The OKS Persistent In-memory Object Manager

R. Jones[1], L. Mapelli[1], Yu. Ryabov[2] and I. Soloviev[1,3]

[1] CERN, European Laboratory for Particle Physics, Geneva 23, Switzerland, CH-1211

[2] PNPI, Petersburg Nuclear Physics Institute, Gatchina, Leningrad district, Russia, 188350

[3] on leave from the PNPI

Abstract

The OKS (Object Kernel Support) is a library to support a simple, active persistent in-memory object manager. It is suitable for applications which need to create persistent structured information with fast access but do not require full database functionality. It can be used as the frame of configuration databases and real-time object managers for Data Acquisition and Detector Control Systems in such fields as setup, diagnostics and general configuration description.

OKS is based on an object model that supports objects, classes, associations, methods, inheritance, polymorphism, object identifiers, composite objects, integrity constraints, schema evolution, data migration and active notification. OKS stores the class definitions and their instances in portable ASCII files. It provides query facilities, including indices support. The OKS has a C++ API (Application Program Interface) and includes Motif based GUI applications to design class schema and to manipulate objects.

OKS has been developed on top of the Rogue Wave Tools.h++ C++ class library [1].

I. INTRODUCTION

OKS was designed at the Information Technology Department of Petersburg Nuclear Physics Institute. An experience that has been received during design of knowledge based system [2] and it's use [3] had shown that for new generation physical system with a lot of configuration and knowledge data, a true persistent object manager should be used. Such manager must support data capable of describing configuration and knowledge, satisfy soft real-time requirements (i.e. high performance but without time guarantees for transaction completion) and be available on various operating systems for different C++ compilers. Various shareware and commercial data persistent managers (databases, object managers, including ORACLE relational database management system [4], Quid object manager [5], ITASCA distributed object database management system [6], ODE in-memory object-oriented database management system from AT&T Bell Labs [7], YOODA object-oriented database [8] and Rogue Wave Tools.h++ C++ persistent objects [1]) were evaluated but no single system satisfied all the requirements [9], [10]. By this reason it was decided to start a new project to create an object manager (called OKS) capable of fulfilling our needs.

The work with OKS was stared in January, 1996 and the first prototype was implemented in July, 1996. At that time the ATLAS DAQ working group [11] at CERN found similar needs for a persistence data manager to implement configuration databases and OKS was adopted [12].

To satisfy extra requirements from the ATLAS prototype DAQ [13] a second prototype of OKS was implemented that supported multi-schema and multi-data files. Later missing base data types (boolean and bit vectors), dynamic data modification in case of schema changes, queries and a full documentation set (User Guide, Tools Manual, Reference Manual and a tutorial) were added to OKS.

This version of OKS was successfully tested in the ATLAS DAQ prototype -1 environment on several platforms (including Sun OS, Sun Solaris, HP-UX, Windows 95/NT, Lynx OS for Power PC, etc.) and with prototypes of future DAQ applications.

II. OKS DESCRIPTION

This section describes the object model used by OKS, architecture, API and related tools.

A. OKS Object Model

The OKS includes the following entries:

- the basic entity is an **object** which can be named and assigned a unique identity (**object identifier**),
- objects with common properties and behavior are described by one **class**,
- object properties are defined by the set of **attributes** and **relationships**,
- **methods** to act on object properties (e.g. to implement consistency constraints),
- **inheritance** (multiple, i.e. an OKS class can have more than one superclass),
- **polymorphism** (overloading of inherited object properties in a child class),
- **composite objects** (i.e. an object built from dependent child objects),
- **integrity constraints** (i.e. type and value restrictions on attributes and relationships),
- **schema evolution** (i.e. allow modifications to the schema as the application evolves),
- **data migration** (i.e. permit data to be accessed by successive versions of a schema),
- **active notification** (execute a set of callbacks when a class or instance is created/deleted/modified).

An example application using many of the above entries is shown in section III.

B. OKS Architecture

The OKS stores schema and data in separate (multiple) files to simplify schema evolution, data migration and allow partial loading of a database as required.

