IMT
EPFL
ASTRON
MPIA Heidelberg
Observatoire de Genève
University of Leiden/NOVA

Astrometric Survey for Extra-Solar Planets with PRIMA Data Reduction Library

Design

Technical Report

Doc. No.	UL-TRE-AOS-15752-0010
Issue	0.2
Date	24 sep, 2004

	Name	Date	Signature
Prepared	J.A. de Jong	September 24, 2004	
Approved	N.N.		
Released	N.N.		

Change Record

Issue	Date	Section/Parag. affected	Reason/Initiation/Documents/Remarks
0.1	14-sep-2004	all	created

Contents

List of Figures

List of Tables

1 INTRODUCTION

1.1 Scope

This document describes the design of the PRIMA Data Reduction Library (DRL). It should be read by anyone involved in the development of the DRL.

1.2 Overview

The Data Reduction Library (DRL) provides the software for the proper reduction and calibration of the PRIMA astrometric data. It consists basically of a pipeline reduction package for online and offline batch reduction and a Data Analysis Facility (DAF). Both the pipeline and DAF share the same algorithm libraries.

The ESO compliant pipeline part of the DRL will reduce and calibrate single observation runs. The Data Analysis Facility (DAF) will do the integral analysis of the complete dataset at any given stage. The angular separations must be solved as function of at least the parallax and proper motion. Furthermore, a large number of unknown and known systematic errors on the optical path difference need to be derived from all the available data. The fitted errors must be adjusted everytime new data arrives into the system and the parameter space may be subject to change when new trends in the data are discovered. This error analysis and correction will be the main task of the DAF. The DAF needs to provide the calibration data to the pipeline.

The DRL should provide at all times a consistent and verifyable reduction of the data, while allowing for regular changes in the reduction and analysis methods. This requires a design with a robust base system and a flexible top layer. This top layer should consist of APIs which make sure that every addition to the system is standardized and consistent.

The DAF design follows a client-server approach in which the main user interface runs inside a webbrowser. Only the interactive command console runs on the server. This has the advantage that all data can be stored and processed on a single (cluster of) computer(s). No installations on the client side are necessary, except for some browser plug-ins. Developing a web based user interface is nowadays even easier than developing a standard GUI application. Web application servers like Tomcat (AD 1) take care of user logins and the actions from the user on the system can be controlled better through a web interface. A client-server approach will anyway be required to ensure a controlled interaction with the database: there has to be a middleware portal between the clients and the database.

Following the argumentation and requirements above we split the DRL in the following hierarchical list of modules:

- *Common modules* (used by ESO pipeline and DAF)
    1. **Mathematical and Astronomical Library (MAL,2.1):** Provides all the routines for the numerical computations (fitting, FFT, optimization, interpolation, etc.), as well as generic astronomical algorithms.
    2. **Data Reduction Algorithms (DRA,2.2):** The core algorithms for the astrometric data reduction. Must be integrated in the ESO compliant pipeline.
    3. **FITS I/O Library (FIL,2.3):** Provides standardized functions for dealing with VLTI files and generic tools for flexible handling of any kind of FITS files.

4. **Configuration files (CF,2.4):** All configurable parts of the system are defined in an extended set of well designed configuration files.
5. **Common Tools Library (CTL,2.5):** General tools which have to be shared by both the pipeline and DAF.

- *Pipeline modules* (form the ESO standard pipeline)

1. **Plugin interfaces to DRA routines (PDRA,3.1):** Connect the DRA routines to the pipeline infrastructure.
2. **Pipeline Recipe Executor (PRE,3.2):** Executes the recipe on a given set of data.
3. **Pipeline Configurator (PC,3.3):** Creates a new pipeline recipe based on the configuration files.
4. **Calibration Matrix Interface (CMI,3.4):** Library for reading a calibration matrix and applying the calibrations.

- *Non-interactive DAF modules*

1. **Database (DB,4.1):** Contains all information about the raw and intermediate data: FITS headers, dependencies between steps, results of computations and references to intermediate files.
2. **Analysis Chain Manager (ACM,4.3):** Sets up the data analysis chains for the input data according to the available recipes. Manages branches of different recipes for trying out algorithms.
3. **Sequencer (SEQ,4.4):** Executes the DRA and TA and stores the results in the database.
4. **Calibration Engine (CE,4.5):** Generates the calibration matrix (CM) for external usage in the archive and online pipelines.

