A litmus test is the ability to support a reasonable subset of the Object Management Group's Query Services Specification with an interface that can be supported consistently by both a lightweight object manager and a true object database. We describe our efforts in this direction, and their connection with efforts to implement the OMG's persistence services specification, which offers a different (and in some ways, philosophically conflicting) view of how objects, apart from databases, manage persistence.

BR9737149

ABS_44

Data compression in the DELPHI experiment

N.Smirnov (Protvino and CERN)
E.Tcherniaev (Protvino and CERN)

An application of general data compression methods to the data involved in the physics analysis at the DELPHI experiment is considered. The main goal of this is to save disk space without essential changes in the data processing chain.

The DELPHI data processing chain consists of the following experimental and simulated data types: RAW data, Full DST, Long or Leptonic DST, Short DST, and mini DST. It is clear that the most essential data for physics analysis (LDST,SDST and mDST) should be located on disks. At the present time this requires 250 Gbytes of disk space. The 1995 data will require approximately the same space. Such an amount of information produces definite difficulties even for large computer centres like the DELPHI off-line analysis centre at CERN, and for home labs it can be a real problem to keep all the information on disks.

One of the resonable ways to solve this problem is an application of generaldata compression methods. Such an approach has been implemented in the scope of the PHDST I/O package, which is being developed for the DELPHI experement to provide a user-friendly access to the data with computer-independent specification of external media.

The PHDST package uses the ZEBRA memory management system to manipulate internal data structures and for computer-independent input/output. The implementation of data compression in PHDST is essentially based on a possibility of the ZEBRA package to read/write data not only from external media (disk, tapes) but also from the internal memory of the program. Such a possibility allows to introduce the data compression in very natural way without visible changes in the user interface. For the user it is enough just to relink his program with new library to be able to work with compressed data.

We considered several data compression methods as candidates to be used in PHDST, but the final choice was more-or-less evident: it is the deflate/inflate method available in GZIP and some other programs. Based on the GZIP's sources two routines were implemented for in-memory compression/decompression, which are suitable for use inside a FORTRAN program.

In addition to the technical details of the realisation of data compression in the PHDST package, the article contains several tables with I/O timing and compression ratios for different kinds of data. The compression ratio varies between 3050possibilities for further improvement of data compression are also discussed.

BR9737150

ABS_31

Data Analysis in an Object Request Broker Environment

D. Malon (Argonne)
E. May (Argonne), C. Day (LBL), D. Quarrie (LBL), R. Grossman (Chicago)

Computing for the Next Millenium will require software interoperability in heterogeneous, increasingly object-oriented environments. The Common Object Request Broker Architecture (CORBA) is a software industry effort, under the aegis of the Object Management Group, to standardize mechanisms for software interaction among disparate applications written in a variety of languages and running on a variety of distributed platforms. In this paper, we describe some of the design and performance implications for software that must function in such a brokered environment in a standards-compliant way. We illustrate these implications with a physics data analysis example as a case study.

The promise of brokered-request object architectures is alluring. My software will talk to your software, even if I know neither in what language your software is written, nor where it runs. The idea is this: no matter what language you use to implement your software, you describe its interface in a single, application-language-neutral Interface Definition Language (IDL), and place an interface description in a repository. You then register your implementation so that it can be found by system utilities.

When I wish to invoke your software, I use standard utilities to find its interface, and pass my request to an Object Request Broker (ORB). The ORB looks for a server capable of handling my request ,its location may be transparent to me. The ORB may instantiate such a server if none is already running. The

ORB then forwards my request to your software and returns any results, handling the language mapping at both ends.

Services commonly required by many objects-lifecycle services, persistence services, query services, and others are the subjects of standardization specifications as well.

Is this environment appropriate for high-performance physics applications? If the physics community ignores these approaches, does it do so at its own peril?

Among the questions that must be addressed are these:

o Is the Interface Definition Language rich enough to capture the interfaces required by data-intensive physics applications?

o Is the performance penalty of brokered interactions inherently too great?

o Can we use an ORB simply to connect our applications, and then get it out of the way?

o If the ORB does get out of the way, do we lose language-independence, and are we back to home-grown low-level interfaces?

o What is the appropriate level of granularity for brokered interactions?

o The potential location transparency provided by an ORB is appealing, but will performance considerations require that I provide a "smart proxy" to run on your machine when you invoke software on my machine, in order to sustain brokered interactions at a reasonable cost?

o If so, is proxy support a nightmare for providers of general-use software, or can proxy generation be standardized or automated?

o What are the implications of proposed persistence services specifications in this environment?

We explore these and other issues in a case study, in which we use commercially available request brokers in an examination of a variety of potential implementations of a statistical computation on physics data extracted from a persistent data store.

BR9737151

ABS_2

RD45 - Object persistency for HEP - Status report

JAMIE SHIERS (CERN)

RD45 is a CERN project to investigate the issues concerning the storage and manipulation of a persistent objects for LHC era HEP experiments. Objects are typically created by a given process and cease to exist when that process terminates. Such objects are called Transient objects. Persistent objects, on the other hand, are those which continue to exist upon termination of their creating process.

There are a number of existing efforts committed to the exploration of Object Oriented (OO) techniques in a wide variety of key areas for HEP computing in the LHC era. All of these projects will need to handle persistent objects but none of them are addressing this question directly.

The goal of the RD45 project is to address the requirements of these projects and of the LHC experiments themselves in terms of object persistency.

An important theme of the project is the use of standards. We start by identifying the standards involved, we explain their interaction and examine their suitability for HEP computing environments. We then describe the various prototyping activities that we have undertaken and present the results that have been obtained so far. Finally, we examine future directions and discuss the impact of the proposed technologies on HEP computing in general.

BR9737152

B.3 DATA MANAGEMENT AND DATABASES

ABS_116

Experience using a distributed Object Oriented Database (OODB) for a DAQ system

B. JONES (CERN)

To configure the RD13 data acquisition system, we need many parameters which describe the various hardware and software components. Such information has been defined using an entity-relation model and stored in a commercial memory-resident database. During the last year, Itasca, an OODB, was chosen as a replacement database system. We have ported the existing databases (hw and sw configurations, run parameters etc) to Itasca and integrated it with the run control system. We believe that it is possible to use an OODB in real-time environments such as DAQ systems. In this paper, we present our experience and impression : why we wanted to change from an entity-relational approach, some useful features of Itasca, the issues we meet during this project including integration of the database into an existing distributed environment and factors which influence performance.