0018-9499/98$10.00

The OKS kernel is responsible for loading database schema and data files. It is possible to store dependent OKS classes (e.g. from the same inheritance hierarchy) in different schema files. The minimal portion of information stored in a schema file is an OKS class. It is possible to store dependent OKS objects (e.g. as part of a composite object) in different data files. The minimal portion of information stored in a data file is an OKS object. The schema and data files have a portable ASCII format which can be used across different platforms.

The OKS kernel keeps two lists of loaded schema files and loaded data files and two hash tables of defined OKS classes and their instances. A schema file contains a list of defined OKS classes and a data file contains a list of defined OKS objects. An OKS class contains lists of superclasses, attributes, relationships, methods and a hash table of instances. An instance of OKS class (i.e. OKS object) has an array of OKS data structures that can be of built-in types (boolean, character, signed/unsigned short/long integer, float, double, date, time, string, bit array), enumeration type, a reference to an OKS object, or a list of them which are used to store the values of instance attributes and relationships. The in-memory relationships between the described classes are represented below in Figure 1 using OMT notation:

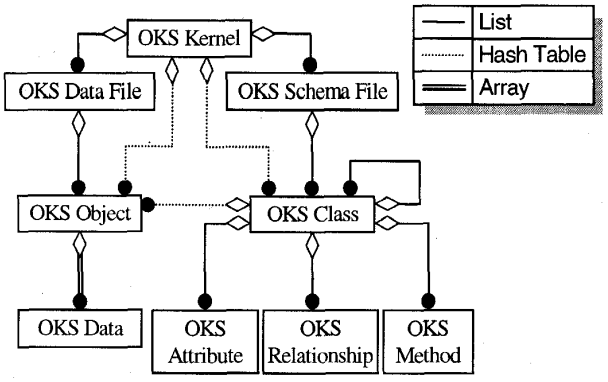

Figure 1: The schema of relationships between OKS classes

The OKS supports queries and indices. A query can be created using the OKS API or read from an ASCII string. An index provides faster access to objects and it can be created for a single attribute of a class.

Since OKS is an in-memory object manager with an emphasis on performance and portability rather that a full ODBMS, the following facilities are not supported:

- distributed access (other than provided by the file system),
- concurrent access control,
- failure recovery,
- transaction management is limited to data saving ("*commit*") and reloading ("*abort*").

The size of an OKS database is limited to the size of the operating system virtual memory allocated to a single process.

C. OKS API

The OKS provides a C++ API. It is a library of classes that can be categorized into three groups:

- ***OKS Kernel*** class manipulates schema and data files, sets notification callback functions for any change of a class or an instance, manipulates classes and dumps contents of OKS kernel.
- ***OKS Schema*** group of classes manipulates OKS classes, class attributes, relationships and methods, classes hierarchy, instances of class and executes query.
- ***OKS Data*** group of classes manipulates OKS objects and sets a list of notification callback functions to change value of attribute or relationship for any existing object.

An example using many of the API calls is presented in section III.

D. OKS Tools

OKS tools use the OKS library. OKS editors allow the graphical tabular manipulation of data and schema but do not provide a diagrammatic representation of the schema layout.

1) OKS Data Editor

The OKS Data Editor provides an interactive Motif based GUI to graphically manipulate objects stored in the OKS data files. An object can be inspected and edited visually. The Data Editor has a graphical query constructor. A view of the OKS Data Editor GUI is shown in Figure 2:

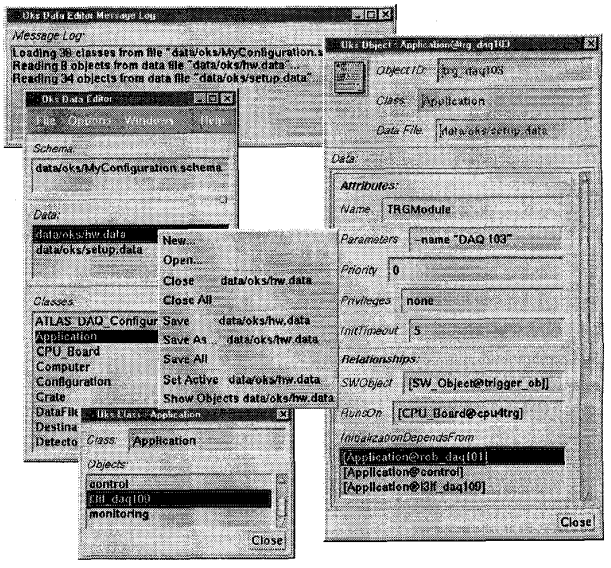

Figure 2: The OKS Data editor with loaded schema file and two data files, message log, OKS class and OKS object dialog boxes.

2) OKS Schema Editor

The OKS Schema Editor provides an interactive Motif based GUI to create, browse and modify OKS schema database files. It is the simplest way to define an OKS database and does not require programming by hand. The Schema Editor has a main window which presents information about loaded database schemes and classes, a log window which lists the load-time and run-time error and information messages generated by the OKS kernel, schema windows with

defined classes, class windows that describe class properties, attribute windows, relationship windows and method windows. A view of OKS Schema Editor GUI is shown in Figure 3:

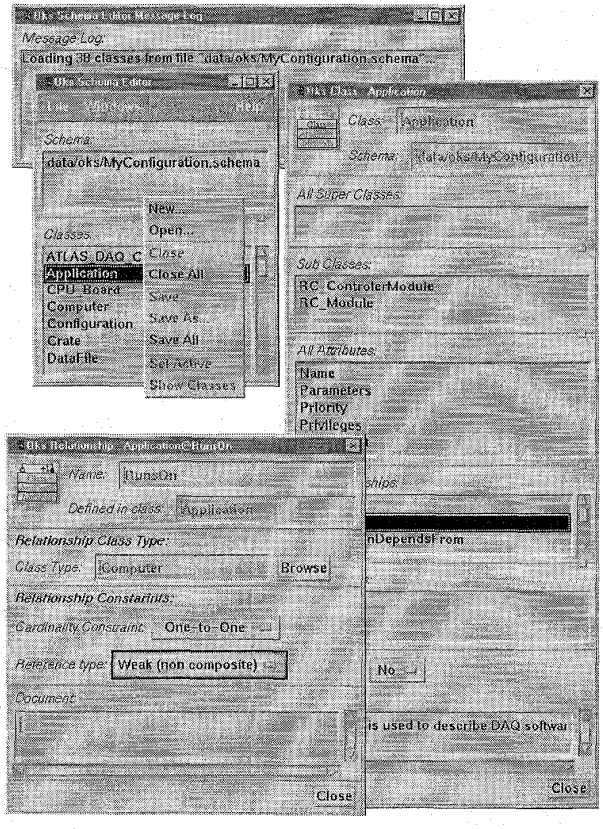

Figure 3: The OKS Schema Editor with loaded schema file, message log, OKS class and OKS relationship dialog boxes.

3) *Tool to compare OKS schema files*

The OKS schema files comparer (like UNIX diff) identifies differences for attributes, relationships, methods, superclasses and produces a list of all sub classes for differed classes. This is a useful tool when different database schemes need to be combined.

4) *Tool to compare OKS data files*

In a similar manner, the OKS data files comparer identifies a list of differences for object's attributes, relationships and produces a list of object composite parents for differed objects.

5) *Tool to dump the contents of schema and data files*

The OKS kernel dump prints the contents of a loaded OKS kernel. This allows the manual offline checking of database contents.

