Status of the GAUDI event-processing framework

M. Cattaneo[1], *M. Frank*[1], *P. Mato*[1], *S. Ponce*[1], *F. Ranjard*[1], *S. Roiser*[1], *I. Belyaev*[1,2], *C. Arnault*[3], *P. Calafiura*[4], *C. Day*[4], *C. Leggett*[4], *M. Marino*[4], *D. Quarrie*[4], *C. Tull*[4]

[1] European Laboratory for Particle Physics (CERN), Geneva, Switzerland
[2] Institute for Theoretical and Experimental Physics (ITEP), Moscow, Russia
[3] Laboratoire de l'Accélérateur Linéaire (LAL), Orsay, France
[4] Lawrence Berkeley National Laboratory (LBNL), Berkeley, USA

Abstract

The GAUDI architecture and framework are designed to provide a common infrastructure and environment for simulation, filtering, reconstruction and analysis applications. Initially developed for the LHCb experiment, GAUDI has been adopted and extended by the ATLAS experiment and adopted by several other experiments including GLAST and HARP. We describe the properties and concepts embodied by GAUDI and recent functionality additions, and how the project has evolved from a product developed by a tightly-knit team in a single site, to a collaboration between multiple teams at geographically dispersed sites, based loosely on open source concepts. We describe the management infrastructure as well as how we accommodate experiment-specific extensions and adaptations as well as an experiment-neutral kernel.

Keywords: Architecture, Framework, and Collaboration

1 Introduction

The GAUDI [1] object-oriented framework is designed to provide a common infrastructure and environment for simulation, filtering, reconstruction and analysis applications for the current generation of HEP experiments. These experiments are expected to run for many years and therefore changes in software requirements and in the technologies used to build software have to be taken into account by developing flexible and adaptable software that can withstand these changes and which can be easily maintained over the long timescales involved.

The development process for GAUDI is architecture driven. This involves identifying components with specific functionality and well-defined interfaces, and which interact with each other to supply the overall functionality of the framework. In addition, the approach to the final software framework is via incremental releases, adding to the functionality at each release according to the feedback and priorities provided by the users. Since all the components are essentially decoupled from each other, they can be implemented one at a time and in a minimal manner, *i.e.* supplying sufficient functionality to do their job, but with initial implementations possibly lacking refinements and optimizations that can be added at a later date. In addition, any component can easily be replaced by another component which implements the appropriate interface and provides the appropriate functionality, making the use of "third-party" software possible. The possibility of refining the functionality, or of replacing of the implementation while keeping the interfaces, has allowed us to easily customize the framework, so that it can be adapted to different tasks and integrated with components from other frameworks and is easily adaptable for use in other experiments. As a result of these features the GAUDI framework, which was developed originally in the context of the LHCb [2] experiment, has been adopted and extended by the ATLAS [3] experiment and is also in use by several other experiments including GLAST and HARP.

2 Extensions to the Framework

Recent enhancements that have been made to the framework include:

- A prototype scripting language interface based on Python has been added [4]. This extends the framework to support both a batch-oriented mode of operation, where a job is configured an then runs to completion, and a more interactive style, where a job can be configured, some events can be processed, the configuration can be changed, more events may be processed, *etc*. The functionality of the prototype is somewhat limited, focusing on flexibility of configuration changes (*e.g*. creation of new Algorithms or Services, modification of configuration parameters). Enhancements to support browsing of the contents of transient stores, and more flexibility in controlling execution are being added.

- More complex control sequences can be established supporting branches and filtering. These can be combined with multiple output streams to support event filtering and selection.
- Extensions have been made to the *Properties* or adjustable parameters. These now support validity range checking, and callbacks such that they may be calculated algorithmically or provide immediate notification of updates.
- Several new services have been added. These include the *Auditor* service, which provides monitoring of resources (e.g. cpu-time, memory usage), on a per Algorithm basis. This service itself uses the *ChronoStat* service which provides timing statistics.
- The Python scripting support is implemented as a concrete implementation of an abstract service (*Runable*), allowing the possibility of providing alternative implementations. Other examples of multiple implementations of the same abstract service interface are the *HistogramPersistency* and *NtuplePersistency* services, where implementations based on HBOOK and ROOT are available.
- Support for dynamic loading of shared libraries has been improved.
- The service management infrastructure is in the process of being redesigned. This will minimize couplings, and allow for the deferred instantiation of services or for the default implementations of standard services to be overridden.

Other work in progress is in support of detector geometry and alignment, and general support for time-varying information based on a Conditions database. Code generation tools that will generate, for example, the transient and persistent representations of an object, together with the converters that transform one to the other, are also under development [5].

The ability of the framework to support different implementations of the same abstract interface is also being taken advantage of by allowing for experiment-specific implementations. One example of this is in the area of transient stores where ATLAS and LHCb have taken different implementation routes, suited to different event data models, but have retained the same abstract interface. Which implementation is used is then just a matter of specifying different configuration parameters.

