Evolution of GIGA*

I.Belyaev†
ITEP (Moscow)

June 25, 2000

Abstract

Several variants of evolution of GIGA[1] are discussed. The proposed schema for communications of algorithms within *GAUDI* framework with *Geant4* structures allows to use *Geant4 Tool Kit* as black-box without detailed knowledge of internal features of *Geant4 Tool Kit*.

Contents

*The document available at `$GIGAROOT/doc/GiGaEvlution.ps`

†`E-mail: Ivan.Belyaev@itep.ru`

[1]See `$GIGAROOT/doc/GiGa.ps`

1 Integration of *Geant4* into $\mathcal{GAUDI}$ framework

The Evolution of GIGA from stand-alone *Geant4* program to a simulation environment fully integrated into $\mathcal{GAUDI}$ framework could be represented by a set of phases and schematic diagrams of data flows. All indicated data flows consist of objects, specific for *Geant4 Tool Kit*.

1.1 Phase 0

Phase 0 corresponds to a stand-alone *Geant4* program. Data flows are schematically represented in figure 1.1.

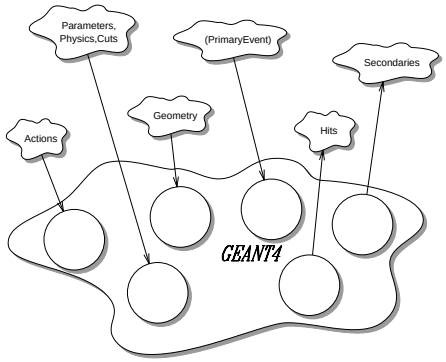

Figure 1: A schematic view of *communications* with *Geant4 Tool Kit*. Data flow directions are indicated by arrows.

1.2 Phase I

At this phase we foreseen the introduction of GIGA *Service* and the direct usage of this *service* and some limited number of *Geant4* classes by *user*'s *Algorithm*s. Data flows are schematically represented in figure 1.2.

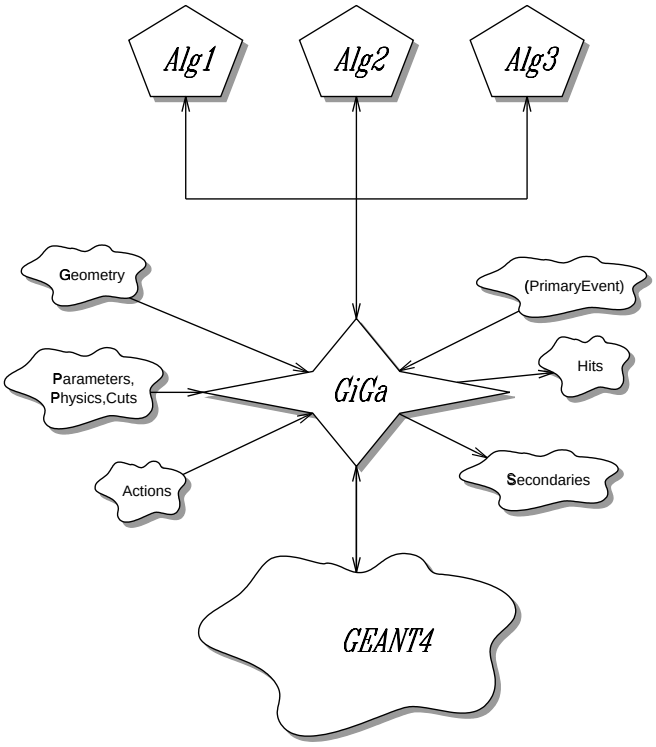

Figure 2: A schematic view of data flows during Phase I of integration of *Geant4* into $\mathcal{GAUDI}$ framework

This case corresponds to a native usage of "stand-alone" *Geant4* application within $\mathcal{GAUDI}$ framework. *User*'s algorithms act like wrappers over main program of *Geant4*. Frankly speaking at very beginning the only one profit that ordinary user gets from usage GIGA Service with respect to an ordinary "stand-alone" *Geant4* program, is the fast and easy access to general and technical $\mathcal{GAUDI}$ facilities like histogram, code profiling and others. On other side, if one has the working *Geant4* code, it could be embedded into $\mathcal{GAUDI}$ framework using GIGA facilities in a straightforward way without changing any line of codes[2].

Data flows between *Algorithm* and GIGA *Service* indicated in figure 1.2 still contain objects specific for *Geant4 Tool Kit*.

