Design and Implementation of the ACSIS Real-Time Data Reduction Pipeline

A.G. Willis

National Research Council of Canada

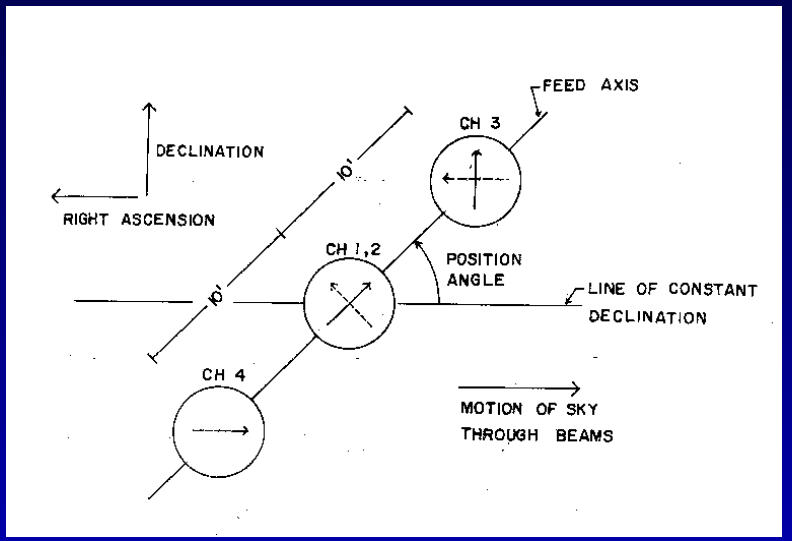
Herzberg Institute of Astrophysics

Penticton, BC, Canada

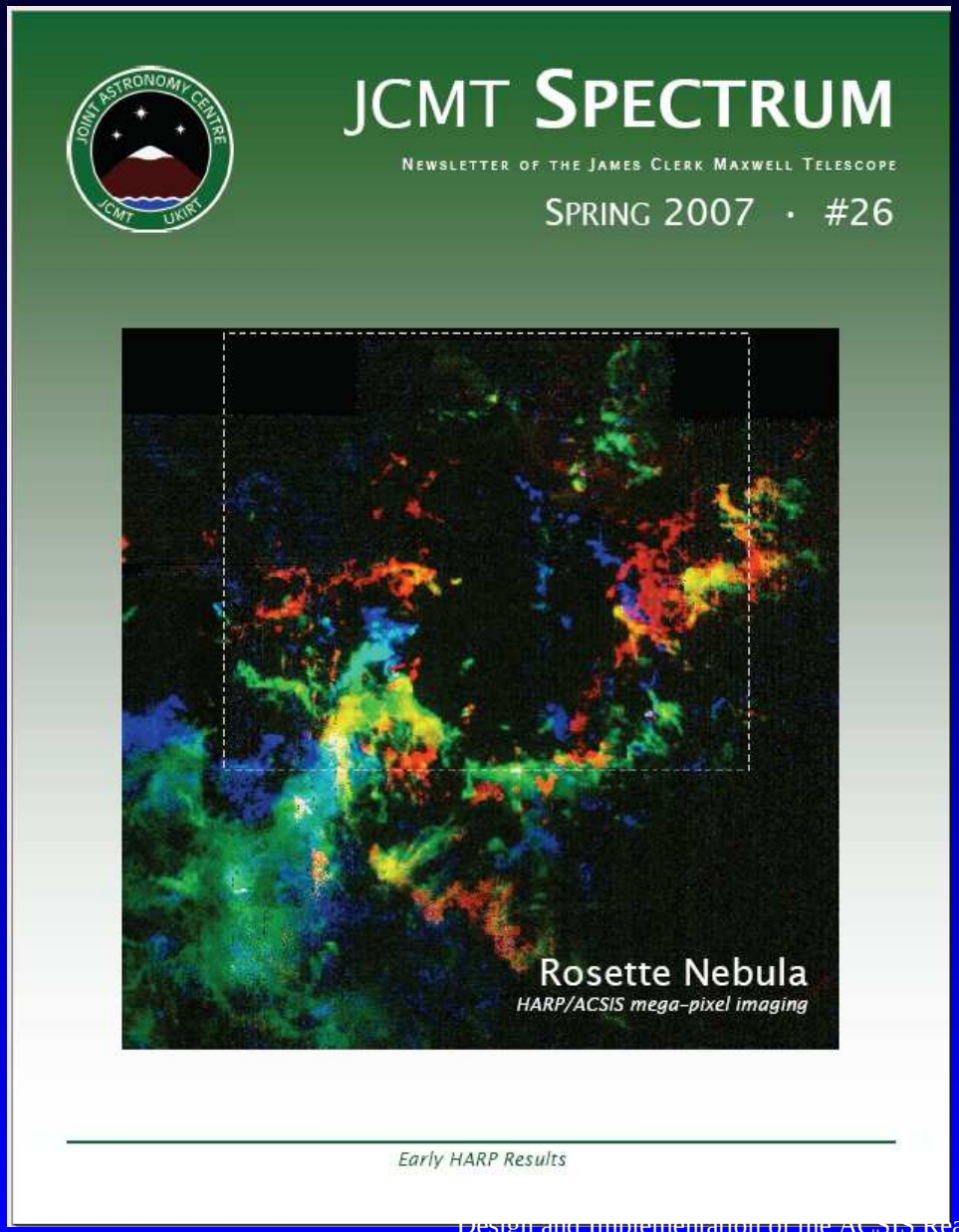
NRC·CNRC HERZBERG INSTITUTE of ASTROPHYSICS

