Report from Geant4 Architecture Review

JOHN APOSTOLAKIS

FOR THE ARCHITECTURE REVIEW TEAM:

M. ASAI, J.APOSTOLAKIS, G. COSMO, H. KURASHIGE, D. WRIGHT

Overview

- What is Software Architecture
  - And why do we care about it?
- What the Review was asked to do
  - Driving issues
- What the Review has studied
  - Highlights of topics identified
- Outcome

What is software architecture ?

- It is the structure of the system (of a program or computing system).
- In the case of Geant4, this includes
  - the software components (high-level categories, interaction of classes and their roles within the categories),
  - the externally visible properties of those components (public interfaces, external dependencies and usability aspects),
  - the dependencies and relationships between them, and
  - some most basic design concepts/decisions.
- The software architecture also includes the documentation of the system
  - The part which describes the public interfaces and the system's architecture itself.

Components of Software Architecture

For Geant4 these are:

- the key **interfaces** and elements of design (categories / classes),
- The **use-cases** that drive the flow between categories,
- the **major choices** made in the design,
- the **constraints** imposed on the design or implementation of Geant4, *and*
- the **mission statements** of key classes or elements of the design.

Architecture: Key interfaces and classes

- External interfaces and dependencies:
  - CLHEP three-vector, Lorentz-vector, transformation matrices, random-numbers generators
- Infrastructure classes:
  - G4Exception, G4PhysicsTable, G4PhysicsVector
- Main interfaces, classes
  - Material and Material-Cuts-Couple
  - Physical-Volume, Logical-Volume, Touchable
  - Particle, Track, Step, Step-Point
  - Particle-Change
  - Process.

Why should we care about Architecture?

- It is the core / skeleton of the Design of Geant4
  - Almost everything in Geant4 is dependent on it
- Most/all key aspects are unchanged since 1997
  - Later choices and additions have been shaped and constrained by those old big choices.
- Many choices have been made to adapt to our Architecture
  - Some may be clumsy or difficult to maintain
- The computing world has changed in the last 15 years, and will be changing in the next 5 / 10+ years
  - Reviewing and re-examining some of our key choices will enable us to adapt and prepare for the future evolution.
- Else we proceed “like we used to”
  - And key parts of the design and code will be impossible to understand or change in a few years time.

So we want to

- Identify issues for improvement
  - Design, interface problems or opportunities
- Prioritize topics
  - For prototyping potential revisions / initiatives
  - For improving the key interfaces.
- Prepare for future design revisions

- Cannot do this randomly – we agreed on principles to guide this
  - Driving issues / forces (next)

Driving Issues / “Forces”

- Identified a Set of goals as ‘driving forces’:
  - Computing performance (CPU time, memory use allocation)
  - Multi-core and future architectures (adapting)
  - Maintainability and code readability (including correctness, simplification, reduce complexity)
  - Usability
- We want to identify limiting factors in the current architecture (high-level design) for these topics
  - So that future revisions can be planned to address them.

Computing Performance

Key topics which impact computing performance

- Challenges of recent and future computing architectures
  - See next slide
- Growing cost of accessing (main) memory
  - Locality of memory use (cache)
  - Number of memory allocations.
- Costs of use of key design and implementation choices
  - Object oriented paradigm, shared libraries;
  - Number of branches, virtual methods in OO code – and their cost.

Note: Our understanding of some of these limitations (esp. frequency of calling methods) is an emerging issues – is still evolving quickly.

New / emerging Computer Architectures

- Large and increasing cost of memory operation
  - Today it is 10-40 cycles – relative to computations (1-4 can be done each cycle)
  - Growing dependence on caches (recent CPUs have 3 levels)
- New CPUs and systems
  - Multi-core CPUs (2, 4, 6, 8 cores today)
  - Memory is attached to each socket (delay to access 'remote' part) – Non-Uniform Memory Access (NUMA)
- Future architectures
  - Even more cores per socket (one hundred up to hundreds!)
  - Graphics Processing Units – much power, different coding;
  - Heterogeneous architectures (mixing CPU, GPU on 1 die).

