Atacama Large Millimeter / submillimeter Array

Alma Common Software Component Code Generation

Project Documentation

ALMA-BBB-CCC-A-DDD

Version: A
Status: Draft

2010-31-01

Keywords: oAW5,migrate ,UML, code generator, Java, component

Prepared	Alexis Tejeda	YYYY-MM-DD	
	Name	Date	Signature
Prepared	Gabriel Zamora	YYYY-MM-DD	
	Name	Date	Signature
Approved			
	Name	Date	Signature
Released			
	Name	Date	Signature

Change Record

Issue/Rev.	Date	Section/Parag. affected	Reason/Initiation/Documents/Remarks
A	2008-02-29	All	Creation.

Table of Contents

List of Figures

1 INTRODUCTION

1.1 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.

1.2 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.

1.3 Reference Documents

[1] ACS Component Code Generation Framework
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/

1.4 Abbreviations and Acronyms

- ACS : Alma Common Software.
- EMF : Eclipse Modeling Framework, a framework of Eclipse to create plug-ins, 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

1.5 Stylistic Conventions

The following styles are used:

bold
In the text, to highlight words.

italic
In the text, for parts that have to be substituted with the real content before typing. Also used to highlight words or section names.

`teletype`
In the text, for commands, filenames, pre/suffixes as they have to be typed. Also used for file content examples.

`<name>`
In examples, for parts that have to be substituted with the real content before typing.

`<<stereotype>>`
Stereotypes used in the code generator.

2 PROJECT OVERVIEW

2.1 Previous Work

The work done before this project is related to the thesis[1] of Nicolas Troncoso, which gives the very beginning to the ACS Code Generation based on UML metamodels, this project will cover and continue the work already done , proposed in the thesis as a 'Future Work', like the implementation of inheritance, more complex models, separation of implemented and generated classes and have a full implemented code generator for ACS.

2.2 Project Objectives

The main goal/objective in the project is have a full Standalone Java Component Code Generator for Alma Common Software under Eclipse Modeling Framework umbrella, this, is divided in specific objectives:

- Project based in Open Architecture Ware 5:
  All the project has to be based on Open Architecture Ware 5, now under the EMF umbrella.
- Managment of complex UML Models:
  In complex UML models, the use of stereotypes that allow to filter the models to generate, this mean that a class will have a stereotype to identify if this class will be generated or not, this is for not re-write classes already generated. This objective is considered a critical factor for the success of the project.
- Alma Common Software - Notification Channels Support:
  A full code generation support for ACS Notification Channels, generate all necessary code from a UML model that represents the association between the Supplier and Consumers, the generator must be able to generate the ready-to-implement code for Notification Channels. This objective is considered a critical factor for the success of the project.
- Standalone Code Generator:
  The development of the generator should be under the EMF, however one of the critical requirements is have the generator as Standalone, this means that the generator have to work outside the Eclipse IDE dinamicly with any UML model/metamodel specified by command line. The generator have to generate all the classes, plus empty skelletons for the user implementation. or regenerate only the basic classes, this mean only generates the templates for user modification. This objective is considered a critical factor for the success of the project.
- The generated code must be Java:
  At first, all the code generated must be in Java, later, if the time allow, then implement the code generator templates for C++ and Python, it's more importante in this project have a full support Java for the code generated than a semi-implementation of all programming languages.
- Separation between the generated classes and implemented classes:
  Complete the separation between the generated classes and the implemented classes,

this mean the code generator has to know how to separate the implemented classes and the already generated classesfor the code generation for not overwriting.

- Inheritance implementation:
  Actually, the implementation in the generator of model-inheritance is not fully supported, the developers should define how implement this model behavior in the code generator without affect all work already done.

2.3 Project Scope

The project scope cover the development of the Code Generator for ACS, initially the project will only generate the Java code implementation from a UML model under the XMI2.0 standard, in which, there are three importants points defined as critical and project milestones.

- Standalone Java Component Code Generator.
- Notification Channels and Inheritance Support for UML models.
- Code Generator based on Open Architecture Ware 5.

