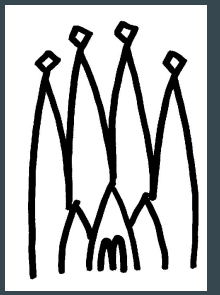
Gaudi Workshop 2017 Summary

Graeme Stewart, Benedikt Hegner
(CERN EP-SFT)

2017-10-09

Gaudi in a nutshell



- Gaudi is an event based experiment data processing framework
  - Designed to marshall physics code in an organised way to manage experiment workflows
- Originally developed by LHCb ~2000
  - Later adopted by ATLAS
  - Other users: Daya Bay, LZ, FCC
- Essential concepts are
  - Separate data and algorithms
  - Different persistent and transient views of data
  - User code encapsulated in Algorithms and Tools
  - Well defined interfaces
- Code in CERN GitLab
  - https://gitlab.cern.ch/gaudi/Gaudi

Original Gaudi interaction diagram

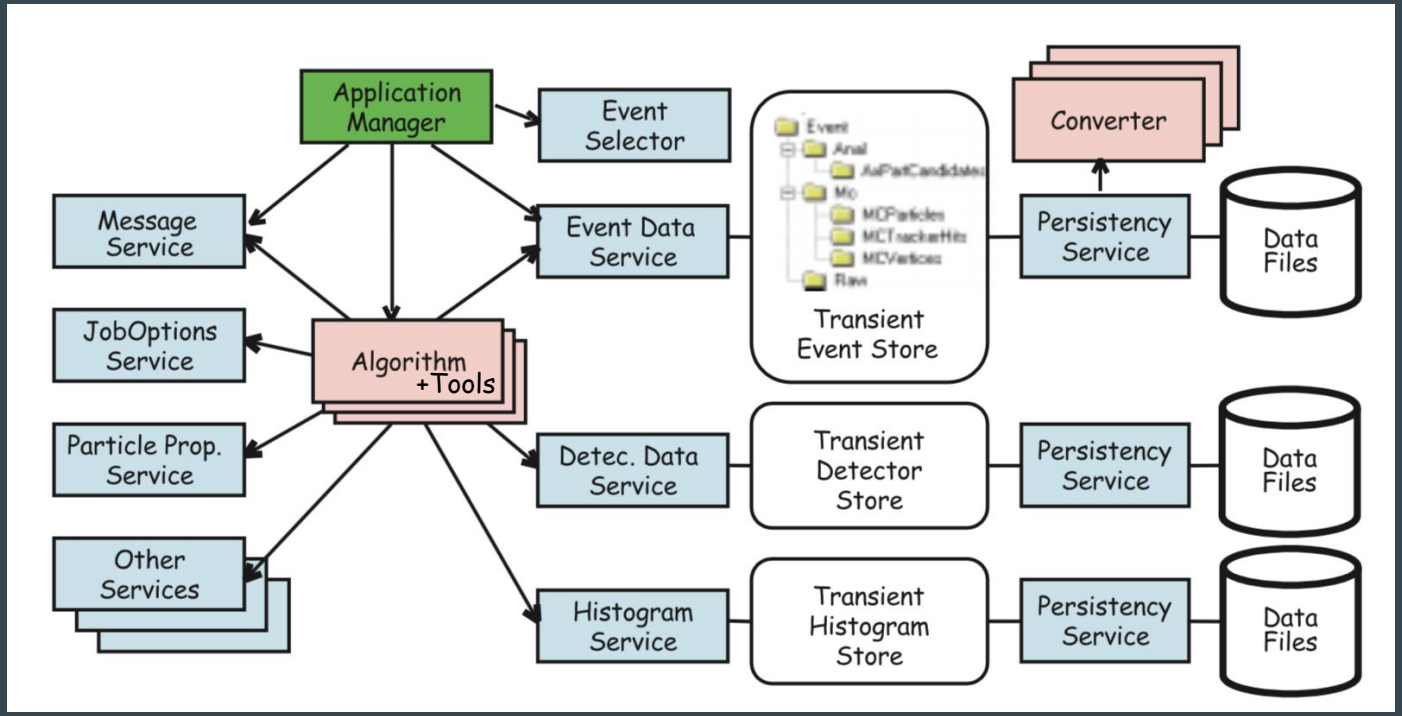


Diagram still valid, vindicating original design

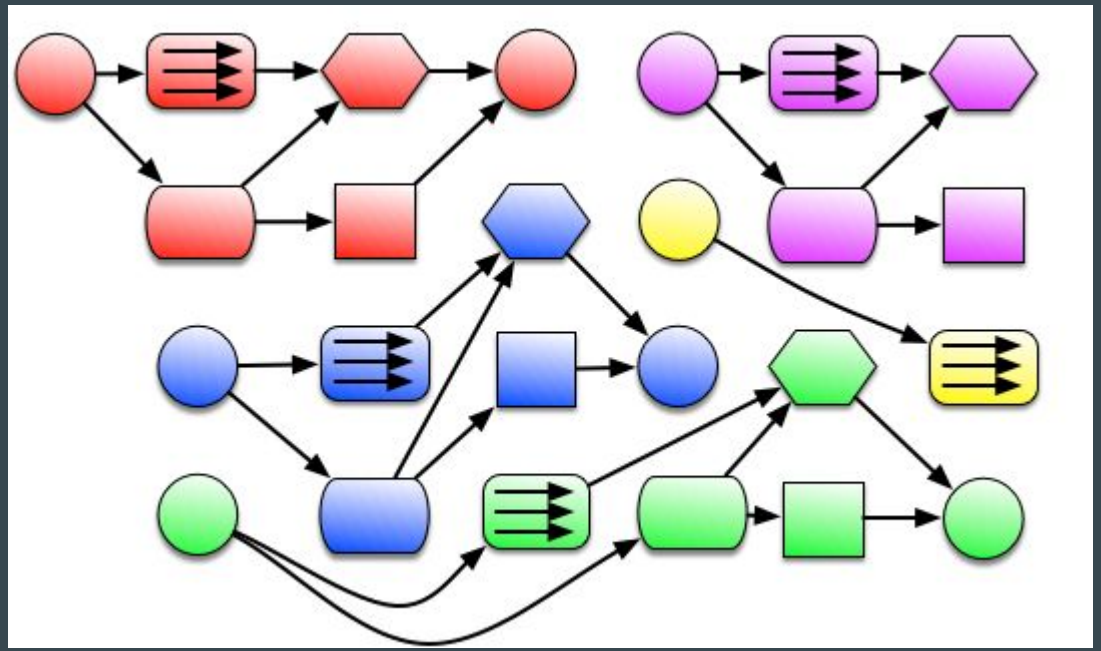
Gaudi at the LHC

- Gaudi worked extremely well for ATLAS and LHCb during Run 1 and Run 2
  - Supported the publication of 100s of physics papers
- However, there was significant divergence between the two main users
  - Early on, ATLAS and LHCb implemented different transient stores
    - ATLAS developed StoreGate
  - Each side developed further workarounds and idiosyncrasies
  - Different ‘templates’ for developing code were found in each experiment
    - ATLAS HLT use case was not served very well by the original Gaudi
      - HLT built their own scheduler into Gaudi as a single super-algorithm
    - ATLAS algorithms became containers for Gaudi Tools, that were used to encapsulate code so it could run online and offline
    - Public tools (shared by all components) became data sources instead of the event store
      - This meant that important data flow relationships between algorithms were invisible to the framework

Gaudi and Concurrency

- The challenge of concurrency really reinvigorated Gaudi
  - Multi-core machines becoming more prevalent
