anyware TECHNOLOGIES

Ecore – oAW – Android

Using EMF to generate an Android Application

Olivier Prouvost
Benjamin Cabé

Anyware Technologies

olivier.prouvost@anyware-tech.com
benjamin.cabe@anyware-tech.com

Agenda

- Presentation
- Technologies
  - EMF/ECore and Editors
  - Android
- Tutorial Description
- Steps and exercises
- Conclusion

Presentation

- Anyware Technologies
  - French Software Company based in Toulouse
  - Specialized in Eclipse Technology
  - http://www.anyware-tech.com
- Olivier Prouvost
  - Eclipse consultant and architect
  - Involved in new technology projects
- Benjamin Cabé
  - Software engineer
  - Eclipse expert

Technologies and prerequisites

Tutorial Prerequisites

- Technologies
  - Eclipse 3.3.1.1
  - EMF 2.3.1
  - openArchitectureWare 4.2.0
  - Android 0.4.0 and its plugin
- Material
  - plugin emf/android

EMF / Modeling

- Eclipse top level project providing :
  - Meta Model language (Ecore)
  - Ecore editors (tree, graphical)
  - EMF model editor generators
  - Tools for model validation (OCL, Check...)
  - Generation platforms (JET, OAW).
  - ...
- EMF is used by lots of other Eclipse Projects
  - BIRT, WTP, TPTP, ...

emf
ECLIPSE MODELING FRAMEWORK

openArchitectureWare (oAW)

- A subset of Modeling project
- Dedicated to model generation :
  - generation language (Xpand)
  - generation scheduler (workflow)
- Fully integrated with Eclipse environment
  - perspective, editors
  - debugger for Xpand and workflows
  - bound to Ecore and EMF platform.

Android

- The new Google mobile platform
- Released in november 2007
- Java syntax / Dalvik VM
- Tools and runtime for mobiles :
  - Eclipse Integrated SDK
    - launch configuration, emulators, views.
  - Mobile Application Architecture
  - Libraries to develop mobile applications
    - Contacts, Messages,
    - Map views

Tutorial Description

Tutorial Description

- Goal : Create a model Editor using Android
- Ideas
  - EMF generates a model editor from an Ecore model
    - This editor is composed of 3 parts
    - We can reuse 2 of them
    - Use oAW to generate the Android specific layer
  - Extending Eclipse to create a dedicated IDE
    - sample plugin given in materials

Tutorial Material Plugins

- com.Eclipsecon08.androidtools
  - provides an action to add emf libraries in android project
  - defines the 'emfoid' nature
- com.Eclipsecon08.androidtools.generation
  - contains the oAW templates and workflows
  - provides action to generate the Android editor
    - call the EMF generation
    - call the oAW generation
- com.Eclipsecon08.androidtools.utils
  - defines tools for the Android file system (bug workaround)

Tutorial steps (1)

- Installation
  - Install Eclipse 3.3.1 and the tutorial plugins
- EMF part
  - Create the model project and eCore model
  - Generate the EMF default Editor
- Android part
  - Install the Android SDK
  - Create a default Android Application
  - Launch it
- Android/EMF Integration
  - "Emfoidization"

Tutorial steps (2)

- Android model Editor
  - Editor Architecture
  - Reusing EMF Layers
  - oAW Project
  - Editing model
- Conclusion
  - Technologies
  - Applying this tutorial to other contexts

EMF Part

EMF Part

- Create an empty EMF Project
- Create an Ecore model
- Create the genmodel file for editor generation
- Generate model editor
- Use model editor

EMF Part – Create EMF/ECore

- Create an empty EMF Project
  - org.Eclipsecon08.emfandroid.model
- Create an Ecore model
  - Eclass Person (name, function)
  - Eclass Task
    - name, description, start/end dates, responsable
  - Eclass Project
    - name, date, description, team (list of persons), manager
    - list of tasks
  - Eclass Company
    - managed projects

EMF Part – Generate editor

- Create the gen model file
  - set packages and generation parameters
  - set composition attributes
- Generate editor
- Use editor
  - launch Eclipse plugin
- Look at editor layers
  - model
  - edit
  - editor

