Advanced Interfaces Between Scientists and Data Reduction Systems

B. Nikolic

Astrophysics Group, Cavendish Laboratory, University of Cambridge
http://www.mrao.cam.ac.uk/~bn204/

HILADO project webinar
5 April 2013

UNIVERSITY OF CAMBRIDGE

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

Aims

- Shortcomings of current interfaces
- How could they be better? What would we gain?
- Outline some design & implementation options
- Discuss agenda for meeting 11th April

Scope

- Relevant for data reduction which has some or all of following characteristic:
  1. Is interactive
  2. Needs to be flexible
  3. Large datasets, expensive to move data and expensive to process
  4. Long-term input from scientists/domain-specialists

Not in scope

This is *not needed* if:

1. Domain specialist A specifies fully the data reduction pipeline
2. Programmer specialist B implements the specification
3. Pipeline runs with adequate results

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

Shortcomings summarised

- User needs to keep track of:
  - Which operations have been done already
  - Which operations are still to be done
  - Which operation depends on which
- User largely needs to keep track of how various intermediate data products have been produced
- User needs to decide how to combine operations to make them efficient
- User needs to decide how to subdivide operations to make them parallel

User $\equiv$ domain specialist?

Implications of these shortcomings

At least in my experience...

- Not efficient in terms of computing resources
  - Repeat all operations ‘just-in-case’
  - Operations *not* combined for efficiency
  - Available parallelism not utilised
- Not efficient in terms of user time
  - Lots of waiting for things to finish
- Not reliable – lapses in user concentration can lead to having to repeat data reduction (or worse!)
- Not reproducible – interactive commands often not collected into final script

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

A better interface could be:

1. Faster
   - Less Wall-clock time
   - Less Scientist's time
   - Fewer computational resources
2. More reliable
   - Fewer opportunities for user error
   - Easier to make fully repeatable
   - Easier to review by reading the script
3. More communicable
   - The data reduction script can be used to communicate what needs to be done to other people as well as the computer

Is it worth it?

- Vastly higher data rates from recent radio-astronomy and forthcoming radio telescopes
- Need to reduce investment in time by scientists to get results from radio interferometers
- Improve reproducibility and maintain correctness of science with radio-interferometers

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

Simple flagging-based example

Note:

- I use flagging here for illustration only
- Similar principles apply to many other operations

Flagging fragment

fragment of an ALMA data reduction script:

```
1  # Python/CASA
2  vis="mydata.ms"
3  flagdata(vis=vis, autocorr=True)
4  flagdata(vis=vis, mode='shadow', diameter=12.0)
5  flagdata(vis=vis, antenna='DV04')
```

This likely causes *three* complete iterations through the data. Why:

- The interface is fully procedural
- Each flagdata only knows about itself – it doesn't know it is followed by another similar command

If Input/Output limited $\Rightarrow$ big performance penalty

Operation folding ‘by hand’

to following hypothetical command[1]:

```
1 # Python/Something like CASA
2 vis="mydata.ms"
3 flagdata_(vis=vis, [ {'autocorr': True} ,
4                      {'mode'='shadow', 'diameter': 12.0},
5                      {'antenna'='DV04'}])
```

- All three operations have been ‘folded’ into a single command
- flagdata_ can execute all of them in a single iteration through the data set

Drawbacks:

1. The user must decide what commands to fold and when
2. Different interaction when doing single commands to script

[1] By coincidence a command like this was implemented around the time I first wrote this slide

Folding multiple operations?

But, maybe there is also a benefit of combining application of calibration and flagging?

```
# Python/Something like CASA
vis="mydata.ms"
gencommand_(vis=vis, [ {'op': 'flagdata', 'autocorr': True} ,
                       {'op': 'flagdata', 'mode':'shadow', 'diameter': 12.0},
                       {'op': 'flagdata', 'antenna':'DV04'},
                       {'op': 'applycal', 'caltable': ['myvis.bpass',
                                                       'myvis.W'] }
                       ])
```

It is clear where this is going:

```
# Python/Something like CASA
gencommand_('myscript.py')
```

Back to square one!

$\Rightarrow$ The 'script' must be in a non-procedural language

Possible requirement

Operations automatically re-ordered and folded to optimise performance:

```
1  # Python/CASA
2  vis="mydata.ms"
3  flagdata(vis=vis, autocorr=True)
4  flagdata(vis=vis, mode='shadow', diameter=12.0)
5  flagdata(vis=vis, antenna='DV04')
```

$\Rightarrow$ Automatic translation ('re-writing') $\Rightarrow$

```
1  # Python/Something like CASA/User does not see this
2  vis="mydata.ms"
3  flagdata_(vis=vis, [ {'autocorr': True} ,
4                       {'mode'='shadow', 'diameter': 12.0},
5                       {'antenna'='DV04'}])
```

$\Rightarrow$ Execution!

