POOL INTEGRATION INTO THREE EXPERIMENT SOFTWARE FRAMEWORKS

G. Govi*, R. Chytracek*, D. Düllmann, M. Frank, M. Girone*, V. Innocente, P. Mato Vila, J.T. Moscicki*, I. Papadopoulos, H. Schmuecker (CERN, 1211 Geneve 23, Switzerland)
K. Karr#, D. Malon#, A. Vaniachine# (Argonne National Laboratory, Argonne, IL 60439, USA)
P. Van Gemmeren (Brookhaven National Laboratory, Upton, NY 11973)
A. C. Schaffer (Laboratoire de l'Accelerateur Lineaire, F-91898 Orsay cedex, France)
W. Tanenbaum (Fermi National Accelerator Laboratory, Batavia, IL 60510, USA)
Z. Xie (Princeton University, Princeton, NJ 08544, USA)
T. Barrass (University of Bristol, Bristol, BS8 1TL, UK)
C. Cioffi (University of Oxford, Oxford, OX13NP, UK)

Abstract

The POOL software package has been successfully integrated with the three large experiment software frameworks of ATLAS, CMS and LHCb. This paper summarizes the experience gained during these integration efforts and highlights the commonalities and the main differences between the integration approaches. In particular, the role of the POOL object cache, the choice of the main storage technology in ROOT (Tree or Named Objects) and the approaches to catalogue integration are discussed.

INTRODUCTION

The POOL project [1] has been created within the LCG Application Area [2], to provide the LHC experiments with a common software framework for persisting data.

Its main aim is to provide access to a generic data storage system for various types of C++ objects, exposing an API independent of any backend technology. This feature means that software architectures can be easily adapted in data handling technology over the LHC lifetime.

POOL has encouraged the concrete involvement of the experiments in the project, including some experiment members as a part of the POOL core developer team. This has been particularly important when defining the specific requirements (from synthesis of often overlapping use cases) and to find a solution as common a solution as possible.

In the first phase, the focus of the project has been concentrated in addressing a solution for the storage of event data objects. Typically, the experiment data models involve complex hierarchal structures, described by non-trivial object types. A well-suited solution for this use case has been found in the file-based object streaming provided by the ROOT [4] framework.

In line with its primary scope, POOL has developed a specific backend for the ROOT-based persistency [5], which allows handling the ROOT I/O mechanism through the generic object storage API.

In the POOL Storage System the functionalities of the ROOT streaming are fully maintained, offering both Key-based and Tree-based storage formats. POOL also provides additional features, like navigation capabilities among object associations, centralized control of file opening and a transaction-based access to the storage system.

The first two years of life of the project have been spent in the development and consolidation of the ROOT-based object storage system, and of file catalogues and collections based on several technologies [6].

In parallel, great efforts have been made by the experiments to integrate the POOL software into their frameworks. In fact, the follow up and the validation of the various POOL releases into the frameworks have significantly contributed to the consolidation of the software.

Although a large part of the existing POOL software has reached a significant level of maturity, important development activities, like the implementation of an RDBMS backend, are still underway. Most of the POOL API has been integrated in the ATLAS [8], CMS [9] and LHCb [10] software frameworks and widely used in large production activities. These experiences show different approaches in the use of POOL and can be seen as a first large-scale validation test for the API.

Component-based Architecture and API

The POOL architecture is structured as a set of hierarchically connected Service APIs (see fig. 1). The I/O for the data objects towards the core POOL *Storage Service* can be controlled through more intermediate layers, depending on the additional feature required.

The lowest-level public interface is represented by the *Persistency Service*. This interface allows the storage and retrieval of data objects through simple non-typed pointers, leaving the control of the object bookkeeping to the client.

Over the *Persistency Service*, POOL exposes an additional interface, the *Data Service*. Its main role is to

*funded by Particle Physics and Astronomy Research Council, UK
#work supported in part by the U.S. Department of Energy, Division of High Energy Physics, under contract W31-109-Eng-38

act as an object store, allowing efficient object re-use and loading-on-demand. The Data Service interface is based on smart pointer classes called *References*, which wrap the individual data objects and enables for type-safe storage or retrieval.