Android Part

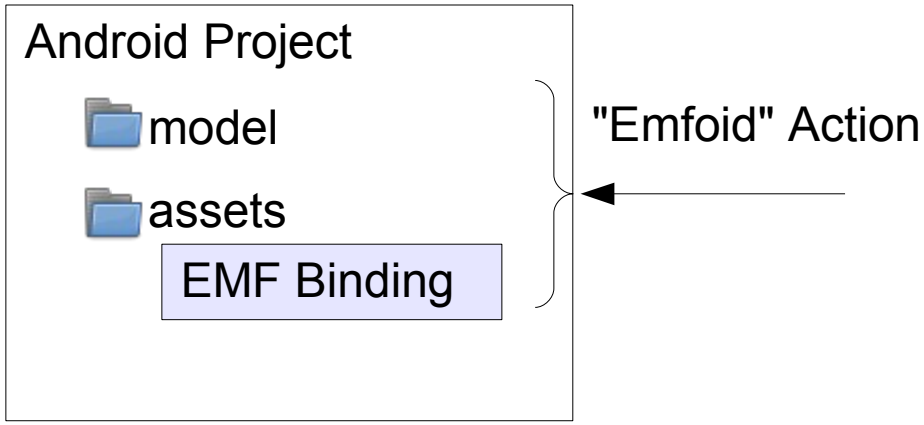
Android Part

- Android installation
  - Download the Android SDK
  - Install it on your PC
  - Download the Android SDK plugin
    - https://dl-ssl.google.com/android/Eclipse
  - Bind Eclipse to Android SDK.
- Android default application
  - Create a new Android project
  - Launch and check it

Android / EMF Integration Part

Tutorial plugin description

- Provides a 1st action on an Android project
  - binds an Android project to an EMF project
  - adds a model directory
  - set the "Emfoid" project nature
  - "Emfoidization"

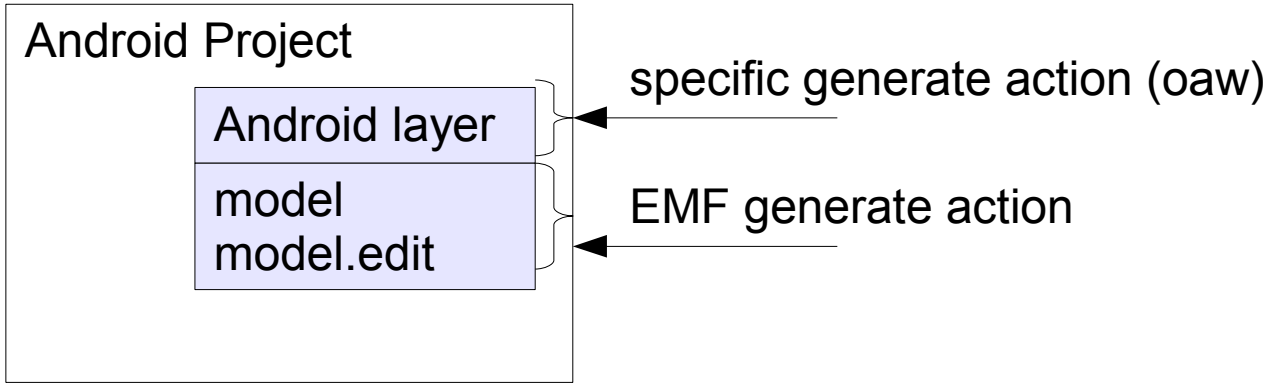


EMF Binding description

- Android/EMF application must use
  - the EMF jar files
  - XML jar files
- These files are
  - copied with the "Emfoid" action into assets
  - added to classpath

Tutorial plugin description

- Provides a 2nd action on a .genmodel file
  - generates the model and model.edit layers
    - calls the default EMF generation action
  - generates the Android screens
    - calls the oAW generators



Android/EMF project

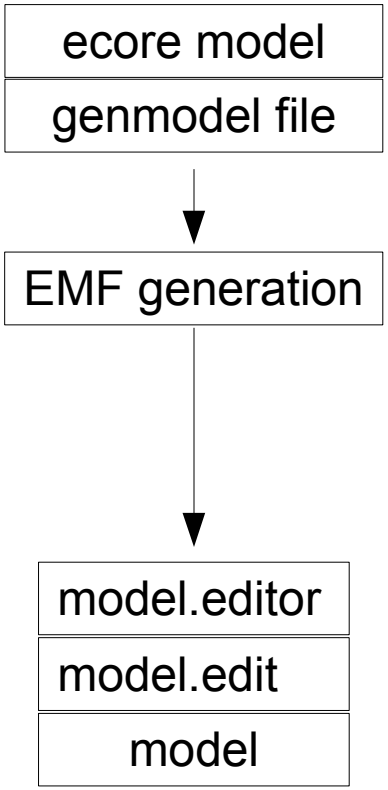
- With the plugin, the Android project
  - keeps its Android nature
    - can be run as an Android project
  - uses the EMF libraries
  - contains its own model directory
  - can be filled with EMF generated code

