Astronomical Data Analysis Software and Systems XV **O7.2**
ASP Conference Series, Vol. XXX, 2005
C. Gabriel, C. Arviset, D. Ponz and E. Solano, eds.

ACS as the Framework for Offline Data Reduction in ALMA

Steve Harrington, David DeBonis, Joseph McMullin, Wes Young

National Radio Astronomy Observatory (NRAO), Socorro, NM 87801

Gianluca Chiozzi, Bogdan Jeram

European Southern Observatory (ESO), Garching, Germany, 85748

Abstract. An integral part of the ALMA system is to process astronomical data offline. Such data reductions are often broken up into discrete units of processing called tasks. While the ALMA system will provide many typical offline tasks out of the box, the ability to easily add new tasks which perform data reductions currently unforeseen by ALMA developers is crucial. It is therefore important that the ALMA Common Software (ACS) provide a mechanism for quickly and easily creating offline data analysis tasks. The mechanism which ACS has implemented includes a set of XML schemas supporting the description of tasks and an Application Programmer's Interface (API) which offline task developers can use to quickly integrate data reduction logic into the ALMA system with minimal additional programming effort beyond that of coding the actual data reduction algorithms. This paper presents the status of offline data processing support in ACS, including mechanisms for defining and implementing data reduction tasks. Future enhancements and extensions will also be discussed.

1. Introduction

The Atacama Large Millimeter Array (ALMA) is an exciting new radio telescope under development, to be located in the Atacama Desert in Chile. ALMA Common Software (ACS) is the framework upon which all ALMA application software is written. ACS provides a component-container model supporting distributed object-oriented software development, common services, and implementations of useful design patterns (Chiozzi et al. 2003). Support for offline processing of astronomical data is an important feature of the ALMA system. These data reductions will typically be broken up into discrete units of processing called tasks. While the ALMA system will provide many typical offline tasks out of the box, the ability to easily add new tasks, which perform data reductions currently unforeseen by ALMA developers, is crucial. This paper discusses how the ACS framework will support the development of offline data reduction tasks.

2. Tasks, Parameters, Parameter Sets, and Metadata

A *task* is a concise process which starts up, performs some processing, and shuts down. A task may or may not require additional services from the ACS framework; it should be able to run, at least in a locally hosted manner, whether or not ACS services are present. While in most cases the full ACS framework will be installed and running, its absence should not completely preclude the possibility of running of a task. However, the absence of ACS services will present various limitations. For example, the ACS framework provides the communication infrastructure for distributed computing and, when missing, a task would not be able to access other distributed objects in the system. In this case, the task would need to be capable of functioning entirely independently in order for any useful results to be computed. Finally, a task requires input data in the form of a set of *parameters*. The particular parameters that are required will vary depending on the task. Values for parameters must be supplied at run time.

A *parameter set* is a group of individual parameters which collectively are used as input to a task. Individual parameters may be of various types, e.g. boolean, string, integer, double, and arrays of these simple types. [1] A parameter may have a default value, constraints (e.g. max and min), a list of (enumerated) valid values, associated help text, and so forth. This information comprises *metadata* about a particular task. This metadata must be defined by the task's implementor, i.e. the task author. The actual values of the parameters for a specific execution of a task must be supplied at run time by the person running the task, i.e. the task user. The task user may or may not supply values for parameters that have default values specified in the metadata.

3. Reduce, Reuse, Recycle

The ACS framework should facilitate developing new tasks and reduce the amount of effort that a developer would invest if writing a task from scratch. It should facilitate the reuse of existing data reduction packages. In the case of ALMA, AIPS++ will be leveraged for this purpose. Finally, ACS should facilitate the reuse of third-party packages other than AIPS++. While ALMA will focus on the reuse of AIPS++, the architecture should allow this decision to be revisited later (e.g. replacing AIPS++ with AIPS) without wreaking havoc.

4. Task Use Cases in ALMA

The primary actors related to ALMA tasks are the task author, the task user, and the AIPS++ developer (or a code generation framework).

4.1. Task Author Defines Metadata and Implements Task

The *task author* is responsible for defining the metadata about a task as well as implementing the core functionality for the task. In many cases, the functionality for a particular task may simply call through to an ACS component

[1] Currently, user-defined types are not supported, although this is a probable future extension.

wrapping AIPS++ or other third-party code. The metadata for a task, describing such things as the names and types of parameters, default values for parameters, parameter constraints, help text associated with parameters, etc., must be defined by the task author in an XML document which adheres to an XML schema definition (XSD) document provided by ACS.

4.2. AIPS++ Developer Writes ACS Component

The *AIPS++ developer* (or potentially a code generation framework) is responsible for implementing a component capable of running within the ACS component-container model and which wraps the underlying AIPS++ algorithms needed for the task's implementation. While reusing AIPS++ logic is expected to be common, in some cases the desired logic will not exist in AIPS++, or elsewhere, for reuse. In this case, the task author must write the algorithms from scratch and wrap them in an ACS component.