E. OKS Query

OKS provides a query language (using a lisp based syntax supporting boolean binary comparator functions with attributes, disjunction or conjunction of a number of queries, **and**, **or**, **not** keywords and queries through relationships) that retrieves objects based on selection criteria. An OKS query can be created using the OKS API or read from an ASCII string. OKS Data Editor can graphically create a query and save it in string format. An example query created by the Data Editor is shown in Figure 4 (it uses the database schema presented in section III):

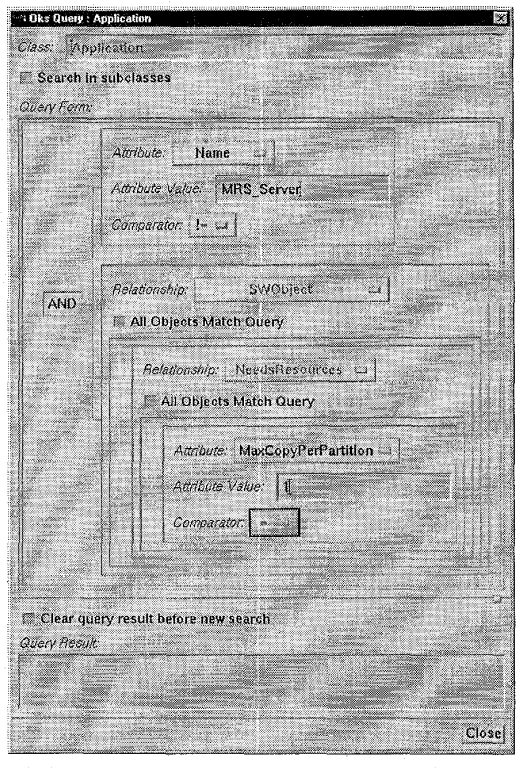

Figure 4: An example of OKS Data Editor constructor query dialog

The example query searches applications that are not *MRS_Server*, and must be unique for a partition (i.e. a requested application has an association with a software object that needs resources with at most one copy per partition). This query can be written as an ASCII string:

```
(all
  (and
    ("Name" "MRS_Server" !=)
    ("SWObject" all
      ("NeedsResources" all
        ("MaxCopyPerPartition" 1 =)
      )
    )
  )
)
```

III. USE OF OKS FOR ATLAS DAQ PROTOTYPE -1 CONFIGURATION DATABASES

The DAQ Configuration Databases are one of the software components of the ATLAS Trigger/DAQ software [14]. A DAQ needs a large number of parameters to describe its system architecture, hardware and software components, operation modes and status.

A. Atlas DAQ Prototype –1 Configuration Databases Requirements

The user requirements for the configuration databases are defined in the ATLAS DAQ Back-End Software User Requirements Document [15].

These requirements include support for the definition of data schemes with user-defined types. An application must be capable of accessing multiple schemes simultaneously. It must be available on a variety of platforms (including UNIX, Windows NT and real-time UNIX kernels such as Lynx OS) with schemes and data portable between the various architectures. The size of the data sets for configuration information is in the megabyte range and is mostly read-only during operation (but updated offline). In the case of real-time kernels running on embedded processors it is important to avoid the overhead of transferring data via remote servers and lock processes for performance reasons.

B. Atlas DAQ Prototype –1 Configuration Databases Architecture

1) Two-tier architecture

It was decided to adopt a two-tier architecture, using a lightweight in-memory persistent object manager to support the soft real-time requirements and a full ODBMS (Object Database Management System) as a back up and for long-term data management. The two-tier architecture schema is shown in Figure 5:

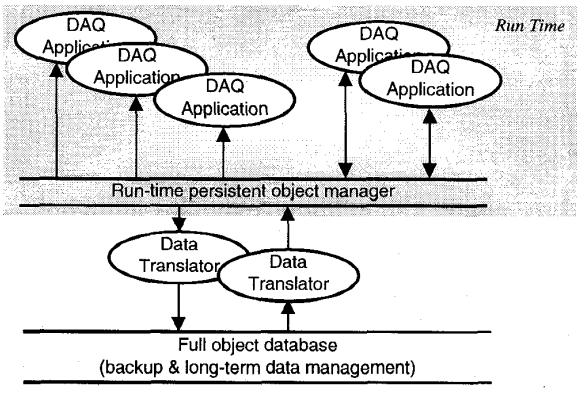

Figure 5: Two-tier Architecture for ATLAS DAQ Prototype -1 Configuration Databases

The ODBMS offers many of the long-term management facilities required such as data versioning, schema evolution, and authorization control while the in-memory object manager provides better performance and portability to embedded processors running real-time operating systems.

