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