Each one of this points are explained more consistently in Goals and Objectives sections in this document.

2.4 Roles and Responsibilities

- AlexisTejeda (UCN) Deployment Manager, Requirements Reviewer, Architecture Reviewer, Configuration Manager, Change Control Manager .
- GabrielZamora (UTFSM) Deployment Manager, Requirements Reviewer, Architecture Reviewer, Configuration Manager, Change Control Manager.
- GianlucaChiozzi (ESO) Client, End User, Requirements Reviewer, Change Control Manager.
- JorgeIbsen (ESO / JAO Computing Manager)Project Manager, Requirements Reviewer.
- NicolasTroncoso (JAO/AUI) Project Manager, Requirements Reviewer, Architecture Reviewer.

3 DESIGN OVERVIEW

3.1 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.

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, this program generate the necessary XMI files for later generate the code component to be implemented. Is important to specify that the models need a profile file, which defines the stereotypes for use in the generator, where they differentiate the characteristics of UML model.

3.1.1 Stereotypes

In UML models, the use of stereotypes helps to distinguish the classes that must be generated and how must be generated. This stereotypes are defined in the UML profile file, which is created when the UML model is exported from MagicDraw.
One of each classes must have a stereotype defined to know how the code must be generated, i.e.:

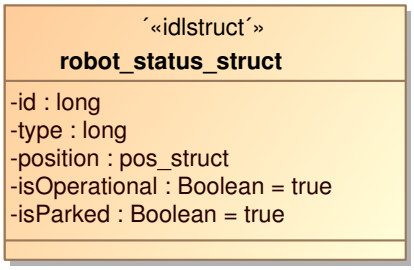

Figure 1: Example class with a stereotype

The class robot status struct has the `<<IDLStruct>>` stereotype, then the generator call the IDLStruct template and generate the code of an IDL file of ACS.
Full detail and description of the stereotypes used in this project, please refer to the Quick How To section in this document.

Exist three types of stereotypes, see 'Quick Reference User Manual' for full description of each stereotype:

- Class stereotype : applies to classes like `<<IDLStruct>>` stereotype.
- Property stereotype : applies to class level variables.

- Operation stereotype : applies to a method of a class.

3.1.2 Xtend/Xpand

Since owA was migrated to EMF, the code generator is based on a mixture of EMF subprojects(Xpand/Xtend/Xtext) , the EMF subprojects which are used, are:

- 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.

3.1.3 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 expoted models, output folder, templates used, code beautifiers and the package name definition for the generated code.

The path of the model to generate, the generated code output folder path and the UML profile file path are specified using dynamic workflow propertys by ${myVariable} sintax. These properties can be configured dynamically in the Java program (using String Hashmap) that call the generator, or by command line.

Also the templates to use in the generation are specified in the configuration file as 'Workflow Components'.

The components 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 three profile files.
- Template : the template to use.
- Beautifiers : XML or Java code beautifiers.
- VetoStrategy : only is specified if the component will use a Veto Strategy, see 'Generator Optimization' section for more info about this.
- File encoding : by default UTF-8.

An example configuration for a component, in this case a java class files component generator :

```
<component id="genjava" class="org.eclipse.xpand2.Generator"
 skipOnErrors="true">
<fileEncoding value="UTF-8" />
    <metaModel idRef="default_profile"/>
    <expand value="templates::java::JavaRoot::Root FOR model"/>
    <outlet path="${ouputFolderURI}">
        <vetoStrategy
         class="cl.alma.acs.ccg.vetostrategy.ACSCCGVetoStrategy" />
        <postprocessor
         class="org.eclipse.xpand2.output.JavaBeautifier"/>
    </outlet>
    <prSrcPaths value="${ouputFolderURI}"/>
</component>
```

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

The workflow file is based in oaW 5, this means that the file presents many changes from his previous version, more info about the migration of the workflow, see 'oaW 4 to oaW 5 Migration How To' section in this document.

3.1.4 Template Files

Template files controll the output code generation. Each class in the UML model is analyzed by the template file, the template check the stereotype of the class and generate the output file. The templates are based in the Xpand programming language [4].
All templates are encoded in UTF-8 by the use of 'guillimets' [4].

