Usage of oAW within KIELER Projects

Xpand/Xtend Tutorial

Christian Motika

Real-Time Systems and Embedded Systems Group
Department of Computer Science
Christian-Albrechts-Universität zu Kiel, Germany

07/02/2009

Overview

- oAW plug-in
  - Basics
  - oAW nature

Overview

- oAW plug-in
  - Basics
  - oAW nature
- Xpand M2T

Overview

- oAW plug-in
  - Basics
  - oAW nature
- Xpand M2T
- Xtend M2M
  - Inplace transformation
  - Java escape

Overview

- oAW plug-in
  - Basics
  - oAW nature
- Xpand M2T
- Xtend M2M
  - Inplace transformation
  - Java escape
- Workflow

oAW basics

- Install from update site:
  - http://www.openarchitectureware.org/updatesite/milestone/site.xml

oAW basics

- Install from update site:
  - http://www.openarchitectureware.org/updatesite/milestone/site.xml
- Generator:
  - Independent of Eclipse

oAW basics

- Install from update site:
  - http://www.openarchitectureware.org/updatesite/milestone/site.xml
- Generator:
  - Independent of Eclipse
  - Run from command line

oAW basics

- Install from update site:
  - http://www.openarchitectureware.org/updatesite/milestone/site.xml
- Generator:
  - Independent of Eclipse
  - Run from command line
  - Run from workbench

oAW basics

- Install from update site:
  - http://www.openarchitectureware.org/updatesite/milestone/site.xml
- Generator:
  - Independent of Eclipse
  - Run from command line
  - Run from workbench
  - Run from Java program / plug-in

oAW basics

- Install from update site:
  - http://www.openarchitectureware.org/updatesite/milestone/site.xml
- Generator:
  - Independent of Eclipse
  - Run from command line
  - Run from workbench
  - Run from Java program / plug-in
- Arbitrary Metamodel (extendable)

oAW basics

- Install from update site:
  - http://www.openarchitectureware.org/updatesite/milestone/site.xml
- Generator:
  - Independent of Eclipse
  - Run from command line
  - Run from workbench
  - Run from Java program / plug-in
- Arbitrary Metamodel (extendable)
- All oAW languages: Common expression language

oAW languages

- Useful in different contexts of MDSD process

oAW languages

- Useful in different contexts of MDSD process
  - Xtend → Model transformations (M2M)

oAW languages

- Useful in different contexts of MDSD process
  - Xtend → Model transformations (M2M)
  - Xpand2 → Code generation (M2T)

oAW languages

- Useful in different contexts of MDSD process
  - Xtend → Model transformations (M2M)
  - Xpand2 → Code generation (M2T)
  - Xtext → Textual editor generation

oAW languages

- Useful in different contexts of MDSD process
  - Xtend → Model transformations (M2M)
  - Xpand2 → Code generation (M2T)
  - Xtext → Textual editor generation
  - Check → Model validation

Type system and expression language

- Common expression language and type system

Type system and expression language

- Common expression language and type system
- Model types:
  - Collections, Lists, Sets (`Collection[my::Type]`, `List[my::Type]`, `Set[my::Type]`)
  - Properties, Operations, Enums/constants

Type system and expression language

- Common expression language and type system
- Model types:
  - Collections, Lists, Sets (`Collection[my::Type]`, `List[my::Type]`, `Set[my::Type]`)
  - Properties, Operations, Enums/constants
- Build-in types:
  - Void
  - Simple types (String, Boolean, Integer)
  - Collection types

Type system and expression language (cont'd)