- *Interactive DAF modules*

1. **Trend Analysis (TA,4.2):** Set of tools and algorithms for identifying and fitting trends and systematic errors in the data.
2. **WWW User Interface (WUI,4.6):** Web application server which provides the standard user interfaces.
3. **Visualization Tool (VT, 4.7):** Library for drawing standard scientific diagrams and displaying images.
4. **Interactive Console (IC,4.8):** Provides a interactive command prompt in Python to experiment with the data and create prototype scripts for the reduction or analysis steps.

The relations between the DRL modules is shown in Fig. 1. The code for the modules DB, ACM, SEQ and WUI can be derived from the pipeline package GANDALF(AD 2) which has been developed in Leiden.

1.3 Programming Languages and inter-language interfaces

The DRL performs tasks of different levels of architectural complexity. For each level of complexity the most suitable programming language should be chosen. These language must also interface easily and logically with each other considering that the scientific algorithms must be available in

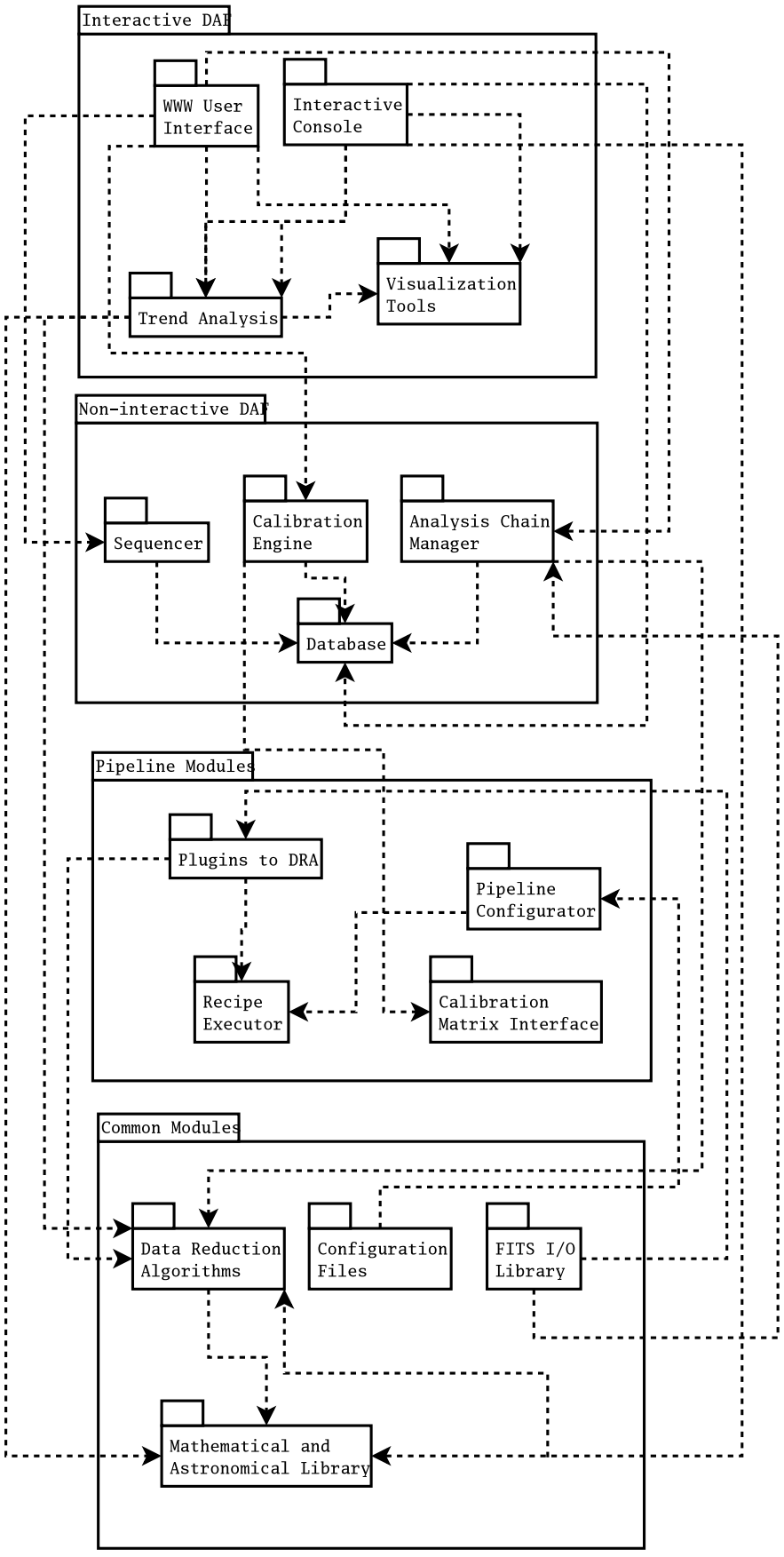

Figure 1: Overview of all DRL modules and their relations.

