A code generation framework for the ALMA common software

Nicolás Troncoso[4,1], Horst H. von Brand[1], Jorge Ibsen[2], Matias Mora[4,1], Víctor González[4], Gianluca Chiozzi[2], Bogdan Jeram[2], Heiko Sommer[2], Gabriel Zamora[1] and Alexis Tejeda[5]

[1]Computer Systems Research Group, Universidad Técnica Federico Santa María (UTFSM), Valparaíso, Chile
[2]European Southern Observatory (ESO), Santiago, Chile
[3]National Radio Astronomy Observatory (NRAO), Socorro, NM
[4]Associated Universities, Inc. (AUI), Santiago, Chile [5]Universidad Católica del Norte, Antofagasta, Chile

ABSTRACT

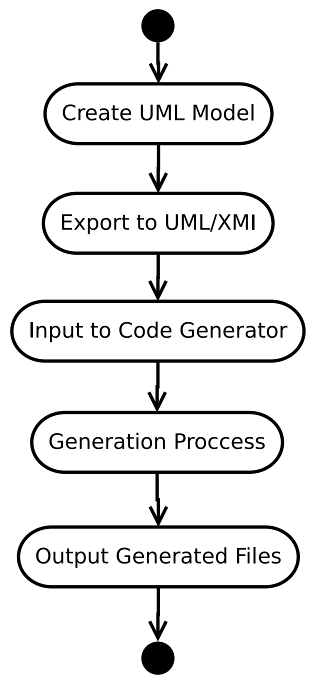
Code generation helps in smoothing the learning curve of a complex application framework and in reducing the number of lines of code that a developer needs to craft. The ALMA Common Software (ACS) has some what adopted code generation, but we are now exploiting the more comprehensive approach of Model Driven code generation to transform directly an UML Model into an implementation in the ACS framework.
This approach makes it easier for newcomers to grasp the principles of the framework. Additional benefits achieved by model driven code generation are: software reuse, implicit application of design patterns and automatic tests generation.

Introduction

The generation framework presented in this poster uses openArchitectureWare as the model to text translator. OpenArchitectureWare provides a powerful functional language that makes this easier to implement the correct mapping of data types, the main difficulty encountered is the translation process. The output is an ACS [1] application readily usable by the developer, including the necessary deployment configuration, thus minimizing any configuration burden during testing. The specific application code is implemented by extending generated classes. Therefore, generated and manually crafted code are kept apart, simplifying the code generation process and aiding the developers by keeping a clean logical separation between the two.
Our first results show that code generation improves dramatically the code productivity.

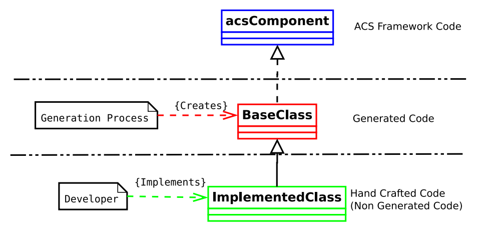
Proposal

1. Code generation starting from a UML model (and/or text representation). UML (Unified Modeling Language) is a software modeling specification maintained by the Object Management Group (OMG)
   a) Create IDL files. This implies creating a full implementation of the interfaces in the model as IDL files so it may be compiled by IDL compilers.
   b) Create a base class implementation starting from a class diagram. Base classes will be functional as they implement the relevant interfaces of the ACS framework.
   c) Define the type mapping between the UML model representation and the specific language implementation.
2. Integrate design patterns with the code during the generation process.
3. Generate code for one of the ACS supported languages; as a start, Java.
4. Finally, validate the proposed solution through external opinions, current experience, and generating example implementations of a variety of models through a prototype implementation.



This steps taken in the generation workflow in order to convert an UML model into an ACS system, The figure to the left shows a general layout of the steps needed to accomplish the end to end generation task.

Generated and Non Generated Code

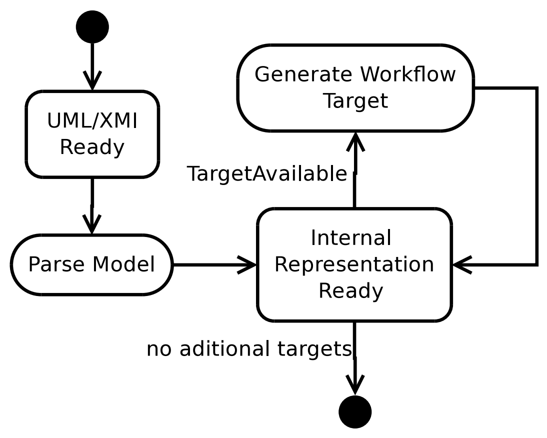


It is not possible to fully generate the logic within the software component. To achieve the full functionality of the component the developer must extend the generated code. This scheme has been shown to work in the ALMA CONTROL software code generation [2,3] where the generated code is extended by class inheritance. The same approach is used in the ACS Code Generator: a minimal working component is automatically generated, but the internal logic must be added by the developer by means of class extension.

The main benefit of this approach is that it isolates the generated code from the user written code. The side effect of this is that code may be regenerated multiple times without affecting user implemented code. This is known as a code separation pattern as seen in the figure. During the development cycle the base (generated) classes are created, then the developer extends these classes and implements complex functionality within the extensions. This approach is used with the generated IDL and generated Java files. It is also possible to use the same approach with the generated XSD files.

