|  |  |
| --- | --- |
| Author Signature: | Date: |

|  |  |
| --- | --- |
| Approved by: | Signature: |
| Institute: | Date: |

Keywords:

|  |  |
| --- | --- |
| Released by: | Signature: |
| Institute: | Date: |

Alma Common Software Component Code Generator asdasddasdasDDo

Software Documentation

Alexis Tejeda <alexis.tejeda@gmail.com>

UCN Universidad Católica del Norte, Chile

Change Record

| REVISION | DATE | AUTHOR | SECTIONS/PAGES AFFECTED |
| --- | --- | --- | --- |
| REMARKS | | |
| 1 | 2010-02-10 | Alexis Tejeda | All |
| Created | | |
| 2 | 2010-04-23 | Alexis Tejeda | All |
| Update the document | | |
| 3 | 2010-10-19 | Alexis Tejeda | All |
| Update the document with the new structure, architecture and stereotypes | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |
|  |  |  |  |
|  | | |

**Table of contents**

No se encuentran entradas de índice.

**List of figures**

Figure 1: Enumerations for stereotype tags 10

# Summary

Asd

# Introduction

## Purpose

The creation of components based over Alma Common Software (ACS), can be a very repetitive task, for example the configuration of the CDB, definition of the IDL files and the classes creation, always is pretty much the same development process. Here is when the Model Driven Development (MDD) becomes a powerfull tool for the code generation, in this case, based on defined templates for UML class diagrams, part of this work is already done in the thesis [1] of Nicolas Troncoso.

In order to improve development time, code refactoring and the software engineering (at pattern desings level) for the component development over ACS, will continue the work already done by Nicolas Troncoso proposed in his thesis [1], please check thesis in Referenced Documents section in this document.

The purpose of this document is to explain the new features, the new development,how the

generator works and how the generation ’generates’ the code from UML model/metamodel/meta-

metamodels, as well as providing guidelines for the future development and extensions of

the component code generator.

## Scope

The content of the document is for developers, software engineers and managers, to allow them to use in a easy and good way the component generator for ACS, to create in a fast way code ready for the implementation.

It is assumed that the reader has a good command in the creation of components for ACS and understand the functioning of ACS and knows Java OOP paradigm and UML model specifications.

## Glossary

**ACS** Alma Common Software.

**EMF** Eclipse Modeling Framework, a framework of Eclipse to create plugins, Eclipse Applications for code generation base in Model Driven.

**NC** Notification Channels of ACS.

**oAW** Open Architecture Ware, a free open architecture for code generation.

**XMI** XML Metadata Interchange, standard for exchanging metadata information via Extensible Markup Language (XML).

**UML** Unified Modeling Language, is a standardized general-purpose modeling language in the field of software engineering.

**MDD** Model Driven Development

**OOP** Object Oriented Programming

**Stereotype** asdasd

**EProfile** asdasd

## References

[1] ACS Component Code Generation Framework – Nicolas Troncoso Thesis

<https://csrg.inf.utfsm.cl/twiki4/pub/ACS/AlmaTheses/thesis-ntroncos09.pdf>

[2] Eclipse Modeling Framework Documents

<http://www.eclipse.org/modeling/emf/docs/>

[3] Agile Model Driven Development with UML 2

Cambridge University Press, 2004 ISBN#: 0-521-54018-6

[4] Open Architecture Ware Reference

<http://www.openarchitectureware.org/pub/documentation/4.3.1/>

[5] Catalog Of OMG Modeling And Metadata Specifications

<http://www.omg.org/technology/documents/modeling_spec_catalog.htm>

# Technology Overview

The generator is based in several technologies and is imperative to know how the generator was designed and his architecture for the code genration base in UML models.

## Eclipse EMF - Open Architecture Ware 5

The component code generator for Alma Common Software initially created before, was supported on Open Architecture Ware 4, since is a requirement have the code generator based

on Open Architecture Ware 5, the previous work was migrated to oaW 5, this means that the code generator is based now on Eclipse Modeling Framework (EMF), mainly in Xtend and Xpand subprojects, to know more about this, see the References section in this document.

EMF supports many types of models like Ecore (EMF native models), UML2 models, or in this case meta-metamodels in XMI2.0 format. Actually the generator support a XMI2.0 meta-metamodels, mostly of this models are created and exported from MagicDraw UML Diagram Tool, MagicDraw generate the necessary XMI files for later generate the code components to be implemented by the final user. Is important to specify that the models needs a profile file, which defines the stereotypes for use in the generator, where they differentiate the many characteristics or custom features in the UML model.