all languages at all times. Furthermore, some modules remain relatively stable over a long time while others contain experimental code which likely changes on a timescale of days or weeks. Finally, requirements on certain modules lead also to obvious language choices (e.g. the ESO pipeline standard requires C and visualization in a web interface can by far most easily be achieved with Java applets). The languages C, Java and Jython provide a very complete base for all these requirements. Jython (, AD 3 AD 4) is the Java implemtation of Python, which gives instant access to all Java classes from an interactive scripting environment. This is very useful for experimental coding in the Interactive Console (IC, see 4.8). Java in turn has a good two way interface with C through its JNI (Java Native Interface, AD 5) technology. Considering all this the following language usage should be optimal:

Language	*Modules*
C	All common and pipeline modules
Java	Almost all modules of the DAF, interfaces with common modules
Jython	Interactive Console of the DAF, plugins for the Trend Analysis

1.4 Applicable Documents

[1] "Jakarta Tomcat project," *http://jakarta.apache.org/tomcat/index.html* , 2003.

[2] J. A. de Jong, R. Bacon, E. Emsellem, J. Falcón-Barosso, R. McDermid, and R. F. Peletier, "GANDALF - a generic system for pipeline data reduction with interactive analysis," **in preparation**, 2004.

[3] "Overview of Jython documentation," *http://www.jython.org/docs/index.html* .

[4] S. Pedroni and N. Rappin, *Jython Essentials*, O'Reilly & Associates Inc., 1st ed., 2002.

[5] " Java Native Interface ," *http://java.sun.com/j2se/1.5.0/docs/guide/jni/index.html* , 2004.

[6] "The GNU Scientific Library," *http://www.network-theory.co.uk/gsl* , 2004.

[7] K. Banse, P. Ballester, C. Izzo, *et al.*, "The Common Pipeline Library: standardizing pipeline processing," *Proceedings of the SPIE on Optimizing Scientific Return for Astronomy through Information Technologies* **5493**(34), 2004.

[8] " The Starlink Project ," *http://www.starlink.rl.ac.uk* , 2004.

[9] T. McGlynn, "Java FITS package nom.tam.fits," *http://heasarc.gsfc.nasa.gov/docs/heasarc/fits/java/v0.9* , 2003.

[10] P. Grosbol, "Java FITS package org.eso.jfits," *http://www.hq.eso.org/∼pgrosbol/fits_java/jfits.html* , 2003.

[11] R. L. Poole, F. Delplancke, and R. Tubbs, "Phase referenced imaging and micro-arcsecond astrometry: its errors and the expected quality of their suppression," *Proceedings of the SPIE conference on New Frontiers in Stellar Interferometry* **5491**(64), 2004.

[12] " Servlet specification 2.3 ," *http://java.sun.com/products/servlet/2.3/javadoc/index.html* , 2003.

[13] "The Scientific Graphics Toolkit," *http://www.epic.noaa.gov/java/sgt/index.html* , 2004.

1.5 Acronyms

ACM	Analysis Chain Manager
AOS	Astrometric Operations and Software
API	Application Programmer Interface
CE	Calibration Engine
CF	Configuration Files
CTL	Common Tools Library
CM	Calibration Matrix
CMI	Calibration Matrix Interface
DAF	Data Analysis Facility
DB	Database
DRA	Data Reduction Algorithms
DRL	Data Reduction Library
ESO	European Southern Observatory
FIL	FITS I/O Library
FSU	Fringe Sensor Unit
IC	Interactive Console
IDL	Interactive Data Language
JNI	Java Native Interface
MAL	Mathematical and Astronomical Library
PC	Pipeline Configurator
PDRA	Plugin Interfaces to DRA routines
PRE	Pipeline Recipe Executor
SEQ	Sequencer
TA	Trend Analysis
VT	Visualization Tool
WUI	WWW User Interface

2 Common modules design

2.1 Mathematical and Astronomical Library

A number of libraries will be collected to provide the numerical algorithms required for the project (if possible open source). Also many astronomical algorithms are needed for ephemeris and Earth rotation related computations. The following libraries are currently being investigated:

1. The GNU Scientific Library (AD 6)
2. FFTW (Fastest Fourier Transform of the West)
3. CPL (AD 7)
4. MIDAS libraries
5. Starlink libraries (AD 8)

Libraries may be added for specialized algorithms during the project. Java wrapper classes need to be developed to provide easy access from the DAF environment. The development of these classes must be easy and quick enough to do on-the-fly when a function is needed.