Generation and Open Architecture Ware

The internal steps taken by the code generation, it is a state machine that will transform the input model into an output for every target that has been specified. The targets described in the following table are tasks taken by the ACS code generator in order to generate the ACS software module.



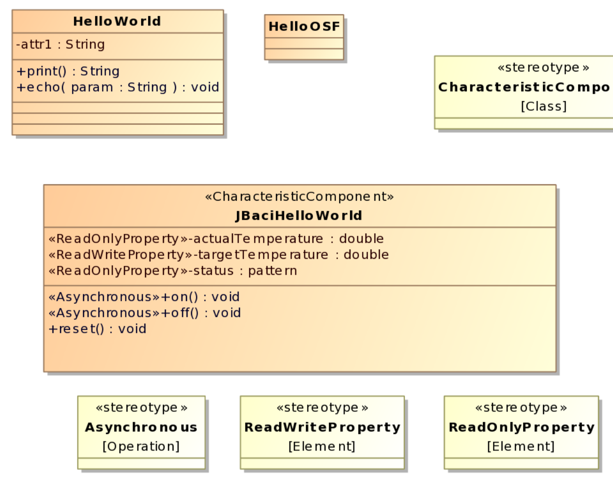
The tasks described in the following table show the different OAW targets which are processed in order to generate an ACS software module. Every task is independent of the others, so they may be interchanged, eliminated or new steps could be added.

Task Name	Description
IDL tasks	
`genidlCommon`	Generate common IDL files. Contains structs and enumerated types.
`genidl`	Generate IDL files for every ACS component.
Java Tasks	
`genjava`	Generate Java Helper and Java Base classes.
ACS module tasks	
`genschema`	Generate Schemas in case of ACS characteristic component.
`gencdb`	Generate test CDB.
`genmake`	Generate Makefile.

The UML Model

When an input model is created for the ACS code generator, the UML model make use of new stereotypes not specified in the UML standard but specific for ACS.

Stereotype
Class Stereotype
`<<CharacteristicComponent>>`
`<<NOGenerated>>`
`<<IDLStruct>>`
Type Member Stereotype
`<<ReadOnlyProperty>>`
`<<ReadWriteProperty>>`
Function Member Stereotype
`<<Asynchronous>>`


HelloWorld
-attr1 : String
+print() : String
+echo(param : String) : void
HelloOSF
«stereotype»
CharacteristicComponent
[Class]
«CharacteristicComponent»
JBaciHelloWorld
«ReadOnlyProperty»-actualTemperature : double
«ReadWriteProperty»-targetTemperature : double
«ReadOnlyProperty»-status : pattern
«Asynchronous»+on() : void
«Asynchronous»+off() : void
+reset() : void
«stereotype»
Asynchronous
[Operation]
«stereotype»
ReadWriteProperty
[Element]
«stereotype»
ReadOnlyProperty
[Element]

An important feature in the generation process is to define some class as `<<NOGenerated>>`, this instructs the code generator not to generate any implementation for it. This is significant when doing complex extensions.

Some times the developer does not want to generate code for a particular class, but wants to keep the model complete so the generator may define extending classes appropriately, so the `<<NOGenerated>>` stereotype may be used.
At the time when this poster was prepared the `<<NOGenerated>>` feature was under discussion and most probably it will evolve into generating only the classes that have ACS specific stereotypes and not generating the others by default. Keeping with the spirit of having a complete model of the application, but not generating all the classes in the model.

Conclusions

The development process was based on an iterative process with hands on experience and real examples that made shortcomings and modifications needed in the design evident. The shortcomings detected were addressed in the evolution to the actual implementation, achieving a maturity such that the generated IDL files were used for the 6th ACS Workshop Basic Track Live Example.
Some of the achievements of the ACS Code Generator are:

- A pluggable module design was achieved, in the sense that supporting an additional implementation language or target does not disrupt, nor cause great impact, in the generation framework.
- The generated module is ready for compilation and usage as generated. No further development is required for initial testing.
- The version of OAW used during development is the same as the one shipped by ACS-8.1. No porting efforts are required for integration.

This ACS Code Generator provides and excellent starting point (training) for a new comer to the ACS Framework. Its possible to have a working software module in under 10 seconds (time the generation process takes). The novel ACS user has only to fill in the stubbed functionality. The alternative is to adapt an ACS Example module, but this has always been error prone and new developers will take longer to achieve a simple working implementation.

Acknowledgments

This work was supported by ALMA-CONICYT Fund project #31060008 *"Software Development for ALMA: Building Up Expertise to Meet ALMA Software Requirements within a Chilean University"*, under development at Universidad Técnica Federico Santa María, and the National Radio Astronomy Observatory, through Associated Universities, Inc.

References

[1] Chiozzi, G. et al., "Common Software for the ALMA project" in [*Accelerator and Large Experimental Physics Control Systems*], (2001).
[2] Farris, A., "Generating Software Modules Using Model Driven Software Development" in [*Proceedings of Astronomical Data Analysis Software and Systems XVI*], (2006).
[3] Farris, A. et al., "Device Driver Code Generation Framework" in [*ALMA Documentation*], (2007).

Atacama Large Millimeter Array

Contact e-mail: ntroncos@alma.cl / For more information about ALMA development and research at UTFSM Computer Systems Research Group, please visit our web site http://alma.inf.utfsm.cl, http://www.almaobservatory.cl