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

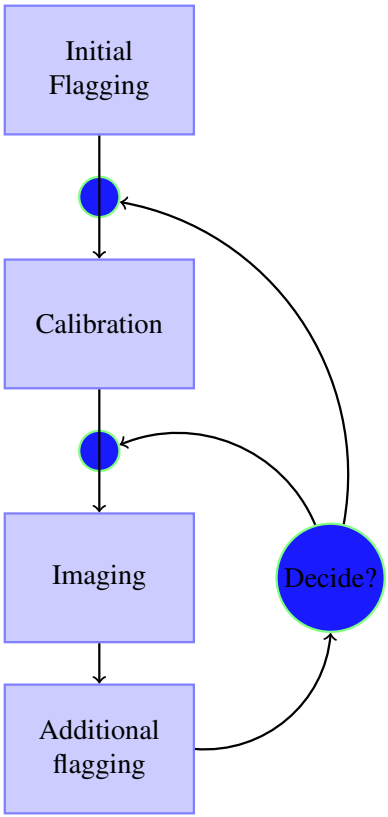
Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

Restart after additional calibration



Where should be reduction continue after additional flagging:

- Before calibrations if they affected by the new flags
- After calibrations if the new flags only affect the science target
- In each case only the SPWs, fields, etc that can be affected should be redone

Restart – the old fashioned way

```
#Python/CASA
if 1:
    #Initial flag
    flagdata()
      ....
if 1:
    #Calibration
    caltable=root+'.W'
    gaincal(...)
if 1:
    applycal(..., caltable)
    ...
if 1:
    #Imaging
    clean(....)
    ....
```

```
#Python/CASA
#Initial flag
flagdata()
#      ....
#Calibration
#gaincal(...)
#applycal(...)
#      ...
#
# Imaging
clean(....)
#clean(....)
#
```

1. Error prone!
2. The script looks different from the interactive commands

$\Rightarrow$ The computer should decide which steps need to be done and which not!

Possible requirement

- The user always runs the entire script
- Only the operations which could have different outcome are executed by the computer

Advantages

- The script is always in final version
- No possibility of mistake due to incorrect restart
- Save time by avoiding unnecessary full restarts

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

Intermediate data product tracking

vs. scan based gain calibration tables:

```
#Python/CASA
gaincal(vis = split1, field = '0', gaintable = root+'bandpass.bcal'
        refant = 'DV02', caltable = root+'intphase.gcal',
        calmode = 'p', solint = 'int',
        minsnr=2.0,  minblperant=4)
gaincal(vis = split1, field = '0', gaintable = root+'bandpass.bcal'
        refant = 'DV02', caltable = root+'infphase.gcal',
        calmode = 'p', solint = 'inf',
        minsnr=2.0,  minblperant=4)
```

...

```
#Python/CASA
gaincal(vis = split1,
        field = '0',
        gaintable = root+'bandpass.bcal'
        refant = 'DV02',
        caltable = root+'inf_phase_dv02_field0_blperant4.gcal',
        calmode = 'p',
        solint = 'inf',
        minsnr=2.0,
        minblperant=4)
```

$\Rightarrow$ eventually the *name* of the table encodes all the parameters!

Possible requirement

1. The computer should be keeping track of intermediate data products, calibration tables, plots, images etc
2. We should access them by primarily calling the commands that created them!
3. Closely related to dependency tracking, data reduction branches

