Astronomical Instruments Software System Design

Fabricio Ferrari

fabricio.ferrari@unipampa.edu.br

Universidade Federal do Pampa
Brasil

CEFCA Meeting, Teruel, Feb 2010

Facts

- Data is beyond astronomers processing capabilities
- Cycle ***observe–go-home–reduce–analyze–publish*** not practical
- Instruments are sub utilized; lots of unused data
- Telescopes automatized, data analysis mostly manual
- Solving problems: easier in hardware, cheaper in software

Rationale

Why do we need a software {system|group|meetings}?

- Complex instruments $\longleftarrow$ complex software
- **And** data not useful without its software:
- **Thus** software is part of instrument

modern instruments = software + hardware

What do we need?

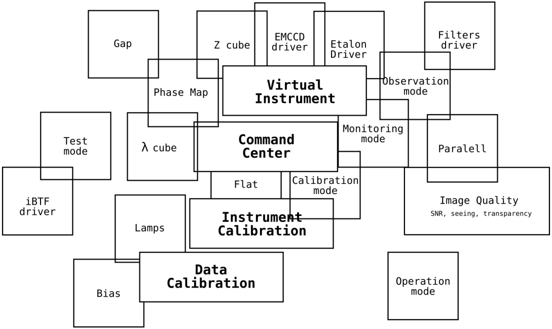
We need

hi-level	user not aware of hardware details
state-of-art	we must trust what we get from software
intelligent	system take decisions by itself
automated	few decisive user interactions
data processing	science out of raw data.

system.

Besides: On-the-fly procedures, pipelined, *astronomer*-friendly

How to put so many pieces together?



Individual processes known (mostly),
but how to integrate them into **One System**?
Humans: Portuguese, Spanish, French, English, ...
Computers: C/C++, Python, IDL, LabView, ArcView, SML, VBasic.

Software System Guidelines

development,
portability,
integration,
maintenance,
usability

Modular Design

- discrete, scalable, reusable modules of isolated, self-contained functional elements;
- simple modules with objective tasks

- good interface design (what it needs and what it provides)
- information hiding (abstraction)
- object oriented design

- disadvantage is increase in communication network sockets and hi-level remote objects.

- simple examples of modular design:
  hardware: computer parts   software: IRAF

Open Source Tools – Operating System: **Linux**

- Many flavors (distributions):
  Debian (servers), Ubuntu (development), Fedora (SOAR).
- Tools quality and availability.
  * my system: 1427 installed packages (8 Gb), 24692 available.
  * programming languages (C/C++, Java, Perl, Python, Fortran),
  * text processors and editors (OpenOffice, LaTeX),
  * scientific and data analysis tools (Scilab, Octave, Maxima, Gnuplot)
- Native multitask, multiuser, networkable system.
- Extensively tested on many environments:
  desktops, servers, development.
- Huge (and growing) scientific community of users and developers.

Open Source Tools – Programming language: **Python**

- Object oriented paradigm, imperative, dynamic typed;
- Emphasizes programmer productivity, code readability;
- Multi-plataform (Linux, Mac, MS-Windows, cell phones, ...)
- Large and comprehensive standard library, *"batteries included"*
- Powerful and easy to integrate external libraries
  PyFITS, NumPy, SciPy, PyRAF, PyRO, PyLab/Matplotlib.
  STSDAS, astLib, AstroLib (IDLs twin), PyMIDAS, EphemPy, ...
- data handling capabilities
  high level data types (lists, dictionaries, sets, arrays, ...)
- *Exceptions*: C/C++ for bottlenecks, hardware drivers

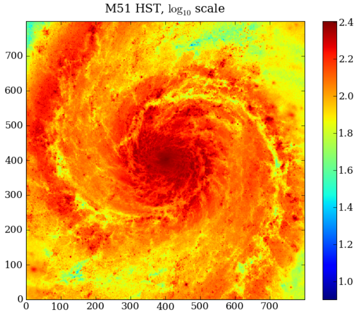
(1 minute Python Example)

```
# import libraries
import pyfits
import pylab
import numpy

# reads and operates on data
data = pyfits.getdata('m51hst.fits')
logdata = numpy.log10(data)

# show, format and save figure
pylab.imshow(logdata)
pylab.title('M51 HST, $\log_{10}$ scale')
pylab.colorbar()
pylab.savefig('m51hst.png')

pylab.show()
```



Open Source Tools – Formats

Data: (dependant on file complexity and size):

- **FITS** images and tables
- normal or Gzip compressed plain text files
- optionally XML for config and small structured files

Documentation:

- LaTeX, OpenOffice, PDF, HTML
- preference for convertable and web formats

Open Source Tools Philosophy

Scientific reserch has the paradigm of “open source”

Focus on human resources, not only on products.