3.1.5 Xtend Util Helper

In the generator templates folder exists a Xtend file, this file is a helper for the templates files, in which, are defined helper functions like if a class is inherited from other class. More info about the functions, see 'Quick Reference User Manual' in this document.

3.2 Generator Optimization

With a simple model (20 classes) the generator can take about 8 or 10 seconds to generate the code in a Dual CPU@1.73GHz with 2048 MB RAM, but, if the model is more complex, then the generation can take longer, this is no problem at all, the problem comes when the code must be generated again for any reason, like add a method in a class or add another class in the model. This is a problem, because, every time we want re-generate the code, will take the same time as the first generation even if there are no changes in the model (same code to generate.)

To fix this, the use of EMF Veto Strategy is implemented in the components workflow file configuration. This strategy is class that implements a EMF Veto interface class, in which each file to be generated, is analyzed if presents any changes, if there any changes, the files

is generated again, if not, the file is not generated again.

This strategy improves the generation time in regeneration, complex models, and model refactoring.

4 COMPONENT CODE GENERATOR

4.1 Source Code Structure

The generator is packaged in a JAR Java file, under the reverse domain name cl.alma.acs.ccg which follow the Java package folder structure.

```
cl.alma.acs.ccg/
|-- cl/
| '-- alma/
|     '-- acs/
|         '-- ccg/
|             '-- mwe/
|             | |-- CppWorkflow.mwe
|             | |-- JavaWorkflow.mwe
|             | |-- PythonWorkflow.mwe
|             |
|             |-- strategy/
|             | |-- CodeCppGeneration.java
|             | |-- CodeJavaGeneration.java
|             | |-- CodePythonGeneration.java
|             | |-- ContextCodeGeneration.java
|             | |-- ICodeGenerationStrategy.java
|             |
|             '-- vetostrategy/
|             | '-- ACSCCGVetoStrategy.java
|             |
|             |-- vo/
|                 '-- VOGenerator.java
|
' templates/
    '-- java/
        |-- AllOverride.xpt
        |-- CDB.xpt
        |-- CommonRoot.xpt
        |-- IDLCommon.xpt
        |-- IDLComp.xpt
        |-- Implements.xpt
        |-- info.xpt
        |-- Interfaces.xpt
        |-- JAVAChComp.xpt
        |-- JAVAComp.xpt
        |-- JAVAHelper.xpt
        |-- JavaRoot.xpt
        |-- Makefile.xpt
        |-- Override.xpt
        |-- SchemaRoot.xpt
        |-- Schema.xpt
        '-- util.ext
```

4.1.1 Class Diagram

In the class diagram, the only classes not developed (in the project) are those which belongs to the packages `org.eclipse.xpand2.output` and `org.eclipse.emf.mew.core`.

