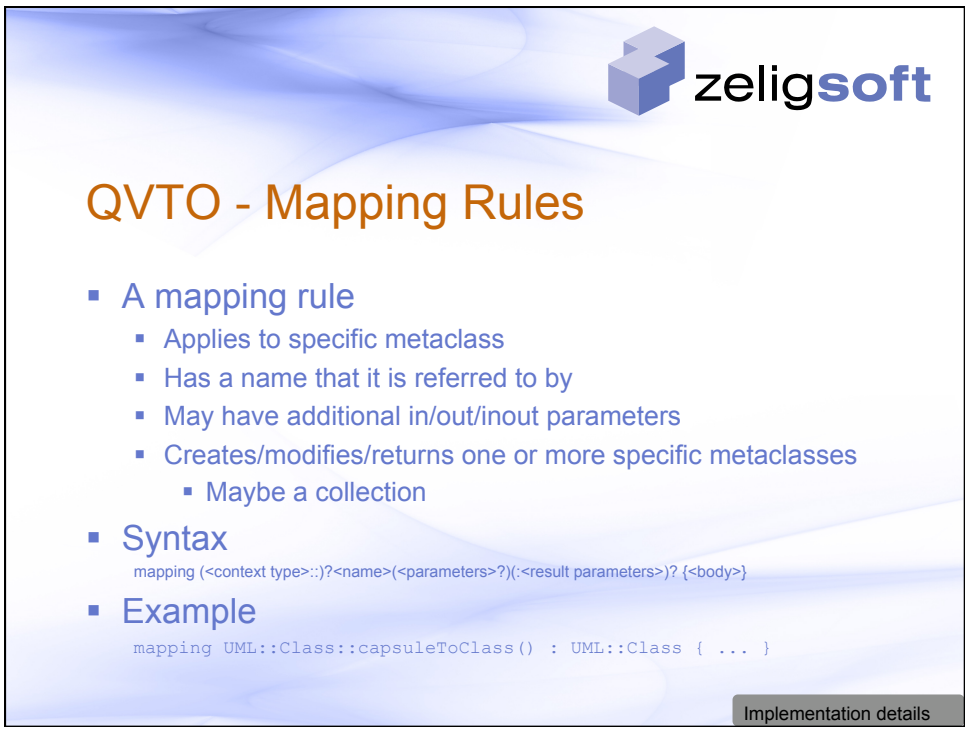
QVTO - Metamodels

- 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

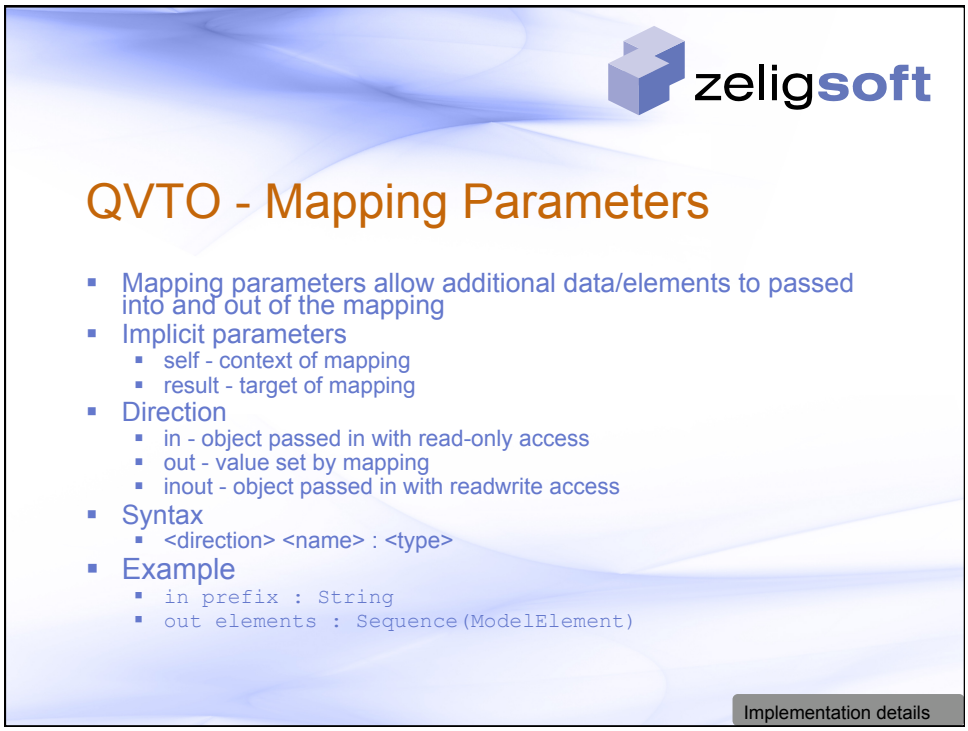
  modeltype <local name> **uses** '<ns uri>';

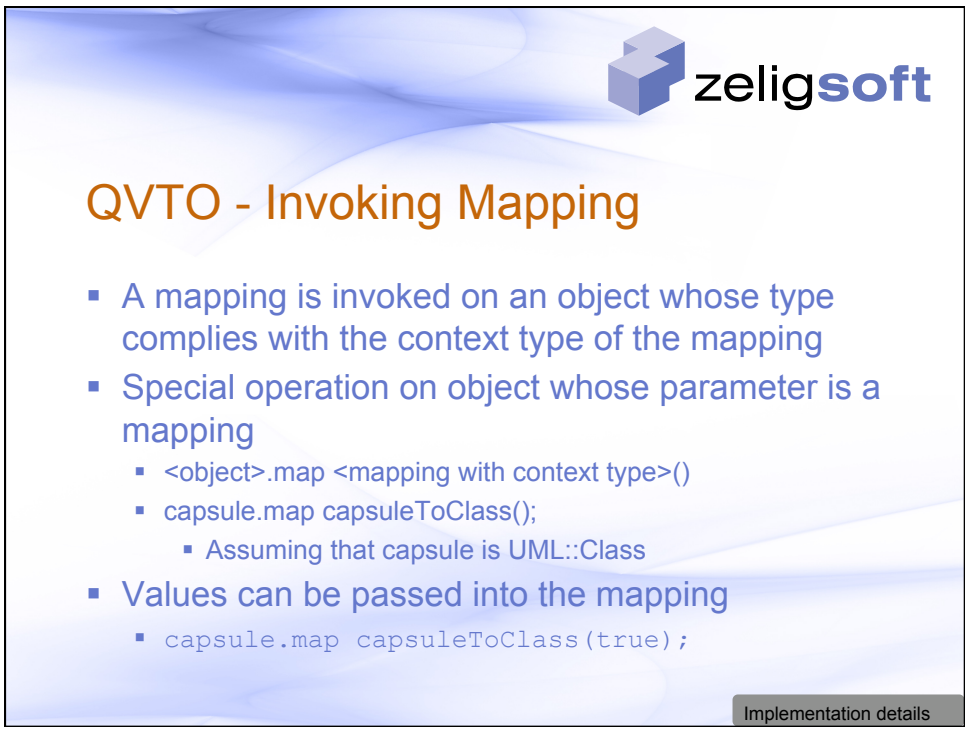
  modeltype UML **uses** 'http://www.eclipse.org/uml2/2.1.0/UML';





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

zeligsoft

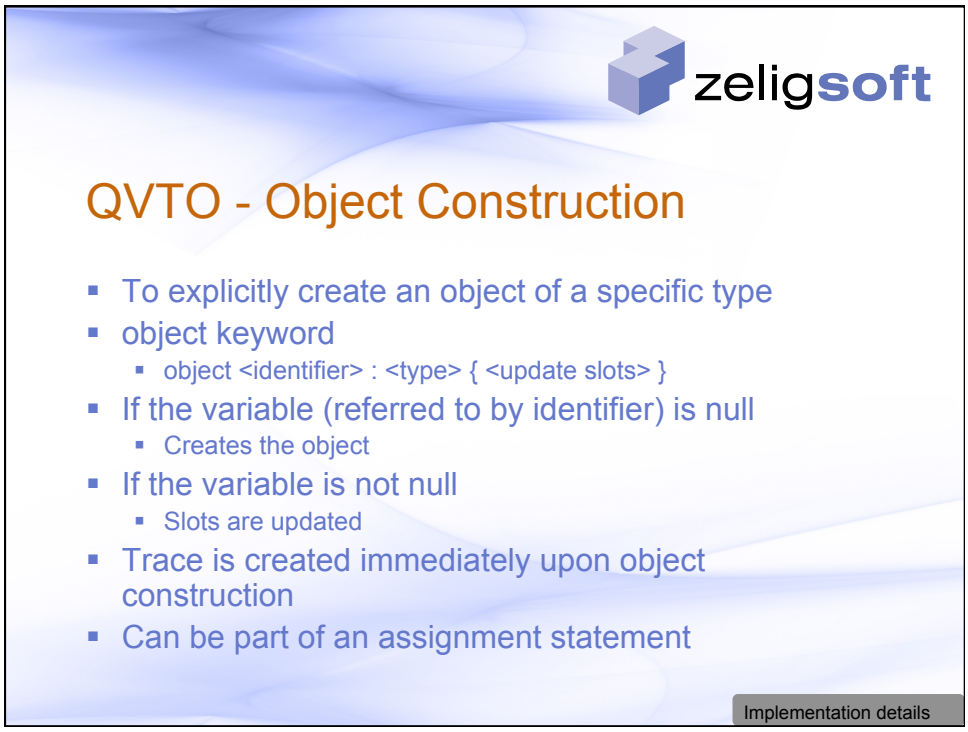
QVTO - Constraining Mappings

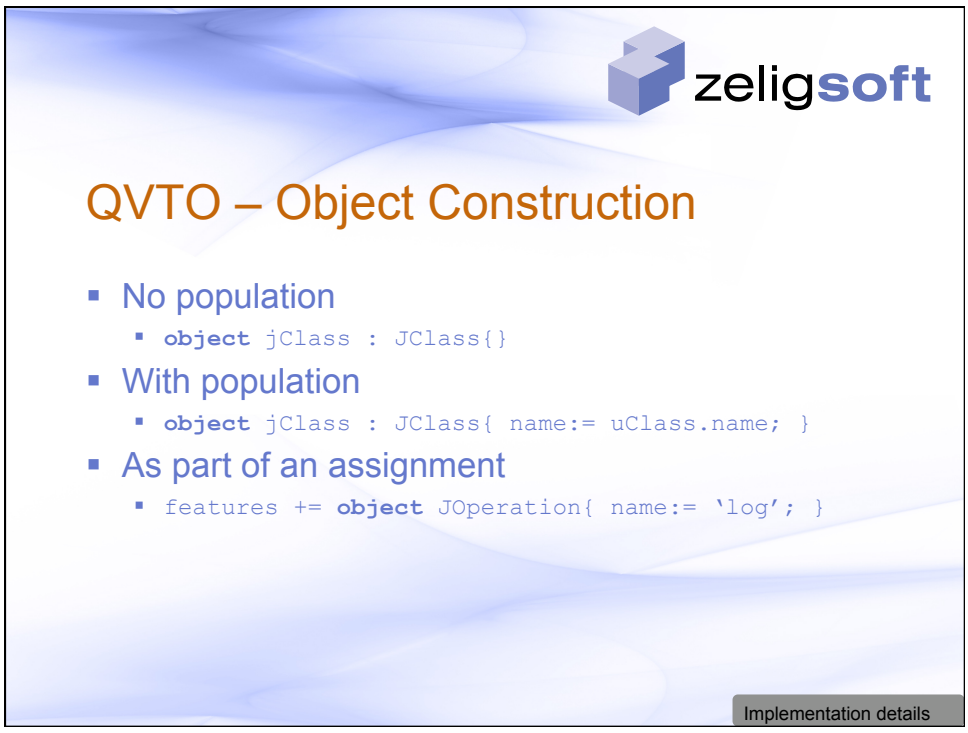
- It is possible to restrict **when** a mapping will execute
  - when clause constrains the input parameters that are accepted for the mapping to execute
  - ... (:<result parameters>)?(**when {** <constraint> **}**)?
- Example
  - `mapping uml::Class::class2JClass() : JClass when { self.isStereotypedBy('Capsule') }`
- Two modes of invocation
  - Standard .map if the context doesn't satisfy the **when** clause then the mapping is not executed and control is returned to the caller
    - **when** clause acts like a guard
  - Strict .xmap if the context doesn't satisfy the **when** clause then an exception is thrown
    - **when** clause acts like a pre-condition

zeligsoft

QVTO - Implementing Mappings

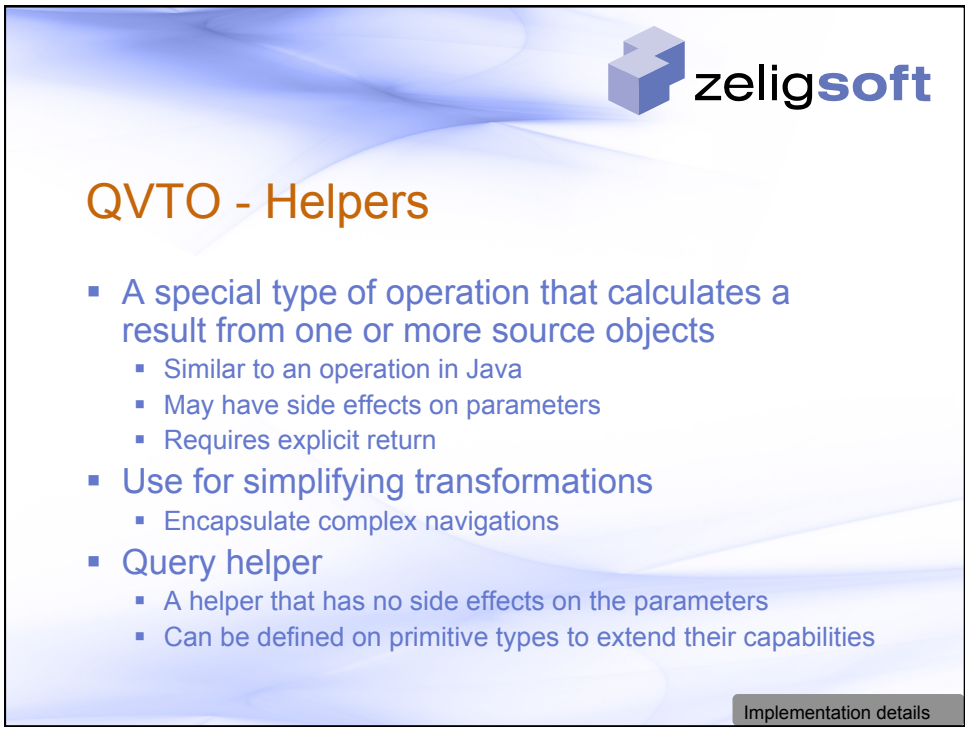
- Four sections to the body of a mapping
  - init
    - variable assignments
    - out parameter assignments
  - instantiation
    - implicitly instantiates out parameters that are null
  - population
    - updating result parameters
  - end
    - mapping invocations
    - logging, assertions, etc.

