HepMC 2 $\rightarrow$ 3 plans / proposals

Andy Buckley
University of Glasgow

HepMC development workshop, CERN, 21 Jan 2014

Introduction

By this point we've hopefully had a good discussion of the organisational issues in HepMC development – organising an active development/maintenence team is probably the biggest issue!

Issue #2: HepMC is *horrible* to use!
⇒ broad agreement among (would-be) developers that a "v3" rewrite is needed to address fundamental issues.

This will clearly need to maintain "99%" backward compatibility (cf. LHAPDF 5 → 6) Likely tied in with a rethink of the persistency (and I/O speed improvements)

Issue #3: More standardisation would help
⇒ Can we agree on a standard treatment of barcodes and status codes in afterburner generators & detector simulation?

Issue #4: Minor functionality enhancements needed
⇒ HI impact parameter, generation provenance info, ...

I'll visit these in reverse order, i.e. increasing in complexity!

This is largely a brain-dump (after a lot of thinking and code-prodding), so let's discuss "inline"

Minor improvements (version “2.7”)

No-brainers (?)

- Make iterator ranges work with Boost and C++11 `for`
  - Works on trunk for C++11, needs wider testing
- Provide direct particle range access from particle – how often are vertices really needed?
  - But does the internal iterator design screw this up? Access only to *ranges*, not begin/end?
- Remove unnecessary virtual declarations – they just slow the code down

No-brainers (2)

- Return e.g. `momentum()` by (const) reference for better efficiency
- Add new HeavyIon IP variable to I/O ASCII format. Units? Not mm/cm!!?
- `Polarization` and `Flow` should be nullable. . . *minor* API change, but can it happen in 2.7?
  - Can't overload on return type, unfortunately
- ASCII I/O speedup, cf. Salam's fast ASCII parser (avoid `istringstream`)
  - No format change for this one.
  - Also note possibility of `LD_PRELOAD=uncompress.so`

Some-brainers

- LHE weights: existing named weights are fine. . . right?
  - But WeightContainer API is horribly dysfunction: can't even get a list of keys!
  - Add `keys()`. Other missing features? Would be good to test
  - Suggestions about embedding LHE blobs in HepMC ASCII!!!
- Mass treatment mess
  - Docs say `GenParticle::mass()` returns `generated_mass` if set, otherwise computed mass
  - But that's not what's implemented! Always returns computed mass: often unphysical due to $E^2 - p^2$ computation
  - No flag to declare whether generated mass has been set. Default of 0 is physical (and one of the most useful values!)

  $\Rightarrow$ Implement $m^2 = (E - p)(E + p)$ in `FourVector` for improved numerical stability

  $\Rightarrow$ Could it be assumed/enforced that a set generated_mass must be positive, and use default value of -1 as implicit flag?
  - General question re. on/off-shellness in MC records, and whether a mass-based or "5-vector" momentum rep is needed

Some-brainers (2)

- Constness consistency/completeness issues, e.g.

```
const GenVertex* gv = get_vertex_somehow();
GenVertex::particle_const_iterator pi =
    gv->particles_begin(HepMC::ancestors);
```

  Last fn doesn't exist: need to cast to non-const for const access! Deep roots...
- Feature *removal*:
  - Can event-level alpha and scale values be dropped?

  ⇒ Unused (almost?) always? Meaning unclear! Duplication with PDFInfo
  - Same re. vertex weights
  - Probably needs to be v3, but mentioning here because the impact is probably minimal if the ASCII I/O can be updated

Event record standards (no code version)

Standards

- Clarify what status 0 and 3 *mean*!
- Standardise Les Houches 2013 vertex status/ID proposal? (And add `GenVertex::status()` alias?)
- Status and barcode extensions:
  - ATLAS has an ad hoc scheme. Not even well documented or implemented in ATLAS. Other expts?
  - |barcode| $>$ 10k for afterburners? Or more?
  - |barcode| $>$ 200k or 1M for detsim?
  - Afterburners should rewrite status 1 $\rightarrow$ 2? ATLAS uses a mod-1000 scheme for detsim mods
- Standard non-interacting BSM particle = 99 or 19 or 1000022?
- Ban directed cycles in the event graph?

Overhaul (version “3.0”)

The problem

The code is a complete mess!
And is unnecessarily painful to use

Not pointing any fingers!

Even when tidied up, as has been done, lots of the implementation is unmaintainable (e.g. iterators, I/O)

⇒ not *impossible*, but beyond the time & pain threshold of those who could work on it

Some features clearly driven by HEPEVT compatibility. Moot point now: new gens don't use HEPEVT, and none use HepMC internally

Certain problems have their roots in fundamental design choices, e.g. desire for STL emulation

⇒ Some fixes unavoidably involve interface changes

Requirement of common event record is to pass enough info between gen/sim steps, and *be expressive for quick, clean analysis*

Some of this is a personal view of "Rivet/ATLAS HepMC requirements"

Expressiveness

An example

Good:

```
for (const GenParticle* gp : gv->particles()) { ... // C++11
//
BOOST_FOREACH (const GenParticle* gp, gv->particles()) { ... // Boost
```

Bad:

```
for (vector<const GenParticle*>::const_iterator ip = gv->particles().begin();
  ip != gv->particles().end(); ++ip) {
  const GenParticle* gp = *ip; // STL
//
for (size_t ip = 0; ip < gv->particles().size(); ++ip) {
  const GenParticle* gp = gv->particles()[ip]; // C! Not so bad, actually...
```

Ugly:

```
for (HepMC::GenVertex::particle_iterator ip =
  gv->particles_begin(HepMC::relatives);
  ip != gv->particles_end(HepMC::relatives); ++ip) {
  const GenParticle* gp = *ip; // STL
  // HepMC! No const iterator available! Presumably authors lost will to live
```

Compatibility

There's a *huge* codebase of "crappy HepMC cut 'n' paste". Typically not actively maintained...