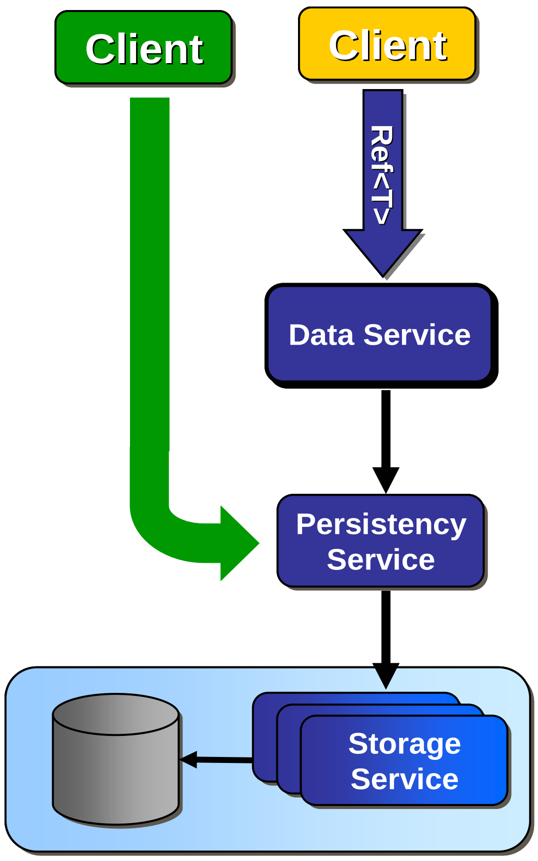

Figure 1 Hierarchical structure of the POOL Storage Service package, with the main Service APIs.

Both the *Persistency Service* and the *Data Service* provide access to the *Session* object, which can be used to control the transaction management, and the explicit handling of *Database* and *Container* objects.
The integration of POOL in a software framework can follow different methods, depending on which POOL components are involved and in which configuration. From the early stage of the POOL developments, the need to produce a minimal impact on the existing experiment code was an important influence on the design of the POOL API.
The three LHC experiment (ATLAS, CMS and LHCb) who have adopted POOL have chosen three integration approaches which differ for the POOL components used and/or the configuration.

INTEGRATION IN ATLAS

The ATLAS offline software is based on the Athena framework [11]. Athena provides the common services needed by simulation, reconstruction, and analysis, and in particular includes persistence services. The Athena architecture, constructed on the GAUDI kernel framework [12], provides access to a generic I/O protocol. The POOL object storage system is seen by Athena as a particular I/O technology. A dedicated *Conversion Service* manages the POOL-specific aspects of object storage through an interface that is independent of POOL or any other specific technology choice.

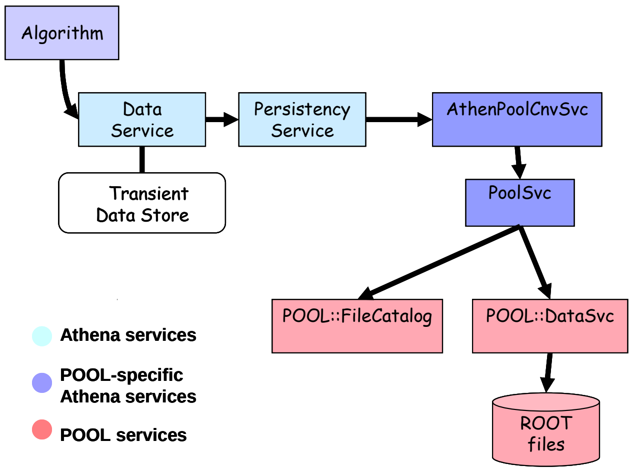

Figure 2 Integration of the POOL components into the Athena framework.

In the Athena framework, object bookkeeping is managed by dedicated transient data stores. When an object storage or retrieval operation involves POOL, the *POOL Service* is invoked. This component accesses the POOL *Data Service* using the *Reference* interface. In this case, however, the lifetime of the objects is not controlled by the POOL *Reference*, being explicitly managed by the Athena Data Store. Separate POOL *Data Services* are instantiated for input and output, with the *POOL Service* managing shared POOL configuration and access to catalogues.
In this approach, the functionality of the POOL *Data Service* cache duplicates some of the functionality already present in the Athena/GAUDI object store machinery.
In Athena, both Tree-based and Key-based ROOT streaming format are supported and they can be selected in a job configuration file or script (*JobOptions* file).
The other POOL domains adopted by ATLAS and integrated in Athena are File Catalogues and Collections. The POOL File Catalogue interface has been integrated, and specific backends have been selected to serve different use cases: the XML Catalogue for local data access, EDG-RLS as a master catalogue. In controlled production, writing is separated from publication: output files are first registered in XML catalogues, and later published (often, after a QA step) in the master catalogue.
ATLAS is also using POOL ROOT and MySQL Collections to build tag databases, integrating them into Athena via *Registration Services* on output, and *Event Selectors* on input.

