PELICAN: Pipeline for Extensible Lightweight Imaging and Calibration.

# Introduction

This document describes the first stage design for the PELICAN software project. This project has the goal of delivering a lightweight, scalable imaging and calibration pipeline for aperture array stations for deployment in the near future on the SKA pathfinder and demonstrator telescopes.

The software is envisaged to consist of two separate elements: A data server for receiving and distributing visibilities generated by a station level correlator and a modular pipeline engine which carries out the post processing requirements.

This system will provide both a rapidly deployable solution for the short term requirements of delivering imaging pipeline for LOFAR stations as well as a solid, extensible base for future post-processing studies and developments. The modular pipeline design will be fully compatibility with the other major post-processing software developments in the radio astronomy community, namely the CASA.

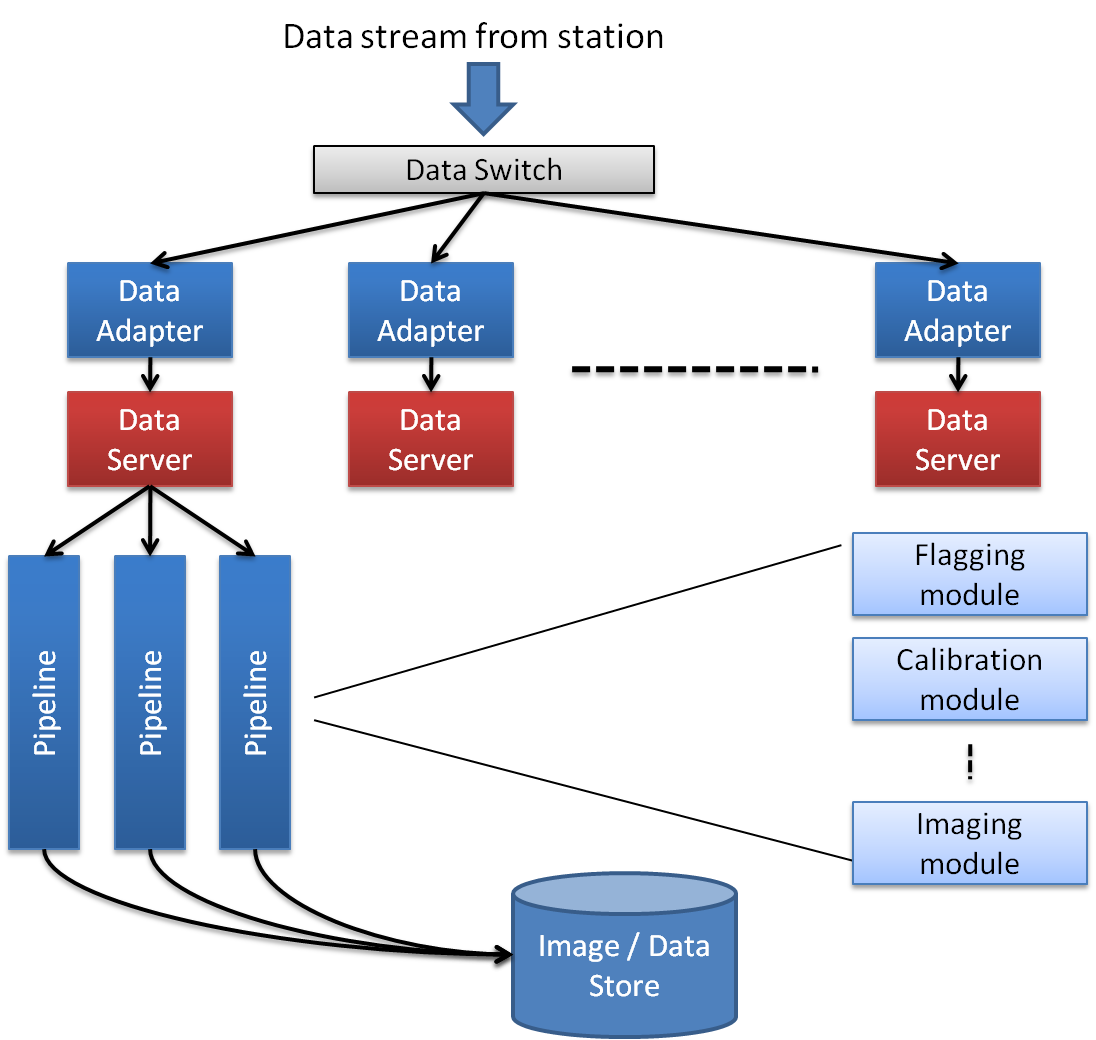


Figure : Envisaged top level system design for PELICAN. A stream of data from the station correlator is passed via a switch and a telescope specific data adapter to one or more data servers. Data servers then pass out sets of visibility data to a number of post-processing pipelines which perform a number of operations on the data sets (eg. Calibration and imaging). Data from the pipelines is then sent to a data store for further offline analysis.

# Design Strategy

## Testing

# Development Infrastructure

## Code Repository

The structure of the code repository level of structure (illustrated in Figure 3) contains subfolders for the main code areas of the design and each of these are explained in Section 4.

Figure : Second level structure

## Using the repository

## Typical usage pattern

# Code Structure

## Adapters

Telescope specific functional unit providing conversion of data formats for compatibility with the data server.

* Very lightweight modules.
* Input format defined by the telescope data stream, file etc.
* Output data format determined by the server data blob format.

## Data Server

Module to and serve data to the rest of the pipeline. In doing so this also defines the pipeline data format.

Must be able to handle and serve raw correlated station data as well as additional metadata required for pipeline such as station telemetry (eg. for flagging) and external sky models though two methods of data service

* “Online” visibility data which is unique per pipeline.
* “Offline” service data which is shared between pipelines.

## Pipelines

Connect modules together, defining data flow though a set of modules in the context of a pipeline driver.

## Modules (for the pipeline)

Provide functional units to the pipelines. Follow standard module interface inherited from the core. Modules on the current development plan include