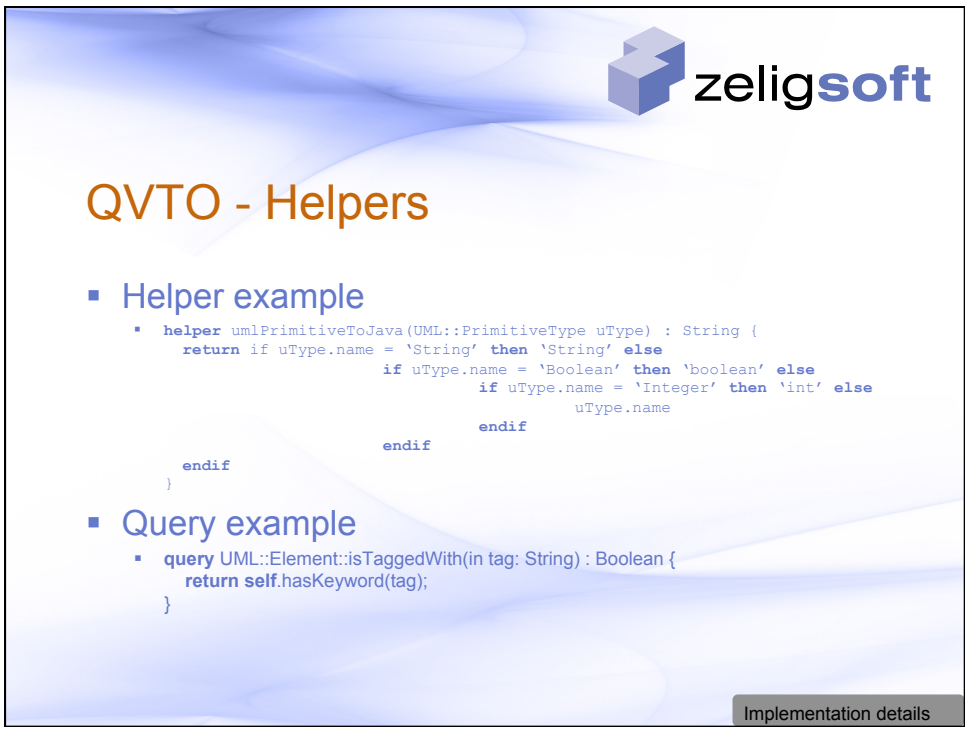




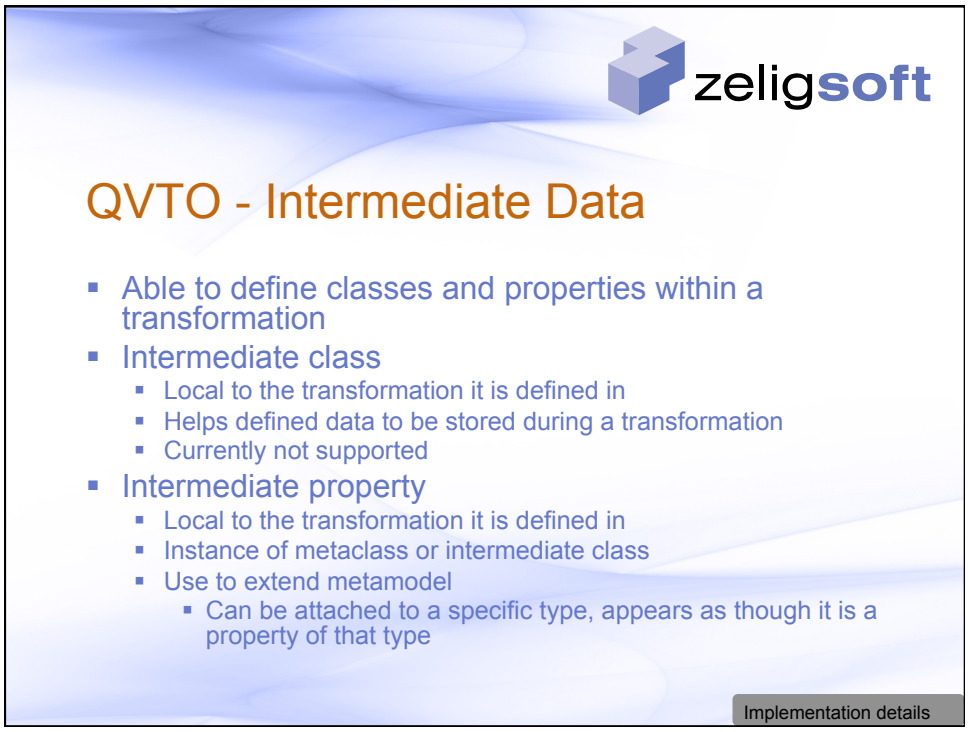
Toby McClean, Zeligsoft, 2008

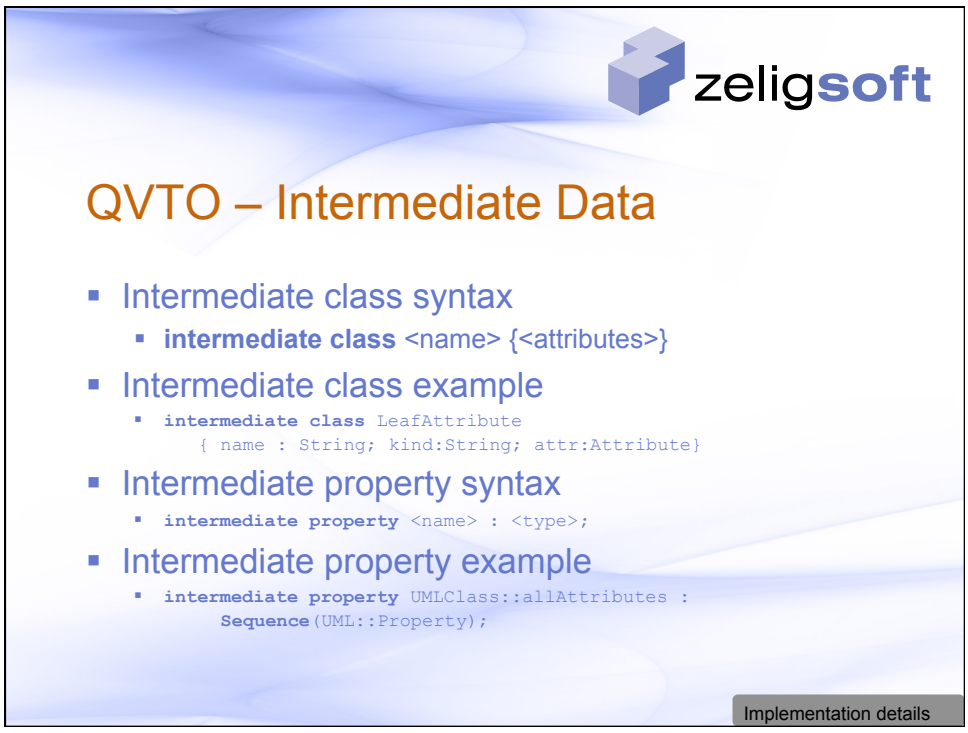
Toby McClean, Zeligsoft, 2008



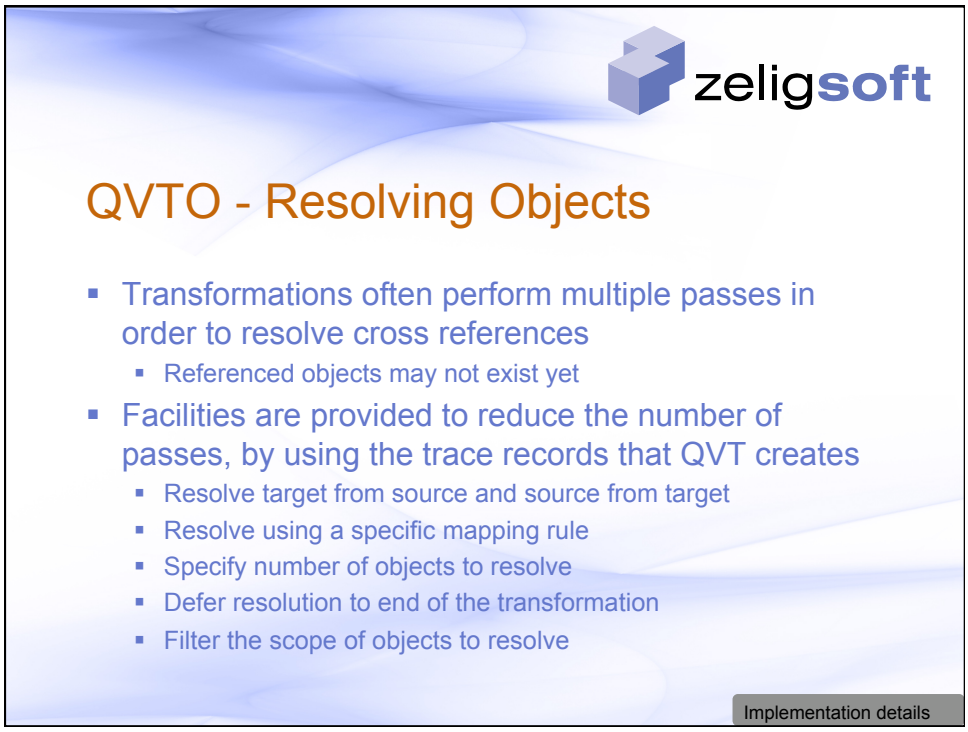


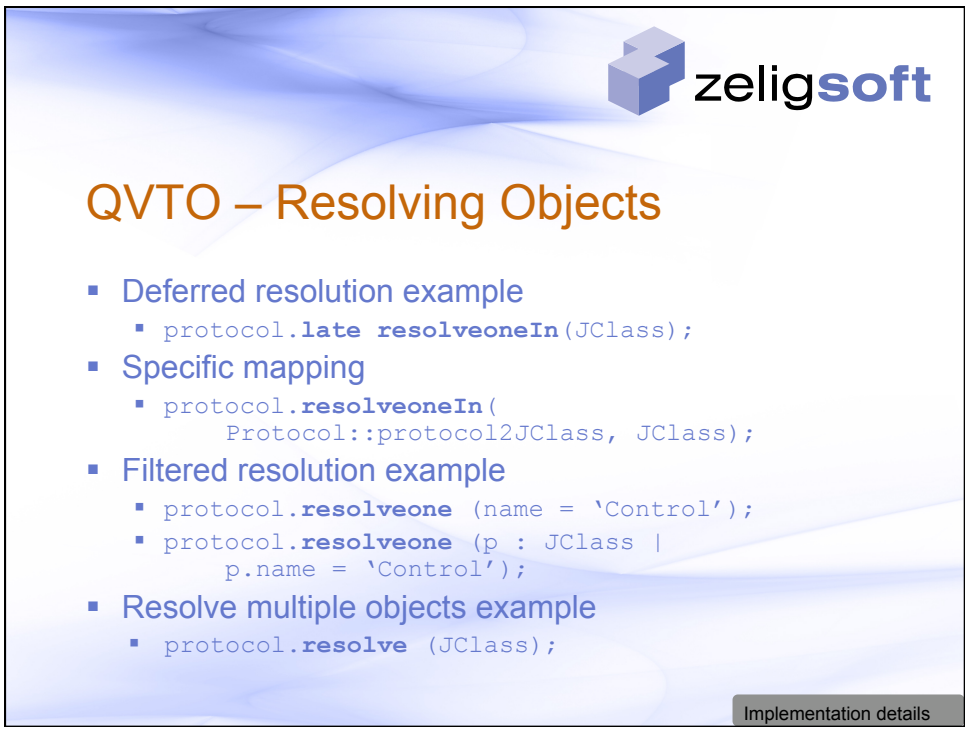
Toby McClean, Zeligsoft, 2008



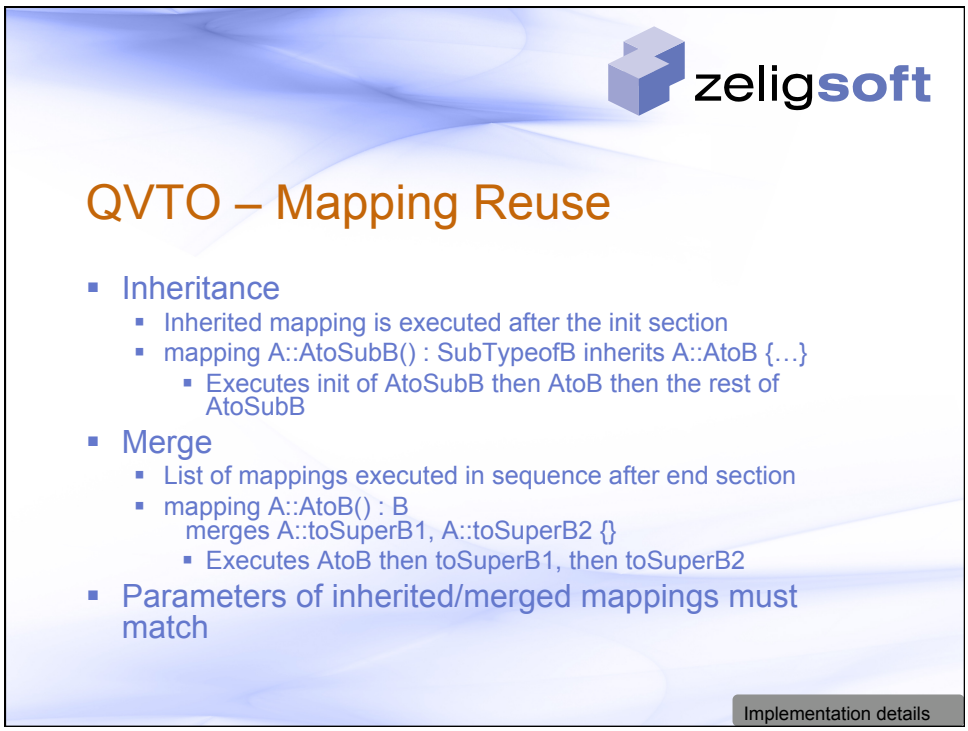


Toby McClean, Zeligsoft, 2008

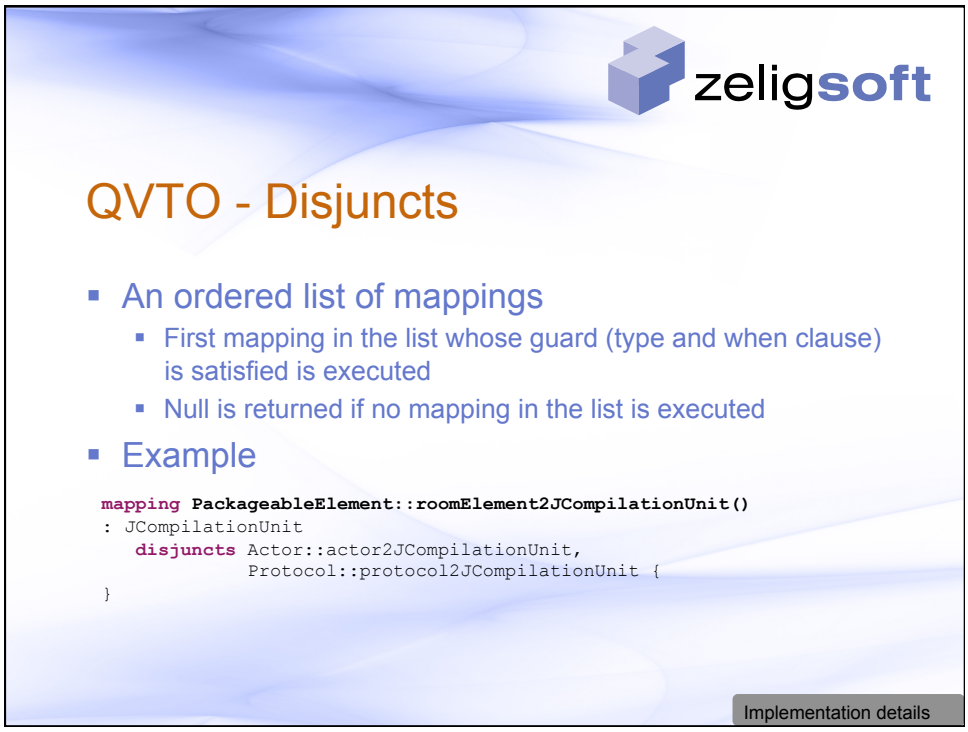


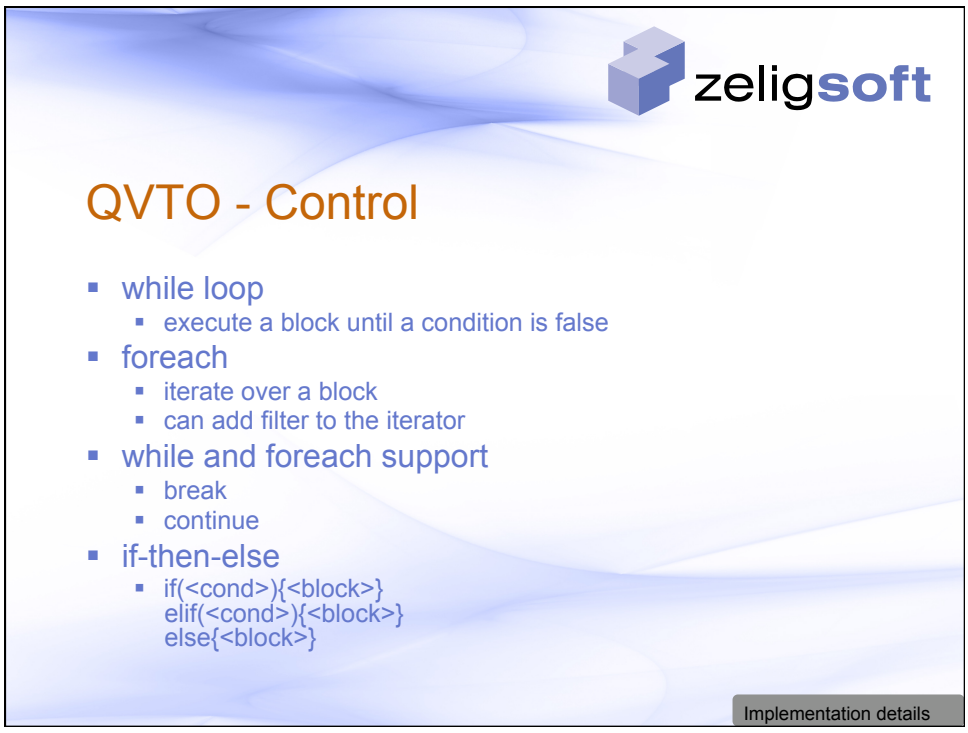


Toby McClean, Zeligsoft, 2008

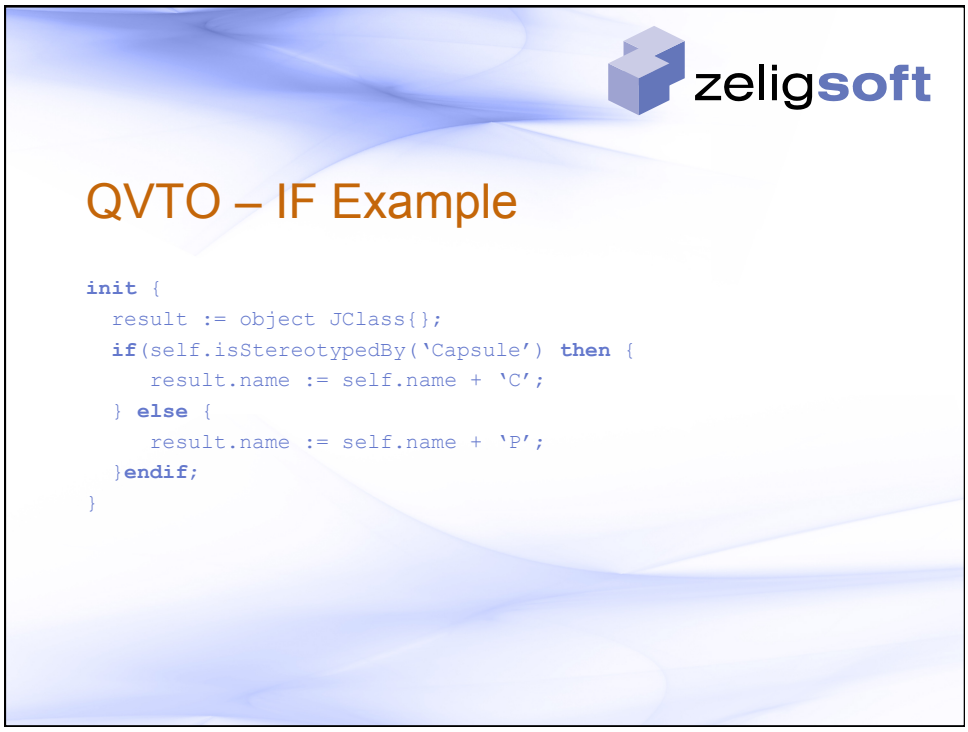


Toby McClean, Zeligsoft, 2008





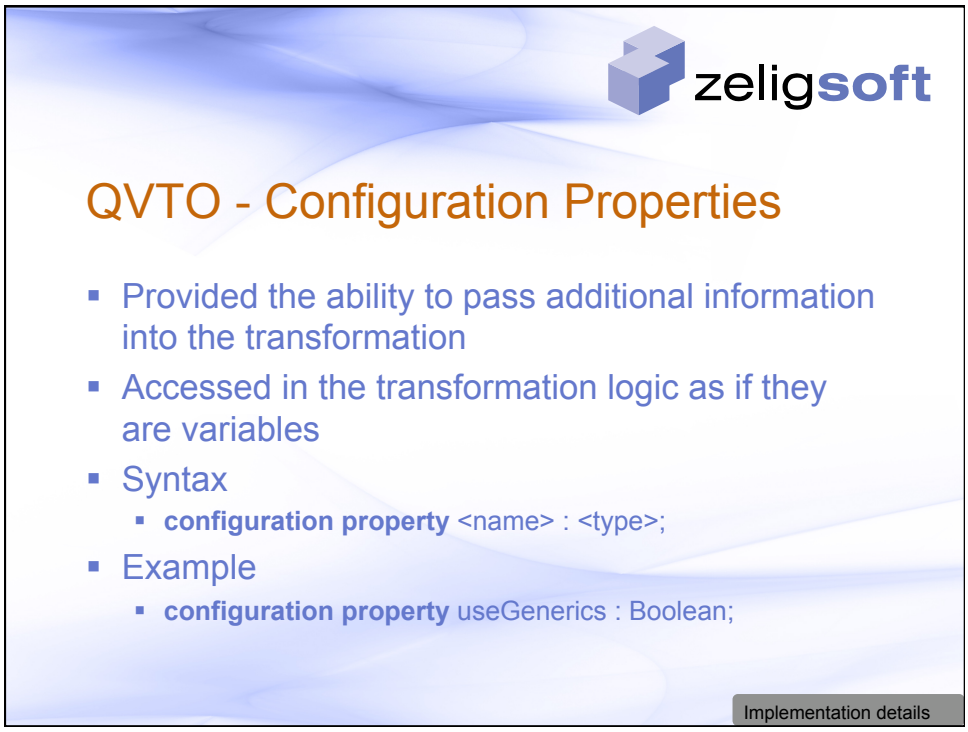
Toby McClean, Zeligsoft, 2008

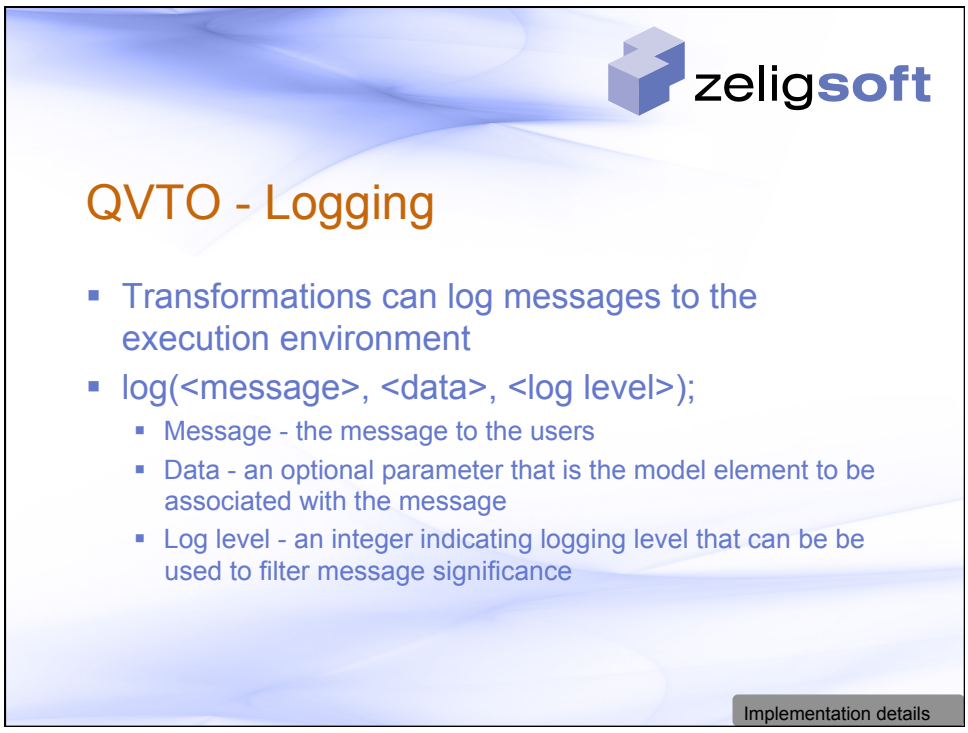


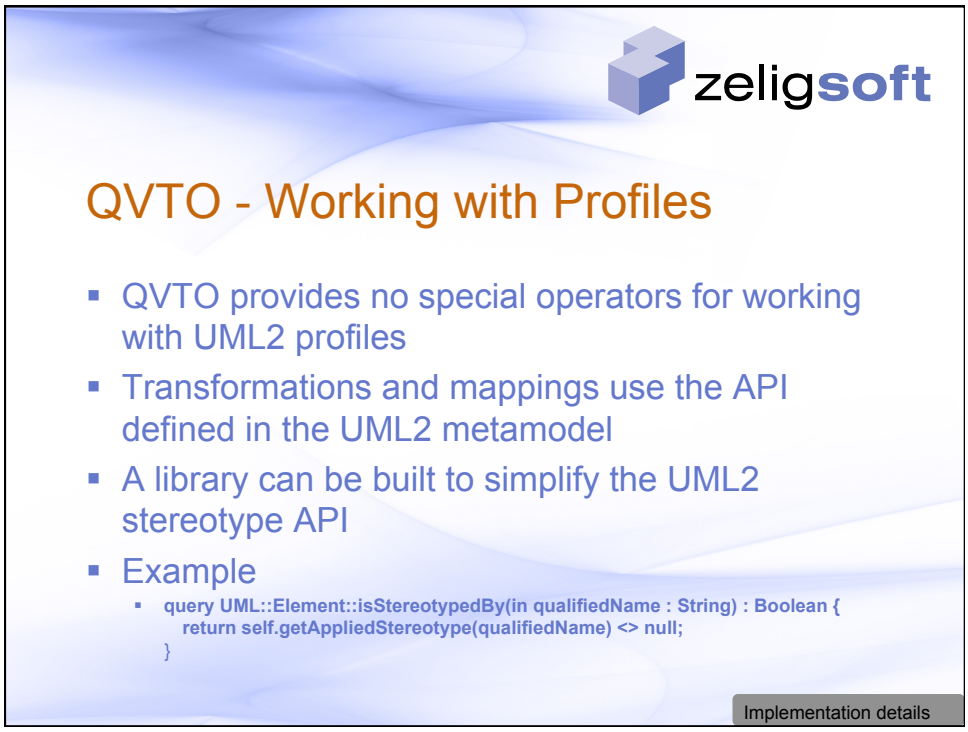
Toby McClean, Zeligsoft, 2008

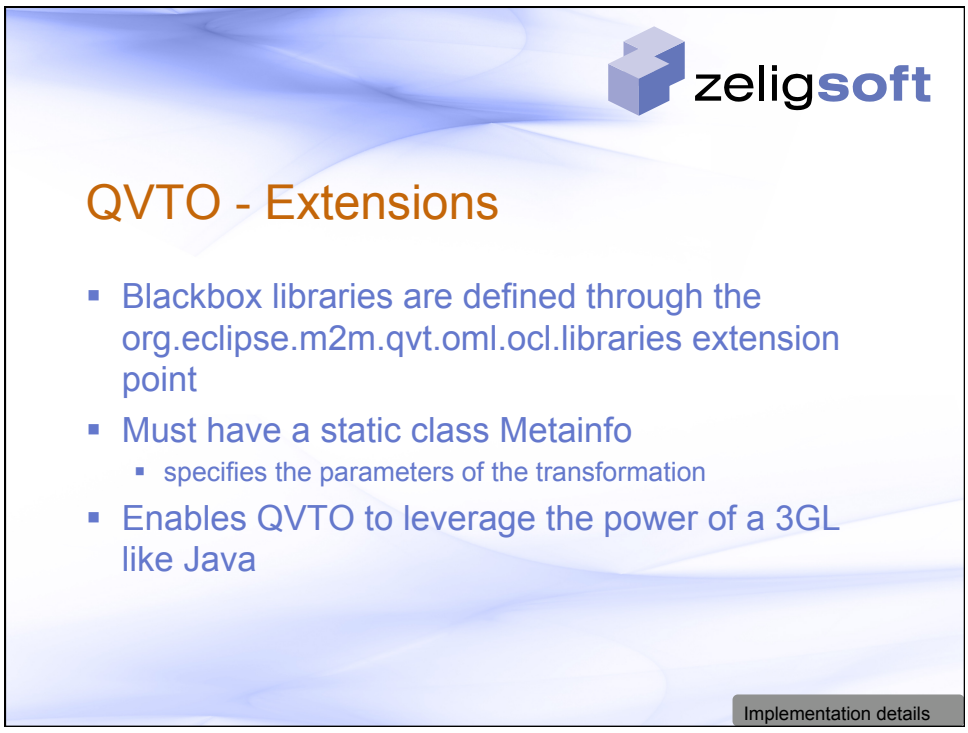

zeligsoft
QVTO - More than 1 Target Model
When multiple target models are specified, need to be able to indicate which one a model is instantiated in
object <type>@<target model>{}
mapping <type>::<name> : <type>@<target model> {}
Implementation details


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;
}
Implementation details

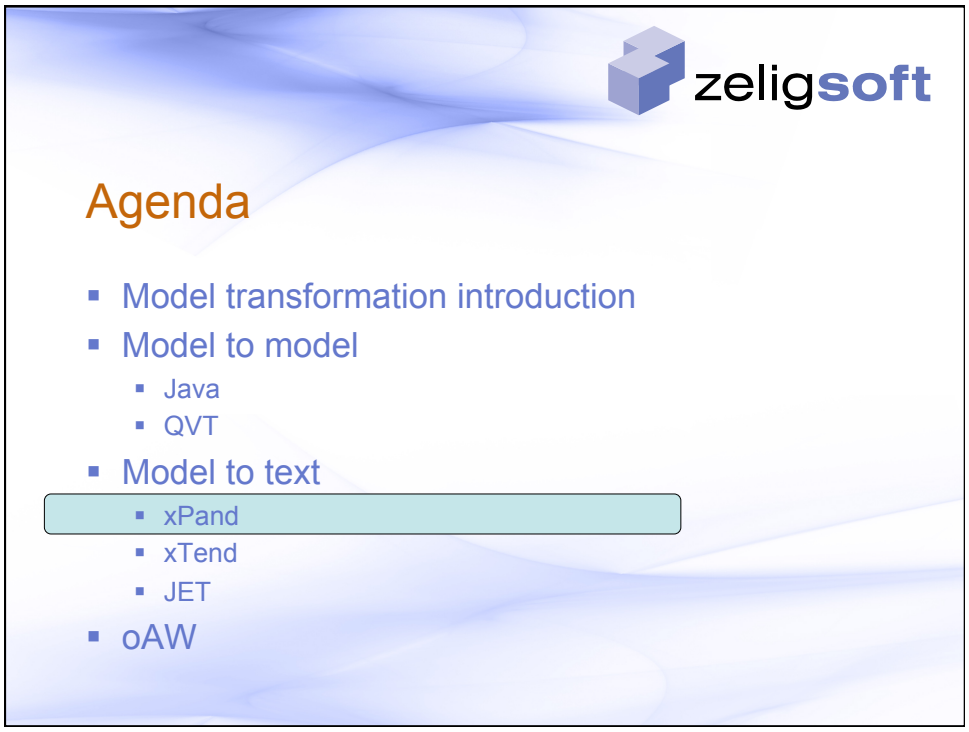

zeligsoft
QVTO - Extensions
Blackbox libraries are defined through the org.eclipse.m2m.qvt.oml.ocl.libraries extension point
Must have a static class Metainfo