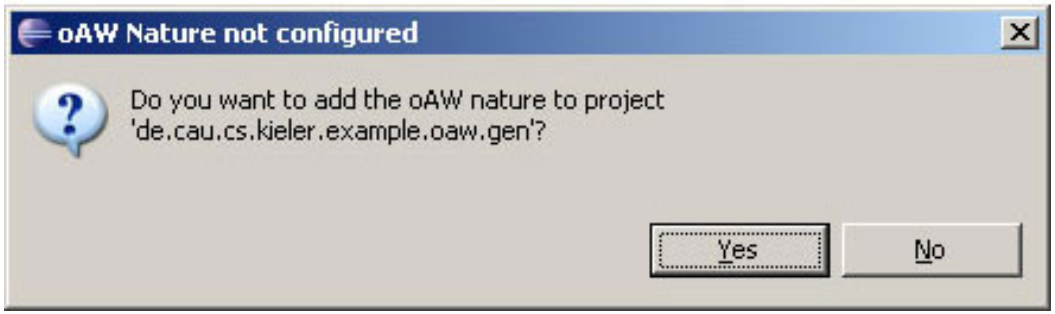
- Expressions:
  - Arithmetic, boolean (`1+2*3`, `!(true && true)`)
  - Operators: $==$, $!=$, $<$, ...
  - Strings: `''this is a string''`
  - Integer, Real: `10`, `3.4`
  - Collection operations: `collection.select(i|i >3)`, ...

Type system and expression language (cont'd)

- Expressions:
  - Arithmetic, boolean (`1+2*3`, `!(true && true)`)
  - Operators: $==$, $!=$, $<$, ...
  - Strings: `''this is a string''`
  - Integer, Real: `10`, `3.4`
  - Collection operations: `collection.select(i|i >3)`, ...
- Details here:
  - http://www.openarchitectureware.org/pub/documentation/4.3.1/html/contents/core_reference.html

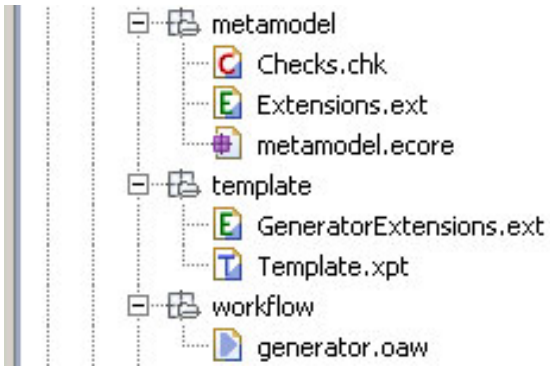
oAW nature

- oAW nature available



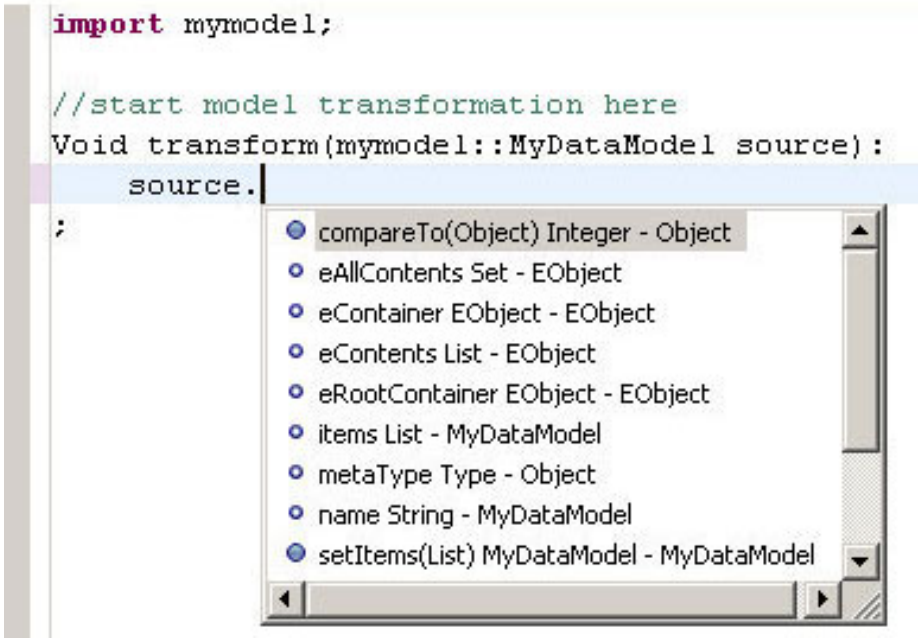
oAW nature

- oAW nature available
  - File decorations (*.oaw, *.xpt, *.ext, *.chk)



