GEANT4 — a simulation toolkit

GEANT4 Collaboration

S. Agostinelli[ae], J. Allison[as,*], K. Amako[e], J. Apostolakis[a], H. Araujo[aj], P. Arce[ℓ,m,x,a], M. Asai[g,ai], D. Axen[i,t], S. Banerjee[bh,ℓ], G. Barrand[an], F. Behner[ℓ], L. Bellagamba[c], J. Boudreau[bc], L. Broglia[ar], A. Brunengo[c], S. Chauvie[bi,bk], J. Chuma[h], R. Chytracek[a], G. Cooperman[az], G. Cosmo[a], P. Degtyarenko[d], A. Dell'Acqua[a,i], G. Depaola[y], D. Dietrich[af], R. Enami[ab], A. Feliciello[bi], C. Ferguson[bg], H. Fesefeldt[ℓ,o], G. Folger[a], F. Foppiano[ac], A. Forti[as], S. Garelli[ac], S. Giani[a], R. Giannitrapani[bn], D. Gibin[m,bb], J.J. Gómez Cadenas[m,bo], I. González[q], G. Gracia Abril[n], G. Greeniaus[p,h,ag], W. Greiner[af], V. Grichine[f], A. Grossheim[m,z], P. Gumplinger[h], R. Hamatsu[bj], K. Hashimoto[ab], H. Hasui[ab], A. Heikkinen[ah], A. Howard[aj], A. Hutton[d], V. Ivanchenko[a,ba], A. Johnson[g], F.W. Jones[h], J. Kallenbach[aa], N. Kanaya[i,h], M. Kawabata[ab], Y. Kawabata[ab], M. Kawaguti[ab], S. Kelner[at], P. Kent[r], T. Kodama[aw], R. Kokoulin[at], M. Kossov[d], H. Kurashige[am], E. Lamanna[w], T. Lampen[ah], V. Lara[a,ℓ,bp], V. Lefebure[ℓ], F. Lei[bg,bd], M. Liendl[ℓ,a,bq], W. Lockman[j,bm], F. Longo[bℓ], S. Magni[k,au], M. Maire[ao], B. Mecking[d], E. Medernach[a], K. Minamimoto[aw,ai], P. Mora de Freitas[ap], Y. Morita[e], K. Murakami[am], M. Nagamatu[aw], R. Nartallo[b], P. Nieminen[b], T. Nishimura[ab], K. Ohtsubo[ab], M. Okamura[ab], S. O'Neale[s], Y. Oohata[bj], K. Paech[af], J. Perl[g], A. Pfeiffer[a], M.G. Pia[ad], F. Ranjard[n], A. Rybin[ak], S. Sadilov[a,ak], E. Di Salvo[c], G. Santin[bℓ], T. Sasaki[e], N. Savvas[as], Y. Sawada[ab], S. Scherer[af], S. Sei[aw], V. Sirotenko[i,aℓ], D. Smith[g], N. Starkov[f], H. Stoecker[af], J. Sulkimo[ah], M. Takahata[ay], S. Tanaka[bf], E. Tcherniaev[a], E. Safai Tehrani[g], M. Tropeano[ae], P. Truscott[bd], H. Uno[aw], L. Urban[v], P. Urban[aq], M. Verderi[ap], A. Walkden[as], W. Wander[av], H. Weber[af], J.P. Wellisch[a,ℓ],

T. Wenaus[u], D.C. Williams[j,be], D. Wright[g,h], T. Yamada[aw], H. Yoshida[aw], D. Zschiesche[af]

[a]*European Organization for Nuclear Research (CERN)*

[b]*European Space Agency (ESA), ESTEC, Holland*

[c]*Istituto Nazionale di Fisica Nucleare (INFN), Italy*

[d]*Jefferson Lab*

[e]*KEK, Japan*

[f]*Lebedev Institute*

[g]*Stanford Linear Accelerator Center (SLAC)*

[h]*TRIUMF, Canada*

[i]*ATLAS Collaboration*

[j]*BaBar Collaboration*

[k]*Borexino Collaboration*

[ℓ]*CMS Collaboration*

[m]*HARP Collaboration*

[n]*LHCb Collaboration*

[o]*RWTH Aachen*

[p]*University of Alberta*

[q]*ALICE Collaboration*

[r]*University of Bath*

[s]*University of Birmingham*

[t]*University of British Columbia*

[u]*Brookhaven National Laboratory*

[v]*Kfki, Budapest*

[w]*Università della Calabria and INFN*

[x]*CIEMAT*

[y]*University of Cordoba*

[z]*University of Dortmund*

[aa]*FNAL*

[ab]*Fukui University*

[ac]*IST Natl. Inst. for Cancer Research of Genova*

[ad]*INFN Genova*

[ae]*Università di Genova*

[af]*Inst. für Theoretische Physik, Johann Wolfgang Goethe Universität, Frankfurt*

[ag]*HERMES Collaboration*

[ah]*Helsinki Institute of Physics (HIP)*

[ai]*Hiroshima Institute of Technology*
[aj]*Imperial College of Science, Technology and Medicine, London*
[ak]*IHEP Protvino*
[aℓ]*North Illinois University*
[am]*University of Kyoto*
[an]*LAL, Orsay*
[ao]*IN2P3/LAPP, Annecy*
[ap]*LLR/IN2P3, Palaiseau*
[aq]*EPFL, Lausanne*
[ar]*Patron of Lyon University*
[as]*University of Manchester*
[at]*Mephi, Moscow*
[au]*INFN Milan*
[av]*MIT*
[aw]*Naruto University of Education*
[ay]*Niigata University*
[az]*Northeastern University*
[ba]*Budker Institute for Nuclear Physics, Novosibirsk*
[bb]*Università di Padova*
[bc]*University of Pittsburg*
[bd]*QinetiQ, UK*
[be]*SCIPP/UCSC, Santa Cruz*
[bf]*Ritsumeikan University*
[bg]*University of Southampton*
[bh]*TIFR, Mumbai, India*
[bi]*INFN Torino*
[bj]*Tokyo Metropolitan University*
[bk]*Università di Torino*
[bℓ]*Università di Trieste and INFN Trieste*
[bm]*UCSC, Santa Cruz*
[bn]*Università di Udine and INFN Udine*
[bo]*University of Valencia*
[bp]*IFIC Instituto de Fisica Corpuscular de Valencia*
[bq]*Vienna University of Technology*

Abstract

Geant4 is a toolkit for simulating the passage of particles through matter. It includes a complete range of functionality including tracking, geometry, physics models and hits. The physics processes offered cover a comprehensive range, including electromagnetic, hadronic and optical processes, a large set of long-lived particles, materials and elements, over a wide energy range starting, in some cases, from 250 eV and extending in others to the TeV energy range. It has been designed and constructed to expose the physics models utilised, to handle complex geometries, and to enable its easy adaptation for optimal use in different sets of applications. The toolkit is the result of a worldwide collaboration of physicists and software engineers. It has been created exploiting software engineering and object-oriented technology and implemented in the C++ programming language. It has been used in applications in particle physics, nuclear physics, accelerator design, space engineering and medical physics.

Key words: Simulation, Particle interactions, Geometrical modelling, Software engineering, Object-oriented technology, Distributed software development
PACS: 07.05.Tp, 13, 23

1 Introduction

Modern particle and nuclear physics experiments pose enormous challenges in the creation of complex yet robust software frameworks and applications. Of particular importance is the ever-increasing demand for large-scale, accurate and comprehensive simulations of the particle detectors used in these experiments. The demand is driven by the escalating size, complexity, and sensitivity of the detectors and fueled by the availability of moderate-cost, high-capacity computer systems on which larger and more complex simulations become possible. Similar considerations arise in other disciplines, such as: radiation physics, space science, nuclear medicine and, in fact, any area where particle interactions in matter play a role.

In response to this, a new object-oriented simulation toolkit, Geant4, has been developed. The toolkit provides a diverse, wide-ranging, yet cohesive set of software components which can be employed in a variety of settings. These range from simple one-off studies of basic phenomena and geometries to full-scale detector simulations for experiments at the Large Hadron Collider and other facilities.

* Corresponding Author. Address: Department of Physics and Astronomy, The University of Manchester, MANCHESTER M13 9PL, UK. Telephone: +44-161-275-4179. Fax: +44-161-273-5867. E-mail: John.Allison@man.ac.uk

In defining and implementing the software components, all aspects of the simulation process have been included: the geometry of the system, the materials involved, the fundamental particles of interest, the generation of primary particles of events, the tracking of particles through materials and external electromagnetic fields, the physics processes governing particle interactions, the response of sensitive detector components, the generation of event data, the storage of events and tracks, the visualisation of the detector and particle trajectories, and the capture for subsequent analysis of simulation data at different levels of detail and refinement.

Early in the design phase of the project, it was recognised that while many users would incorporate the GEANT4 tools within their own computational framework, others would want the capability of easily constructing standalone applications which carry them from the initial problem definition right through to the production of results and graphics for publication. To this end, the toolkit includes built-in steering routines and command interpreters which operate at the problem setup, run, event, particle transportation, visualisation, and analysis levels, allowing all parts of the toolkit to work in concert.

At the heart of this software system is an abundant set of physics models to handle the interactions of particles with matter across a very wide energy range. Data and expertise have been drawn from many sources around the world and in this respect GEANT4 acts as a repository that incorporates a large part of all that is known about particle interactions; moreover it continues to be refined, expanded and developed. A serious limitation of many previous simulation systems was the difficulty of adding new or variant physics models; development became difficult due to the increasing size, complexity and interdependency of the procedure-based code. In contrast, object-oriented methods have allowed us effectively to manage complexity and limit dependencies by defining a uniform interface and common organisational principles for all physics models. Within this framework, the functionality of models can be more easily seen and understood, and the creation and addition of new models is a well-defined procedure that entails little or no modification to the existing code.

GEANT4 was designed and developed by an international collaboration, formed by individuals from a number of cooperating institutes, HEP experiments, and universities. It builds on the accumulated experience of many contributors to the field of Monte Carlo simulation of physics detectors and physical processes. While geographically-distributed software development and large-scale object-oriented systems are no longer a novelty, we consider that the GEANT4 Collaboration, in terms of the size and scope of the code and the number of contributors, represents one of the largest and most ambitious projects of this kind. It has demonstrated that rigorous software engineering practices and object-oriented methods can be profitably applied to the production of a coherent and maintainable software product, even with the fast-changing and open-ended requirements presented by physics research.

In the following sections we present a detailed overview of GEANT4 and its features and capabilities, including the design and implementation of the various categories of physics models. Many new physics models have been developed, and others have been refined or extended. They have been created to support a growing range of applications for the software, including particle, nuclear, medical, accelerator and space physics. The code and documentation, as well as tutorials and examples, are available from our Web site [1].

1.1 History of GEANT4

The origin of GEANT4 development can be traced back to two studies done independently at CERN and KEK in 1993 [2]. Both groups sought to investigate how modern computing techniques could be applied to improve what was offered by the existing GEANT3 program [3]. These two activities merged and a proposal was submitted to the CERN Detector Research and Development Committee (DRDC) [4] to construct a simulation program based on object-oriented technology. The resulting project was RD44, a world-wide collaboration that grew to include the efforts of 100 scientists and engineers, drawn from more than 10 experiments in Europe, Russia, Japan, Canada and the United States.

The design choices faced by RD44 and the decisions arrived at are described in later chapters, but key to its success was a careful design adapting object-oriented methodology and an early decision to use the practical C++ language.

The R & D phase was completed in December 1998 [5] with the delivery of the first production release. Subsequently the GEANT4 Collaboration was established in January 1999 to continue the development and refinement of the toolkit, and to provide maintenance and user support.

1.2 Organisation of the Collaboration

A Memorandum of Understanding (MoU) [6] signed by all participating parties governs the formal collaboration. It is subject to tacit renewal every two years and sets out a collaboration structure composed of a Collaboration Board (CB), a Technical Steering Board (TSB) and several working groups. The MoU also defines the way in which collaboration resources — money, manpower, expertise, and key roles and activities (such as program librarian and documentation manager) — are measured in Contribution Units (CU), and it further delineates how the boards are constituted depending on the CU count for each signatory. Participating groups include experimental teams and collaborations, laboratories and national institutes.

It is the CB's mandate to manage these resources and to monitor the agreed responsibilities among the affiliates. This body is also charged with the evolution of the MoU. The TSB, on the other hand, is the forum where technical matters, like software engineering details and physics model implementation issues, are discussed and decided and where priorities are given to user requests. Its primary tasks are the supervision of the production service and the user support and the overseeing of ongoing further development of the program. The TSB is chaired by the spokesperson of the Collaboration, who is appointed by and reports to the CB. The spokesperson is (re)elected every two years.

Every domain of the Geant4 software that corresponds to a releasable component (library) is individually managed by a working group of experts. In addition, there is a working group for each of the activities of testing and quality assurance, software management and documentation management. A coordinator who is selected by the TSB heads each group. There is also an overall release coordinator. This clean overall problem decomposition makes the distributed software design and development possible in a worldwide collaboration. Every group can work in parallel, allowing an optimal use of manpower and expertise.

1.3 User support, documentation and source code

The Collaboration provides documentation and user support for the toolkit. The support model is described in more detail in Section 3.6.

Documentation [7] includes installation, user and reference guides, and a range of training kits (see also Section 1.4). It is intended to cover the need of the beginner through to the expert user who wishes to expand the capabilities of Geant4.

User support covers help with problems relating to the code, consultation on using the toolkit and responding to enhancement requests. A user may also expect assistance in investigating anomalous results.

A Web-based reporting system and a list of frequently asked questions (FAQs) are available on the Geant4 Web site [1]. The Collaboration also runs a Web-based user forum [8], with sub-forums according to areas of different interest.

Regular releases of the source code and documentation are freely available on the Web.

1.4 Examples and training kits

The toolkit includes examples at three levels:

- *Novice*: for understanding basic functionalities;
- *Extended*: focused on specific domains of application (they may also need additional third party libraries);
- *Advanced*: full programs created to utilise Geant4 in HEP experiments, and for space and medical applications.

They are intended to develop the user's understanding in many areas. Initial emphasis is on the classes describing the user's setup, which are required by the toolkit. These classes are explained in Section 2.4.

GEANT4 also provides a training kit. It consists of a modular set of units, each covering a specific domain. The units contain descriptive material and examples, such as code excerpts, or plots with performance results. They are modular in themselves, providing different levels of coverage and complexity.

1.5 Structure of this paper

For the reader who wishes to obtain a broad overall view of the project from inception to realisation we describe basic principles of the design in Section 2.

Details that are needed to understand how to extend the toolkit, tailor it for special use and obtain optimal performance, are postponed to Sections 4 and 5.

In between, we devote Section 3 to the important issue of the software process as it applies to a large, dispersed collaboration. It is here that the exploitation of modern software engineering techniques and object-oriented methods are discussed.

The basic algorithms and capabilities of the kernel are described in Section 4 and an overview of available physics processes and models is presented in Section 5. The latter also includes a sample of results and comparisons with GEANT3 and experimental measurements.

Additional capabilities are discussed in Section 6 and interactivity (user interfaces, visualisation and analysis) in Section 7.

2 Design overview

2.1 General considerations

GEANT4 is driven by the software needs of modern experiments. A typical software system contains components — event generator, detector simulation, reconstruction and analysis — that can be used separately or in combinations. The toolkit has been built as the basis for the simulation component. Thus it was required

- to have well-defined interfaces to other components, and
- to provide parts to be used by the other components.

Other design requirements are that it is modular and flexible, and that its implementation of physics is transparent and open to user validation. It should allow the user to understand, customise and extend it in all domains. Its modular architecture should enable the user to pick only those components he/she needs.

The high-level design was based on an analysis of the initial user requirements [9]. This ultimately led to a modular and hierarchical structure for the toolkit (see Figure 1), where sub-domains are linked by a uni-directional flow of dependencies.

The key domains of the simulation of the passage of particles through matter are:

- geometry and materials;
- particle interaction in matter;
- tracking management;
- digitisation and hit management;
- event and track management;
- visualisation and visualisation framework;
- user interface.

These domains naturally led to the creation of class categories with coherent interfaces and, for each category, a corresponding working group with a well defined responsibility. It also led to the concept of a "toolkit", which implies that a user may assemble his/her program at compile time from components chosen from the toolkit or self-supplied.

GEANT4 exploits advanced software engineering techniques to deliver these key requirements of functionality, modularity, extensibility and openness. The techniques used for the Architectural Design were based on the Booch Methodology and followed an iterative approach with progressive refinement of user requirements, architectural and detailed design. These issues are discussed in more detail in Section 3.

2.1.1 General capabilities and properties

The toolkit offers the user the ability to create a geometrical model with a (possibly) large number of components of different shapes and materials, and to define “sensitive” elements that record information (hits) needed to simulate detector responses (digitisation).

The primary particles of the events can be derived from internal and external sources.

GEANT4 provides a comprehensive set of physics processes to model the behaviour of particles. The user is able to choose from different approaches and implementations, and to modify or add to the set provided.

In addition the user can interact with the toolkit through a choice of (graphical) user interfaces and visualise the geometry and tracks with a variety of graphics systems through a well-defined interface and is given the ability to implement this interface over other systems of his/her choice.

In general, the classes in the toolkit are designed in a highly reusable and a compact way so that the user can extend or modify their services for his/her specific applications. The user can realise this by following the discipline of object-oriented technology.

Maximum use has been made of the experience acquired from previous simulation packages, in particular GEANT3.

As computing performance is a crucial issue for a detector simulator, the goal was to demonstrate performance comparable to GEANT3 or better.

2.1.2 Openness

One of the most important design goals was to make the design and implementation of the physics open and transparent. Exploiting object-oriented technology has enabled us to establish a clear and customisable correspondence between particles and processes and offer a choice of models for each process. The result is a highly granular implementation, each component of which can be inspected at source code level.

The way cross sections are calculated — via formulas, parameterisations or interpolation of databases — is exposed. In the last case the information extracted from the database is separated from the way it is accessed and used, giving the opportunity of using different databases and allowing their applicability to be tailored by particle, energy, material, etc.

Similarly the generation of the final state is separated from the calculation of the cross-sections used for tracking and is also split into alternative or complementary models, according to the energy, range, particle type and material.

2.2 Global structure

The design has evolved during development. It currently includes 17 major categories, identified from an analysis driven by our User Requirements. Figure 1 shows the top level categories and illustrates how each category depends on the others. There is a uni-directional flow of dependencies, i.e., no circular dependencies, as required.

Categories at the bottom of the diagram are used by virtually all higher categories and provide the foundation of the toolkit. These include the category *global* covering the system of units[1], constants, numerics and random number handling; *materials*; *particles*; *graphical representations*; *geometry* including the volumes for detector description and the navigation in the geometry model; and *intercoms* which provides both a means of interacting with Geant4 through the user interface and also a way of communicating between modules that should not otherwise depend on one another. *Intercoms* is also the repository of abstract interfaces for "plug-ins", namely Fast Simulation (Section 6.1) and Visualisation (Section 7.2).

Above these reside categories required to describe the tracking of particles and the physical processes they undergo. The *track* category contains classes for tracks and steps, used by *processes* which contains implementations of models of physical interactions. Additionally, one such process, *transportation*, handles the transport of particles in the geometry model and, optionally, allows the triggering of parameterisations of processes (see Section 6.1). All these processes may be invoked by the *tracking* category, which manages their contribution to the evolution of a track's state and undertakes to provide information in sensitive volumes for hits and digitisation.

Over these the *event* category manages events in terms of their tracks and *run* manages collections of events that share a common beam and detector

[1] A key design feature is independence from the internal representation of quantities. The internal representation can be chosen at compile time to suit the application. This feature is provided by CLHEP's Units package [10].

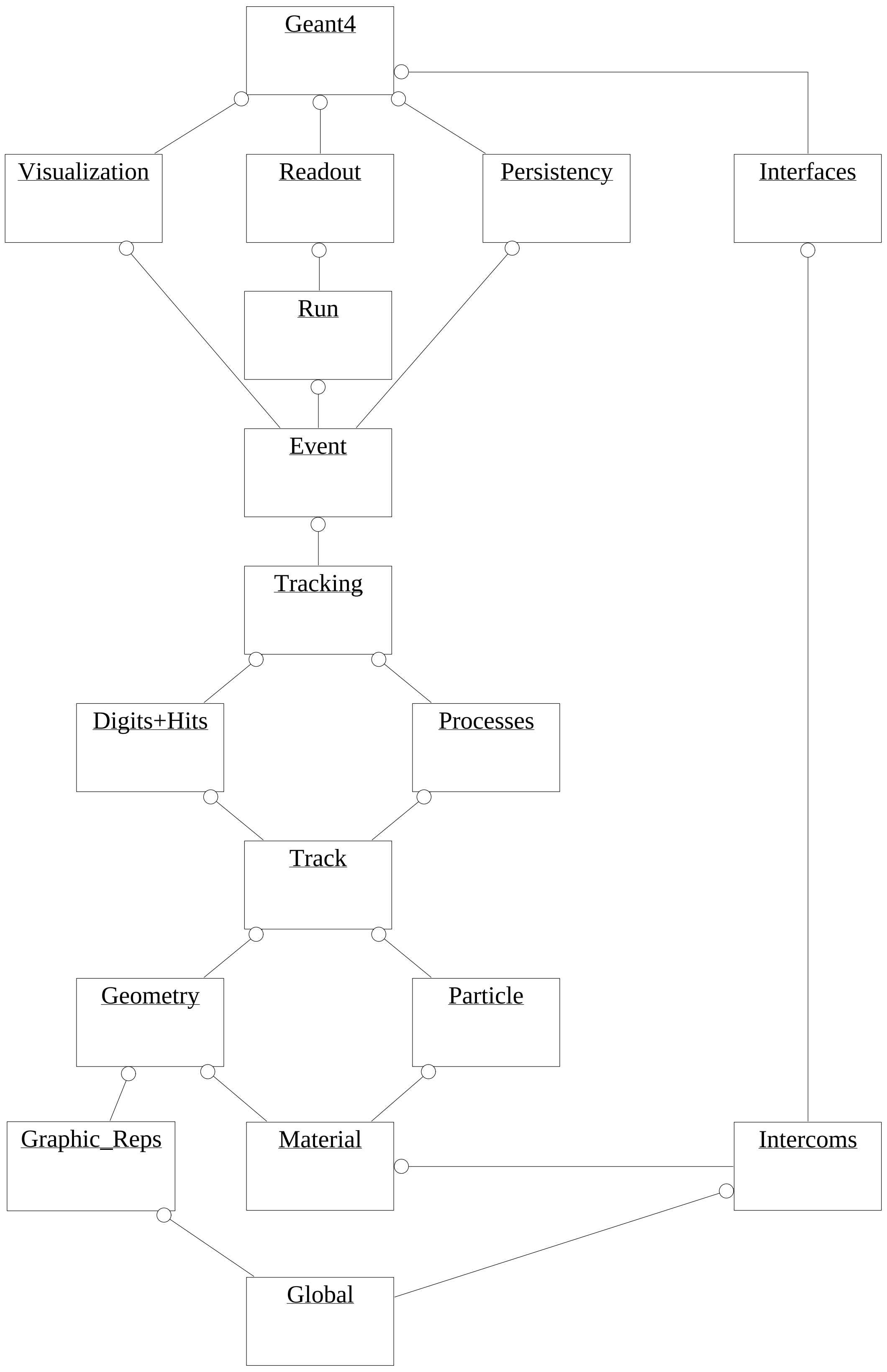


Fig. 1. The Top Level Category Diagram of the GEANT4 toolkit. The open circle on the joining lines represents a using relationship; the category at the circle end uses the adjoined category.

implementation. A *readout* category allows the handling of "pile-up".