Not cheaper, neither easier,
but works better for longer.

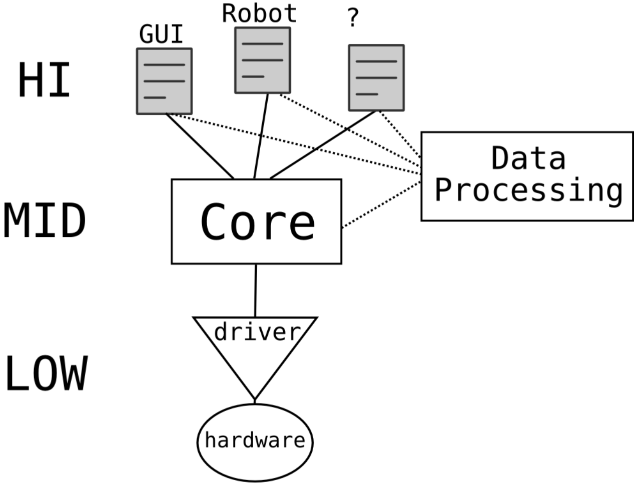
System Availability

- Source code available, even if ©
- Monthly snapshots (at least), data server or CVS
- **code freeze** versions regularly
- Documentation is critical (means availability)
  - **user's manual**: what is, who did, what does, how to and not to use, real examples
  - **programmers manual:** program structure, API, protocols, interfaces, tools, external codes, ...
  - **source code comments:** header – file and author name, date, version, comments; classes or functions – descriptions, interface, on relevant code.

Comments are better the farther the author is.
If your program is not well documented, it is useless without you.

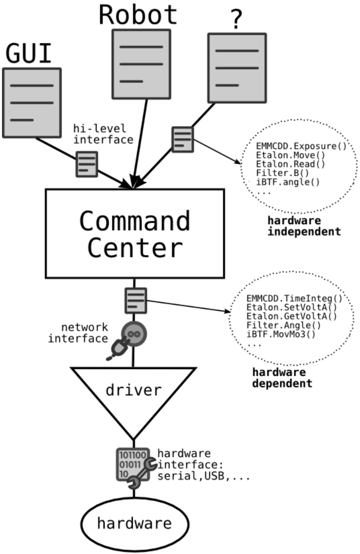
General Structure

Software Point of View



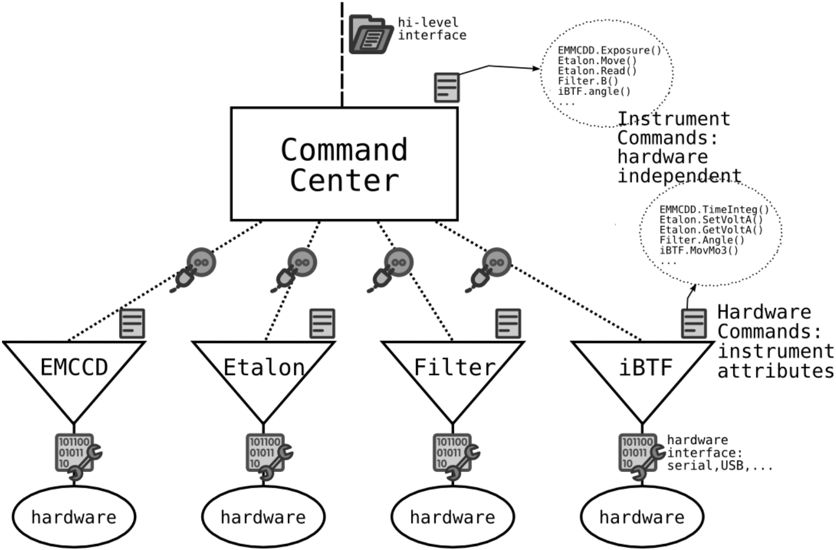
core = control center = command center

The Command/Control Center

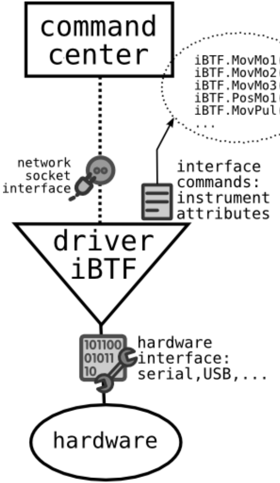


- Abstraction layer between world and hardware
- World communication no hardware dependent commands
- Hardware communication hardware dependent commands
- one software module per hardware part (Etalon, iBTF, EMCCD, ...)