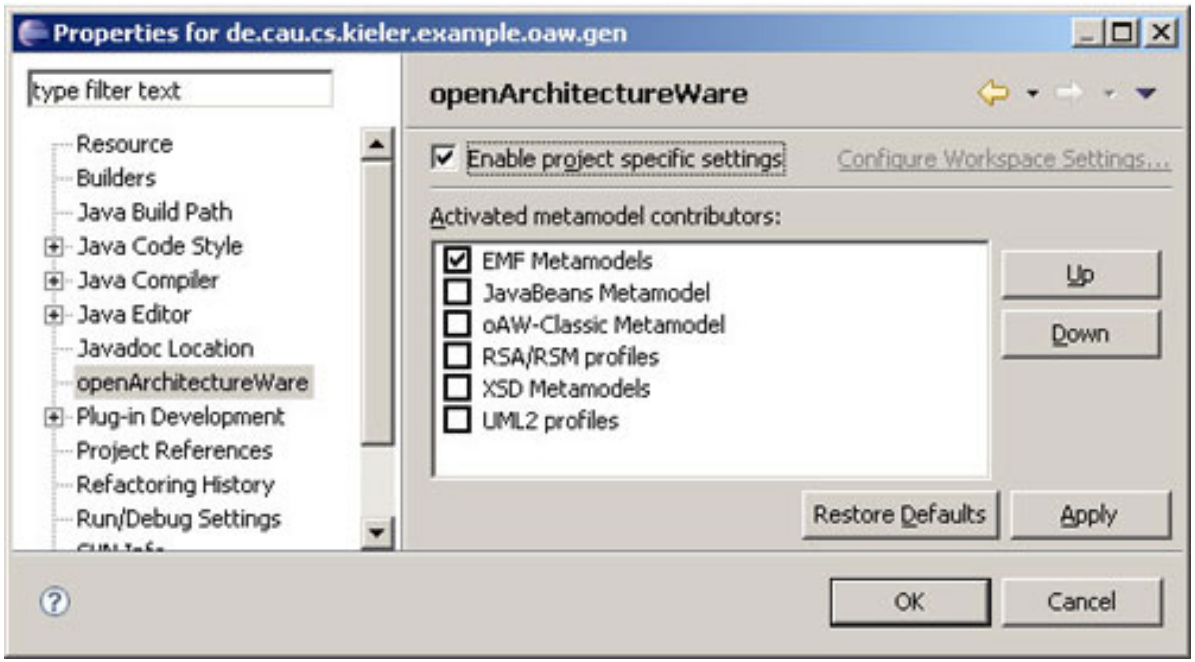
oAW nature

- oAW nature available
  - File decorations (*.oaw, *.xpt, *.ext, *.chk)
  - Syntax coloring and code completion



oAW nature

- oAW nature available
  - File decorations (*.oaw, *.xpt, *.ext, *.chk)
  - Syntax coloring and code completion
  - Meta model awareness



Xpand

- Template language

Xpand

- Template language
- Generating text (e.g., code) out of arbitrary models

Xpand

- Template language
- Generating text (e.g., code) out of arbitrary models
- Escape sequence <*shift*> + <*control*> + "<"
  $\longrightarrow \ll$ xpand commands $\gg$

Xpand

- Template language
- Generating text (e.g., code) out of arbitrary models
- Escape sequence $<shift>$ + $<control>$ + "<"
  $\longrightarrow \ll$ xpand commands $\gg$
- Use Xtend for additional functionality (s.b.)

