EXPERIENCE USING A DISTRIBUTED OBJECT ORIENTED DATABASE FOR A DAQ SYSTEM

C.P. Bee[a], S. Eshghi[b], R. Jones, S. Kolos[c], C. Maidantchik[d], L. Mapelli[e], G. Mornacchi[f], M. Niculescu[j], A. Patel[g], D. Prigent, R. Spiwoks[h], I. Soloviev[c.]
CERN, Geneva, Switzerland

M. Caprini[i], P.Y. Duval, F. Etienne, D. Ferrato, A. Le Van Suu, Z. Qian,
Centre de Physique des Particules de Marseille, IN2P3, France

I. Gaponenko, Y. Merzliakov
Budker Institute of Nuclear Physics, Novosibirsk, Russia

G. Ambrosini[j], R. Ferrari, G. Fumagalli, G. Polesello
Dipartimento di Fisica dell'Universita' e Sezione INFN di Pavia, Italy

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 object oriented database management system (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.

1 Introduction

A data acquisition system needs a large number of parameters to describe its hardware and software components. The RD13 DAQ [1] system currently uses four databases to store such information: Hardware configuration (describes the layout of the hardware in terms of crates, modules, processors and interconnects); Software configuration (describes the layout of the software in terms of processes, services provided, connections and host

a. Now at University of Zurich, Switzerland.
b. Now at University of Utrecht, The Netherlands.
c. On leave from the Petersburg Nuclear Physics Institute, St. Petersburg, Russia
d. Also with Federal University of Rio de Janeiro, Brazil.
e. Spokesperson.
f. Contact Person.
g. On leave from School of Computing and Information Systems, University of Sunderland, UK.
h. Also at the University of Dortmund, Germany.
i. On leave from the Institute of Atomic Physics, Bucharest, Romania.
j. Now at University of Bern, Switzerland.

machines); Run parameters (e.g. run number, recording device, Level 2 trigger state etc.) and Detector parameters (information pertaining to the detector and defined by the detector group themselves within a fixed framework).

The current version of the DAQ databases are implemented with the Quid [2] commercial in-memory entity-relationship database. Even though Quid satisfies the main requirements, there are some important restrictions:

- the whole database is in the memory of the application and this sets a limit on the database size;
- there is no reference integrity control, hence dangling objects may appear;
- there is no concurrency control, hence the consistency of the database is in danger when several processes access the database file and one or more processes have write access;
- there is no facility for restoring the data after schema modification;
- creation of new entities in the database from the Quid application programming interface (API) is cumbersome;
- there is no versioning control of database contents.

For these reasons a more sophisticated distributed object database management system was selected as a replacement.

2 Itasca object-oriented database (OODB)

Itasca [3] is a commercial client-server architecture database that runs on UNIX platforms with a server engine based on Common LISP. APIs exist for LISP, C and C++. Advanced functions supported include object versioning, change notification, composite objects, clustering, indexing, dynamic schema evolution and multimedia data management. We run the database server on a Sun workstation and use the client APIs for access from other workstations (including HPs) and front-end processors (running LynxOS or EP/LX).

After performing some initial tests and evaluations[4][5], we have produced Itasca based implementations of the schema and access libraries for the DAQ configuration databases described above. We relate our experiences during this work and outline the techniques and facilities employed.

2.1 Schema creation

Itasca does not provide a graphical schema editor but there are several means by which a developer can create a database schema (see Figure 1): the *Dynamic Schema Editor* (DSE) tool which allows the user to interactively manipulate class definitions; the server's LISP interpreter interface or the client APIs. The DSE does not require any programming but it is more cumbersome to use if many classes and relationships are to be defined or manipu-

lated. We have used either the DSE or chosen to write C client programs to create the database schemas.

