Astronomical Data Analysis Software and Systems IX
ASP Conference Series, Vol. 216, 2000
N. Manset, C. Veillet, and D. Crabtree, eds.

The Design of the Reduction Block Scheduler for the VLT Pipeline

K. Banse, P. Grosbøl

European Southern Observatory, Karl-Schwarzschild-Str. 2, D-85748 Garching, Germany

Abstract. All new data frames from the VLT are transfered to the data reduction pipeline which classifies new frames and selects the corresponding standard reduction tasks (Reduction Recipes) for each frame.

The observed data together with the relevant calibration data and other necessary parameters are then combined with the selected recipe in a component called Reduction Block. The Reduction Blocks represent the smallest executable units for the reduction pipeline.

The task of the Reduction Block Scheduler is to schedule and trigger the execution of these Reduction Blocks in the context of a given Data Reduction System on a given, possible parallel (Beowulf), system.

We describe the evolution of the RBS from a prototype implementation used to test the concept to the design of the baseline version of the RBS to be implemented in the pipeline within the coming months.

1. Introduction

As part of the Pipeline subsystem of the VLT Data Flow System (DFS) the Reduction Block Scheduler processes the incoming Reduction Blocks (RBs). The RB combines a set of input/reference frames with the explicit reduction task (Reduction Recipe) to be performed. Each Reduction Recipe is linked to at least one DRS which can be used for its execution, i.e. there exists a script in that DRS to execute the Reduction Recipe of the RB. The RBS has the task of scheduling the RB for execution in the framework of the given Data Reduction System (DRS).

A more detailed discussion of the DFS pipeline was given by Grosbøl et al. (1996, 1998).

2. Requirements and Challenges

We want one general pipeline infrastructure for many different instruments used at several telescopes. The VLT instruments will have quite a long lifetime, so we must be prepared for evolving hard- and software environments.

That means we must plan for new computer architectures, different operating systems and new data reduction software. The pipeline will be executed online at the VLT observatory as well as off-line at ESO headquarters in Garching. Furthermore, we want to support the individual recalibration of VLT data elsewhere, thus it must be possible to export the pipeline to other institutes.

That leads to the following requirements:

- Separation of RBS and DRS,
- Support more than one DRS,
- Single as well as parallel execution of RBs,
- The RBS must be portable and maintained and adapted easily,
- Use of an object oriented and portable programming language.

3. Evolution

A prototype of the RBS for UT1 (also used for UT2) of the VLT was written in C++ and uses MIDAS as the DRS. Conceptually it was possible to also use another DRS in parallel. But, we would need an upgrade of the code of the RBS each time a supported DRS was significantly upgraded or a new DRS was added. Also, the RBS had to be linked with the interface libraries needed for communication with all the supported DRSs. Thus, another DRS besides MIDAS was never realized.

However, the current RBS was extended to support multiple DRS of the same kind (MIDAS) to have several RBs running in parallel. This was done within the framework of the Beowulf project at Caltech, where RBs were executed in parallel on the different nodes of a Beowulf system, each running MIDAS.

To obtain more flexibility, a new baseline version of the RBS for UT3 and UT4 was planned and designed. The main emphasis was placed on the possibility to integrate a new DRS any time in a safe and easy way without the need of upgrading or even relinking the RBS.

That lead to the concept of removing all knowledge (and dependency) of a given DRS from the RBS and pushing it into a "DRS Wrapper" class. The RBS extracts from a given RB the Reduction Recipe and obtains, via the implementation of that Recipe, a DRS which is capable of executing the Recipe. Then, the RBS passes the RB on to the Wrapper of the specified DRS for execution using a single, generic interface for all Wrappers.

4. Architectural Design

4.1. RBS Package

RBScheduler class: RBS performs the high level administration of Data Reduction Systems available to reduce Reduction Blocks. It finds (via RecipeImpl) the DRS which can execute a given RB (i.e. its Reduction Recipe) and submits that RB to a pool of Wrappers (DRSWrapperSet) for execution. That is, DRSs are only accessed through their respective Wrappers which provide a clean, unique interface to any DRS.