Xpand

- Template language
- Generating text (e.g., code) out of arbitrary models
- Escape sequence $<shift>$ + $<control>$ + "<"
  $\longrightarrow \ll$ xpand commands $\gg$
- Use Xtend for additional functionality (s.b.)
- Java escape possible

Xpand example file

```
1  «IMPORT mymodel»
2
3  «DEFINE main FOR MyDataModel-»
4     «FILE "My.c"-»
5
6  /* some c comment */
7
8  #include <stdio.h>
9
10 «FOREACH items AS n ITERATOR i-»
11    «IF n.metaType.name.matches("mymodel::MyString") -»
12          String myString«i.counter1-» = "«((MyString)n).text»";
13             printf("String : %s", myString«i.counter1-»);
14    «ENDIF-»
15 «ENDFOREACH-»
16
17 «ENDFILE-»
18 «ENDDEFINE»
```

- Hint: Use ”-≫” to suppress white space!

Xtend

- Functional language

Xtend

- Functional language
- Can be used to define extensions on meta model types

```
attributes(Entity this) :
features.typeSelect(Attribute);
```

Xtend

- Functional language
- Can be used to define extensions on meta model types

```
attributes(Entity this) :
features.typeSelect(Attribute);
```

- Extensions can be used within the Xpand language

Xtend

- Functional language
- Can be used to define extensions on meta model types

```
attributes(Entity this) :
features.typeSelect(Attribute);
```

- Extensions can be used within the Xpand language
- Became ”full” transformation language (in oAW 4.0)
  - New keywords `create` and `cache`

Xtend

- Functional language
- Can be used to define extensions on meta model types

```
attributes(Entity this) :
features.typeSelect(Attribute);
```

- Extensions can be used within the Xpand language
- Became ”full” transformation language (in oAW 4.0)
  - New keywords `create` and `cache`
- Can be escaped to Java code (e.g., for iterative code)

Xtend example file

```
1  import sourcemetamodel;
2  import targetmetamodel;
3
4  //create a new TargetModelType and PropertyType
5  create TargetModelType this transform(sourcemetamodel::MyModel source):
6      let propety = new PropertyType:
7      property.setName("some name") ->
8      property.setValue("some value") ->
9      this.propertylist.add(property)
10 ;
11
12 //iterating over a list
13 Void createItems(BaseEntityType baseEntity,
14         List[sourcemetamodel::MyData] myDataList) :
15    let current = myDataList.last():
16    baseEntity.values.add(currentData.value) ->
17    if (myDataList.size > 1) then
18       createItems(baseEntity, myDataList.withoutLast())
19 ;
```

Escape to Java

```
1  //oAW xtend *.ext file
2
3  Void dump(String s) :
4     JAVA package.XtendJavaClass.dump(java.lang.String)
5  ;
6
7  String hasValue(sourcemodel::MyData item) :
8     JAVA package.XtendJavaClass.hasValue(sourcemodel.MyData)
9  ;
```

Escape to Java

```
1  //oAW xtend *.ext file
2
3  Void dump(String s) :
4     JAVA package.XtendJavaClass.dump(java.lang.String)
5  ;
6
7  String hasValue(sourcemodel::MyData item) :
8     JAVA package.XtendJavaClass.hasValue(sourcemodel.MyData)
9  ;
```

```
1  //*.java file
2
3  package de.cau.cs.kieler.example.oaw.gen;
4  import sourcemodel.*;   //import EMF generated model code!
5
6  public class XtendJavaClass {
7     public final static void dump(String aString) {
8        System.out.println(aString);
9     }
10
11    public final static boolean hasValue(MyData item) {
12       return (item.values.size > 0);
13    }
14 }
```

Inplace transformation

- Just (actively) modify your ”source” model ...

```
1  import mymodel;
2
3  Void transform(mymodel::MyDataModel source):
4      source.setName("CHANGED NAME :-)")
5  ;
```

Inplace transformation

- Just (actively) modify your ”source” model ...

```
1  import mymodel;
2
3  Void transform(mymodel::MyDataModel source):
4      source.setName("CHANGED NAME :-)")
5  ;
```

- ... and save this (source) slot!

```
1  <component class="org.eclipse.mwe.emf.Writer">
2     <uri value="${source}" />
3     <modelSlot value="source"/>
4  </component>
```

