Dual-use CP Tool Issues

Steve Farrell

US ATLAS Software and Computing / Physics Support Meeting

Ernest Orlando Lawrence
Berkeley National Laboratory

Aug 19, 2014

Introduction

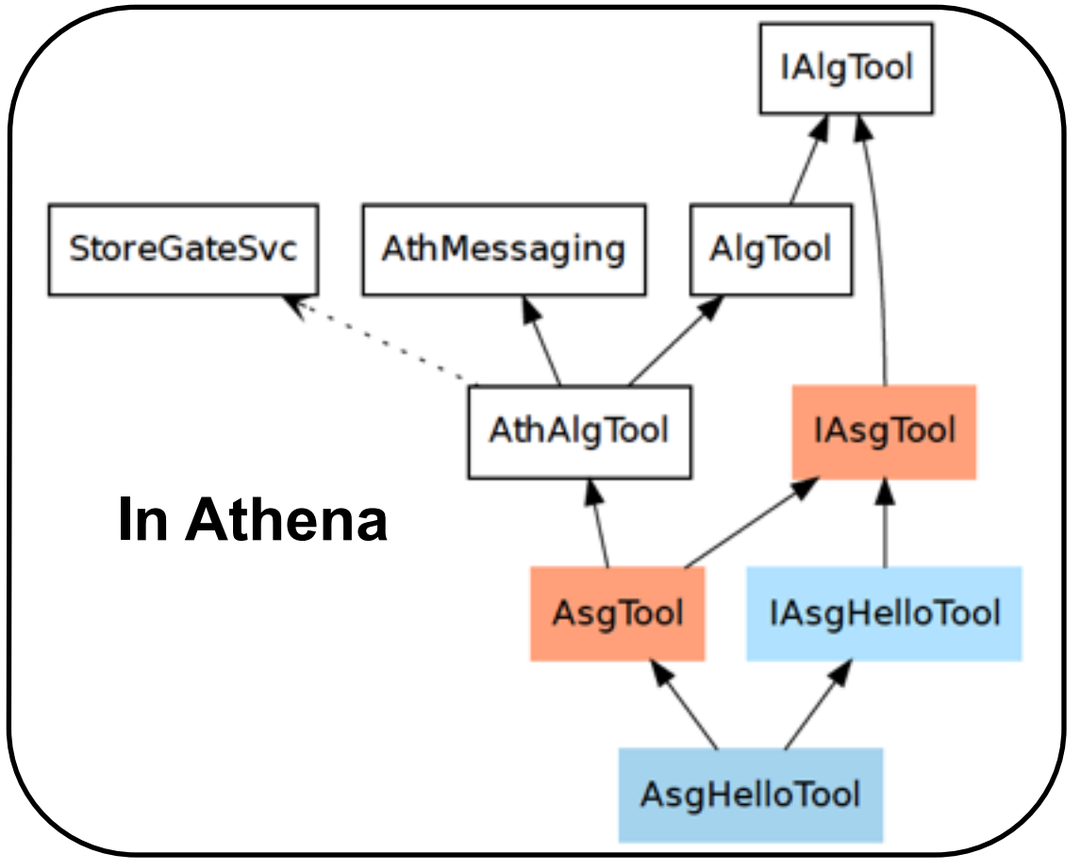
- The purpose of this talk is to present and discuss the various issues that users/developers are experiencing due to the migration and use of dual-use CP tools.
- Most issues that have popped up on the mailing lists fall into these categories
  - Passing user-defined info to tools
  - Dealing with deep/shallow copies of objects in TEvent/TStore
  - Tool interface design
- These issues are not all unresolved. In fact, it's possible that by now they are all resolved!