Translators are being developed to move data between the in-memory object manager and the ODBMS.

OKS has been adopted as the in-memory persistent object manager and Objectivity/DB [16] as the full ODBMS.

2) Configuration Databases

The ATLAS DAQ prototype-1 configuration databases use several interconnected object schemes:

- ATLAS-DAQ Configuration,
- Setup,
- Detector Parameters,
- Software,
- Hardware,
- Run Control,
- Data Flow,
- General Run Parameters.

Two schemes (Software and Setup configuration database views) are shown in Figure 6. The description of its basic structure is given below.

The *SW_Object* class is used to describe a software object from an abstract point of view; i.e. gives the name, description, default parameters and environment. An instance of *SW_Object* class has one or more implementations (i.e. programs) and may use software resources. A set of User objects (i.e. DAQ human operators) having the necessary permissions and privileges to run an implementation of this software object can be defined.

The *Program* class is used to describe an implementation of a software object. A program has a host operating system type, a name and location of its executable file, and platform specific parameters and environment variables.

The *Process* class is used to complete the description of a software object at run-time, i.e. logical name of the process, process id, time started, allocated process priority and privileges. It keeps lists of allocated resources and a reference to the host computer where it is running.

The *Partition* class acts as a container to describe all the hardware and software needed to run the DAQ and allows the DAQ to support multiple concurrent data taking activities.

The *Application* class is used to describe an application that has to start inside a partition. It maps software objects and computers (the same software object can be started several times as different processes and on different computers). An application can have a list of parameters. The possibility to start or shutdown an application can depend on other applications (e.g. a server has to be started before its clients).

The *Computer* class is used to describe a computer (PC, workstation or embedded processor), that can execute processes and where it is possible to start applications. It describes basic parameters such as host name and type of operating system.