2.2 Data Reduction Algorithms

This module contains all the algorithms for the pipeline reduction. The module will be used in the DAF to get intermediate reduction data in the database and to allow for the trend analysis module to redo reductions in order to improve the calibration. The module has to be fully compliant with the ESO pipeline standards (coded in ANSI-C and compatible with the Common Pipeline Library (CPL, AD 7)). For each step an algorithm will be developed and an entry will be created in the reduction recipe configuration file. The DAF takes care of the database table and associations. Figure 2 shows a preliminary diagram of the reduction recipe.

2.3 FITS I/O library

This library should have different categories, depending on the required flexibility of the FITS file handling. For standardized VLTI/PRIMA FITS files a CFITSIO based library will be developed based on the standard ESO VLTI library. However, in the experimental environment of the TA and IC the format of the FITS file must be auto-detected. The Java packages nom.tam.fits (AD 9) or org.eso.jfits (AD 10) can be used for this generic object oriented handling of FITS files. The latter will have preference once it is complete, since it is developed by ESO. However, according the current documentation the ESO package is not yet able to write FITS binary tables. Also, row-by-row reading of tables seems to be impossible with the ESO package.

2.4 Configuration files

2.5 Common Tools Library

Contains generic support tools for all parts of the system, such as a logging system, filename and directory conventions.