3 Setting-up a collaboration

Collaboration between experiments to share a framework can only be done if all parties foresee some benefits from doing so. A single experiment, in particular if it is a rather small one, cannot provide all the functionality required by a general framework (*e.g.* data management and object persistency, data visualization, interactivity, *etc*.). Therefore it is a real benefit if the contributions from all the development teams can be put together as part of a coherent framework and can directly be used by all the experiments. The expected quality of the product is also naturally improved since the framework can be expected to be used in rather different environments (platforms, problem domains, *etc*.) having a larger use case coverage. In addition, the designs of framework components are discussed within a much broader development team, and thus in principle should be of better quality. In the case of an experiment starting to develop their data processing applications it is always better to start with an existing architecture design and adapt it to their particular needs than starting from a clean sheet of paper. Finally, once the maintenance phase has been reached, the effort can be shared between development teams. This is of particular importance given the long running time of the current generation of experiments.

3.1 Common software repository and configuration tool

When setting-up the collaboration between ATLAS and LHCb it was felt essential to share a single common repository for all the experiment independent software. This is important to avoid potential divergences, which are very natural during the iterative and incremental development of the software in both collaborations. Having a single repository guarantees that the software developed on top of the common framework is directly usable by all partners. One problem that must be faced is to decide what constitutes experiment neutral, rather than experiment specific, software. Basically, the primary interfaces and base classes of the framework constitute the common part in addition to a number of common service implementations that are also of general interest (*e.g.* the histogram persistency service and message service). Services specific to the experiment event model are obviously not part of the experiment-neutral kernel. Detector data services (geometry and detector conditions) are also candidates if more than one experiment decides to use the same model. Another approach to minimizing divergence is to review components that were initially developed in the context of a single experiment to decide whether they are general enough for more common adoption. This implies that there should be a mechanism for migrating components from experiment-specific repositories to the experiment-neutral repository.

The common code repository is based on CVS, is independently managed from experiment code repositories, and is subject to its own set of rules and conventions. It is actually located in the GAUDI project area on the CERN AFS space with an associated CVS server for remote access from sites without AFS.

The rest of the common GAUDI project area includes a web for documentation and project information, installation kits, and a release area with the latest versions of all common packages that can be used directly. The GAUDI common software is managed with the CMT [6] configuration management tool.

3.2 Project Management

Our philosophy has been to produce a very lightweight structure to manage the common GAUDI project. The minimal identified roles were the Project Manager, Software Architect and Librarian. The Project Manager is in charge of defining priorities, coordinates interactions with the experiment development teams, and generally keeps the project focused on the right goal. The Software Architect leads and coordinates technical activities, and establishes the overall structure for each architectural view: the decomposition of the view, the grouping of components, and the interfaces between these major components. Finally, the Librarian supports the development activity so that developers and integrators have appropriate workspaces to build and test their work.

Scheduling releases of the common kernel is done by consensus of the experiments involved and by always trying to make the best match with their specific project milestones and their priorities.

Regular meetings are essential for keeping the different development teams informed and aim for a common goal. It is not unusual that members of one experiment participate in meetings of the other experiment on topics of common interest. For example data dictionary services, detector conditions integration, etc.

3.3 Difficulties

The main difficulties are, on one hand, due to the differences on the requirements and constraints for the different experiments and in having to find a compromise on what could be common and what needs to be left in the experiment specific repositories. On the other hand, the differences on the development environments for the collaborations in terms of tools, platforms, compilers *etc.* can also complicate the task. It is clear that if both collaborations use the same software development tools it simplifies enormously the work required in developing and re-using code.

The experiment framework is a piece of software that by its nature will need to interact with many different packages in different domains, *e.g.* database, interactivity, graphics, *etc*., in order to create a coherent structure for the development of experiment data processing applications. Therefore, there is a risk of having difficulties in finding a coherent set of packages and versions that fulfill all the dependencies for both experiments. In addition, each experiment may have their conventions about naming versions, naming platforms *etc*. and this could cause difficulties in using directly the released packages in the project area.

Resolving potential problems involves good communication between the development teams and a clear common goal. The focus on abstract interfaces is a crucial component of this.

4 Conclusions

The common GAUDI kernel packages are released few times a year with improvements and added functionality developed mainly by LHCb and ATLAS core software development teams and with very valuable feedback from the other experiments using the framework. A common project organization is in place and has been kept very lightweight for the time being. The project area provides a common software repository and a release area that is used directly by LHCb end-users running their programs and will soon be by ATLAS users.

Perhaps is too early to see the direct benefits of our approach but we are convinced that the extra effort we have put at the beginning to setup this project and separation of experiment-neutral software from the specific one will pay in the long term.

References

[1] G. Barrand *et al*., *GAUDI: A Software Architecture and Framework for building HEP Data Processing Applications*, CHEP 2000 proceedings, Padova, Feb. 2000 (see also http://cern.ch/Gaudi)

[2] S. Amato *et al*., *LHCb Technical proposal*, CERN/LHCC 98-4, Mar. 1998

[3] W. Armstrong *et al*., ATLAS Technical Proposal, CERN/LHCC 94-43, Dec. 1994

[4] C. Day *et al*., *Adding a Scripting Interface to Gaudi*, CHEP 2001, Beijing, Sept. 2001

[5] A. Bazan, *et al*., *The Athena Data Dictionary and Description Language*, CHEP 2001, Beijing, Sept. 2001

[6] C. Arnault, *CMT : A Software Configuration Management Tool*, CHEP 2000 proceedings, Padova, Feb. 2000