Finally capabilities that use all of the above and connect to facilities outside the toolkit (through abstract interfaces) are provided by the *visualisation*, *persistency* and [user] *interface* categories.

In the following sections, some important aspects of the architectural design are covered. A study of these is essential for understanding the structure and the behaviour of the toolkit.

2.3 Design and architecture

2.3.1 Events

The *event* category provides an abstract interface to external physics event generators for the generation of the primary particles which define a physics event. Primary vertices and primary particles are represented by special classes which are free from any other. Through these special classes, the user can interface to the physics generators by preparing his/her own conversion codes. (The General Particle Source Module [11] can simplify this task by allowing a source with arbitrary energy, spatial and angular distribution to be defined at run time.) This isolation allows a GEANT4-based simulation program not to rely on specific choices for physics generators and also to be independent of the specific persistency solution adopted for storing the "simulation truth". Moreover, the primary particle can represent any kind of particle, even one that cannot be treated by GEANT4, such as a quark or a gauge boson. It keeps the mother-daughter relations between primary particles so that the specific decay chain can be imported from the physics generator. For example, the user can specify the decay products of each of two B-mesons separately.

The class `G4Event` represents an event, which is the main unit of simulation. This class avoids keeping any transient information which is not meaningful after the processing of an event is complete. Thus it is objects of this class that the user can store for processing further down the program chain, such as reconstruction. It contains primary vertices and primary particles before processing the event. After processing, it has hits and digitisations generated by the simulation and, optionally, trajectories of the simulated particles for the recording of "simulation truth". For performance reasons, `G4Event` and its content classes are not persistent. Instead, the user is assumed to provide his/her own conversion code between them and corresponding persistent classes [12] (see also Section 6.3).

The fact that `G4Event` is independent of other classes also benefits pile-up simulation. Digitisation can be postponed until after the processing of two or more events on a rolling basis and `G4Event` objects can be "added" to each other, making use of information about primary timing, so that detector output signals can be generated as the consequence of signal overlapping.

2.3.2 Geometry and detector representation

The *geometry* module (category) offers the ability to describe a geometrical structure and propagate particles efficiently through it. Some concepts have been borrowed from previous simulation packages but improvements, refinements and advances have been made in some key areas to cope with the greater number and different organisation of detector volumes now experienced. In particular, the requirement to exchange detector geometry with CAD systems — via the ISO STEP standard [13] — and navigate efficiently in such geometries led to a new optimisation technique.

The concepts of *logical* and *physical* volume are not unlike those of GEANT-3. A logical volume represents a detector element of a certain shape that can hold other volumes inside it and can have other attributes; it also has access to other information that is independent of its physical position in the detector, such as material and sensitive detector behaviour. A physical volume represents the spatial positioning or placement of the logical volume with respect to an enclosing mother (logical) volume. Thus a hierarchical tree structure of volumes can be built, each volume containing smaller volumes (which may not overlap). Repetitive structures can be represented by specialised physical volumes — replicas and parameterised placements — with sometimes enormous saving of memory.

In GEANT4 the logical volume has been refined by defining the shape as a separate entity, named *solid*. Solids with simple shapes, like rectilinear boxes, trapezoids, spherical and cylindrical sections or shells, each have their properties coded separately, in accord with the concept of *Constructive Solid Geometry (CSG)*. More complex solids are defined by their bounding surfaces, which can be planes, second order surfaces or higher order B-spline surfaces [14], and belong to the *Boundary Representations (BREPs)* sub-category. This variety matches those described by the ISO STEP standard for CAD systems.

Another way of obtaining solids is by boolean combination — union, intersection and subtraction. The solids should be either CSGs or other boolean solids (the product of a previous boolean operation). One of the components may have an optional transformation relative to the other. Some actual shapes lend themselves to this approach and their navigation is efficient.

Although a detector is naturally and best described by a hierarchy of volumes, efficiency is not critically dependent on this. An optimisation technique, called voxelisation, described in Section 4.4.1, allows efficient navigation even in “flat” geometries, typical of those produced by CAD systems.

2.3.3 Tracking

It is well known that the overall performance of detector simulation depends critically on the CPU time spent moving the particle by one step. This is a key consideration in the object design of the *tracking* category.

A consequence of this is that in Geant4 particles are *transported*, instead of being considered *self moving*. However, this is done by the *transportation* process, described later in Section 4.2. The *tracking* category simply steers the invocation of processes.

Contrary to GEANT3, Geant4 treats physics processes in a very generic way [15]. Geant4 tracking does not depend on the particle type nor on the specific physics process, including particle transportation.

In Geant4, each particle is moved step by step with a tolerance that permits significant optimising of execution performance but that preserves the required tracking precision. All physics processes associated with the particle propose a *step*. For a particle at rest this is a *time* rather than a *length*. The smallest of the following is chosen:

- the *maximum allowed step* stipulated by the user (through the `SetMax-AllowedStep` method in the `G4UserLimits` class);
- the *steps* proposed by the actions of all attached processes, including that imposed by the geometrical limit as proposed by the transportation process.

Depending on its nature, a physics process possesses one or more characteristics represented by the following actions handled by the tracking:

(1) *at rest*, for particles at rest (e.g., decay at rest);
(2) *along step*, which implements behaviour such as energy loss or secondary particle production that happen “continuously” along a step (e.g., Čerenkov radiation);
(3) *post step*, which is invoked at the end of the step (e.g., secondary particle production by a decay or interaction).

Along step actions take place cumulatively, while the others are exclusive. The tracking handles each type of action in turn. For these three actions, each physics process has a `GetPhysicalInteractionLength`, which proposes a *step*, and a `DoIt` method that carries out the action. A process can stipulate that its action must always be done (multiple scattering and transportation are examples). The tracking scans all physics processes and actions for the given particle, and decides which one is to be invoked.

More details of the class structure are given in Section 4.3.

The physical values associated with each step are exchanged between the tracking and each physics process using objects of the `G4Step` class.

2.3.4 Physics

The three types of action described above and the corresponding virtual methods are defined in the base class `G4VProcess` (see Section 4.3). All physics processes conform to this basic interface. However, different approaches for the detailed design of the subdomains have been developed; for hadronic processes, the abundance and complexity has required an additional decomposition described in outline in the section on hadronic physics below.

Particle Decay

The step length (or life time for the *at rest* action) is straightforwardly calculated from the mean life of the particle. The generation of decay products is more difficult, requiring a knowledge of branching ratios and, for 3 or more body decays, theory or parameter or data driven distributions. The issues are discussed in Section 5.5.

Electromagnetic Physics

Geant4 electromagnetic physics manages the electromagnetic interactions of leptons, photons, hadrons and ions.

The electromagnetic package is organised as a set of class categories:

- *standard*: handling basic processes for electron, positron, photon and hadron interactions;
- *low energy*: providing alternative models extended down to lower energies than the standard category;
- *muons*: handling muon interactions;
- *X-rays*: providing specific code for X-ray physics;
- *optical*: providing specific code for optical photons;
- *utils*: collecting utility classes used by the other categories.

It provides the features of openness and extensibility resulting from the use of object-oriented technology; alternative physics models, obeying the same process abstract interface, are often available for a given type of interaction; an example of such case is shown in Figure 2.

Public evaluated databases are used in electromagnetic processes without introducing any external dependence, while keeping the physics open to future

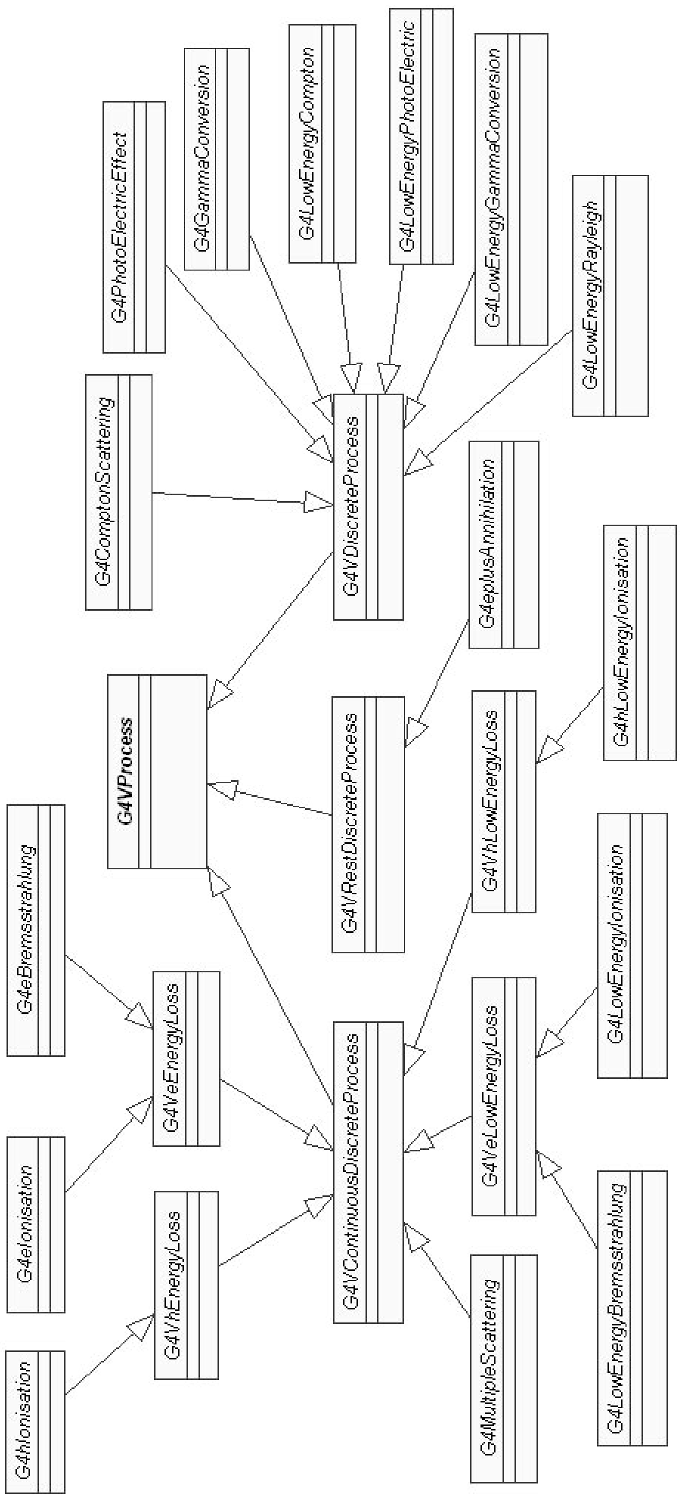


Fig. 2. A class diagram of electromagnetic processes, showing how alternative processes, obeying the same abstract interface, are provided.

evolutions of available data sets. This feature also contributes to the reliability and the openness of the physics implementation.

The package includes the processes of ionisation, bremsstrahlung, multiple scattering, Compton and Rayleigh scattering, photo-electric effect, pair conversion, annihilation, synchrotron and transition radiation, scintillation, refraction, reflection, absorption and Čerenkov effect.

In the *standard* electromagnetic processes category, the class `G4eIonisation` calculates for electrons and positrons the energy loss contribution due to ionisation and simulates the "discrete" part of the ionisation, namely the Moller and Bhabha scattering and δ-ray production. For each material and for e^+ and e^-, it produces an energy loss, range and inverse range table. The class `G4eBremsstrahlung` computes the energy loss contribution due to soft bremsstrahlung and simulates the "discrete" or hard bremsstrahlung. These two physics processes are closely connected by the design adopted. For the electromagnetic processes of hadrons, the `G4hIonisation` class computes the continuous energy loss and simulates δ-ray production. In this case, energy loss, range and inverse range tables are constructed only for proton and anti-proton; the energy loss for other charged hadrons are computed from these tables at the *scaled* kinetic energy (see Section 5.6.3). The energy loss also depends on the *cut in range*, which is described in more detail in Section 5.6.2.

The *low energy* electromagnetic processes category adopts a more complex design approach, by distinguishing the concepts of "physics process" and "model". A physics process may aggregate various components, each one being represented by a model; models can play complementary or alternative roles. A *strategy* [16] design pattern is adopted to define the family of physics models, encapsulate them and make them interchangeable. Thanks to this design, the system is open to evolution, without requiring any internal modification if new or alternative features are introduced. An example is, for instance, in the low energy hadron ionisation process (`G4hLowEnergyIonisation`) where a strategy pattern handles the complementary models of energy loss — Bethe-Bloch, parameterisation, free electron gas, quantum harmonic oscillator — depending on the energy range and charge of the incident particle. Other strategy patterns handle the models for electronic and nuclear stopping power respectively, while energy loss fluctuation models are treated separately. The development of an additional stopping power parameterisation, based on new data, is straightforward; the new algorithm would just be required to satisfy the common abstract interface and to be registered in the list of available parameterisations.

The *muons* category is modelled on the *standard* category. The energy loss of muons is computed by the class `G4MuEnergyLoss` using a scheme of computation which is the same as in the case of e^+/e^-. The `G4MuIonisation` class computes the contribution to the continuous energy loss due to ionisation and simulates the corresponding "discrete" process, knock-on electron or δ-ray production. The `G4MuBremsstrahlung` class calculates the continuous loss due to *soft* bremsstrahlung and simulates "discrete", *hard* bremsstrahlung. The `G4MuPairProduction` class gives the contribution to the continuous energy loss due to soft e^+/e^- pairs and performs the simulation of pair production.

The features of energy loss are very similar for e^+/e^-, μ^+/μ^- and charged hadrons so, by design, a common description for them has been adopted. The continuous energy loss is calculated as a sum of the contributions of the different processes. It also proposes a step that, by the mechanism of chosing the smallest step described above, limits the step of all processes in order to preserve precision in a situation where the energy is changing along the step; for example, the stopping range may be required to decrease by not more than some fraction of the total ionisation range, if this limit is not less than some *finalRange* parameter. It is worth mentioning that the lower limit used here is more natural and physical than the one used in GEANT3's automatic calculation of the tracking parameters.

Hadronic Physics

Given the vast number of possible modelling approaches, we have chosen to design an additional set of implementation frameworks to help generate the corresponding code in a distributed manner, and allow significant flexibility to the final user. Figure 3 illustrates the various framework levels in an annotated package dependency diagram.

The so called "Russian dolls" approach for the implementation framework design was followed. In this approach, an abstract top-level framework provides the basic interface to other GEANT4 categories. It satisfies the most general use-case for hadronic shower simulation, namely to provide inclusive cross-sections and final state generation. The framework is then refined for more specific use-cases by implementing a hierarchy of frameworks. Each sub-level implements the interface specification of the ancestor framework level. It adds implementation for the common logic of a particular use-case package, like the information flow between parton string models and codes simulating de-excitation of nuclear matter into hadrons, and provides the abstract interfaces for the associated use-case package. By so doing, the granularity of abstraction and delegation is refined at each framework level. The delegation mechanism is implemented through abstract classes[2].

[2] The same can be achieved with template specialisations with slightly improved CPU performance but at the cost of significantly more complex designs and, with present compilers, reduced portability.

Fig. 3. Package diagram of implementation frameworks and example implementations available for the hadronic physics category.

To illustrate this, in the following we present the second framework level in some detail. (For a complete descriptions of all levels, please see [17].) This framework level is very relevant for GEANT4, and defines at the same time some of the most relevant abstractions. It concerns processes that occur for particles in flight. For these cases, one soon finds that the sources of cross-sections and final state production are rarely the same. Moreover, different sources will come with different restrictions. The most important use-cases of the framework address these issues. A user might want to combine different cross-sections and final state or isotope production models provided by GEANT4, and a physicist might want to implement a model for a particular situation and add, in a seamless manner, cross-section data sets that are relevant for a particular analysis. The requirements on this framework level are the following:

- The ability to add user defined data sets, final state and isotope production models in a seamless manner.
- The ability to use different data sets, different isotope production and final state production codes for different parts of the simulation, depending on the conditions at the point of interaction.
- A flexible choice of inclusive scattering cross-sections, final state production models and isotope production models, to run in parasitic mode to any kind of transport model.

These requirements are implemented in three framework components, one each for cross-sections, final state production, and isotope production. These three parts are integrated in the `G4HadronicProcess` class, which serves as base-class for all hadronic processes of particles in flight. Each process holds a list of “cross section data sets”. The term “data set” represents an object which encapsulates methods and data for calculating total cross sections for a given process in a certain range of validity. The implementations may take any form: it can be a simple equation or a sophisticated parameterisation or evaluated data. All concrete cross section data set classes are derived from the abstract class `G4VCrossSectionDataSet`, which declares methods that allow the process to inquire about the applicability of an individual data set through `IsApplicable`, and to delegate the calculation of the actual cross-section value through the method `GetCrossSection`. `G4HadronicProcess` has provision for registering data sets. A default covers all possible conditions to some approximation. The process stores and retrieves the data sets through a data store that acts like a FILO stack (a “Chain of Responsibility” with a First In Last Out decision strategy). This allows the user to map out the entire parameter space by overlaying data sets and hence optimise the overall result. An example is the use of the cross-sections for low energy neutron transport; if these are registered last by the user they will be used whenever low energy neutrons are encountered. In all other conditions the system will fall back on the default or other data sets with earlier registration times. The fact that the registration is done through abstract base classes with no side effects allows the user to implement and use his/her own cross-sections. An example is the use of special reaction cross-sections for K^0-nuclear interactions for ϵ/ϵ' analysis to control systematic error.

For final state production we provide the `G4HadronicInteraction` base class. It declares a minimal interface of just one pure virtual method for final state production: `ApplyYourself`. `G4HadronicProcess` provides a registry for final state production models inheriting from this class. Again, the final state production model is meant in very general terms; it might be an implementation of a quark gluon string model [18], a sampling code for ENDF/B data formats [19], or a parameterisation describing only neutron elastic scattering off silicon up to 300 MeV. The `G4HadronicProcess` delegates final state production to the applicable final state production model. `G4HadronicInteraction` provides the functionality needed to define and enforce the applicability of a particular model. Models inheriting from `G4HadronicInteraction` can be restricted in applicability in projectile type and energy and can be activated/deactivated for individual materials and elements. The design is a variant of the Chain of Responsibility pattern [16]. This allows a user to use models in arbitrary combinations and to write his/her own models for final state production. An example is the likely CMS scenario — the combination of low energy neutron transport with a quantum molecular dynamics [20] or chiral invariant phase space decay [21–23] model in the case of tracker materials and fast parameterised models for calorimeter materials, with user defined modelling of interactions of spallation nucleons with the most abundant tracker and calorimeter materials.

For dedicated isotope production codes, a base class, `G4VIsotopeProduction`, is provided. It declares a method `GetIsotope` that calculates and returns the isotope production information. Any concrete isotope production model inherits from this class and implements the method. Again, the modelling possibilities are not limited, and the implementation of concrete production models is not restricted in any way. By convention, the `GetIsotope` method returns `NULL` if the model is not applicable for the current projectile-target combination. If no applicable isotope production model is registered the `G4HadronicProcess` calculates the isotope production information from the final state given by the transport model. In addition, it provides a registering mechanism for isotope production models that run parasitically to the transport models and inherit from `G4VIsotopeProduction`. The registering mechanism behaves like a FILO stack, and the first model that returns a non-`NULL` value will be applied. In addition, the `G4HadronicProcess` provides the basic infrastructure for the accessing and steering of isotope production information. It allows one to enable and disable the calculation of isotope production information globally or for individual processes, and to retrieve the isotope production information through the `GetIsotopeProductionInfo` method at the end of each step. The

`G4HadronicProcess` is a finite state machine that ensures that the method `GetIsotopeProductionInfo` returns a non-zero value only at the first call after isotope production occurred. An example of the use of this functionality is the study of the activation of a germanium detector in a high precision, low background experiment.

In general we want to stress that finding the functional requirements of frameworks through use-case analysis has proven to be a highly effective tool. Framework components were found through bundling use-cases. Framework interfaces were defined by the need for delegation and flexibility; framework functionality was defined from detailed requirements analysis. The "Russian dolls" approach to framework design is very effective. Layering the implementation frameworks, and keeping simple and general abstractions in the upper levels of the framework hierarchy, has proven to result in a structured and well suited solution for a complex problem. Addressing more specific use-cases in lower level frameworks that implement the interfaces of the more general framework has kept the system surprisingly extendible. It has facilitated the distributed and largely decoupled contributions of many scientists.

2.3.5 Particles and materials

These two categories implement facilities necessary to describe the physical properties of particles and materials for the simulation of particle-matter interactions.

Particles are based on the `G4ParticleDefinition` class, which describes the basic properties of particles, like mass, charge, etc., and also allows the particle to carry the list of processes to which it is sensitive. A first-level extension of this class defines the interface for particles that carry cuts information, for example range cut versus energy cut equivalence. A set of virtual intermediate classes for leptons, bosons, mesons, baryons, etc., allows the implementation of concrete particle classes, such as `G4Electron`, `G4PionMinus`, etc., which define the actual particle properties and, in particular, implement the actual range versus energy cuts equivalence. All concrete particle classes are instantiated as singletons to ensure that all physics processes refer to the same particle properties.

The design of the *materials* category reflects what exists in nature: materials are made of a single element or a mixture of elements, elements are made of a single isotope or a mixture of isotopes. Because the physical properties of materials can be described in a generic way by quantities which can be either given directly, like density, or derived from the element composition, only concrete classes are necessary in this category.

Characteristics like radiation and interaction length, excitation energy loss, coefficients in the Bethe-Bloch formula, shell correction factors, etc., are computed from the element, and if necessary, the isotope composition.

The *materials* category also implements facilities to describe surface properties for the tracking of optical photons (see Section 5.10).

2.4 User actions

GEANT4 provides the abstract interface for eight user classes. The concrete implementation, instantiation and registration of these classes are mandatory in three cases, optional in the other five. This enables the user to customise GEANT4 to his/her specific situation. These user classes and their functionality result from an analysis of the user requirements document.

The three mandatory user class bases are:

- `G4VUserDetectorConstruction` for defining the material and geometrical setup of the detector. Several other properties, such as detector sensitivities and visualisation attributes, are also defined in this class.
- `G4VUserPhysicsList` for defining all the particles, physics processes and cut-off parameters.
- `G4VUserPrimaryGeneratorAction` for generating the primary vertices and particles.