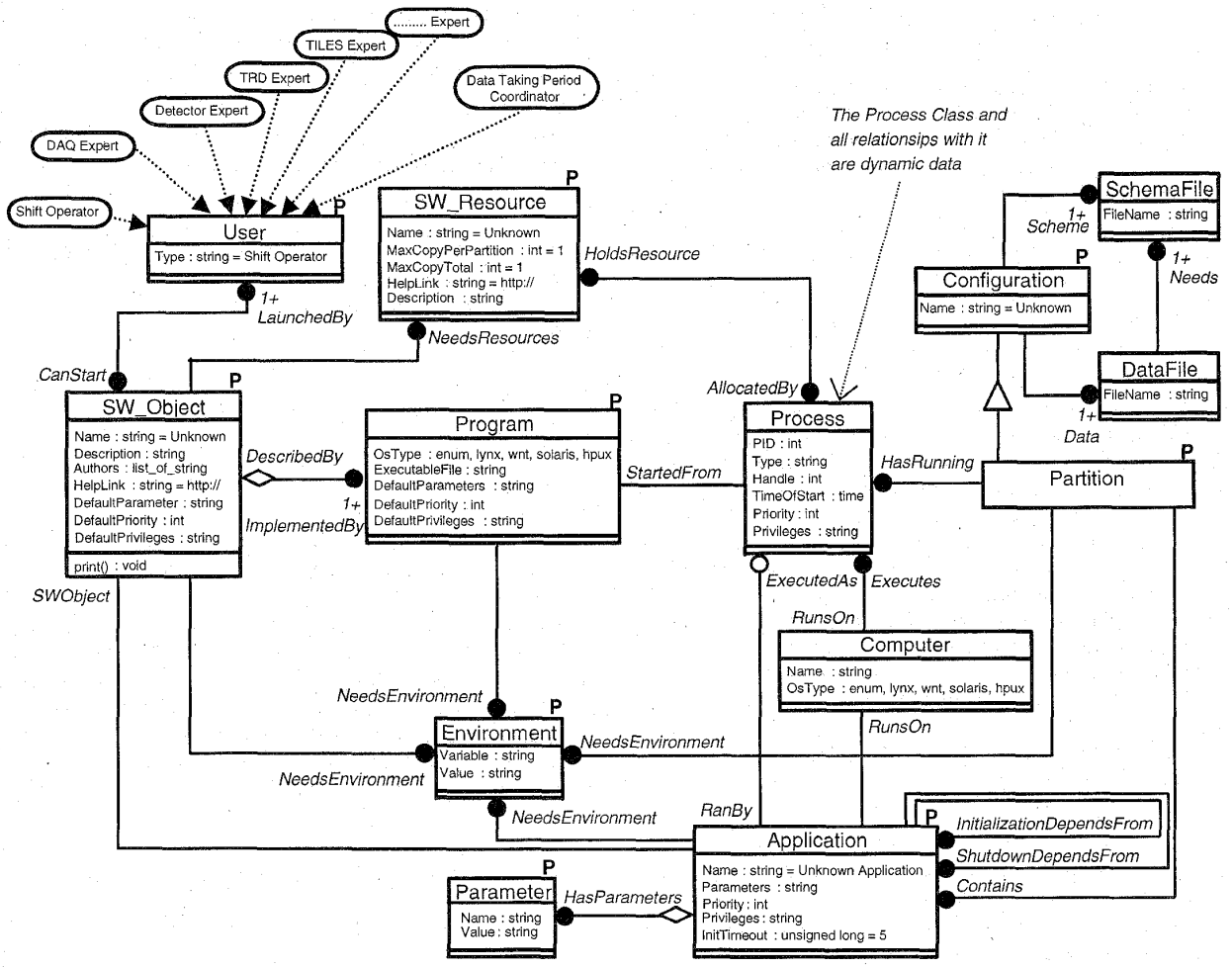

Figure 6: The schema of the Software and Setup configuration database for ATLAS DAQ prototype -1

3) *Data Access Libraries*

The use of a DAL (Data Access Library) in DAQ configuration databases simplifies the interface to the persistent object manager (i.e. OKS or Objectivity/DB), presents the schema as true C++ objects, hides details of the underlying persistent object manager and hence allows applications to be ported between OKS and Objectivity/DB without modification.

4) *Design Database Schemes*

The DAQ configuration database object schemes have been created using the StP (Software through Pictures [17]) CASE tool. The OMT methods supported by StP has been adopted as part of the development environment for ATLAS DAQ software [18] and used to model the schema of configuration databases (via object models). Code generators have been developed that produce DDL (Data Definition Language) for Objectivity/DB, native OKS definitions (via the C++ API) and the DAL implementation on top of OKS from models stored in StP.

The overall development process for the configuration databases is shown in Figure 7.

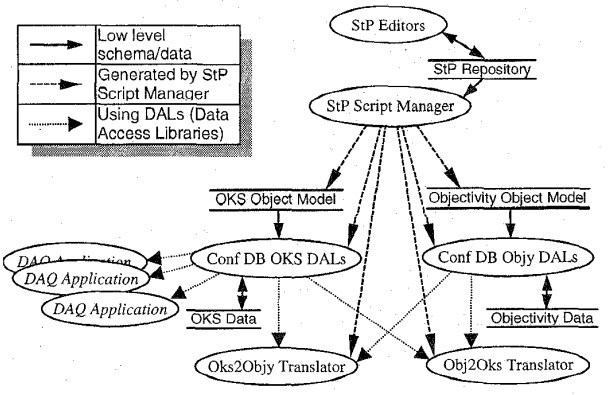

Figure 7: Development Process for ATLAS DAQ Prototype -1 Configuration Databases

C. Code Examples

The examples presented below show code that loads software and setup configuration database and lists partitions and corresponding applications. For clarity and size limitations no error checking is shown. To understand these examples we assume that reader is familiar with C++ and container classes.

The first example shows native OKS code and the second example shows generated DAL code. The DAL and OKS classes and methods are shown in **bold**, Rogue Wave Tools.h++ classes and methods are shown in ***bold italic*** and comments are shown in *small italic*.

