Journal of Physics: Conference Series **513** (2014) 022030 doi:10.1088/1742-6596/513/2/022030

Recent Developments in the Geant4 Hadronic Framework

Witold Pokorski, Alberto Ribon

CERN, Geneva, Switzerland

E-mail: witold.pokorski@cern.ch

Abstract. In this paper we present the recent developments in the Geant4 hadronic framework. Geant4 is the main simulation toolkit used by the LHC experiments and therefore a lot of effort is put into improving the physics models in order for them to have more predictive power. As a consequence, the code complexity increases, which requires constant improvement and optimization on the programming side. At the same time, we would like to review and eventually reduce the complexity of the hadronic software framework. As an example, a factory design pattern has been applied in Geant4 to avoid duplications of objects, like cross sections, which can be used by several processes or physics models. This approach has been applied also for physics lists, to provide a flexible configuration mechanism at run-time, based on macro files. Moreover, these developments open the future possibility to build Geant4 with only a specified sub-set of physics models. Another technical development focused on the reproducibility of the simulation, i.e. the possibility to repeat an event once the random generator status at the beginning of the event is known. This is crucial for debugging rare situations that may occur after long simulations. Moreover, reproducibility in normal, sequential Geant4 simulation is an important prerequisite to verify the equivalence with multi-threaded Geant4 simulations.

1. Introduction

The Geant4 [1][2] simulation toolkit is undergoing constant development in order to improve the physics models leading to a more predictive power. This leads to a growing complexity of the code and therefore optimization and improvements on the programming side are necessary. In this paper we discuss a number of technical developments recently done. We start with the description of a new mechanism to share cross section objects between different parts of the application. We continue with the description of the generic physics list approach allowing the selection of the physics models at the initialization time. In the following section we discuss the reproducibility of events and the technical developments that it had implied. Finally we briefly discuss the multi-threading capabilities recently introduced in the code, as well as the use of fast mathematical functions.

2. Sharing hadronic cross-sections via factory pattern

To share cross-section objects between different 'users' (physics processes, models, physics lists, etc) we have introduced the factory pattern for the instantiation of the objects and we have extended the functionality of the G4CrossSectionDataSetRegistry class to store and provide the pointers to those objects. This mechanism, shown on Figure 1, works as follows:

- 'Cross-section user' asks G4CrossSectionsDataSetRegistry for a given G4CrossSectionDataSet by specifying its name (string).
- The registry checks if this cross section has been already instantiated.
- If yes, it returns the pointer to it (shared between all 'cross-section users').
- If not, the registry uses the factory to instantiate the given cross section. If the factory does not exist, it returns an error 'cross section not found'.