specifies the parameters of the transformation
Enables QVTO to leverage the power of a 3GL like Java
Implementation details

QVTO - Programatically Invoking

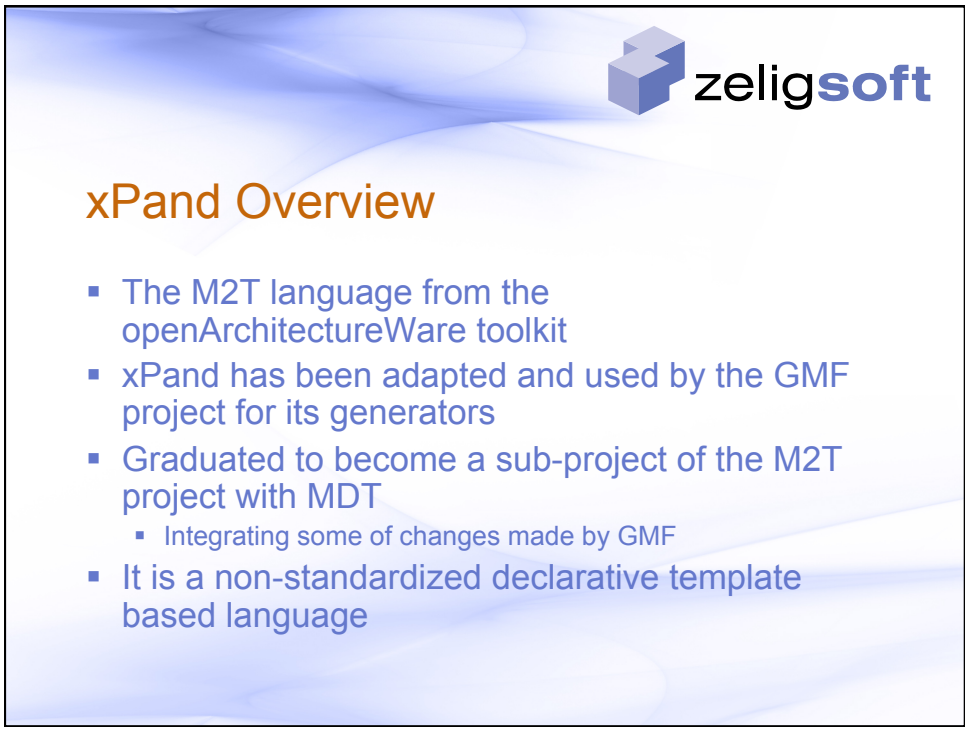
```
URI transformation =
  URI.createURI("platform:/resource/com.eett.exercises.m2m.qvt/transforms/
Room2Java.qvto");
IFile qvtFile = getIFile(transformation);
IContext qvtContext = new Context();
QvtInterpretedTransformation trans =
   new QvtInterpretedTransformation(qvtFile);
EObject inputModel =
   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());
}
```

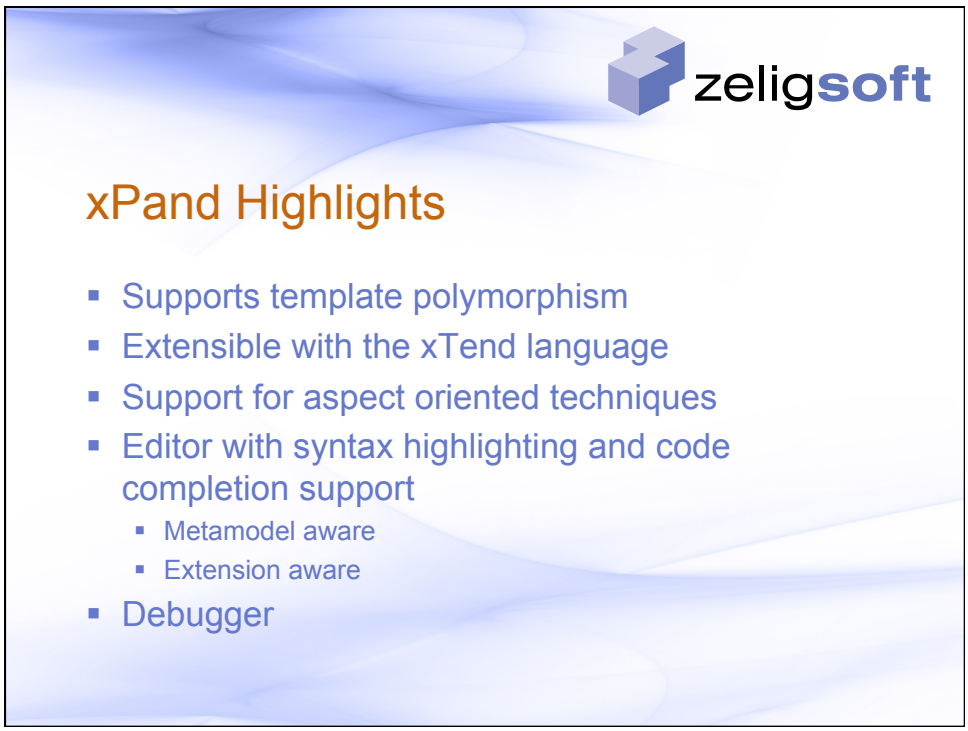
Summary

- 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

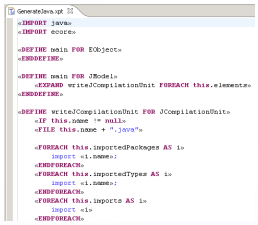

zeligsoft
Eclipse Transformation Technologies
M2T with xPand


zeligsoft
xPand Overview
The M2T language from the openArchitectureWare toolkit
xPand has been adapted and used by the GMF project for its generators
Graduated to become a sub-project of the M2T project with MDT
Integrating some of changes made by GMF
It is a non-standardized declarative template based language


zeligsoft
xPand Highlights
Supports template polymorphism
Extensible with the xTend language
Support for aspect oriented techniques
Editor with syntax highlighting and code completion support
Metamodel aware
Extension aware
Debugger

