Architecture of collaborating frameworks: simulation, visualisation, user interface and analysis

A. Pfeiffer (CERN, Geneva, Switzerland)
G. Cosmo (CERN, Geneva, Switzerland)
B. Ferrero Merlino (CERN, Geneva, Switzerland)
R. Giannitrapani (Università di Udine and INFN Sezione di Trieste, Italy)
F. Longo (Univ. of Ferrara and INFN Sezione di Ferrara)
P. Nieminen (ESA/ESTEC, Noordwjik, The Netherlands)
M.G. Pia (INFN Sezione di Genova, Italy)
G. Santin (Università di Trieste and INFN Sezione di Trieste, Italy)

Abstract

In modern high energy and astrophysics experiments the variety of user requirements and the complexity of the problem domain often involve the collaboration of several software frameworks, and different components are responsible for providing the functionalities related to each domain.

For instance, a common use case consists in studying the physics effects and the detector performance, resulting from primary events, in a given detector configuration, to evaluate the physics reach of the experiment or optimise the detector design. Such a study typically involves various components: Simulation, Visualisation, Analysis and (interactive) User Interface.

We focus on the design aspects of the collaboration of these frameworks and on the technologies that help to simplify the complex process of software design.

Keywords: OO Frameworks Components Design Simulation

1 Introduction

A characterisation of progress in software development has been the regular increase in levels of abstraction. As the size and complexity of software systems increase, the design problem goes beyond algorithms and data structures: designing and specifying the software architecture, that is the overall system structure, emerges as a new kind of problem.

This is especially true in the case of high energy physics and astrophysics experiments, where the variety of user requirements and the complexity of the problem domain often involve the collaboration of several frameworks, and different components are responsible for providing the functionalities related to each domain.

For instance, a common use case in high energy physics or astrophysics experiments consists in studying the physics effects and the detector performance, resulting from primary events, in a given detector configuration. This kind of study may be performed in various phases of the life-cycle of an experiment: it may be needed to evaluate the physics reach of the experiment in its early stages, to optimise the detector design or its protection from the effects of radiation backgrounds, or to estimate quantitatively physics effects contributing to the final experimental results, such as the subtraction of backgrounds or the evaluation of systematic errors. This study is usually performed by means of the simulation of the passage of particles originated by the primary physical event through the experimental set-up, accounting for their interactions and production of secondaries; it requires the display of the experimental apparatus and of the events; eventually it involves the analysis of the effects resulting from the particles traversing the detector. The whole set of operations is handled through a user interface.

In the use case illustrated above various software entities are involved: Simulation, Visualisation, Analysis and User Interface. In many experiments, whose software requires advanced

functionalities and a high degree of flexibility, these entities are often represented by individual frameworks, consisting of various components.

In this paper we focus on the design aspects of the collaboration of these frameworks and on the technologies that help to simplify the complex process of software design for high energy physics and astrophysics experiments.

2 Object Oriented Analysis and Design

Various technologies are helpful in the context of Object Oriented design. The Unified Modeling Language (UML) [1] has been adopted to simplify the complex process of software design. UML is the industry-standard language for specifying, visualising, constructing and documenting the design of software systems. In the project described in this paper, UML has provided the language for class and object modeling, and component modeling.

The Booch methodology [2] has been applied for the object oriented analysis and design of the software. A spiral approach, that is both iterative an incremental, has been adopted in the software process related to the work described in this paper. Steps among analysis of the requirements, design and implementation have been repeated, incrementing and refining the overall architecture.

Patterns play a significant role both at the architectural and the design level.

Architectural patterns are high-level strategies: they concern the large-scale components and express a fundamental structural organization for the software system. An architectural pattern provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.

Design patterns are medium-level strategies: they define micro-architectures of subsystems and components, and address some of the structure and behaviour of entities and their relationships. In the work exposed in this paper design patterns have been utilised to describe a commonly recurring structure of communicating components, exploiting their power to solve a general design problem within a particular context.

3 The components

3.1 The simulation core

A high level of flexibility, coupled with a rich and diverse functionality, is needed to cope with the complexity of typical detectors in high energy physics and astrophysics experiments.

The Geant4 Toolkit [3] is a simulation code designed and developed to address the requirements [4] elicited for such experiments.

One of the main tasks for a detector simulation code consists in the ability to model the experimental set-up: this includes its geometrical description, the specification of materials and the generation of the response of the detector components.

The wide diversity of physics measurements in high energy and astrophysics experiments requires the capability to handle electromagnetic and hadronic interactions of particles with matter over a wide energy range. Because of the variety of physics applications involved, the possibility of providing different models for the same physics process should be envisaged.

Additional requirements address the availability of various utilities essential to the simulation domain, such as the management of particle features, like, for instance, the Particle Data Group database, the management of random numbers, the possibility of permanent storage of simulation results or the interface to primary physics generators.