History Repeats Itself

- 300 ft telescope with 3-feed focal plane array at 11 cm wavelength (1971)

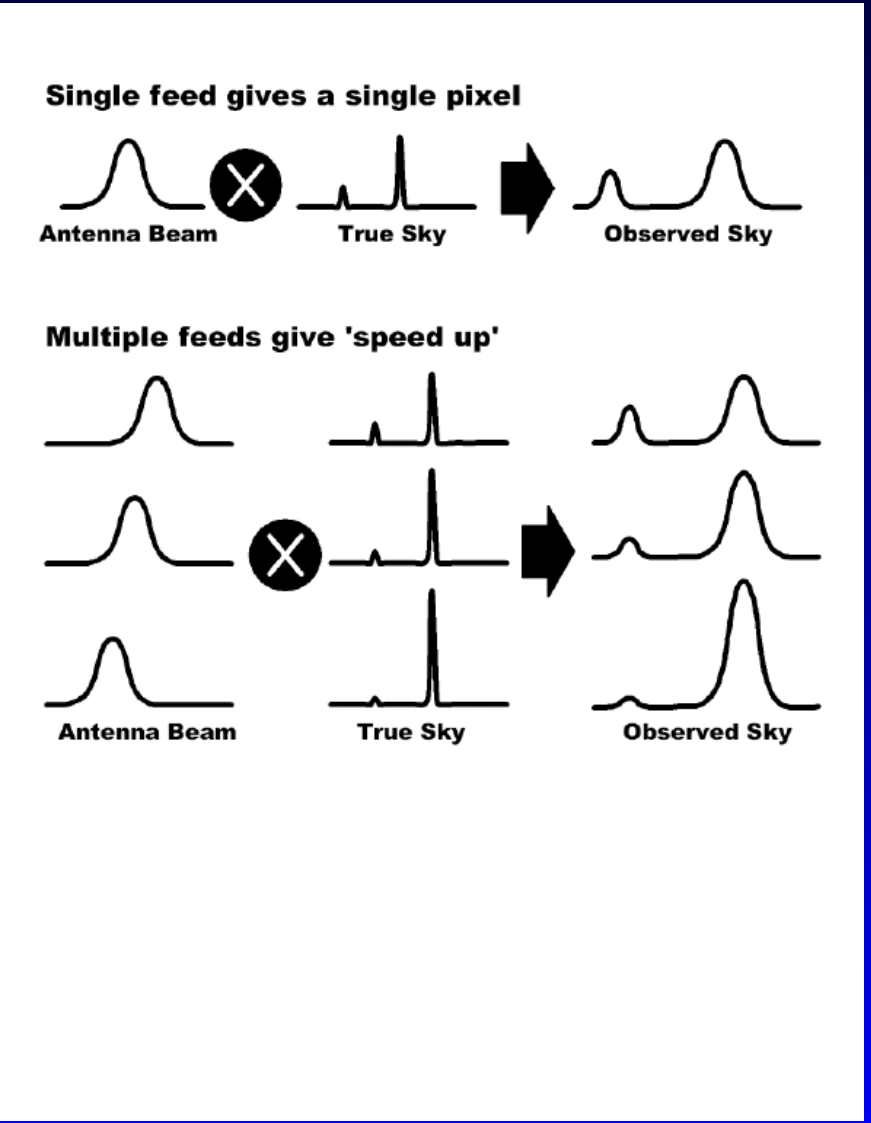


ACSIS Does Work!