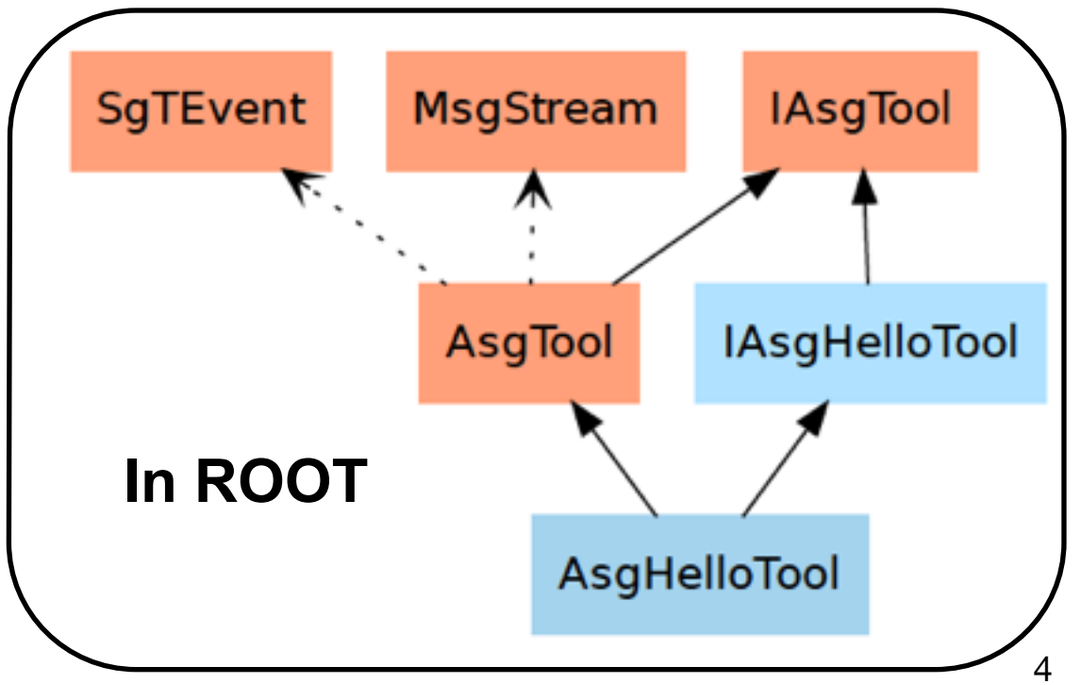
What are CP tools?

- ***Combined Performance*** tools are used by <u>everyone</u> doing analysis to apply the recommendations of the CP groups
- Operations performed by CP tools
  - Apply a correction to objects
    - E.g., scale the jet energy in MC to match data
  - Select objects that are considered “good” for analysis
    - E.g. reject muons with insufficient number of ID tracks
  - Provide scale factors to correct selection efficiencies
    - E.g. correct the electron trigger efficiency in MC to match data
- Many of these tool operations are associated with *systematic uncertainties*, so tools must also be able to provide systematic variations
- In Run 1, most CP tools were written to work on D3PDs
  - With long lists of method arguments

Dual-use tool design

- Described here:
  - https://cds.cern.ch/record/1639568
- Developed to facilitate CP tool development for the dual-use xAOD EDM
- Dual-use functionality provided in the form of an **AsgTool** base class and the **IAsgTool** interface
  - Only need to write one tool for both environments!
- Tool interface classes can be reused by different implementations
- Tools can implement more than one interface





Dual-use tool guidelines

- Described here
  - https://cds.cern.ch/record/1667206
- Recommendations are given for tool naming and method forms; e.g.,

```
CorrectionCode Tool::applyCorrection(xAODObjectType& inputObject) ;
```

- Systematics interface
  - Systematics tools should implement the ISystematicsTool interface, and define the following methods:

```
SystematicCode Tool::applySystematicVariation(const SystematicSet& systConfig);
bool Tool::isAffectedBySystematic(SystematicVariation) const ;
SystematicSet Tool::affectingSystematics() const ;
SystematicSet Tool::recommendedSystematics() const ;
```

  - In addition, they should register their affecting and recommended systematics in the SystematicRegistry
  - Described in more detail on the dedicated twiki: https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/CPToolsSystematicsInterface

Tool migration

- Arguably the biggest issue is getting developers to do the migration!
- Current status is shown on the twiki:
  - https://twiki.cern.ch/twiki/bin/viewauth/AtlasProtected/ASGUserAnalysisToolsxAODMigration
- At the time of this writing, the twiki reports:
  - 11 tools are done and in the release
  - 7 tools are ~done but not in the release
  - 10 tools are on the way (range of expected delivery dates)
  - 3 tools still need volunteers or need to get started
- Are we happy with this progress? What can we do to push the remaining tool developers?
  - We can volunteer to help more
  - Host another sprint-like event?

Tool migration - systematics interface

- It’s unclear how many tools are currently utilizing the systematics interface
  - The final form didn’t make it into the first guidelines note version, but it is there in the current version
- We can’t be sure the interface satisfies all use-cases and workflows are satisfied until all developers try to implement the interface
  - We already know of some cases where the situation gets complicated; e.g., b-tagging uncertainties
- CPAnalysisExamples has a couple of examples demonstrating the usage
  - …though they differ slightly and should maybe be unified

