EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH

CERN/DRDC/94-30
DRDC / P59
11 August 1994

A Persistant Object Manager for HEP

Predrag Buncic, Ryzsard Zybert
NA49

Gunter Folger, Jamie Shiers (spokesman)
CERN/CN, Geneva, Switzerland

Otto Schaile
OPAL Collaboration

Bill Greiman, Dennis Hall, Doug Olson, Craig Tull
Lawrence Berkely Laboratory, Berkeley, California, USA

Abstract

We propose to perform research in the area of a Persistant Object Manager for HEP. Persistant Objects are those which continue to exist upon process termination and may be accessed by other processes. It is expected that any system based upon this research will work primarily but not necessarily exclusively in an Object Oriented environment. Target applications include follow on or replacement products for existing packages such as GEANT, HEPDB, FATMEN, HBOOK, experiment specific code and event storage. In this respect, it is expected that more functionality will be required than simple object persistance. It will be one of the goals of the project to define this extra layer of functionality.

Strong emphasis will be placed on the use of standards and/or existing solutions where-ever possible.

1 Introduction

An essential component that was identified early on in the PASS project [1] was that of a persistant object manager. The PASS project wrote their own system, named Ptool [2], and also worked with a commercial solution, ObjectStore [3] from ObjectDesign Inc, which is available for numerous platforms.

In today's environment, data structure persistance is provided by packages such as ZEBRA [4]. ZEBRA, however, offers much more than just data structure persistancy and it is likely that many of these high level functions will still be required in the future.

This high level functionality is used to advantage by many of the current CERN Program Library packages, which rely heavily on ZEBRA, both as a memory manager but more importantly as an I/O package. Although 'modern' languages provide basic memory management features, e.g. allocation and deallocation of memory at run time, I/O of derived data types is still an unsolved problem in these languages, even on a single machine, let alone machine independent derived data type I/O. For these reasons it is clear that some sort of package will be required to fulfil a similar role to ZEBRA, and in which Object Oriented features are clearly desirable, if not mandatory. However, it is important to stress that the proposed package should not be considered as primarily a memory manager but as a flexible, scalable, object oriented I/O system.

This document is not an attempt to specify the proposed system. It is proposed that a much more complete document be produced in the first phase of the project, after several months' brainstorming.

1.1 Not a memory management system

Although systems such as BOS [13], CHEETAH [14], HYDRA [15], JAZELLE [16], YBOS [17], ZBOOK [18] and ZEBRA were principally considered as memory management systems, as stated above the principal role of the new system would not be that of memory management. Already in ZEBRA the memory management part is a relatively small fraction of the total - roughly 10% of the complete package. Applications such as KUIP, which previously used ZEBRA for memory management, have now been rewritten using the memory management facilities provided by the language. Others, such as FATMEN, use the memory management facilities of ZEBRA in as simple a manner as possible - basically to support a single derived data type. Common Fortran extensions (e.g. VMS structures),

Fortran 90 derived data types or C structs could have been used instead.

However, the powerful I/O capabilities of ZEBRA have proved essential, and permitted the construction of complex client-server systems.

1.2 Avoid over specifying the system

As there are many unknowns concerning the computing environment that will exist in the LHC era, care should be taken to avoid overspecifying the system at this early stage. The features cited in this paper are by no means an exhaustive list of what is required. Prototyping is obviously required and will constitute an important component of the proposed project.

1.3 Target applications

The proposed system should clearly be capable of solving the problems of large scale packages like GEANT [8], HBOOK [11], PAW [12] etc., experiment specific code and also client-server applications such as HEPDB [10]. It should be usable in both on- and off-line environments. The requirements of the above packages call for more than simple object persistancy. It will be one of the goals of the project to define this layer of functionality, matching it if possible to standard solutions. In today's terms, the proposed system should be capable of handling GEANT geometry databases, HBOOK histogram and Ntuple files, FATMEN file catalogues and HEPDB databases. However, it is important to stress that standard solutions will be sought and adopted where possible.