4.3. Task User Runs Task

The *task user* runs a task and provides, at run time, the necessary input data for the task's parameters. The task user can specify the values for parameters either in an XML document adhering to an XML schema definition (XSD) document provided by ACS or on the command line in the form of name/value pairs. When parameters are specified on the command line, the ACS infrastructure converts them to XML; in all cases XML is used.

5. Task Implementation

Figure 1 shows who does what. ACS provides an interface and an abstract base class. The ACS-provided base class, *ParameterTask*, implements the *run* method, which parses the parameters given on the command line (or in an XML file), fills in parameters with default values as needed, and validates them against the metadata provided by the task author. If the validation fails, an error is generated; otherwise, an in-memory representation of the parameter set is passed to the *go* method, which must be implemented by the task author.

6. Future Directions

Figure 2 shows the plan for offline data processing in ALMA. A code generator takes XML metadata as input and generates various bindings. One binding will be the ACS task binding described in this paper. Additional bindings are also possible (McMullin et al. 2005).

References

Chiozzi, G. *et. al.* 2003, in Proceedings of the 9th International Conference on Accelerator and Large Experimental Physics Control Systems, ed. J. Choi & I. Ko (Pohang, PAL), 214

McMullin, J. *et. al.* 2005, this volume, [P.99]

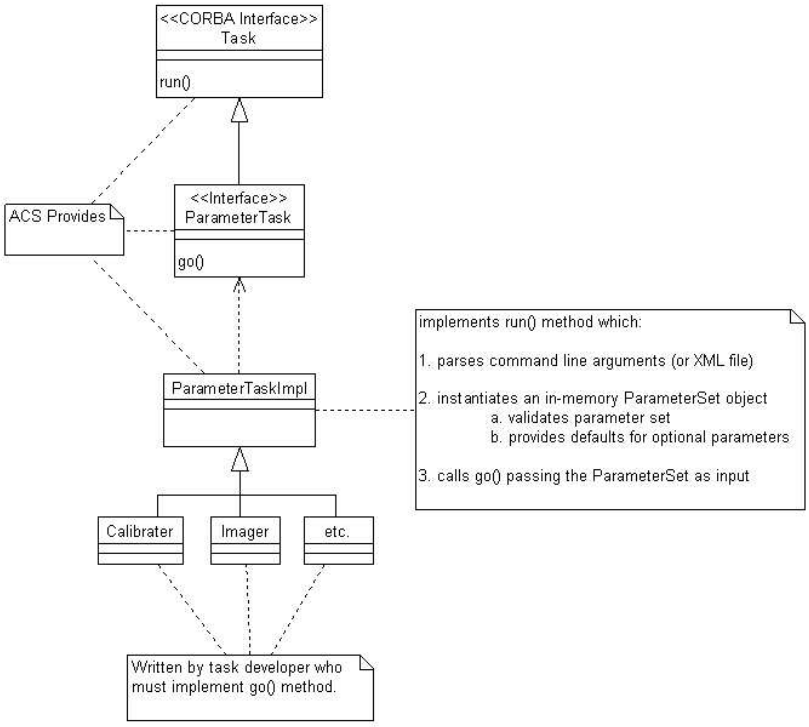

Figure 1. Responsibilities in Implementing a Task

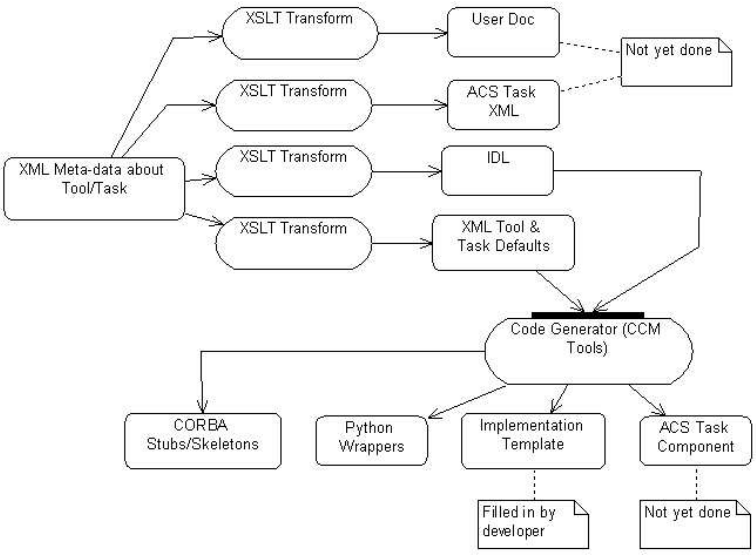

Figure 2. Future Direction of Offline Data Processing in ALMA