For these three user classes, GEANT4 provides no default behaviour; instead there are pure abstract definitions from which the user must derive her/his own concrete classes. For example, GEANT4 defines no default physics process. Even the particle transportation process must be registered by the user, otherwise GEANT4 will not transport any particle. On the other hand, because of this, the user can easily switch the way transportation or any specific physics process without affecting any other processes or the behaviour of GEANT4. And because it is impossible to provide a set of processes which are sure to apply in every situation, and since a user needs to optimise performance for his/her application, instead of providing defaults the GEANT4 distribution provides various examples, described briefly in Section 1.4, which the user can draw on.

The optional user classes allow the user to modify the default behaviour of GEANT4. The five optional user class bases are:

- `G4UserRunAction` for actions at the beginning and end of every run.
- `G4UserEventAction` for actions at the beginning and end of every event.
- `G4UserStackingAction` for customising access to the track stacks.
- `G4UserTrackingAction` for actions at the creation and completion of every

track.
- `G4UserSteppingAction` for customising behaviour at every step.

For example, as described in detail in Section 4.5, the user can optimize the priority of processing any particle by implementing the `G4UserStackingAction` class.

3 Software process

The term *software process* refers collectively to the set of processes used by an organisation or project to plan, manage, execute, monitor, control and improve its software-related activities. Software processes define the practices that are used in the production and evolution of the software.

Although the Geant4 software product has been in production and available to the public since December 1998, a number of categories are still under active development and therefore require different treatment in terms of the application of software processes.

Most of the procedures and methods used in the Geant4 software process are derived from the RD44 project specifications. They were applied during the development phase of the project, but to a large extent are still valid.

There are many software processes applicable to Geant4, both in software development and in organisational matters. The complexity of the software involved, the wide areas of application of the software product, the huge amount of code, the number of code categories, and the size and distributed nature of the collaboration itself are all ingredients which motivate an ongoing software process improvement program [24].

Processes fall into several categories: primary life-cycle of software development, supporting life-cycle, management processes, organisational life-cycle, and user-supplier processes. A particular process can be deployed at different levels of generality. Tailoring of processes in the different domains is sometimes required, for instance for reasons of quality or stability, or for the evolution phase related to a specific domain, or due to personnel issues [25].

By software *life-cycle* is meant the phases the software product goes through between when it is conceived and when it is no longer available for use. The software life cycle therefore includes: requirements analysis, design, construction, testing (validation), installation, operation, maintenance, and retirement.

As discussed in the following sections, the life-cycle model adopted for most domains in Geant4 is both iterative and incremental (also called the *spiral* approach) [26]. In the current production and maintenance phase, the life-cycle model is essentially iterative for most domains; it allows the application of successive refinements to the existing architecture and the consideration of experienced solutions for analysis and design iterations. Concerning *software construction*, we adopted from the beginning flexible and well-defined programming and coding guidelines [27], basically dealing with adhesion to the object-oriented paradigm (data-hiding, encapsulation, etc.), performance, and portability of the software. Packaging of the software has strictly followed the domain decomposition into categories and sub-categories that resulted from the design process.

In order to achieve maintainable software and ensure its quality, the adoption of standards, wherever possible, is promoted in Geant4.

3.1 Methodology

Because of the wide variety of requirements for Geant4, not only from the HEP community but also from other fields, we expected that the final product would be a large and complex software system. It was also envisaged during the planning stage of the project that the creation of Geant4 would require HEP software expertise dispersed all over the world, which inevitably required the formation of a worldwide software collaboration. We considered that it was absolutely essential to employ an engineering discipline in the design and construction of the software.

The study of software engineering (the application of the engineering discipline to large-scale complex software systems) has led to the emergence of various "software methodologies" which prescribe a comprehensive development process. We spent considerable effort at the beginning of the project to study the software methodologies that were available. Because a software development process based on the object-oriented (OO) approach was considered to be the most promising technology at the time, we studied the feasibility of various OO methodologies during the first year (1995) of the project. We evaluated, for example, the Booch method [26], the OMT method [28], and the Fusion method [29]. For the evaluation we created a set of requirements for the methodology of which the essential points are as follows: (1) the process must be flexible, (2) it should provide a way to decompose a system into independent subsystems so that a clear job-sharing scheme (not only for code implementation, but also for analysis and design) can be defined, (3) it should provide models, notations and tools which help to exchange the ideas of design even if people are dispersed over the world, (4) it should provide for an incremental development strategy, (5) it should provide a reverse engineering capability to guarantee a way of following the evolution of the methodology.

Our most important conclusion from the evaluation was that there was no absolute measure for the selection of the best methodology. The methodologies had basically the same philosophy and approach, and defined similar phases in the software process (for example, requirement gathering, analysis, design, implementation and so on). Each methodology had its strong and weak points. The OMT method provided a base for other methodologies. For example, the Fusion method imported the OMT object model for its own object model. The Booch method provided the richest description for the analysis and design of a system. The Fusion method had an excellent capability in describing the object interactions. We also found that multiple methodologies were often employed simultaneously in the development of large-scale software.

Based on these observations, we finally decided to employ the Booch methodology for the construction of the GEANT4 software. The major reasons for this choice were as follows: (1) it provides a concept of dividing a system into independent subsystems in the design and implementation phases, (2) it has a pragmatic and common-sense approach with an incremental and iterative process, (3) it has easy-to-understand models with rich notations which fill the gap between design and implementation, (4) it has easy-to-obtain supporting software for Unix and PC environments. Although we selected the Booch method, our feasibility studies showed that the basic approach and essential components of the major methodologies were similar.

There was another reason we decided to use the Booch method: it was announced in 1994 that the OMT and Booch methods would be united. There was a strong belief at that time that this new combined version would become a *de facto* standard of OO methodology and would be supported by the software engineering community.

Our basic principle in using the Booch method was to adapt it for our purpose, not to blindly adopt it — we did not expect that it would automatically provide a series of well-defined steps which would generate the necessary output products. The essential point, one we consider key to our successful usage of the software methodology, was that we needed to judge for ourselves which elements of the methodology were important and were applicable to our project.

3.2 Object-oriented analysis and design

Although it is impossible to introduce all important concepts of object-oriented methodology in this section, we present here a brief illustration of key aspects to help those who are not familiar with the methodology.

Object-oriented analysis and design (OOA/OOD) and code implementation define major phases of the software development process of an OO methodology. OOA and OOD provide an object-based decomposition of a software system into smaller and smaller parts, each of which can be refined independently. They also offer a set of logical and physical models with which the developer can understand both the entire architecture and the fine detail of a class design. To construct such models, OOA and OOD provide a coherent set of processes the programmer can follow.

In the Booch method, the software development is structured into *micro* and *macro* processes. The micro process serves as the framework for an iterative and incremental approach in each phase of the development. It is similar to the so-called “spiral model”. The macro process serves as the controlling framework for the micro process, and is similar to the “waterfall model”. It consists of four phases: (1) requirement gathering, (2) OOA, (3) OOD, (4) code implementation and evolution. In the following we summarise activities done in each phase of the macro process for Geant4 development. Various aspects which we found to be important for the worldwide collaboration are also described.

3.2.1 Requirement gathering and OOA phase

We started this phase by collecting user requirements for a new detector simulation software product. The Geant4 core team (see below) created a draft requirements document and it was distributed among the GEANT3 user community. After receiving feedback from the community, we elaborated the requirements and summarised them using the ESA standard format [30]. The resulting requirements document was not a static one; it was updated many times during the development process.

Then we analysed the requirements document to identify all major objects in the problem domain, including all data attributes and major operations that would be needed to provide the required functionality. We produced central models (class diagrams) containing all the semantics of the system in a set of concise but accurate definitions. We also identified clusters of classes (class categories) that were themselves cohesive, but were loosely coupled relative to other clusters. Major products in this phase were the requirements document, class diagrams, scenario (object interaction) diagrams and a class category diagram. In this phase the core team (6–7 people) played an essential role. With this relatively small number of people, the members could work closely together even though they were located in Switzerland, England, France and Japan.

3.2.2 OOD phase

The goal of this phase was to elaborate the models created during the analysis phase so that the objects and classes could be coded and executed. The major products we produced in this phase were the updated class diagrams and scenario diagrams. We found that in most class categories the OOA and OOD were concurrent processes. About two-thirds of the first year of the project were spent on OOA and OOD, and no C++ code was written during this phase. The "class category diagram" created at the end of the OOA had a fundamental importance during the further course of the worldwide development. We used each class category as a unit to share tasks in the OOD and in the implementation phase. The loose coupling between categories allowed us to implement each category relatively independently. A working group was established for each category, and members of a group could work without interfering with the work in other categories. Also, each working group could be kept relatively small because a category covered only limited functionality. This enabled members to work in a very efficient manner.

3.2.3 Code implementation and evolution phase

We started this phase by writing C++ prototype code based on the design created during OOA and OOD. Further refinements of the design were done based on performance of the program or addition of new requirements. Regular incremental releases of the code and progress reports to the CERN review committee provided clear milestones for the project. Each release was preceded by an acceptance testing phase and thus gave us the opportunity of regularly testing the quality of the product. In this phase, the micro process played a major role in the code development.

3.3 Software process improvement

Software Process Improvement (SPI) is an activity which belongs to the organisational life-cycle category, and therefore must be indeed "life-cycle driven" and regularly applied. As such, SPI is a process which cannot be forced and must be gradually applied, after identifying the right priorities and objectives [25].

The main goal of the SPI program [24] in GEANT4 is to understand, determine and propose applicable procedures for software development and maintenance in the *production* phase of our software. In view of this, we periodically perform process assessments making reference to the ISO/IEC SPICE Model [31,32]. Experience from members in the GEANT4 Collaboration is sometimes used to help identify weaknesses in some areas and where to apply SPI. Establishing well defined methods and procedures is of vital importance for the GEANT4 project whose mandate is first to provide to users, and then to maintain, a software product with a reasonably long lifetime, good reliability and robustness.

3.4 Configuration and release management

The creation and modification of requirement and design documents, user documentation and source code are activities shared by collaboration members located at many different sites. To avoid incompatible revisions and to ensure consistency, changes in the code and documentation must be controlled and tracked. In GEANT4 we use the Concurrent Versions System (CVS) [33], which maintains a central repository for all documents and source code and provides all the necessary functionality for change management. The repository consists of a tree of directories on backed-up disk space in an AFS [34] distributed file system. For collaborators without direct access to AFS, the repository is also accessible though a server using the CVS built-in client-server protocol

The source code directory structure follows directly from the decomposition of the software into domains or categories. In each category the coordinator is responsible for coordination of development, testing, bug fixing, and release of the software and documentation in that category. Most categories are split further into sub-categories with corresponding subdirectories for source code, and so on. This splitting eases the development task for a given developer by separating development in one sub-category from that in other sub-categories. In addition several developers can share the work on a given file, as CVS supports concurrent editing of files. At installation time, the user has the option of building one object library for each category or building independent "granular" libraries in each sub-category.

The build system is based on GNU Make [35]. Each source file is analysed for dependencies on other source files so that the system knows which parts need to be rebuilt after a change. For building applications, the dependencies are used to obtain a list of the needed category or sub-category libraries in correct linking order.

For each file in the repository CVS tracks versions, keeps comments on the changes made and allows symbolic tagging of related versions. A new version is generated whenever a developer "checks in" a modified file. It is possible to retrieve any previous version or the differences between any two versions, including the developers current modified version. A tag is a symbolic name given to a set of files with each file having its specific version. Tags are used in the preparation of releases, for bug fixes and for regular testing. New or modified code in a category must be submitted as a tag for testing [36] and pass all tests before this code can be included in a major public release [37].

Important bug-fixes are periodically collected and publicly made available in the form of patches or minor releases. Bug fixes to a particular release are done on a CVS branch starting from the released version. Changes on this branch can be merged back into the current development version, assuring that no fixes get lost. Branches are also useful if global changes have to made and tested. The changes are then first done on a branch without disturbing ongoing development, and once successfully tested, these changes are merged back into the development version. This was successfully used on several occasions, for example when the code was ported to strict ISO C++ compilers requiring the use of the `std::` namespace prefix for standard library classes.

3.5 Quality assurance and testing

As part of our software process, the development of effective tests and testing procedures has been a major effort of the Collaboration as a whole and, particularly, of the dedicated Testing and Quality Assurance working group. As with the other software methodologies, we have been guided by established and well-documented practices in software engineering [36].

Software testing protocols typically fall into two classes: *unit* testing and *integration* or *system* testing. Each category team is responsible for unit testing, or testing the functionality of their own code in relative isolation from other categories. For example, there are unit tests dedicated to geometry, tracking, and the various physics processes (electromagnetic, hadronic, and so on). A team is expected to devise test programs with enough coverage to exercise the code within their category and to perform these tests regularly as new or revised code is developed. As a baseline for testing each category team uses the latest reference version of Geant4 as tagged and announced by the System Testing Team (STT).

In practice, unit tests can detect many faults but comprehensive testing requires the inclusion of interactions between categories. Testing the code as a whole with categories working in concert is the task of the integration or system testing procedure.

In this procedure, new code in a category is tagged and the tag is proposed for testing to the STT. If several category tags have been proposed, the STT must choose the order and timing of their introduction into the test cycle. This requires careful judgement, keeping in mind the dependency relationships of code changes, the need for prompt feedback to developers, and the virtues of moderating the amount of new or modified code that is brought into each round of tests.

System testing is done in parallel on a set of "test platforms" representing as far as possible the range of actual systems in use in the Geant4 user community. Each platform comprises a machine architecture, operating system, and C++ compiler. For example "Linux-g++" designates a Linux (Intel) system with the GNU C++ compiler.

Within each platform, testing is further diversified due to variations in building the tests, such as compiling "debug" or "optimised" versions, and the selection of code variants such as ISO compliant C++.

With about 6 platforms and about 35 tests to be performed on each, it is clear that system testing is a large and complex task and must be well organised and streamlined to make it a sustainable process. Consequently the STT have developed a *system testing framework* which utilises a set of scripts and tools to perform almost all phases of testing.

The launching and controlling of tests is done through standard mechanisms such as `ssh`, `cron`, and `mail`. Source code, test cases, and test logs are maintained on a global file system (AFS), although local disk space is preferred for building libraries and running executables.

When a category team tags some code for testing, CVS automatically sends the relevant information to an extended version of Bonsai [39], a Web-based CVS query and database system. Bonsai has been modified by the STT to support tags-based processing and to provide an on-line form where category teams can submit new tags for system testing.

Another Web-based tool, Tinderbox [40], is being expanded and adapted to Geant4 and will provide automated monitoring, logging, and problem detection and reporting (including hyperlinks to suspect source code) for all system tests, both completed and in progress. To aid with both development and testing, the STT also maintains an indexed and cross-referenced source code browsing facility based on LXR [41].

At the conclusion of a test cycle, accepted tags are incorporated into the next reference version of Geant4. For rejected tags, reports about test failure modes are inserted in the Bonsai system for operative recognition and elimination. Problems that cannot be resolved quickly may be logged into the problem tracking system [38].

Although unit and system testing are critical to ensuring the integrity and correctness of the Geant4 code, it is also important to build quality in from the start. To this end we employ various Quality Assurance tools such as CodeWizard, which detects unsafe, nonstandard and error-prone coding practices in the source code, and Insure++, which detects data integrity and memory management problems in the running executable. These tools, as well as our adopted coding guidelines, help to provide a front-line defence against the introduction of actual or potential problems into the software.

The Collaboration offers user support for Geant4, providing assistance with software problems, consultancy on results, and response to enhancement requests. In this section we explain in more detail our support model and the process by which we provide it.

Not only is the maintenance and development of the various Geant4 domains distributed amongst the collaborators, but so is the responsibility for user support and documentation. Object-oriented technology makes such a wide distribution of responsibility among the experts of different parts of the software package possible and effective.

Users of the software who encounter a problem in running the code can use an Internet-based problem reporting system. The system is accessible from the World Wide Web and is open to all users. It is set up automatically to assign problem reports to the responsible person according to the category affected. He or she may accept the report and respond directly or forward it to a colleague. This system is a customised version of the open source reporting tool Bugzilla [42]. Besides routing the problem to specialists, it tracks and documents the responses until the problem is resolved.

New requirements, such as requests for new functionality or refinements of existing abilities, are presented to and decided by the TSB. The TSB sets the priorities and agrees on time-scales for the fulfillment of new requirements. Such support is guaranteed to collaboration members, while requests from outside are handled on a "best effort" basis.

For each member organisation a contact person has been designated who acts as a first reference for Geant4 users in that locality, which may include affiliated institutions, user groups, and others in the same geographic area. The contact person is expected to respond to enquiries, to help resolve simple problems, and to forward more specialised queries to the relevant expert(s). This method is chosen to avoid the overhead of channelling all problems through a single central group.

This distributed user support model arose naturally from the existing distribution of expertise and manpower across experiments, laboratories, and institutes which have contributed to the creation and maintenance of Geant4. It offers major advantages over the traditional central support: a larger number of people are involved, each in the domain of their competence, and in many instances supporting code that they developed.

To exploit these advantages, an adequate structure is needed to filter, analyse, dispatch, or prioritise the users' requests, and also to provide the user with a direct interface to which one can refer without knowing the details of the user support mechanism. The needed structure is provided by the TSB and the working groups.

4 The kernel

4.1 Global structure

The kernel manages the tracking of particles taking account of the geometry, fields and physics processes. Efficiency is a key issue and various optimisation techniques are used. GEANT4 provides ways of controlling the order of processing of tracks. User code is invoked when particles enter *sensitive* volumes so that *hits* and *digitisations* can be scored. All this is described below.

GEANT4's logical structure and the *user action* classes were essentially described in Section 2.2 and 2.4.

4.2 How a particle is tracked

In spite of its name, tracking in GEANT4 does not transport particles, transportation is performed by the transportation process which is handled as one of the generic processes (the transportation process itself is described in some detail in Section 4.4).

`G4TrackingManager` is an interface class brokering transactions between the event, the track and the tracking categories. The tracking manager (a singleton instance of the class) handles the necessary message passing between the upper hierarchical object, which is the event manager (a singleton instance of class `G4EventManager`), and lower hierarchical objects in the tracking category. The tracking manager receives a track from the event manager and takes the necessary actions to complete tracking it.

`G4SteppingManager` is the class which plays an essential role in tracking the particle. It takes cares of all message passing between objects in the different categories which are relevant to transporting a particle (for example, geometry, interactions in matter, etc.). Its public method `Stepping` steers the stepping of the particle. In the implementation of the algorithm, the inheritance hierarchy of the physics interactions plays an essential role. This hierarchical design of the physics interactions enables the stepping manager to handle these as abstract objects; the manager does not need to be knowledgeable of the concrete interaction objects, for example, bremsstrahlung, pair creation, etc. The actual invocations of various interactions during the stepping are done through the dynamic binding mechanism. This is a powerful programming technique which makes the tracking catgory completely shielded from any change in the design of classes in the physics process, i.e., if we add in future a new physics process for a particle, it is not necessary to change anything on the tracking side.

Objects of class `G4Track` represent the particles which are handled by the stepping manager. Each object holds information particular to each step of a particle, for example, the current position, the time since the start of stepping, the identification of the geometrical volume where the particle is, etc. The dynamic information of the particle, like momentum and energy, is held through a pointer to an object of type `G4DynamicParticle`. Also the static information of the particle, like mass and charge, is stored through the pointer to an object of type `G4ParticleDefiniton`. Here the aggregation technique is extensively employed to keep the tracking performance very fast.

As described in section 2.3.3, each physics process proposes a step length, returned by `GetPhysicalInteractionLength` (see Section 4.3). For example, for interaction processes it is the distance to an interaction in the current material. The stepping manager selects the process that proposes the shortest step length. This selection, which has to take account also of geometrical boundaries and user parameters, is described in more detail in Sections 5.3 and 5.4.

An object of type `G4TrajectoryPoint` holds the state of the particle after propagating one step. It includes information about space-time, energy-momentum, geometrical volume, etc. A `G4Trajectory` object aggregates all `G4TrajectoryPoint` objects which belong to the particle being propagated.

4.3 Process management

A large variety of interactions is experienced by particles passing through matter. In GEANT4 this variety is expressed by a division into seven major process categories: *electromagnetic*, *hadronic*, *transportation*, *decay*, *optical*, *photolepton hadron*, and *parameterisation.*

In designing GEANT4, we focused on the generalisation and abstraction of physics processes before considering the implementation of the varieties. Our approach enables anyone to create a process and register it for a particle type in a GEANT4 simulation much more easily than in GEANT3. This openness should allow the creation of processes for novel, domain specific or customised purposes by individuals or groups of users.

All physics processes are treated in a “unified manner” to describe how particles behave in a material. As described in section 2.3.3 on tracking, two kinds of process methods play an important role: one is `GetPhysicalInteraction-Length` (abbreviated `GPIL`) and the other is `DoIt`. There are three kinds of `DoIt` methods or actions together with three `GPIL` methods corresponding to each `DoIt`. These are `AtRestDoIt`, `AlongStepDoIt`, and `PostStepDoIt`. All physics processes are derived from the base class of `G4VProcess`, which provides three virtual `DoIt` and `GPIL` methods. In other words, all physics processes can be treated in the same manner from the tracking point of view.

Each process can perform any combination of these three `DoIt` action. This is a major innovation in GEANT4, which goes beyond the categorisation of processes made by previous simulation packages that distinguished, at most, two types of process, discrete and continuous. Those two are still available as special cases but, in addition, several novel types of process are possible.

As a result, the tracking code is completely general and common to all processes of all particle types. This unified model for physics processes gives flexibility in design of a physics process. For example, the transportation of particles is a kind of process in GEANT4 and a specific transportation process can be applied in various cases (such as for transportation in electric fields).

Each particle type contains a list of physics processes that the particle can undertake. The process manager of each particle manages the list of processes. Users can choose physics processes which are necessary for their own simulation and register them via the process manager. The process manager also contains information about the ordering of `DoIt` actions for each process in the list.

In GEANT4, the concept of *particle change*, represented by the class `G4V-ParticleChange` and its derivatives, is introduced to keep the results in `DoIts`, i.e., the final state of the track and secondary tracks. Thus, only these objects know which properties the physics process has updated. A physics process can define its own *particle change* derived from the base class `G4VParticleChange` to gain performance.

Clear separation between process and tracking functionality can be realised by using *particle change*. Processes cannot change track information directly; they can only propose changes as a result of an interaction. On the other hand, the tracking accepts and judges proposals from processes and triggers their action. In addition, the tracking controls the timing of updating the step and track information based on the *particle change*.

This approach of using *particle change* and the abstraction of physics processes ensures that we can easily develop new physics processes and/or extend the functionality of existing processes.

4.4 *Propagation in the detector model*

The primary task of the *geometry* category is to supply information to the transportation process and ultimately to the tracking manager for the geometrical propagation of tracks. This includes propagation in a field, for example, a magnetic field.