1.4 Object oriented features

The DSPACK [5] system, a data management system developed by the NA49 collaboration, introduces object oriented concepts that are available to the C or Fortran programmer. DSPACK supports the following OO concepts: polymorphism, methods and inheritance. In true OO style, the appropriate method is invoked by sending a message to an object. DSPACK may be used with or without ZEBRA and can certainly be considered as an interesting prototype for any future system. By offering OO features in both C and Fortran environments, it allows the benefits of OO to be demonstrated in a real experiment without the trauma of a complete change of programming language.

1.5 The PASS project

Much of the work performed in the PASS project is directly relevant to this area. Although it is unclear whether the PASS project will continue to be funded, we must clearly benefit from their experience and establish appropriate links with members of the PASS team. In particular, we wish to gain from their experience with CORBA [6] and CORBA compliant systems. Other important items in this area include the Object Database Standard, [7], which is also being closely followed by the MOOSE [19] project. [1]

However, given the importance of a persistant object manager, it is clear that strong support must be provided at CERN, particularly if any resulting package is to be used as a building block for other applications, as was true for ZEBRA.

1.6 Relationships with other projects

This project, if approved, would clearly have to establish strong links with other projects including MOOSE, the PASS project mentioned above and the proposed GEANT project [20]. One would hope for a free exchange of ideas, experience and requirements between the different groups.

2 Towards ZOO

The name ZOO has been proposed for the ZEBRA replacement, although OOZE may be more appropriate (Object Oriented ZEbra). This term will be used in the rest of this note, without implying any particular implementation of such a package.

We have many years of experience of both memory and I/O managers in HEP. These have proved extremely successful, whilst at the same time being criticised for having unfriendly, complex interfaces. [2] The latest generation of such packages provide significant enhancements over previous generations. Similarly, we can expect that any new package would be both more powerful and thus inherently more complex than existing packages.

[1] See URL http://www.nikhef.nl/www/pub/oord/meetings/coll-940714/minutes.html#io for more information.

[2] The correct balance between complexity and power is hard to achieve. "Novice" and "expert" APIs might be applicable, cf: VMS LIB$ routines and equivalent system services.

The required features of a memory manager are well understood. [3] However, in the future any such package should be able to cope with shared memory, MPPs with message passing libraries etc, and future unknown developments. We do not know what the computing environment will be like at the start of LHC but we do know that we can expect significant changes during its lifetime. Unless the new system is designed in such a fashion that it can evolve to match changes in the computing environment, it is unlikely to succeed.

2.1 A layer model

Given the existance of commercial persistant object managers today and their possible combination with mass storage systems, it makes sense to adopt a layer model such that components can be replaced with alternate solutions as appropriate. For example, one might interface to ZEBRA, Ptool and ObjectStore as I/O subsystems, eventually migrating to a standard solution in some years' time.

2.2 The components of ZOO

- Generalised data structure manipulation utilities (link lists, trees etc.)
- Dynamic creation of arbitrary objects
- File based I/O package (on both sequential and random media)
- Support for industry standards formats, e.g. Sun xdr, cpio format etc.
- Support for native language features, such as C structs, C++ classes, Fortran 90 derived data types.
- Exploitation of parallel file systems[4]
- Interfaces to packages such as ObjectStore and Ptool should be investigated.
- 'Object' based I/O package

[3] The functionality provided by *malloc()* and *free()* can be compared to that of e.g. *MZLIFT* and *MZDROP* and is clearly insufficient, unless each application writer is to develop their own set of higher level utility routines.

[4] Both the IBM and Meiko parallel file systems support a default mapping *per file system*. In the case of the IBM PFS, this can be overridden by the user for a given file.

- Simple debug package (a la DZEBRA)
- Support for DDL
- Interactive graphical debug and monitoring package
- Support for parallel processing using arbitrary communications mechanisms (shared memory, sockets, pipes, message passing libraries etc.)
- Client-server support
- Multi-language support
- Read access to existing ZEBRA FZ and RZ files (restricted to binary exchange format only).