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

Data Reduction Branches – script

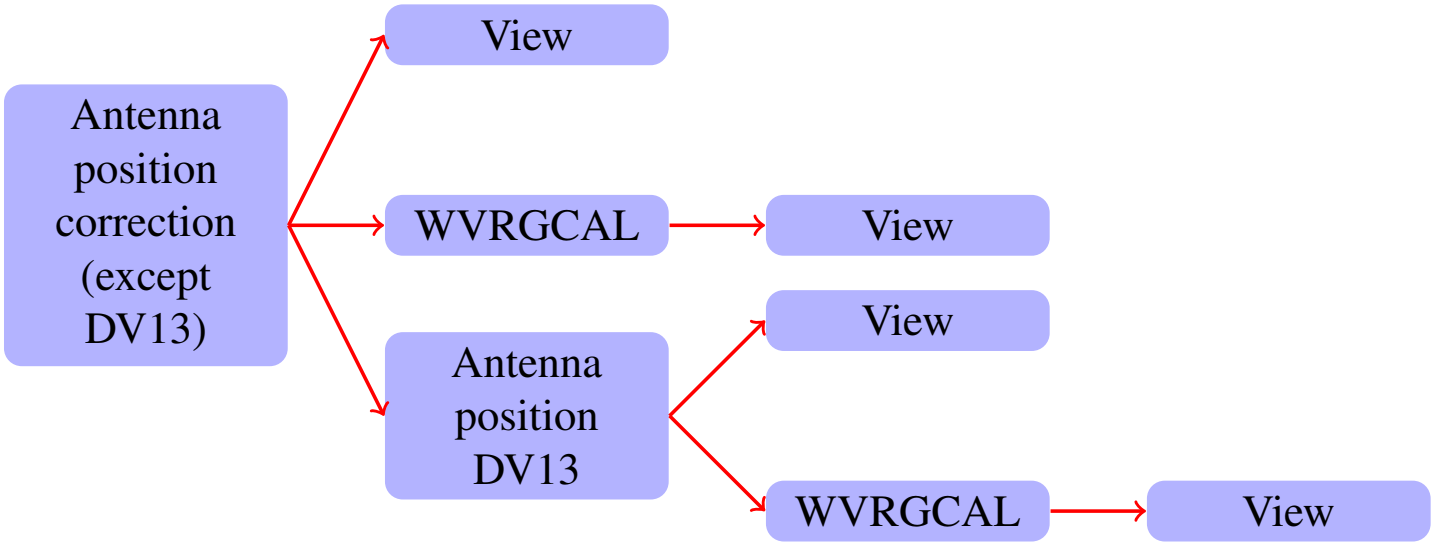
```
default(gencal)
vis = myvis.ms'
caltable = 'antpos_fix'
caltype = 'antpos'
antenna = 'DV02,DV04,DV05,DV06,DV07,DV08,DV10,DV12,DV13,PM01,PM03'
parameter = [
      0.000228,  -0.000334,  -0.000013,
      0.000163,  0.000239,  0.000025,
      0.000060,  -0.000092,  0.000384,
      0.000053,  0.000158,  0.000001,
      0.000103,  0.000328,  0.000351,
     -0.000039,  -0.000085,  -0.000041,
     -0.000331,  -0.000056,  0.000246,
      0.000133,  -0.000210,  -0.000160,
     -0.000045,  0.000104,  0.000109,  # Not sure about this one! (BN)
      0.000191,  0.000010,  0.000119,
      0.000159,  0.000005,  -0.000054
]
gencal()

os.system('../WVRGCAL/bin/wvrgcal --ms myvis.ms \
          --output myvis.W --toffset -1 ')

default(applycal)
vis = 'uid___A002_X219601_X4cd.ms'
#gaintable = 'antpos_fix'
gaintable = ['antpos_fix', 'uid___A002_X219601_X4cd.W']
applycal()
```

Data reduction branches – notes

Note:

1. The user obviously wants to try with/without the WVR calibration – uses the commenting out technique
2. Also want to try with/without correction for antenna DV13
3. Note also the difficulty of attaching antenna names to position correction values

Data reduction branches – graph



- Recording just the data reduction path that happened to work in one case is not enough
- Decisions need to be made by scientists
- But, all inputs for their decisions prepared automatically

Possible Requirement

1. The user should be able to specify multiple branches of data reduction where different parameters/procedures/options are invoked
2. The computer should keep track of the results of computation and present them to the user
3. The reductions would ideally be parallelised

Strawman summary of requirements

1. Commands should be designed to best communicate to other scientists what needs to be done
2. Trying out different parameters/commands should be easy, efficient – should recognise there is no single 'correct' result
3. Concise
4. Efficient, automatically parallelised, storage aware, fast

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

Current scripts/interfaces follow *Procedural* model

Also called the ‘von Neumann’ model:

1. Key feature are pervasive *states*
2. Program consist of creation and sequential execution of blocks of manipulations of *states* (i.e., procedures)
3. The major part of most compilers today is undoing the von Neumann model behind the scenes
4. Originally states $\equiv$ program variables
5. Our state? The measurement set
   - A very large state
   - Very expensive to manipulate

   Sequentially, blindly executing unoptimised blocks of operations on measurement set is very inefficient

Backus on the Procedural model

Conventional programming languages are growing ever more enormous, but not stronger. Inherent defects at the most basic level cause them to be both fat and weak:
[...]
their inability to effectively use powerful combining forms for **building new programs from existing ones**, *and their lack of useful mathematical properties for reasoning about programs*

John Backus, ACM Turing Award Lecture, **1977**

Functional/declarative languages

Work started on solving the problems of procedural programming a long time ago....