### Xtend/Xpand

Since owA was migrated to EMF, the code generator is based on a mixture of EMF subpro-

jects(Xpand/Xtend/Xtext), the EMF subprojects which are used:

**Xtend**: provides the possibility to define rich libraries of independent operations and non-invasive metamodel extensions based on either Java methods or oAW expressions. Those libraries can be referenced from all other textual languages, that are based on the expressions framework, in this case Xpand.

**Xpand**: A programming language which allow to define the templates of the generator and controll the output generation.

## Java

Since the project is supported by EMF, the generator was programmed in Java using the EMF API and librarys to process the UML model to generate the code, for this reason the generator can be executed in diferent platforms using the Java VM. The Java version used is java version "1.6.0\_21".

# Design Overview

## Overall Structure

Since the generator is strongly based in Eclipse Modeling Framework the code of the project was designed following a common EMF structure. This structure is organized by packages for the future extensions, migrations or integrations with other projects related to the code generation.

Some packages contains important non-java-files that controll the workflow of the model reading and code generation, also some of this files provide common functions developed to read UML features such tags, artifacts, suppliers, manifestation, etc.

### Workflow File Configuration

The workflow, is a EMF XML configuration file that controll the workflow of the generator, in which where configured the paths of UML exported models, output folder, templates used, Java code beautifiers which module to use, and the package name definition for the generated code.

The path of the model to generate, the generated code output folder path, the UML profile file path and the module to use are specified using dynamic workflow propertys by ${myVariable} sintax.

These properties are set by command line to the Java program, in which are set to the workflow file with a WorkflowRunner object using String HashMap (see the EMF reference in the References section in this document).

Also the templates to use in the generation are specified in the configuration file as ’Workflow Components’ in which every UML element of the model is processed by this components which the template using functions and xtend/xpand programming language decide if generate the code using the template.

The components of the workflow file must have defined :

* **Output path**: the output path for the generated file, this is specified by a global property in the workflow configuration as ${ouputFolderURI} property.
* **UML profile**: this is specified by a global property in the workflow configuration as ${profileFileURI} property, the generator only support maximum one profile file.
* **Template**: the template to use.
* **File encoding** : by default is UTF-8.
* **A Global Variable named *ACSPackage***: This is a global variable defined as ${acsPackage}, this is for choose the module or package of the model to use.

An example is the workflow component that generates the Makefile:

* 1. <!-- Makefile -->
  2. <component id="ACS\_Makefile" class="org.eclipse.xpand2.Generator" skipOnErrors="true">
  3. <fileEncoding value="UTF-8" />
  4. <metaModel idRef="default\_profile"/>
  5. <genPath value="${ouputFolderURI}"/>
  6. <expand value="templates::common::ACSMakefile::Root FOR model"/>
  7. <globalVarDef name="ACSPackage" value="'${acsPackage}'"/>
  8. </component>

1. Comment
2. Define the component id, the a internal EMF class for the template processing, skipOnErrors skip code generation when the preceding model verification finds errors or error in the workflow.
3. File encoding
4. The metamodel to use, is the profile to use that have the metamodels and stereotypes, this is defined only once in the workflow file with a dynamic property and has to be referenced by all components.
5. The ouput folder for the code generated by the template that is using the component.
6. Is the internal path of the template ACSMakefile in templates/common/ for the *model* which is the model file and is defined only once in the workflow file with a dynamic property and has to be referenced by all components.
7. By a property the module of the model to use is defined, and the ACSMakefile template would know which module to process.

The component definition in the workflow file will be always the same, only the template specification and the component id has to be differents.

In the project, exists three workflow files, a Java, C++ and Python. Python and C++ workflow files are ready to be implemented, for now only the Java workflow named *JavaWorkflow.mwe* is full implemented for the generator.

The workflow file is based in EMF, this means that the file presents many changes from his previous version.

### Template files

Template files controll the output code generation. Each element in the UML model is analyzed by the template file, the template check the stereotype and the type of the UML element and decide if the UML has to be generated as a speceific code. These files are based in the Xpand programming language [4].

The templates use a custom function to process in a clean way the model, these functions are developed as a xtend to be used by all templates file.

