A NEW APPROACH FOR ATLAS ATHENA JOB CONFIGURATION

23rd INTERNATIONAL CONFERENCE ON COMPUTING IN HIGH ENERGY AND NUCLEAR PHYSICS - 19-13 JULY 2018, SOFIA

Configuring what?

The **ATLAS** offline software (Athena) consisting of thousands of **components** (Algorithms, Services, Tools) written in C++.

Configuration: Assembling a subset of these components into an executable program, respecting their interdependence. The configuration depends on the work-flow (Simulation, Reconstruction, HLT, various calibration jobs, …) and the input data.

A bit of history …

Until 2003: Pure text job-options, almost no programming-language capabilities.

2004: Switch to python with some tweaks to ease backward compatibility with text job-options. Includes a global name-space.

2008/09: Introduce the concept of AutoConfiguration. The full power of the python language is used.

Since then: The job configuration code grew increasingly messy and unmaintainable over the years.

2018: Proposal to replace the job configuration (presented here).

Basic principles of the new system:

No global namespace! Configuration is produced by python functions with arguments and return values.

Self-containment: The configuration of each component contains the configuration of all other components it needs to work.

- Typically by calling other configuration functions
- Implies that the configuration of an event-processing algorithm is standalone run-able (as long as the input can be read from a file).

Composability: Bigger jobs are assembled from smaller configuration fragments.

Deduplication: Many basic components will be declared multiple times. Explicit step to drop (or reconcile) duplicates.

Static vs dynamic configuration

- Athena jobs can be started from a static configuration read from a file (python pickle), the configuration code discussed here produces such a file.
  - Generally done in one go
- Working only with static configurations is not practical.
  - Too many possible combinations workflows and inputs leading to too many different configurations.

In practice:

```
components=configFunction(flags, args, **kwargs)
```

Return value: Instance of ComponentAccumulator
- Lists of algorithms, service, etc.
- Merging-method including de-duplication

Argument: Flags and possibly more args and kwargs

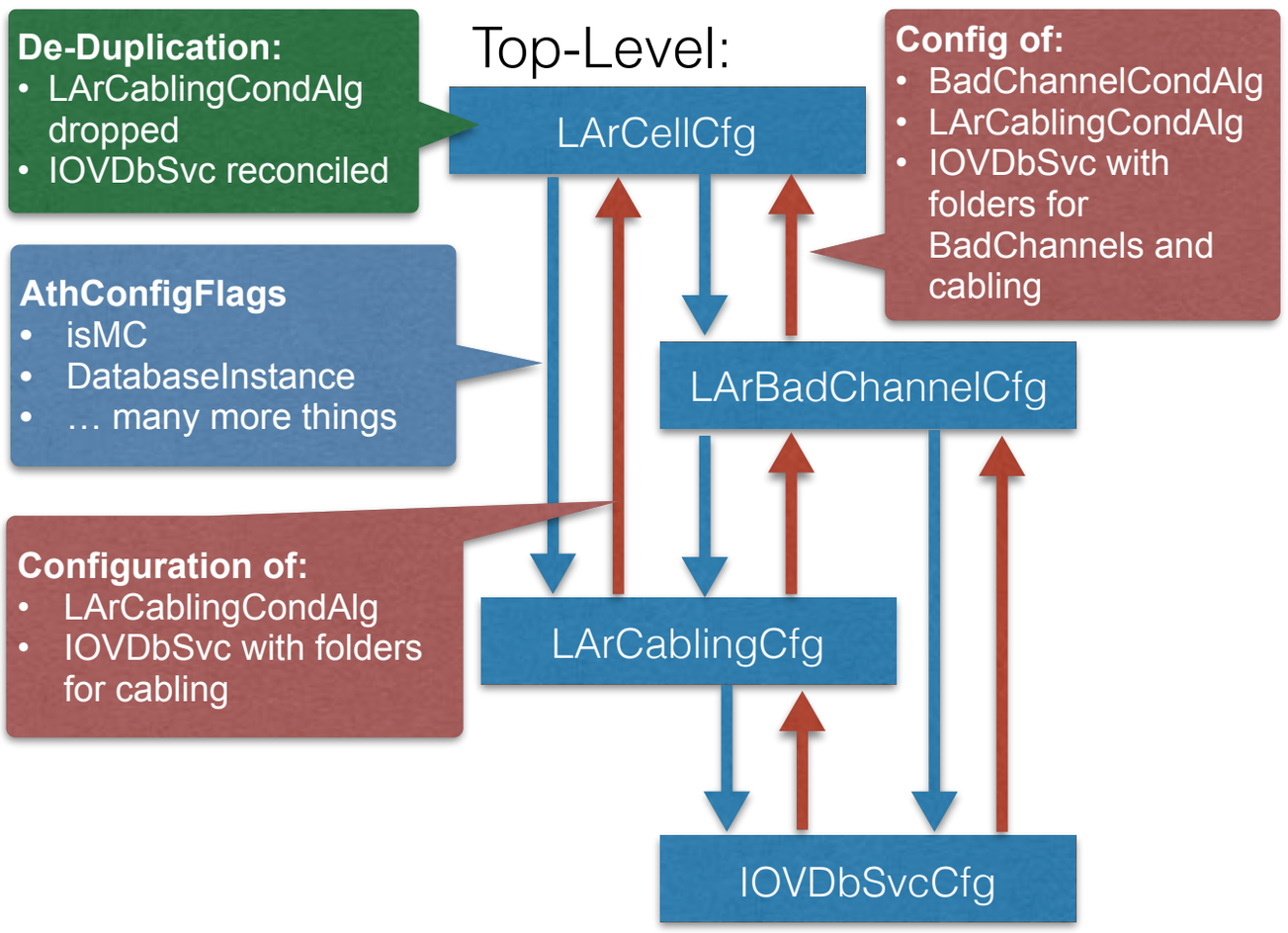
Function adding components and setting their properties. May call other config-functions

Configuration Flags:

Container of **key-value** pairs steering the configuration.

- Examples: real data or MC, cut-values, turning corrections on/off.
- Flags are passed through the call chain from top-level to each configuration method.
- **Interdependence**: Flag knows how to set itself based on previously set flags if not explicitly set by the user.
  - E.G. Flag steering a correction not applicable on MC is turned off if the *isMC*-Flag is set.

Auto-configuration: Setting flags depending on the input (data/MC, B-Field on/off, …).



Walter Lampl (University of Arizona)
on behalf of the ATLAS Collaboration