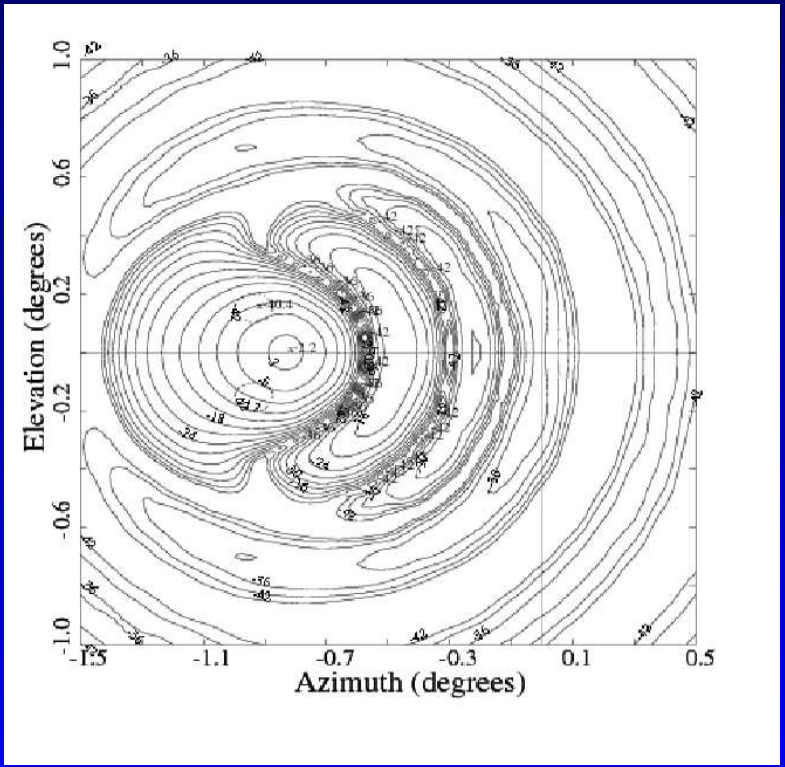
Why We're Here

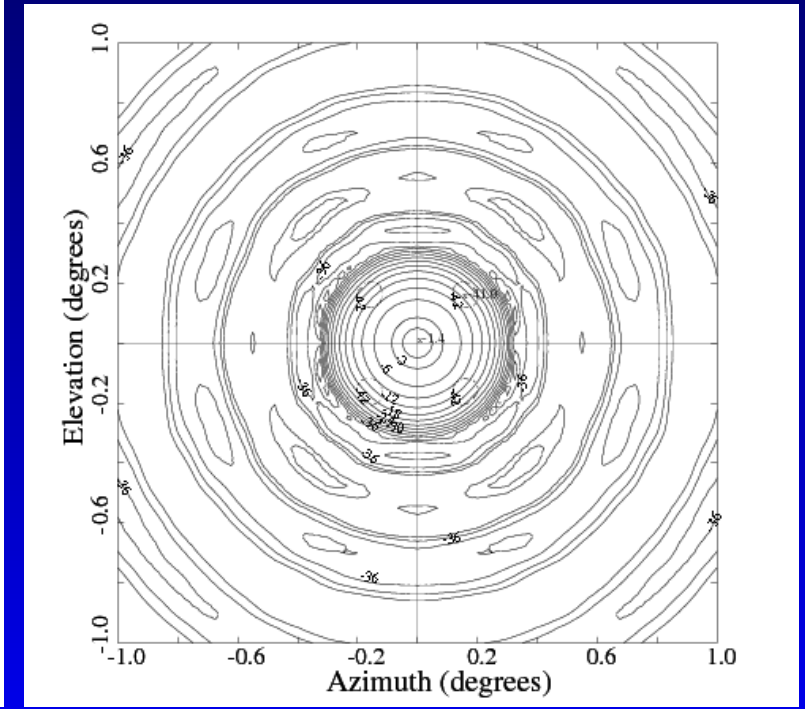
- Focal plane arrays give enormous observation speedup



But No Free Lunch

- Off-axis beams can have ugly shape and high instrumental polarization
- The figures show the central and one of the circumferential beams of the Parkes 21 cm multi beam system





Main ACSIS Configuration

- HARP receiver has 16 pixels
- 16×8192 channel mode with 50ms integration time produces 10 Mb of data per second
- originally we planned a distributed system for data processing on a small ‘beowulf’ cluster of 32 450MHz PCs
- final system had 10 dual processor 2GHz PCs
- parallel system needed because of
  - data rate vs available CPU speed
  - 32-bit memory limitations (image cube has to be spread over multiple machines)

ACSIS Computing Hardware

ACSIS Implementation

- use c++ class library provided by aips++
- client c++ programs are connected by glish scripting and messaging system
- communication takes place by means of events. Clients can send events directly to each other by ‘links’
- glish is a ‘good thing’ for this type of system
- see aips++ Note 230 ‘Advanced Programming with AIPS++ and Glish’ for a detailed discussion of distributed programming with glish
- system comes with simulation tasks for testing and debugging purposes

glish communication with client

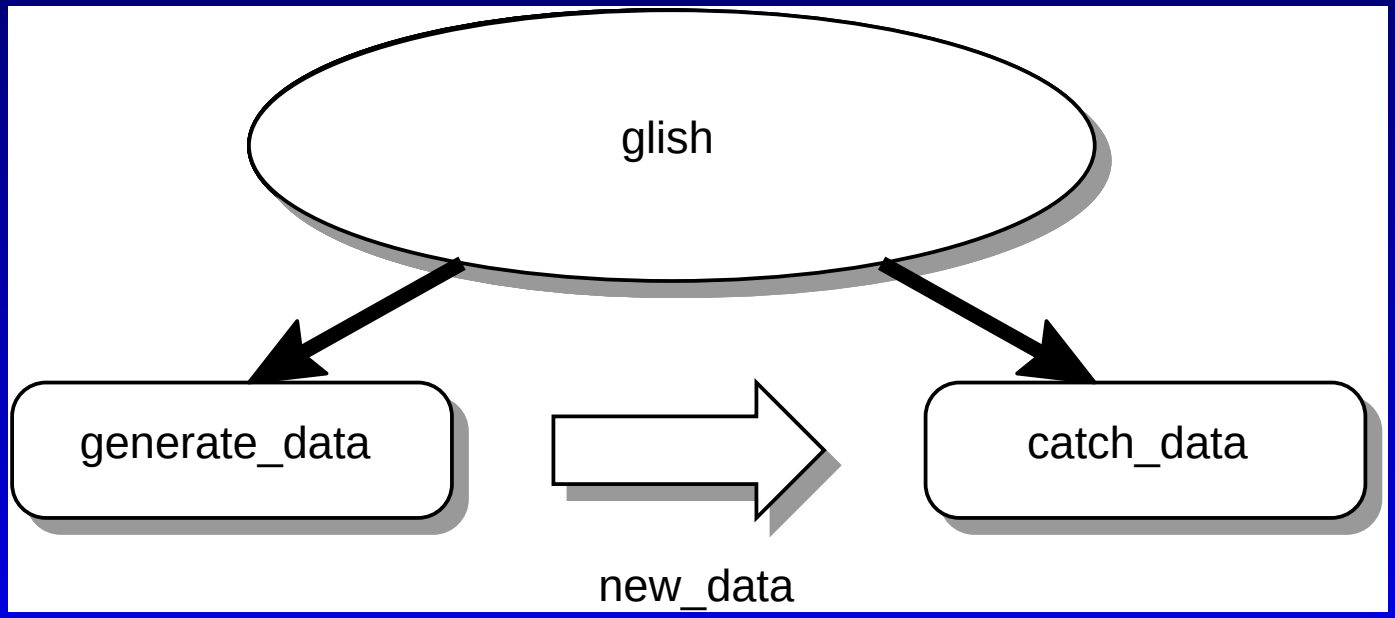
```
########## begin hello.g ###############
# a glish record containing the
# string 'hello world'
info := [=]
info.data_string := 'hello world'

# start c++ glish client on machine slave5
demo:= client("hello_world",host ="slave5")

# send this client the glish 'info' record
# by means of a hello event
demo->hello(info)
########## end hello.g ################
```