In GEANT3, as in GEANT4, particles are moved in steps that are determined by physics processes or by the detector geometry; however, in GEANT3, small "pushes" are adopted to guarantee the change of volume at a boundary in the face of computational rounding errors. This mechanism may cause errors, especially in the case of photon reflection, and it has been demonstrated to be inefficient when volume boundaries are not coincident, since a series of many small steps may be required when the "push" is not large enough. GEANT4's propagation methods were designed to overcome these limitations without sacrificing accuracy and efficiency. After a step to a boundary a track's state records whether it is on a boundary, whether it is exiting the current volume, etc.

Volumes effectively have boundaries of a very small but finite "thickness" to take into account the round-off and accumulated errors of floating point arithmetic. In this "tolerant" geometry [43], intersections with boundaries less than the tolerance from the current point are ignored if the direction of the particle is away from the boundary. The thickness (or tolerance) is chosen to be very small compared to detector features but much larger than the expected arithmetic errors. Note that the internal unit of length can be chosen at compile time (see Section 2.2) so that this condition can generally be satisfied.

The number of steps a particle must take to traverse a detector is therefore much reduced. However, in order to traverse a detector model geometry efficiently, it is also critical to reduce the number of candidate volumes for which intersections have to be calculated and GEANT4 has adopted an optimisation technique which is described next.

4.4.1 *Tracking optimisation*

While tracking through the detector, a particle may encounter any one of several detector parts at each step. Calculating the intersection of a track with every daughter volume at each tree level would be extremely inefficient. Different methods, some inspired by techniques used in ray tracing, to lower the number of candidate volumes to be tested for intersection, have been evaluated.

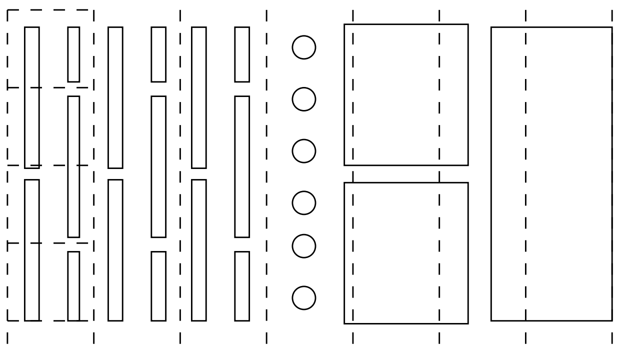


Fig. 4. Smart Voxels. A mother volume with divisions along the horizontal axis. Each one of these slices has an independent set of vertical divisions (voxels). Here, the first one from the left is shown.

The technique of *virtual divisions*, like the one adopted in GEANT3, consists of having mother volumes sliced evenly along one axis into sections. Each section stores pointers to the sub-volumes it contains, compressed by bunching together common lists. This scheme works well in a hierarchical detector description, where the number of daughter volumes at each node is small. It will fail for a "flat" geometry, i.e., where many volumes are placed at the top or high level node without regard to relationship, as might happen for geometries imported from a CAD system, particularly if the level of detail varies with position.

Other techniques based on *fixed* or *variable grids* were investigated. Each cell of the grid stores a list of pointers to the volumes intersected. The contents of each cell can be determined entirely at initialisation time, based on the bounding box of each volume. Fixed size grids have the disadvantage of memory consumption for fine granularity dictated by small detector components. The use of variable grids would overcome this problem but would make the testing of intersections and the determining of which cell a particle is inside more complex.

In GEANT4 we have devised a new technique derived from the *voxel* based method, used in ray tracing, where space is subdivided into cubic volume elements (voxels) and a tree based map is created by recursively dividing the detector into octants. This traditional voxel based technique retains the disadvantage of grid based methods in that every voxel intersected along the particle's trajectory must be tested for intersection of its contents. In GEANT4's *smart voxels* technique [44], for each mother volume, a one dimensional virtual division is performed. The best axis for the virtual division is chosen by using an heuristic. Subdivisions (slices) that contain the same volumes are gathered into one in order to optimise memory and performance (see Figure 4).

Each division containing too many volumes is then refined by applying virtual division again, using a second Cartesian axis. If the resultant subdivisions still contain too many volumes, a further refinement can be performed by dividing again along a third Cartesian axis.

For a hierarchical detector description the mother volume local coordinate system is usually a sufficient guide to the choice of voxel decomposition axes. For a “flat” geometry, the smart voxels technique produces a simple virtual division if volumes are regular placed or a tree up to three levels deep if it contains many volumes of different sizes and placements.

Smart voxels are computed at initialisation time and do not require large memory and computing resources. At tracking time the searching is done in the hierarchy of virtual divisions. This method for tracking has been found to be very efficient. Also it very much reduces the need to tune the detector description, since the performance in inadvertent or unavoidable “flat” regions of an otherwise hierarchical description is not much compromised.

4.4.2 Transportation in a field

Charged particles moving in a field do not follow linear trajectories between interactions. In a uniform magnetic field their trajectories are helical (in the approximation of small energy loss), while in non-uniform fields they are curves that, in most cases, cannot be described analytically.

Propagation in the detector thus involves two tasks: first the calculation of the trajectory (numerically if the field is non-uniform) and then the finding of its intersection with volume boundaries.

We solve the particle’s motion using a selection of methods, the majority involving Runge-Kutta integration. The default method is a fourth order Runge-Kutta, while lower order methods are available for fields that are not smooth enough and higher order methods are available for fields that are smooth and do not vary greatly.

Furthermore, for magnetic fields in particular, a new set of integration methods has been created that combines Runge-Kutta and the known helical solution for uniform fields. Here the “baseline” linear solution of first order Runge-Kutta solution is replaced with a helical one in a scheme similar to the implicit or explicit Euler schemes of Runge-Kutta.

These new integration methods have the ability to integrate over a large number of “turns” of a near-helical path in an almost uniform field. Thus they are best suited for fields that are nearly uniform and reasonably smooth. In these circumstances, and for particles whose motion takes them over several to thousands or more “loops” or turns of a helix, these new methods can offer a large performance benefit compared to “ordinary” Runge-Kutta methods, while taking into account field variations.

To calculate the intersection with volume boundaries we split the curved path into sections and approximate each section by its chord. The algorithm's accuracy and performance is controlled by a set of parameters, which can be specified by the user. The sections are chosen so that the maximum estimated separation between the real (curved) path and the corresponding chord is smaller than the "miss-distance" parameter. The chord is then used to test for intersection with the boundary of a volume and, of course, it might miss where the curved track would intersect. However the maximum depth in that 'missed-volumed' that is entered by the curved track should be no less than the "miss-distance".

Once a candidate intersection is found, it is refined to within a distance defined by another accuracy parameter. Currently the final intersection point is taken to lie on the chord. Thus the intersection accuracy parameter must be chosen carefully to limit the systematic error in tracks whose position is measured accurately, else reconstructed momenta will be influenced by this error.

The effects of a particle's motion on the precession of its spin angular momentum in slowly varying external fields are simulated. The relativistic equation of motion for spin is known as the BMT equation [45]. The equation demonstrates a remarkable property; in a purely magnetic field, in vacuum, and neglecting small anomalous magnetic moments, the particle's spin precesses in such a manner that the longitudinal polarization remains a constant, whatever the motion of the particle. But when the particle interacts with electric fields of the medium and multiple scatters, the spin, which is related to the particle's magnetic moment, does not participate, and the need thus arises to propagate it independent of the momentum vector. In the case of a polarized muon beam, for example, it is important to predict the muon's spin direction at decay-time in order to simulate the decay electron (Michel) distribution correctly.

4.5 Priority control of tracks

Since the tracks are represented by C++ objects, it is quite straightforward to use standard containers to stack them. Event handling in GEANT4 has three stacks (by default), namely "urgent", "waiting" and "postpone to next event". Each is a simple *first-in-last-out* stack.

In the former GEANT3, there was only one stack and each track was assigned a priority. For every pop request it was necessary to scan for a track with the highest priority, a rather heavy procedure once many secondaries were stacked.

In GEANT4, priorities can be controlled easily by using the two stacks "urgent" and "waiting". A user simply chooses in which stack to store the newly pushed track by implementing, instantiating and registering a concrete derivative of `G4UserStackingAction`. When the "urgent" stack becomes empty, the user code is notified so that the current event can be examined to see if it is worth continuing to simulate or whether it is better to abort. If continuing, tracks in the "waiting" stack are re-examined and some or all are transferred to the "urgent" stack for the next stage of simulation. This continues recursively until the event is complete or aborted. Not only the speed of popping the track with the highest priority but also the capability of easy abortion of uninteresting events make the simulation much more powerful.

4.6 Hits and digitisation

4.6.1 Detector sensitivity

In GEANT4, a *hit* is a snapshot of a physical interaction or an accumulation of interactions of a track or tracks in a "sensitive" detector component. On the other hand, the term *digit* represents a detector output, for example, an ADC/TDC count or a trigger signal. A *digit* is created from one or more *hits* and/or other *digits*. Given the wide variety of applications of GEANT4, how to describe the detector sensitivity and the quantities a user needs to store in the *hit* and/or *digit* vary greatly. Thus GEANT4 provides only the abstract classes for both detector sensitivity and *hit/digit*.

Each *logical* volume can have a pointer to a *sensitive detector*, which is an object of a user class derived from the abstract base class `G4VSensitiveDetector`. A *sensitive detector* creates *hits* using the information given in the current step. The user has to provide his/her own implementation of the detector response. *Hits*, which are user-defined objects derived from class `G4VHit`, are collected in an *event* object. At tracking time, when the step is inside of a volume which has a pointer to a *sensitive detector*, this *sensitive detector* is invoked with the current step information.

In contrast to *sensitive detector*, which is invoked automatically at tracking time, the *digitisation module* must be invoked by the user's code. Digitisation may be done during event processing, at the end of each event, and/or even after some number of events had been processed to simulate "pile-up".

4.6.2 Readout geometry

In some cases, the readout segmentation can be different to the geometrical structures of the detector. For example, the user may implement a detailed sandwich structure of a sampling calorimeter, while the readout collects the energy deposition of some of the layers. The *readout geometry* is an artificial geometry which can be associated with a *sensitive detector*. Each *sensitive detector* can have its own *readout geometry*. (Note that the transportation process does not see the volume boundary of *readout geometry* and thus a step does not end at the boundary of *readout geometry*.) Once a step belongs to a *sensitive detector*, geometrical information of both the "real" tracking and the *readout geometry* geometries are available to the *sensitive detector*.

5 Physics processes

5.1 *Scope*

The GEANT4 toolkit contains a large variety of complementary and sometimes alternative physics models covering the physics of photons, electrons, muons, hadrons and ions from 250 eV up to several PeV. The hierarchical structure of the *processes* category was introduced in Sections 2.3.4 and 4.3. There are seven major sub-categories — *electromagnetic*, *hadronic*, *transportation*, *decay*, *optical*, *photolepton_hadron*, and *parameterisation*. The first two, *electromagnetic* and *hadronic*, are further sub-divided, as mentioned in Section 2.3.4 and further described below.

We stress our design goal of achieving openness of physics implementation. Object-oriented programming makes the structure apparent; the result is a highly granular implementation, each component of which can be inspected at source code level. The abstract interface common to all processes makes the tracking independent from the type of process. This, together with the modular architectural framework, also allows the continuous development of new models without affecting the previous code.

5.2 *Processes and models*

For a particle interaction or decay it is useful to distinguish between the *process*, i.e., a particular initial and final state, which therefore has a well-defined cross-section or mean-life, and the *model* that implements the production of secondary particles. It allows the possibility of offering multiple models for the same process. One way this is exploited was described in outline in Section 2.3.4 and further examples are given below.

5.3 Interactions and decays

A particle in flight is subject to many competing processes. Moreover, in a real detector, it will often travel through many regions of different materials, shapes and sizes before interacting or decaying. In simulation, the particle proceeds in steps, and we have to find an efficient and unbiased way of choosing what limits the step and, if the particle continues, of updating the parameters for the next step.

Let us consider the interaction or decay of a particle in flight. (Similar considerations apply to a particle at rest.) Firstly, we calculate a distance to the point of interaction or decay. This is characterised by the mean free path λ. The probability of surviving a distance ℓ is

$$P(\ell) = e^{-n_\lambda},$$

where $n_\lambda = \int_0^\ell d\ell/\lambda(\ell)$.

For a decay, $\lambda = \gamma v \tau$, where v is the velocity and τ the mean life. For an interaction, if the cross-section on isotope i of mass m_i that has fraction x_i by mass in the current material of density ρ is σ_i, then $1/\lambda = \rho \Sigma_i \{x_i \sigma_i / m_i\}$. We must keep in mind that λ varies as it looses energy and changes discontinuously at a geometrical boundary.

The key point to note is that the probability distribution of n_λ is a simple exponential independent of material and energy. So, at the point of production of the particle we set

$$n_\lambda = -\ln \eta$$

where η is a random number uniformly distributed in the range $(0, 1)$, and this is used to determine the distance to the point of interaction or decay in the current material. This information from all processes for the particle (each process using a different random number, of course) is used to decide what happens.

5.4 Deciding which process limits the step

Processes other than interaction or decay also compete to limit the step. *Continuous energy loss* may limit the step to preserve precision. Also, *transportation* insists that the step should not cross a geometrical boundary. The user can also request a maximum allowed step.

The process which returns the smallest distance is selected and its *post step action* is invoked. If this is an interaction or decay, the particle is *killed* and secondaries are generated. If not, the particle gets another chance to interact or decay; n_λ for each process is decremented by an amount corresponding to the step length and the whole algorithm is repeated at the next step.

5.5 Decay processes

Class `G4Decay` implements *at rest* and *post-step* actions for decay at rest and in flight respectively. It chooses a decay time or path according to the above algorithm. It also chooses a decay mode from the branching ratios in the decay table for the particle. (The user has, nevertheless, the freedom to fix the proper decay time and decay mode of primary particles.) GEANT4 provides default decay tables for most particles, such as π, K mesons, Σ, Λ hyperons and resonant baryons, based on data from the Particle Data Group [46].

There are many models for determining the distribution of secondaries, for example V-A theory for muon decay, Dalitz theory for π^0 decay, or simple phase space. In GEANT4, concrete classes derived from `G4VDecayChannel` are implemented for specific models and attached to each decay mode.

Decays of heavy flavour particles, such as B mesons, are very complex, with many decay modes and decay mechanisms. GEANT4 does not attempt to model these but provides two ways of dealing with them that take advantage of external event generators. In the first way, the *external decayer* approach, the `G4VExtDecayer` class provides an interface to the external package that decides the decay mode and secondary particle momenta. This activated by attaching a concrete implementation of this class to the `G4Decay` object of that particle.

The second way, the *pre-assigned decay mode* approach, decays of heavy particles are simulated by the primary event generator, which attaches these daughter particles to the parent using the `PreAssignedDecayProducts` method of `G4DynamicParticle`. `G4Decay` adopts these pre-assigned daughter particles instead of asking `G4VDecayChannel` to generate the decay products.

5.6 Electromagnetic processes

The range of available electromagnetic processes is extensive. Whenever available, use is made of the public evaluated databases distributed by a variety of international sources; this contributes to the reliability and openness of the physics implementation.

Geant4 electromagnetic physics is usable in a wide variety of simulation domains; a selection of applications and results can be found in [47–49].

5.6.1 Standard electromagnetic processes

Geant4 standard electromagnetic physics provides a variety of implementations of electron, positron, photon and charged hadron interactions. Photon processes include Compton scattering, γ-conversion and the photo-electric effect. Electron/positron processes handle bremsstrahlung, ionisation and δ-ray production, positron annihilation and synchrotron radiation. The energy loss process manages the continuous energy loss of particles due to ionisation and bremsstrahlung. A significant feature of this is an algorithm [50] which can generate low energy δ-rays only near the boundaries of volumes, which can lead to an improved performance while keeping the quality of physics. The ionisation and energy loss of hadrons has several models to choose from, including Photo-Absorption Interaction (PAI) [51].

The Geant4 multiple scattering process can handle all charged particles. It is based on a new model that simulates the scattering of the particle after a step, computes the mean path length correction and the mean lateral displacement. Its performance is compared to GEANT3 and experimental data in Figure 5.

A shower profile resulting from Geant4 standard electromagnetic physics processes is compared to GEANT3 and experimental data in Figure 6.

Standard electromagnetic processes average the effects of the shell structure of atoms and cannot expected to simulate details below 1 keV.

5.6.2 Range cuts

In Geant4, charged particles are tracked to the end of their range. However, for performance, when generating the particles produced in an interaction, a process may, optionally, choose to suppress particles whose range, as defined below, would be less than a user-defined value that we name the range cut. In this case the process must add the energy of the particle to the energy deposited during or at the end of the step.

Range is used, rather than energy, as a more natural concept for designing a coherent policy for different particles and materials. For photons, the absorption length defines the cut - see Section 5.6.3.

For some processes, such as δ-ray and bremsstrahlung production, the use of a cut is not an option but a necessity, in order to suppress the generation of large numbers of soft electrons and gammas. The energy of non-produced particles is

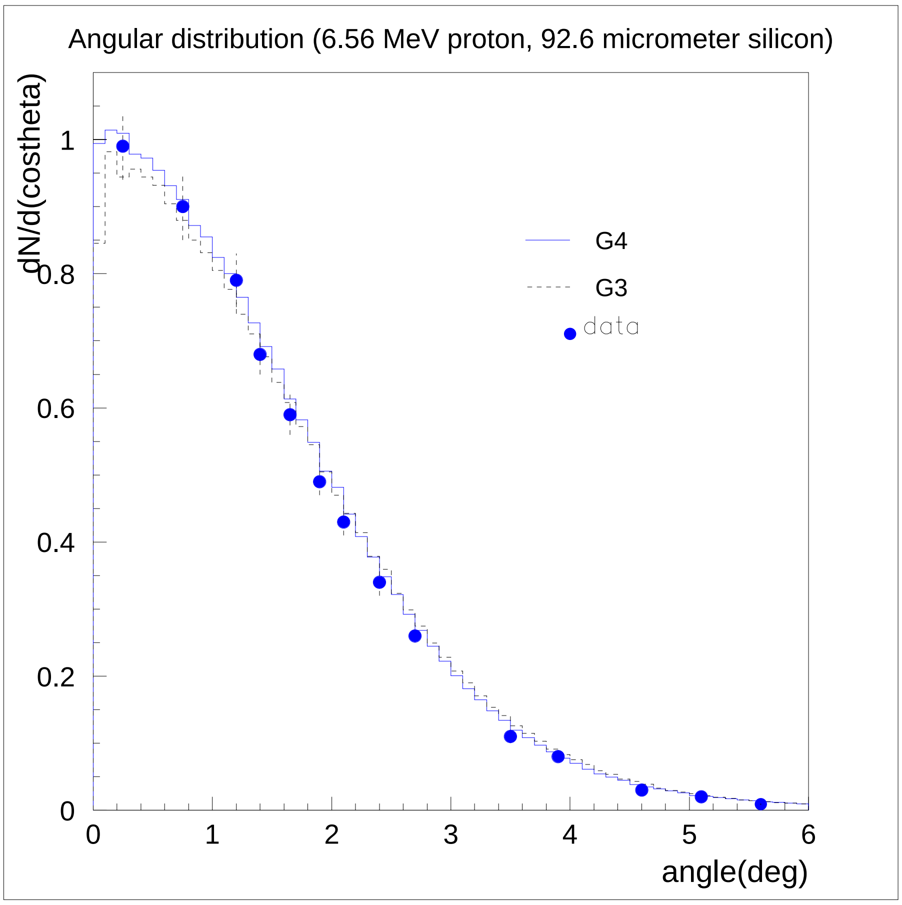


Fig. 5. Multiple scattering of 6.56 MeV protons by 92.6 μm of silicon: comparison of Geant4, GEANT3 and experimental data from [52] — the angular distribution of exiting protons.

transferred from the *discrete* component of a process to the *continuous* (*along-step*) component. This also means that the interaction length also depends on the cut.

All this needs a fast way of finding the range of charged particles and the absorption length of photons in each material; Section 5.6.3 describes this in more detail.

For electromagnetic physics it is important to have a range cut which is uniform across particles and materials in order to design a coherent set of processes. We use range to ensure uniformity between different particles (in particular between electrons and photons). This production threshold concept is used by the electromagnetic processes, in particular by ionisation and bremsstrahlung.

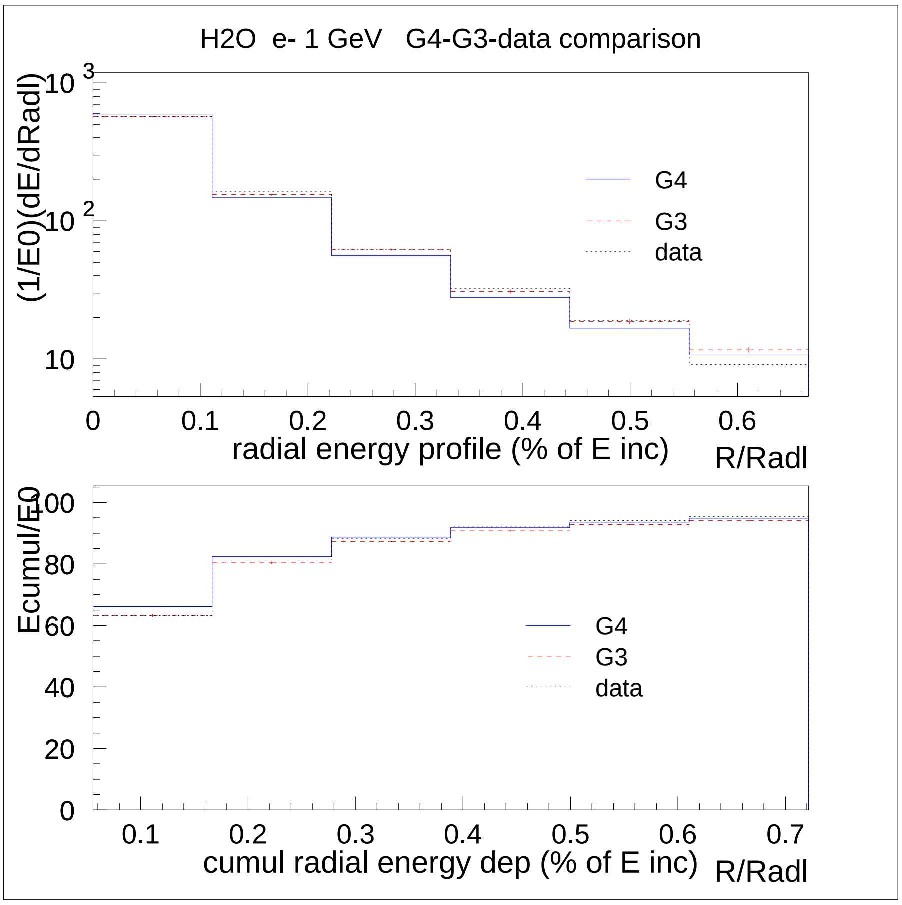


Fig. 6. Shower profile of 1 GeV electrons in water: Geant4, GEANT3 and experimental data from [53]

5.6.3 Range and absorption length tables

In order to implement the range cut policy described in Section 5.6.2, the relevant electromagnetic processes produce range-energy and aborption length-energy tables for each material for use by all processes. The range is computed by numerical integration of energy loss for electrons/positrons, muons, protons and antiprotons. The range for other charged hadrons is computed from the proton table by using the scaled kinetic energy $T_s = Tm_p/m$, where T is the particle kinetic energy, m is the particle mass and m_p is the proton mass, which is the energy of a proton with the same velocity as the tracked particle. This approach can be used because ionisation losses depend only on the velocity.