INTEGRATION IN CMS

The main offline analysis in CMS is driven by the Cobra framework [13]. POOL was adopted by Cobra at an early stage in POOL's development, and Cobra continues to integrate new releases.
POOL has been integrated in Cobra by replacing the functionality previously provided by Objectivity-based code. With respect to Objectivity [14], the ROOT-based POOL system has some limitations due to the file-based storage: it does not allow the concurrent update of a database from two processes, and it requires an additional mechanism (like RFIO [15] or dCache [16]) for remote access. However, the POOL system is less intrusive in the Data Model (it does not require a base class), it provides native support of STL Containers and enables the declaration of 'transient' specific object attributes.

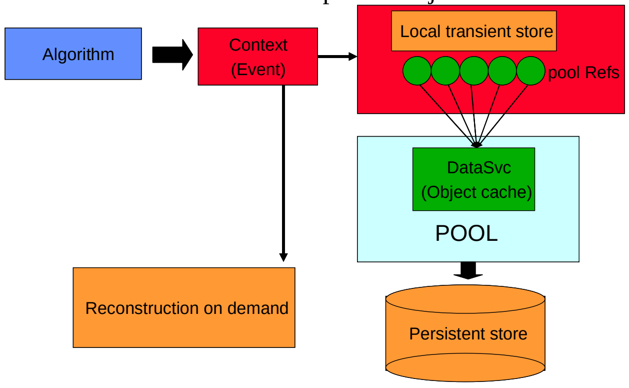

Figure 3 General approach adopted in the COBRA framework for the use of the POOL object storage system.

In Cobra the data related to a specific detector is retrieved through the current Event object. Objects in use are kept in the local *Transient Store*, which creates on demand and populates the object requested using POOL *References*. The access to and from the storage system is controlled through the POOL *Reference*, which also manages the object lifetime. In practice, since the object associations are defined through the POOL references, Cobra fully relies on the POOL navigation feature provided by the *Data Service*, with no other access mechanism.
In Cobra the POOL *Session* is only used for Transaction management, with no explicit handling of *Database* and *Containers*; the ROOT format adopted in Cobra is Keyed Objects.
Cobra has also adopted the POOL *File Catalogue*, mainly through the XML implementation in the physics application. In addition, both RLS and MySQL catalogues have been extensively used in production activities.

INTEGRATION IN LHCb

The LHCb core software is based on the Gaudi framework [12].
The integration of POOL in Gaudi has been implemented without changing the existing architecture or event model description.
In Gaudi transient data objects reside in *Data Stores*, which are sources for conversion to persistent or graphical representation. The client algorithms access objects by logical name from a data store.
The *Data Store* processes the storage and retrieval requests through the Gaudi *Persistency Service*, which acts as a technology dispatcher towards the underlying POOL *Persistency service*.

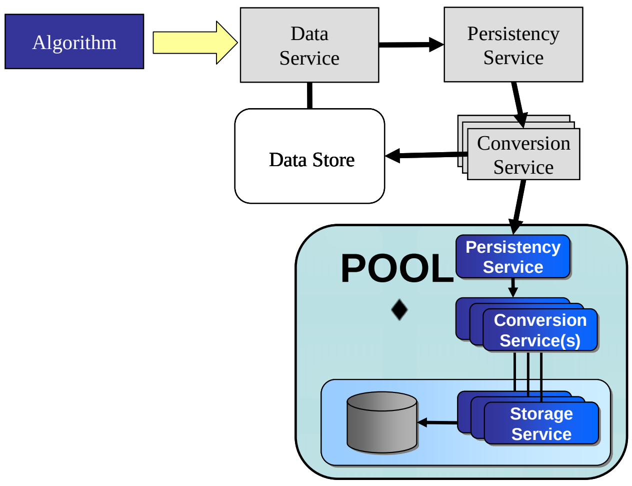

Figure 4 Integration of the POOL storage service components in the Gaudi framework.