Software and Hardware



The Hardware Drivers



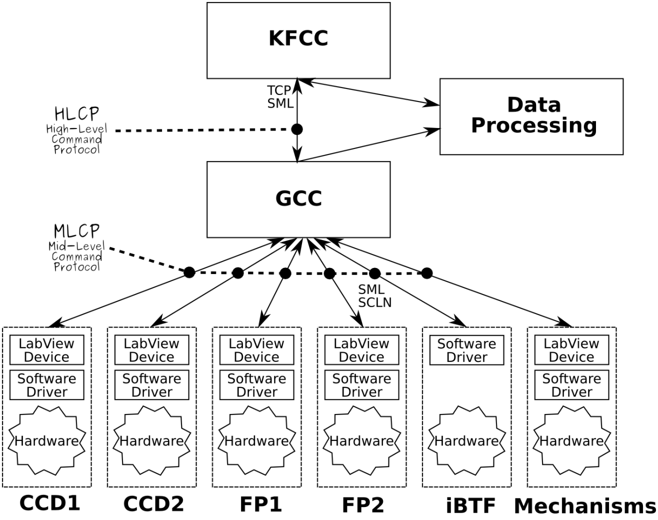
- Complete *Dummy* mode for no-hardware tests
- Results in relevant physical units calibration curve inside drivers
  pulse→degrees,

  capacitance→distance, ...
- LabView, ArcView, SML ?!

The BTFI Example

Brazilian Tunable Filter Imager

BTFI Software Structure



Elements

- KFCC – Keith-Fernando Control Center
  Phase I, II, III, IV
- GCC – Giseli Control Center
  Middleware, **Core** of the system
- Instruments
  Device(LabView), Software driver, Hardware
- Data Analysis
  Corrections, Calibrations, Data science-ready

KFCC – Keith-Fernando Command Center

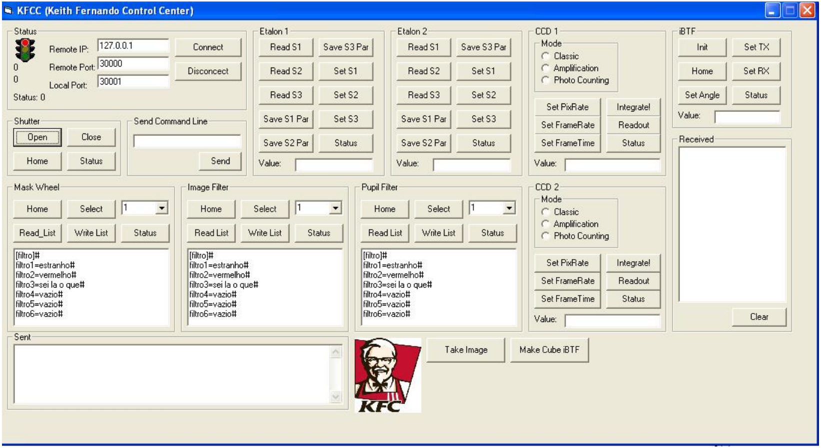
- **Phase I**: simple elementar (*atomic*) commands
  one-to-one correspondence with GCC set of commands
  `shutter.open(), ccd.integrate()`

- **Phase II**: small set of atomic (*molecular*) commands
  `take_image(), make_datacube(), ...`

- **Phase III**: high complexity commands
  `lambda_calibrate(), gap_determination(), ...`

- **Phase IV**: Final KFCC for SOAR
  LabView'ed, Inspired in SOI, ready for use

KFCC – Phase I


KFCC (Keith Fernando Control Center)
Status
Remote IP: 127.0.0.1
Remote Port: 30000
Local Port: 30001
Connect
Disconecct
0
0
Status: 0
Shutter
Open
Close
Home
Status
Send Command Line
Send
Etalon 1
Read S1
Save S3 Par
Read S2
Set S1
Read S3
Set S2
Save S1 Par
Set S3
Save S2 Par
Status
Value:
Etalon 2
Read S1
Save S3 Par
Read S2
Set S1
Read S3
Set S2
Save S1 Par
Set S3
Save S2 Par
Status
Value:
CCD 1
Mode
Classic
Amplification
Photo Counting
Set PixRate
Integrate!
Set FrameRate
Readout
Set FrameTime
Status
Value:
iBTF
Init
Set TX
Home
Set RX
Set Angle
Status
Value:
Received
Clear
Mask Wheel
Home
Select
1
Read_List
Write List
Status
[filtro]#
filtro1=estranho#
filtro2=vermelho#
filtro3=sei la o que#
filtro4=vazio#
filtro5=vazio#
filtro6=vazio#
Image Filter
Home
Select
1
Read List
Write List
Status
[filtro]#
filtro1=estranho#
filtro2=vermelho#
filtro3=sei la o que#
filtro4=vazio#
filtro5=vazio#
filtro6=vazio#
Pupil Filter
Home
Select
1
Read List
Write List
Status
[filtro]#
filtro1=estranho#
filtro2=vermelho#
filtro3=sei la o que#
filtro4=vazio#
filtro5=vazio#
filtro6=vazio#
CCD 2
Mode
Classic
Amplification
Photo Counting
Set PixRate
Integrate!
Set FrameRate
Readout
Set FrameTime
Status
Value:
Sent
KFC
Take Image
Make Cube iBTF