For bremsstrahlung, a cut based on the absorption length for photons is approximated as described below.

The energy loss processes for e^+/e^-, μ^+/μ^- and charged hadrons are very similar, so it is quite natural to have a common description for them.

5.6.4 Energy loss of electrons/positrons

The `G4VeEnergyLoss` class computes the continuous energy loss of electrons and positrons. The continuous energy loss is calculated as a sum of the contribution of the different processes. At present there are two processes contributing to the continuous energy loss, they are: the ionisation process (class `G4eIonisation`) and the bremsstrahlung process (class `G4eBremsstrahlung`). `G4eIonisation` calculates the contribution due to ionisation and simulates the "discrete" part of the ionisation — Moller and Bhabha scattering or δ-ray production. `G4eBremsstrahlung` computes the energy loss due to soft bremsstrahlung and simulates "discrete" or hard bremsstrahlung.

The `G4VeEnergyLoss` class also constructs energy loss and range tables for every material. First the energy loss tables are constructed and filled, simply summing the contributions computed for ionisation and bremsstrahlung. After this, it creates range tables and their inverses for e^+/e^- for every material. All the tables are constructed at the beginning of a GEANT4 run, at initialisation time. Later, during the simulation, the energy loss process performs two tasks: it imposes a limit on the step size of the particle and computes the energy loss during a step travelled by the particle.

The computation of the *mean energy loss* during a step uses the dE/dx and inverse range ($T(r)$) tables. The mean loss is

$$\Delta T = T(r_0) - T(r_0 - s),$$

where r_0 is the range at the beginning of the step of length s. For $s < \kappa r_0$, where κ is an arbitrary parameter (the *linear loss limit*), an approximation is used:

$$\Delta T \approx s \left| \frac{dE}{dx} \right| .$$

After the mean energy loss has been calculated, the process computes the *actual* energy loss, i.e., the loss with fluctuation. The fluctuation is computed in the fluctuation model GLANDZ [54], also used in the GEANT3 code.

5.6.5 Energy loss of muons

The energy loss of muons is computed by the class `G4VMuEnergyLoss`. The scheme is the same as in the case of e^+/e^-, except that now there are three processes contributing, namely the ionisation process (class `G4MuIonisation`), the bremsstrahlung process (class `G4MuBremsstrahlung`) and the direct production of e^+/e^- pairs (class `G4MuPairProduction`). They each also simulate the corresponding discrete processes — δ-ray production, hard bremsstrahlung and hard direct e^+/e^- pair production, respectively.

5.6.6 Energy loss of charged hadrons

The continuous energy loss of charged hadrons is calculated by the class `G4VhEnergyLoss`. Here there is only one process which contributes, namely ionisation (class `G4hIonisation`), which also simulates the discrete process of hard δ-ray production.

5.6.7 Bremsstrahlung

In Geant4, the user specifies the cuts for the suppression of soft particles as a distance. This is straightforwardly interpreted as a range for δ-rays, as described above; for bremsstrahlung we interpret it as follows. We use the fact that, to a good approximation at low energies, averaging over atomic shell effects, the absorption length decreases as energy decreases. In each material, a cut is established at an energy such that five absorption lengths equals the user defined distance cut. Only $e^{-5} \approx 0.7\%$ of suppressed photons of cut energy, and a lesser proportion for photons for lower energy, would travel further than the user defined distance cut. We thus obtain an approximate correspondence between the delta-ray and bremsstrahlung cuts.

An approximate empirical formula is used to compute the *absorption cross section* of a photon in an element. The *absorption cross section* means here the sum of the cross sections for gamma conversion, Compton scattering and the photo-electric effect. These processes are the "destructive" processes for photons, i.e., they destroy the photon or decrease its energy. (Coherent or Rayleigh scattering only changes the direction of the gamma, so its cross section is not included.)

5.6.8 Multiple scattering

The `G4MultipleScattering` class simulates the multiple scattering of charged particles in material. It simulates the scattering of the particle after a given step, computes the mean path length correction and the mean lateral displacement. However, it uses a new multiple scattering model [57] which does not use the Molière formalism.

Liljequist et al. [58] have calculated tables of parameters for electrons and positrons in the kinetic energy range 0.1 keV to 20 MeV in 15 materials. Our model uses these values, corrected for a nuclear size effect, withan appropriate interpolation or extrapolation in the atomic number and in the velocity of the particle when necessary.

5.6.9 Low energy extensions

A set of physics processes is implemented in GEANT4 to extend the range of validity of electromagnetic interactions down to lower energy than the standard electromagnetic processes. The currently available extensions cover processes for electrons, photons, positive and negative charged hadrons and positive ions; further extensions to cover positron and negative ion interactions are in progress. The current implementation of low energy electron and photon processes [59] can be used down to 250 eV.

The low energy package includes the photo-electric effect, Compton scattering, Rayleigh scattering, bremsstrahlung and ionisation; for completeness, a photon conversion process has also been implemented and based on the same data sources as the other low energy ones. In addition, fluorescence emission from excited atoms is also generated; the implementation of the Auger effect is in progress. The implementation of electron and gamma processes is based on the exploitation of evaluated data libraries (EPDL97 [60], EEDL [61] and EADL [62]) that provide data for the determination of cross-sections and the sampling of the final state. A simulation based on GEANT4 low energy processes for photons and electrons is compared with experimental data in Figure 7, with evidence of shell effects.

A low energy process is also available to handle the ionisation by hadrons and ions [64,65]. It adopts different models depending on the energy range and the particle charge. In the high energy (> 2 MeV) domain the Bethe-Bloch formula and in the low energy one (< 1 keV for protons) the free electron gas model are applied respectively. In the intermediate energy range parameterised models based on experimental data from the Ziegler [66] and ICRU [67] reviews are implemented; corrections due to the molecular structure of materials [68] and to the effect of the nuclear stopping power [67] are taken into account. The Barkas effect is described by means of a specialised model Figure 8 shows a comparison with experimental data for ions.

5.7 Photo- and electro-production of hadrons

GEANT4 includes photonuclear and electronuclear reactions which convert the energy flow of electrons, positrons and photons into the energy flow of mesons, baryons and nuclear fragments [55]. In the nuclear giant resonance region the cross section of the photonuclear process is comparable with the other electromagnetic processes. At high energies, because of the Froissart increase of interaction cross sections and large energy transferred to nucleus, this kind of reaction can be very important [56].

The electronuclear reactions use the equivalent photon method [55]. Approxi-

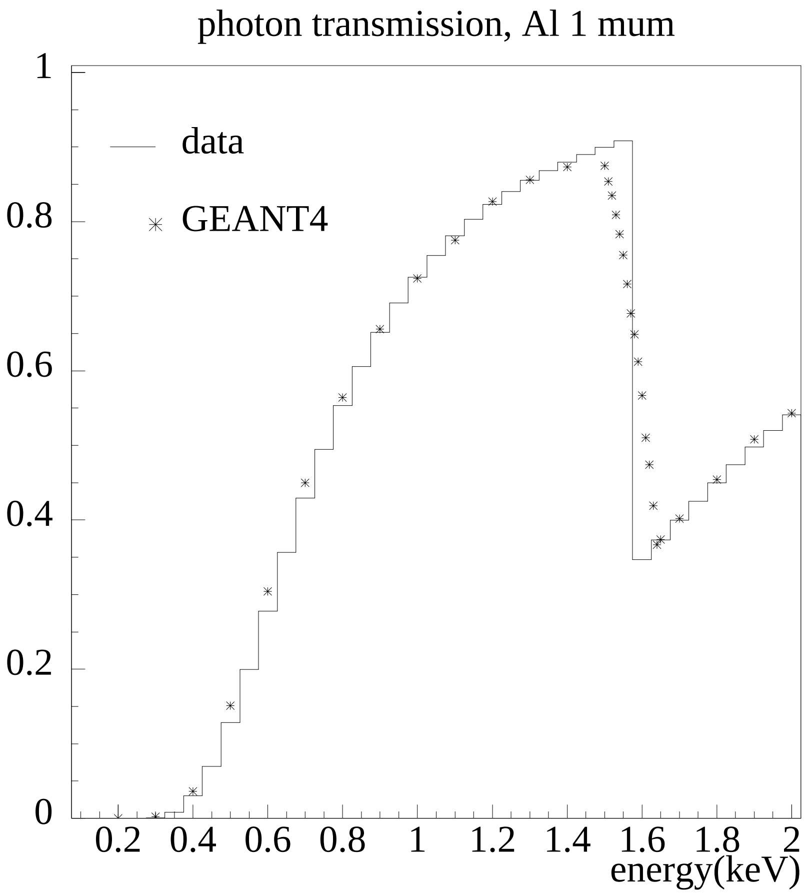


Fig. 7. Comparison of the Geant4 low energy photon simulation and experimental data, showing relevance of shell effects: photon transmission in 1 μm Al; data from [63]

mation of structure functions of nucleons and simulation of DIS reactions are under development.

5.8 Muo-production of hadrons

Geant4 also provides the nuclear interaction of muons with production of hadrons. This is important for the simulation of detector response to high energy muons, muon propagation and muon-induced hadronic background at energies above 10 GeV and relatively high energy transfers, in particular in light materials [57]. The average energy loss for this process increases almost lineary with energy, and at TeV muon energies constitutes about 10% in standard rock. Extension to lower energies, starting from the nuclear disintegration threshold, on the basis of the equivalent photon method [55], is under development.

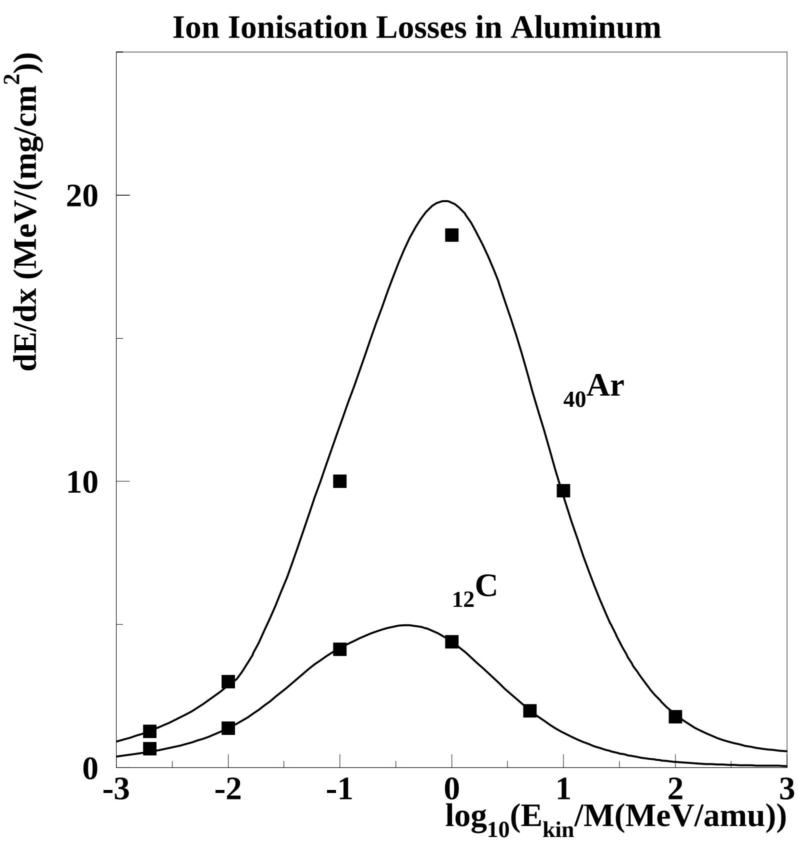


Fig. 8. Electronic stopping power of ions in Al; the accuracy of the data is approximately 5%; experimental data from [69]

5.9 Hadronic processes

The basic requirements on the physics modelling of hadronic interactions in a simulation toolkit span more than 15 orders of magnitude in energy. The energy ranges from thermal for neutron cross-sections and interactions, through 7 TeV (in the laboratory) for LHC experiments, to even higher for cosmic ray physics. In addition, depending on the setup being simulated, the full range or only a small part might be needed in a single application. The complex nature of hadronic showers and the particular needs of the experiment require the user to be able easily to vary the models for particular particles and materials depending on the situation.

For calorimeter simulation at colliders, for example, pion nuclear interactions are fundamental, and leading particle effects, transverse momentum distributions, inclusive cross-sections, and the prediction of nuclear excitation energies largely define the quantities of interest for measurement and detector design. When simulating backgrounds in the muon systems of the large LHC experiments, critical items are the production of muons in hadronic showers, as well as the simulation of punch-through and low energy neutron interactions. When studying the impact of a neutron irradiated gadolinium rod on a tumour, precise Doppler broadening of cross-sections and energy distributions of the capture photons are vital.

A simulation tool-kit is therefore required to include the calculation of cross-sections for the scattering of any incident meson or baryon (having a mean-life long enough for interactions to be non-negligible) off any stable or long lived nuclear isotope target and to include models of these interactions. Lepton-nucleus scattering should also be included. A good toolkit will offer flexible choice of alternative cross-section algorithms and interaction models that the user can choose according to his computer memory and performance and his needed precision. The ability of the expert user to extend or adapt the provided set of models is also a fundamental requirement.

Of this huge domain of energy, incident particle, target isotope and level of precision, much is already available in the standard distribution. The hadronic process category comprises several families of classes: *processes* which define each possible process and provide connections to the underlying cross sections and models which implement the process; *management*, containing classes which abstract some common properties of hadronic interactions and provide steering mechanisms for the application of appropriate interaction models; *cross sections* which encapsulate all cross section data and associated calculation methods for computing the occurrence of processes; *stopping* processes, a distinct category for particles stopping or at rest; *models*, each of which implements the final state generation of a particular process or a set of processes for a particle or class of particles within a specified energy range; and *utility* classes which provide various standardised computational methods for use by the models.

Within the *models* category are a number of sub-systems: *low energy*, *high energy*, *generator*, *neutron_hp*, *radiative decay*, etc., organised according to energy range, methodology, reaction type, and so on. This large array of models, many newly developed or adapted for Geant4, is the result of a wide-ranging development effort with many contributors. As a result of the design of the process management and steering facilities, a set of models for a given application can be chosen with great flexibility, combining broadly-applicable models with specialised ones in a well-defined way and invoking the appropriate model for a given interaction depending on particle types, energy ranges, and other characteristics.

5.9.1 Interaction cross-sections

The total cross-sections for inelastic scattering, capture of neutral particles, induced fission and elastic scattering have been carried over from GEANT3. The software design in Geant4 allows one to overload these defaults with specialised data-sets. Custom data sets are provided for proton induced reactions [70] and neutron induced reactions [71] at particle energies below 20 GeV, and ion spallation reactions [72], as well as neutron interactions at energies below 20 MeV.

5.9.2 Modelling final states

Three classes of models are distinguished for modelling final states. There are models that are largely based on evaluated or measured data, models that are predominantly based on parameterisations and extrapolation of experimental data under some theoretical assumptions, and models that are predominantly based on theory. In the following, we describe the usage of data driven, parameterisation driven and theory driven modelling approaches in Geant4.

Data driven models

When experimental or evaluated data are available with sufficient coverage, the data driven approach is considered to be the optimal way of modelling. Data driven modelling is used in the context of neutron transport, photon evaporation, absorption at rest, calculation of inclusive cross-sections, and isotope production. We also use data driven modelling in the calculation of the inclusive scattering cross-sections for hadron nuclear scattering. Limitations exist at high projectile energies, for particles with short life-times, and for strange baryons, as well as the K^0 system. Theory based approaches are employed to extract missing cross-sections from the measured ones, or, at high energies, to predict these cross-sections.

The main data driven models in Geant4 deal with neutron and proton induced isotope production, and with the detailed transport of neutrons at low energies. The codes for neutron interactions are generic sampling codes, based on the ENDF/B-VI data format [19], and evaluated neutron data libraries such as ENDF/B-VI [73], JENDL3.2 [74], and FENDL2.2 [75]. Note that any combination of these can be used with the sampling codes. The approach is limited by the available data to neutron kinetic energies up to 20 MeV, with extensions to 150 MeV for some isotopes.

The data driven isotope production models that run in parasitic mode to the transport codes are based on the MENDL [76] data libraries for proton and neutron induced production. They complement the transport evaluations in the sense that reaction cross-sections and final state information from the transport codes define the interaction rate and particle fluxes, and the isotope production model is used only to predict activation.

The data driven approach is also used to simulate photon evaporation at moderate and low excitation energies, and for simulating radioactive decay. Both codes are based on the ENSDF [77] data of nuclear levels, and transition, conversion, and emission probabilities. The decay of almost three thousand nuclide species are covered, and all emitted α, β, ν and $\overline{\nu}$ particles can be tracked by Geant4 and their interactions simulated. Since the residual nucleus is often in an excited state, the isomeric transitions are treated using the photo-evaporation classes in Geant4 (future developments planned for the toolkit include the treatment of internal conversion and atomic de-excitation following decay). In the case of photon evaporation the evaluated data are supplemented by a theoretical model (giant dipole resonance de-excitation) at high excitation energies.

Finally, data driven modelling is used in the simulation of the absorption of particles coming to a rest, mainly for μ^-, π^-, K^-, and $\bar{p}$, in order to describe the fast, direct part of the spectrum of secondaries, and in the low energy part of the modelling of elastic scattering final states in scattering off hydrogen.

Parameterised models

Parameterisations and extrapolations of cross-sections and interactions are widely used in the full range of hadronic shower energies, and for all kinds of reactions. In Geant4, models based on this paradigm are available for low and high particle energies respectively, and for stopping particles. They are exclusively the result of re-writes of models available from GEANT3, predominantly GEISHA [78]. They include induced fission, capture, and elastic scattering, as well as inelastic final state production.

Theory based models

Theory based modelling is the basic approach in many models that are provided by Geant4 or are under development. It includes a set of different theoretical approaches to describing hadronic interactions, depending on the addressed energy range and computing performance needs.

Parton string models for the simulation of high energy final states ($E_{CMS} > O(5\,GeV)$) are provided and in further development. Both diffractive string excitation and dual parton model [79] or quark gluon string [18] model are used. String decay is generally modelled using well established fragmentation functions [80]. The possibility of using quark molecular dynamic [81] is currently in preparation.

Below 5 GeV centre of mass energy, intra-nuclear transport models are in preparation. For cascade type models a re-write of HETC [82] as well as INUCL [83] is in preparation, as well as an implementation of a time-like cascade [84]. For quantum molecular dynamics models, an enhanced version of UrQMD [85] is being written.

Note that the cascade models are based on average geometrical descriptions of the nuclear medium, and take effects like Pauli-blocking, coherence length and formation times into account in an effective manner. Scattering is done as in the QMD model, with the possibility of using identical scattering implementations. The QMD models calculate the interaction Hamiltonian from two- and three-body interactions of all particles in the system, and solve the Newtonian equations of motion with this time-dependent Hamiltonian numerically. Scattering is done using smeared resonance cross-sections, taking Pauli's principle into account by investigating local phase-space. The approach promises to give all correlations in the final state correctly, and has no significant limitations in its applicability at low energies. It is very CPU expensive.

At energies below O(100 MeV) we provide the possibility of using exciton based pre-compound models [86] to describe the energy and angular distributions of the fast particles, and of softening the otherwise too steep behaviour of the quasi-elastic peaks. In this area one model is released, and an alternative is in preparation.

The last phase of a nuclear interaction is nuclear evaporation. In order to model the behaviour of excited, thermalised nuclei, variants of the classical Weisskopf-Ewing model [87] are used. Specialised improvements such as Fermi's break-up model [88] for light nuclei, and multi-fragmentation [89] for very high excitation energies are employed. Fission [90], and photon evaporation [57] can be treated as competitive channels in the evaporation model.

As an alternative for all nuclear fragmentation models, including evaporation models, the chiral invariant phase space (CHIPS) model [21,22] is under development. It is a quark-level 3-dimensional, SU(3)xSU(3) symmetric event generator for fragmentation of excited hadronic [21] and nuclear [22] systems into hadrons. It is expected to be applied to a wide range of hadron- and lepto-nuclear [23] interactions. This is already released in the toolkit as an event generator for the reactions of pion capture at rest, anti-proton capture at rest, as a fragmentation model for photo- and electronuclear reactions, and as nuclear fragmentation model for residual nuclei absorbing the soft part of the Quark-Gluon String. The kaon capture at rest, decays of hyper-nuclei, hadron-nuclear and in particular antiproton-nuclear interactions at low energies, Nuclear Giant resonance fragmentation are under development.

A theoretical model for coherent elastic scattering was added recently, using the Glauber model and a two Gaussian form for the nuclear density. This expression for the density allows one to write the amplitudes in analytic form. Note that this assumption works only since the nucleus absorb hadrons very strongly at small impact parameters, and the model describes nuclear boundaries well.

For lepton nuclear interactions, muon nuclear interactions are provided. Here the leptonic vertex is calculated from the standard model, and the hadronic vertex is simulated using a suitable set of models from the above described. Neutrino nuclear interactions will be added in due course.

Modelling summary

Already when taking only the view of the large HEP experiments, it has become evident that all modeling techniques — data driven, parameterisation driven, and theory driven — are necessary to satisfy the needs for hadronic simulation in an optimal manner. Data driven modeling is known to provide the best, if not only, approach to low energy neutron transport for radiation studies in large detectors. Parameterisation driven modeling has proven to allow for tuning of the hadronic shower for particle energies accessible to test-beam studies, and is widely used for calorimeter simulation. Theory driven modeling is the approach that promises safe extrapolation of results toward energies beyond the test-beam region, and allows for maximal extendibility and customisability of the underlying physics.

The use of state of the art software technology is the key that allows for distributed development of the physics base of a tool-kit for simulation of hadronic physics in the GEANT4 context. It allows the work of many experts in the field to be combined in a coherent manner, and offers the user the possibility of unifying his/her knowledge in a single executable program in a manner that is deemed optimal for a particular problem. This is a completely new situation. In a very short time it has lead to an unexpectedly wide range of modelling possibilities in GEANT4, and an unprecedented ease of flexibility of usage of models and cross-sections.

5.9.3 Sample data driven models

As an example of a data driven model, we briefly describe the models for neutron and proton induced isotope production. These models are running in parasitic mode to the transport models, and can be used in conjunction with any set of models for final state production and total cross-sections. They have been written to allow for detailed isotope production studies, covering most of the spallation neutron and proton energy spectrum. They are based on evaluated nucleon scattering data for kinetic energies below 20 MeV, and a combination of evaluated data and extrapolations at energies up to 100 MeV. The upper limit of applicability of the model is 100 MeV nucleon kinetic energy.