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

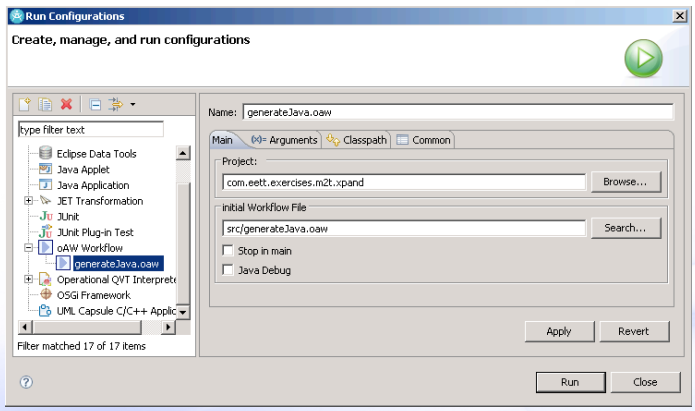
zeligsoft

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

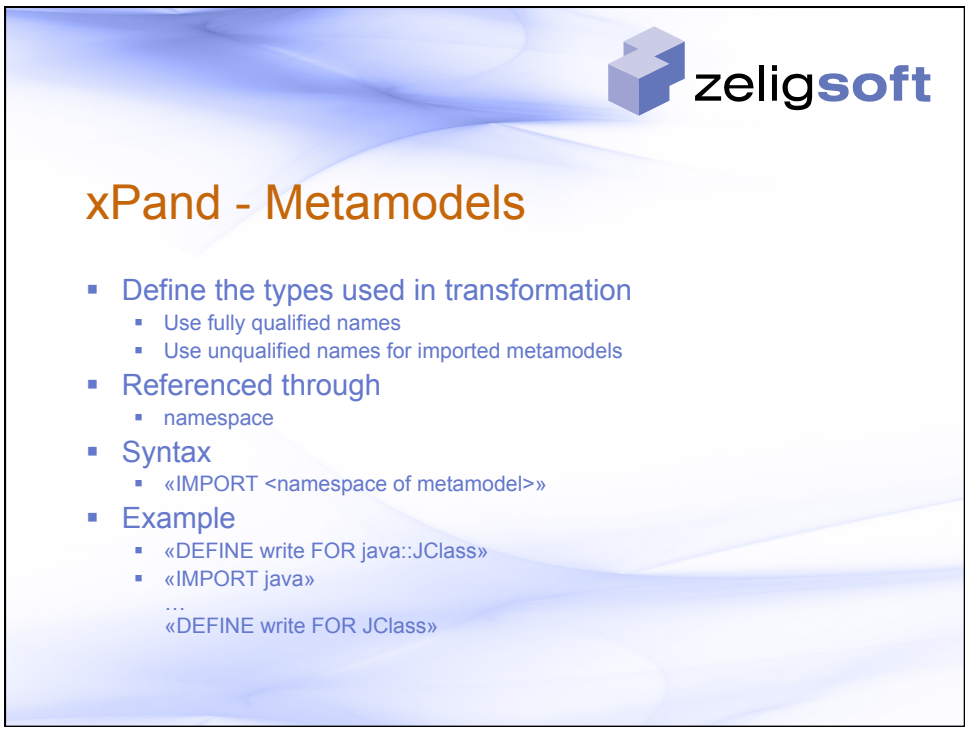
Executing an xPand Transformation



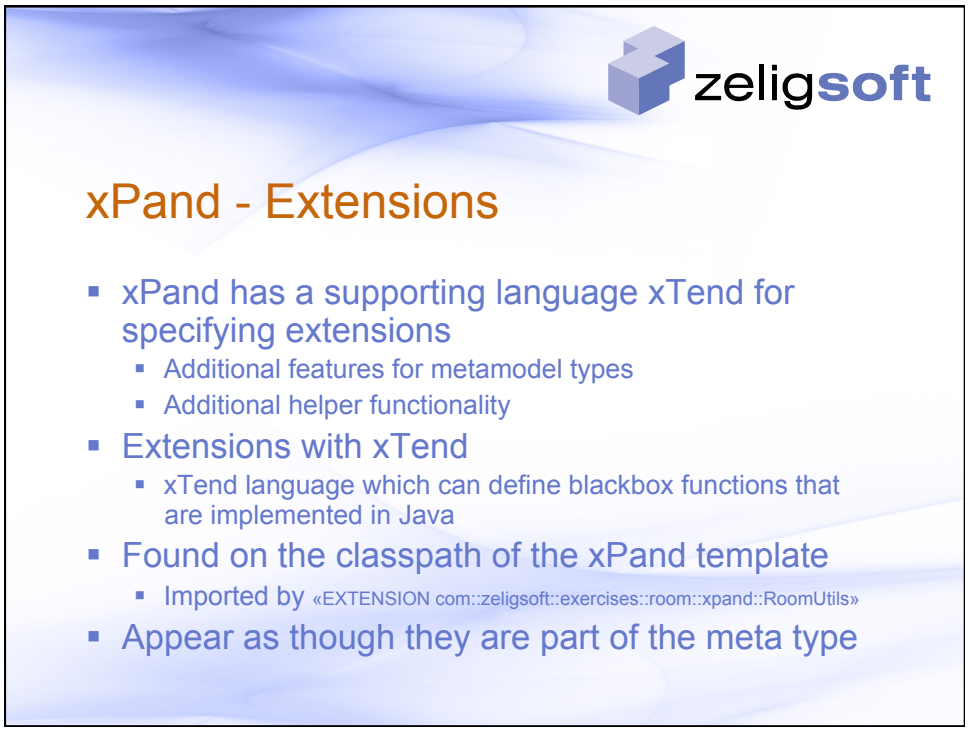
zeligsoft

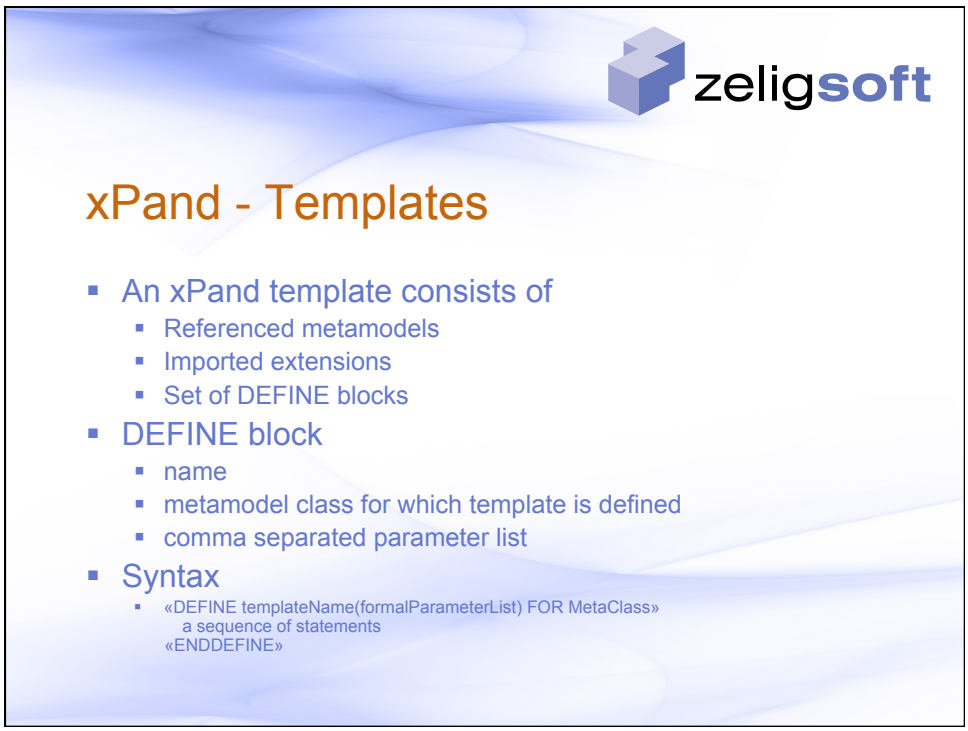
Integrating with Workbench

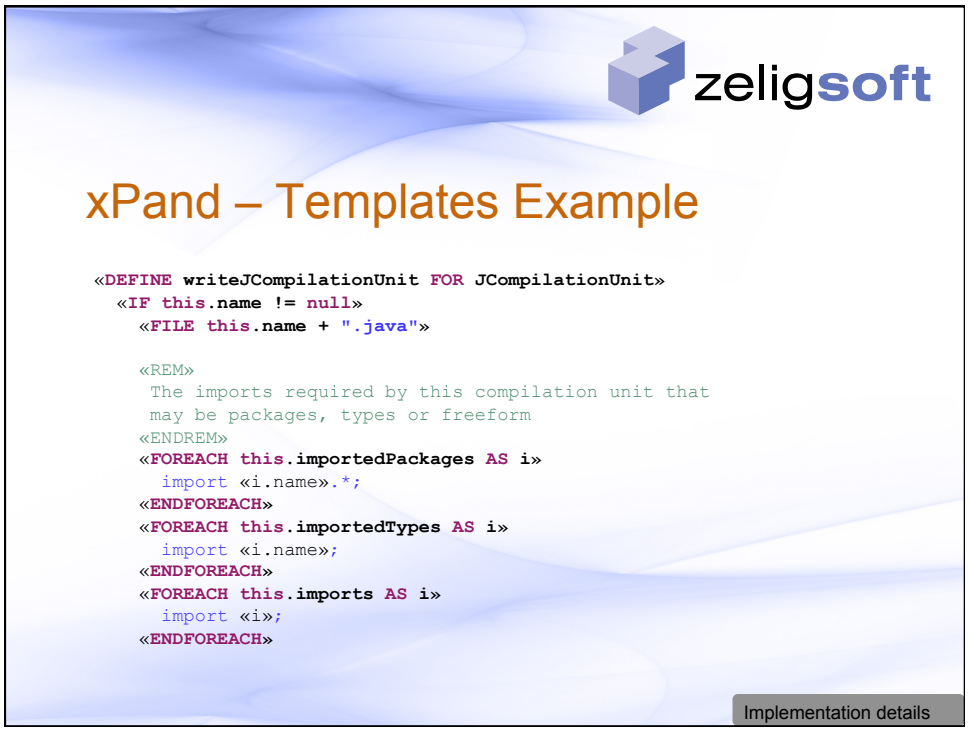
- Contribute an action to the editor's menu
  - Implement the action handler
    - Call the workflow passing in the selected element as the source
- Alternative is append to an existing action
  - e.g. the ROOM to Java transformation
- Test
  - In runtime workbench
- Deploy
  - Publish the plug-in

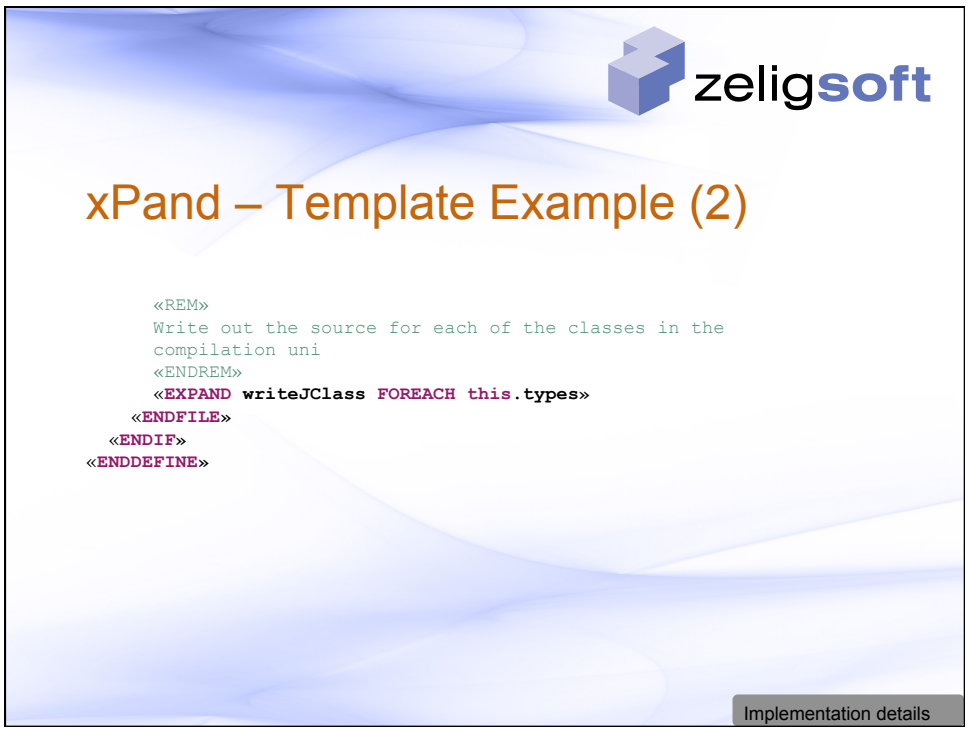


Toby McClean, Zeligsoft, 2008

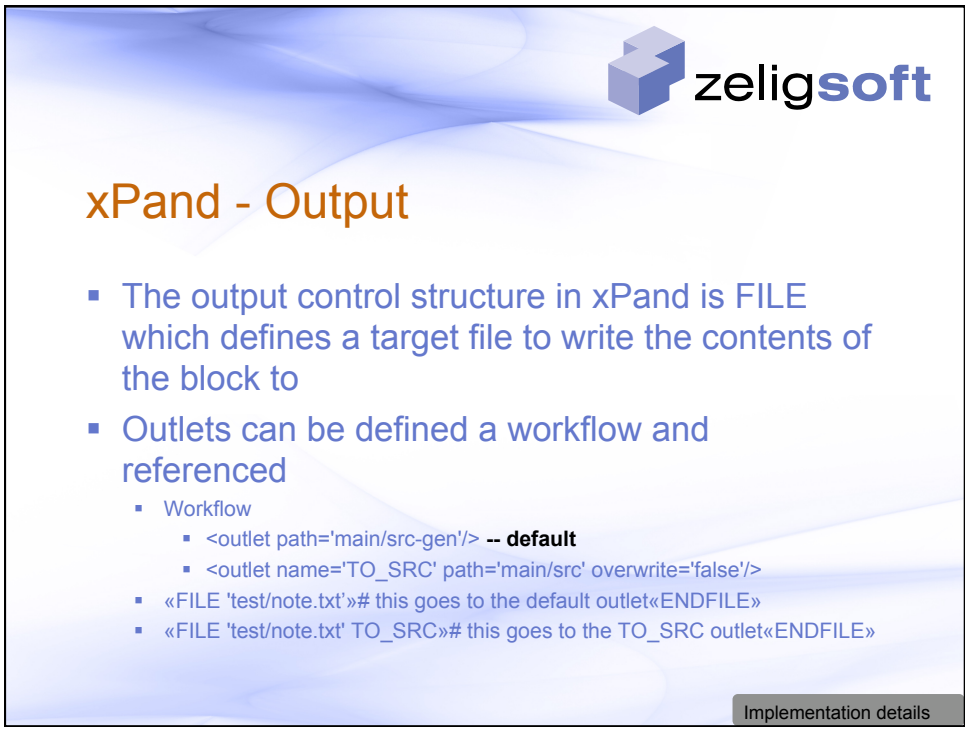

zeligsoft
xPand - Extensions
xPand has a supporting language xTend for specifying extensions
Additional features for metamodel types
Additional helper functionality
Extensions with xTend
xTend language which can define blackbox functions that are implemented in Java
Found on the classpath of the xPand 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»

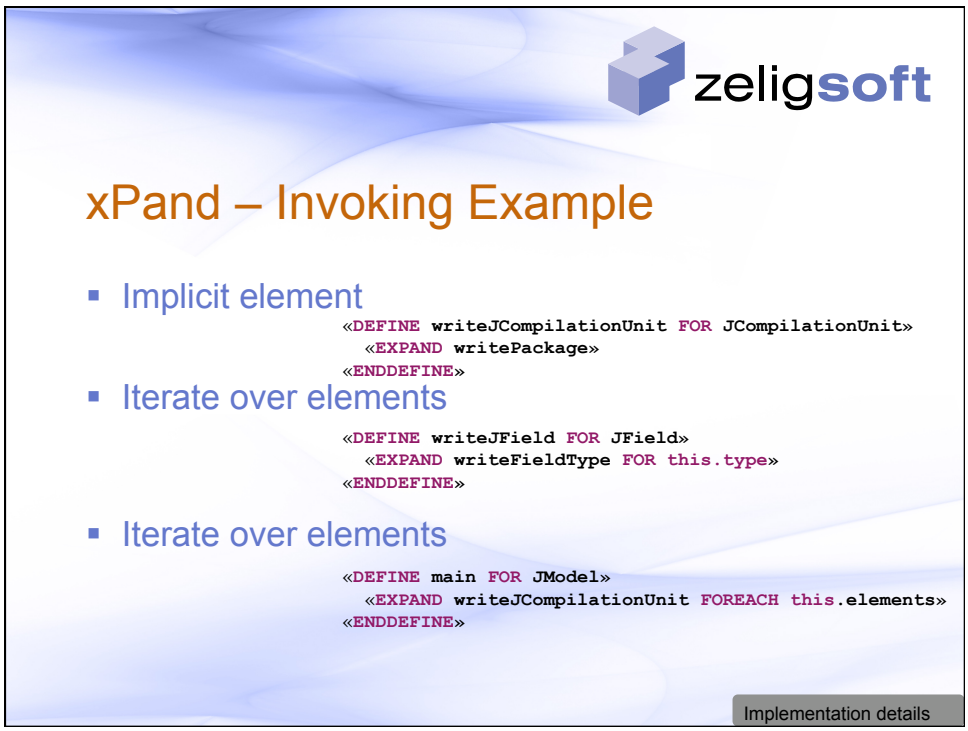




Toby McClean, Zeligsoft, 2008

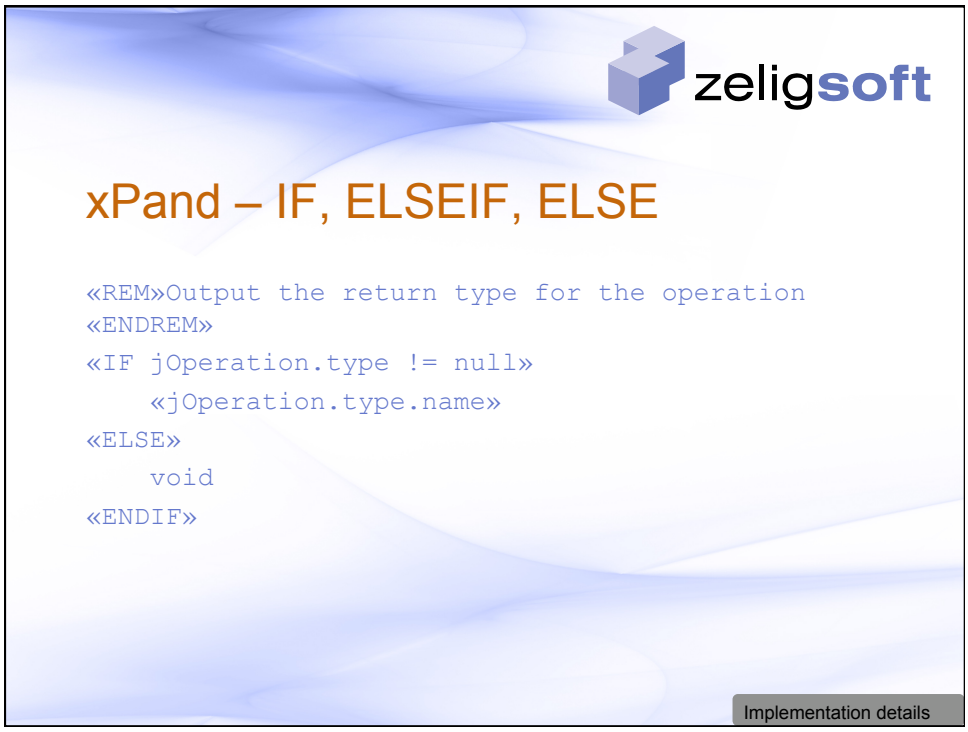

zeligsoft
xPand - Output
The output control structure in xPand is FILE which defines a target file to write the contents of the block to
Outlets can be defined a workflow and referenced
Workflow
<outlet path='main/src-gen'/> -- default
<outlet name='TO_SRC' path='main/src' overwrite='false'/>
«FILE 'test/note.txt'»#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