    - Memory consumption of multi-processing was not sustainable
  - The original implementation was firmly of the serial processing era
- Concurrency Forum provided a stimulating environment for discussions
  - General discussions on concurrency pointed to common solutions, e.g., using Intel's Threaded Building Blocks as the underlying concurrency library
  - Brought the Gaudi community back together
- GaudiHive project was started that proved that
  - Gaudi could be evolved to support concurrent event processing
    - With both multiple events in flight and in-event parallelism
  - The objective for the LHC experiments, to substantially reduce memory consumption, was achieved

Concurrent Gaudi processing sketch



Colours represent different events, shapes different algorithms

- Essential idea is simple
  - Exploit parallelism between events
  - Within events
  - Within algorithms
- Of course, there is a lot of devil in the details...

Experiments' timescales and needs for LHC Run 3

- Both experiments have to tackle many things in common
  - Core framework upgrades and refactoring
  - Large legacy code base
    - Need to engage a large developer community in a running experiment
- Still, large difference between experiments in Run 3 challenge
  - ATLAS running conditions will be much the same as in Run 2
  - LHCb will undertake a major upgrade
    - Software trigger running at 40MHz
    - Huge pressure on effective performance

Gaudi workshop 2017

- The 2017 workshop happened against this backdrop of a lot of activity in both ATLAS and LHCb
  - ATLAS migrating their code to more parallelization
    - Two AthenaMT workshops held this year
  - LHCb working full steam towards the next run
    - Frequent upgrade workshops and hackathons
- This intense activity in each experiment is very welcome, but carries some dangers
  - Lack of time for discussion has led to some incomplete convergence of opinion on core matters