The evaluated data libraries that are the basis of the Geant4 neutron transport and activation data library are Brond-2.1 [91], CENDL2.2 [92], EFF-3 [93], ENDF/B-VI.0 [73], ENDF/B-VI.1, ENDF/B-VI.5, FENDL/E2.0 [75], JEF2.2 [94], JENDL-FF [74], JENDL-3.1, JENDL-3.2, and MENDL-2 [76]. Our selection was guided in large part by the FENDL2.0 selection. Additions to and small modifications of this selection were possible due to the structure of the Geant4 neutron transport code and the use of the file system to maximise the flexibility of the data formats. The inclusion of the MENDL data sets is fundamental for these models.

Figure 9 shows an example of the simulated cross-section in comparison to evaluated data from the MENDL collection, using 10^6 events at each energy. A systematic error of 15% was added to the simulation results to take the error in the extrapolation of the total cross-sections into account. For a complete description and more comparisons, see [95].

5.9.4 Sample parameterised models

Parameterisation based models have been found to be very powerful in the case of calorimeter simulation. Without giving a detailed description of these models, we want to illustrate the predictive power for the case of high energy models in Figure 10 for production of neutral pions in interactions of kaons and pions with gold and aluminum.

5.9.5 Sample theory driven models

Given that the chiral invariant phase-space decay model CHIPS is a rather new development and is developed only within Geant4, we choose this as an example for a theory based model. CHIPS is a quark-level 3-dimensional event generator for fragmentation of excited hadronic systems into hadrons. An important feature is the universal thermodynamic approach to different types of excited hadronic systems including nucleon excitations, hadron systems produced in e^+e^- interactions, high energy nuclear excitations, etc. Exclusive event generation, which models hadron production conserving energy, momentum, and charge, generally results in a good description of particle multiplicities and spectra in multi-hadron fragmentation processes. To illustrate the predictive possibilities of this ansatz, we show a comparison between CHIPS predictions and measurement in the case of proton anti-proton annihilation in Figure 11. For details of the model see [21–23].

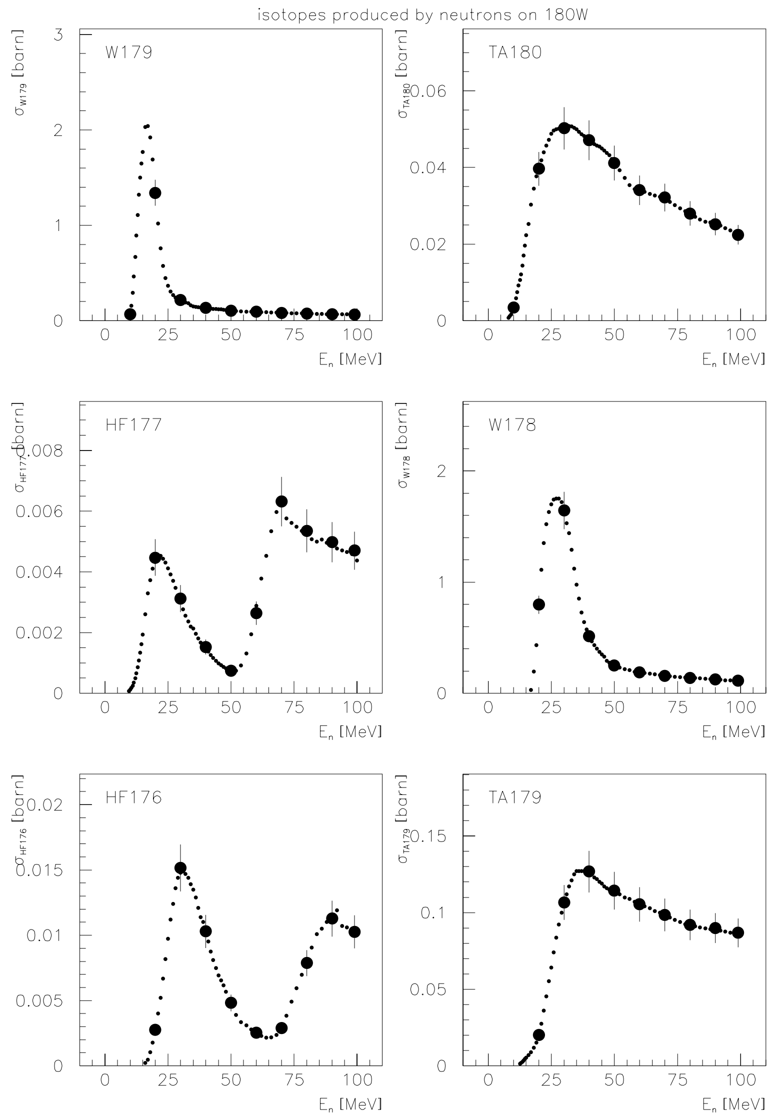


Fig. 9. Isotope production cross-sections for neutron induced production of important isotopes as simulated using the isotope-production code in Geant4. Large points are simulation results, small points are evaluated data from the MENDL2 data library.

5.10 Optical processes

Geant4 is an ideal framework for modelling the optics of scintillation and Čerenkov detectors and their associated light guides. This is founded in its unique capacity of commencing the simulation with the propagation of a charged particle and completing it with the detection of the ensuing optical photons on photo sensitive areas, all within the same event loop.

A photon is called *optical* when its wavelength is much greater than the typical atomic spacing. In Geant4 the concept of *optical photons* is a class of particles

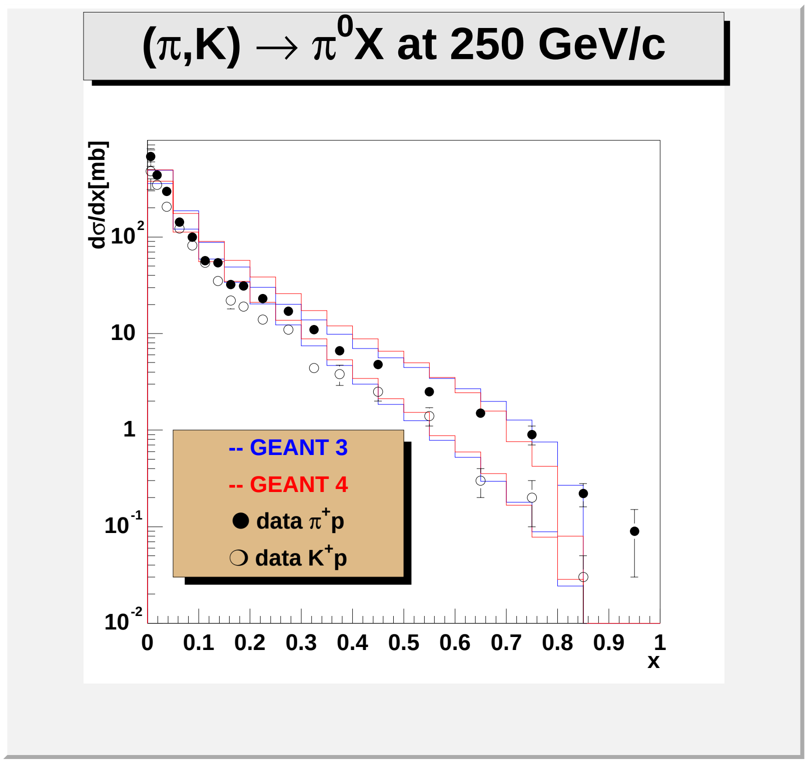


Fig. 10. Comparison of production cross-sections of neutral pions in kaon and pion induced reactions with measurement.

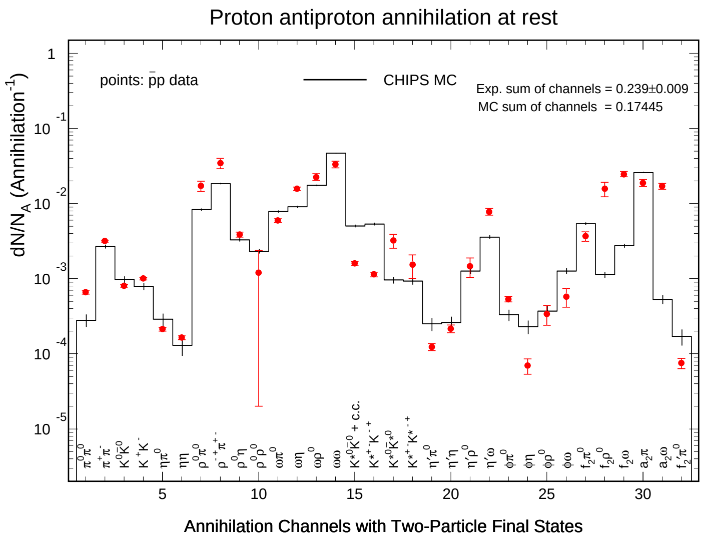


Fig. 11. Comparison of the branchings in two particle final states in proton anti-proton annihilation with the predictions of CHIPS.

detached from their higher energy *gamma* cousins. This implementation allows processes to be associated to them arising from the wave like property of electromagnetic radiation.

The catalog of processes at optical wavelengths includes refraction and reflection at medium boundaries, bulk absorption and Rayleigh scattering. The optical properties of the medium which are key to the implementation of these types of processes are stored as entries in a properties table linked to the material in question. They can be expressed as a function of the photon's wavelength.

5.10.1 Čerenkov process

The flux, spectrum, polarization and emission of this radiation follow well known formulae. The time and position of Čerenkov photon emission are calculated from quantities known at the beginning of the charged particle's step, which is assumed to be rectilinear even in the presence of a magnetic field. The need to suspend the primary charged particle track arises in the production of Čerenkov photons because the number of such photons generated during the length of a typical step, as defined by energy loss or multiple scattering, is often very large. Hence, the tracking of the Čerenkov radiating particle can be suspended by putting it on its own stack of generated secondaries, so that its proteges are tracked in turn before it is revived and transported further.

5.10.2 Scintillation

Every scintillation material has a characteristic light yield and an intrinsic resolution, which generally broadens the statistical distribution, due to impurities; typical examples are doped crystals like NaI(Tl) and CsI(Tl). The average yield can have a non-linear dependence on the local energy deposition. Scintillating materials also have emission time spectra with one or more exponential decay time constants, with each decay component having its intrinsic photon emission spectrum. These empirical parameters are particular to each material and must be supplied by the user. GEANT4 provides a framework in which this can be done effectively. A Poisson distributed number of photons is generated according to the energy lost during the step. The photons originate evenly along the track segment and are emitted isotropically with a random linear polarization.

5.10.3 Absorption and Rayleigh scattering

The implementation of optical photon bulk absorption is trivial in that the process merely *kills* the particle. The procedure requires the user to fill the relevant material property table with empirical data for the absorption length. The differential cross section in Rayleigh scattering is proportional to the square of the cosine of the angle between the new photon's polarization vector and that of the original photon. The Rayleigh scattering process samples this angle accordingly and then calculates the scattered photon's new direction by requiring that it be perpendicular to the photon's new polarization in such a way that the final direction, initial and final polarization are all in one plane.

5.10.4 Reflection and refraction

Before explaining the simulation of reflection and refraction at medium boundaries, it is necessary to introduce the concept of optical surface in GEANT4.

The optical boundary process design relies heavily on the concept of *surfaces*. The information is split into two classes. One class in the materials category keeps information about the physical properties of the surface itself, and a second class in the geometry category holds pointers to the relevant physical or logical volumes involved and has an association with the physical properties class. Objects of the second type are stored in a related table and can be retrieved by either specifying the logical volume entirely surrounded by this surface or the pair of physical volumes touching at the surface. The former is called *skin surface*, while the latter is referred to as a *border surface*. The first type of surface is useful in situations where a volume is coded with a reflector and is placed into many different mother volumes. A limitation is that the skin surface can only have one and the same optical property for all of the enclosed volume's sides. The *border surface* is an ordered pair of physical volumes, so the user can choose different optical properties for photons arriving from different sides of the same interface.

The physical surface object also specifies which model the boundary process should use to simulate interactions with that surface. In addition, the physical surface can have a material property table all its own. The usage of this table allows all specular constants to be wavelength dependent. In case the surface is painted, wrapped or has a cladding, the table may include the thin layer's index of refraction. This allows the simulation of boundary effects both at the intersection between the medium and the surface layer, as well as at the far side of the thin layer, all within the process itself and without invoking the GEANT4 navigator. Combinations of surface finish properties, such as polished or ground and front painted or back painted, enumerate the different situations.

When a photon arrives at a medium boundary its passage depends on the nature of the two materials that join at that boundary. The user can specify the medium boundary as between two dielectric materials, one dielectric and a metal, or one dielectric and a *black* medium. In the case of two dielectric materials, the photon can be totally internal reflected, refracted or reflected, depending on the photon's wavelength, angle of incidence, (linear) polarization and the refractive indices on both sides of the boundary. The photon can be absorbed by the metal or reflected back into the dielectric. If the photon is absorbed it can be detected according to the photo-electron efficiency of the metal. Reflection and transmission probabilities are sensitive to the state of linear polarization.

The user has two choices in Geant4 for modelling a realistic surface, either the UNIFIED model [96] of the DETECT [97] program, or the original GEANT3 implementation via the GLISUR methods flag. Using GLISUR, the roughness of a surface is specified by a single parameter, while with the UNIFIED model a more comprehensive description is possible that deals with all aspects of surface finish and reflector coating.

5.11 Transition radiation

Transition radiation emitted by a relativistic charged particle crossing an interface between two materials with different dielectric properties is implemented as a boundary process. (An alternative implementation is described in Section 6.1.) The class `G4ForwardXrayTR` is responsible for the description of X-ray transition radiation (XTR) photon generation from one interface between two different materials. One of those material states should be gas or vacuum. The base class `G4TransitionRadiation` consists of methods and data members which can be used in both optical and X-ray transition radiations.

6 Additional capabilities

The real power of the object-oriented approach lies in the ability to extend the basic functionality either by implementing classes derived from the kernel base classes or writing “plug-ins” which use Geant4.

An example of the former, in fact, is the whole of the physics processes; the kernel is written for the generic process defined by the abstract interface and any process which conforms to this interface can be used. Below we describe another extension, namely to “parameterised processes” or “fast simulation processes”, which allows the user to define what happens when a particular particle enters a particular volume. All this can happen without modifying the kernel.

“Plug-ins” simply use Geant4. In this category are user interfaces, visualization and analysis, described in Section 7 and persistency, described below. Like processes, these are distributed with Geant4 but, if the user wishes, can be replaced. This might happen in a large project that has already defined its software framework and already made decisions about such functions. It is a relative straightforward matter to integrate GEANT4 into an existing software framework.

6.1 Parameterisation for fast simulation

Fast simulation or parameterisation allows one to take over the tracking and implement, for example, a fast algorithm of detector response. The typical use case is shower parameterisation where the several thousand steps per GeV computed in the detailed simulation are replaced by a few tens of energy deposits [98,99]. Very fast simulation, in which the tracking is intercepted to produce reconstructed-like objects, is also useful [100].

Parameterisation characteristics which have been identified as impacting the design are: parameterisations are generally experiment dependent; they take place in an envelope, which is typically the mother volume of a sub-detector, for example a calorimeter; they apply to specific particles types. Parameterisation may also be required not to trigger in complicated regions, like module overlaps of a calorimeter, and may require kinematic criteria, like a sufficiently high energy, to be valid.

The above requirements have been expressed in the following way. Parameterisations take place at tracking time so that access to kinematic and geometrical information is natural. Parameterisations compete with the normal tracking; at the beginning of each step starting inside an envelope, parameterisations are given a chance to issue a trigger. If it does so, it is applied, otherwise the tracking proceeds with a normal step.

Parameterisations are designed as models, applicable to specific particle types and defining a trigger method. GEANT4 provides an abstract interface only, since parameterisations are usually experiment dependent. This interface, `G4-VFastSimulationModel`, defines three pure virtual methods `IsApplicable`, `ModelTrigger` and `DoIt`. The `IsApplicable` method allows the user to ascertain for which particle types the model is valid. `ModelTrigger` defines the trigger and is the place where the user makes the decision not to trigger in complicated regions or in non suited kinematic ranges. `DoIt` is the user parameterisation code proper, invoked if the model has previously issued a trigger. The `DoIt` signature is general enough to change the state of the current particle and to create secondaries.

The notion of "envelope" has been explicitly introduced: envelopes are the geometrical regions where (and only where) parameterisations may trigger. Envelopes can be defined in two ways. They can be volumes of the geometry for tracking or they can be volumes, called "ghost volumes", independent of that geometry. The former are, perhaps, more usual while the latter allows a general envelope definition.

In this latter case, envelopes are volumes that are "overlayed" on the normal tracking geometry and an extra navigation method is performed to check whether the current particle is inside this volume. This allows one in particular to create envelopes for geometries produced by CAD systems, where no hierarchical structure exists. Those ghost volumes are, in addition, particle type sensitive, allowing the creation of different envelopes for different particle types. For example, defining an envelope solely for pions enclosing both electromagnetic and hadronic calorimeters is possible.

In practice, envelopes are `G4LogicalVolume` objects. Parameterisation models are bound to this volume through a `G4FastSimulationManager` object, which gathers and messages those models. The pointer of this manager is recursively propagated to the daughters of the `G4LogicalVolume`, allowing fast checking of the presence of such a manager at tracking time.

The interface between the parameterisation and the tracking is provided by a `G4VProcess`, the `G4FastSimulationManagerProcess` (`G4FMP`), which checks for the presence of a `G4FastSimulationManager` object and messages it if any. Modification of the current track and information of possible secondaries created by a parameterisation model are communicated to the tracking through a `G4VParticleChange`, like any `G4VProcess`. The `G4FMP` also provides the extra navigation in case ghost volumes are used. The `G4FMP` makes use of the "exclusive" signal in the case that a model triggers to tell the tracking that it is to be the only process applied in the current step.

As a concrete example for the use of fast simulation we give X-ray transition radiation (XTR) generation from radiators [101]. It is described by a family of classes inheriting from `G4VFastSimulationModel`. (It has also been implemented as a boundary process — see Section 5.11.) The base class `G4-XrayTRmodel` is responsible for the creation of tables with integral energy and angular distributions of XTR photons. It has also the `DoIt` function providing XTR photon generation and moving the incident particle through the XTR radiator. Particular models like `G4IrregularXrayTRmodel` realise the pure virtual function `GetStackFactor`. The latter calculates the response of the XTR radiator.

Figure 12 shows the scheme of working of the `G4XrayTRmodel::DoIt` function. An incident charged particle with a Lorentz factor $\gamma \geqslant 100$ enters the logical volume *G4Envelope* at the point p_1 and exits at p_2. It moves along the direction given by the unit vector $\vec{v}$. XTR photons are generated randomly along the particle trajectory (point g_1) inside *G4Envelope* with energies and polar angles relative to $\vec{v}$ randomly selected from the corresponding tables of the integral energy and angular distributions. Each XTR photon then is moved to the

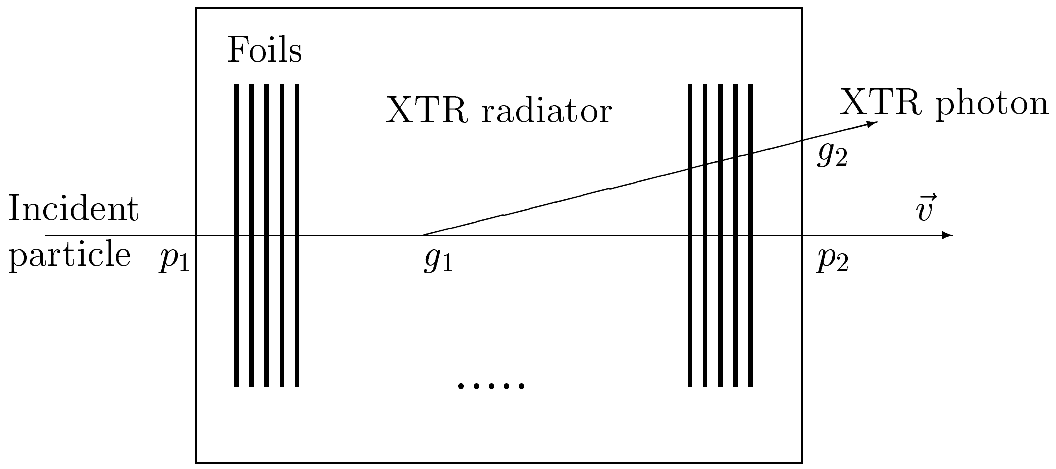


Fig. 12. Illustrating the working of the `G4XrayTRmodel::DoIt` function.

border of *G4Envelope* (g_2). Finally the sum of the XTR photon energies is subtracted from the kinetic energy of the incident particle.

6.2 Event biasing

Variance reduction techniques are an important aspect of most Monte Carlo calculations and allow the user to tune the simulation to the part of the problem space (particle species, energy, position, etc.) most relevant to his/her application [102]. In GEANT4 facilities exist to allow the user to modify the statistical weight associated with each `G4Track` object so that, for example, the user can modify the probability of interaction processes (currently through user-written code) to increase the sampling of high-energy secondary particles, which will have correspondingly lower track weights. More general facilities to allow a user to bias the simulation conditions, without the need for significant code development, are currently being developed. However, there are already two classes that allow biasing schemes to be applied.