1) Native OKS code

```
main() {
  // Initialize OKS kernel
  OksKernel kernel();

  // Load database
  kernel.oksLoadSchema("schema");
  kernel.oksLoadData("data");

  // Find class Partition to iterate instances
  OksClass *c;
  kernel.oksFindClass("Partition", &c);

  // Make iterator of instances
  RWTPtrHashDictionaryIterator<RWCString,
OksObject> *i = c.oksCreateObjectIterator();

  // Iterate instances
  for(;++i;) {
    // Receive a pointer to OKS object
    OksObject *po = i.value();

    // Gives values of attribute and relationship
    OksData *pName, *pApps;
    po->oksGetAttributeValue("Name", &pName);
    po->oksGetRelationshipValue(
                        "Applications, &pApps);

   // Prints object identity and name
    cout << "Partition " << po->getObjectId()
         << " has name " << *pName
         << " and applications:" << endl;

    // Is an object (application) in relationship?
    if(pApps->data.LIST &&
               pApps->data.LIST->entries()) {
      // Build an iterator for list of objects
      RWTPtrSlistIterator<OksData>
                          j(*pApps->data.LIST);
      // Iterate this list
      for(;++j;) {
        // Gives an object from list
        OksObject *ao = j->key()->data.OBJECT;

        // Prints the object identity
        cout << ao->getObjectId() << endl;
      }
    }
  }
}
```

2) Generated DAL code

```
main() {
  // Initialize object manager
  ConfDB conf_db();

  // Build list of parameters for conf. db
  RWTPtrSlist<RWCString> params;
  RWCString schema("schema"), data("data");
  params.append(&schema);
  params.append(&data);

  // Initialize conf. db
  MyConfDaqConf my_conf(conf_db, &params);

  // Build list of partitions we have and iterator
  RWTPtrSlist<Partition> *pl =
              my_conf.getListOfPartition();
  RWTPtrSlistIterator<Partition> i(*pl);

  // Iterate all partitions
  for(;++i;) {
    // Gets pointer to Partition C++ object
    Partition *pt = i.key();
    // Prints object identity and name
    cout << "Partition " << pt->getOID()
         << " has name " << pt->getName()
         << " and applications:" << endl;

    // Createss list of Applications in partition
    RWTPtrSlist<Application> *al =
        pt->createPartitionApplicationList();
    // Is the list empty?
    if(al && al->entries()) {
      // Builds iterator of applications
      RWTPtrSlistIterator<Application> j(*al);
      // Iterate it
      for(;++j;) {
        // Gives an Application C++ object
        Application *a = j->key();
        // Prints application object identity
        cout << a->getOID() << endl;
      }

      // Deletes list of applications
      delete al;
    }
  }

  // Deletes list of partitions
  delete pl;
}
```

IV. Future Plans

Initially we want to improve the support for automatic generation of schemes from the StP CASE tool and implement OKS/Objectivity data movers.

Later we intend to investigate a CORBA compatible Persistent Object Service [19] that will allow the storage of CORBA objects in OKS and share them between applications at run-time. This should provide the possibility to implement a data server for concurrent updates. However this implies potential network delays.

V. CONCLUSIONS

Based on the results of using OKS for the ATLAS DAQ prototype -1 configuration databases we can conclude that OKS is a suitable object manager to implement configuration databases. According to our tests it shows high performance, robustness, compatibility with used platforms and compilers and supports a sophisticated object model. Our initial survey failed to uncover an existing object manager that could satisfy all these requirements. OKS does not support all the features of a full object database management system but it is distributed in source code, requires few system resources and needs only one license for the popular Tools.h++ C++ library.

The two-tier architecture adopted for the ATLAS prototype project has allowed us to factorize the configuration data storage needs into an object manager for performance and portability reasons coupled to the ODBMS for long-term data management issues.

VI. ACKNOWLEDGMENTS

The following people have used the evolving versions of OKS in their work on the ATLAS DAQ prototype -1 system and provided valuable feedback and input: Giuseppe Mornacchi, Michaela Niculescu and Louis Tremblet for data-flow, Pierre-Yves Duval for the run control and the process manager, Ashruf Patel created the original version of StP/OMT to OKS object model translator, Elisabeta Badescu and Mihai Caprini implemented the Message Reporting System database and Viatcheslav Khomoutnikov used OKS for the ATLAS Control system configuration database.