Code generation

1. Read in (source) model

Code generation

1. Read in (source) model
2. Make meta models available to generator

Code generation

1. Read in (source) model
2. Make meta models available to generator
3. Perform transformation or generation

Code generation

1. Read in (source) model
2. Make meta models available to generator
3. Perform transformation or generation
4. Write out (target) model/code

Code generation

1. Read in (source) model
2. Make meta models available to generator
3. Perform transformation or generation
4. Write out (target) model/code

- $\longrightarrow$ *Workflow* defines these steps

Code generation

1. Read in (source) model
2. Make meta models available to generator
3. Perform transformation or generation
4. Write out (target) model/code

- $\longrightarrow$ *Workflow* defines these steps
- Readers/Writers for EMF/XML Schema/Java (meta) models available

Code generation

1. Read in (source) model
2. Make meta models available to generator
3. Perform transformation or generation
4. Write out (target) model/code

- $\longrightarrow$ *Workflow* defines these steps
- Readers/Writers for EMF/XML Schema/Java (meta) models available
- Extendable: Own workflow components (this is quite easy!)

Defining a workflow

```
1  <?xml version="1.0"?>
2  <workflow>
3     <!-- define properties -->
4
5     <!-- load model -->
6
7     <!-- register meta model(s) -->
8
9     <!-- generate code or transform model /> -->
10
11    <!-- write model -->
12 </workflow>
```

Properties

```
1  <property name="srcmetamodel"
2      value="platform:/resource/.../sourcemetamodel.ecore" />
3
4  <property name="dstmetamodel"
5      value="platform:/resource/.../targetmetamodel.ecore" />
6
7  <property name="source"
8      value="platform:/resource/.../My.sourcetype" />
9
10 <property name="target"
11     value="src-gen/generated.targettype" />
```

- Can be modified from within Java program

Read model

```
1  <component class="org.eclipse.mwe.emf.Reader">
2
3      <uri value="${source}" />
4
5      <modelSlot value="source" />
6
7  </component>
```

- Other model readers available

Read model

```
1  <component class="org.eclipse.mwe.emf.Reader">
2
3      <uri value="${source}" />
4
5      <modelSlot value="source" />
6
7  </component>
```

- Other model readers available
- Own workflow component possible

Transform (Xtend)

```
1  <component class="oaw.xtend.XtendComponent">
2
3     <metaModel class="oaw.type.emf.EmfMetaModel">
4        <metaModelPackage value="sourcemodel.SourcemodelPackage"/>
5     </metaModel>
6
7     <metaModel class="oaw.type.emf.EmfMetaModel">
8        <metaModelPackage value="targetmodel.TargetmodelPackage" />
9     </metaModel>
10
11    <invoke value="XtendFileName::transform(source)" />
12
13    <outputSlot value="target"/>
14
15 </component>
```

- Also register meta models here

Transform (Xtend)

```
1  <component class="oaw.xtend.XtendComponent">
2
3     <metaModel class="oaw.type.emf.EmfMetaModel">
4        <metaModelPackage value="sourcemodel.SourcemodelPackage"/>
5     </metaModel>
6
7     <metaModel class="oaw.type.emf.EmfMetaModel">
8        <metaModelPackage value="targetmodel.TargetmodelPackage" />
9     </metaModel>
10
11    <invoke value="XtendFileName::transform(source)" />
12
13    <outputSlot value="target"/>
14
15 </component>
```

- Also register meta models here
- Declare by package or ecore filename (s.b.)

Generate (XPand)