glish link

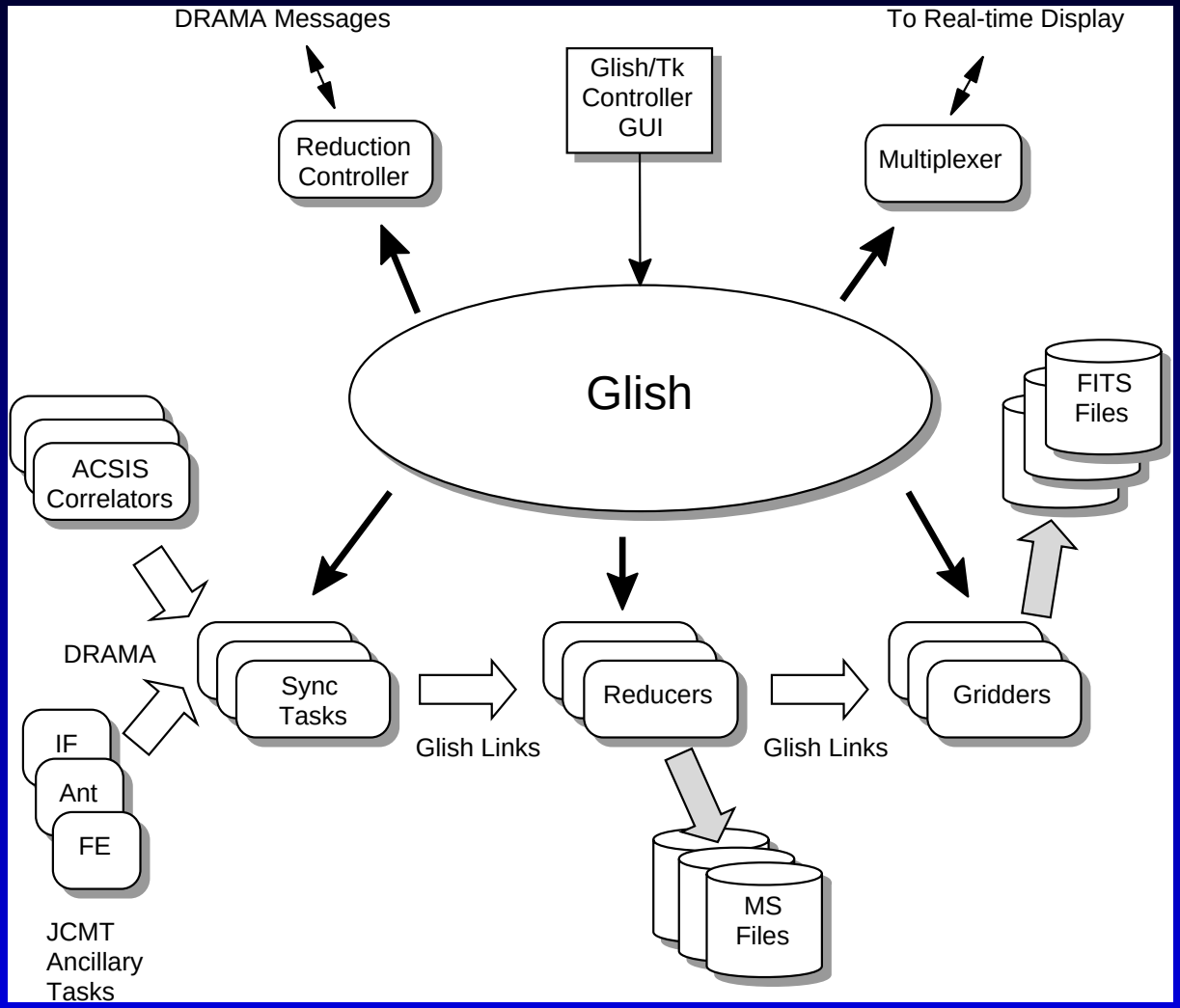
```
############## begin link.g ####################
a:= client("generate_data", host="slave1")
b:= client("catch_data", host="slave2")
link a->new_data to b->new_data
############## end link.g   ####################
```