Android Architecture

- Activity
  - screen of an application
  - GUI described in an XML file
- Intent
  - action to be done
  - matched by IntentFilter
- ContentProvider
  - the Android storage system
- Adapters
  - displays structured data into widgets

Android model editor design

EMF basic editor



The editor specific framework

The editor common framework

The java object model

Model framework

- Generated by 'generate model code'
- Contains :
  - the java source code for all eCore objects
    - interface
    - implementation
  - factory to instantiate these objects
  - helper classes
    - to select an object (switch)
    - to navigate on model with reflective API
- Can be used in our Android project

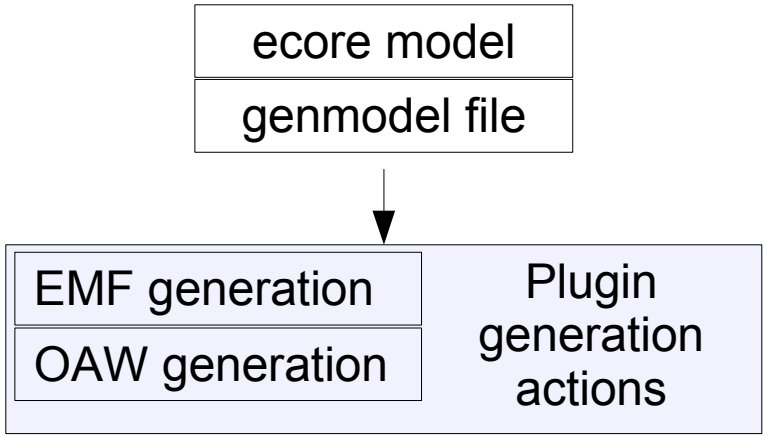
model.edit framework

- contains common code for all editors
- the objects label providers
  - to describe a model object
  - getText : compute the text for an object
  - getImage : compute icon for an object
- these methods may be overriden
  - add 'NOT' in generated annotation
- can be used in our Android project

model.editor framework

- a plugin extending org.Eclipse.ui.editors
- specific to Eclipse ui
- provides several views for the same editor:
  - in a tree
  - in a table
  - in a tree table
- can not be used directly in our Android project
  - this part will be generated