1.3 Phase II

This phase could be considered as *"transition phase"* between Phase I and Phase III and it could be easily subdivided into some steps:

[2]Only the `main()` function is to be removed

1. At this step we enhance the functionality of GIGA by making possible to extract the event record from *GAUDI tvent store*
2. At this step we enhance the functionality of GIGA by making possible to get the Detector Description by pointing into the root of already constructed *Geant4* tree
3. At this step we foreseen to implement the automatic translation of *GAUDI* Detector Description into *Geant4* detector description.
4. At this step we foreseen to implement the automatic creation of *Geant4 hits* and *sensitive volume* from their description via XML. A some common brainstorming within close collaboration with sub-detector groups is necessary for performing of this step.
5. At this step we foreseen to automatic translation of *Geant4 hits* into *GAUDI* Monte Carlo objects[3]
6. At this step we foreseen the automatic population of *GAUDI event store* by information from *Geant4 trajectories*[4]

The data flows at the end of Phase II are presented in figure 1.3. All input and output data flows from *Algorithm*s consist of only *GAUDI* specific objects. An intermediate layer of *Converter*s an d*conversion services* prevents any contacts between *Algorithm*s and objects, specific for *Geant4*. One sees that now GIGA *Service* itself becomes an internal and invisible object for ordinary user.

In addition one could foreseen the existence of intermediate sub-phase, sketched in figure 1.3.

After implementing first two steps GIGA Service could be considered as *"frozen"*, since all other steps to Phase III will not require any change in functionality of GIGA Service itself.

1.4 Phase III

At this phase no any *user*'s algorithm deals directly with GIGA Service and *Geant4* classes. All knowledge of *Geant4* will be absorbed by set of specific *converters*. This set of specific *converters* corresponds to an additional layer in the data flow, making the user free from the knowledge of *Geant4* machinery.

Being at this phase we could start to think about configuration of *Geant4 physics list* and/or *cut-offs* using internal *GAUDI* features like *jobOptions Service* and/or *interactive scripting language*. Moreover at this stage we foreseen the embedding of the essential commands from *Geant4* interactive *user interface* into *GAUDI interactive scripting language*.

[3]Obviously this step could not be done without close collaboration with sub-detector groups
[4]Communications physics and generator groups are required

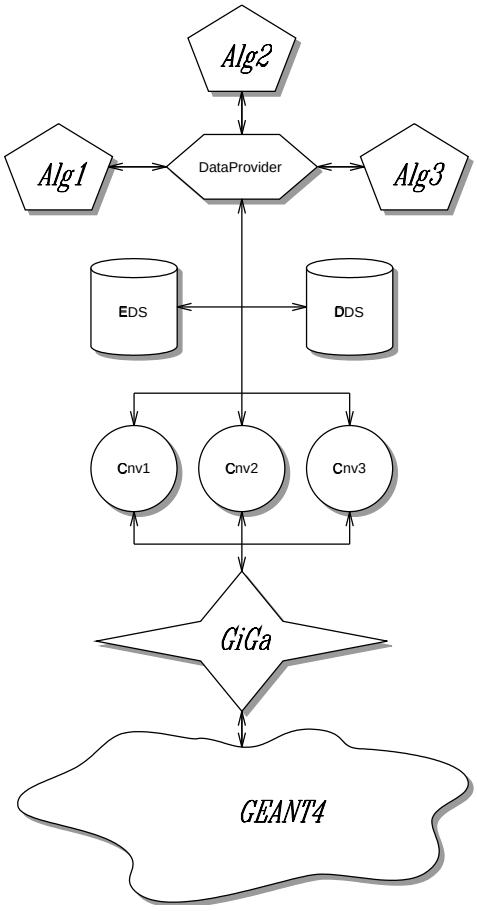

Figure 3: A schematic view of data flows at the end of Phase II of integration of *Geant4* into $\mathcal{GAUDI}$ framework

2 Alternative scenarios

For some proposed components one could consider also some alternative variants.

2.1 Communication with *Generator*

In the scheme sketched above we have assumed that for defining the primary event GIGA will use the whole event record from Transient Store, prepared outside the GIGA and *Geant4*. Such approach allows us to make a clear separation between *generator* and *simulation.*