When simulating radioactive decay following energetic particle interactions, it is often necessary to treat the decays of a range of nuclide species over many generations. Furthermore, results are often required only for decays within particular "time-windows" that are much shorter than the times between nuclide production and observation. To deal with the former, the GEANT4 radioactive decay processes applies recursive formulae based on the work of Bateman [103] and Truscott [104] to allow determination of the decay rates for all nuclear generations. This is used in combination with the variance reduction modes

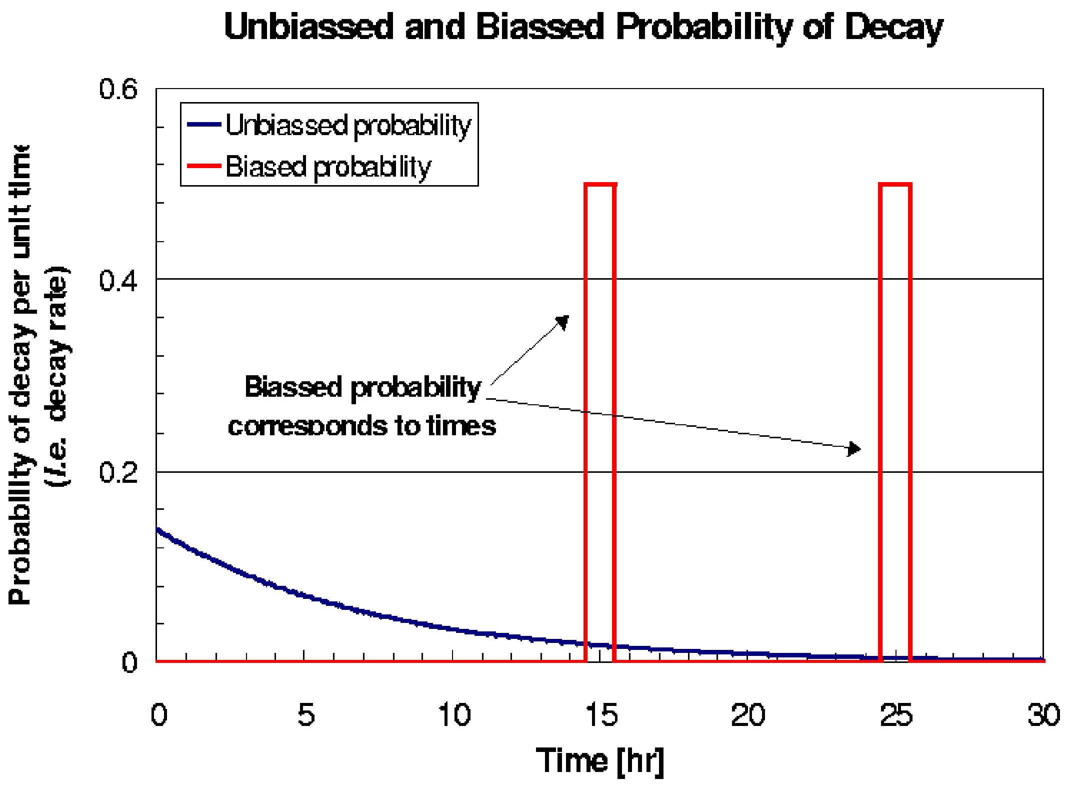


Fig. 13. Illustration of Decay Probability Biasing.

that can be invoked in the class `G4RadioativeDecay` preferentially to sample the times of the decay according to the times of observation (as illustrated by Figure 13). In the extreme case, all radionuclides and their progeny can be forced to decay at a user-defined observation time. An additional feature of this variance reduction mode in `G4RadioactiveDecay` is that the user can infer the effects of a time-varying source (e.g. if assessing radiation effects on a satellite from a solar particle event) from each radionucleus created in the simulation at time $t = 0$. This is achieved by convolving the decay rate over a user-defined source time profile. (Clearly performing a similar calculation using an analogue Monte Carlo approach to sample source particles as a function of time would be very inefficient in comparison.)

Other variance reduction schemes within `G4RadioactiveDecay` include:

- splitting of radionuclides prior to decay, which can be used to increase the sampling of decay radioactive decay products;
- re-biasing of the decay branches, so that there is increased sampling of low-probability branches that may, because of particle energy or species, have a more important effect on the observation.

The class `G4GeneralParticleSource` also provides facilities for the user to bias the source particle distribution in energy, in the x-, y- or z-direction for the point at which it is created, or in angular direction. Here the user has only to provide the desired sampling distribution in the form of a cumulative probability distribution histogram, and `G4GeneralParticleSource` recalculates the weights of the biased source particles sampled from the distribution. This feature permits, for example, more particles to be sampled closer to the volumes of the simulated geometry where greater sensitivity to radiation effects is expected, or increased sampling of the high-energy portion of a cosmic ray spectrum, which can produce more secondary particles.

6.3 Persistency

The *persistency* category provides an interface for storing and retrieving run, event, hits, digits and geometry information in and from ODMG-compliant object databases so that users may perform post-simulation analysis in separate processes. The object persistency has been achieved by using the ODMG class description and HepODBMS [105].

It is a functional requirement that the kernel part of Geant4 must be able to run with and without HepODBMS and the related commercial packages. This category can be built optionally. Users must set an environment switch before the installation of the toolkit to use this category. This requirement leads to a design decision that normal kernel objects (transient objects) must have corresponding persistent objects to perform a deep copy of data members. The ODMG-compliant class description allows one to one mapping of the association between complicated objects such as geometry.

A persistent class must inherit the persistency characteristics from the class `HepPersObj` in HepODBMS. Data members of the persistent class are updated and committed during the database transaction methods defined in `HepDbApplication`. Schema information is extracted from the class header files and compiled as meta-data with a preprocessor. Currently a commercial object database package Objectivity/DB is supported by HepODBMS.

To use persistency, users must instantiate a singleton object of class `G4PersistencyManager`. This object is messaged by the Geant4 run manager to trigger the actual deep copy of the objects related to run, event and geometry. Location of the database and the transactions to the database are specified through the user interface class `G4PersistencyMessenger` and the information is transferred to an object of class `G4TransactionManager`, which handles the database transaction messages using `HepDbApplication`. The deep copy operations are performed during these transactions.

When a pointer of a transient `G4Event` object is handed to `G4Persistency-Manager`, it is handed on to a `G4PersistentEventMan` object which then constructs a persistent object of class `G4PEvent` with data member which are values of `G4Event`. A constructor of `G4PEvent` then constructs `G4PPrimaryVertex`, and makes associations to the primary vertex, hit collections and digit collections if they exist. `G4PPrimaryVertex` then constructs `G4PPrimaryParticle` objects. Similar deep copy transactions occur for run and geometry when triggered by `G4RunManager`. The relation of the persistent and the transient objects is reversed in the retrieve transaction.

Hit and digit classes and their collections belong to the user domain and actual implementation of these classes differ from detector to detector. GEANT4 persistency category provides persistent abstract base classes so that users can directly store and retrieve their hits and digits collections. In a user implementation, hit and hits collection inherits persistency from `G4PVHit` and `G4PVHitsCollection` respectively. Actual storing and retrieving of the data members is triggered in the methods of `G4TransactionManager`. In this case, the user must explicitly take control of database transactions for their user defined persistent classes by specifying a transaction type parameter in `G4-PersistencyManager` methods.

7 Interactivity and visualisation

Interactivity and visualisation span three related categories, i.e., *intercoms*, *interfaces*, and *visualisation* categories. At the lowest level resides *intercoms*, which provides, amongst other things (see Section 2.2), command definition and interpretation tools. User interaction is realised through the concept of a "session" and graphical and non-graphical concrete sessions are available in the *interfaces* category.

Visualisation is a high level category which uses *intercoms* and — if interactive graphical tools are shared, such as the X Windows Toolkit (Unix) or Microsoft Windows — also uses *interfaces*, where the windows event handlers are coded. Drivers for several graphics systems are offered and can be instantiated in parallel.

Below, we describe these categories in turn. We also describe how the visual debugging of detector geometry models can be realised in GEANT4.

7.1 User interfaces

The design of GEANT4 (graphical) user interfaces was influenced by two considerations: the categories of users and the phases of user actions.

Three categories of users were envisioned:

(1) *End user* who runs a Geant4 application by setting run-time parameters and executing commands with the (graphical) user interfaces.
(2) *Application programmer* who creates application programs specific to his/her simulation. He/she may wish to define specialised commands and sets of associated parameters. Available commands may vary from one application to another.
(3) *Framework provider* who is a Geant4 developer.

This leads us to separating the creation of (graphical) user interfaces from the creation of commands.

During the execution of an application, two phases of user actions arise, namely for initialisation of simulation and for control of event generation and processing. The user interaction is different in each phase and this requires that a Geant4 application is a state-machine and that its available commands and their parameters may vary according to state.

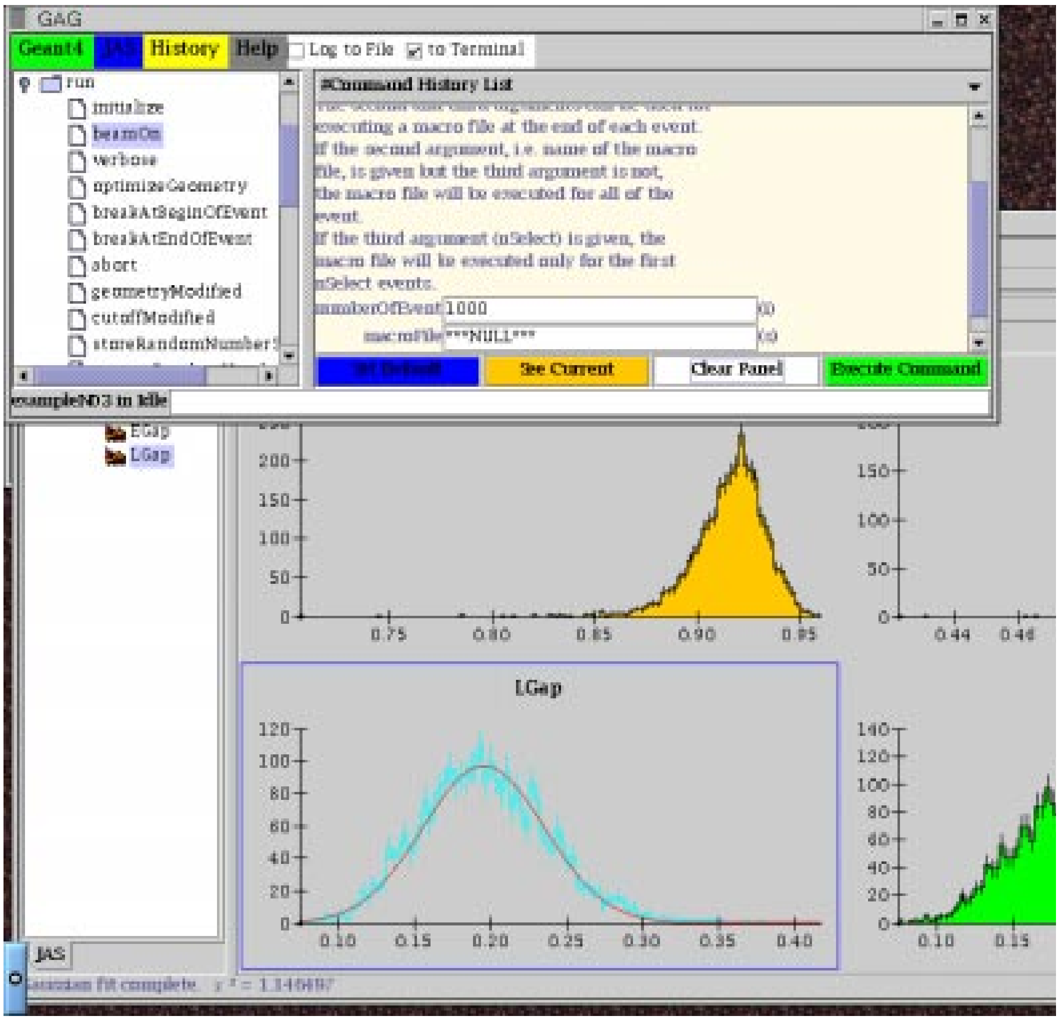
It was our design choice to have the *intercoms* category separate from the user *interfaces* category. The *intercoms* category implements an expandable command interpreter which is the key mechanism in Geant4 for realising customisable and state-dependent user interactions with all categories without being perturbed by the dependencies among classes.

The capturing of commands is handled by a C++ abstract class `G4UIsession` of the *intercoms* category. Various concrete implementations of the command capturer are contained in the [user] *interfaces* category. Taking into account the rapid evolution of graphical user interface (GUI) technology and consequent dependence on external facilities, it was decided to offer plural and extensible GUI's. Application programmers and framework providers, however, are asked only to know how to register their commands and parameters appropriate to their problem domain; no knowledge of GUI programming is required to allow the application to use them through one of the available GUIs.

7.1.1 Concrete implementations

Various user interface tools like Motif, Tk/tcl, JAVA, etc., have been used to implement the command "capturer". The richness of the Collaboration has permitted different groups to offer various front-ends to the Geant4 command system. We list below the currently available implementations according their main technologies:

- *batch* to read and execute a file containing commands,
- *tcsh-like terminal* for interactive sessions,



```
executing a macro file at the end of each event.
If the second argument, i.e. name of the macro
file, is given but the third argument is not,
the macro file will be executed for all of the
event.
If the third argument (nSelect) is given, the
macro file will be executed only for the first
nSelect events.
numberOfEvent 1000
     macroFile ***NULL***
```

Fig. 14. GAG working with JAS.

- *Xm, Xaw, Win32* variations of the above using a Motif, Athena or Windows widget to retrieve commands, useful if working in conjunction with visualisation drivers that use the Xt library or the WIN32 one,
- *GAG* [106,107], a client/server type adaptive GUI reflecting GEANT4 states, and
- OPACS [108], an OPACS/Wo widget manager implementation.

Figure 14 is a display dump of GAG co-working with JAS (Java Analysis Studio) [109].

7.1.2 Other tools for application programmers

GEANT4 requires the application programmer to create three mandatory classes relating to detector geometry, physics processes and the primary generator. The following tools have been developed to help him/her to create the first two of these classes without memorising the straight-forward but tedious names and methods of relevant classes like materials, solids, particles, etc. These have proved very useful for rapid prototyping of simulation applications [107,110].

- *GGE (*GEANT4 *Geometry Editor)* is a tabular tool written in Java which has a material editor and a volume editor. Complete C++ source codes, ready for compilation, implementing the programmer's geometry are produced from the tables filled by the user. It can be saved in a persistent file for reuse.
- *MGA (Material Generation and Association)* is similar with GGE. It uses CAD's STEP output and associates it with materials to generate C++ source codes.
- *GPE (*GEANT4 *Physics Editor)* has tables for particles and electromagnetic processes. The programmer associates a particle with a process by filling the physics list table. From the table are generated complete C++ source codes including the default cut value.

7.2 Visualisation

GEANT4 visualisation is designed to visualise detector geometry, particle trajectories, tracking steps, hits, texts (character strings), etc., to help users to prepare and execute detector simulation.

There is a wide variety of user requirements on visualisation. For example:

- very quick response to survey successive events;
- high-quality outputs for presentation and documentation;
- impressive special effects for demonstration;
- flexible camera control for debugging detector geometry and physics;
- selection of visualisable objects;
- interactive picking of graphical objects for attribute editing or feedback to the associated data;
- highlighting wrong intersections of physical volumes;
- cooperative working with graphical user interfaces.

It is very difficult to respond to all of these requirements with only one built-in visualiser, so we have designed an abstract interface which supports several complementary graphics systems. Here the term "graphics system" means either an application running as a process independent of GEANT4 or a graphics library to be compiled with GEANT4. A concrete implementation of the interface is called a *visualisation driver*, and this can use a graphics library directly, communicate with an independent process via pipe or socket, or simply write an intermediate file for a separate viewer.

The current distribution of Geant4 contains several established drivers. Those which need external libraries or packages may only be activated if the corresponding external system is installed, and it is the user's responsibility to check this and set environment variables to control the compilation of the drivers. In addition, in principle, the user may extend this list by implementing his/her own driver to the specification of the abstract interface.

The following drivers write intermediate files for a separate viewing. The drivers themselves need no external libraries or packages and are normally built by default into any Geant4 executable.

- DAWNFILE [111]: produces files for the Fukui Renderer, DAWN [112], which is well suited to preparing technical-high-quality PostScript outputs for presentation and/or documentation. Figure 15 shows an example.
- HepRepFile: generates files in the HepRep [113] format, suitable for viewing with several viewers, notably the WIRED [114] event display viewer. As well as various 3D representations of the geometry model and trajectories, etc., the geometry hierarchy can be viewed as a tree structure and used to select visible components.
- RayTracer This driver performs ray-tracing rendering using the tracking algorithms of Geant4, and generates a JPEG file. The driver is, therefore, a useful debugging tool for Geant4 developers. For users, it is useful for generating photo-realistic high-quality images.
- VRMLFILE [111]: generates VRML (versions 1 and 2) files that can be viewed with one of the many attractive VRML viewers now available. These enable one to perform interactive spinning of detectors, flying inside detectors or particle showers, and so on.

The following link directly to external libraries and require activating by the setting of environment variables. These graphics systems establish their own graphical database for fast refreshing and view re-orienting, except for the so-called immediate mode of OpenGL. Free implementations of all these libraries are available.

- OPACS [108]: The OPACS library supports many useful functions such as an event display and picking.
- OpenGL [115]: OpenGL is widely available and is well suited to real-time, fast visualisation. Its immediate mode has no limitation on picture complexity.
- OpenInventor [116]: This driver supports high interactivity, e.g., attribute editing of picked objects, virtual-reality visualisation, and other advanced functions.

The following use the socket mechanism to communicate with viewers running in daemon mode and require activating by the setting of environment variables.

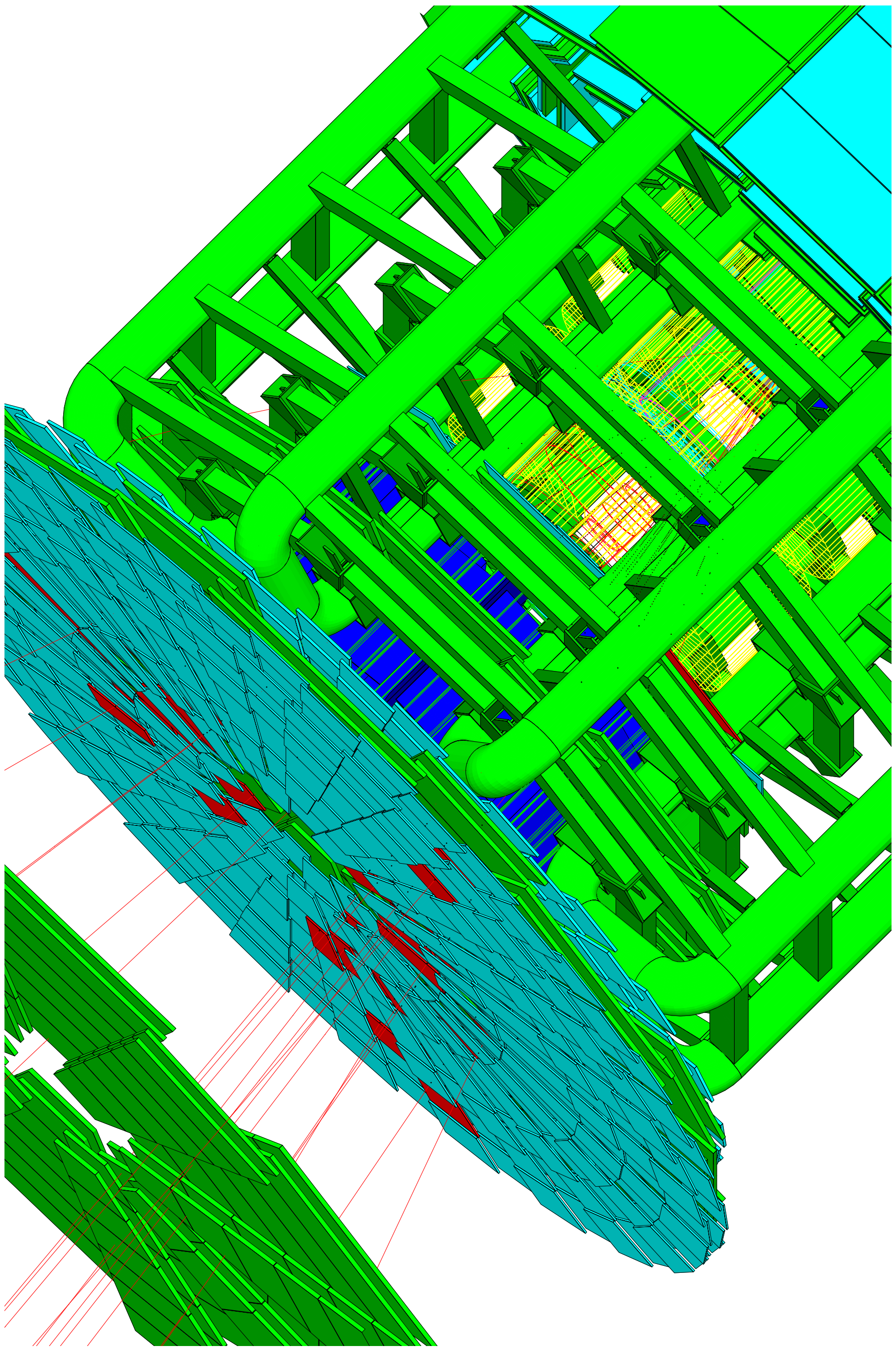


Fig. 15. Visualisation with the DAWNFILE driver.

They have the features of their file-writing equivalents above but have the advantage of remote visualisation.

- DAWN [111]: has the features of DAWNFILE above.
- VRML [111]: has the features of VRMLFILE above.

The following use the visualisation driver mechanism to provide an alternative tree representation of the geometry model. The drivers themselves need no external libraries or packages and are normally built by default into any GEANT4 executable.

- ASCIITree: simply tabulates the geometry model hierarchy on standard output.
- GAGTree: communicates with the GAG user interface (Section 7.1.1), if instantiated, and allows a tree-widget-like viewing of the geometry model hierarchy.

Visualisation procedures are controlled by the visualisation manager described in a user class, say, `MyVisManager` that inherits the class `G4VisManager` defined in the *visualization* category [117]. The visualisation manager accepts a user's requests for visualisation, processes them, and passes the processed requirements through the abstract interface to the currently selected visualisation driver. In this process, the visualisation manager uses various classes for visualisable 3D objects (volumes, lines, texts, markers, etc.) defined in the *graphics_reps* category and the *geometry* category. The visualisation manager also uses utility classes defined in the *visualization/modeling* sub-category to generate 3D data ready for visualisation.

7.3 Data analysis

There are various analysis systems that generate histograms, analyse event data statistically, and so forth. It is possible to plug in many of these to GEANT4. Examples of plugging in analysis systems supporting the AIDA abstract interface [118], e.g., JAS [109], Lizard [119], and OpenScientist [120], are included in the GEANT4 package.

7.4 Geometry verification

The application developer's job includes describing a detector geometry, usually by writing C++ codes. In debugging, the most time-consuming work is the checking of the detector model. Visualisation is indispensable. It is a requirement and assumption of the GEANT4 tracking that volumes do not overlap and that daughter volumes are fully contained within the mother. This means that there should be no intersections of physical volume surfaces. Correct tracking behaviour is not guaranteed if this happens.

In general, the most powerful intersection detection algorithms are provided by CAD systems, treating intersections between the solids in their topological form, and users are encouraged to use such facilities. In principle, the STEP interface enables transfer between Geant4 and CAD systems, but this might not always be practical.

Geant4 visualisation supports a powerful way of visually debugging intersections of physical-volume surfaces. Physical volume surfaces are decomposed into 3D polygons, and intersections of the generated polygons are investigated. If a polygon intersects with another one, physical volumes to which these polygons belong are drawn with a highlight colour (red by default). Figure 16 is a sample visualisation of detector geometry with intersecting physical volumes highlighted. This visual debugging of physical volume surfaces is performed with the DAWNFILE visualisation driver (see 7.2) in cooperation with the debugger application DAVID [121].

Geant4 also provides some facilities for helping in detecting geometry errors, these facilities are also available as run-time commands. A series of linear trajectories are used to calculate the points of intersection with the solids. The requirement of no overlaps and full containment proscribe the ordering of these points along the trajectory. This is an approach which is complementary to that of CAD systems and DAVID. The disadvantage is that small errors may be missed if the linear trajectories do not happen to pass through the problematic region of space. On the other hand, it uses the geometry algorithms built into Geant4 itself and the geometry is sampled where the user is most interested in its validation.