Android/Emf Editor



The android specific editor

The editor common framework

The java object model

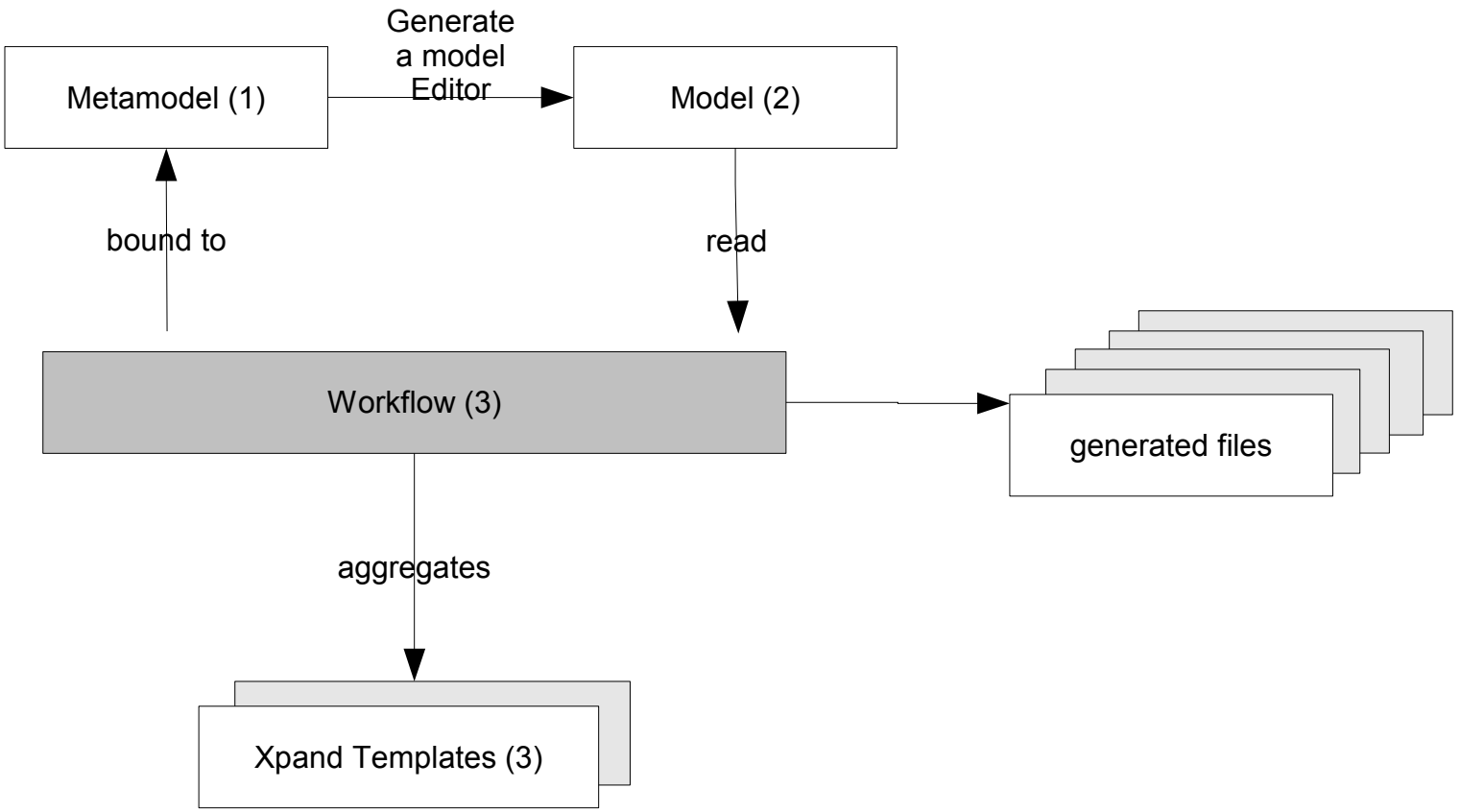
Tutorial plugin execution

Tutorial Editor

- Copy the Ecore model into project
- Generate the Android Editor
- Use it
- Explore com.eclipsecon08.androidtools.generation
  - look at xpt files
  - add the date support for EDate type

Generation part openArchitectureWare

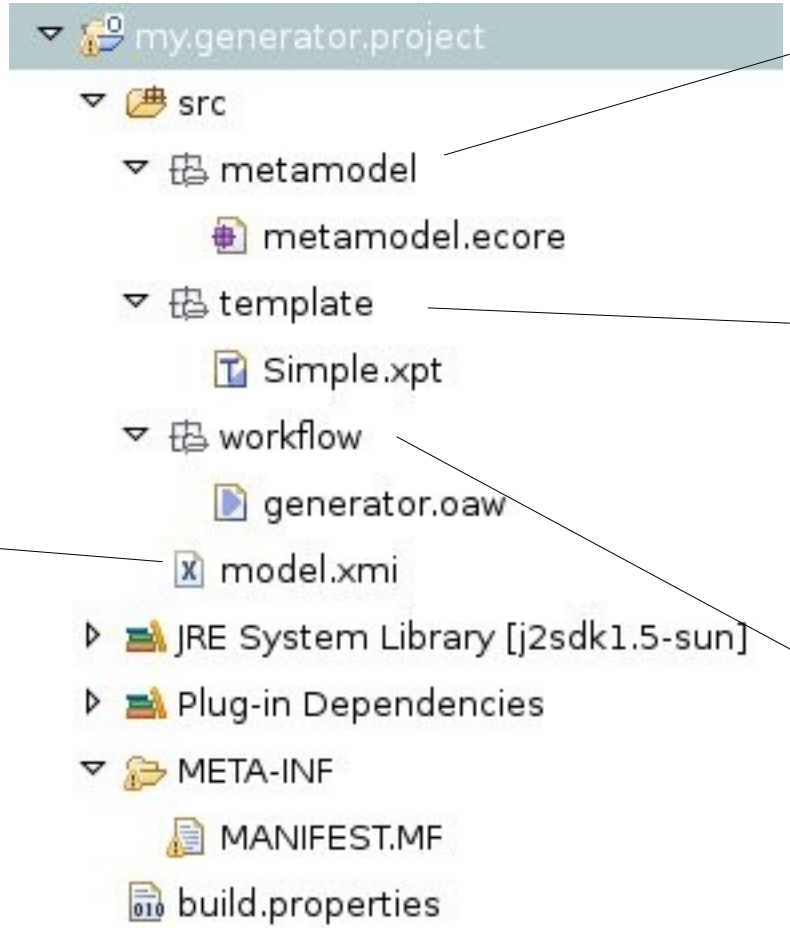
oAW Principle.