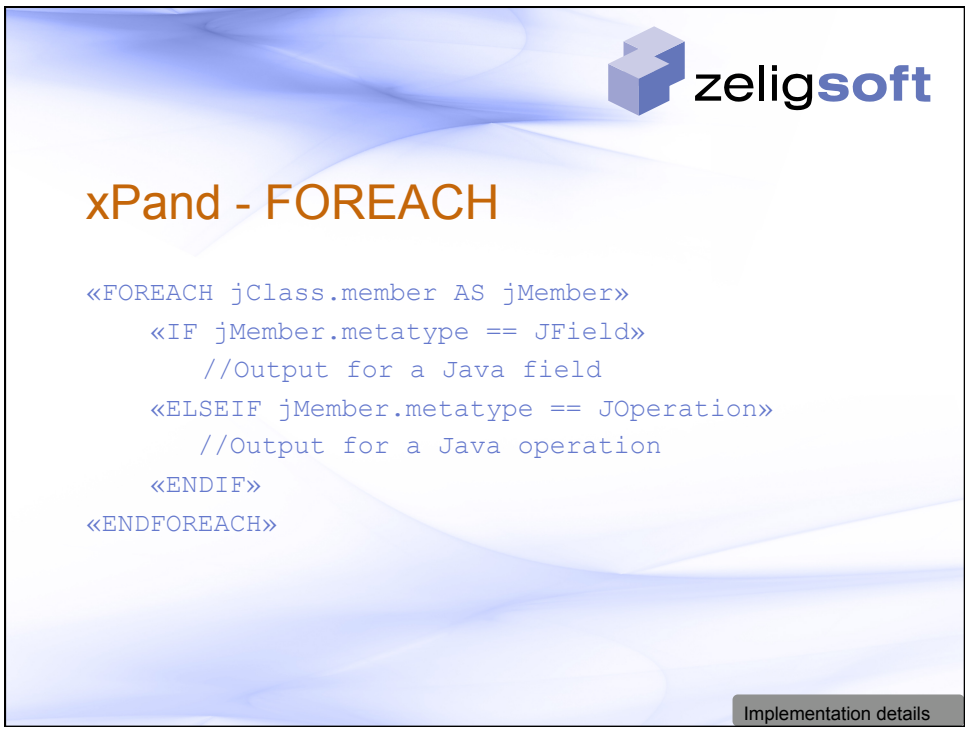

zeligsoft
xPand - Invoking a Template
The xPand language uses EXPAND to invoke a template
«EXPAND definitionName [(parameterList)]
[FOR expression | FOREACH expression [SEPARATOR expression]]»
FOR
Invokes the template on the specified element
FOREACH
Invokes the template on each element in the collection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



Toby McClean, Zeligsoft, 2008

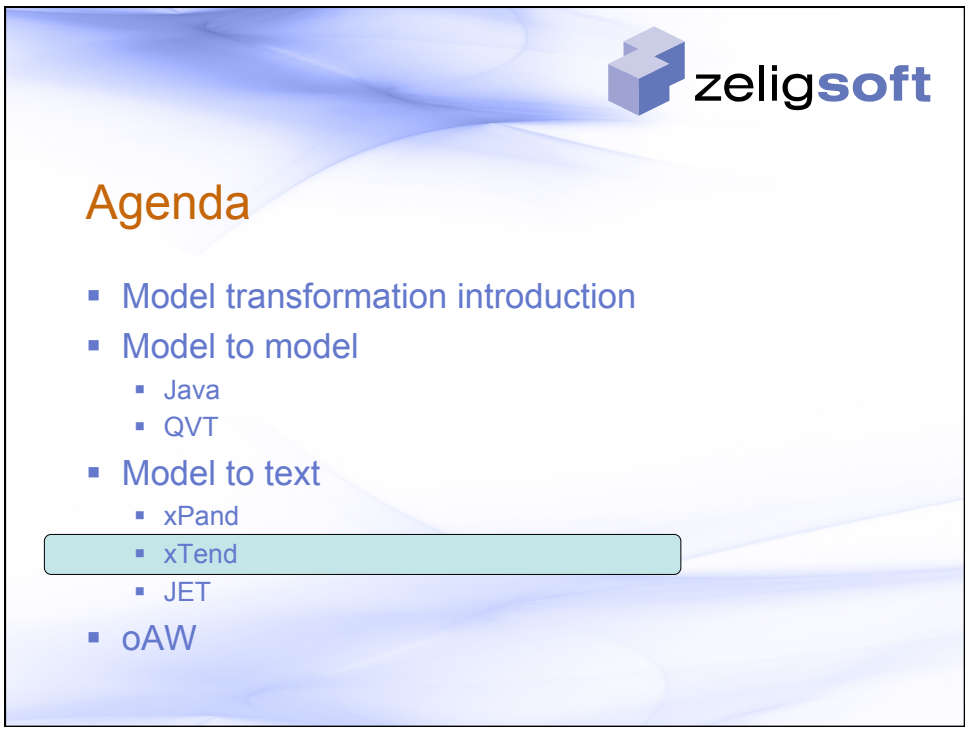
Toby McClean, Zeligsoft, 2008

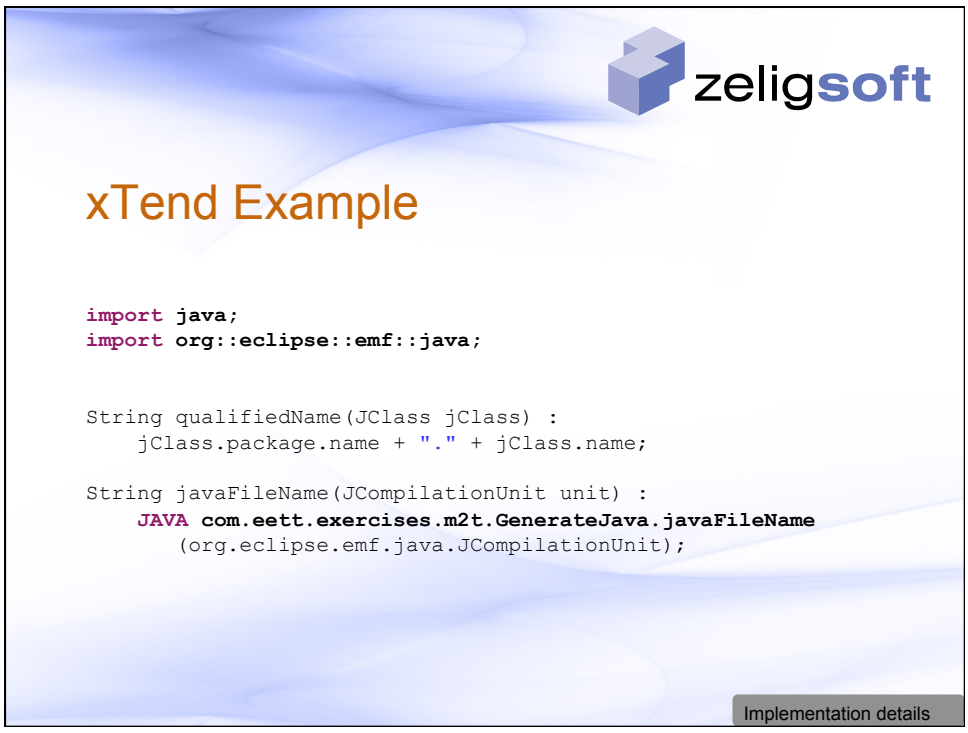




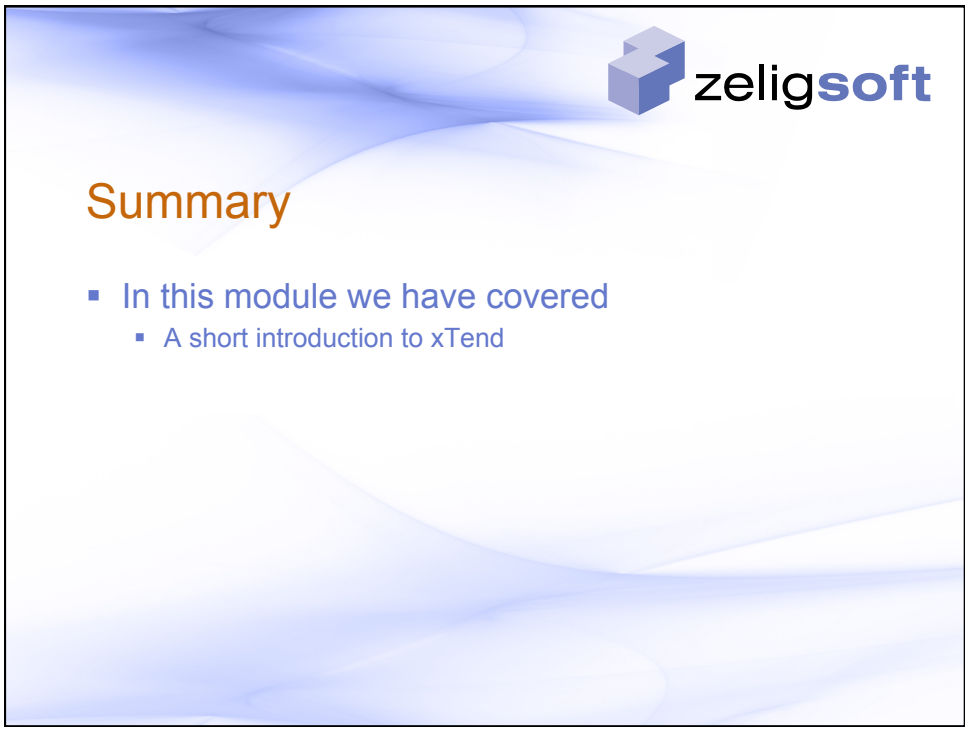
Toby McClean, Zeligsoft, 2008

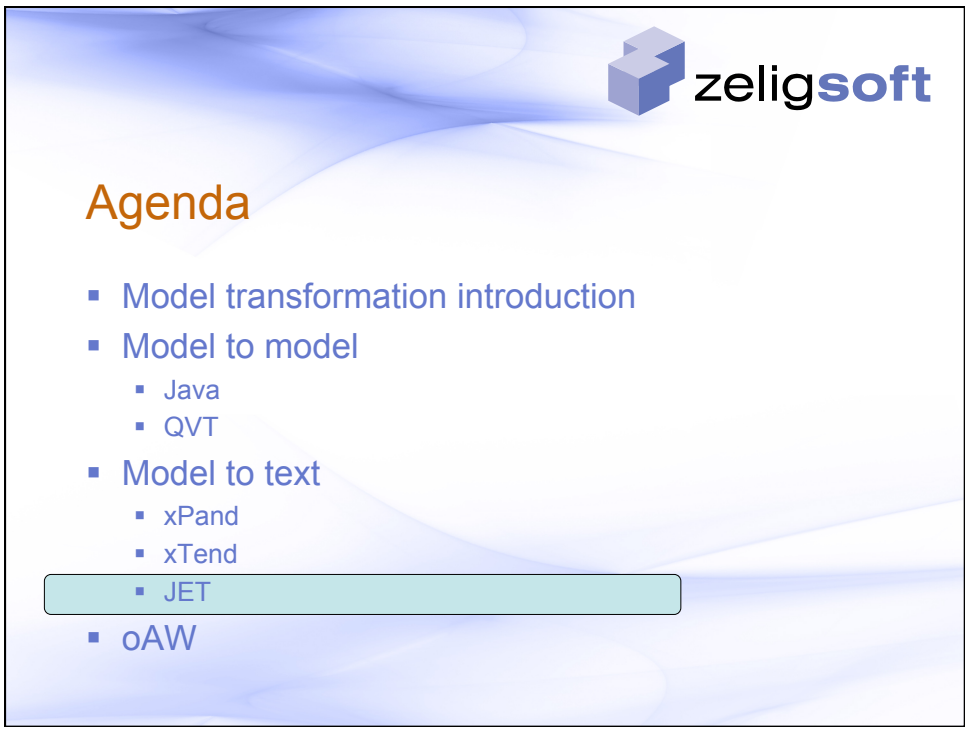

zeligsoft
Summary
In this module we explored
xPand transformations
Invoking xPand
xPandsyntax

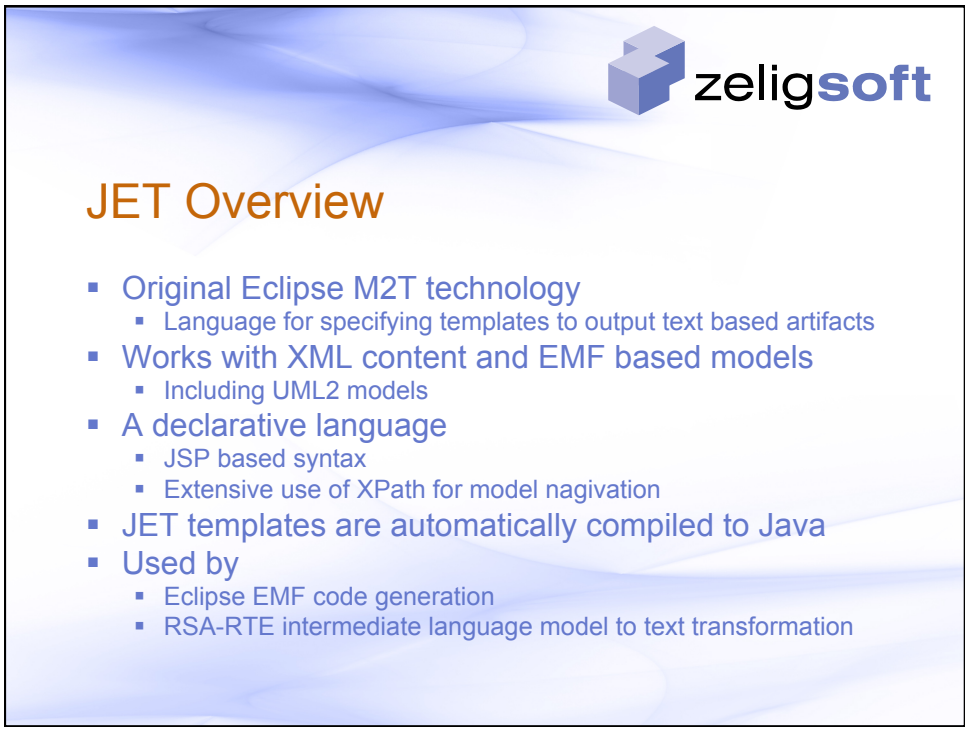

zeligsoft
Agenda
Model transformation introduction
Model to model
Java
QVT
Model to text
xPand
xTend
JET
oAW



Toby McClean, Zeligsoft, 2008


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

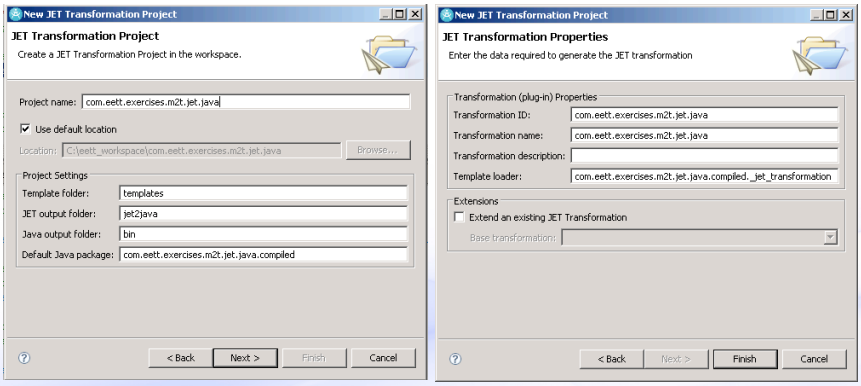

zeligsoft
Agenda
Model transformation introduction
Model to model
Java
QVT
Model to text
xPand
xTend
JET
oAW


