Some thoughts on Athena Job Configuration

Mostly my thoughts + some input from previous meeting on this topic

Walter Lampl

University of Arizona

A bit of history

- Initially athena (Gaudi) used a text-base jobOption files
  - Almost no program-language capabilities
  - There are still some *.txt job opts flying around in the share directories
- Was judged to be not powerful enough and we moved to python in 2003
  - Actually a somewhat tweaked version of python that eased the conversion of txt to py jobOpts
  - I took me a little while to grasp that include("..") is actually a not a python construct
  - In fact, athena.py uses include that calls `execfile` on the jobO file it parses
- Python jobOpts grew big (and messy) over time
  - We do use the full power of the language, including "Autoconfiguration"
- Several attempt to better structure the jobOpts
  - Always stayed back-ward compatible, just optional functionality added on top of the existing framework
  - `Configured` base class
    - Attempts to mirror the data-flow in the cxx part
  - ConfiguredFactory, CfgGetter
    - Mostly intended to provide consistent configuration of AlgTools or sets of AlgTools and avoid boiler-plate code

Previous job-config brainstorming meetings

- We had so far 2 (inconclusive) meetings on this topic:
  - Proposal by Luc & myself in 2013 ("Minerva", see later)
    - https://indico.cern.ch/event/274228/
    - Not too warmly welcomed in particular by the HLT
    - Not a time where we were really ready to turn things upside-down
  - Splinter Meeting at Software & Computing week June 2015
    - https://indico.cern.ch/event/342880/

- Discussions about a declarative vs procedural approach
  - Declarative: Some like a static xml describing the job config
  - Procedural: A program with control flow as we have now
  - See next slide …

About a purely declarative approach

- Some argue that we should be able define our job configuration statically
  - E.g. one xml or json file defining “reconstruction”
  - Instead of a python program with loops and if-statements
- We actually do have a declarative layer: The pickled configuration at the end of job-configuration
  - The problem is rather how to get this point
- My opinion: We do need programming-language features to accommodate our needs

Autoconfiguration

- At the configuration stage we peek into the input file (BS or POOL) to get various pieces of metadata
- The run-number obtained this way is then used to peek into the COOL database to obtain even more metadata
- Autoconfiguration was introduced during the initial commissioning (~2008) to avoid Tier0 job configuration changes if the ATLAS magnets where ramped up or down (in the middle of the night)
- Somewhat contradictory to the athena framework design: Time-varying changes should be handled by IOV callbacks at the C++ level
  - Not really do-able for fundamental changes like collisions/cosmics or field on/off
- With Autoconfiguration available it was happily used by many other clients
  - Example: LAr checks DAQ configuration and adjusts it's reco accordingly

My Conclusion: Although Autoconfiguration may be "ugly" from the philosophical/ architectural point of view, I think it's not possible or at least not worth getting rid of it

What I dislike about the current system

- It's a big mess!
- Lots of try-error needed to get something running
- Often, we run more Alg/Tools/Services than actually needed for the job
- Little encapsulation, everything is global
  - You need to know what happened upstream of your fragment
  - You may accidentally overwrite someone else's configuration

- Ever tried setting up a job doing anything non-trivial from scratch?
  - I think only a handful of you ever managed that
  - Consequently RecExCommon is used for many purposes for which it's not really ideal

Methods vs global namespace

- Usually programmers encapsulate functionality in functions or methods that have a list of parameters that they depend on and a return value
  - Explicit dependencies
- Our current python configuration hides the dependencies
  - Modules (often just included files) depend on globally defined flags and attach their “result” the globally defined topSequence, ToolSvc or ServiceMgr

Impact of the new framework

- The order of algorithm in the `topSequence` is not important any more
  - Algorithms are scheduled according to data-flow and even in parallel if possible
- Getting rid of shared, public tools changes the picture as well
  - Every tool clearly belongs to an algorithm and can be configured with it

Wish-list for a new configuration system

- No global name-space!
- Fewer, clearer steering flags
- jobOption modules (functions!) should spell out what they depend on as parameters
  - Grouping of parameters might be necessary
- Modules should be compose-able and independently run-able (if they contain at least one sensible algorithm)
  - Example:
    - InDetRecExamle, CaloRecExample could run on RDO or BS input
    - egammaRec could run on ESD-input
  - Concatenating them should run egamma on BS or RDO with no additional configuration

De-Duplication

- Concatenation of independently-runable modules leads inevitably to duplication of components
- On the other hand, services are frequently needed by many clients and should be shared
- Current strategies to avoid duplicate/clashing modules are
  - include.block(“ ..”)
  - python’s inherent import behavior
- Alternative/Complement: Explicit de-duplication step at the end
  - Three cases:
  - Case of multiple components with identical configuration and name: Unify
  - Case of multiple components with different configuration and different name: Keep separate
  - Case of multiple components with different configuration and identical name: ERROR about name clash

How this could work:

- Instead of include/execfile the jobOption files:

```
from inputfile import inputfile_cfg
cfgdict=inputfile_cfg(flags….)
```

  - Can internally calls the _cfg method of modules it depends upon
  - Returns object containing a list of (configured) algorithms+Tools and (configured) services
    - These lists can be concatenated
  - Duplicates will be eliminated automatically
  - Trickier parts:
    - Some components 'accumulate' their configuration from many places:
      - IOVDbSvc.Folder, StreamESD.ItemsList, …

This is basically the proposal that Luc & I came up in 2013 called "Minerva"

- Because it replaces athena.py

Summary/Conclusion

- Still an open discussion
  - A bit of Minerva-prototyping done in 2013
- I am pretty convinced that our current jobO are not maintainable in the long term, so something needs to be done
- Migrating to a new system will be a lot of work (have some 500k LOC python)
- Whatever new system we choose should, it should so much better that users are happily and voluntarily move to it