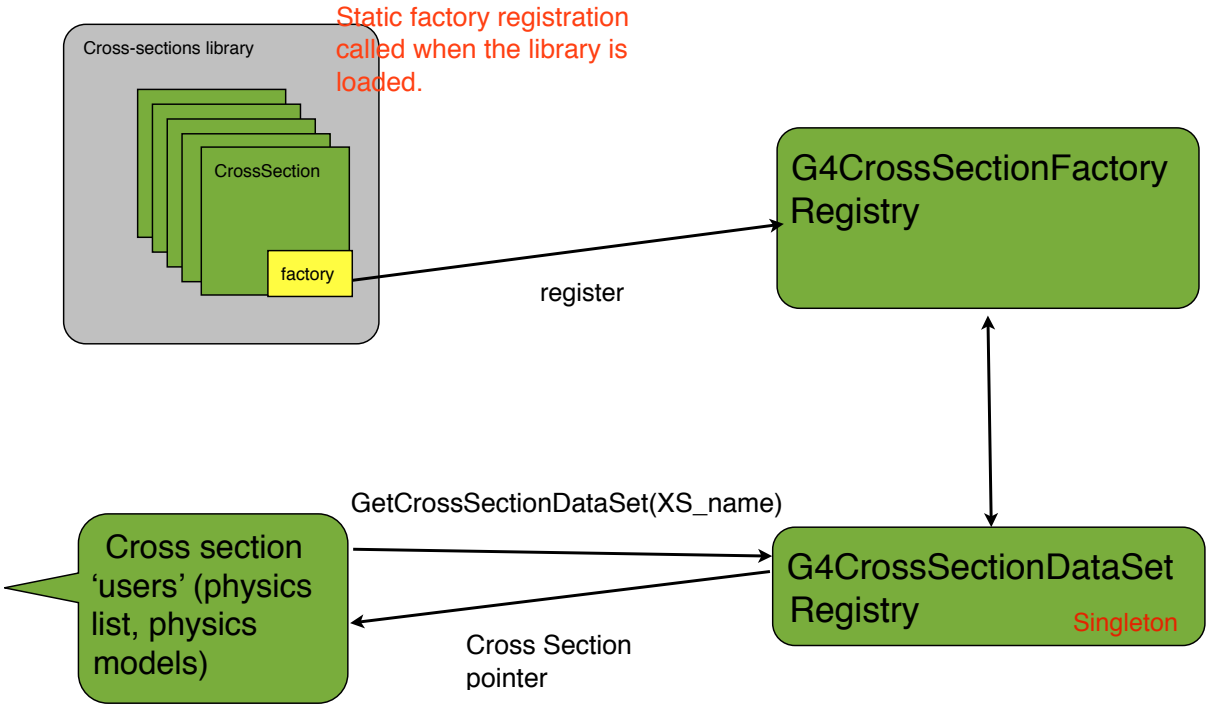

Figure 1 Use of factory pattern to instantiate cross section objects.

Currently, this mechanism is available for all the cross sections inheriting from the G4VCrossSectionDataSet class, however, its extension to the classes of the type G4VComponentCrossSection is ongoing. The only restriction for this sharing mechanism to work is that the cross sections objects need to be instantiated using the default constructor (without any arguments).

3. Run-time configuration of hadronic physics lists

The G4GenericPhysicsList class allows removing the compile- and link-time dependency between the user code and the specific physics models. In order to achieve this we have introduced the registry of physics "constructors" and we have instrumented them to provide factories that get registered in the registry.

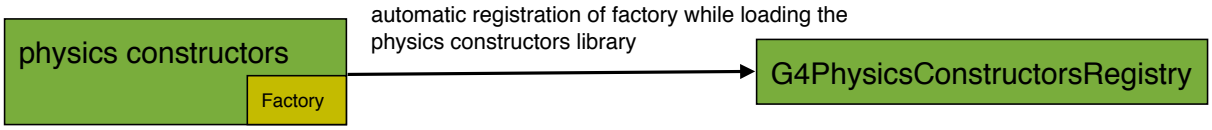

Figure 2 Physics constructors registry.

Physics lists can now be constructed in two new ways: through the G4GenericPhysicsList class using a macro file (see Figure 1) or by passing a vector of physics 'constructors' names at the instantiation time. For example for an equivalent of the FTFP_BERT physics list, this macro file would look like this:

```
/PhysicsList/defaultCutValue  0.7
/PhysicsList/SetVerboseLevel 1
```

```
/PhysicsList/RegisterPhysics G4EmStandardPhysics
/PhysicsList/RegisterPhysics G4EmExtraBertiniPhysics
/PhysicsList/RegisterPhysics G4DecayPhysics
/PhysicsList/RegisterPhysics G4HadronElasticPhysics
/PhysicsList/RegisterPhysics HadronPhysicsFTFP_BERT
/PhysicsList/RegisterPhysics G4BertiniAndFritiofStoppingPhysics
/PhysicsList/RegisterPhysics G4IonFTFPBinaryCascadePhysics
/PhysicsList/RegisterPhysics G4NeutronTrackingCut
```

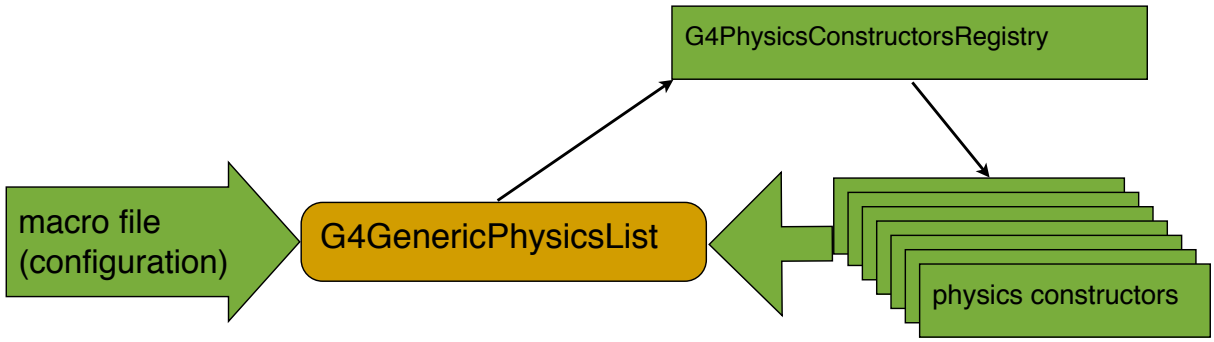

Figure 3 Generic physics list.