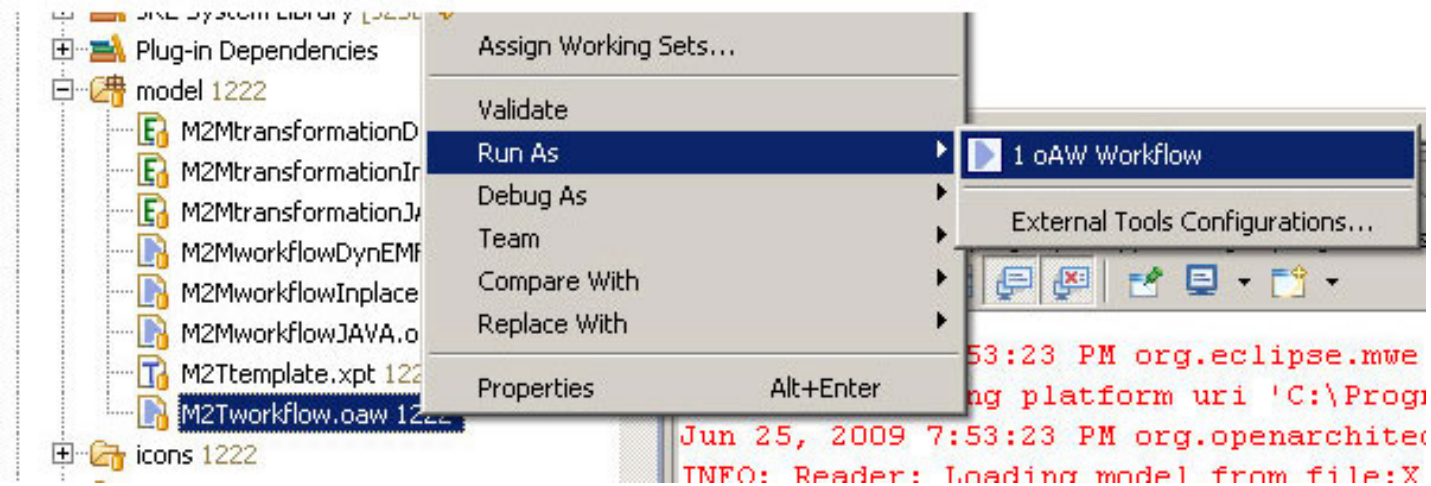
```
1  <component class="org.openarchitectureware.xpand2.Generator">
2
3     <metaModel class="oaw.type.emf.EmfMetaModel">
4        <metaModelFile value="${sourcemetamodel}" />
5     </metaModel>
6
7     <metaModel class='org.eclipse.m2t.type.emf.EmfRegistryMetaModel' />
8        <expand
9           value="SubDir::TemplateFileName::main FOR model" />
10       <outlet path="${src-gen}" />
11
12 </component>
```

Write model

```
1  <component class="org.eclipse.mwe.emf.Writer">
2
3     <uri value="${target}" />
4
5     <modelSlot value="target"/>
6
7  </component>
```

How to run a workflow file?

1. Start from Eclipse *Workbench*!



How to run a workflow file?

1. Start from Eclipse *Workbench*!
2. Start from command line! `java`

```
org.openarchitectureware.workflow.WorkflowRunner
path/workflow.oaw
```

How to run a workflow file?

1. Start from Eclipse *Workbench*!
2. Start from command line!
3. Start from Java program/plug-in!

```
1  Map<String,String> properties = new HashMap<String,String>();
2  Map<String, Object> slotContents = new HashMap<String, Object>();
3
4  String WorkflowFile = "myWorkflow.oaw";
5  properties.put("sourcemodel", PluginRoot + "My.mymodel");
6  properties.put("metamodel", PluginRoot + "model/sourcmetamodel.ecore");
7  properties.put("src-gen", GenFolder);
8
9  boolean success = false;
10 try {
11      WorkflowRunner runner = new WorkflowRunner();
12      success = runner.run(WorkflowFile, null,
13                           properties, slotContents);
14 } finally {}
```

How to run a workflow file?

1. Start from Eclipse *Workbench*!
2. Start from command line!
3. Start from Java program/plug-in!
4. Start from Ant task

To go further

1. www.openarchitectureware.org
2. www.openarchitectureware.org/forum
3. www.ibm.com/developerworks/library/os-eclipse-dynamicemf/
4. oAW example project
5. SVN: /trunk/common/examples/program-s/de.cau.cs.kieler.example.oaw
6. blog.efftinge.de/2006/04/model2model-transformation-with-xtend_15.html

Thank you for your attention and participation!

Any questions or suggestions?