VII. REFERENCES

[1] Tools.h++ Foundation Class Library for C++ programming User Guide and Class Reference, Rogue Wave Software, Inc., March 1996

[2] An Approach for description and interpretation of procedural knowledge for complex physical installations, V.Filimonov, V.Khomutnikov, Yu.Ryabov, Preprint PNPI #1828, September 1992

[3] A knowledge Based Control Method: Application to Accelerator Equipment Setup, G.Daems, V.Filimonov, V.Homutnikov, F.Perriolat, Yu.Ryabov, P.Skarek, Nucl. Instr. and Meth. A 352 (1994), pp. 325-328

[4] ORACLE Documentation Set, version 6, ORACLE, Inc., 1994

[5] Quid version 2.0 User Manual, Artis srl, November 1992

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

[7] Information on ODE is available through Internet at URL http://www-db.research.att.com/ode-announce.univ.html

[8] Information on YOODA is available through Internet at URL ftp://ftp.uu.net/pub/database/yooda

[9] Object Oriented database system evaluation for the DAQ system, M.Skiadelli, Diploma thesis at the University of Athens, 1995

[10] Experience using a distributed Object Oriented Database for a DAQ system, G. Ambrosini, C.P. Bee, M. Caprini, P.Y. Duval, S. Eshghi, F. Etienne, R. Ferrari, D. Ferrato, G. Fumagalli, I. Gaponenko, R. Jones, S. Kolos, C. Maidantchik, L. Mapelli, Y. Merzliakov, G.Mornacchi, M. Niculescuj, A. Patel, G. Polesello, D. Prigent, Z. Qian, I. Soloviev, R. Spiwoks, A. Le Van Suu, CHEP'95 conference at Rio de Janeiro, September 1995

[11] ATLAS DAQ software development environment working group, information is available through Internet at URL http://atddoc.cern.ch/Atlas/

[12] Design of the Configuration Databases for CERN ATLAS DAQ Prototype -1, R.Jones, M.Michelotto, A.Patel, I.Soloviev, ATLAS DAQ technical note 30, information is available through Internet at URL http://atddoc/Atlas/Notes/030/Note030-1.html

[13] ATLAS DAQ Back-end software User Requirements Document, 1996, is available through Internet at URL http://atddoc.cern.ch/Atlas/DaqSoft/document/draft_1.html

[14] The ATLAS DAQ and Event Filter Prototype -1 Project, G.Ambrosini, D.Burckhart, M.Caprini, M.Cobal, P-Y.Duval, F.Etienne, R.Ferrari, D.Francis, R.Jones, M.Joos, S.Kolos, A.Lacourt, A. Le Van Suu, A.Mailov, L.Mapelli, M.Michelotto, G.Mornacchi, R.Nacasch, M.Niculescu, .K.Nurdan, C.Ottavi, A.Patel, F.Pennerath, J.Petersen, G.Polesello, D.Prigent, Z.Qian, J.Rochez, F.Scuri, M.Skiadelli, I.Soloviev, R.Spiwoks, F.Touchard, L.Tremblet, G.Unel, V.Vercesi, S.Wheeler, A.Wildish, proceedings of CHEP'97 conference, April 1997

[15] OKS Documentation (User's Guide, Tools & Reference Manuals), I.Soloviev, CERN ATLAS DAQ technical note 33, information is available through Internet at URL http://atddoc.cern.ch/Atlas/Notes/033/Welcome.html

[16] Using Objectivity/C++, version 4, Objectivity, Inc., July 1996

[17] Information on StP/OMT is available through Internet at URL http://www.ide.com/Products/StP/StP_OMT.html.

[18] Use of Object oriented CASE tools for Automating the Development of DAQ Software, A.Patel, X-th IEEE Real Time Conference, September 1997

[19] CORBAservices: Common Object Services Specification, Object Management Group, Inc., November 1997