GCC

- Middleware: KFCC – Instruments
  bridge between languages, protocols, plataforms

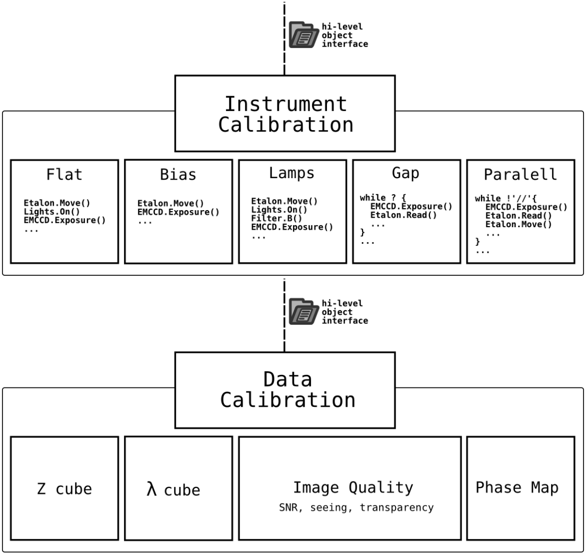
- **Core** of the system
- Set of elementar atomic commands **only**

- **hi-level to observer** software (HLCP),
  mid-level to instruments software (MLCP)

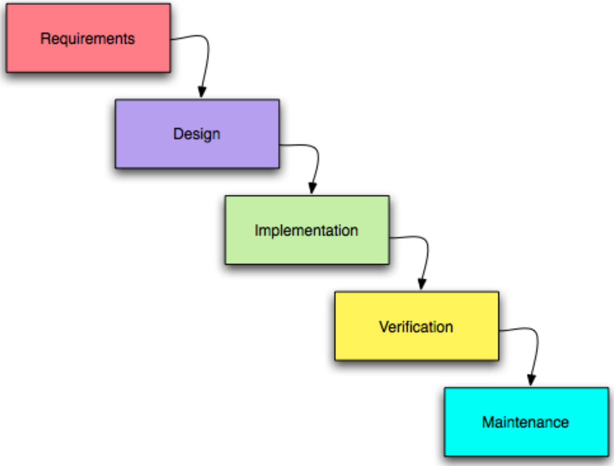
- Basic error checking
- Resource locking (race conditions avoidance)

- Configuration variables acessible to all systems
- Status GUI with all information (read-only)

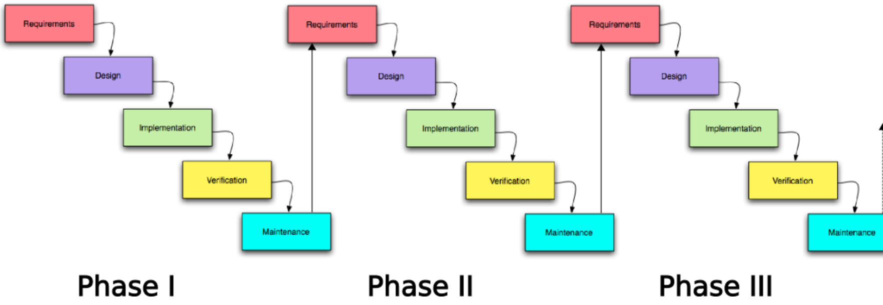
Data Processing



Waterfal Development Model



Iterative Incremental Development Model



Hardware drivers Data Processing Command Center GUI

Scientific Visualization

3Dslicer http://www.slicer.org/

VTK - Visualization Toolkit http://public.kitware.com/VTK/index.php

VisIt visualization Tool https://wci.llnl.gov/codes/visit/home.html

Teem - representing, processing, and visualizing scientific raster data.

http://teem.sourceforge.net/

Scientific Computing and Imaging (SCI) Institute
(many OpenSource data visualization tools)

http://www.sci.utah.edu/index.html

DISLIN Scientific Plotting Software http://www.dislin.de/

Python Resources

PyRO - Remote Objects http://pyro.sourceforge.net/

Psyco - otimization http://psyco.sourceforge.net/

AstroPy - astronomical resources
http://www.astro.washington.edu/owen/AstroPy.html

PyEphem http://rhodesmill.org/pyephem/

MatPlotLib http://matplotlib.sourceforge.net/

Interactive Data Analysis in Astronomy with Python (IDL style), *Perry Greenfield and Robert Jedrzejewski*
http://www.scipy.org/wikis/topical_software/Tutorial