8 Conclusion

The Geant4 toolkit provides a versatile and comprehensive software package for modern simulation applications that involve the interaction and passage of particles through matter. It can handle complex geometries efficiently and compactly, and allows visualization of the geometry and particle tracks through a variety of interfaces. It provides simulation for a wide range of physics processes based on theory, data or parameterisation. These treat, for example, hadronic interactions from thermal energies up to 1 PeV, electromagnetic interactions of charged hadrons, ions, leptons and photons from 250 eV to 1 PeV or more, as well as the production and propagation of optical photons. The implementation of the toolkit in an object-oriented design allows it to be easily extended, where appropriate, to meet the requirements of the user, through class inheritance. In addition, there are a growing range of utilities for visualization and analysis of resulting data. The software itself, which is freely available at source-code level over the Web [1], has been developed in accordance with software engineering standards in order to attain a high quality, reliable product.

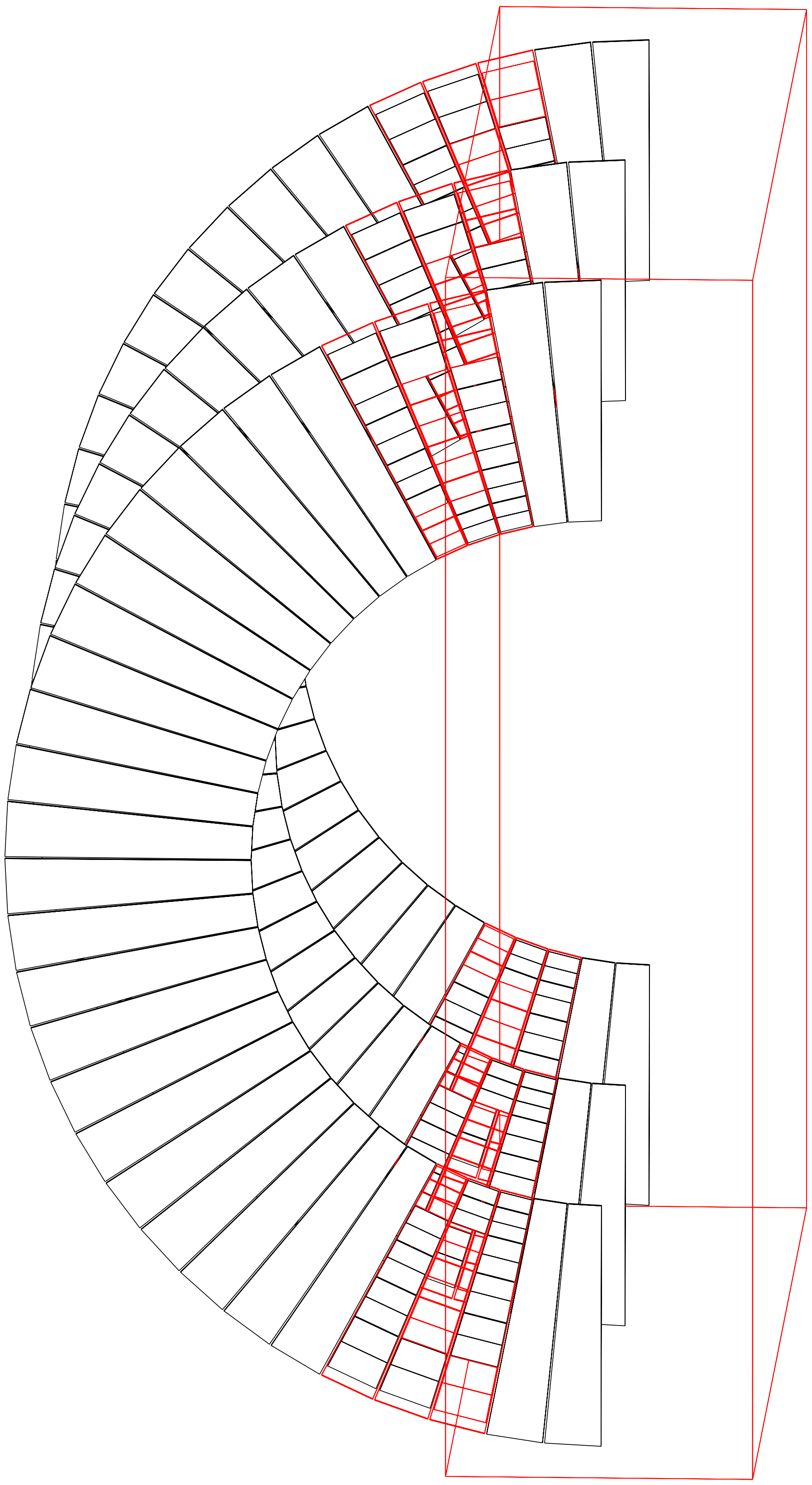


Fig. 16. Highlighting wrong intersections of physical volumes.

Geant4 has been, and continues to be developed and maintained through a Memorandum of Understanding agreed between the many collaborating institutes. This world-wide international collaboration, and the user community, is continuing to expand due to the applicability, accessibility and versatility of this toolkit, with current applications ranging from medical physics to high-energy astrophysics, as well as of course particle physics and accelerator design. The reader is referred to user forums and separate papers, many associated with joint projects and experiments, for more detailed discussions, development and validation of the Geant4 toolkit.

Acknowledgements

The support for this work has come directly or indirectly from many sources. Primarily, we acknowledge the member institutes: European Organization for Nuclear Research (CERN), European Space Agency, Helsinki Institute of Physics (HIP), Inst. für Theoretische Physik, Johann Wolfgang Goethe Universität, Frankfurt, Jefferson Lab, Karolinska Institutet, KEK (Japan), Stanford Linear Accelerator Center (SLAC), TERA Foundation, TRIUMF (Canada) and Universitat de Barcelona and the Collaborating National Organisations: IN2P3 (France), Istituto Nazionale di Fisica Nucleare (INFN), Italy, Lebedev Institute and UK Particle Physics and Astronomy Research Council (PPARC). From those we particularly recognise the contribution of CERN in terms of manpower (both local and visiting associateships) and of equipment through its IT Division (Simulation Group), with the support of the Large Hadron Collider Board during both the R & D phase and the current production phase, and of SLAC in terms of manpower through Computer Services. Without this direct support, this project would not have come to fruition.

But perhaps, when all is added up, most of the effort has come from the generous support of experiment collaborations and universities that have allowed and encouraged their members to contribute to Geant4 in a spirit of world-wide collaboration to everyone's mutual benefit. Experiment collaborations who are formally members of Geant4 are: ATLAS, BaBar, CMS, HARP and LHCb. There have been significant contributions from non-member experiments too. The universities are too numerous to mention.

Satoshi Tanaka would like to acknowledge the support of Prof. Hiroaki Yamamoto, Department of Information Science, Fukui University.

Marc Verderi would like to acknowledge many useful discussions with Claude Charlot.

References

[1] GEANT4 Web page: http://cern.ch/geant4.

[2] K. Amako et al., Proceedings of CHEP94, San Francisco, CA, USA, LBL-35822 CONF-940492.

[3] R. Brun, F. Bruyant, A.C. McPherson, M. Maire, P. Zanarini, CERN Data Handling Division, DD/EE/84–1, 1987.
Also, GEANT - Detector Description and Simulation Tool, CERN Program Library Long Write-up W5013, IT Division, API group, CERN.

[4] A. Dellacqua et. al, GEANT4 an object-oriented toolkit for simulation in HEP, CERN/DRDC/94-29 DRDC/P58, 1994.

[5] S. Giani et al., GEANT4 An Object-Oriented toolkit for Simulation in HEP, CERN/LHCC 98-44, 1998.

[6] At URL http://cern.ch/geant4/organisation/MOU.html.

[7] All user documents are found at URL
http://cern.ch/geant4/G4UsersDocuments/Overview/html.

[8] A user forum based on the hypernews system can be found at
http://geant4-hn.slac.stanford.edu:5090/Geant4-HyperNews/index.

[9] RD44, GEANT4 *User Requirements Document*, CERN, 1998.

[10] CLHEP - A Class Library for High Energy Physics,
http://wwwinfo.cern.ch/asd/lhc++/clhep.

[11] P. Truscott, F. Lei and C. Ferguson, The General Particle Source Module,
http://www.space.dera.gov.uk/space_env/gspm.html.

[12] M. Asai, ODBMS for LHC detector simulation,
Comp. Phys. Comm. 110 (1998) 125.

[13] ISO 10303-203. Application protocol: Configuration Controlled Design, Industrial automation systems and integration - Product data representation and exchange, ISO TC 184/SC4, 1994.

[14] J. Sulkimo and J. Vuoskoski, GEREP, a Boundary Representation Modeller proposal for GEANT4 , IT Division Internal Report, CERN.

[15] M. Asai et al., Design of tracking and generic processes in Geant4, Proceedings of the MC2000 Conference, Lisbon (2000).

[16] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns, Addison-Wesley, 1995.

[17] J. P. Wellisch, Hadronic shower models in GEANT4 - the frameworks, Comp. Phys. Comm. 140 (2001) 65-75.

[18] A. B. Kaidalov, K. A. Ter-Martirosyan, Phys. Lett. B117 (1982) 247.

[19] Data Formats and Procedures for the Evaluated Nuclear Data File, National Nuclear Data Center, Brookhaven National Laboratory, Upton, NY, USA.

[20] H. Stocker et al., Nucl. Phys. A538 (1992) 53.

[21] P. V. Degtyarenko, M. V. Kosov and H. P. Wellisch, Chiral invariant phase space event generator. I: Nucleon anti-nucleon annihilation at rest, Eur. Phys. J. A 8 (2000) 217.

[22] P. V. Degtyarenko, M. V. Kossov and H. P. Wellisch, Chiral invariant phase space event generator. II: Nuclear pion capture at rest and photo-nuclear reactions below the Delta(3,3) resonance, Eur. Phys. J. A 9 (2000) 411.

[23] P. V. Degtyarenko, M. V. Kossov and H. P. Wellisch, Chiral invariant phase space event generator. III: Modeling of real and virtual photon interactions with nuclei below pion production threshold, Eur. Phys. J. A 9 (2000) 421.

[24] G. Cosmo, Software Process Improvement in Geant4 , Geant4 Internal Status Report, January 2001.

[25] D. A. Reo et al., Measuring Software Process Improvement: there's more to it than just measuring processes, ESI - FESMA 99, September 1999.

[26] G. Booch, Object-Oriented Analysis and Design with Applications, The Benjamin/Cummings Publishing Co., 1994, ISBN 0-805-35340-2.

[27] At URL `http://cern.ch/geant4/collaboration/coding_guidelines.html`.

[28] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lornsen, Object-Oriented Modeling and Design, Prentice-Hall International Editions, ISBN 0-13-630054.

[29] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, P. Jeremes Object-Oriented Development: The Fusion Method, Prentice Hall International Edition 1994, ISBN 0-13-101040-9.

[30] ESA PSS-05-0 Issue2, ESA Software Engineering Standards, European Space Agency,

[31] ISO/IEC Joint Technical Committee 1 (JTC1), ISO/IEC DTR 15504 Software Process Assessment.

[32] ISO/IEC Joint Technical Committee 1 (JTC1), ISO/IEC DTR 15504-5 Part 5: An Assessment Model and Indicator Guidance.

[33] CVS Web page: `http://www.cvshome.org`.

[34] AFS is an acronym for the Andrew File System, developed at Carnegie-Mellon University, Pittsburgh, under a sponsorship from IBM. Today AFS is marketed by IBM Transarc Lab (`http://www.transarc.ibm.com/Product/EFS`)

[35] At URL `http://www.gnu.org/software/make/make.html`.

[36] At URL
`http://cern.ch/geant4/working_groups/testing/testing.html`.

[37] At URL
`http://cern.ch/geant4/working_groups/testing/`
`tag_release_policy.html`.

[38] At URL `http://wwwinfo.cern.ch/asd/cgi-bin/geant4/problemreport`.

[39] Bonsai Web page: `http://www.mozilla.org/bonsai.html`.

[40] Tinderbox Web page: `http://www.mozilla.org/tinderbox.html`.

[41] LXR Web page: `http://lxr.linux.no`.

[42] Bugzilla Web page: `http://bugzilla.mozilla.org`.

[43] P. Kent, Minimising Precision Problems in GEANT4 Geometry, GEANT4 working note, April 1995.

[44] P. Kent, Pure Tracking and Geometry in GEANT4 , GEANT4 working note, April 1995,

[45] V. Bargmann, L. Michel, and V.L. Telegdi, Phys. Rev. Letters 2, 435 (1959) and J.D. Jackson, Classical Electrodynamics, 2nd Edition, John Wiley & Sons, New York, p. 559 ff.

[46] Review of Particle Physics, The European Physical Journal C, 15 (2000).

[47] A "gallery" of results comparing GEANT4 with GEANT3 and experimental data can be found at URL
`http://cern.ch/geant4/reports/gallery/electromagnetic`.

[48] E. Daly et al., Space Applications of the GEANT4 Simulation Toolkit, Proceedings of the MC2000 Conference, Lisbon (2000).

[49] S. Chauvie et al., Medical Applications of the GEANT4 Simulation Toolkit, Proceedings of the MC2000 Conference, Lisbon (2000).

[50] J. Apostolakis et al., CERN-OPEN-99-299 (1999)

[51] J. Apostolakis, S. Giani, V. Grichine et al., Nucl. Instr. Meth. A453 (200) 597.

[52] J. Vincour and P. Bem, Nucl. Instr. Meth. 148 (1978) 399.

[53] J. H. Crannel, Phys. Rev. 184 (1969) 2.

[54] Energy loss in thin layers in GEANT, K.M. Lassila-Perini, L. Urba'n, Nucl. Instr. Meth. A362 (1995) 416.

[55] M.V. Kossov, Approximation of photonuclear interaction cross-sections, submitted to EPJA, 2002.

[56] R. Engel, J. Ranft, S. Roesler, Phys. Rev. D 55 (1998) 69.
R. Engel, A. Schiller, V.G. Serbo, Z. Phys. C 71 (1996) 651.

[57] See Physics Reference Manual at the GEANT4 Web page [1] under Documentation.

[58] D. Liljequist et al., J. Appl. Phys. 68 (1990) 3061.

[59] J. Apostolakis et al., CERN-OPEN-99-034 and INFN/AE-99/18 (1999).

[60] D. Cullen et al., EPDL97: the Evaluated Photon Data Library, 97 version, UCRL–50400, Vol. 6, Rev. 5 (1997).

[61] S. T. Perkins et al., Tables and Graphs of Electron-Interaction Cross Sections from 10 eV to 100 GeV Derived from the LLNL Evaluated Electron Data Library (EEDL), UCRL–50400 Vol. 31 (1997).

[62] S. T. Perkins et al., Tables and Graphs of Atomic Sub-shell and Relaxation Data Derived from the LLNL Evaluated Atomic Data Library (EADL), Z=1-100, UCRL–50400 Vol. 30 (1997).

[63] R. Shimizu et al., J. Phys. D9 101 (1976).

[64] S. Giani et al., CERN-OPEN-99-121 and INFN/AE-99/20 (1999).

[65] S. Giani et al., CERN-OPEN-99-300 and INFN/AE-99/21 (1999).

[66] H. H. Andersen and J. F. Ziegler, The Stopping and Ranges of Ions in Matter Vol.3, Pergamon Press (1977).

[67] A. Allisy et al., ICRU Report 49 (1993).

[68] J.F. Ziegler and J.M. Manoyan: Nucl. Instr. and Meth. B 35 (1988) 215.

[69] H. Paul, at URL
`http://www.uni-linz.ac.at/fak/TNF/atomphys/STOPPING/welcome.htm`.

[70] J. P. Wellisch, D. Axen, Phys. Rev. C 54 (1996) 1329.

[71] M. Laidlaw, J. P. Wellisch, private communication.

[72] A. Tripathi, et al., NASA technical paper 3621.

[73] ENDF/B-VI, Cross Section Evaluation Working Group, ENDF/B-VI Summary Document, BNL-NCS-17541 (ENDF-201) (1991), National Nuclear Data Center, Brookhaven National Laboratory, Upton, NY, USA.

[74] T. Nakagawa, et al., JENDL-3 Japanese Evaluated Nuclear Data Library, Version 3, Revision 2, J. Nucl. Sci. Technol. 32 (1995) 1259.

[75] FENDL/E2.0, The processed cross-section libraries for neutron-photon transport calculations, version 1 of February 1998. Summary documentation H. Wienke and M. Herman, report IAEA-NDS-176 Rev. 0 (International Atomic Energy Agency, April 1998). Data received on tape (or: retrieved on-line) from the IAEA Nuclear Data Section.

[76] Yu. N. Shubin, V. P. Lunev, A. Yu. Konobeyev, A. I. Ditjuk, Cross section data library MENDL-2 to study activation as transmutation of materials irradiated by nucleons of intermediate energies, INDC(CCP)-385 (International Atomic Energy Agency, May 1995).

[77] M. R. Bhat, Evaluated Nuclear Data File (ENSDF), Nuclear Data for Science and Technology, page 817, Springer Verlag, Berlin, Germany, 1992.

[78] H. C. Fesefeldt, Simulation of hadronic showers, physics and application, Technical report PITHA 85–02, 1985.

[79] A. Capella and J. Tran Thanh Van, Z. Phys. C 10 (1981) 249.

[80] B. Andersson, G. Gustafson, G. Ingelman, T. Sjöstrand, Phys. Rep. 97 (1983) 31, or A. B. Kaidalov Sov. J. Nucl. Phys. 45 (1987) 1452.

[81] M. Hofmann, J. M. Eisenberg, S. Scherer, M. Bleicher, L. Neise, H. Stocker and W. Greiner, Non-equilibrium dynamics of a hadronising quark-gluon plasma, nucl-th/9908031.

[82] R. G. Alsmiller, F. S. Alsmiller, and O.W. Hermann, Nucl. Instr.and Meth. A 295 (1990) 337.

[83] Yu. E. Titarenko et al., Experimental and computer simulations study of radio-nuclide production in heavy materials irradiated by intermediate energy protons, nucl-ex/9908012.

[84] M. G. Pia, Object-oriented design and implementation of an intra-nuclear transport model, Proceedings of Computing in high energy and nuclear physics CHEP 2000 Padova.

[85] S. A. Bass et al., URQMD: A new molecular dynamics model from GANIL to CERN energies, Wilderness 1996, Structure of vacuum and elementary matter, 399-405.

[86] V. Lara and J. P. Wellisch, Pre-equilibrium and equilibrium decays in GEANT4, Proceedings of Computing in high energy and nuclear physics CHEP2000 Padova, page 52.

[87] V. E. Weisskopf, D. H. Ewing, Phys. Rev. 57 (1940) 472.

[88] E. Fermi, Prog. Theor. Phys. 5 (1950) 1570.

[89] J. P. Bondorf, A. S, Botvina, A. S, Iljinov, I. N. Mishustin, K. Sneppen, Phys. Rep. 257 (1995) 133.

[90] N. Bohr, J. W. Wheeler, Phys. Rev., 56 (1939) 426.

[91] A. I. Blokhin et al., Brond-2.2: Current status of Russian Nuclear Data Libraries, Nuclear Data for Science and Technology, Vol. 2, page 695, American Nuclear Society, LaGrange, IL, 1994.

[92] CENDL-2: Chinese Nuclear Data Center, CENDL-2, The Chinese Evaluated Nuclear Data Library for Neutron Reaction Data, IAEA-NDS-61, Rev. 3 (1996), International Atomic Energy Agency, Vienna, Austria.

[93] H. D. Lemmel, EFF-2.4: The European Fusion File 1994, including revisions up to May 1995, Summary Documentation, IAEA-NDS-170, June 1995

[94] C. Nordborg, M. Salvatores, Jef-2: Status of the JEF Evaluated Data Library, Nuclear Data for Science and Technology, American Nuclear Society, LaGrange, IL, 1994.

[95] J. P. Wellisch, Neutron Induced Isotope Production On Selected CMS Elements Using GEANT4 , CMS-Note 1999/07.

[96] A. Levin, and C. Moisan, A More Physical Approach to Model the Surface Treatment of Scintillation Counters and its Implementation into DETECT, TRIUMF Preprint TRI-PP-96-64, Oct. 1996.

[97] G. F. Knoll, T. F. Knoll and T. M. Henderson, Light Collection Scintillation Detector Composites for Neutron Detection, IEEE Trans. Nucl. Sci., 35 (1988) 872.

[98] G. Grindhammer et al., Nucl. Instr.and Meth. A 290 (1990) 469.

[99] J. del Peso, E. Ros, Nucl. Instr.and Meth. A 306 (1991) 485.

[100] BOGUS (BaBar Object-oriented GEANT4 -based Unified Simulation) (to be published).

[101] J. Apostolakis, S. Giani, V. Grichine et al., Comp. Physics Comm., 132 (2000) 241.

[102] L.L. Carter and E.D. Cashwell, Particle transport simulation with the Monte Carlo method, TID-26607, Published by the US National Technical Information Center, Energy Research and Development Administration, 1975.

[103] H. Bateman, Cambridge Philosophical Society Proceedings, 15, 423-427, 1910.

[104] P.R. Truscott, Ph.D. Thesis, University of London, 1996.

[105] J. Shiers, Massive-Scale Data Management using Standards-Based Solutions, 16th IEEE Symposium on Mass Storage Systems, San Diego, USA, 1999.

[106] M. Nagamatsu, T. Kodama, H. Uno, H. Yoshida, K. Ohtsubo, S. Tanaka, M.Asai, Computing in High Energy Physics '98 (Chicago).

[107] At URL `http://erpc1.naruto-u.ac.jp/~geant4`.

[108] OPACS Web page: `http://www.lal.in2p3.fr/OPACS`.

[109] JAS (Java Analysis Studio) Web page: `http://jas.freehep.org`.

[110] H. Yoshida, T. Kodama, S. Sei, H. Kurashige, Computing in High Energy and Nuclear Physics 2000 (Padova).

[111] Latest information and download can be found at URL `http://geant4.kek.jp/GEANT4/vis`.

[112] S. Tanaka, Computing in High Energy Physics '97 (Berlin).

[113] HepRep Web page: `http://heprep.freehep.org`.

[114] WIRED Web page: `http://wired.freehep.org`.

[115] Information on the OpenGL library and the GEANT4 OpenGL driver at URL `http://www.opengl.org`.

[116] Information on the HEPVis library and the GEANT4 OpenInventor library at URL `http://cactus.phyast.pitt.edu/~joe/hepvis/hepvis.html`, `http://www.lal.in2p3.fr/Inventor`.

[117] J. Allison and S. Tanaka, Computing in High Energy Physics '97 (Berlin).

[118] AIDA Web page: `http://aida.freehep.org`.

[119] Lizard Web page: `http://cern.ch/anaphe/Lizard`.

[120] OpenScientist Web page: `http://www.lal.in2p3.fr/OpenScientist`.

[121] S. Tanaka and K. Hashimoto, Computing in High Energy Physics '98 (Chicago, 1998).

List of Figures