* Imaging
* Calibration

### Module Configuration

Provided by an XML configuration file(s). An example configuration file might look like

<pipeline name=”test”>  
 <module type =”Imager::DFT” name = “imager1”>  
 <param .../>  
 <param .../>  
 </module>  
 <module ...>  
 </module>  
 ...  
</pipeline>

## Core (for the pipeline code base)

* Module base class.
  + Module configuration reader.
* Pipeline base class
  + Module factory (instantiating modules with configuration options).
* Data client.

## Utility

Utility classes used throughout pelican. For example:

* Configuration.
* Logging.
* Transport mechanisms for the data client & server.

## Data

Data classes used throughout pelican. For example:

* Data blob.
* Matrix.

# Code Design

## Coding conventions

# Testing

Testing is divided into three main areas each of which will be covered in more detail in the following sections.

* Pipeline tests
* Server tests
* Algorithm / Module tests

## Pipeline tests

Pipelines tests check functionality of the pipeline which primarily acts as a mechanism for execution of a collection of modules. Tests will include:

* Pipeline configuration test
* No-config test
* Inconsistency test
* Consistency test
* Module timeout test
* Empty pipeline test
* Verification of data flow

Pipeline tests will be expected to work with a TestDataClient object which can provide data for verification purposes.

## Server tests

Server tests provide methods for load testing and checking failure modes for the server and data client. The data client will be tested with a dummy data server and tests will include:

* No reply (from server) test.
* Inconsistent data test.
* Truncated data test.
* Incomplete data test.
* Data flow test
  + Checking if the telescope data is correctly received.
  + Checking if the service data is updated correctly

The data server tests will include:

* Dummy requests
* Chucking test (via data adapter)
* Buffer filling / empty / locking test (multiple requests threads).
* Load test (handling ~O(1000) requests at once)
* Chunking and associated buffer overflow test.

## Algorithm / Module tests

Algorithm and module tests will include both tests for module functionality the algorithms themselves (numerical accuracy, functional verification etc.). Tests will include:

* Memory leak testing (loop).
* Test of module configuration.
* Data input testing via test data set library).
* Data Set comparison test.

The test data library will provide test data sets (generated or by file etc.) to data sets suitable for testing the various classes of modules (image manipulation, uvw data manipulation etc). Test data sets might include:

* Null set
* Repeated patterns eg. 01010101...
* Synthetic (known result) data sets. eg. uvw set for image of a pelican!

# Deployment

We expect to deploy new pipelines by allowing the pipeline to read configuration data file from a shared file system.