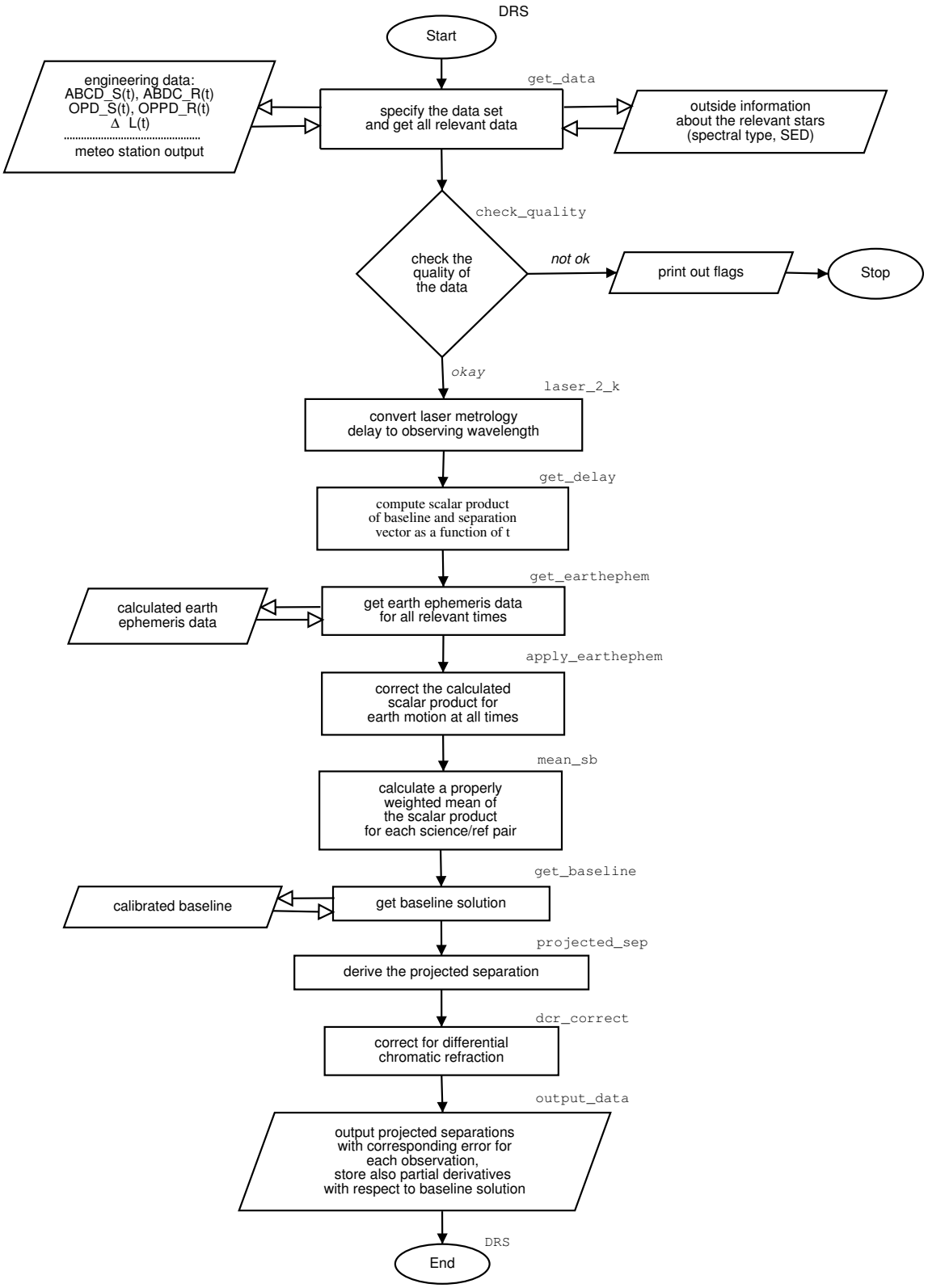

Figure 2: Preliminary diagram of the reduction recipe.

3 Pipeline design

3.1 Plugin interfaces to DRA routines

3.2 Pipeline Recipe Executor

3.3 Pipeline Configurator

3.4 Calibration Matrix Interface

4 Data Analysis Facility design

4.1 Database

The database contains at all times all information about what has been done with the raw data in order to obtain the scientific data product. So, a copy of the database should be sufficient to recreate the science products from the raw data. In order to make sure that the database is always complete, each analysis step should always finally be run through the standard user interface. Experimental analysis done within the interactive console can never be registered and lead to scientific results.

The database cannot have a static model, since changes in the analysis recipe have to be incorporated. These changes will inevitably lead to new tables and alterations in the relations between the tables. Therefore, we describe here only an abstract model. The actual model will be maintained by the ACM based on the configuration files which describe the current data analysis chains. Figure 3 shows the generic database model. The database will contain the following table groups:

Raw Data tables These tables contain information about the input data. This includes the organization in Observation Blocks and of each raw data file the date, time, type of observation and other relevant FITS keywords (to be specified in a configuration file). The tables **ObsBlock**, **Collection** and **RawData** belong to this category. The amount of RawData tables is equal to the amount of input sources. The collection table is used to treat a number of raw data records as a single entity (e.g for taking an average).

Command and parameter tables Each analysis step has an associated command in the **Command** table. Each command has associated parameters in the **Parameter** table.

Analysis step tables Each reduction or analysis step has its own table in which the dependencies on other steps and results are stored. These tables are coupled to algorithms from either the DRA or TA. Two such tables are shown in Fig. 3 (AnalysisStepX boxes). The associated record in the **Command** table is used to execute the algorithm. The names of the produced output files are stored in the **OutputFile** table.

Source tables Contains the astrophysical information about the science and calibration sources (e.g. color, spectral type, orbital elements, proper motion, parallax). In Fig. 3 only one catalog is shown, but several catalogs are possible. Raw data of science or calibration source is always associated with a record in the one of the catalog tables.

Administration tables Contain configuration information as obtained from the configuration files. They are mainly required for integrity checks. Among these tables are the **CommDef** and