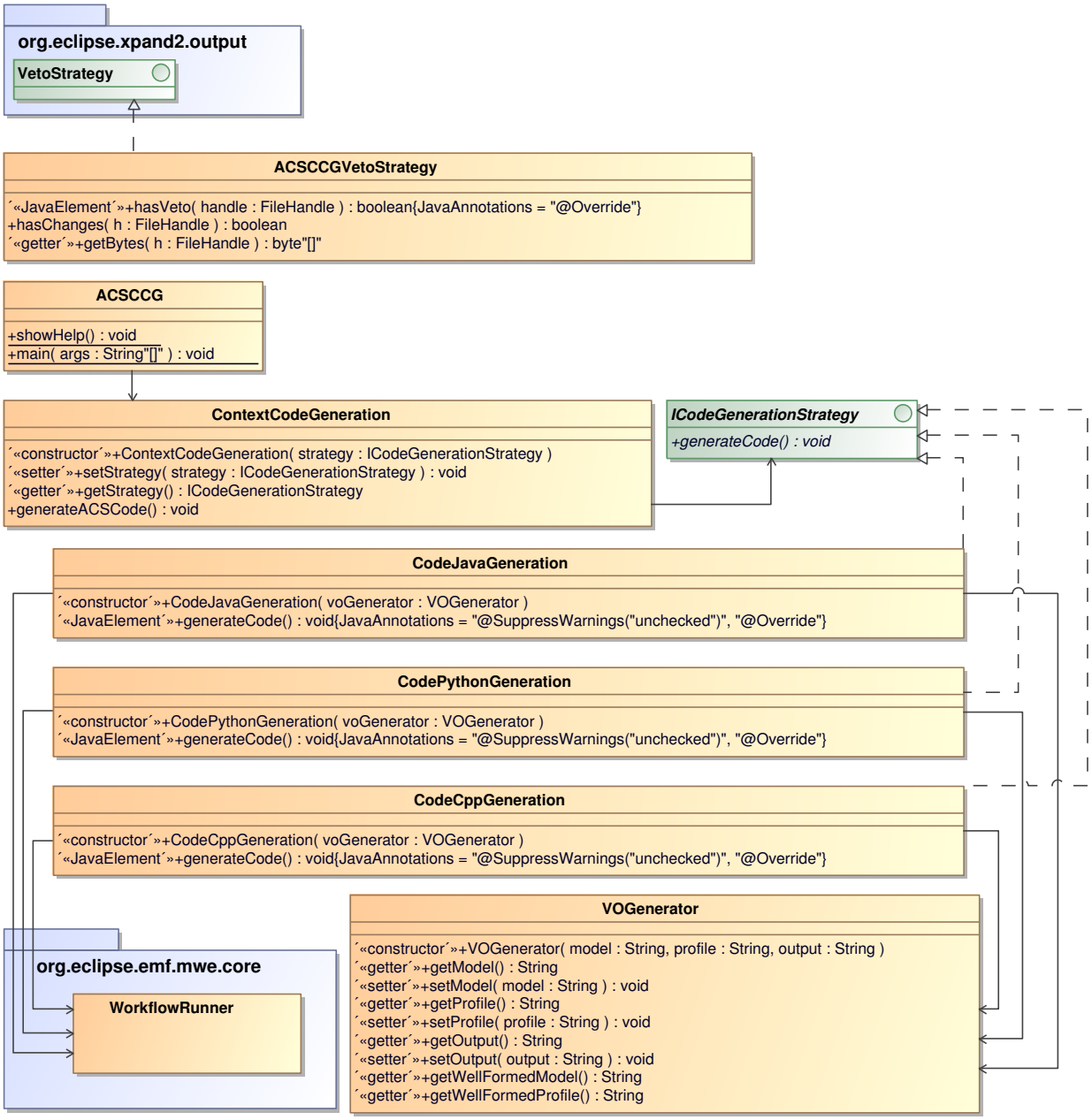

Figure 2: Component Code Generator Class Diagram

4.1.1.1 Strategy Pattern

A Strategy Pattern (Policy Pattern) was implemented in the component code generator to generate the code for each programming languaje in ACS whithout change top levels algorithms, due to the scope of the project, only Java strategy is full implemented.