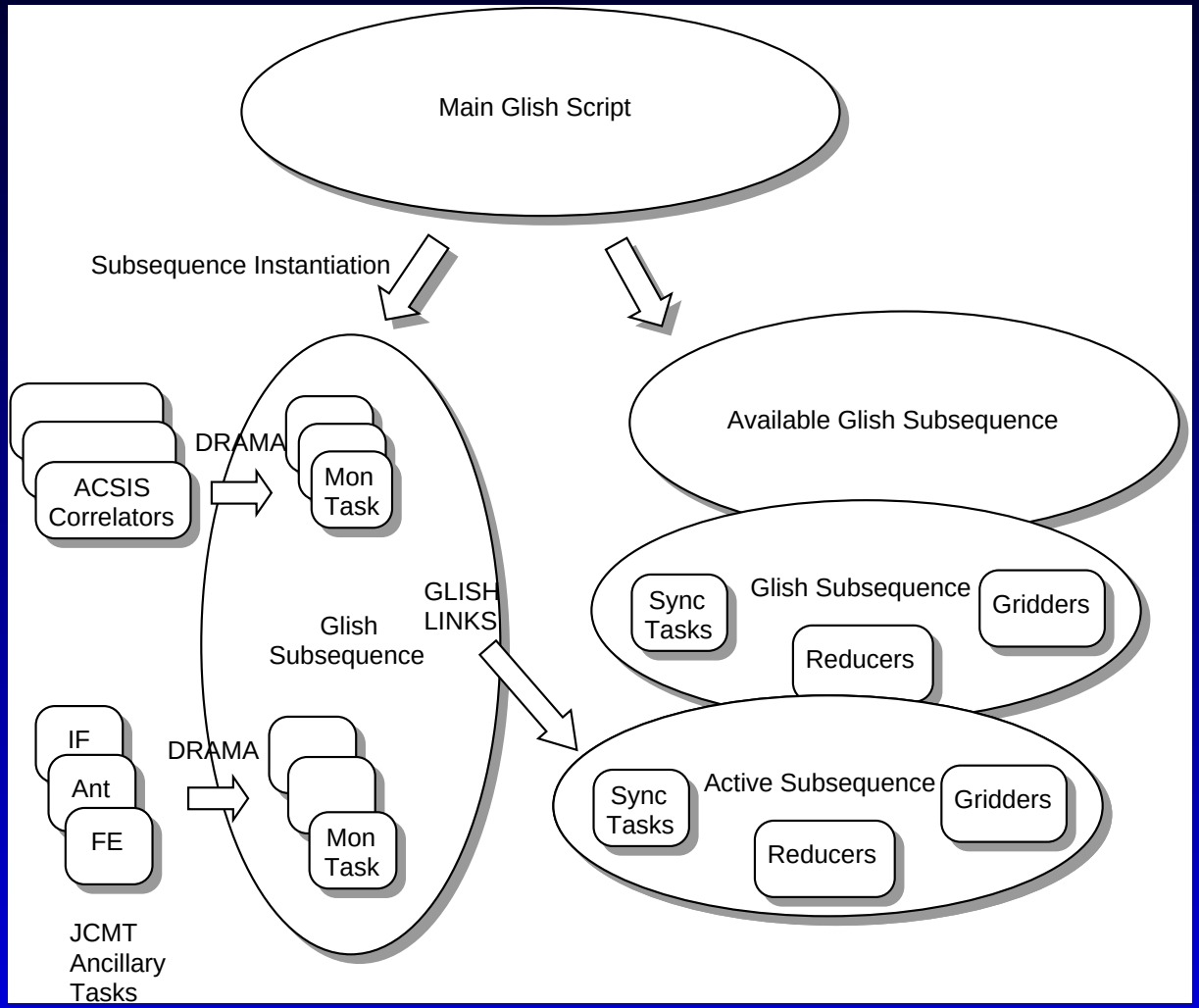
System Scalability?

- Yes!
- System can be configured to run on anything from a laptop to (at least) a 32 processor PC cluster
- Upper limit is unknown
- glish scripts read XML-based configuration files to load and link tasks together across the network
- Can be used with any feed configuration from 1 to n where n has a maximum of ??

Reduction System Overview



Pipeline Details



Main ACSIS Tasks

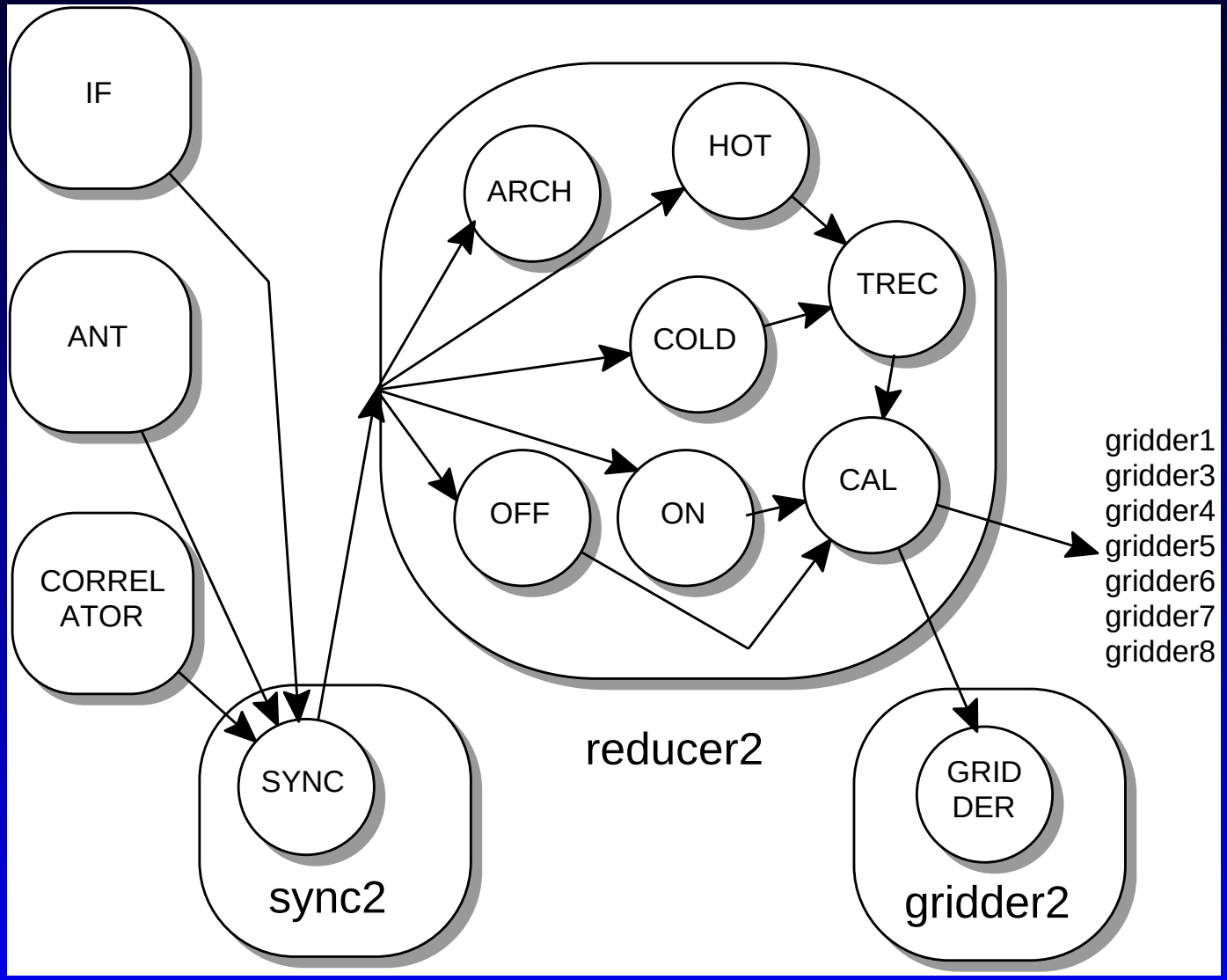
- Reduction controller: interface between the reduction system and the rest of the JCMT telescope control software
- Multiplexer: a shared glish client that can be connected to by multiple glish interpreters
- Sync task: synchronize data from a correlator crate with the ancillary data that are required to completely process a spectrum
- ReducerProcess: convert raw correlator lags (time domain) into calibrated spectra (frequency domain)
- Gridder: insert calibrated spectra into correct location of a 3-dimensional data cube