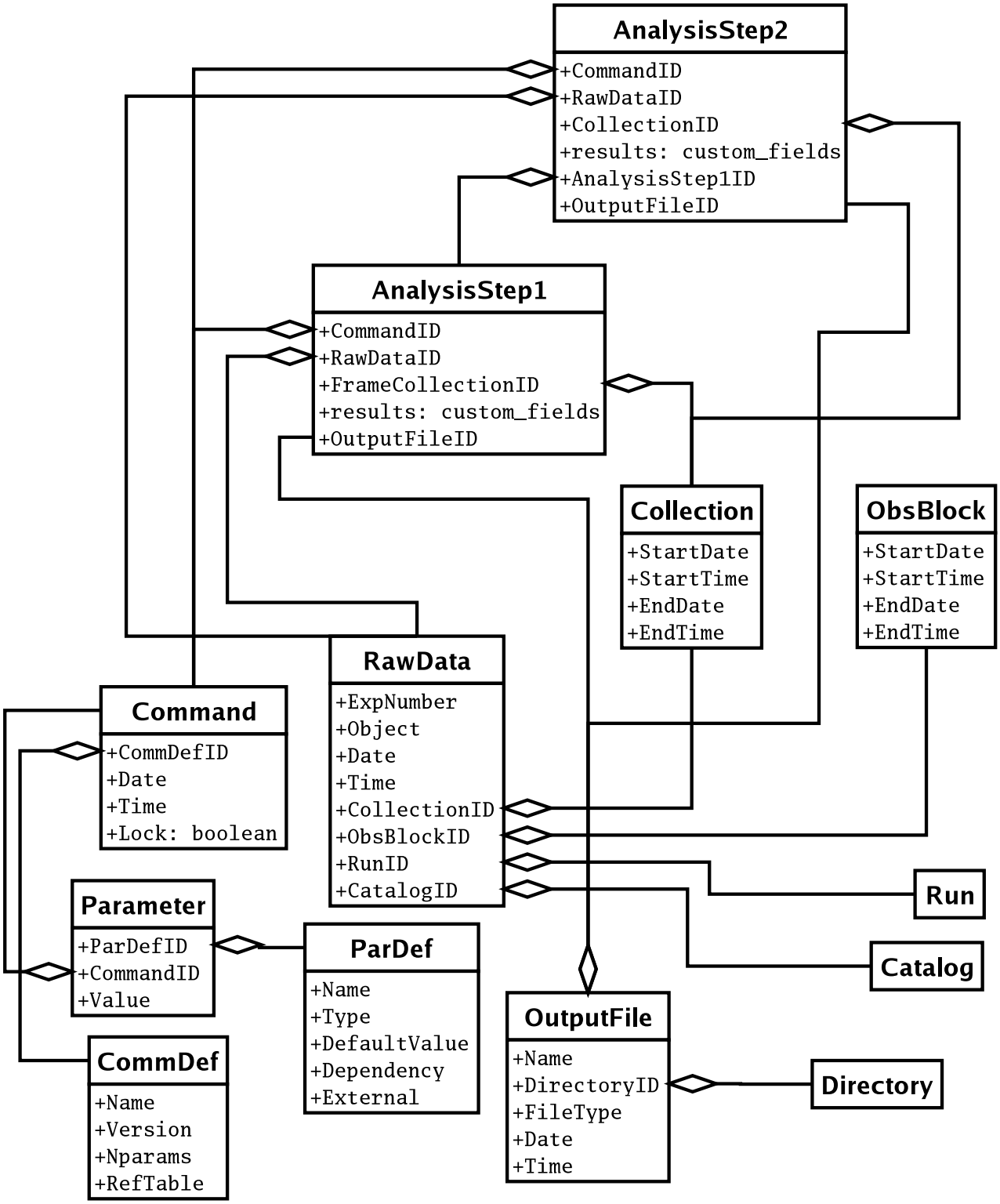

Figure 3: Generic database model for the DAF. The arrows show where the IDs of records are inserted in related tables.

ParDef tables which define the commands and their allowed parameters. There is also a **Directory** table which contains the available output directories.

The SQL queries will be concentrated as much as possible in an object to relational mapping package with a uniform API. This allows for optimizing the queries without affecting higher level code.

4.2 Trend Analysis

This is the most important module of the DAF. It contains the algorithms and interactive tools for analyzing the complete dataset of projected angular separations in order to find systematic trends. Currently, most possible sources for systematic errors are being indentified by the PRIMA consortium (AD 11). This study will lead to a series of parameters on for which trends may have to be fitted, as well as how large the systematic errors due to these parameters may be. The following steps show a simple example of how such an analysis could go:

1. Select a number of calibration measurements of known visual binaries from a sky distribution which limits the number of possible dependencies as much as possible. E.g. when trends as function of altitude are to be found then the preferable two or more binary stars with similar colors and different declination should be selected. In this case different altitudes with the same azimuth can be compared to make sure that altitude trends are distinguished from azimuth trends.
2. Compute the expected projected separations with the orbital elements of the binary stars and the baseline at the times of the observations. The orbital elements are received from one of the astrophysical tables and updated each time a calibrator is observed.
3. Plot the difference between the observed and expected projected angular separation to visually identify trends and try to fit the trend with a fitting routine.
4. Repeat from step 1 for many other selections which emphasize on other parameters.
5. Select a number of parameters for which clear trends are visible.
6. Do a multidimensional fit on these parameters with their expected boundary conditions.
7. Write the coefficients into the database for later use by the Calibration Engine.