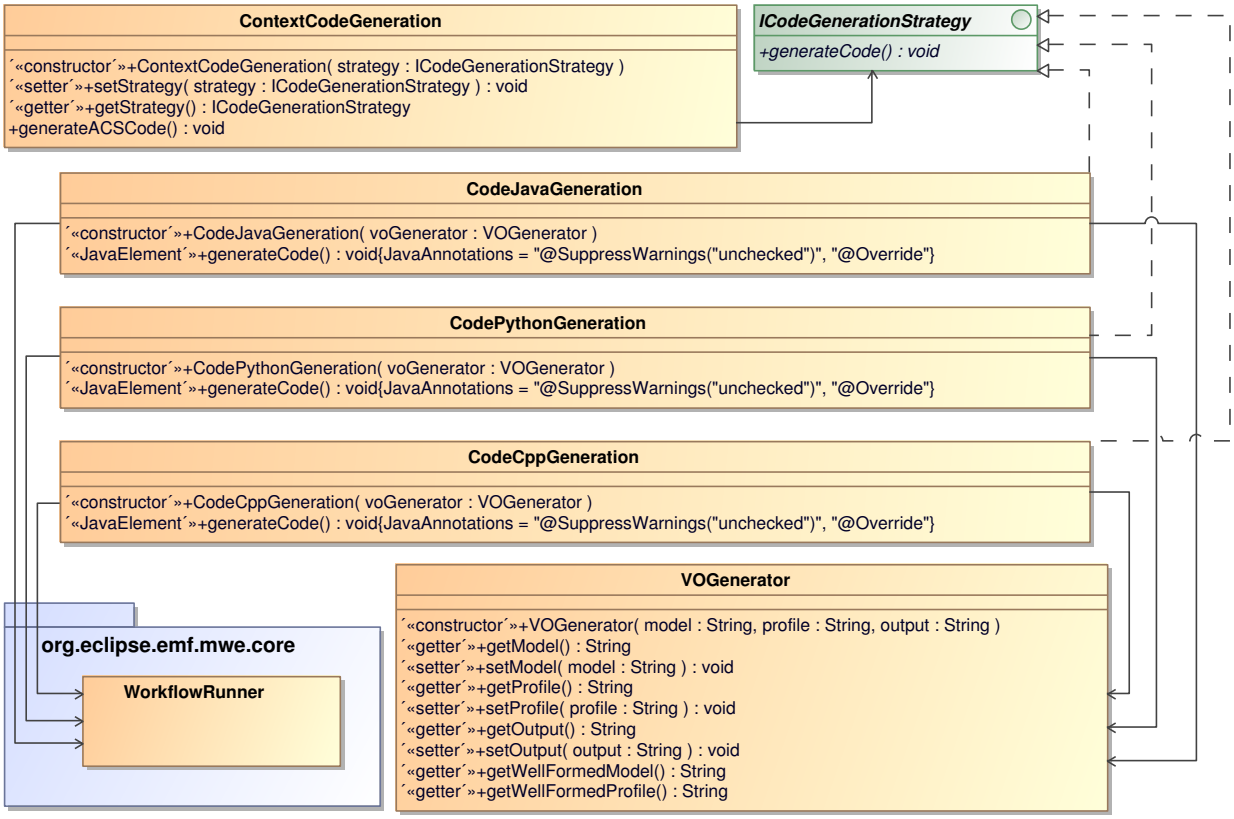

Figure 3: Strategy Pattern

The `ContextCodeGeneration` is the main class to be called for implement the generator in plugins, Java programs or other enviroments.

4.1.1.2 EMF Veto Strategy

The class `ACSCCGVetoStrategy` is called by WorkflowRunner in runtime of the component code generator, for more info about Veto, see Generator Optimization section in this document.

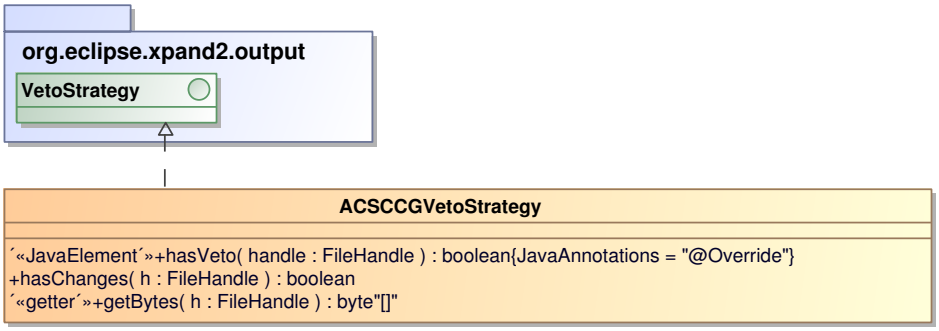

Figure 4: EMF Veto Strategy

4.1.2 Sequence Diagram

To be implemented

4.1.3 Process Diagram

To be implemented

4.2 Stand-alone Generator

The component code generator was designed to be a stand-alone code generator, this means, can be executed in any O.S. as a Java Program, JAR Package or Eclipse Plugin. More info how to use, see the Quick How To section in this document.