XML for Task Instantiation

- Glish scripts are essentially ‘fixed’.
- System configures itself for each individual observation by reading and processing XML files.

```
<process-init>
 <process name="sync2" exe="sync_task"
                          machine="slave2">
 </process>
 ...
</process-init>
```

Reducer Process Instantiation



ReducerProcess XML

```
<rt-process type="REDUCER">
 <red-object name="ON" type="reducer">
  <red-object-source-list>
   <string> SELECT FROM SYNC
      WHERE LOAD=="SKY" && ON_SOURCE</string>
  </red-object-source-list>
  ...
```

XML for ReducerProcess con't

```
  <red-object-recipe>
   <string>   corr_data.FLOAT_DATA </string>
   <function> rvalue               </function>
   <string>   SPECTRAL_WINDOW_ID   </string>
   <function> rvalue               </function>
   <function> window               </function>
   <function> fft                  </function>
   <string>   corr_data.FLOAT_DATA </string>
   <function> lvalue               </function>
  </red-object-recipe>
 </red-object>
 ...
</rt-process>
```

RPN Algorithm Equivalent

- push incoming data field `corr_data.FLOAT_DATA` on to the stack
- pop data from the stack, multiply it by an appropriate window function and push result
- pop data from the stack, FFT it and store it in the `corr_data.FLOAT_DATA` field
- forward the updated data record to other objects that have ON in their `<red-object-source-list>` definitions

Flexibility

- Reducer Processes are basically programmable RPN calculators for 8000 channel vectors
- Can write a new program for a new observing procedure
- System was delivered to the JCMT with six ‘standard’ observing recipes

System Output

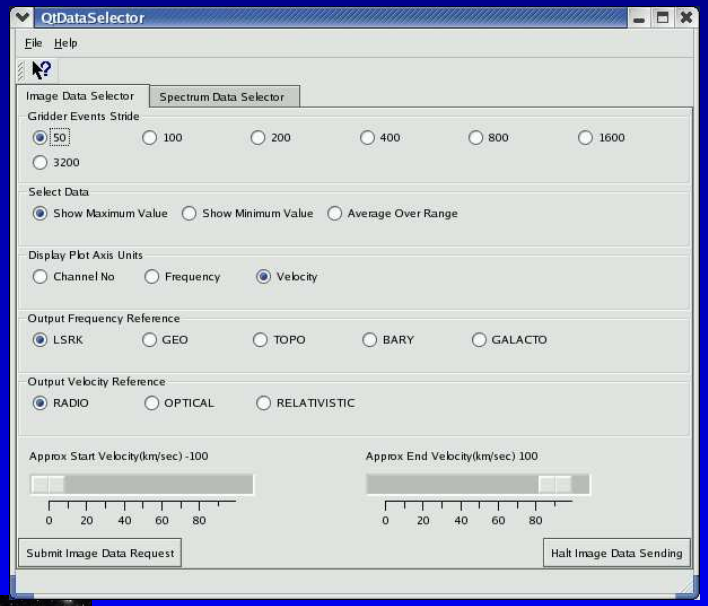
- Reducer Processes write raw input data to disk in the form of aips++ Measurement Sets
- Gridders write out (sub) data cubes as FITS files
- Data can be reprocessed with a ’replay’ script. This script can read in the saved MS data and re-reduce it with a modified recipe (e.g. with different baseline fitting regions or a different griding kernel)

Real Time Display Overview

- Real time displays available for
  - meta data
  - Reducer Process ‘nodes’
  - Gridders
- For meta data and Reducers, data are shown collectively, but can be examined on an individual basis
- meta data displays can run continuously
- new Reducer and Gridder displays are created for each observation
  - are created by XML specification
- displays provide immediate feedback during an observation