In this approach, since the object bookkeeping is already performed by the Gaudi object cache, the POOL *Data Service* is not used.
Up to now, the focus of Gaudi persistency has been essentially Event data, stored with ROOT-Tree technology.
The POOL *File Catalogue* interface has been also successfully integrated in Gaudi, and used in production through the XML implementation.

COMMONALITIES AND DIFFERENCES

The main approach adopted by the three experiments in POOL integration has been driven by the need to minimize the impact on already existing offline code, taking care in some cases of preserve the ability to read data already written with previous technologies.
The requirements set by the three offline software architectures on the persistency framework are largely overlapping for the main tasks of the core Data Storage, while some diversity appears at higher level on how data is presented and exchanged. Some requirements are different in the area of Object Navigation and Object Lifetime Control, mainly because are imposed by the different offline framework architectures.

For this reason, while the POOL core object storage system has been adopted by the three experiments, the POOL components concretely integrated are not the same. For the storage of Event data, the three experiments have adopted the ROOT-based streaming, focusing on Tree technology (ATLAS and LHCb) or on Named Objects (CMS).
As seen above, the integration of the POOL object storage system has been implemented through different components.
ATLAS has integrated the POOL *Data Services* API (POOL *Reference*) in the Athena framework, adapting the loading-on-demand feature to the existing *Data Store*. In this way, the higher level POOL API is fully integrated, but the feature provided by the corresponding layer are in part not used.
CMS approach was essentially meant to replace the existing Objectivity-based services with POOL. Therefore the *Reference* based POOL API integrates neatly into the code, because it provides similar rules for object rending and navigation.
LHCb has chosen to integrate POOL through a lower level component, the *Persistency Service*. The main reason is that the Gaudi framework has its internal *Object Store*, which provides object bookkeeping, and navigation. Therefore, the functionality of the POOL *Data Service* is not required and the object I/O is handled through the inner layer.
Other POOL components have been integrated in the three experiment frameworks. The *File Catalogue* interface has been mostly adopted through the XML implementation, although the other backends have been also used in production activities (EDG-RLS by ATLAS and CMS, MySQL by CMS).

SUMMARY

During the last year the POOL persistency framework has been adopted by three LHC experiment (ATLAS, CMS and LHCb), integrated into their offline software and used in large-scale production activities. The transition of the pre-existing software to the POOL technology has been facilitated by the direct involvement of each experiment in the project. The POOL API has been fully validated and it has been demonstrated to provide a suitable solution for most of the requirements for production. The three integration approaches differ in the object bookkeeping area, because of the different requirements set by the existing frameworks.
All the POOL core components are currently used by at least one experiment.

REFERENCES

[1] The POOL Project,
http://pool.cern.ch
[2] The LHC Computing Grid
http://lcg.web.cern.ch
[3] D. Duellmann,
"The LCG POOL Project General Overview and Project Structure", CHEP 2003 Proceedings, MOKT007
[4] R.Brun and F.Rademakers, "ROOT-An Object Oriented Data Analysis Framework",
Nucl. Inst.& Meth. in Phys.Res.A389(1997)81-86.
see also: http://root.cern.ch
[5] M.Frank *et al*., "The POOL Data Storage, Cache and Conversion Mechanism".
CHEP 2003, proceeding, MOKT008
[6] Z.Xie *at al*., "POOL File Catalog, Collection and Meta Data Components",
CHEP 2003, proceeding, MOKT009
[7] D. Duellmann *et al*, "POOL Development Status and Plan", CHEP 2004 Proceedings
[8] The ATLAS experiment,
http://atlas.web.cern.ch/Atlas/Welcome.html
[9] The CMS experiment,
http://cmsinfo.cern.ch/Welcome.html/
[10] The LHCb experiment,
http://lhcb-public.web.cern.ch/lhcb-public/
[11] The Athena framework,
http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/index.html
[12] The Gaudi framework,
http://lhcb-comp.web.cern.ch/lhcbcomp/Frameworks/Gaudi/
[13] V.Innocente *at al*., CMS Software architecture: Software framework, services and persistency in high level trigger, reconstruction and analysis,
Computer Physics Communication, 140 (2001) 31-44
See also http://cobra.web.cern.ch/cobra/
[14] Objectivity database systems,
http://www.objectivity.com/
[15] The CERN Castor project,
http://castor.web.cern.ch/castor/
[16] The dCache project,
http://www.dcache.org/