The generator supports three ways to be a standa-lone :

- JAR Package : The binary files are packaged in a JAR Java file.
- Eclipse Plugin : A simple Eclipse plugin.
- Command Line : A Java command line program.

Also, the generator contains all the packages to be executed without EMF and can be imported to other environments running Java.

4.3 Notification Channel

The ACS Notification Channels API provides all the necessary to allow the communication of data between components. In more detail, this is the minimal list of a necessary set of components.

- Channel Data:

  This struct provides the sharing of data events between two or more diffent components. Basically consists in common data types:

  - A common data
  - A common Channel name

- Simple Supplier:

  This component has the function of control over all the data in the context of send the events to the Notification Channel:

  - Send simple data events
  - Send special data events
  - Data control in case of problems

- Consumer:

  This component has the functionality of obtain all the data from the common Notification Channel. The consumer gets the data automatically when a event is available on the Notification Channel.

  – Receive the data events

- Simple Supplier Client:

  This client activate the Simple Supplier component and set the user data events to send between the Simple Supplier and the Consumer.

- Consumer Client:

  This client maintains the activation of the Consumer component to receive the data events from the Notification Channel

4.3.1 Diagrams

This is a overview of the Notification Channel implementation on the ACS Code Generator

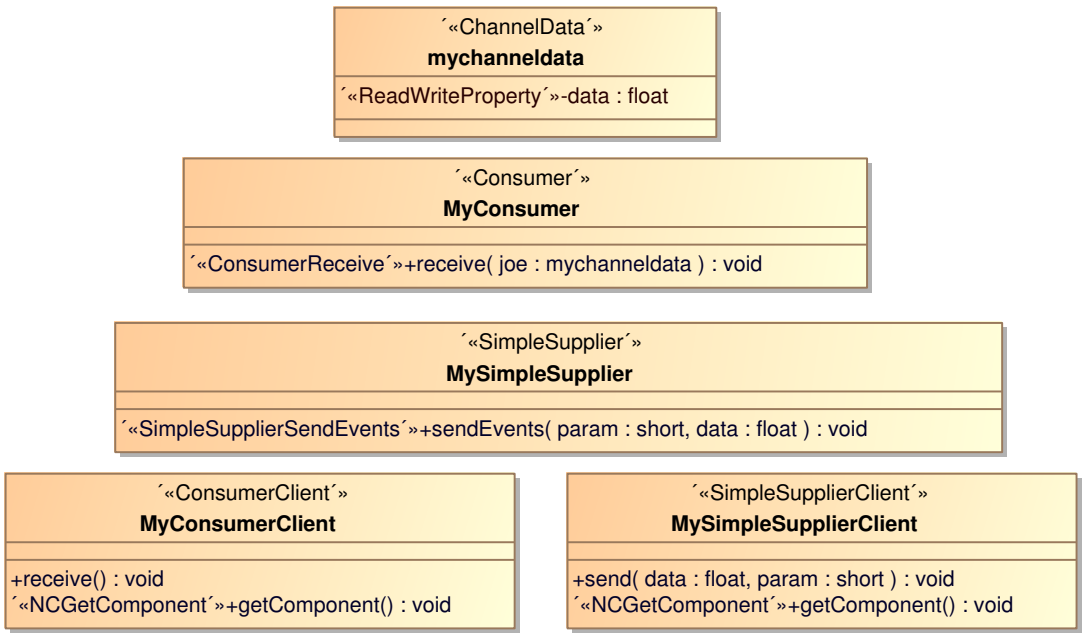

Figure 5: Example Notification Channels class diagram

The generator provides a fully implementation of all the necessary to make use of this functionality. The code is generated in base of the Notification Channel UML Model, taken attributes and operations from the model and building a customizable Supplier-Consumer structure. The code generated in detail:

`src-gen/`

```
|-- config/
|   '-- CDB/
|       '-- MACI/
|           |-- Channels/
|           |   '-- mychanneldata
|           |       '-- mychanneldata.xml
|           |-- Components/
|           |   '-- Components.xml
|           '-- Managers/
|               '-- Manager/
|                   '-- Manager.xml
|-- idl/
|   |-- MyConsumerBase.idl
|   |-- MyConsumerClientBase.idl
|   |-- MySimpleSupplierBase.idl
|   |-- MySimpleSupplierClientBase.idl
|   '-- NCCommon.idl
'-- src/
    |-- Makefile
    '-- alma/
        '-- acsgenerator/
            '-- NC/
                |-- MyConsumerBase.java
                |-- MyConsumerBaseHelper.java
                |-- MyConsumerClientBase.java
                |-- MySimpleSupplierBase.java
                |-- MySimpleSupplierBaseHelper.java
                '-- MySimpleSupplierClientBase.java
```

In the tree view is possible see three sub directories:

- config: Includes the manager, components and notification channel configurations to the CDB.
- idl: Includes the IDL Interfaces for the Supplier, Consumer, clients and the common data to share.
- src: Includes the source code of the Supplier and Consumer components and clients.

4.4 Inheritance

4.4.1 Multiple Level

The generator can support inheritance in one ore more inherited levels with the ability to override the inherited methods from the parenet, or, override all methods inherited in all inherited levels.
This can be viewed in figure 5.

The class SMN is extended from FD class and inherits implicitly the methods from FD without override them.

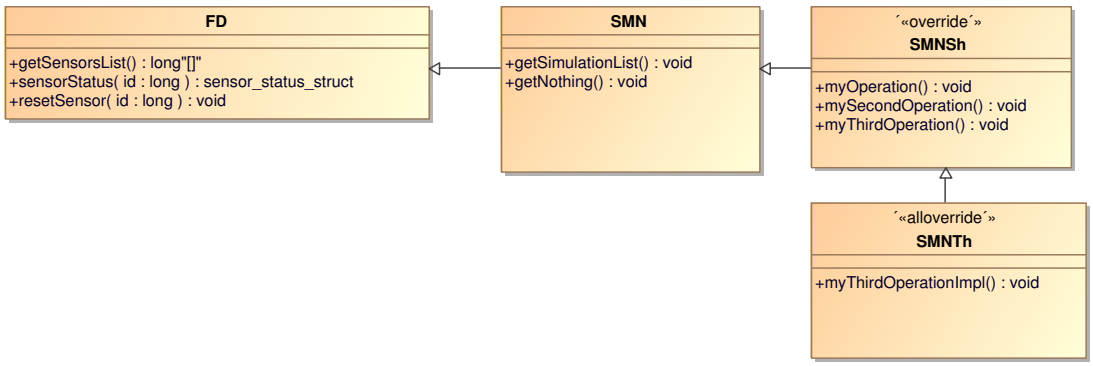

Figure 6: Inheritance in the generator

The class SMNSh is extended from SMN, and override the methods from SMN.
The class SMNTh is extended from SMNSh and override all methods inherited, the methos from FD, SMN and SMNh.

A stereotype will differentiate if the class generated must apply the override policy in the inhertance.

4.4.1.1 Characteristic Component

A class with the stereotype `<<Characteristic>>`, if is extended from other class, the generator will not implement the inheritance in the Java code, because the Java OOP paradigm not support multiple inheritance (parallel inheritance) and the class with `<<Characteristic>>`, the generator will generate the class already extended from ACS class 'CharacteristicComponentImpl'.

4.4.2 Interface

The inheritance can be applied to Interface Classes, the interface will not override the methods (Java OOP aspect) from the super-interface (parent interface). But, the class will have to implement the methods from child Interface and all his inherited methods.

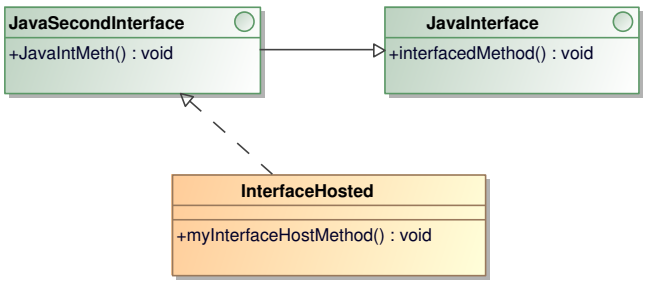

Figure 7: Inheritance in interfaces.