Problem domain decomposition and object oriented analysis result in Geant4 in a clean, unidirectional dependency structure of class categories, as shown in the architectural design

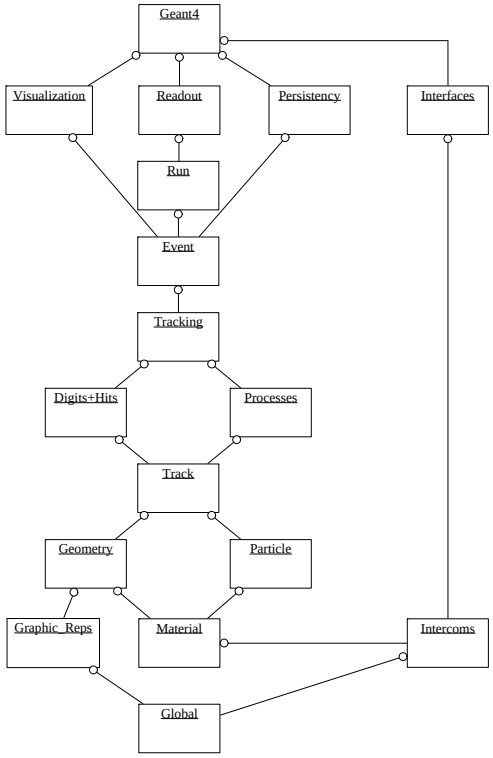

Figure 1: Architecture of the Geant4 Simulation Toolkit: the class category diagram

diagram in figure 1. Categories correspond to releasable components (libraries).

3.2 The analysis

Similar to the other categories of Geant4, the analysis category separates the required *functionality* from the actual implementation. Any analysis of data within Geant4 is done by relaying the request to an specific instance of an abstract Analysis-System (class G4VAnalysisSystem). The actual instances of the Analysis-Systems are organized by an instance of the G4AnalysisManager class, which deals with initialisation and control of the selected Analysis-Systems. The selection of which Analysis-System to use is eventually done by the user at the startup of the simulation, allowing a very flexible customization.

The communication with the actual Analysis-System is presently done via a set of Abstract Interfaces provided by the AIDA project [5]. Therefore the (internal) framework used by the specific implementation of the Analysis-System is not exposed (or imposed on) the simulation framework. The use of Abstract Interfaces also allows to interchange the Analysis-System without the need to recompile the code; if the implementation is done using shared libraries, this could be done even at run-time.

3.3 The visualisation

The visualization category controls Geant4 visualization, i.e., visualization of detector geometry, particle trajectories, hits, and so on. The *visualization manager*, which is implemented as the class G4VisManager, plays a main role in the visualization procedures. It takes three-

dimensional data from the geometry and graphics representation categories and pass them to the abstract *visualization driver*.

The abstract visualization driver consists of the three abstract classes, to initializing a graphics software, to processing the 3D data for visualization, and to render the processed 3D data, respectively. These three steps are nothing but abstraction of the typical visualization procedure which are common to most modern graphics software.

A concrete visualization driver can be easily implemented by inheriting from each of the above three abstract classes. Several concrete visualization drivers are already available by default.

3.4 The user interface

In designing the (Graphical) User Interfaces of Geant4, two coordinates had to be analyzed; the categorization of users and the phases of user actions.

We have identified three categories of users, these are the **end user** who runs Geant4 by controlling and setting run-time parameters with the (G)UI; the **application programmer** who creates simulation programs which are specific to his detector, thereby defining his own commands and sets of associated parameters to be set by end users; finally the **framework provider** who is a Geant4 developer. For both – end-user and application developer frameworks – (G)UIs were requested. The former is to control the execution of Geant4, while the latter help programmers to create their classes.

There are two phases of user actions during the execution of Geant4; one for the set-up of the simulation, and another for the control of event generation and processing. User requirements revealed that user interactions are distributed in all categories of Geant4 for the above two phases. For example, the detector geometry and the physics processes are defined in the first phase while the "beam" is defined in the second.

4 Conclusions

Applying object-oriented methodologies and tools as well as identifying and using patterns (on the architectural and the design level) has been proven to be of great advantage when dealing with complex software systems.

Well defined components together with Abstract Interfaces defining their behaviour can collaborate together, providing an increase of functionality and flexibility at very little additional cost.

Thanks to this approach of "collaborating frameworks", user applications can address transparently all domains of simulation, visualisation and analysis through the same user interface. As a concrete example, in the simulation domain proper, a large number of features, related to all the simulation categories, can be controlled and customised through a unique user interface: the run control, the detector configuration, the selection of physics processes etc.

References

[1] I.Jacobson, G.Booch, J.Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1999, ISBN 0-201-57169-2.

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

[3] The Geant4 Object Oriented Simulation Toolkit, see `htpp://cern.ch/Geant4`

[4] RD44, *GEANT 4 User Requirements Document*, CERN, 1998.

[5] See `http://aida.freehep.org/`