2.3 Dynamically defined data types

The system must support the creation of arbitrary data types defined at run time. That is, the user must not be obliged to create a definition of a data type and compile and link in the code before it can be used.

2.4 OO support for dynamically defined data types

The system should support object oriented concepts, such as polymorphism, methods and inheritance, for all dynamically defined objects.

2.5 Run time identification of derived data types

A feature provided by the current systems but not by native languages is that of run time identification of derived data types. It is essential that arbitrary user-defined data types be creatable at run time and that these data types be self describing. This is particularly important for machine-independent I/O.

2.6 Machine-independent I/O

A great strength of the existing system is that of machine-independent I/O (exchange format). Files written in this format can be freely exchanged between machines, either through ftp (in binary mode) or via network file systems. [5]

The only file format that should be supported is the 'bitfile' - an unstructured stream of bytes with no system control words. On sequential media, this would be written as a series of fixed length records without control words. [6]

2.7 Support for native language derived data types

2.8 Direct access to data

One of the problems of existing systems, such as ZEBRA, is that certain data types, such as CHARACTER and DOUBLE PRECISION, cannot be accessed directly but must be copied in/out. This is both inefficient and inconvenient.

2.9 Read access to existing ZEBRA files

As was the case with ZEBRA, the new system should be capable of reading existing files. For practical reasons this should be limited to binary exchange format files of both RZ and FZ formats.

[5] ZEBRA defines exchange format as follows:

- 32 bit data, big endian
- IEEE floating point
- ASCII character set
- Two's complement

This is also the native format on most systems, although some, such as MS/DOS, Windows/NT, OSF and Ultrix are little endian. In the former case the conversion code simply disappears and is replaced by copies to/from the I/O buffers. In the latter case, the byte ordering is inverted at the lowest level in the I/O buffers.

[6] Which today implies C I/O on most systems.

2.10 Shared memory support

Multiprocessor systems with shared memory are now becoming common place. Although some packages layered on top of ZEBRA provided some level of shared memory support, notably HBOOK first on VMS and later also Unix systems, this was performed in a highly application specific manner.

It is important that support for shared memory be provided by ZOO directly in as efficient a manner as possible. This will be used both on single processors, e.g. for developing and debugging client/server applications, but also on multiprocessor systems such as the Silicon Graphics Challenge, the Convex Exemplar range etc.

2.11 Socket support (distributed processing)

Support for a less tightly coupled style of computing is also required. Typically, this might be achieved using sockets, or message passing libraries in an environment such as the IBM SP2. It is important to point out that the latencies involved imply a different style of computing to that in shared memory systems.

2.12 Efficiency through exploitation of language features

The current system involves a certain amount of inefficiency that could be avoided by the exploitation of features inherent to the language. This is particularly important on superscalar systems where significant performance hits can result from the compiler's inability to efficiently exploit the architecture.

Ideally, one would hope to obtain all of the required functionality from language features eventually coupled with standard products. Whereas this goal is unlikely to be attained at the startup of LHC, we can expect developments though its lifetime which will permit the replacement of various components by language features and/or standard products. This again emphasises the need for modularity.

2.13 Debugging aids

Powerful debugging tools are clearly required. As in the existing package, both line-oriented output and full graphical support will be required. The debugging tools will clearly also be an invaluable tool during the development stage.

2.14 Built-in verification and specification procedures

As software becomes ever more complex, so does the requirement to ensure that it is actually working correctly. The effort required to test the existing software is very large, and clearly does not scale to the LHC. It is therefore essential that verification procedures be designed in to the system so that automatic verification can be performed as part of the installation procedure.

2.15 Parallel file systems

A number of parallel file systems are now becoming available. In addition, file striping of ZEBRA RZ files has recently been introduced into PIAF. Exploitation of parallel file systems will be important for performance and efficient integration will be required e.g. so that optimum blocksizes etc. are chosen.

3 TIMESCALES

3.1 PHASE 1

Assuming that the project is approved, the first phase would be to involve various experts from both inside and outside CERN, CN, ECP and PPE in a serious of brainstorming meetings and discussions. This phase would be expected to last between 3 and 6 months and would develop a detailed report of some 50 pages to be presented to various committees (DRDC, COLLECT etc.)