Data Selection GUIs

- GUI used to select data from either Reducer Processes (spectrum selector) or Gridders (image selector)
- Reducer Process ‘nodes’ can be examined individually (especially useful for debugging)
- GUIs and displays mostly use Qt widget library with QWT plotting widgets



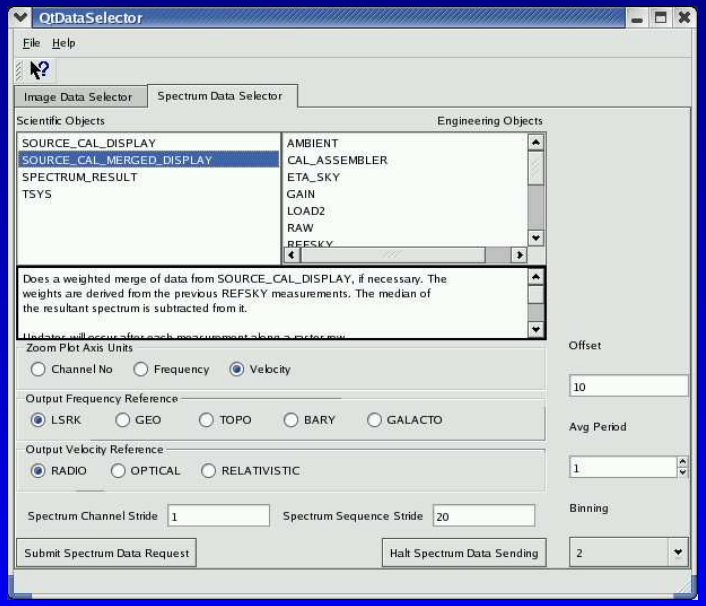
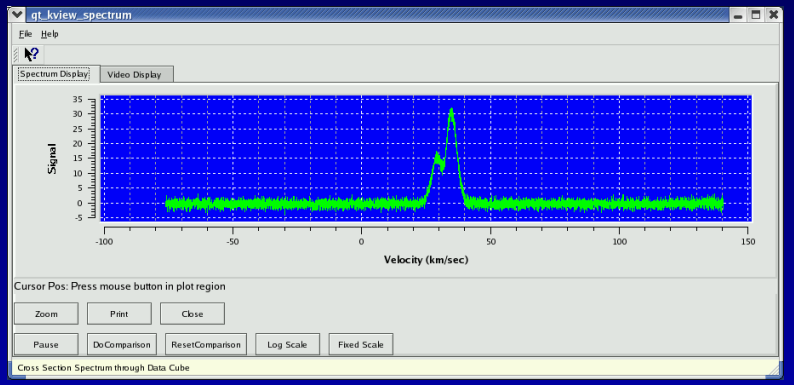
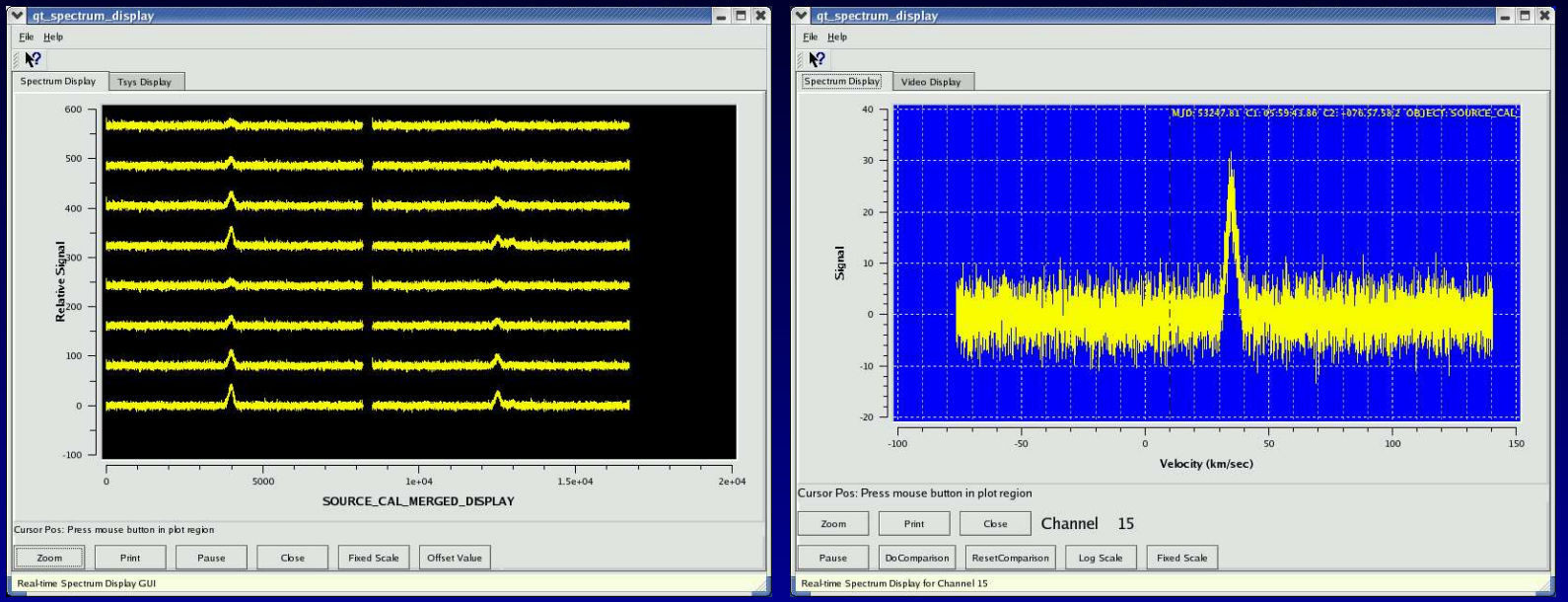