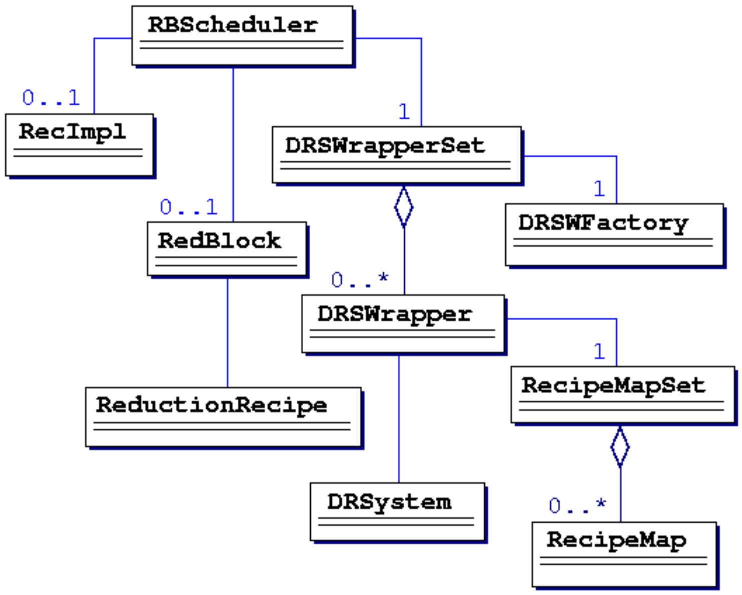

Figure 1. Main ReductionBlock Scheduler.

RecipeImpl class: RecipeImpl defines the implementation of a Reduction Recipe in a DRS. It provides a list of one or more DRS (in order of preference) for executing the recipe. In case of a DRS list the choice of a specific DRS depends upon availability of that DRS and/or the required hardware environment, e.g. parallel system.

RedBlock class: RedBlock combines for a reduction task the Reduction Recipe name and all input, reference and result frames, as well as all other parameters required by the Recipe's signature. Furthermore, a Reduction Block has a priority for execution and may depend on other Reduction Blocks. The Reduction Blocks are independent entities, i.e. all input parameters must be available and no side effects are possible.

ReductionRecipe class: ReductionRecipe defines a data reduction algorithm including its signature (i.e. the detailed list of all input parameters and the result(s) it produces) and return values.

4.2. DRSWrapper Package

DRSWrapperSet class: DRSWrapperSet contains a collection of active DRS represented via their wrapper. At startup an initial list of available wrappers (and their DRS) exists but it can dynamically create new wrappers using a factory class.

DRSWFactory class: DRSWFactory creates new DRS wrappers according to a set of specifications.

DRSWrapper class: DRSWrapper serves as adaptor for a given DRS. It provides the mechanism to convert generic RB specifications to individual scripts and commands of that DRS. For each DRS there must exist a DRSWrapper and it is the only main class in the pipeline system which needs specific knowledge about the specific environment of a given DRS.

DRSystem class: DRSystem is an interface class connecting a wrapper to its DRS. The class handles all communication to the DRS.

RecipeMapSet class: RecipeMapSet contains the set of all Recipe Maps for a given DRS. For each recipe (of an RB) it locates the corresponding Map and requests it to create the final script.

RecipeMap class: RecipeMap defines additional mapping for each recipe to the corresponding DRS script.

5. Implementation

The design of the RBS was based on UML and we used Together/J (from Object International) as environment.

The design document is currently published internally at ESO and will be finalized after the cycle of peer feedback and comments is terminated. The code for the new RBS and Wrapper package will be written in JAVA 1.2 (and SWING for any GUI). The `org.eso.fits` package will be used for access to FITS files.

The main target OS will be HP-UX (the hardware used at the VLT) but the code will be tested on Solaris and Linux as well. The first two DRSWrappers will be developed for MIDAS and Unix shell.

References

Grosbøl, P., Peron M. 1996, in ASP Conf. Ser., Vol. 125, Astronomical Data Analysis Software and Systems VI, ed. G. Hunt & H. E. Payne (San Francisco: ASP), 22

Grosbøl, P., Banse, K., Ballester, P. 1998, in ASP Conf. Ser., Vol. 172, Astronomical Data Analysis Software and Systems VIII, ed. David M. Mehringer, Raymond L. Plante, & Douglas A. Roberts (San Francisco: ASP), 151