Key4HEP - Turnkey Software for Future Colliders

IAS Program on High Energy Physics (HEP 2021)

Valentin Volkl (CERN) for the Key4HEP Team
Jan 20, 2021
Software Challenges in HEP

- Long lifetimes of experiments
- Shift of priorities throughout the evolution of an experiment
  - Conceptual and design work with quick iterations relying on simulation
  - Production and dealing with the real world requires stability but also continual updates
  - Upgrade to better (sub-)detectors
  - Avoid amassing “technical debt”
- New technological developments potentially lead to new paradigms
  - How to get the most out of the available resources?
  - How to minimize the required resources?
- Data preservation
HEP Software Stack

Application layer of modules / algorithms / processors performing physics tasks (PandoraPFA, FastJet, ACTS, ...)

Data access and representation layer including Event Data Model

Experiment core orchestration layer (Marlin, Gaudi, CMSSW, ...)

Specific components reused by many experiments (DD4hep, Delphes, Pythia, ...)

Commonly used HEP core libraries (ROOT, Geant4, CLHEP, ...)

Commonly used tools and libraries (Python, CMake, boost, ...)
• Pieces of software are not living in isolation
• Ecosystem of interacting components
• Compatibility between different elements doesn’t come for free
  • Common standards can help a lot
• Choosing the right interoperability point between packages correctly is one of the main challenges
• Building a consistent stack of software for an experiment is highly non-trivial
  • Benefits can be gained from using common approaches
Key4HEP Motivation

- Future detector studies rely on well maintained software to properly study possible detector concepts and their physics reach and limitations
- Existing scattered landscape of HEP software
  - Dedicated tools for specific tasks
  - Integrated frameworks tailored to specific experiments
- Aim for a low maintenance common stack for future collider projects with ready to use “plug-ins” to develop detector concepts
- A consensus to develop such a common software stack has been reached among all communities for future colliders in the “2019 Bologna Future Collider Software Workshop”
- Identified as important project in the CERN EP R&D initiative
Key4HEP Goals

- Connect and extend individual packages towards a complete data processing framework
  - Convert a set of disconnected packages into a **turnkey** system
  - Share as many components as possible to reduce overhead for all users

- Re-use existing tools as much as possible
  - e.g. from ILC/CLIC and FCC studies

- Easy to use for librarians, developers and users
  - Easy to deploy (e.g. CVMFS, containers)
  - Easy to set up
  - Easy to extend

- Provide full functionality for different use cases

- Provide examples and documentation for simulation, reconstruction, ...

---

![Diagram](xkcd.com/927)
To facilitate interoperability, different components should talk the same language.

In HEP this is the Event Data Model:
- Describes the structure of HEP Data.
- Defining an EDM is not entirely trivial.
  - Is it possible that lepton and hadron colliders share an EDM?

Heavily inspired by LCIO and FCC-edm:
- LCIO has been very successfully shared by CLIC and ILC.

Generated via podio.
• Originally developed for ILC and CLIC but with all of HEP in mind

• Provides a complete detector description
  • Geometry, materials, visualization, readout, alignment, calibration, ...

• Covering the full life cycle of an experiment
  • Detector concepts, optimization, construction and operation

• From a single source of information
  • Used in simulation, reconstruction, analysis

• Comes with a powerful plug-in mechanism that allows customization

• More or less “industry standard” by now
  • ILC, CLIC, FCC, CEPC, LHCb, ...
  • CMS is switching to DD4hep
k4SimDelphes - First steps towards physics

- key4hep/k4SimDelphes uses delphes to do the simulation and reconstruction and creates output files in EDM4HEP format
- Quick way to get your hands dirty and do some physics with EDM4HEP
- Currently available as standalone executables
  - E.g. DelphesPythia8_EDM4HEP, DelphesSTDHEP_EDM4HEP, ...
- Part of a coherent approach to generation / simulation in Key4HEP
  - Ideally no difference between the different approaches of simulating a detector response
  - Work on full integration is ongoing

FCC-ee Simulation (Delphes)

\[ \sqrt{s} = 240.0 \text{ GeV} \]
\[ L = 5 \text{ ab}^{-1} \]
\[ e^+ e^- \rightarrow ZH \rightarrow \mu^+ \mu^- + X \]

Selection: \( N_Z = 1; 80 \text{ GeV} < m_Z < 100 \text{ GeV} \)

Z leptonic recoil [GeV]

Events / 0.40 GeV

courtesy of C. Helsens
Experiment Framework

• Traditionally HEP has not done too well with sharing efforts towards a common experiment framework
  • Notable exception is Marlin used by ILC and CLIC