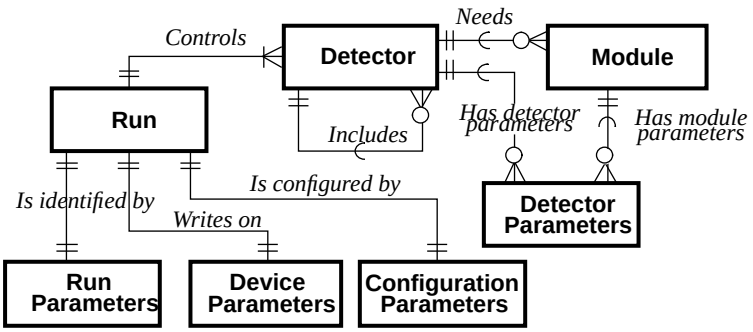

Figure 1. Run Control and Detector Parameters database schema in Itasca (Martin/Odell notation [10])

2.2 Performance

It is clear that moving from an in-memory system (Quid basically implements C structures and uses pointers for addressing) to a client/server architecture using a local-area network has performance overheads and we have used several facilities provided by Itasca (described below) to counteract this effect. The DAQ applications access the database via an implementation independent library that allowed us to swap between Quid and Itasca without modifying the application code. However, since Quid was a very fast (though not very flexible) in-memory system, some optimization of the applications was necessary in order to preserve performance (for example, by reducing the number of calls to the access library.)

The organisation of the machines on which a client-server OODB runs has a major effect on performance of the applications. Network delays mean that applications running on the server machine execute far more quickly and the location of the disk used for data storage (i.e. local to the server machine or mounted via NFS) also affects access time. LISP functions that execute in the server are much faster than making several message exchanges from the client API. We have implemented the most common and time consuming operations performed by the applications as LISP functions.

2.3 Versioning

The Itasca object versioning mechanism does not support updating composite objects which are used in the DAQ database schemes. In order to store multiple DAQ configurations we created a super class (inherited by all DAQ component classes) which adds a list attribute representing the set of DAQ configurations in which an instance is used. In this manner we can identify which instances are used in each DAQ configuration and it has the advantage of saving space since we only need to make a new instance when its contents change. Clustering of composite objects in the same segment to minimize disk input/output was also used when the schema permitted.

2.4 Locks

Locks are placed on Itasca objects to guarantee that concurrent access does not result in corruption of data. To enforce this, Itasca employs a pessimistic locking scheme and supports two type of object locks: shared locks (for read operations) and exclusive locks (for update operations). When a process attempts to update an object, but the exclusive lock cannot be granted because of lock contention, a wait period ensues. If the contention is relieved during this wait period, the exclusive lock is granted and the update takes place. To reduce the number of lock conflicts, Itasca provides a dirty read capability that allows one transaction to read a snapshot of an object while allowing another concurrent transaction to update the object.

We have used this facility in the data access library so an application enables dirty read mode at start-up and disables it before attempting to modify the database contents. Only one application can modify the database at any given time; other applications receive a lock error when attempting to modify the same database. But any number of applications can read the database while one application modifies the database contents.

2.5 Client and Server Caches

Itasca implements a cache in the server for frequently accessed data and a client-side cache in each client process (on request). Most of the DAQ databases tend to be used at the initialization phase of the applications and the database contents are read with very few modifications during a DAQ run.

2.6 Integration of Itasca with other packages

Our initial work involved reproducing the functionality of Quid with Itasca by making a new implementation of the application defined data access library functions in order to prove that Itasca could provide the same services. This work uncovered some integration problems especially those related to use of a synchronous message based client/server architecture in the Itasca API. The DAQ run control applications use ISIS [6] for interprocess communication and were designed for an asynchronous, multi-threaded environment. Calls to the Itasca API imply synchronous Remote Procedure Calls (RPCs) to the server and thus process blocking delays that proved incompatible with ISIS timing constraints. We also had problems related to the overloading of UNIX signals by the various packages and the bad chaining of the signal handlers they provide. We had nevertheless to keep some concurrency protection inside the application because of UNIX SIGIO signal handlers. This application level debugging phase proved difficult due to the asynchronous and concurrent aspects of the applications for which traditional debugging tools were inadequate. Nevertheless, these problems have been overcome and applications that integrate Itasca with ISIS and other packages can now run on all the supported platforms. We have developed applications that automate the task of porting our data from Quid to Itasca, dumping the database contents and managing DAQ configurations within Itasca.

2.7 New applications

We have developed a number of new applications that use Itasca directly without attempting to provide compatibility with the old Quid implementations and do not use the data access library described above. The most important of these applications is the DAQ message logger and browser. This application captures all the messages generated by all the components of the DAQ during a run and stores them using Itasca. For performance reasons and so that more than one user can browse the database simultaneously, the application is split into two parts: the *message loader* which receives messages from the DAQ and stores them in Itasca; and the *message browser* which allows the user to browse the contents of the database using a graphical user interface. The message loader and browser were developed using the C client API for Itasca. The message browser GUI code was developed using the X-Designer [7] user interface builder and the XRT table widget [8].