Passing user-defined info to tools

- This was brought up in the context of the JES uncertainties tool, which requires a few things from the user:
  - Number of primary vertices (NPV)
  - Number of selected jets → Very analysis-specific; *cannot* be generically coded into the tool
  - Which jets are considered b-jets → Very analysis-specific; *cannot* be generically coded into the tool
- NPV is mostly considered a non-issue because it "*should*" be standardized in run 2
  - But there's still the (slightly related) question of where/how to get it. Via a small standard tool, decorated on EventInfo?
- The remaining two points are examples of things that can easily pop up again and are maybe not covered clearly enough in the design guidelines:
  - How to pass analysis-specific object-level and event-level information into tools

Passing user-defined info to tools

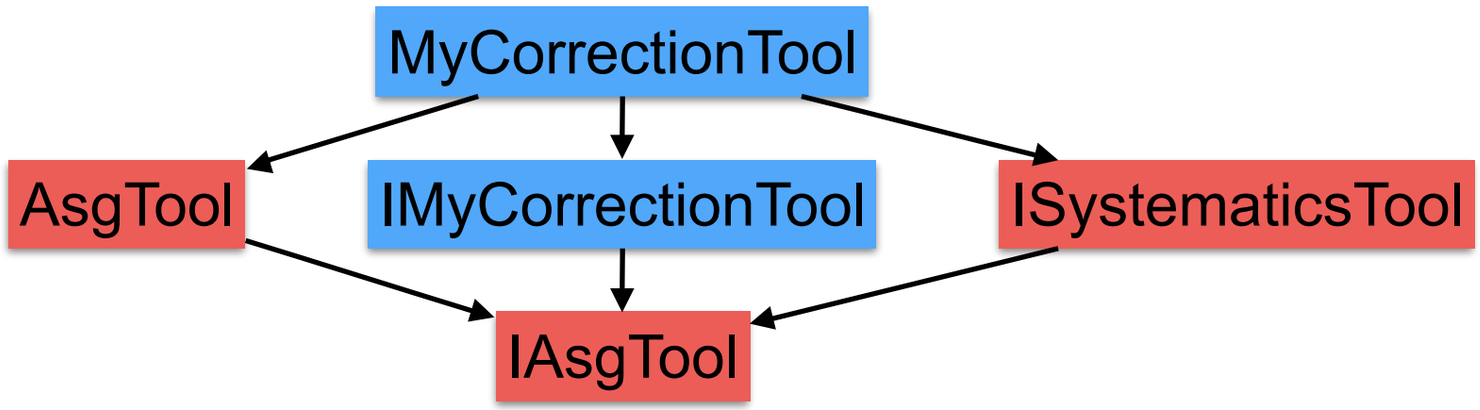
- Solutions proposed
  - User decorates objects/EventInfo as appropriate
    - e.g., numSelectedJets, isBJet
  - Extend tool methods to take additional arguments
  - For object-level info, user could specify sub-containers to tools
    - e.g., MySelectedJets, MyBJets, etc.
- Decoration seems to be the preferred route in most cases
- However, user-info often changes with systematics!
  - For objects, the user will probably use deep or shallow copies for each systematic
  - For event-level info, the solution is less obvious
    - Clear the decorations at every systematic?
    - Use shallow copies of EventInfo?

Container operations

- The guidelines mostly recommend method forms like this:
  - void MyTool::applyCorrection(xAOD::Object* obj)
- To my knowledge, there is no existing precedent or recommendation for container-level operations like this:
  - void MyTool::applyCorrection(xAOD::ObjectContainer* container)
- Such an interface that operates on entire containers could be useful
  - User can supply optional input decoration or use view-containers to control which objects are operated on in the container
- This is an open question for discussion