Alternative schema could be *Generator* embedded into GIGA and/or

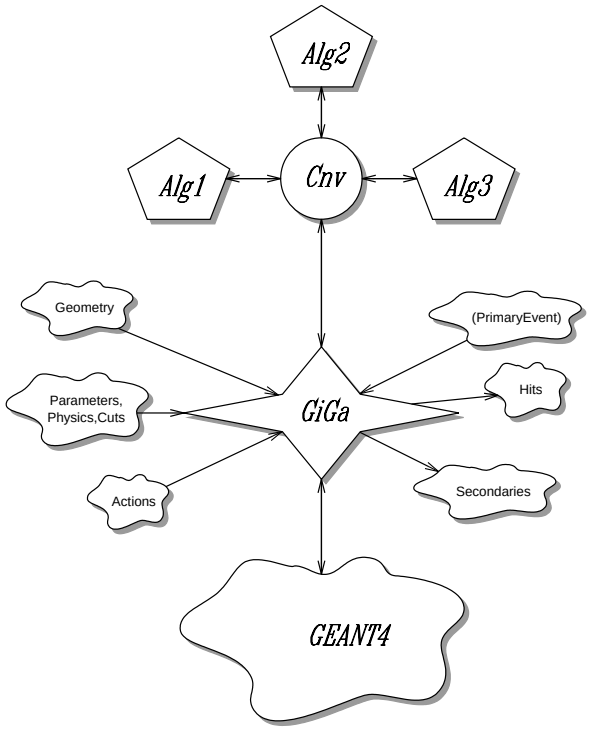

Figure 4: A schematic view of data flows at some intermediate sub-phase during Phase II of integration of *Geant4* into $\mathcal{GAUDI}$ framework

Geant4[5].

*Pro*s of this variant are obvious:

- This is the way, recommended by *Geant4* team
- Such feature is foreseen for `PYTHIA 7`, written in C++
- Since now `HepMC` indicates the tendency to be ”standard” for LHC, probably we could benefit from some developed `HepMC`-*Geant4* interface

*Contra*s are also obvious:

- Embedding *generator* into the *simulation* we loose the flexibility
- Dedicated B-decays *generator* is written in `FORTRAN`
- Tuned B-production and minimum bias *generator*s are written in `FORTRAN`
- Code support becomes problematic
- `PYTHIA 7` is not yet ready

[5]As it done in SICBMC

Making comparison between *pro*s and *contra*s one could conclude, that *now* we should no follow the way of embedding the *generator* into our *simulation* environment, but in some day in the future after successful tests of new generation of object-oriented *generator*s we could come back to the rediscussion of the question.

3 General ”to be done” list and hints for implementation

3.1 Architectural issues

3.1.1 Geometry Conversion Service

Geometry Conversion Service is considered to be responsible for automatic conversion of geometry, stored in Detector Data Store into Geometry tree known for *Geant4 Tool Kit*.

Since geometry description within $\mathcal{GAUDI}$ framework is build on the same basic principles[6] as geometry within *Geant4 Tool Kit*, we are pretty sure that each individual converters (for solids, physical and logical volumes) used within Geometry Conversion Service is to be quite simple.

For implementation one could use the general principles of the Conversion Service and concrete Converters, developed for visualisation converters within *GaudiLab* package by Guy Barrand. The notion of ”converter without IOpaqueAddress” fits perfectly into the general schema of Geometry Conversion Service.

One aspect is to be taken into account. It concerns the notion of repeated volume. Within *Geant4 Tool Kit* there exist 3 kinds of positioned volumes - single positioned, parametrised and replicated volumes. Currently within $\mathcal{GAUDI}$ framework we also have notions of single positioned physical volume and parametrised physical volume. Both of them must be converted into single positioned *Geant4* volumes.

Also one need to extend the functionality of $\mathcal{GAUDI}$ with introduction of the notion of replicated volume, which is to be converted into replicated volume in *Geant4*. But One should take into account that no any *Alignment* could not be applied to replicated volumes. It immediately results in the restriction that $\mathcal{GAUDI}$ *Detector Elements* could not be of the type ”replicated volume”.

[6] see `$DETDESCROOT/DetDesc/Volumes/doc/volume.man.ps`, `$DETDESCROOT/DetDesc/Solids/dos/solid.man.ps` and `$DETDESCROOT/DetDesc/Solids/doc/Solid.ps`

3.1.2 Material Conversion Service

Schema of material description within $\mathcal{GAUDI}$ framework is almost identical to the material description within *Geant4*, thats why we could not expect any difficulties implementing such Conversion Service.

For implementation one could also use the general principles of the Conversion Service and concrete Converters, developed for visualisation converters within *GaudiLab* package by Guy Barrand.

This Service could be public service or it could be just a private service, visible only for Geometry Conversion Service.

3.1.3 Primary Event Conversion Service

The same scheme of "converters without IOpaqueAddress" also fits well into the implementation of Conversion Service for Primary Event.