In practice this analysis will probably become much more complicated. Especially for the selection of proper datasets and disentangling the trends advanced optimization and search techniques may have to be utilized. Furthermore the found trends will be fed back into the reduction to provide a better calibration. After this calibration the trend analysis may have to be repeated to refine it.

APIs will be provided for the plug-ins (see Sect. 4.6) and interface to the Sequencer (see Sect. 4.4).

4.3 Analysis Chain Manager

The Analysis Chain Manager (ACM) takes care of filling the analysis step records, which contain the associations with other steps, input parameters and results. Altogether these step records will form the analysis chain which has to be followed for a particular observation. It classifies the raw data

from the FITS headers and creates for each type of observation appropriate records in the database. Classification routines may be obtained from the ESO Common Pipeline Library (CPL, AD 7).

The ACM is also responsible for keeping the DB consistent at all times. This means that the appropriate records and dependencies must be removed when data has been invalidated. Also alternative raw data must be searched for in order to ensure that there is always a complete chain from the raw data to the scientific data product.

4.4 Sequencer

The Sequencer is responsible for executing the analysis steps and updating the database with their results. Each algorithm interfaces with the Sequencer through an API which specifies the following:

1. Input parameters for optimization of the algorithm.
2. Reference files which the algorithm may require.
3. Results from previous steps which provide input to this step.
4. Where to put the results of the algorithm (files, database points).
5. Error conditions.
6. Plugin mechanism for the visualization tool.

This API will be implemented as an abstract Java class. The derived Java class will contain native methods for which a corresponding C template will be generated with the JNI tools. Calls to the C algorithm for the analysis step will be inserted in this auto-generated template. The advantage of this API will be that the infrastructure software can easily dynamically integrate new analysis steps and adapt the WUI.

The algorithms will be executed by the Sequencer in the following way:

1. A Command object is created for the top-level reduction step. This object contains the dependencies to the previous steps, a method `isReduced()` to check for these dependencies and an `execute()` method to run the algorithm from the API class.
2. Invoke the `execute()` method of this top-level method. This method in turn invokes the `isReduced()` method to check whether the step has been done.
3. The `isReduced()` method creates Command objects for each dependend step and invokes their `execute()` methods. These objects repeat this recursively until the last step has been reached.
4. The database also contains lock flags. In case a step is locked a wait loop will be entered until the lock has been released, which means that the other process has finished that particular reduction step.
5. A reduction or analysis step will be (re)calculated when:
   (a) A file or execution date is missing.
   (b) The result of one of the dependencies is newer than the result of the current step.