As an evaluation, we have developed a prototype application that provides a link between the Object Management Workbench (OMW) CASE tool [9] and Itasca. The motivation for this prototype was to see if an alternative technique for object persistency could be used with OMW and also to provide a graphical schema editor for Itasca. The prototype is implemented as an OMW application that uses the tool's library to access information stored on object diagrams (representing class structures and their relationships) and Itasca's C client library to access corresponding definitions in the database. The prototype can define classes with attributes and relationships, store OMW object instances in Itasca and reload instances from Itasca to OMW.

3 Assessment and future work

Itasca has provided a very useful and convenient means for investigating the application of an OODB to HEP DAQ systems. We have learnt much about the use of an OODB, its advantages and limitations and we are certain that most of this knowledge is valid for other commercial OODB as well. We will continue to use Itasca in the immediate future by extending its use within the DAQ. We will also classify all the types of data access that are required by the DAQ in an attempt to identify areas for which a fully distributed database is necessary against those which are read-only, not shared or so dynamic that the overheads involved in using a database would be a hindrance to the performance. Nevertheless, there are features such as dynamic object creation, change notification and highly concurrent access that still need to be evaluated and which may prove useful for our applications.

Several projects (Pass [11] [12], RD45 [13], Cicero [14] etc.) are also investigating the use of OODB but for different aspects of HEP software. The projects are using a mixture of commercial and privately developed database systems. We think it is important that this work continues and that it is too early to attempt to standardise on one single product. The ODMG [15] is bringing into existence a standard for defining the facilities and interfaces an OODB should provide so that, when the need arises, users can move their data and applications to another product. It is clear that some system offering facilities equivalent to those provided by an OODB will be needed for LHC style experiments and that such a system must be available to all areas of the experiment (DAQ, trigger, off-line etc.)

Given the financial, integration and training investments at stake it is important to try and bring all this research work together to provide a consistent persistent data service for the whole experiment.

References

Information on the RD13 project and the tools discussed are available through the World Wide Web at URL http://rd13doc.cern.ch/.

[1] L.Mapelli et al., A Scalable Data Taking System at a Teatbeam for LHC, CERN/DRDC 90-64, CERN/DRDC 91-23,CERN/DRDC 92-13, CERN/DRDC 93-25. CERN/LHCC 95-47

[2] G. Bruno, Model Based Software Engineering; Chapman & Hall, 1995. ISBN 0 412 48670 9.

[3] Itasca Distributed Object Database Management System. Technical Summary for Release 2.1. Itasca Systems, Inc. 1992.

[4] M. Skiadelli, Object Oriented database system evaluation for the DAQ system. RD13 Technical Note 108. Available through WWW at “http://rd13doc.cern.ch/”.

[5] G.Ambrosini et al. OODBMS for a DAQ system, pp 143 Proceedings of CHEP94. Lawrence Berkeley Laboratory LBL-35822; CONF-940492; UC-405.

[6] K.P.Birman and Robert Cooper, The ISIS project: Real experience with a fault tolerant programming system. European SIGOPS Workshop, September 1990; also available as Cornell Univ. Computer Science Dept. Tecn. Report TR90-1138.

[7] Imperial Software Technology. X-Designer User Manual.

[8] KLC group Inc., XRT Builder Guide & Reference Manual, Ref No. BLGDE-GRAPH/M/240-07/94.

[9] See “Applications of an OO Methodology and CASE tool to a DAQ system” in these proceedings.

[10] OOIE - Object Oriented Methods A Foundation, James Martin and James J. Odell, Printice Hall, ISBN 0-13-630856-2

[11] Petabyte Access Storage Solutions. The PASS Project Architectural Model. Proceedings of CHEP94. Lawrence Berkeley Laboratory LBL-35822; CONF-940492; UC-405.

[12] Ptool. The Design and Evaluation of a High Performance Object Store. Laboratory for Advanced Computing, University of Illinois at Chicago, March1994.

[13] J.Shiers et al. A Persistent Object Manager for HEP, CERN/DRDC 94-50

[14] J.M. Le Goff et al. CICERO: Control information system concepts based on encapsulated real-time objects. CERN/DRDC/93-50,CERN/LHCC/95-15

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