1 VEGAS Manager Updates

1.1 Coherent Mode Backend Class

Implement a 'backend' class which implements the setups necessary to acquire load/configure and run the coherent pulsar modes. Requires a concept of 'who is the owner of the coherent mode roach'.

1.2 Incoherent Mode Backend Class

Likewise implement a incoherent mode backend class to load/configure and run incoherent bofs.

1.3 Add Parameters Unique to Pulsar Modes

Merge any parameters which are not already covered by the spectral line mode parameters.

1.4 Add Samplers for Pulsar BOF Signal Level

The vegasdm display uses VEGAS Manager samplers to display power levels. This would not define new samplers, but rather how the samplers are populated.

1.5 Update VEGAS Coordinator

Integrate new pulsar mode parameters (and any inter-bank dependencies) into the VEGAS Coordinator.

1.6 Update VEGAS Manager to use new API

This would update the manager code to use the component services which interface with the HPC data transport layer.

1.7 Revise Status Memory Interface to HPC

Update the shared memory status/configuration keyword interface with an interface to the value-keyword 'service'. (The service being hosted in the HPC program.)

1.8 Add Unittests for all Backend Classes

The DIBAS project has a number of backend unit tests which could be useful in the VEGAS Manager context. We should make use of this.

4 HPC Cleanup/Merge

4.1 Testing Strategies

How do we confirm the new codes get the same answers as the old code? I list this as a task because it requires some discussion. Of course "by observing X" is always an answer, but does not aid in test driven development.

What we are looking for here is a deterministic put X in get Y out type of test. Perhaps capturing some raw data, and running through the existing HPC and recording the output is the definative test. Alternatively a packet generator, which generates a known signature of data could be used. Even still, these are really system level tests.

It would be useful to get datasets of stage inputs and outputs. I think this could be derived (i.e. captured from) the current HPC, and used for refactoring tests. This requires some strategy.

4.2 Identify Common Code

Both guppi and vegas have a number of functions which are common and support some of the inner workings of both HPC programs. These should be merged at some level, and integrated into the updated framework.

4.3 Design Component and Support API

The idea here is to create an API for use by the system to configure, control, and run components in a consitent manner. Likewise, support routines which are agnostic of the details of implementation of the support routines should be defined. I have some ideas on this, but the details need to be filled in.

4.4 Prototype ZMQ Pipeline, Performance Analysis

There are some advantages to using a ZMQ pipeline for inter-thread and interprocess communications. However, what is not precisely known is "Will it be fast enough?". This task intends to outline the limitations by prototyping and measuring throughput. The decision from this effort will determine the implementation (but not interface) of the data exchange mechanisms.

4.5 Implement Component and Support API's

Implement the API's for a generic component and the supporting infrastructure.

4.6 Implement Keyword-Value Service

The current HPC systems rely upon a shared memory keyword-value in FITS format (with the limitations included) for configuration and status. This would be replaced by a much more flexible keyword-value service, accessible locally or remotely.

4.7 Revise Processing Stages to use API

This takes the existing processing stages and refactors them to use the Component and Support API's.

4.8 Unit Tests for Components

Create component-level unit tests to verify correct operation of the new components. This is where deterministic test data sets would be useful.

4.9 System Level Tests

With the revised code, test operation in each of the HPC modes (eg:HBW, LBW, coherent search, etc.).