Future Directions of Gaudi

...

Benedikt Hegner
(CERN EP-SFT)

Gaudi Workshop 2017
25.9.2017

General Direction

I think the general direction of current activities is very clear

⇒ The devil is mainly in the details!

General Direction

I think the general direction of current activities is very clear

⇒ The devil is mainly in the details!

Apart from one little question...

General Direction

I think the general direction of current activities is very clear

⇒ The devil is mainly in the details!

Apart from one little question...

Is the long term vision for Gaudi the *functional approach*?

Remove legacy

Plenty of concepts in Gaudi that are either outdated or never really worked

- Interface versioning
- QueryInterface
- Reference Counting

My (old) proposal is to remove query interface completely

Update code

Use the understandable (*) phase-space of C++17 as a base

Fix namespacing

⇒ no explicit migration, but more nitpicking in MR

⇒ volunteers for migration welcome though

(*) don’t be too smart and remain nice to later maintainers!

Code organization

Using GIT allows us to clean up the repository and move components around for clarity

- have one central GaudiCore
- A **small** number of other packages reflecting 100% the include directory
- Optional components separated out
- Every interface still gets a minimal implementation

⇒ **A topic for a hackathon session**

IAlgorithm and other inheritance trees

On removing states we violate a few base assumptions of our base classes

- For backwards compatibility we have stateful implementations
- For the future we have stateless algorithms

Do they belong into the same inheritance tree and how?

- Putting them into the same makes the transition quicker
- Putting them into a separate ones makes them future proof
- Decision between pragmatism and future maintenance costs

In the ideal world the proper approach is starting from scratch

⇒ In algorithm and scheduling case decouple end-users from this and keep freedom to completely wipe-out old code

Replacing components

There are many components to be fixed for thread safety

We should always ask ourselves

1. Do we fix the existing code?
2. Do we implement a new design to eventually replace the other

I would prefer option (2) as we can as well re-design the use case

- Have to make sure we really *deprecate the old implementation at a well defined time-scale*

⇒ Needs agility on Gaudi side and commitment on experiment side

⇒ If you don't pay a bit now, you will pay much more later

Gaudi Copyright and License

Licenses are currently a hot topic in HEP software

The license decided by Gaudi has a direct impact on its users

- Experiments currently 'suffer' from GPL'ed externals

It as well has a direct impact on its developers

- DOE labs and Openlab cooperations either prefer or require other open-source licenses

So, the questions are

- Can we identify the authors of Gaudi?
- Can we agree on a license?
- Where do we depend on GPL ourselves?

⇒ Experiments rely on an answer by us!

Action Items of Last Year

Action item 1 - new DataHandle interface
Action item 2 - new Data store interface
Action item 3 - finish interface of new property declaration
Action item 4 - Drop interface versioning
Action item 5 - Drop custom reference counting
Action item 6 - Assess which components are used by which collaboration
Action item 7 - Drop zombie components
Action item 8 - Port Gaudi to Mac
Action item 9 - Port Gaudi to Ubuntu
Action item 10 - Separate optional subsystems

Action Items of Last Year 2

Action item 11 - Follow up on trigger config persistency solutions
Action item 12 - Check ATLAS' THistSvc
Action item 13 - Integrate Gaudi::functional with examples and docs.
Action item 14 - Lay out structure for Gaudi introductory course and documentation
Action item 15 - Investigate common ICondSvc and Handle interface
Action item 16 - Check control flow syntax with ATLAS
Action item 17 - Higher-level config concepts Gaudi could provide