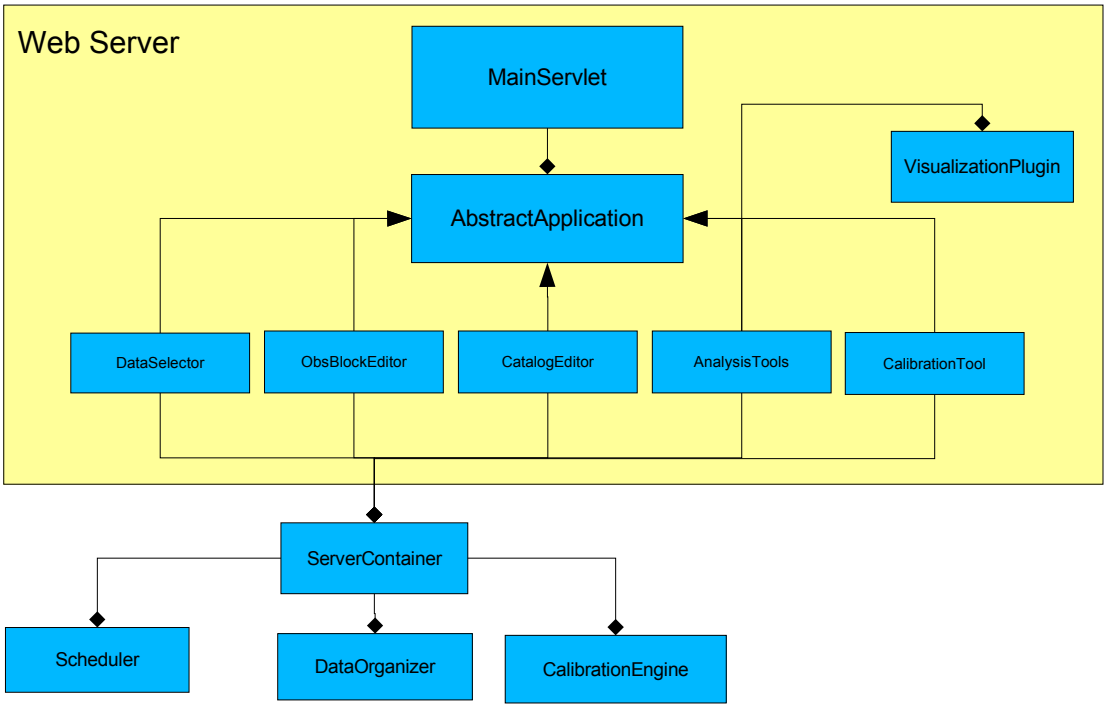

Figure 4: Class structure diagram for WUI, ACM and Sequencer

(c) The step has been invalidated by the ACM because parameters have been changed.

This scheme ensures that steps are only recomputed when necessary. Furthermore, the locking mechanism provides a means to run several reductions in parallel when multiple CPUs are available.

4.5 Calibration Engine

The calibration engine computes from the latest contents of the database a set of calibration files, which has to be used in the ESO pipelines. With these calibration files any dataset should be calibrated properly to the best knowledge available at the time, and within the parameter space of the observed calibrators. The calibration engine will use the same APIs as all the other reduction/analysis steps in the chain.

4.6 WWW user interface

All user interfaces are webbased. They will run on the Apache Tomcat web application server (AD 1), which implements the Servlet 2.3 specification (AD 12). The server runs several managers which manage the input data, intermediate data, running reductions and exporting the calibration data. It is accessed through a single servlet, which handles the user authentication, stores the user data and determines which application page should be displayed. All application pages are derived from a single abstract class which provides a uniform look and feel. The manager methods are called upon

user actions in the application framework. Fig. 4 shows a class structure diagram of the server and underlying software layer. The following classes are defined in this diagram:

MainServlet Handles the user authentication and determines which application to show, depending on the state of the current session of the user.

AbstractApplication Framework class for a uniform look and feel of the applications in the UI.

DataSelector Deals with the selection of the data to analyse in the current session.

ObsBlockEditor Examine the observation blocks and unselect bad data.

CatalogEditor Edits catalogs of science and calibration targets.

AnalysisTools Provides access to the complete reduction/analysis chain. Shows the parameters and plug-ins of the steps and starts/monitors/aborts the analysis of certain or all OBs.

VisualizationPlugin Base class for all visualization plug-ins.

CalibrationTool User interface to the calibration engine.

ServerContainer Collection of business logic classes to be used by the UI. An instance of this container is stored in the session of the user. The container contains the main classes of the different modules of the system.

4.7 Visualization Tool

The visualization tools will be derived from one or several open source scientific graphics tools. These tools must be available as Java packages and fully integratable in Java GUIs. Furthermore, they should provide all standard 2D visualizations, such as line, contour, grayscale/colour and scatter plots. Overlaying and combining diagrams should be easy. The Scientific Graphics Toolkit (AD 13) developed by NOAA seems currently to be the most complete package. Other packages are under development for the StarLink (AD 8) and Virtual Observatory projects.

4.8 Interactive Console

This is a scripting extension to the system which runs under Jython (Java implemntation of Python). It provides the ability to interactively analyse the data in the system and try out algorithms. An environment similar to IDL will be provided in which the same algorithms are available as for the non-interactive C and Java programs. There should be no extra overhead involved in the development of this module, because all used classes are already developed for other parts of the DRL. Jython just provides an easy interactive interface to these classes.

Since Jython compiles into Java scripts written in Jython can directly be plugged into the system by extending the Java API classes for the Sequencer (see 4.4). There is no interlanguage interface programming needed for this. This tight integration with the DAF gives a major advantage over e.g. IDL.

The interactive console will only get readonly access to the database, because it is much too dangerous to alter the database from an experimental environment. Write access to the database is anyway only allowed through methods provided by the Sequencer API.

___oOo___