No point in putting a lot of work into a rewrite which can't be used

Any rewrite will have to clean out the implementation but preserve most of the API – or provide a compatibility wrapper

Use `Gen*` class names for wrappers, `HepMC::Particle` etc. for new implementation?

I hope that we can allow removal/changing of little-used features or minor changes such as constness

A migration to v3 should be **rapidly iterated with users**: one HepMC developer per expt/MC community would help greatly

API rethink

Design proposal

- **Canonical list of event contents as single event-level vectors of vertex & particle pointers**
  - Faster access for most common access pattern, and ROOT auto-persistence compatibility (good idea?)
  - Creation/deletion here. Consistency managed through graph. Any value to using smart pointers? (Boost or C++11?)
  - Use a smaller "technical vertex" object if there is no new displacement, and compute the displacement recursively from ancestors?
- **Graph edge/vertex relations via `vectors` of pointers**
  - Returned by reference. The other major access pattern.
- **Ancestors/descendents iteration is a harder problem**
  - Writing iterators to efficiently walk a graph in a predictable order without duplication is not easy
  - And no-one knows exactly how the current implementation works (or if it is safe) due to the code mess!
  - Could be done with an iterator containing a stack of vertex/particle pointers + vector indices? (cf. a recursive call stack)
  - Otherwise compute and return a vector of ptrs on demand: inefficient! Recommend recursive (lambda) functions in this case

API rethink (2)

Design proposal

- **In this picture we discard (most) explicit range objects:**
  - `vector`s naturally support the range protocol
  - Iterators can now just be standard STL iterators "for free", without needing to be defined within GenEvent/Particle/Vertex
  - Need a range proxy for ancestors/descendents if an iterator can be written: easy, but a vector view would also be nice
- **Suggestion to aim for compatibilty with graph libraries e.g. BGL, NetworkX.**
  - Any real value? What graph analysis do we want? (Maybe we do...)
  - Obsession with STL-(in)compatibility is a major current design flaw (because the STL itself is quite nasty). Let's not repeat that: don't compromise the natural/established use-case for hypothetical ones.
  - Is this even *possible* while maintaining compatibility?

I/O

- I/O class interface hierarchy is a nightmare: discard and rethink
  - Discard all I/O interfaces other than one for format compatibility with `IO_GenEvent` files
  - Should be possible (easy!) to make it a *lot* easier and simpler to use (don't need to emulate STL)
  - Allow "plugins"?
- New ASCII format: generalisation to store more strings? Escaping? YAML header / custom body cf. LHAPDF6?
  - *Extremely* vague at present: need to consider carefully and iterate proposal
  - Format improvement already suggested for more compact ASCII representation: see Les Houches 2011.
- ROOT and other binary formats

Misc

- **Constness**
  - AGAIN! Small changes, needed for consistency. Part of general API overhaul.
  - Could fake the old inconsistent way with `const_cast` in wrapper, but *really*?
- **`GenRun` for e.g. cross-section**
  - Some important information like generation provenance, cross-secs is not really event-level
  - Persistency maybe needs to declare store some (particularly cross-sec) in the event blocks, but API does not
- **Units**
  - The current system's implementation was never functional enough to get significant uptake
  - Quite a mess! Can it be done much simpler, i.e. just declare the current units and provide rescaling functions for the *user* to call?
  - Any possibility for an agreement on HepMC units? Or expt/theory conventions unbridgeable?

Questions

- Are off-shell momentum vectors needed? Otherwise we could use a safer mass-based vector (implementation in mcutils)
- For polarization does a spinor basis need to be provided as well as a pair of angles? Feasible?
- Any other physics objects that you would *really* like to be able to pass to experiments / other generators?
- Your own questions...

Summary

- HepMC's current codebase is a horror to work with: inconsistencies, surprising/unclear ownership behaviours, unnullable quantities, etc.
- But there is a *lot* of established code which depends on it: any rewrite needs to avoid widespread breakage since massive migration work will probably not happen
- Propose to first fix a few of the obvious flaws in version 2.x: is *any* incompatible API change permitted?
- More standardising of event encoding would also be welcome.
- HepMC 3 rewrite has been much discussed: I think we are approaching a big-picture design plan. Need a small, active (5–10% of time – or more if you can do it!) group of developers. Input/feedback from expts/MC authors, but *lightweight*, please
- Timescale: we are probably now too late for a full HepMC 3 rollout by the end of the year. Unless work really starts in earnest now, we probably need to make changes as non-disruptive as possible, to allow upgrading in early LHC Run 2.