oAW Project

- Specific project type
  - new wizard (template available)
  - associated actions
- Specific directories
  - logic view of the project
- Specific launch configuration ("Run as oAW Workflow")
- Specific components
  - model reader
  - beautifiers (xml, java)
- Specific tools : debugger, editors.

oAW Project contents



The metamodel directory

The template directory for templates and extensions

The model instance

The workflow directory for workflows

Xpand template language

Template Sample

```
1«IMPORT metamodel»
2
3«DEFINE javaClass FOR Entity»
4    «FILE name+".java"»
5        public class «name» {
6            «FOREACH attributes AS attr»
7                public «attr.type» «attr.name»;
8            «ENDFOREACH»
9
10           «FOREACH references AS ref»
11               public «ref.type.name» «ref.name»;
12           «ENDFOREACH»
13       }
14   «ENDFILE»
15«ENDDEFINE»
```

Xpand specific characters

- The specific tag delimiter character '«' and '»'
  - ctrl '<' or ctrl '>'
- Double colon '::'
  - Delimiter for namespaces a::b::myType

Xpand Contents

- The type System
  - each object managed in OAW has a Type.
  - access to all types defined in your metamodel
  - access to the built-in types defined in Xpand
- The expression sub language
  - Defines the syntax
  - Use the type system

Xpand type system

Xpand Type system

- Access to :
  - simple type : String
  - qualified type names : a::b::theType
- Defines collection types :
  - Collection[a::type], List[a::Type], Set[a::Type]
- Builtin types
  - String, Real, Integer...

Xpand expressions

- Access to a property : '.'
  - myElement.name : get the name of myElement instance
- Access to an operation : '.'
  - myElement.doStuff() : run this operation
- Arithmetic :
  - 1+ (5*myElement.size())
- Boolean :
  - !(myElement.name.startsWith('test') && myElement.ok)

Collection operations (1)

- Several operations are available on collection type.
- select
  - extract a sublist according to criteria
  - myCol.select(v | boolean expression with v)
  - {1,2,3,4}.select(i | i >= 3 } // returns {3,4 }
- typeSelect
  - extract objects of specified class (also inherited)
  - myList.typeSelect(a::b::myClass)
- reject
  - same as select, but return object that do not match

Collection operations (2)

- collect
  - evaluate an expression on elements and return list
  - myElements.collect(elt | elt.name)
  - shorthand : myElements.employee.name
- forAll
  - evaluate a boolean for all objects in collection
  - myCollection.forAll(v | boolean expression with v)
  - true if all values are true
- exists
  - evaluate if one boolean expression is true for one elt
  - myCollection.exists(v | boolean expression with v)

Structural expressions

- if
  - condition ? expression : elseExpression
  - myElement.name != null ? myElement.name : 'unknown'
- switch
  - switch(exp)
    - { case expression : thenExpression
    - default : defaultExpression }
  - switch (myElement.name)

```
{
    case 'apple' : 'fruit'
    default  : 'unknown'
}
```

Xpand language details

Xpand language

- The template file is composed of DEFINE blocks
- Define blocks contain
  - literal and expressions
  - text to be generated
  - all literal or expression to be evaluated is inserted between «»

«DEFINE»

- The main and basic tag of Xpand
- Defines a template for an object model instance (this)
- The template will be applied to each instance of classname (subclasses included)

```
52«DEFINE displayValue FOR FieldEntity»
53        displayValue.append(«EXPAND fieldUtil::getterName FOR this-»())
54«ENDDEFINE»
55
56«DEFINE displayValue FOR DatesField»
57        displayValue.append(new java.text.SimpleDateFormat().format(«EX
58«ENDDEFINE»
59
```

«FILE»

- Use to redirect output to a given file
- The contents of FILE/ENDFILE is generated
- The FILE tag is used in the root DEFINE of template

```
1«IMPORT core»
2
3«DEFINE generateBean(String packageName) FOR CardEntity»
4«FILE name +".java"»
5
6package «packageName»;
7import com.anwrt.medany.server.util.AbstractMedanyBean;
8public class «name» extends AbstractMedanyBean
9{
10
11}  // end class «name»
12
13«ENDFILE»
14«ENDDEFINE»
```