4.5 Interfaces and Abstract classes

For desing pattern implementations, the generator support basic features of Java OOP as Interfaces or Abstract Classes.

4.5.1 Interface Classes

The generator support the interface implementation specified in the model and provides the generated code under the Java OOP standard. It follow the OOP Java Paradigm.
Any class can implemented one or more interfaces, the methods implemented from the interface will be generated. An example of this can be viewed in Figure 8.

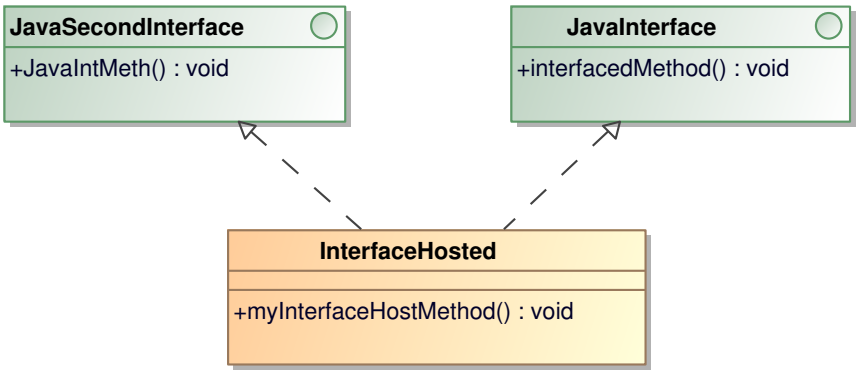

Figure 8: Interface Example

Will generate :

```
public class InterfaceHosted implements ComponentLifecycle,
InterfaceHostedOperations, JavaInterface, JavaSecondInterface {
...
}
```

4.5.2 Abstract Classes

Same as interface implementation, the generator support the abstract class definition and generate the code.
Classes with the `<<Characteristic>>` stereotype, the generator will no generate this classes as Abstract.

4.6 Implemented Code v/s Generated Code

In every model driven development and code generators, is important how make a strategy to separate the code already generated with the code implemented over the generated code. They are three ways solve this.

4.6.1 EMF Veto Strategy and Inheritance

The Veto EMF strategy (see Generator Optimization section) is a good solution using inheritance to implement new things, but, must be create a new class for every change in our code to void override the whole code already generated, so, this is not the best solution, but, it works.

4.6.2 GAP Desing Pattern

The generation GAP desing pattern provides separate the code with all hand-modifications implemented in sub-classes, this mean that the core classes are generated only once and the hand-made implementations, are extended as subclases from core classes.

4.6.3 Protected Areas

Xpand, provides protected areas to our generated code, this mean, that certain areas with the protected tag with a unique autegenerated ID, can be modified by the developer without loosing the hand-modifications over the generated code when the code is re-generated (even if classes are changed in the model). i.e.: This code went generated with protected areas.

```
/*PROTECTED REGION ID(getSimulationList) ENABLED START*/
public void getSimulationList() {

    //Implementation Method here!
}
/*PROTECTED REGION END*/
```

Protected areas are specified in generator templates, the generator analize the generated code, check the code with the template, and protect the area from the regeneration. If the protected region is not in the templates, the generator will not protect the area.
For the code that scape from UML model every class has a protected area for custom imports, variables and methods.

5 COMPONENT CODE GENERATOR INSTALL GUIDE

5.1 System Requirements

5.2 JAR File

6 QUICK REFERENCE USER MANUAL

6.1 Command Line

7 QUICK REFERENCE DEVELOPER MANUAL

7.1 Eclipse

7.2 Workflow

7.3 Templates

8 MIGRATING FROM oAW 4 TO oAW 5 - EMF

8.1 Eclipse

8.2 META.INF

8.3 Workflow