• Gaudi, originally developed by LHCb, now also used by ATLAS, FCCSW and smaller experiments
  • Supports concurrency
  • “Battle-proven” from data taking during LHC operations
  • Currently undergoing a modernization

• Key4HEP has decided to adapt Gaudi as its experiment framework
  • Contribute to its development where necessary

• Integration and migration of iLCSoft algorithms into Key4HEP with the help of a Marlin→Gaudi wrapper
  • Allows to use Marlin processors within the Gaudi framework
  • v00-03 recently released: [key4hep/k4MarlinWrapper](key4hep/k4MarlinWrapper)
Key4HEP Framework - k4FWCore

- **k4FWCore**
  - Core Key4HEP framework providing core functionality, e.g.
    - Data Service for podio collections
    - Overlay for backgrounds
  - Recently switched to Gaudi v35

- **k4-project-template**
  - Template repository showing how to build new components on top of the core Key4HEP framework

- Ongoing work to collaborate more with Gaudi ecosystem (Gaussino)
- Ongoing work to integrate more components (ACTS, ...)

[Key4HEP framework - k4FWCore](key4hep/k4FWCore)

[Template repository showing how to build new components on top of the core Key4HEP framework](key4hep/k4-project-template)
Software Infrastructure

- Documentation
  - [key4hep.github.io/key4hep-doc](https://key4hep.github.io/key4hep-doc) (main documentation)
  - [cern.ch/edm4hep](https://cern.ch/edm4hep) (doxygen code reference)
- Modern CMake configuration for the individual components
  - Facilitates their usage as dependencies for other components
- Automated builds and continuous integration (CI) where possible
  - Regular nightly builds of the complete stack
  - Use of the spack package manager
- Distribution via CVMFS
  - Latest release can be found at /[cvmfs/sw.hsf.org/key4hep](https://cvmfs/sw.hsf.org/key4hep)
- **Release early and release often**
  - Make fixes available early
  - Discover problems and collect feedback as early as possible
Spack for Key4HEP

- **Spack** is a package manager
  - Does not replace CMake, Autotools, ...
  - Comparable to apt, yum, homebrew, ...
  - Independent of operating system
  - Builds all packages from source

- Originally developed by the HPC community
  - Emphasis on dealing with **multiple configurations** of the same package
    - Different versions, compilers, external library versions, ...
    - Several versions of the same package can coexist on the same system

- The whole Key4HEP software stack can be built from scratch using spack
- Spack allows different workflows for setting up consistent software stacks
  - Currently testing which one fits our purposes the best
Current status

- “Real work” has started in the beginning of 2020
- Key contributions from ILC/CLIC, FCC(-ee and -hh), CEPC, ...
- Spack based installation exists on cvmfs and can be setup via
  `source /cvmfs/sw.hsf.org/key4hep/setup.sh`
- First basic examples exist
- We try to keep the documentation up to date even though things move fast sometimes
- **FCCSW is currently switching to Key4HEP**
  - Migration is aimed to be done in spring 2021
How to collaborate

At the moment more work than people!

- Active weekly meetings, alternating between EDM4HEP and Key4HEP
  [https://indico.cern.ch/category/11461/](https://indico.cern.ch/category/11461/)

- Check out documentation at [https://cern.ch/key4hep](https://cern.ch/key4hep)
  - Also contains some examples

- **Any feedback is welcome**, this should not just be an academic exercise!

- If you find any issues, **do not hesitate to report them**
  - Documentation not up to date?
  - Examples not working?

- **We also greatly appreciate pull requests**
Summary & Outlook

• Setting up a consistent and working HEP software stack is non-trivial

• General agreement to move to a common HEP software stack for future experiments
  • Re-use existing tools where possible and ensure their interoperability
  • Make everybody benefit from new developments

• Gaudi will be used as core framework, but integration / migration strategy for Marlin and iLCSoft exists

• Since the meeting in Bologna in June 2019 a lot of work has been invested to get working prototypes for the core components and a common EDM4HEP

• A lot of work still ahead!
Pointers to Resources

- Key4HEP
  - [key4hep.github.io/key4hep-doc](key4hep.github.io/key4hep-doc)
  - [key4hep](key4hep)
- EDM4HEP
  - [key4hep/EDM4hep](key4hep/EDM4hep)
  - [cern.ch/edm4hep](cern.ch/edm4hep)
- k4SimDelphes
  - [key4hep/k4SimDelphes](key4hep/k4SimDelphes)
  - [delphes/delphes](delphes/delphes)
  - [cp3.irmp.ucl.ac.be/projects/delphes](cp3.irmp.ucl.ac.be/projects/delphes)
- podio
  - [AIDASoft/podio](AIDASoft/podio)
- DD4hep
  - [AIDASoft/DD4hep](AIDASoft/DD4hep)
  - [dd4hep.web.cern.ch](dd4hep.web.cern.ch)