This phase would include an analysis of the suitability of Ptool and commercial packages such as ObjectStore.

3.2 PHASE 2

Although phase 1 would also involve some prototyping, it is expected that the report would identify a number of areas where detailed prototyping would be required. Some of these areas may be competing, others complementary.

After approximately one year of prototyping, a detailed design document would be written.

The experience of NA49 with DSPACK would be an important component of this phase.

3.3 PHASE 3

Phase 3 would be the implementation phase in which the results of the two previous phases would be realised. Assuming that a system of at least the complexity of ZEBRA is required, this will probably last 1-2 years.

3.4 Interaction with other projects

At all stages, close interaction with other projects will clearly be required. As GEANT is expected to be one of the large customers of this work, they would also be expected to test early prototypes and provide feedback on required functionality.

4 Resources

Our financial requests to the DRDC amount to 100 KSF per year. In particular, the following resources are requested:

- Funds for trial licenses of appropriate software (e.g. ObjectStore): 50 KSF per year.
- Funds to pay for associates to work on the project at CERN as estimated below. Note that these numbers should be considered as the bare minimum if the project is to succeed. For PHASE 1 we request 50 KSF.

Phase 1 6 man months per year

Phase 2 12 man months per year

Phase 3 24 man months per year

5 Manpower

ZEBRA and its precursors required many man-years of development. ZEBRA itself is central to much of the software supported by the CN/ASD group and also to a very large fraction of experiment specific software. It is clear that this project will require a team of dedicated, hard-working, programmers. However, the long term support and maintenance issue should not be overlooked. This can only be handled by a permanent member of staff.

References

[1] Petabyte Access Storage Solutions. *The PASS Project Architectural Model.* Proceedings of CHEP94 (to be published).

[2] Ptool. *The Design and Evaluation of a High Performance Object Store.* Laboratory for Advanced Computing, University of Illinois at Chicago, March 1994.

[3] Object Design Inc., Burlington, MA. *ObjectStore Reference manual.*

[4] The ZEBRA Data Structure Management Package. *Overview of the ZEBRA System,* CERN Program Library Q100.

[5] DSPACK Object Oriented Data Manager. *DSPACK tutorial* CERN Program Library Q125.

[6] Object Management Group. *The Common Object Request Broker: Architecture and Specification, Revision 1.1,* OMG TC Document 91.12.1, 1991.

[7] *The Object Database Standard,* ODMG-93, Edited by R.G.G.Cattell, ISBN 1-55860-302-6, Morgan Kaufmann (publishers).

[8] GEANT detector simulation package, *Long writeup,* CERN Program Library W5013.

[9] FATMEN Distributed File and Tape Management System, *Long writeup,* CERN Program Library Q123.

[10] HEPDB database management system, *Long writeup,* CERN Program Library Q180.

[11] HBOOK Statistical analysis and histogramming package, *Long writeup,* CERN Program Library Y250.

[12] The Physics Analysis Workstation, *Long writeup,* CERN Program Library Q121.

[13] The BOS dynamic memory management system, *DESY Internal report R1-99-01,* DESY, Germany.

[14] The Cheetah Data Management System, *Proceedings of the 14th INFN Eloisatron Project: Data Structures for Particle Physics*, Erice, Italy, 1990.

[15] The HYDRA data structure package, *CERN TC Program Library*.

[16] Jazelle - an Enhanced Data Management System for High Energy Physics, *Proceedings of CHEP 90*, Santa Fe, New Mexico.

[17] The YBOS memory management system, *Programmers Reference Manual*, CDF Computing Group, Fermilab.

[18] The ZBOOK dynamic storage management system, *Long writeup*, CERN Program Library Q210.

[19] RD41 (MOOSE), *An Object Oriented Software R&D project.*

[20] DRDC proposal P58, *GEANT 4 : an Object Oriented toolkit for the simulation in HEP.*