Tool interface usage and design

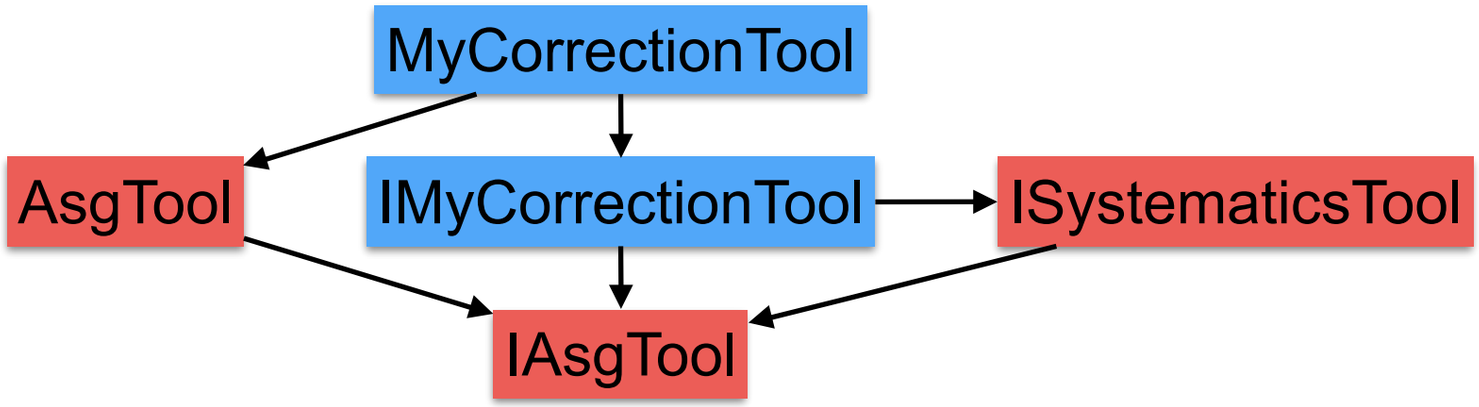
- The relevant thread in this case was started by me:
  - https://groups.cern.ch/group/atlas-sw-pat-am-framework-tf/Lists/Archive/Flat.aspx?RootFolder=%2Fgroup%2Fatlas-sw-pat-am-framework-tf%2FLists%2FArchive%2FTool%20interface%20questions&FolderCTID=0x0120020022999 26C44B9FC4F80B9CBCBEB042C7D
- Recommendations in the guidelines and examples in CPAnalysisExamples lead to a design where a top level tool inherits from multiple unrelated interfaces:



- Then, in Athena (also now in ROOT), one can retrieve and manage tools with a handle:
  - ToolHandle<IMyCorrectionTool>
- However, such an inheritance scheme doesn't allow to simply use both the correction and systematics interfaces from one handle
  - So what's the best way to handle/fix it?

Tool interface usage and design

- An easy solution would be to simply change the inheritance scheme



- But the idea of interfaces is that they should typically be separate and not depend on each other
  - Different ToolHandles can be used to access the different tool functionalities
- Systematics actually *can* be separated, though whether you'd want to will depend on how you setup your code
  - E.g., one could probably set systematic behavior of all tools in one place. Attila suggested the SystematicRegistry
- In RootCore I expect many users won't need to worry about the interface layout since they will deal with simple pointers to the explicit tool types rather than ToolHandles

Deep copies of objects

- This issue came up on the PATHelp mailing list from Ruggero Turra:
  - https://groups.cern.ch/group/hn-atlas-PATHelp/Lists/Archive/Flat.aspx?RootFolder=%2Fgroup%2Fhn-atlas-PATHelp%2FLists%2FArchive%2Fcopy%20electron%20for%20CP%20tool&FolderCTID=0x0120020084D7E80CB0C3394A8D54EF07C309B044
- When doing a deep copy of an object, it is easy for the user to make a mistake that the compiler will not catch:
  - xAOD::Electron* electron_copy = new xAOD::Electron();
    electron_copy->makePrivateStore(***electron_ptr***);
- Of course, the correct form is to provide a reference to the original particle, rather than a pointer:
  - xAOD::Electron* electron_copy = new xAOD::Electron();
    electron_copy->makePrivateStore(*****electron_ptr***);
- This will probably happen to a lot of users unless dealt with
  - Karsten suggested using the explicit keyword in the method definition

Using deep/shallow copies with TEvent/TStore

- There was some confusion with the METUtility tool on how to use transient containers with TEvent and TStore, seen by Kerim Suruliz and Teng Jian Khoo
  - https://groups.cern.ch/group/atlas-sw-tf4/Lists/Archive/Flat.aspx?RootFolder=%2Fgroup%2Fatlas-sw-tf4%2FLists%2FArchive%2Fdeepshallow%20copy%2C%20new%20containers%20%2B%20deleting%20a%20container&FolderCTID=0x01200200170AFF90D76F904A8E2A4602FEB7686C
- Naively storing and retrieving containers with evtStore record/retrieve calls of course won't work without properly instantiating a TStore and recording the containers there.
- To the user, this stuff about stores and copy-containers might look a little complicated, but this is maybe just a documentation issue.
  - Attila gave a nice thorough explanation in the thread, but maybe the examples, twikis, and or printouts need to be made more clear
    - In this case, the error message they received *was* informative, but not quite enough to help them solve the issue on their own.
- In the same thread, the issue of using VarHandles was brought up and discussed as a way to possibly simplify the user's interface to TEvent and TStore
  - But that is a discussion for Paolo's presentation

Conclusions

- Migration is (finally) coming along nicely, though it may take a while to have everything fully validated
  - Systematics in particular hasn't been fully field tested
- Most of the "issues" encountered were minor
  - Misunderstandings about how to properly use the technology
  - Discussions on how to properly apply the guidelines
  - Discussions on how to handle new use-cases
- No major roadblocks foreseen, but more issues may arise later