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



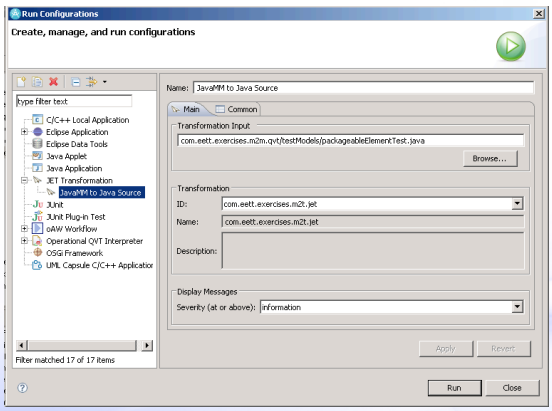
zeligsoft

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

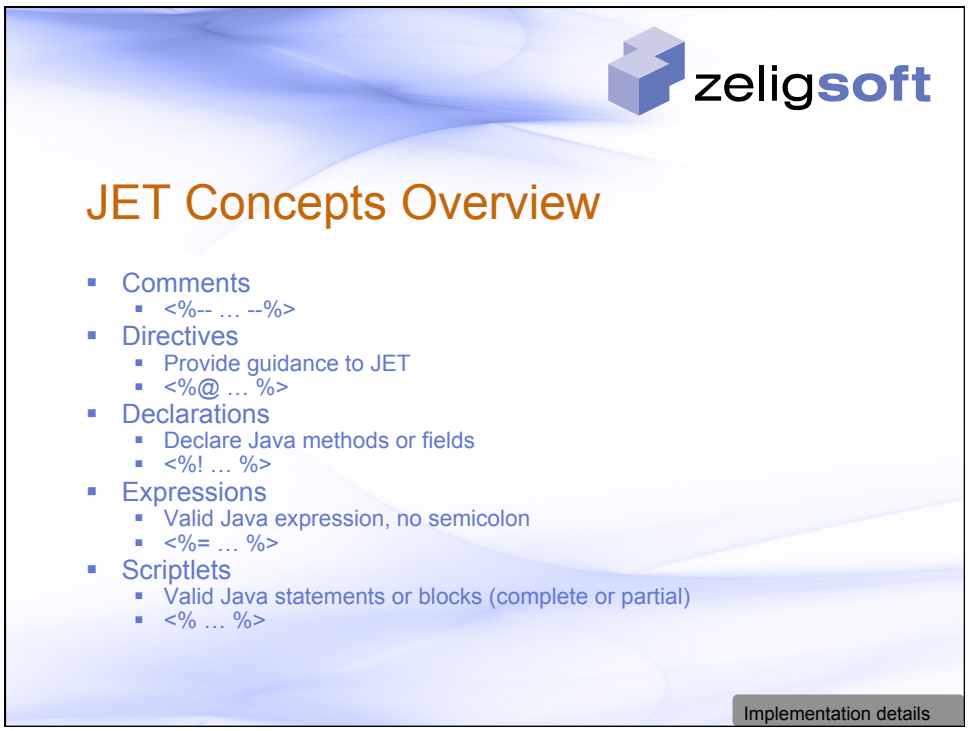


zeligsoft

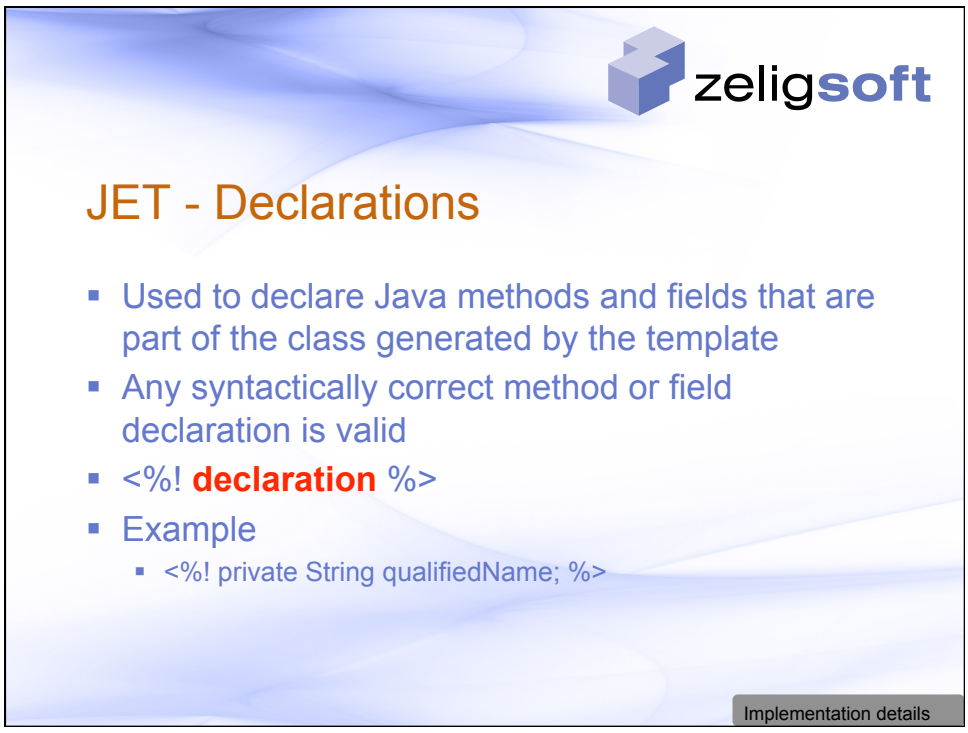
Integrating with Workbench

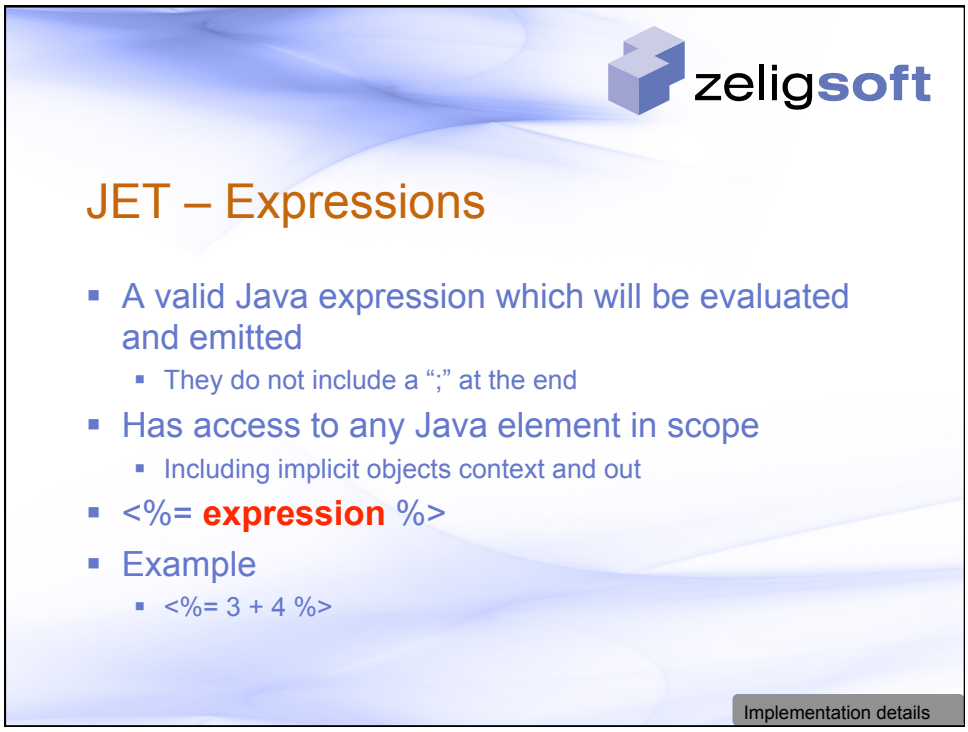
- Contribute an action to the editor's menu
  - Implement the action handler
    - Call the workflow passing in the selected element as the source
- Alternative is append to an existing action
  - e.g. the ROOM to Java transformation
- Test
  - In runtime workbench
- Deploy
  - Publish the plug-in

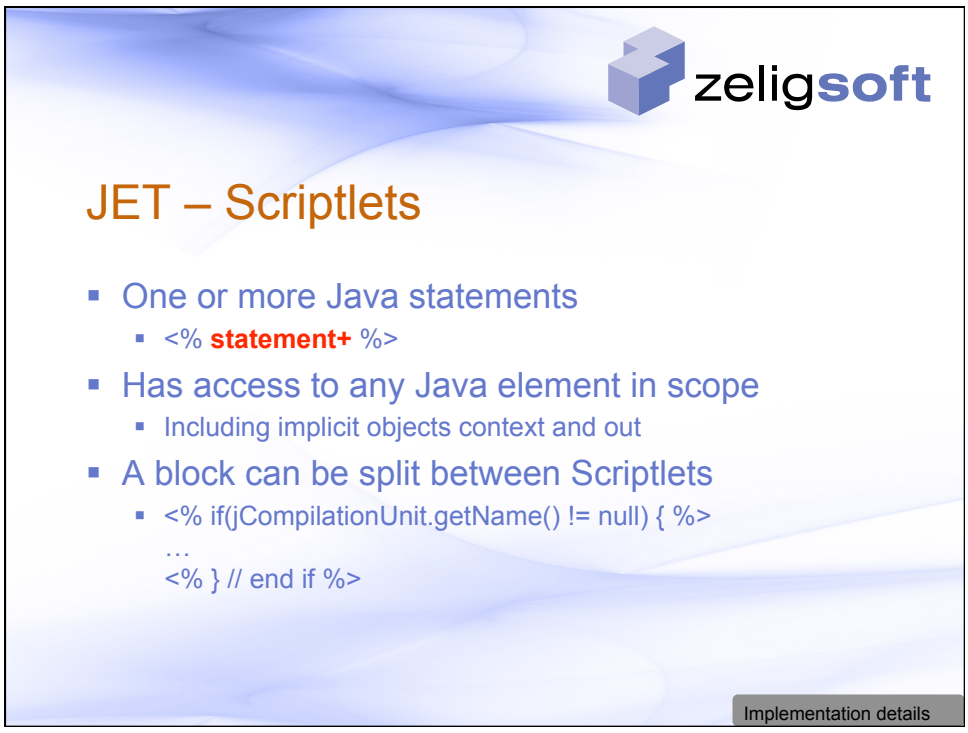

zeligsoft
Integrating with RSA-RTE
Contribute an action
to the Project Explorer context menu
to the Diagram Editor context menu
Implement action handler
Class that implements IEditorActionDelegate
Test
In runtime workbench
Deploy
Publish the plug-in


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


zeligsoft
JET – Directives
@jet
Control or affect the Java code that is created by the JET compiler
<%@jet package="" class="" imports="" startTag="" endTag="" />
@taglib
Import a tag library that is used in the template and assign it to a namespace
Control tags – org.eclipse.jet.controlTags
Workspace tags – org.eclipse.jet.workspaceTags
Java tags – org.eclipse.jet.javaTags
Format tags – org.eclipse.jet.java.formatTags
Implementation details


zeligsoft
JET - Declarations
Used to declare Java methods and fields that are part of the class generated by the template
Any syntactically correct method or field declaration is valid
<%! declaration %>
Example
<%! private String qualifiedName; %>
Implementation details