One comment should be mentioned here. With current version of `MCParticle` class, one gets a significant overhead in the performance during the conversion procedure. The reason of this overhead is the notion of `ParticleID`. In current version it contains the particle identifier used in `GEANT3` program. Thus the conversion service should convert this `GEANT3` identifier into `StdHep(Pythia)` identifier[7] and then using look-up table from *Geant4*, convert this `StdHep(Pythia)` identifier in the form, acceptable for `Geant4`. One could easily skip this additional overhead in the performance, using `StdHep(Pythia)` particle coding schema for `ParticleID`.

3.1.4 Hit Conversion Service

The service for conversion of hits from *Geant4* structures into $\mathcal{GAUDI}$ Transient Store, the service for conversion of secondaries and the standard stacking actions are very closely connected. And it should be taken into account during the implementations of these notions. For the fist implementation one could assume no any cleanup action during conversion process of hits and secondaries. Under this assumption, hits conversion service becomes very primitive. And its implementation depends mainly on the definition of hit in *Geant4*. Here several strategies could be considered, but all of them utilise the advantage of the approach then *G4Hit*s in *Geant4* and *MCHit*s in $\mathcal{GAUDI}$ are essentially "the same" objects. It means that in the simplest case Conversion Service is just change the representation of the objects, e.g. it could just change the base of the object from `G4Hit` into `Contained Object`. Under this assumption converters become trivial.

[7] It is not a very fast operation due to internal implementation of `IParticlePropertySvc` using `std::map`

The structure of the Hit Conversion Service may follow either the idea of ”converters without IOpaqueAddress” or it could use the idea of extended IOpaqueAddress, since hit-collections in GIGA and *Geant4* could be addressed directly using some name and collection identifier, defined for sensitive detector.

3.1.5 Secondaries Conversion Service

Nothing specific could be said about this service. All consideration applicable to previous conversion services are valid for this concrete service

3.1.6 Declaration of sensitive detector and hits

The possibility of external definition of sensitive detector and definition of hit structure is under study now. Such possibility could substantionally reduce the direct communications of ordinary users with GIGA and *Geant4*. It is not yet clear, if it could be done in a quite generic way, but personally I am still quite optimistic.

3.2 Infrastructure

To get the working simulation program, it is not enough to establish the set of converters described in the previous sub-section. Also some general actions are to be performed.

3.2.1 Magnetic Field and Steppers

Definition of Magnetic field for GIGA and *Geant4* is to be done on the basis of `IMagneticFieldSvc`.

Some attention is to be paid to the system of units used by this service.

Geant4 defines several steppers for transport of the particles. The optimal choice of the stepper depends on the magnetic field description. Tracking performance depends strongly on a correct choose of the stepper. Usage of unappropriate combination of stepper and the field could result in a very bad performance and even in incorrect results. Some study should be performed on the best choice of the stepper for our magnetic field description, or even the revision of the magnetic field description is to be done.

3.2.2 Physics List, Particle List and Generic Cuts

Probably it is the most important aspect for GIGA and *Geant4*. At initial stage we could use the physics list and generic cuts, adopted by other groups, and the particle list consistent with `SICB` particle list. But obviously in

future we should pay a lot of attention for appropriate definition of our own physics list and cuts, specific for our experiment.

3.2.3 Implementation of specific Trajectories and their relations with Hits

It seems to be unavoidable to define a specific trajectories for implementation of effective stacking action and final track/hit clean-up. Unfortunately now we could not estimate the efforts needed.

3.2.4 Declaration of specific parameters

A special technique is to be developed to provide GIGA and *Geant4* with *specific* parameters needed for specifig geoemtry elements e.g. maximal step size for a given geometry volume.

This information should not be a part of geometry description and is to be supplied additionally either via a separate XML-file with description of specific features and cuts of named volumes.

A good alternative to XML-description could be definition of such parameters using some scriptic language and user interface.

3.2.5 Stepping, stacking and tracking actions

The corrrect definition of stepping, stacking and tracking actions looks as one of the most essential parts of teh whole simulation environment with respect of CPU performance, especially it concerns the stacking technique for secondaries. An optimal definition of our own trajectories and their relations with hits could help a lot.

As it seems to me, the implementation of such actions shoulw be iterative, making the optimal physics and CPU performance.

3.2.6 Interactivity and user interface and scripting

Currenlt $\mathcal{GAUDI}$ provides us with a pure batch environment without event-by-event user interface. It is not acceptable for tuning of parameters of simulation environment and visualisation.

In the current implemenattion of GIGA a set of user interfaces from *Geant4* is used in a some primitive way. Obviously it is only temporary solution, and user interfaces from *Geant4 Tool Kit* shoule be replaced by user interface from $\mathcal{GAUDI}$ framework.

The implementation of an appropriate interactive user interface and scripring in $\mathcal{GAUDI}$ could be considered as one of the most important request from *simulation.*