4. Reproducibility of events

Using pseudo-random number generators implies that events should be reproducible. Non-reproducibility of events makes it difficult to debug the code. Simulation results should be reproducible not only at the level of the run (starting from the same random-number generator seed), but also at the level of each event (starting from any event within a run). Sources of irreproducibility can be bugs (uninitialized variables), history-dependent approximations or incorrect caching. The last issue has been particularly present in a number of places in the Geant4 hadronic code. In some hadronic cross sections, a caching mechanism was used, where the value was calculated for the energy the cross section was called for and then it was reused for all energies within some (small) energy bin. This kind of caching was, of course, incorrect as it depended on the argument of the first call. The correct behaviour was to calculate and cache cross section values for the middle of each energy bin. In such a way the cross section values would not depend on the history of calls. A very similar problem was present for cross sections that were calculated for the first encountered isotope and then reused for all other isotopes of the given material. Again, the correct way of caching was to calculate the cross section for some average atomic number, not depending on the history of calls.

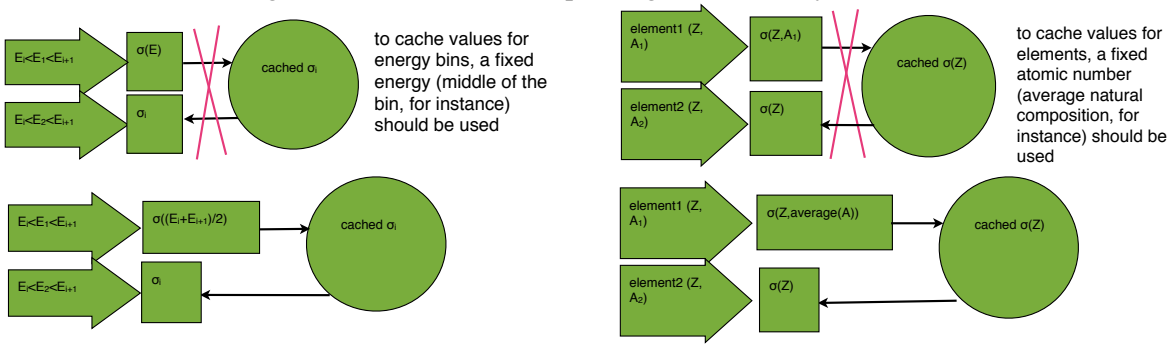

Figure 4 Caching mechanism - incorrect, history dependent, and correct based on mean values.

Several fixes have been done in the latest version of Geant4. All the history-dependent mechanisms have been replaced by new ones based on mean values. The Geant4 code is now fully reproducible, however, this problem requires constant attention to avoid newly introduced non-reproducibility.

5. Multithreading in Geant4 hadronic physics

The Geant4 code is undergoing a major development in order to run in the multi-threading mode. A number of technical issues needed to be addressed to eliminate any interference between several threads accessing the hadronic physics classes. Objects that can be easily shared are those that are read-only. In particular, caching becomes a delicate issue because of possible simultaneous write-access to the cache. A number of classes needed to be redesigned to assure that there are no conflicts when accessing the cache.

In order to validate the multi-threaded code we require that the calorimeter (and other) observables remain statistically the same between sequential and multi-threaded modes. On Figure 5 we can see an example of such a validation where simplified calorimeter response and lateral shower shape are compared between sequential and multi-threading modes.

The multi-threading increases significantly the event throughput, however one should remember that the reproducibility of events becomes the challenge. After a number of fixes and improvements, full reproducibility has been achieved, however, as in the sequential mode, this requires constant attention and monitoring of any new code.