- Functional/applicative and declarative programming languages

Up until now they have not **in general** gained neither widespread usage nor high efficiency

- However, currently emerging technologies:
  - Map-Reduce
  - 'Dataflow' programing

  paradigms are closely related to functional languages and rapidly becoming popular

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

Principles

- Will need to severely restrict the domain of interface to make the problem tractable
- Must reuse AIPS, CASA or other existing toolsets for all actual processing of data – this is thin layer on top
- Must present familiar concepts and commands to the user (flag, gaincal, split, applycal, etc) – minimise new learning that needs to be done
- First version should reach a useful state in $<$3 months of development:
  - Carefully identify features need to be useful
- Should integrate with existing scripting as far as possible (i.e., runnable from Python/ able to run python)
- Identified path to future enhancement

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

Python hack approach

Advantages

- Keep the existing language foundation
- Incremental development, immediate results
- Easy distribution to users
- Integration with other analysis done in Python
- Feasible!

Disadvantages

- Need to reduce the range of possible programming constructs
- Awkward syntax, will get more complex
- Limited scope for enhancement

How does it work

- *Applicative* on measurement sets – no side effects, each command *transforms* the data
- *Lazy* – results only computed when requested by the user, not as encountered
- No flow control(!)
  Decisions made by scientists
  (Flow control based on original data easily implementable)
- Optimisation/rewriting stage just before execution

Prototype Example

```
def mydata():
    return Vis("uid___A002_X219601_X4cd.ms")

def mostpos(d):
    "Correct for antenna positions that we are sure about"
    d=Antpos(d, "DV02", [0.000228, -0.000334, -0.000013])
    d=Antpos(d, "DV04", [0.000163, 0.000239, 0.000025,])
    d=Antpos(d, "DV07", [0.000103, 0.000328, 0.000351])
    d=Antpos(d, "DV10", [-0.000331, -0.000056, 0.000246])
    d=Antpos(d, "DV12", [0.000133, -0.000210, -0.000160])
    d=Antpos(d, "DV13", [-0.000045, 0.000104, 0.000109])
    d=Antpos(d, "PM01", [0.000191, 0.000010, 0.000119])
    d=Antpos(d, "PM03", [0.000159, 0.000005, -0.000054])
    return d

def maybepos(d):
    "Not sure about this one! Will want to tru with and without"
    d=Antpos(d, "DV08", [-0.000039, -0.000085, -0.000041])
    return d


def antcheckd()
    return Select(mydata(), spw=1)

Plot(VisRaster(mostpos(antcheckd())),
     dims=["time", "phase"])

Plot(VisRaster(maybepos(mostpos(antcheckd()))),
     pdims=["time", "phase"])

go_reduce()
```

Version 0 goals

1. Applicative sub-language of Python
2. Commands for basic calibration, flagging, continuum imaging only

Features

1. Restart/Dependency tracking
2. DR Branching
3. Folding
   - 3.1 Antenna flagging
   - 3.2 BL correction
4. Simple intermediate product cache

Future goals (relatively easy reach)

- Automatic parellelisation multi-thread/SMP/cluster
  - Can parallelise on outermost scale, high efficiency
- Summary reports of all data reduction steps
- Automatic management of cache of intermediate data products

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

Domain Specific Language approach

- DSL script $\Rightarrow$ Automatically generated Python $\Rightarrow$ CASA tasks
- Implement only language features we can support
- Reuse task definitions from CASA, either automatically or with small adjustment
- Integrate closely with Python

DSL Advantages/Disadvantages

Advantages

- Much easier to ensure correctness of scripts
- Better error messages
- More natural and flexible syntax
- Can reuse powerful tools already built
- Clear demarcation between the new language and Python
- Excellent scope for further enhancement

Disadvantages

- Longer time to first working version
- Potentially both users and developers will be confused by new ideas and techniques

Outline

Introduction

Shortcomings of current interfaces

Better interfaces

Examples and Possible Derived Requirements

Operation Folding

Dependency tracking

Intermediate product tracking

Data reduction branches

Related/Existing concepts

Implementing a better interface

Approach: hack Python

Approach: Domain Specific Language

Next steps

Suggested next steps

- Collect a variety of real-life data reduction scripts
- Identify which data processing features are necessary for first useful implementation of a new DR Interface
- Identify which Advanced DR features are necessary and estimate development time on two possible technologies
- Decide on technology and proceed to design and implementation