We significantly diverged from the priorities set out in the 2016 workshop
Less common and more ad-hoc than hoped for

The workshop: sync()

- **Plenty of ideas, prototypes and implementations around**
- **Tried to explain to each other what we came up with**
- **Aim to identify commonality again**
  - Reach consensus *with actual code*
  - e.g. on the whiteboard the common data handle seemed resolved last year only to become later unclear again

Experiment input session - LHCb

- Actively writing their TDR
- Trying to migrate all algorithmic code to functional pattern
  - Touches transient store functionality, bringing it conceptually closer to ATLAS
  - Would like to use more compositional approach to EDM
- Counters and monitoring code needs development
- I/O efficiency and parallelism needs work
- Upgrades for detector conditions and geometry code in the pipeline

Experiment Input Session - ATLAS

- Presenting a todo list for their Athena MT migration
  - Largely on track, but scale of migration remains challenging
  - Changing away from public tool pattern is a lot of work
  - Handling conditions without callbacks requires significant rewrites
- Presenting the ATLAS trigger use case
  - To be properly supported in Gaudi for the first time
  - Requires partial event processing in *Regions of Interest* in *event views*
- Hidden dependencies
  - Algorithms that may have optional data depending on control flow decisions
    - Depend on the possible producer rather than the product itself
    - This use case also turns out to be relevant in LHCb

Event Data Models

ATLAS upgraded their data model in LS1

- xAOD provides basic objects valid for all workflows
- Objects have an *internal store* for extensions
  - Resembles the concept of composition
- This design does not naturally fit the parallelization in Gaudi

LHCb currently in the process of designing a new event data model

- Driving idea is composition instead of inheritance for AoS and SoA representations
  - Providing facade objects to users
- Rather rigid boundaries on what users are allowed to do

Discussed how to make it look more natural for users

- Leading to a follow up in the LHCb hackathon last week

Better development procedures

Plenty of small improvements

- ATLAS will drop their Gaudi fork and use branches in main repository
- The GIT repo will be cleaned from SVN ‘noise’
  - A new git-svn fork will contain all history
- Will have a *weekly meeting* to address stalled MR
  - In addition to the bi-weekly meetings for discussions
- ATLAS will finally prepare a build against Gaudi HEAD
  - Been blind for years in how ATLAS are impacted by changes in the master branch
- Introduction of labels in GitLab
- Right after workshop, we moved from JIRA to GitLab issue tracking

Code layout and cleanup

- **C++17 will be the standard for future developments**
  - ROOT compiled with the same `-std=c++17` option is evidently needed
- **The physical code layout is getting cleaned up**
  - Get rid of old CMT-driven package structure
    - E.g. unifying Gaudi*Svc packages
  - `interface/Gaudi` and `Gaudi::` as default locations
    - Special cases should be really special cases ;-)
  - Investigate the usage of inline namespaces
- **The logical code layout is getting cleaned up**
  - Remove over-abstract interfaces w/o concrete use (e.g. `IAlgorithm`)
  - Separate stateful from stateless (*) code branches

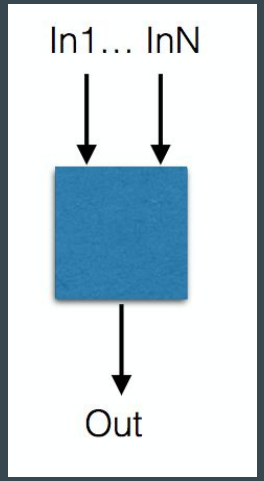
(*) '*stateless*' in terms of event loop states

Shared states and object ownership

- **Shared states in parallel applications lead to synchronization problems**
  - **⇒ try to remove them wherever possible**
- **Gaudi contains its own object ownership and reference counting system for all plugins and components**
  - Rather complex and with broken corner cases
  - Superseded by modern C++
- **Decided to make a major cleanup of *SmartIF<T>* infrastructure**
  - Use references and C++ smart pointers where possible
  - Assign ownership to respective component managers
  - This touches almost every part of Gaudi
- **Work ongoing for counters and statistics that need to ensure they gather information between threads**
  - For monitoring histograms we need to work in sync with ROOT developers

Change of paradigms - functional approach

- **Instead of algorithms as objects of unlimited complexity, treat them as simple transforming *functions without any event dependent state***
  - Users define a const function that given some input computes some output data
  - A full algorithm class is created from this function behind the scenes
- **LHCb created a template infrastructure to make this easy for the user**
  - Works very well for their upgrade project
  - Due to idiosyncratic differences in the data access some adaptor-code for ATLAS is needed
- We did decide that *functional algorithms are the preferred pattern for algorithmic code*




```
template <typename Out,
          typename... In, typename Traits_>
class Transformer <Out(const In&...),Traits_ >
{
   virtual Out operator()(const In&...) const = 0;
};
```