An example is the ACSComponent.xpt template file:

1. «**IMPORT** uml»
2. «**EXTENSION** templates::util::common»
3. «**EXTENSION** templates::util::java»
4. «**DEFINE** Root **FOR** uml::Model»
5. «**FOREACH** getEComponentsImpl(getPackage(**this**)) **AS** eElement -»
6. «**FILE** getPackage(**this**).name+'/src/'+getPackage(**this**).name+'/'+eElement.name+'.java'-»
7. package «getPackage(**this**).name»;
8. import java.util.logging.Level;
9. import java.util.logging.Logger;
10. import alma.ACS.ComponentStates;
11. import alma.acs.component.ComponentLifecycle;
12. import alma.acs.component.ComponentLifecycleException;
13. import alma.acs.container.ContainerServices;
14. import alma.ACSErrTypeCommon.CouldntPerformActionEx;
15. import alma.ACSErrTypeCommon.wrappers.AcsJCouldntPerformActionEx;
16. import alma.«getPackage(**this**).name».\*;
17. «**FOREACH** getEManifestations(eElement) **AS** eManifestationContainer-»
18. «**FOREACH** getESuppliers(eManifestationContainer) **AS** eManifestationClass-»
19. «**FOREACH** ((uml::Class) eManifestationClass).getAllImplementedInterfaces() **AS** eInterface-»
20. import alma.«getPackage(**this**).name».«eInterface.name»;
21. import alma.«getPackage(**this**).name».«eInterface.name»Operations;
22. «**ENDFOREACH**-»
23. «**ENDFOREACH**-»
24. «**ENDFOREACH**-»

This component is for generate the component class of a UML element defined as a component in the model file. All the code in blue, is static code.

1. Import the UML specifications this is a EMF native library.
2. Import the common xtend functions, this file is located in templates/util/common.ext and is defined only for this project.
3. Import the java functions, located in templates/util/java.ext, which have all the functions for generate java code.
4. Load all UML elements of the model file
5. Empty line.
6. Get all component implementations defined in the model by a stereotype using a custom function.
7. Creates the java file under the output folder, also creates a folder with the module name if not exists before.
8. If the module to generate is *MyModule* the generator will generate package *MyModule*;
9. Empty line.
10. to 18 is static code.
11. Same as line 9.
12. to 28, get the IDL interfaces used by the component implementation, this is for a specific model design.

All templates are encoded in UTF-8 by the use of ’guillimets’ «» and use the xpand programming language for the template coding. The example above is specific for the creation of the implementation of a ACS component using a specific model design.

### Xtend files and functions

Xtend files stores common function for the process the model to generate the code, these functions are designed for a specific stereotypes and model configuration, for example, theres a function that get all the components with the stereotype EcomponentImpl and return a list of these components.

/\*\*

\* Get EComponentImpl list from a certain package

\* @param Package

\* @return List

\*/

List getEComponentsImpl(Package package):

package.allOwnedElements().select(e|isEComponentImpl(e));

The elements in the list that return the funtion can be processed by the template, this function is coded using the xtend programming language very similiar to Java. The function is used by ACSComponent.xpt template (example above).

In this project theres several xtend files to support the templates (separated by languaje, usage, etc) for the code generation, these funcions are used as helper functions.

## Internal Lifecycle code generation

Adasd

## Source Code Structure

Adasd

## Class Diagrams

Adsad

# Model definition

The generator use a very specific model design for the generation of the code and this model desing have some constraints to bear in mind.

Any UML design software that support stereotypes definition in the model and export the model to XMI can be used for the code generator, currently ESO is using MagicDraw 16.x version to desing the models for the generator, a trial version can be downloaded from [www.magicdraw.com](http://www.magicdraw.com)

for the model creation.

For the moment, exists only six case use for the model desing, explained below.

## EProfile

EProfile is a file that has all the stereotypes, tags, and metamodels for the model creation for the component generator, all the stereotypes with the E char prefix, the E prefix is for ESO . The generator use this profile to generate the code, for this reason all models must use this profile, currently this profile is developed in MagicDraw and is exported as XMI, but also can be implemented in other modeling softwares.

Some stereotypes use tags to define certain behaviors, this tags are defined by this enumerations:

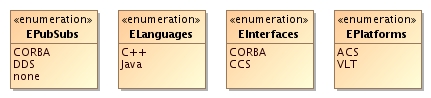


Figure : Enumerations for stereotype tags

An example of this is the stereotype *<<EContainer>>* has the *language* tag, this tag is defined by *ELanguages*.

The stereotypes that the model and generator support are:

### EInterface

Asdsad

### EComponent

Asdsad

### EComponentImpl

Asdsad

### EContainer

Asdsad

# Code Generator

asdasd