Modularization of Geant4

Dynamic loading of modules

Configurable build using CMake

Pere Mato

Witek Pokorski

13.09.2012

Motivation

- Geant4 has successfully brought together many experts from different physics domains (HEP, space, medical, etc) as well as related fields (geometry, visualisation, etc) to contribute to the toolkit
- number of models and complexity of the software has grown to the extend of being difficult to handle as one monolithic bloc
  - compile-time dependencies introduce bindings of the code needed in different (unrelated) domains
  - very little configuration possible at the build time
    - only to switch off modules depending on external software
  - at run time, many not-needed modules (physics models, etc) are loaded into memory
- users (LHCb requested it already) and developers would gain a lot if Geant4 became a framework with dynamically loadable plugins

Examples

- physics lists (builders) introduce a compile and link time dependencies between all physics models in Geant4
  - no way to choose to compile only EmDNA physics, or only HEP physics
  - developers blocking each-other due to dependencies in SVN tags
- libraries (for ex. G4processes) loaded at run time always include all models
  - waste of time and memory (e.g. the 3 top models counts for 30% of G4processes and not used in standard HEP Physics Lists)
  - LHC experiments could gain ~1-2% in memory requirements
    - 1-2 % of the overall memory requirements of 1-2 GB gives ~20MB*100k concurrent jobs = 2 TB of RAM = 20k $
- never all the Geant4 functionality is needed at once
- G4visualisation not used in batch jobs, etc
  - G4analysis not used in experiments frameworks, etc

Proposal

1. remove all possible compile-time dependencies by using factory-based mechanism
2. introduce configuration capability in Geant4 CMake system to allow detailed selection of the required modules
   - should also be accompanied by an SVN tool allowing selective check out of the code
3. introduce a plugin-based, dynamic (on demand) loading capability in Geant4
   - only ‘core’ Geant4 should be linked with the application
   - all specific modules (models, tools) should be loaded dynamically

Ad 1.

- Factory based mechanism allows to instantiate objects without including the concrete header file in the client code
  - concrete classes register themselves in some central registry
  - avoids compile- and link-time dependency
- already done for cross section classes extracted from CHIPS
  - can be applied at any level in Geant4 (tools, modules, physics lists, models, processes, etc...)

Ad 2.

- with compilation dependencies removed (point 1) CMake provides an easy way of configuring (selecting) modules to be built
  - already used for GDML, for instance
- default CMake configuration can build everything but specific flags could switch off not needed modules
- a small script for checking out from SVN only the needed directories would help a lot

Ad 3.

- dynamically loadable plugins will allow users to ‘build’ their applications at run time, using the needed minimum of modules, with no need to recompile to add anything extra
- Geant4 ‘main’ would allow loading of the needed functionality at runtime
  - possibility to choses physics models
    - or to change the models without the need to recompile
  - load necessary tools (visualisation, analysis, etc)
- the same ‘main’ would be used in all cases (development, validation, batch jobs) with just different option (configuration) files