Image Displays



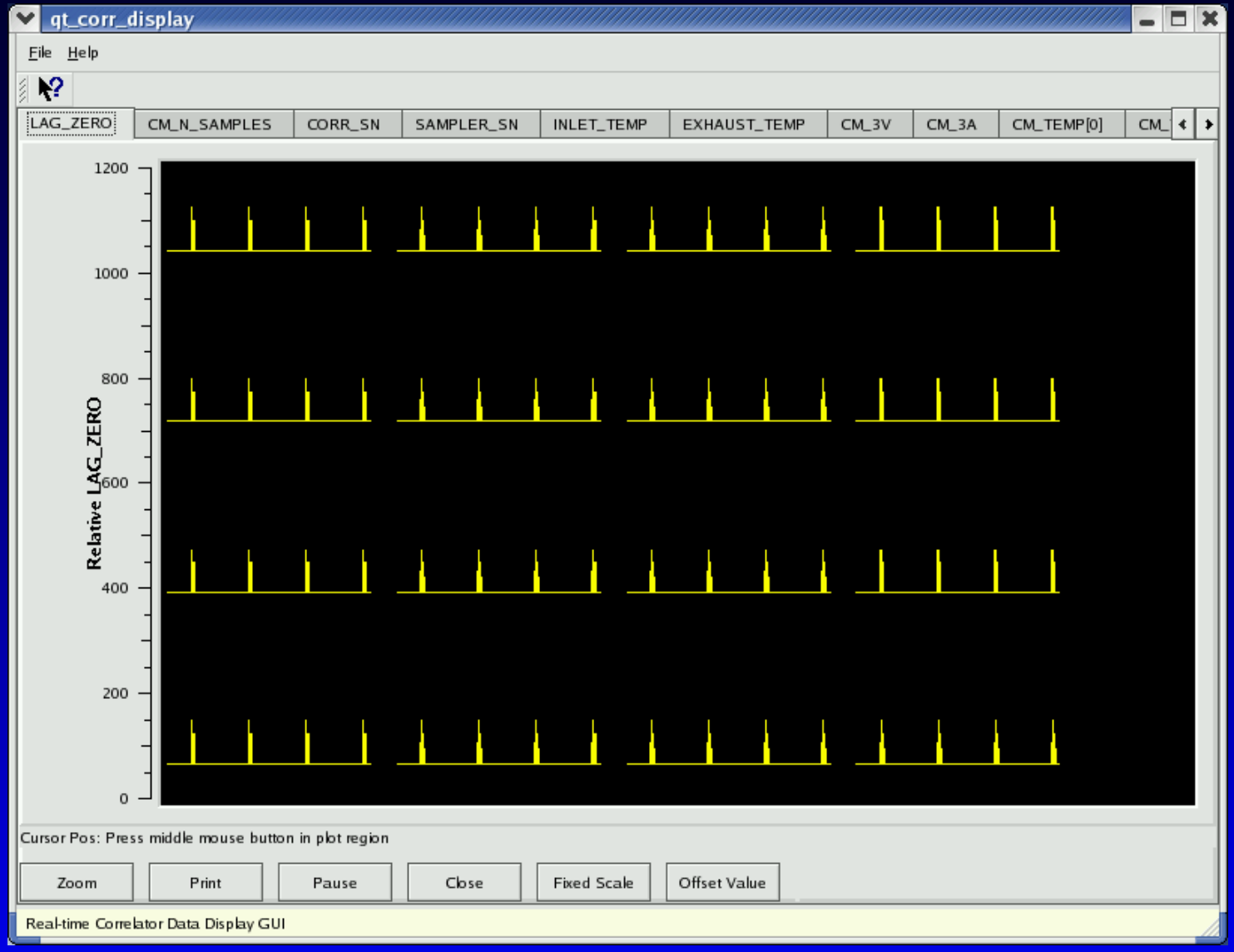
- left GUI shows kview continuously updating during an OTF imaging observation
- right GUI shows spectrum through the entire image cube at current position of the mouse on kview

Spectrum Displays



- left GUI shows all current spectra
- right GUI shows popup that appears when user clicks on an individual spectrum on the left GUI

Sample Meta data Display



ACSIS Pipeline Summary

- Scalable
- Flexible
- Real time feedback on data quality
- Can view intermediate products for debugging and test purposes
- Outputs are aips++ MS for raw data and FITS files for image cubes
- MS data can be reprocessed (non-real time)
- Polarization capability could probably be added with a new reduction recipe

Lessons Learned

- A system for interprocess communication that provides approximately 400 different function calls is not necessarily better than a system that has only two function calls.
  - KISS (not the band)
- Inter agency planning and cooperation needed
  - JAC/DRAO/UKATC squabbles over resource availability, deliverables, delivery times etc
  - distributed development can work but it requires the right people

Some ACSIS People

- Tony Willis, Bill Dent, Gary Hovey, John Lightfoot and Tim Jenness during lab acceptance testing

Addendum

- Glish event loop successfully combined with
  - DRAMA
  - Qt widget set
  - CORBA - Orbacus
  - karma - kview (R. Gooch)
  - Xt (aips++ group)
  - MPI (indirectly - by aips++ group)
  - WWW Boyd Waters et al (ADASS 2002 paper)
- So it’s quite possible to combine glish / aips++ with other messaging systems