ATLAS and LHCb differences - data dependencies and access

- **Both ATLAS and LHCb use DataHandles inside algorithms to**
  - Let the scheduler know about required inputs and outputs at initialisation time
  - Retrieve the actual data from the event store
- **Their interfaces and usage differ significantly**
  - Different ways of aliasing data and dealing with in-place updates
- **Defined concrete interfaces and a common look & feel towards user interactions**
  - Separate the above two concerns into two classes - keys and handles
  - Make the actual store completely invisible to the user

This is a necessary step to allow for an eventual convergence on the same implementation

Event store and persistency

- **ATLAS was looking into how to simplify I/O setup removing layers of abstraction**
  - Using the LHCb infrastructure as a starting point
- **A few hurdles on the way, e.g.,**
  - Interfaces are full of DataObject dependencies
    - ATLAS StoreGate knows nothing about DataObject
  - Different event store interfaces to attach to
- **Agreed on keeping the two steps of**
  - Transient-to-persistent
  - Streaming by actual I/O library
- **ATLAS will work on a prototype, which LHCb and FCC can have a look at**
  - Dropping DataObject dependencies along the way

The proper interplay with parallelized ROOT I/O was not so clear during the workshop

Would deserve a dedicated Gaudi/ROOT meeting

Conditions handling today

- **Conditions data is necessary for event processing**
  - However the lifetime of this data is usually more than a single event
- **ATLAS and LHCb conditions data has quite different cadence**
  - LHCb conditions generally do not change during a run - overheads per event must be very small
  - ATLAS conditions do change during runs, sometimes frequently, with many independent conditions' intervals of validity
- **Two completely different approaches around**
  - Extend the event scheduler to know about conditions objects as well (ATLAS)
    - Conceptually treating these much like event data
  - Keep conditions handling separate and introduce conditions slots (H. Grasland)
    - Need to avoid code duplication (e.g., handling raw to calibrated conditions handling)

Conditions handling next steps

- **Differ significantly in complexity of infrastructure**
  - Separate handling of conditions and event data may go at costs of performance
  - Garbage collection in the ATLAS approach will be rather complex to implement

ATLAS would like to be able to compare both approaches to find out whether there are real-life performance differences

Will hopefully create convergence on user-visible interfaces for the conditions store

Change in core-components - scheduling and monitoring

- **Removing legacy code**
  - New AvalancheScheduler is an optimal implementation
    - Less overheads and better scaling
  - The ForwardScheduler and alike are history and will be removed
- **Defining control-flow via config primitives rather than sequence algorithms**
  - Proposed last year already, now converged on implementation
- **Proper recording of trigger results does not yet exist**
  - Did not define an interface yet!

Gaudi for Analysis

- **ATLAS gave it a try to use a minimal Gaudi for analysis**
  - Managed setting up a simple standalone analysis example
    - Seemed not so hard after all
  - Wrote a new transient store implementation from scratch
- **A nice starting point to**
  - Simplify Gaudi itself
  - Deprecate the stand-alone analysis framework of ATLAS
- **A few todo items though**
  - Better support for standalone builds on laptops
    - Really needed for analysts
  - Better documentation
    - Providing high quality examples will help adoption

Gaudi Copyright and License

Licenses are currently a hot topic in HEP software

- Most software not properly licensed at all
- GPL ‘restrictions’ potentially restricting experiments
  - In particular Openlab and DOE

The Gaudi license was never well defined either

- Who are actually the copyright holders? ⇒ about 20 different parties!
- **What are the licenses of SW we depend on?** ⇒ we do have (factorizable) GPL dependencies
- Which license do we want to go for? ⇒ LGPLv3 or Apache 2.0 as options

The conclusion was to wait for for Openlab and collaborating institutes whether LGPL poses any problems to them

Conclusions

- Gaudi workshop was a real success in bringing our community together
  - All of the hackathon sessions turned into further discussions on future designs and implementations
  - That demonstrated we had a large discussion backlog to overcome
- Both ATLAS and LHCb have tight timescales and are working actively on their migrations
  - Different challenges and focus
  - No fundamental incompatibilities have arisen
- Having regular weekly meetings should help us
  - Accelerate acceptance of merge requests
  - Provide regular discussion on development directions and prototypes
- SFT provides an experiment independent view on design choices, moderating between the different stakeholders