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>
```

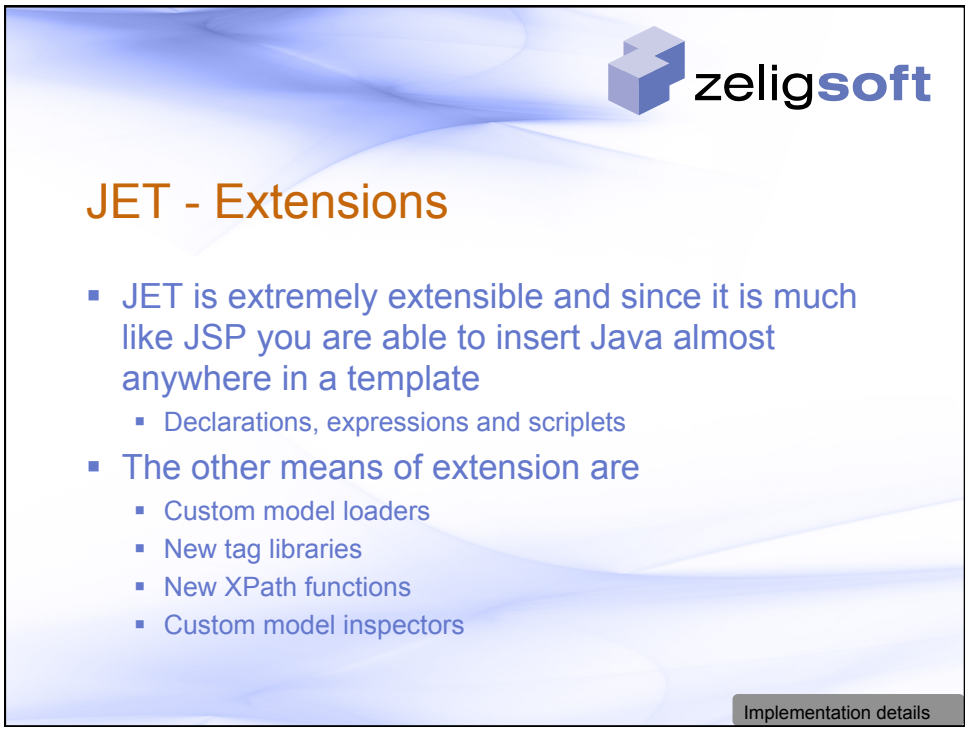
Implementation details

zeligsoft

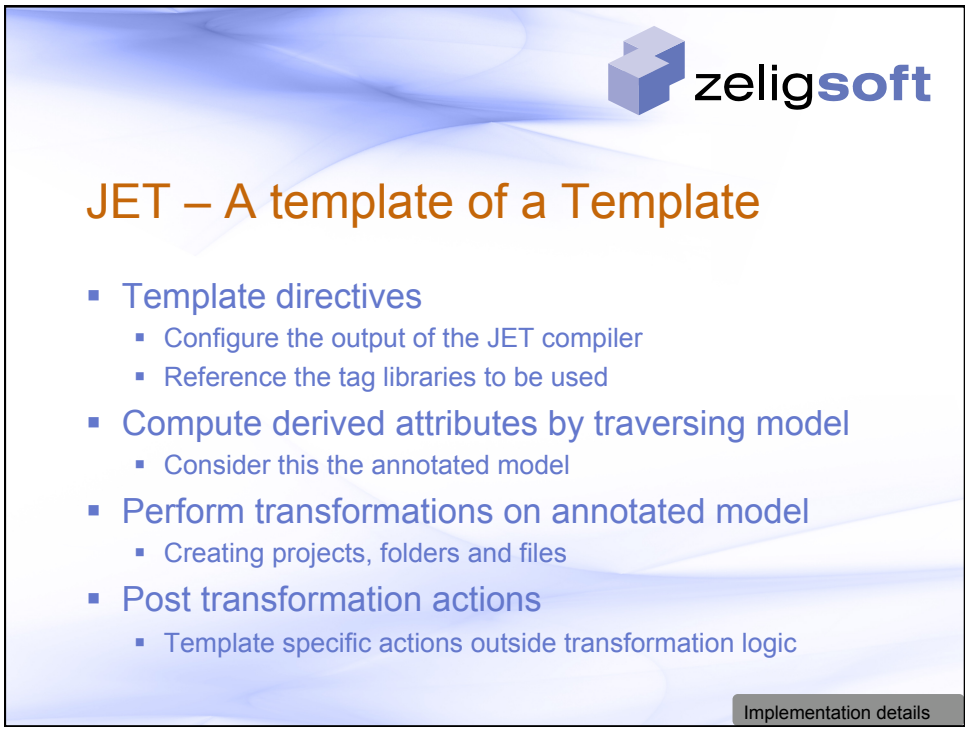
JET – Loading Models

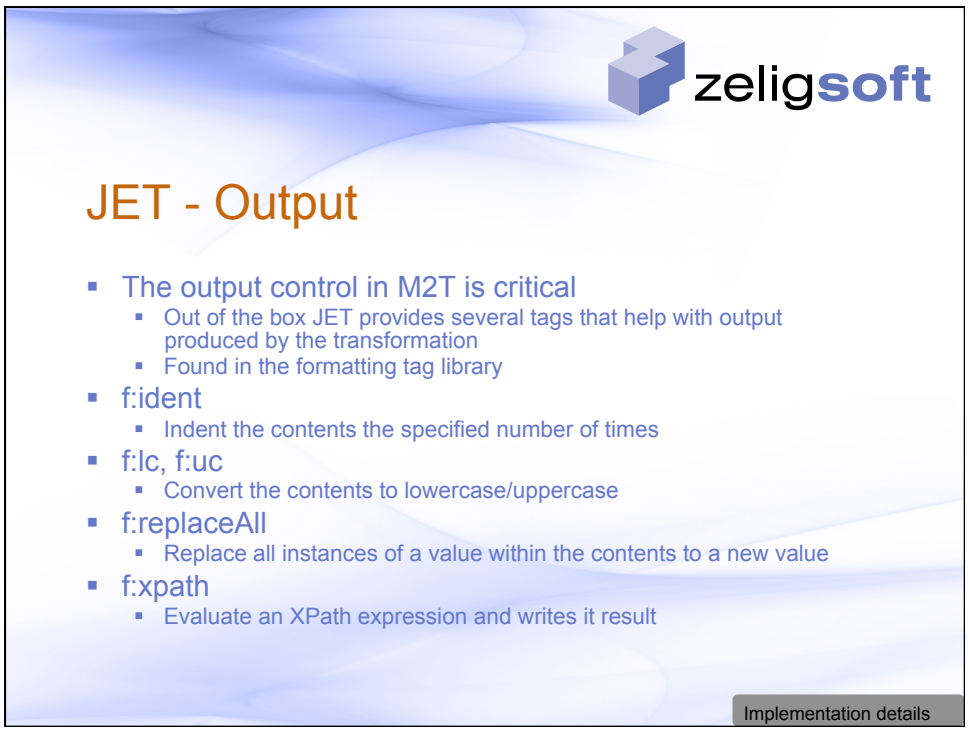
- Tags in JET used to load content
  - Load the content into a variable so it can be referenced
- c:load
  - Loads the referenced model into a specified variable
  - Can be referenced through the variable after that
- c:loadContent
  - Load the content of the tag into the specified variable as XML

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

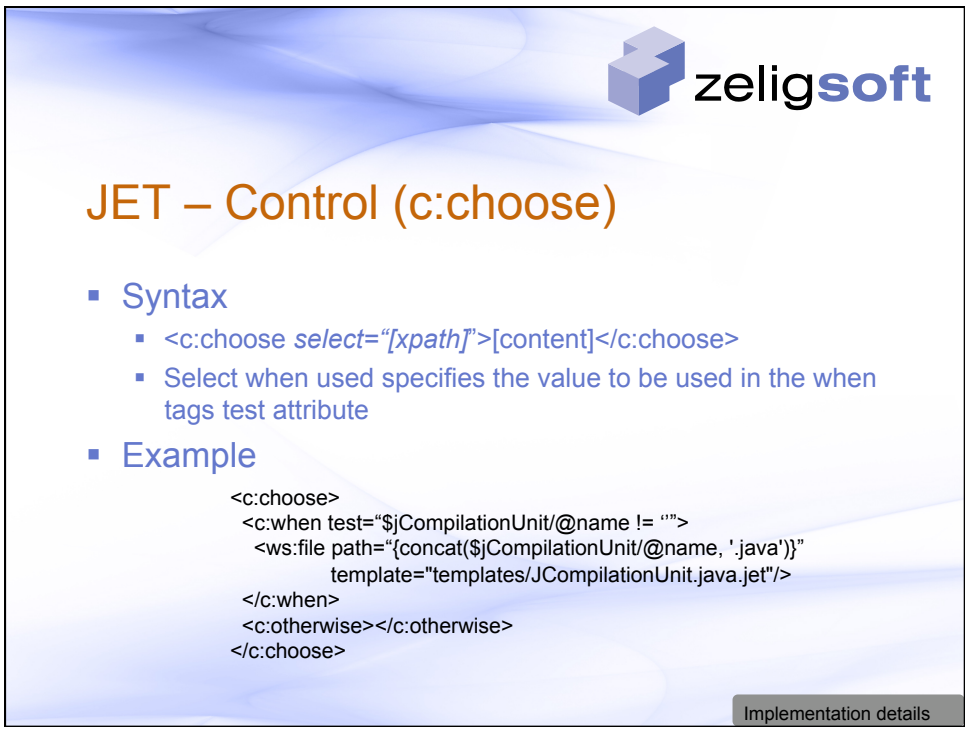

zeligsoft
JET - Templates
A template in JET is defined in a file
There is a 1 to 1 mapping between file and template
Directives configure the template
@jet
affects the code created by the JET compiler
package
class
imports
startTag
endTag
@taglib
imports a tag library for use in the template and assigns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

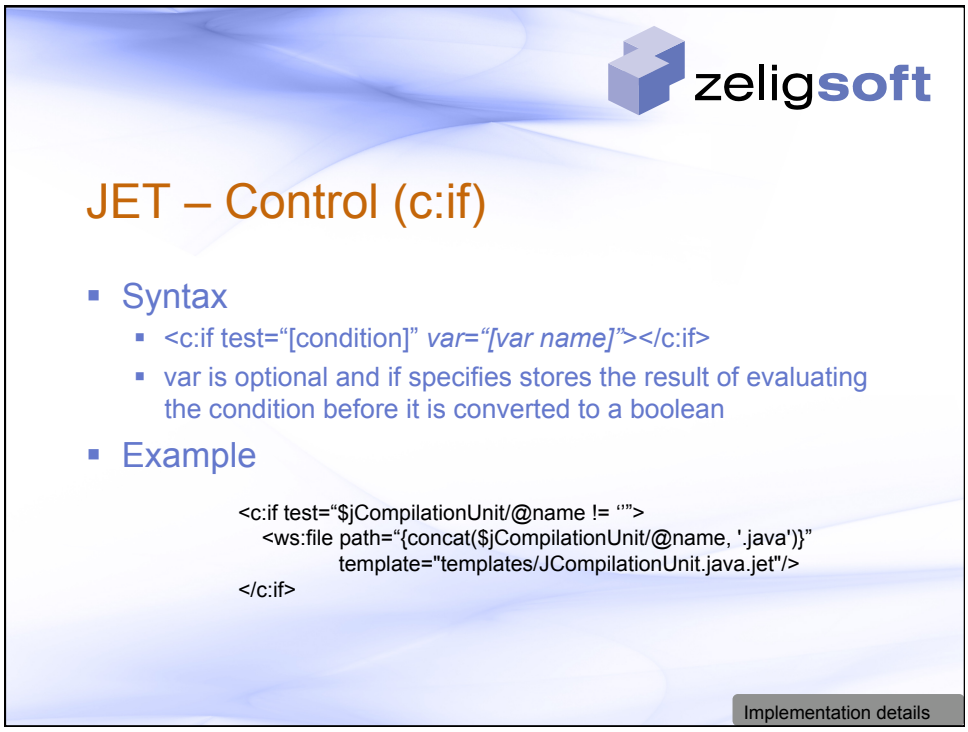


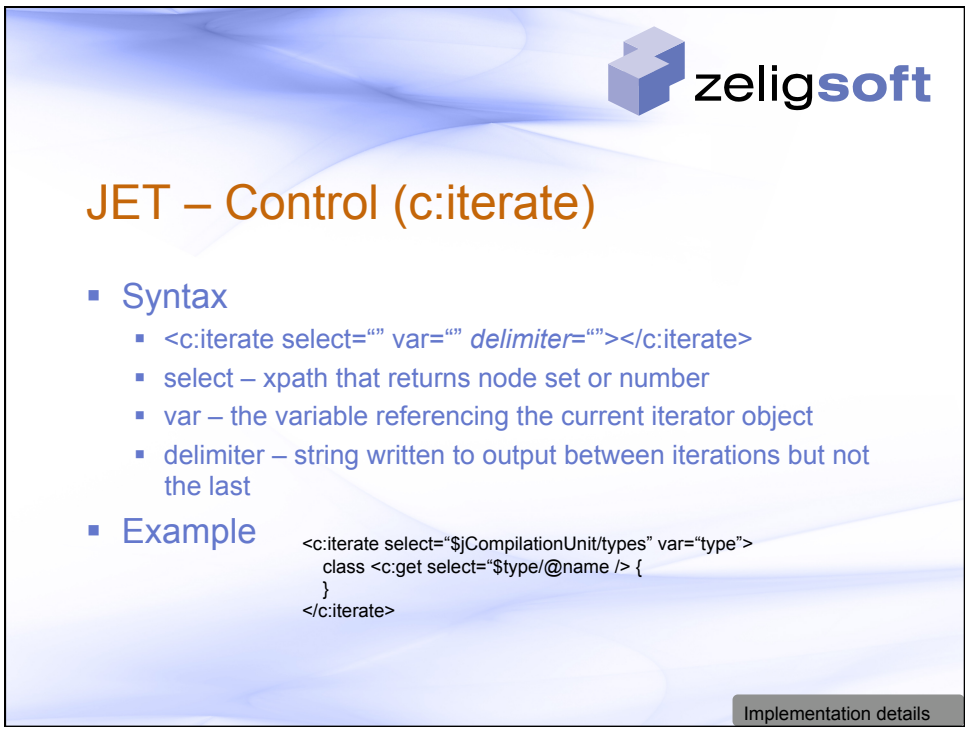


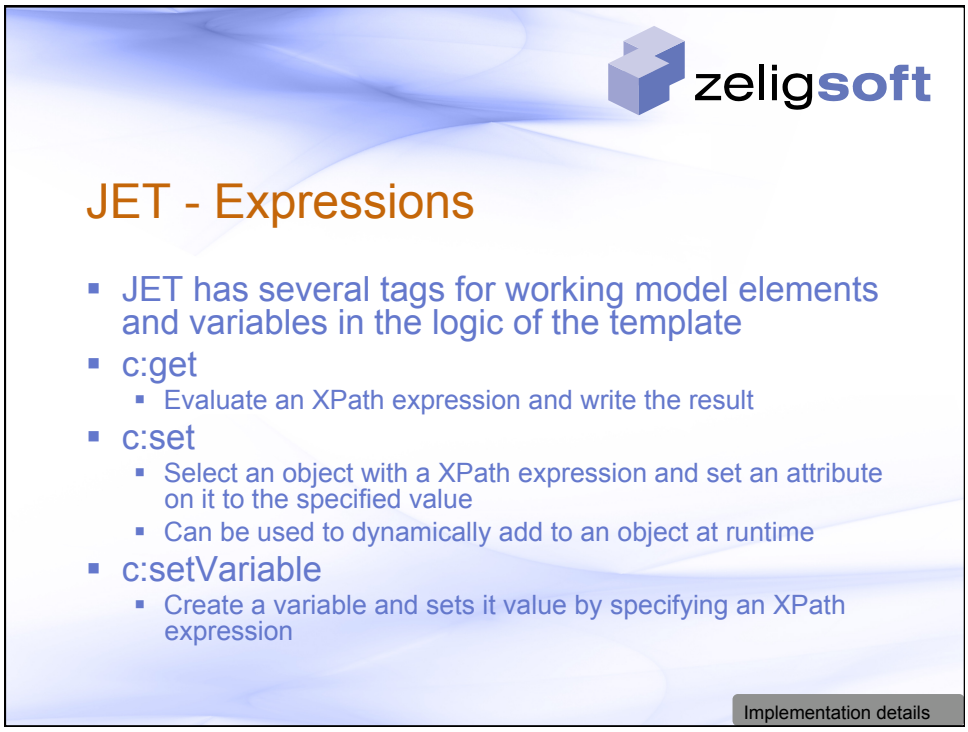
Toby McClean, Zeligsoft, 2008

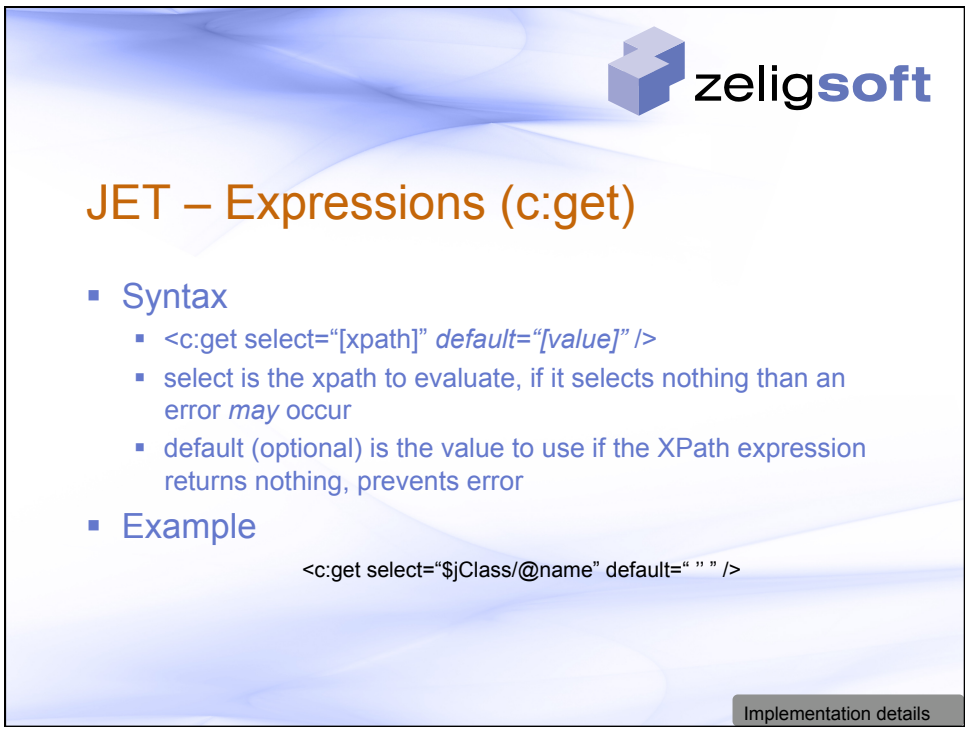

zeligsoft
JET - Control
The control tag library provides capabilities for putting control logic into your template
c:choose
A group of mutually exclusive choices
c:if
Only process the contents if a test condition is satisfied
c:iterate
Process the contents for each element specified by an XPath expression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

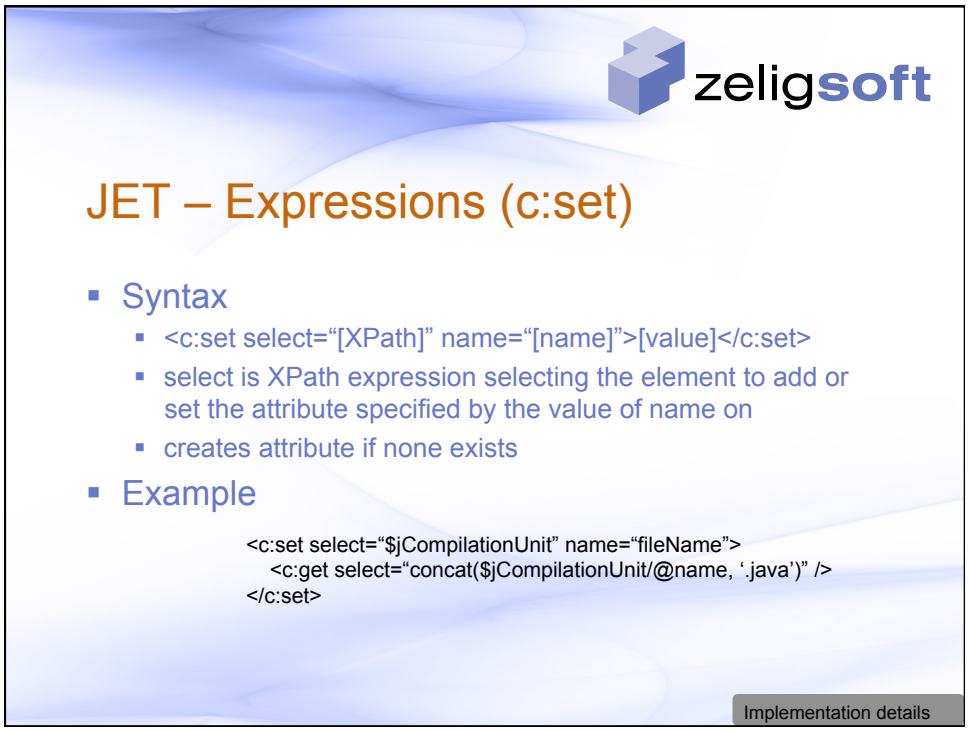

zeligsoft
JET – Control (c:choose)
Syntax
<c:choose select="[xpath]">[content]</c:choose>
Select when used specifies the value to be used in the when tags test attribute
Example
<c:choose>
<c:when test="$jCompilationUnit/@name != ''">
<ws:file path="{concat($jCompilationUnit/@name, '.java')}"
template="templates/JCompilationUnit.java.jet"/>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

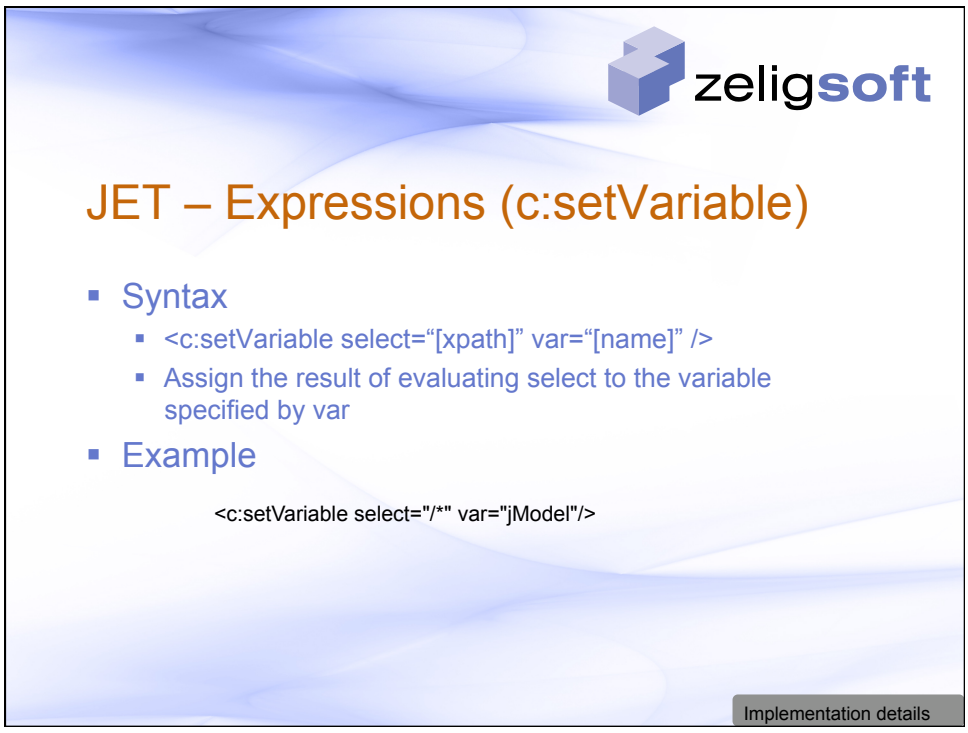

zeligsoft
JET – Control (c:if)
Syntax
<c:if test="[condition]" var="[var name]"></c:if>
var is optional and if specifies stores the result of evaluating the condition before it is converted to a boolean
Example
<c:if test="$jCompilationUnit/@name != ''">
<ws:file path="{concat($jCompilationUnit/@name,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

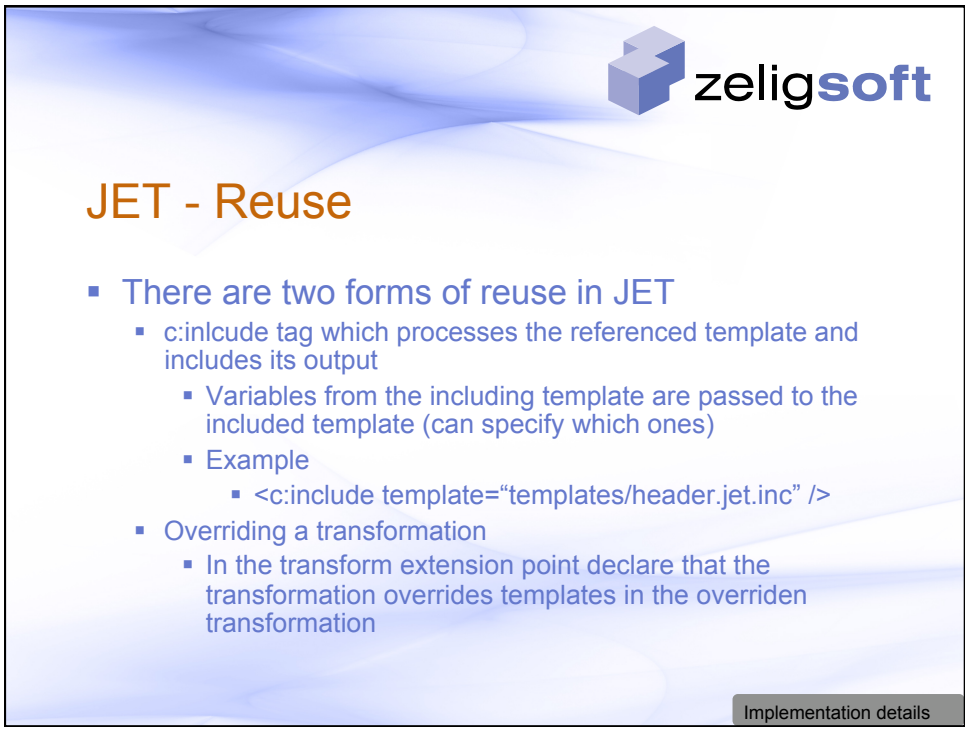

zeligsoft
JET – Control (c:iterate)
Syntax
<c:iterate select="" var="" delimiter=""></c:iterate>
select – xpath that returns node set or number
var – the variable referencing the current iterator object
delimiter – string written to output between iterations but not the last
Example
<c:iterate select="$jCompilationUnit/types"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

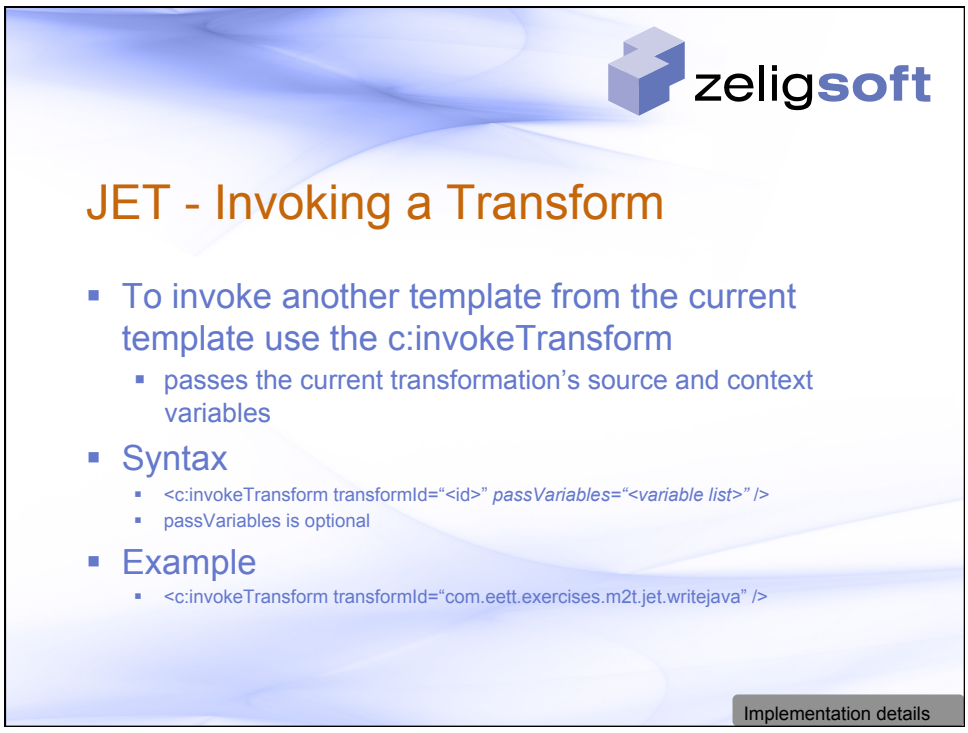

zeligsoft
JET - Expressions
JET has several tags for working model elements and variables in the logic of the template
c:get
Evaluate an XPath expression and write the result
c:set
Select an object with a XPath expression and set an attribute on it to the specified value