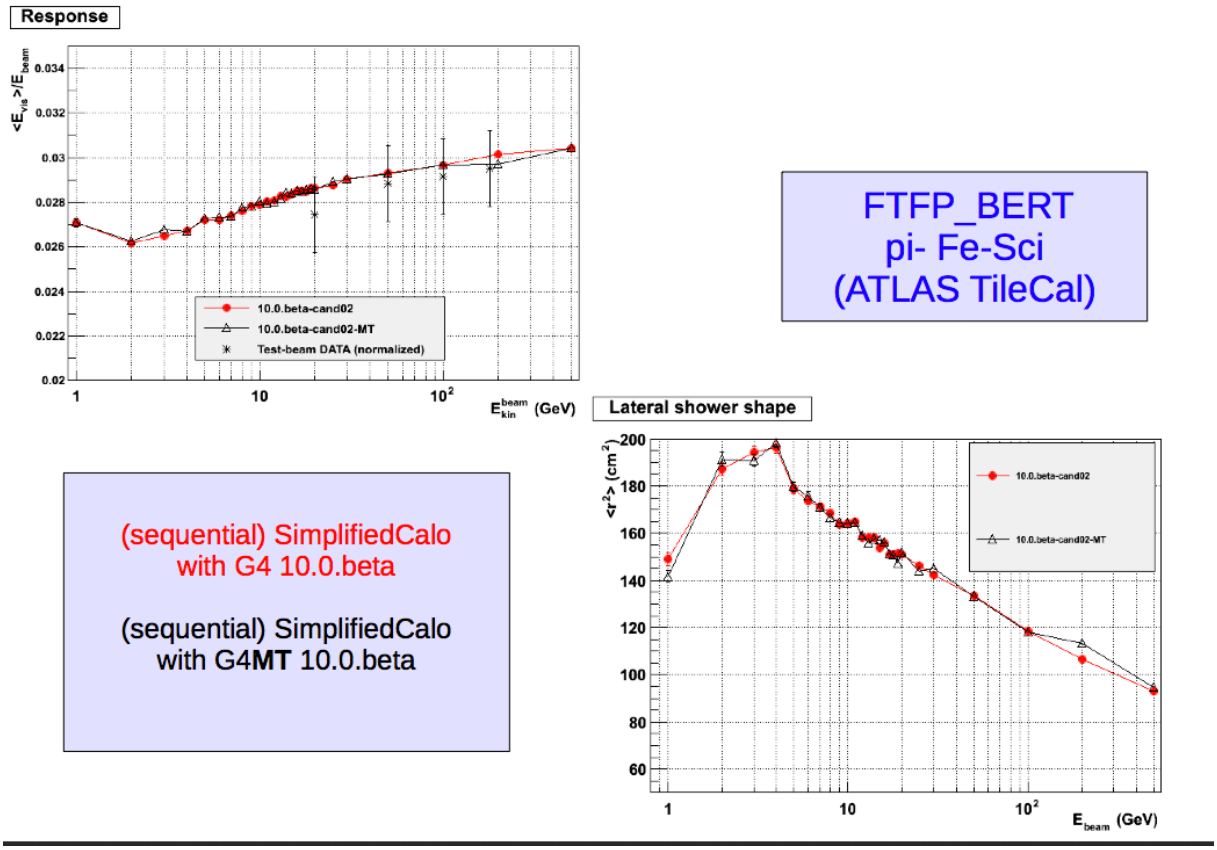

Figure 5 Comparison for sequential and multithreaded codes.

6. Use of fast mathematical functions

The precision of most hadronic cross sections is at the level of 5-10% and therefore there is no need to do high-precision calculations involving those cross sections. Using fast log and exp functions can increase significantly the CPU performance without any significant loss in the precision of the simulation results. We have replaced std::log and std::exp by faster implementations extracted from the VDT library [3][4]. The resulting loss in precision was negligible compared to the precision of the cross sections, while the cross sections calculation was 5% faster.

7. Conclusion

A number of new code developments in the Geant4 hadronic framework are part of the latest Geant4 release. Hadronic cross section objects can now be shared using the factory mechanism. The generic physics list concept allows putting physics models together dynamically during the initialization time. Several fixes have been also made in order to assure the reproducibility of events. This is now the case both for the sequential as well as multi-threading mode. The latter has been validated by running the simplified calorimeter and comparing the different observables. Finally, the code of the new Geant4 version has been made to run faster by using fast mathematical functions from the VDT library.

References

[1] Geant4 – a simulation toolkit, *Nuclear Instruments and Methods in Physics Research* A 506 (2003) 250-303
[2] Geant4 developments and applications, *IEEE Transactions on Nuclear Science* 53 No. 1 (2006) 270-278
[3] Piparo D, Innocente V and Hauth T, 2013, Speeding up HEP experiments' software with a library of fast and auto-vectorisable mathematical functions, submitted to proceedings of 20th International Conference on Computing in High Energy and Nuclear Physics (CHEP13), Amsterdam
[4] https://svnweb.cern.ch/trac/vdt