[xkcd.com/138](xkcd.com/138)
Supplementary Material
k4SimDelphes

- Produces several output collections
  - One collection of `edm4hep::ReconstructedParticle` that comprise all reconstructed particles of an event
    - Photon, Muon, Electron collections that point into this collection
    - Jet collection with constituents from this collection
  - One collection of `edm4hep::MCParticle` for the generated particles
  - Associations between MC and reco particles
- Configurable output contents
  - Which delphes collections should be considered?
- No changes to delphes cards necessary
- Conversion is done on the contents of the delphes `TreeWriter`
  - Could be used to convert already generated files
  - Least invasive method, that does require only minimal changes to the Delphes code
Delphes - Overview

- Delphes is a fast detector simulation response tool based on a parameterized description of the detector
- Understands several common HEP input formats and produces output files using its own EDM
- Needs a “delphes card” that describes the detector
  - Parameterized efficiencies, resolutions, ...
  - Available for ILC, CLIC, FCC-ee/hh (and others)
- Can do basic reconstruction
  - Jet finding, b-tagging, isolated object finding, ...
  - Again based on the parameterized inputs
podio as Generator for EDM4HEP

- Original HEP c++ EDMs are heavily Object Oriented
  - Deep inheritance structures
  - Objects scattered in memory
  - Hard to deal with in multi-threaded environments

- Data access can be very slow with these approaches

- Use **podio** to generate thread safe code starting from a high level description of the desired EDM
  - Target different I/O backends for persistency
  - Users are isolated from implementation details

- Treating python as first class citizen and allow “pythonic” usage

- Recently has experienced revitalisation but to become production ready some work is still needed

```cpp
class MCParticleData {
  int PDG;
  float charge;
  double mass;
  Vector3d vertex;
};
```

```yaml
(*podio code generator)
+ =
```

AIDASoft/podio
The Philosophy of podio

- **User Layer** consists of handles to the EDM objects and offers the full functionality

- The **Object Layer** handles resources and references to other layers

- The actual POD data live in the **POD Layer**

- Easy to use through a clear design of ownership
  - Users should not have to care about this

- Layered design allows for efficient memory layout and performant I/O implementation
  - ROOT I/O is used by default
  - An SIO based implementation is available
Building the Key4HEP stack from source using spack

- Setting up the central spack repository
  ```
git clone https://github.com/spack/spack.git
source spack/share/spack/setup-env.sh
  ```

- Add Key4HEP specific packages not yet present in the central repository
  ```
git clone https://github.com/key4hep/key4hep-spacc.git
spacc repo add key4hep-spacc
  ```

- In order to have consistent stack, some packages need some additional configuration
  - e.g. pinning versions, or specifying build options
  ```
cp key4hep-spacc/config/packages.yaml spack/etc/spack
  ```

- Install it
  ```
spack install key4hep-stack
  ```

- See also: [the documentation](#)
Some Experiences with Spack

- Collaboration with spack developers pretty smooth
  - Some HEP colleagues have merging rights for the central spack repository
  - Some HEP packages actively maintain their package recipes (ACTS)
- Rapid development in central spack repository
  - Stable builds need to pin the used spack version
  - Potentially miss latest features
  - Documentation in general very good, but sometimes not completely up-to-date
- Spack developers very responsive, but roadmap is sometimes a bit opaque
  - E.g. the concretizer development have taken very long to be integrated
- The build recipes are very nice to persistify build-system know-how

```python
conflicts("%gcc@8.3.1",
    msg="There are known issues with compilers from redhat's devtoolsets"
    "which are therefore not supported."
    "See https://root-forum.cern.ch/t/devtoolset-gcc-toolset-compatibility/38286")
```
Usage of spack for developing software is pushing spacks intended purpose, but it is possible

- Spack can build directly from git branches
- Development can be done “as usual” after using spack to build dependencies
- Spack can be used to setup environments
- Spack dev-build compiles local code according to recipe

Development workflows are also being worked on by spack developers

- Mixing system libraries and spack installed libraries can lead to some unexpected issues
  - Missing version information for some libraries that is expected by system packages (usually no problem)
  - Reproducible environments from setup scripts need some work

- Many packages from LCG releases already available as spack recipes