Can be used to dynamically add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

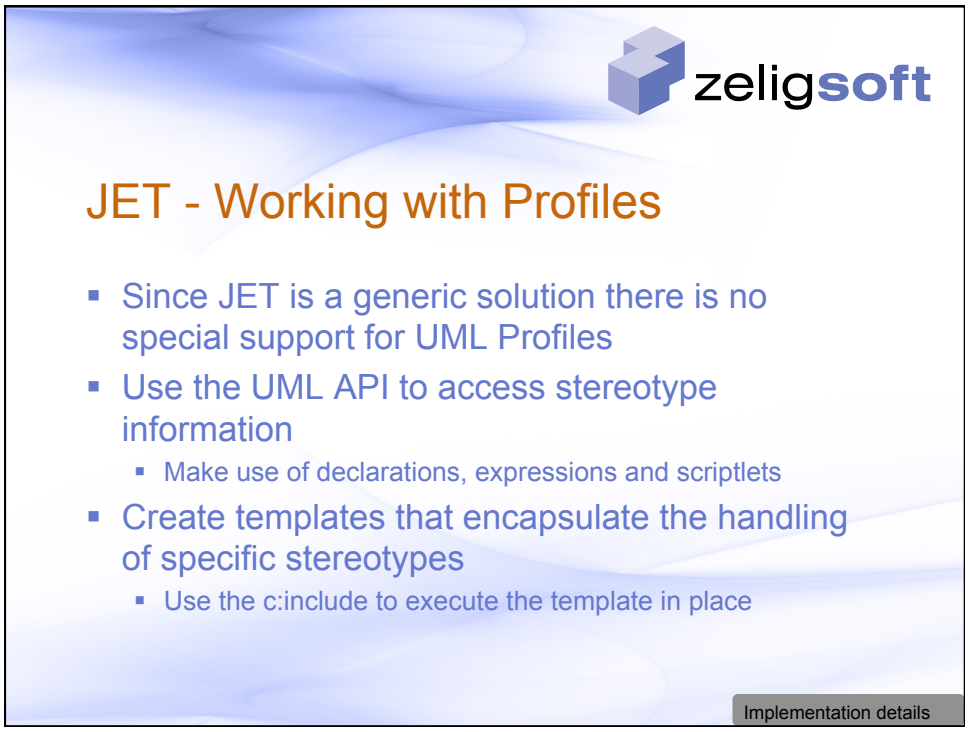

zeligsoft
JET – Expressions (c:get)
Syntax
<c:get select="[xpath]" default="[value]" />
select is the xpath to evaluate, if it selects nothing than an error may occur
default (optional) is the value to use if the XPath expression returns nothing, prevents error
Example
<c:get select="$jClass/@name" default="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

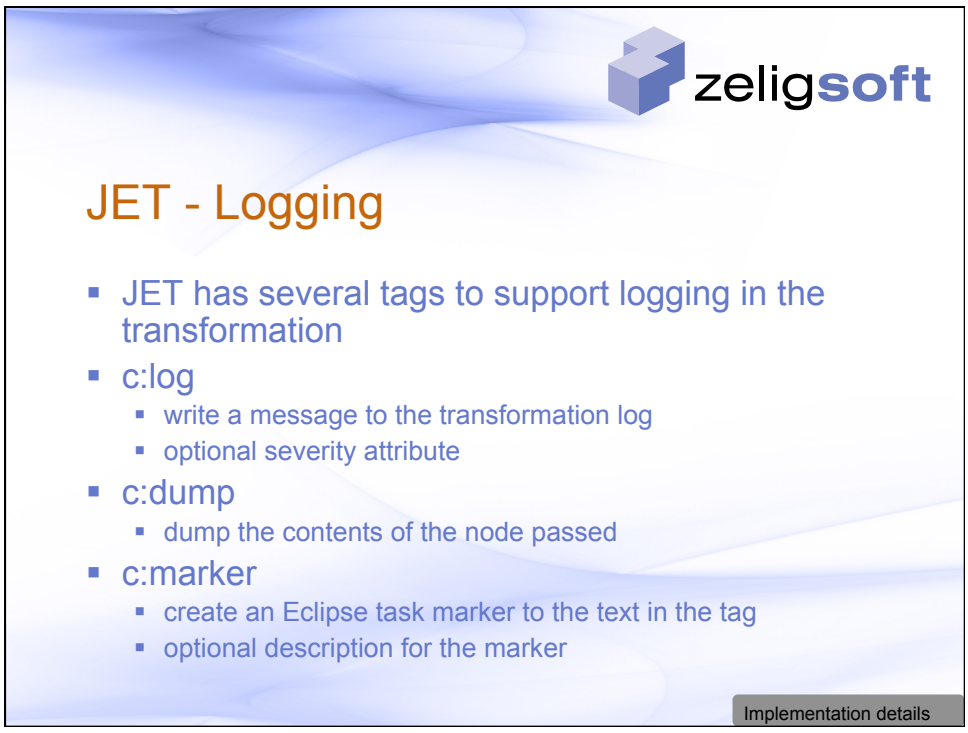

zeligsoft
JET – Expressions (c:set)
Syntax
<c:set select="[XPath]" name="[name]">[value]</c:set>
select is XPath expression selecting the element to add or set the attribute specified by the value of name on
creates attribute if none exists
Example
<c:set select="$jCompilationUnit" name="fileName">
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

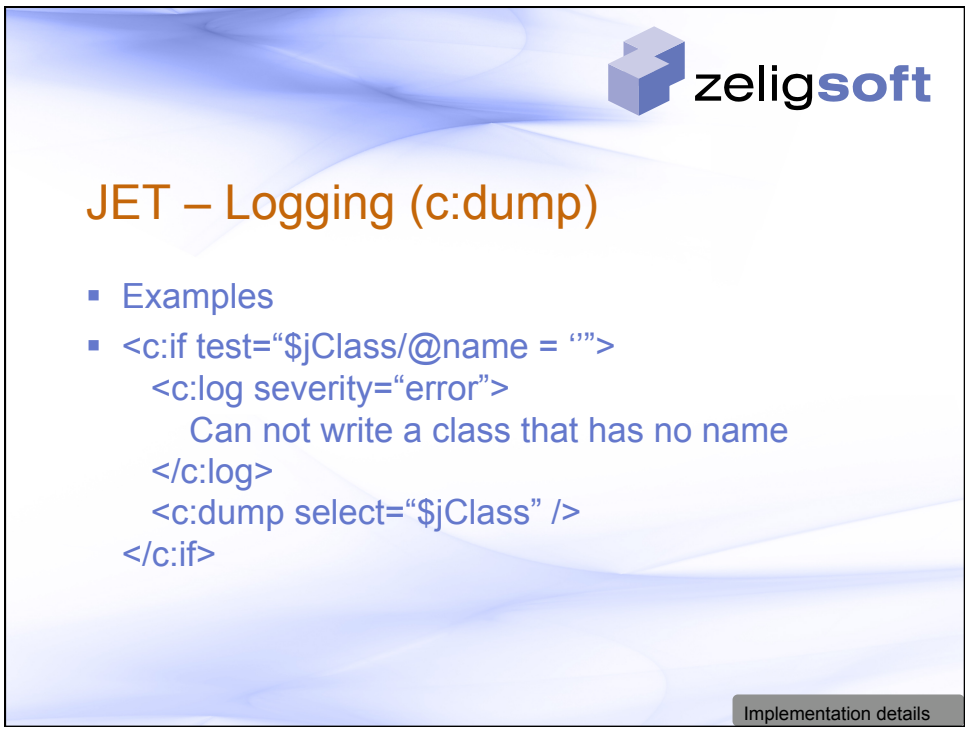

zeligsoft
JET - Reuse
There are two forms of reuse in JET
c:inlcude tag which processes the referenced template and includes its output
Variables from the including template are passed to the included template (can specify which ones)
Example
<c:include template="templates/header.jet.inc" />
Overriding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

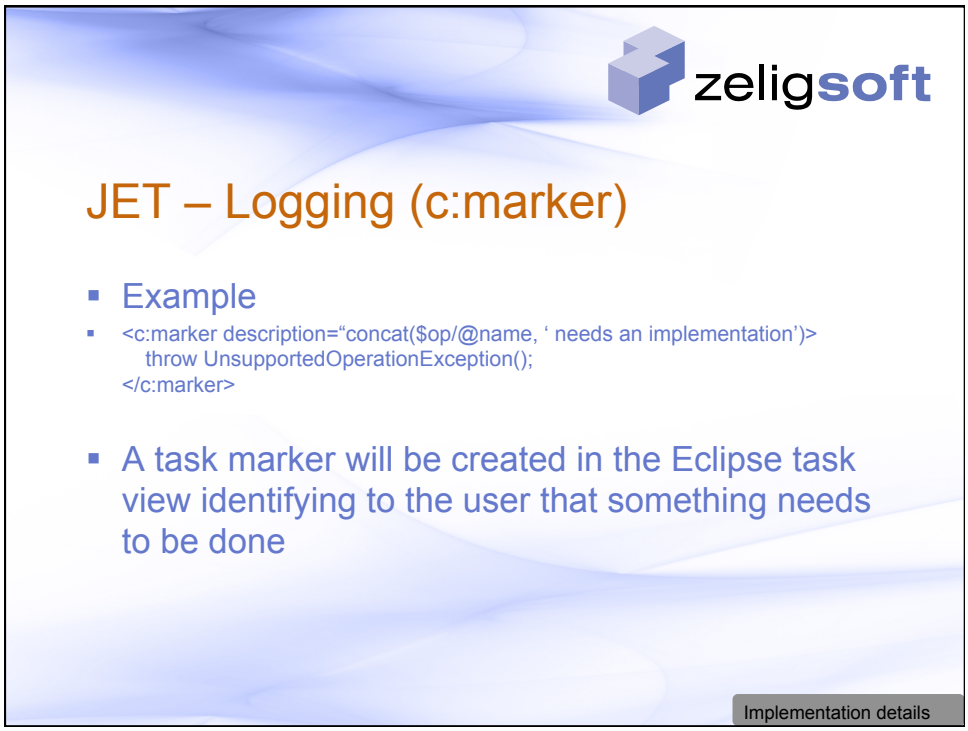

zeligsoft
JET - Invoking a Transform
To invoke another template from the current template use the c:invokeTransform
passes the current transformation's source and context variables
Syntax
<c:invokeTransform transformId="<id>" passVariables="<variable list>" />
passVariables is optional
Example
<c:invokeTransform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

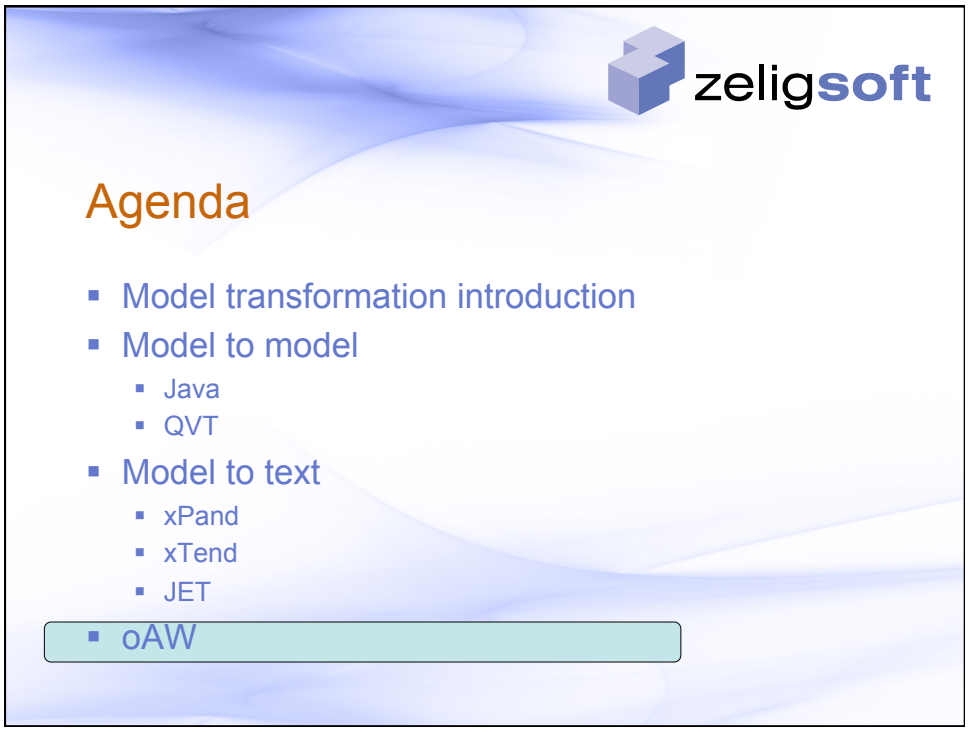

zeligsoft
JET - Logging
JET has several tags to support logging in the transformation
c:log
write a message to the transformation log
optional severity attribute
c:dump
dump the contents of the node passed
c:marker
create an Eclipse task marker to the text in the tag
optional description for the marker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

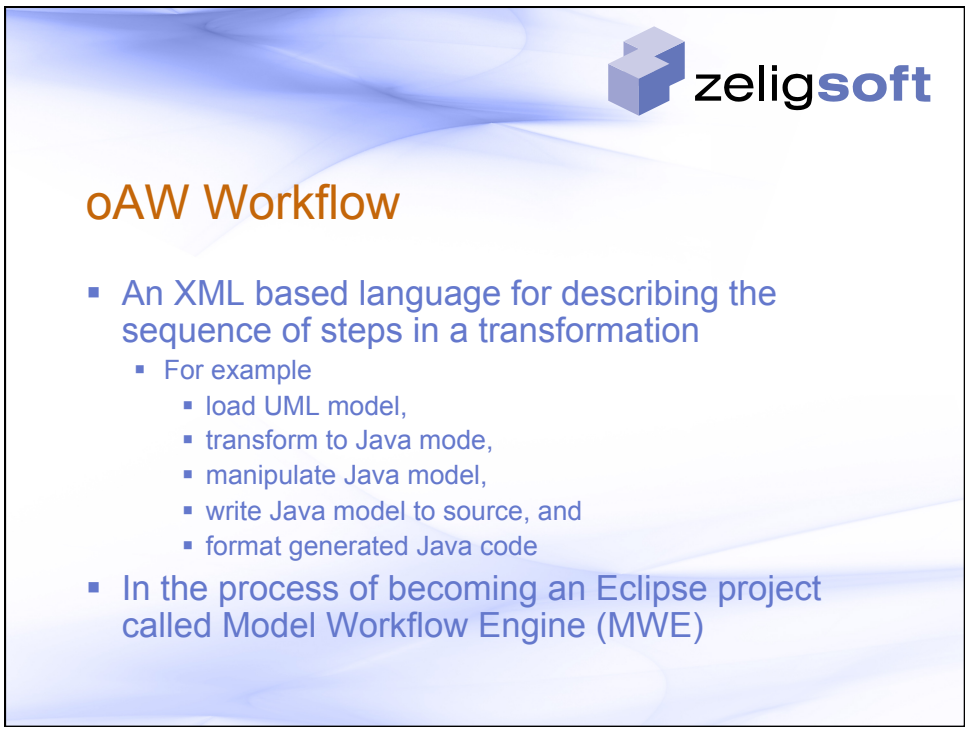

zeligsoft
JET – Logging (c:log)
Examples
<c:if test="$jClass/@name = ''">
<c:log severity="error">
Can not write a class that has no name
</c:log>
</c:if>
<c:log>
Writing <c:get select="$jClass/@name" />
</c:log>
Implementation details

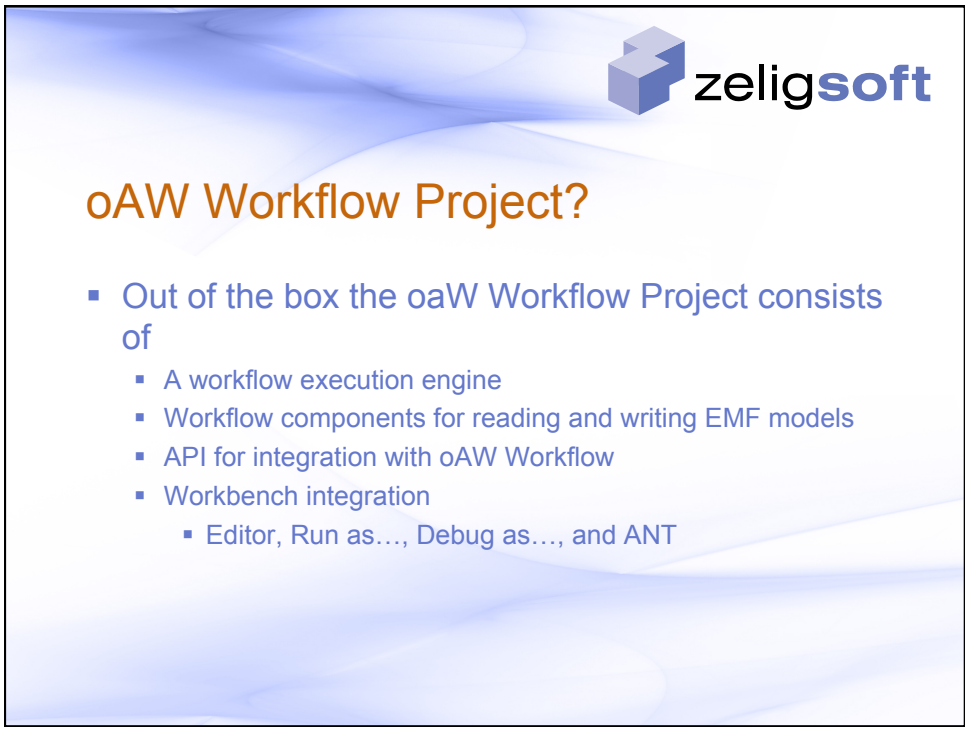

zeligsoft
JET – Logging (c:dump)
Examples
<c:if test="$jClass/@name = ''">
<c:log severity="error">
Can not write a class that has no name
</c:log>
<c:dump select="$jClass" />
</c:if>
Implementation details


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)

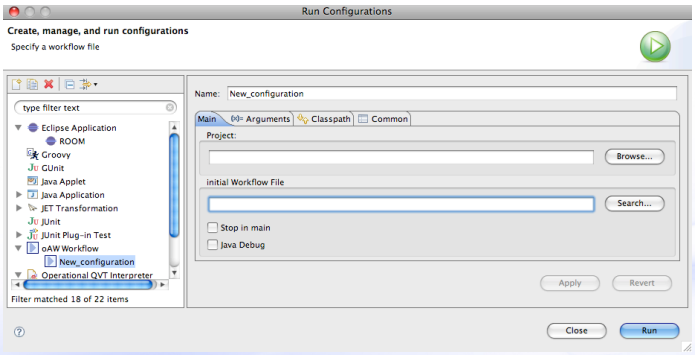

zeligsoft
oAW Workflow Project?
Out of the box the oaW Workflow Project consists of
A workflow execution engine
Workflow components for reading and writing EMF models
API for integration with oAW Workflow
Workbench integration
Editor, Run as…, Debug as…, and ANT


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