«EXPAND»

- Use to expand another DEFINE
- Example with 'FOR' :

```
32    «REM» On genere le display selon les mainFields «ENDREM»
33    public String getDisplayValue()
34    {
35        StringBuffer displayValue = new StringBuffer();
36        «FOREACH mainFields AS mainfield»
37        «EXPAND displayValue FOR mainfield-»
38        «ENDFOREACH»
39        return displayValue.toString();
40    }
```

«EXPAND»

- Example with 'FOREACH ' :

```
73«DEFINE fieldProperty FOR EnumField-»
74    «EXPAND enumValue(this.name) FOREACH enumValues-»
75«ENDDEFINE»
76
77«DEFINE enumValue(String fname) FOR EnumValue-»
78enum.«fname».«name»=«name»
79«ENDDEFINE»
```

«FOREACH»

- Use FOREACH to iterate on objects in a collection
- FOREACH can call any other tag
- FOREACH can contain tests to skip some elements
- FOREACH can define an iterator with 2 properties :
  - counter0 : iterator value starting at 0
  - counter1 : iterator value starting at 1

```
23«FOREACH groups.fields AS f ITERATOR it-»
24field.«it.counter1»=«f.name»
25«EXPAND fieldProperty FOR f-»
26«ENDFOREACH»
```

«PROTECT»

- PROTECT is used to protect code against override
- It generates a source code block with protection

```
public class Person {
/*PROTECTED REGION ID(Person) ENABLED START*/
   this pr is enabled, therefore the contents will always be
   preserved. If you want to get the default contents from the
   template you must remove the ENABLED keyword (or even remove
   the whole file :-))
/*PROTECTED REGION END*/
}
```

- Remove the ENABLED keyword to remove protection

«PROTECT»

- Syntax for PROTECT is :

```
«PROTECT CSTART expression CEND expression ID expression (DISABLE)?»
   a sequence of statements
«ENDPROTECT»
```

- CSTART and CEND are comment tags for the target language (for java : /* */)
- ID is used to generate a unique ID (by template/gen)
- DISABLE is optional

```
«PROTECT CSTART „/*" CEND „*/" ID ElementsUniqueID»
   here goes some content
«ENDPROTECT»
```

Other tips

- REM for comments

```
«REM»
   text comment
«ENDREM»
```

- ERROR to generate an error code

```
«ERROR expression»
```

- Controling line spaces
  - insert a minus before any closing bracket

Workflow

Principle

- Workflows are used to define generators
- A workflow contains Workflow Components :
  - model parsers, validators, beautifiers...
  - called with workflow properties

```
<workflow>
 <property name='genPath' value='/home/user/target'/>
 <property name='model' value='/home/user/model.xmi'/>
 <component class='oaw.emf.XmiReader'>
  <model value='${model}'/>
 </component>
 <component class='oaw.xpand2.Generator'>
  <outlet>
   <path value='${genPath}'/>
  </outlet>
 </component>
</workflow>
```

Workflow components

- XMI Reader
  - used to read the model
  - store model in slotname

```
<component class="org.openarchitectureware.emf.XmiReader">
    <metaModelFile value="${medanyMetaModel}" />
    <modelFile value="${model}" />
    <outputSlot value="p" />
    <firstElementOnly value="true" />
</component>
```

Workflow components

- Xpand generator
  - applies Xpand template on model
  - use slotname produced by XMIReader

```
<component class="org.openarchitectureware.xpand2.Generator">
    <fileEncoding value="iso-8859-1"/>
    <metaModel  class="org.openarchitectureware.type.emf.EmfMetaModel">
         <metaModelFile value="${medanyMetaModel}" />
    </metaModel>
    <expand value='template::hbm::generate("${packageName}") FOREACH p.entities' />
    <beautifier class="org.openarchitectureware.xpand2.output.JavaBeautifier" />
    <genPath value="${home-gen}/src/${packagePath}" />
</component>
```

Conclusion and questions

olivier.prouvost@anyware-tech.com
benjamin.cabe@anyware-tech.com

anyware TECHNOLOGIES

More Information

olivier.prouvost@anyware-tech.com
benjamin.cabe@anyware-tech.com

Eclipse Time 29th of May 2008 in Toulouse (France)

http://www.eclipsetime.org

eclipse Time

emf ECLIPSE MODELING FRAMEWORK

openArchitectureWare

android