Key extensions

Key interfaces have been exercised (used or abused) to achieve

- Extended use of geometry ‘stack’ via touchable-handles
  - in processes, secondaries, and potential use by users' hits
- Introduction of biasing techniques:
  - leading-particle biasing for EM and hadronics - via wrapper process or not.
  - importance biasing (splitting & russian-roulette)
- Error Propagation (Geant4E)
- Reverse Monte Carlo
- Extensions related to geometry
  - Parallel Geometries with joint/coupled transportation
  - Regions & cuts per region
  - Regular navigation (ignoring -some- boundaries).

New developments are being planned or considered for the future:

- Materials whose density varies by volume.

Additional aspects considered

Concerns for how the design influences

- Complexity
  - of lower level design and implementation
- Correctness
  - Ensuring correct results, and finding where errors start/occur
- Open doors
  - Pitfalls of current design; unexpected side-effects
- Redundancy
  - Having two or more ways to achieve the same result.

What have we been doing in the Review?

- Examining
  - Key Geant4 interfaces
  - The relationships between categories
- Identifying
  - What problems exist with the current high-level design
- Understanding

Major design choices

- **Tracking** is *generalized* and unique in type: i.e., all particles use the same tracking code.
- **Physics** and many capabilities use the unique *process interface*
  - Physics processes
  - Transportation (=encountering the geometry)
  - fast simulation
  - Biasing and scoring
- Thresholds are used to limit production of some particle types
  - Current choice spans e-, gammas, e+, and ions.
- Processes are expected to be independent (or almost)

Issues in Key Areas

Geometry

- Geometry navigation changes the state of the ‘live’ geometry tree
- Strong relation with other categories
  - Tracking – via transportation process
  - Electromagnetic physics – via multiple scattering and its use of displacement and isotropic safety
- Performance critical
  - Typical cost of geometry is order (30%) in complex setups
  - Large memory use

Geometry and Touchables

- Geometry navigation changes the state of the 'live' geometry tree
  - ***Action***: evaluate revision to separate changes during tracking to be outside live geometry tree – i.e. not in volume classes.
- Interfacing with other categories
  - Need 'policy' for accuracy and ownership of 'isotropic safety' (navigator, tracking, .. ?)
  - Fragile protocol for calling the navigator by (EM) processes
- Performance related issues
  - New touchable created at each boundary crossing (potential for memory churn, memory use and performance bottleneck.)
  - Optimisation voxels are a major consumer of memory.

Particles

- Status
  - ParticleDefinition for static properties (mass, charge - usually)
  - DynamicParticle for dynamic properties (energy, momentum)
- Challenges
  - New types of particles, e.g. proposed particles, “reverse electron”, ..
  - Handling of ions (creation, initializing physics) – see next

Particles 2: Ions

- Challenge of dealing with ions
  - Must deal with 3000 stable nuclei (or more in future) – so today they are created on the fly (only those that are required)
  - G4DynamicParticle describes **all** the properties of ions; i.e. nucleus and orbital electrons.
  - Because of the large number of 'stable' nuclei in Geant4 (about 3000), the G4ParticleDefinition for each nucleus is created on the fly, as soon as it is required during the simulation.
- How to create a new nucleus, during the simulation ?
  - Some class must construct a process manager for it – dependency issue!
    - It must copy the list of processes from the process manager 'GenericIon'.
  - Today the particle instantiation triggers the PhysicsList (run category) via messenger commands, to create the processes and initialization them physics.
    - This operation is more challenging in the case of multi-threading.
- ***Action***
  - ***evaluate using a call-back mechanism as a solution. This is new potential feature for Geant4, so expect to need extra attention.***

Particles (cont.)

- Clarify policy for creating and tracking unstable / exotics:
  - which particles can be tracked,
  - which can be created internally, but must then be decayed (or forcibly interacted) and not tracked.
  - Document current choices - which can depend on the physics list.
- Document requirements for extending tracking to additional particles (e.g. hypernuclei, anti-ions.)