Example

- building of Physics List

names of loadable plugins

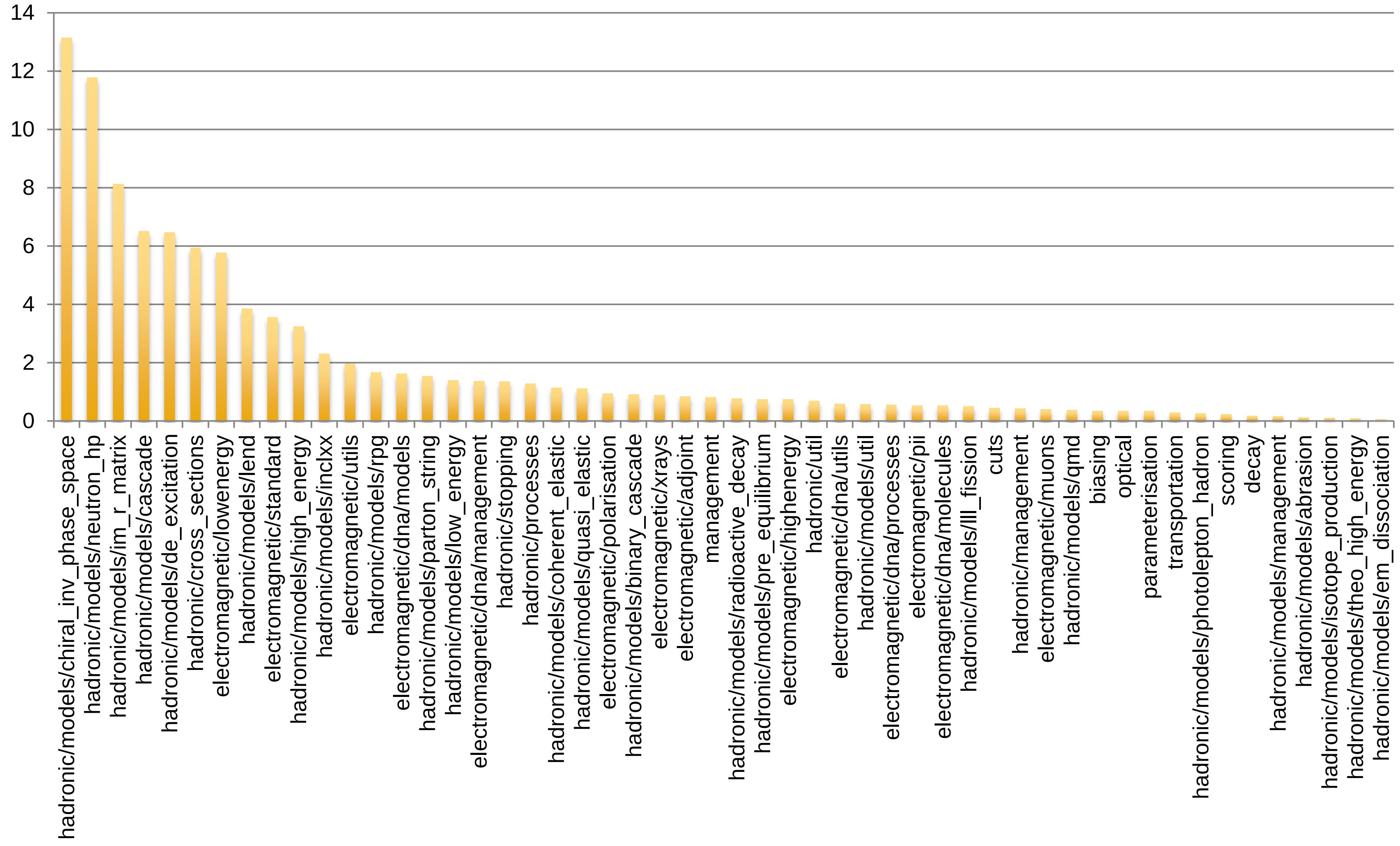
```
G4PhysicsPluginManager->AddPhysics("EmStandardPhysics")
G4PhysicsPluginManager->AddPhysics("HadronPhysicsQGSP_BERT")
etc...
```

Conclusion

- modularization (removing compile-time dependencies) and introducing dynamically loadable plugins would improve a lot the flexibility of the Geant4 toolkit
  - would allow the users to select the needed minimum of the modules and to change them dynamically
  - would speed up the development
- Geant4 major release is certainly a good moment to introduce such a framework

Backup Slides

G4processes contribution (%)


hadronic/models/chiral_inv_phase_space
hadronic/models/neutron_hp
hadronic/models/im_r_matrix
hadronic/models/cascade
hadronic/models/de_excitation
hadronic/cross_sections
electromagnetic/lowenergy
hadronic/models/lend
electromagnetic/standard
hadronic/models/high_energy
hadronic/models/inclxx
electromagnetic/utils
hadronic/models/rpg
electromagnetic/dna/models
hadronic/models/parton_string
hadronic/models/low_energy
electromagnetic/dna/management
hadronic/stopping
hadronic/processes
hadronic/models/coherent_elastic
hadronic/models/quasi_elastic
electromagnetic/polarisation
hadronic/models/binary_cascade
electromagnetic/xrays
electromagnetic/adjoint
management
hadronic/models/radioactive_decay
hadronic/models/pre_equilibrium
electromagnetic/highenergy
hadronic/util
electromagnetic/dna/utils
hadronic/models/util
electromagnetic/dna/processes
electromagnetic/pii
electromagnetic/dna/molecules
hadronic/models/lll_fission
cuts
hadronic/management
electromagnetic/muons
hadronic/models/qmd
biasing
optical
parameterisation
transportation
hadronic/models/photolepton_hadron
scoring
decay
hadronic/models/management
hadronic/models/abrasion
hadronic/models/isotope_production
hadronic/models/theo_high_energy
hadronic/models/em_dissociation