ParticleChange

- It's mission is to act as go-between from processes to tracking for proposed changes of state of a track (including secondaries).
  - It enables tight control of the code which changes the track status – all of it must be in particle change.
  - As it is responsible for modifying the state of a track after each interaction, its implementation affects performance.
- ***Action***: simplify its design and implementation
  - The collection of classes (G4VParticleChange, G4ParticleChange, and several derived classes) is somewhat complex.
  - Goal: simplification and usability first in mind; performance potentially as large an issue.
- Note: Initial assessment is that this is a smaller design/architecture issue than others already identified.

ParticleChange – cont.

- As go-between it is a key class for enforcing checks of conservation laws, and identifying other errors in output of processes
  - It is performance critical
  - So checks must not come with a performance overhead (for production)
- ***Action***: Study whether and how to isolate the performance cost of extra checking
  - Can a "Check mode" be created for this - a new mode for compilation and/or run-time?
- Consider creating a new strategy for verbosity and debugging information
  - Which provides good balance between computing performance (in a production run) and the tools to identify the source of a problem or crash.

G4VProcess

- G4VProcess is
  - Is the key interface between processes and tracking
  - Is pervasive in large parts of Geant4
  - Its design and implementation touch on critical issues of performance, maintainability and usability.
- The main issues for G4VProcess covered by the review are
  - the Get[Type]PhysicalInteractionLength methods, and
  - the process flags used to decide process priority.

G4VProcess: Process flags

- Process flags (and proposed step lengths) are used to determine which process is chosen to occur, and its exclusivity
- There are many Process flags returned by PostStep GPIL
  - forced, strongly forced, exclusively forced,
- The complexity of the conditions is an issue. It drives the complexity of the key SteppingManager class.
  - Examined how many processes use each of these conditions ?
  - Get[Type]PhysicalInteractionLength methods treat safety inadequately. Returning of Isotropic-safety requires improvement. Currently only GetAlongStepGPIL methods are involved in safety (this is likely adequate.)

Process Status – options and motivations

- "Inactive": allows a process to be turned off
  - typically by the user via UI command
- "NotForced": occurs when a process with a discrete action proposes to limit the step
  - Standard way, used by all 'ordinary' discrete processes;
- "Forced": the process requests to be called at every step.
- "StronglyForced”: invoke at every step, also the last step
  - Created for scintillation.
- "ExclusivelyForced”: call *only* this process – no others
  - Even overrides strongly forced. Key use case is for fast-simulation.

Recommendations

In draft priority order

- ***Prototype for simplified tracking (e+, e-, γ)***
  - *Goal: find if it gives a speed advantage (potentially if used together with stacking by type of particle).*
- ***Identify whether there are (major) design implications from addressing the following requirements:***
  - *Interference between Hadronic and Electro-magnetic (elastic) processes*
  - *Kaon Oscillation and Regeneration*
  - *Varying density materials*
- ***Documenting key elements of current design***
  - *Reviewing our design and scenario diagrams (by hand?)*
  - *What parts of step/track are invalid and during what part of the Stepping ?*

Recommendations (cont.)

- ***Examine extending the use of "const" wherever necessary or beneficial***
  - *to protect classes and usage (e.g. const track, touchable, solid, …)*
- ***Additional checking***
  - *Engineer a "check_mode" run modality for extra checks of key results*
    - ***processes must check their results for all applicable conservation laws***
    - ***common code to help doing this is vital - in order to avoid new/bad code proliferation (e.g. energy conservation checks) o***
    - ***Envisage use of assertions as blunt instrument for conditions of total failure***
- ***Multi-process / multi-threading:***
  - *Choose whether to use both of these - and invest in adapting (or not)*
  - *Decision on whether to adopt multi-process and/or multi-core – needed to plan adaptation, testing, validation.*

Recommendations (cont.)

- ***Multi-process***
  - *Improve clustering of read-only data (for mp - also for single thread?)*
- ***Multi-threading***
  - *Clarify proposal to use code annotations to generate multi-threading version "automatically"*
  - *Key classes would need to be reviewed*
    - ***potential side benefit in improving headers - clarity of state, data which is initialised at start and only used in event loop, "scratch" data members, .. ?***