Applications of CORBA in the Atlas prototype DAQ

R. Jones[1], S. Kolos[2,3], L. Mapelli[1], Y. Ryabov[2]
[1]CERN, European Laboratory for Particle Physics, Geneva 23, Switzerland, CH-1211.
[2]PNPI, Petersburg Nuclear Physics Institute, Gatchina Leningrad distr., Russia.
[3]on leave from PNPI

Abstract

This paper presents the experience of using the Common Object Request Broker Architecture (CORBA)[1] in the ATLAS prototype DAQ project[2]. Many communication links in the DAQ system have been designed and implemented using the CORBA standard.

A public domain package, called Inter-Language Unification (ILU)[3] has been used to implement CORBA based communications between DAQ components in a local area network (LAN) of heterogeneous computers.

The CORBA Naming Service[4] provides the principal mechanism through which most clients of an ORB-based system locate objects that they intend to use. In our project, conventions are employed that meaningfully partition the name space of the Naming Service according to divisions in the DAQ system itself. The Inter Process Communication (IPC) package[5], implemented in C++ on the top of CORBA/ILU, incorporates this facility and hides the details of the naming schema is described. The development procedure and environment for remote database access using IPC is described.

Various end-user interfaces have been implemented using the Java language that communicate with C++ servers via CORBA/ILU. To support such interfaces, a second implementation of IPC in Java has been developed. The design and implementation of such connections are described.

An alternative CORBA implementation, ORBacus[6], has been evaluated and compared with ILU.

I. Introduction

The ATLAS prototype aims to design and implement a DAQ/Event Filter (EF) prototype[2] based on the Trigger/DAQ architecture described in the ATLAS Technical Proposal[7] and to support studies of the full system functionality. A requirements document[8] has been written for the back-end software which covers the needs of the prototype DAQ.

The Object Management Group's (OMG)[9] CORBA standard has been considered as one of the candidates to form the basis of all inter-component communications for the back-end DAQ. The ILU system has been chosen as a CORBA implementation and evaluated. The rationale for the choice of a free CORBA implementation was that it seems unlikely that we could find a commercially supported CORBA product for all front-end platforms used in the ATLAS DAQ. We needed an implementation that was distributed in source code form so that we could port it to new platforms as required. ILU has in addition some nice features which are explained in the overview below. The ILU evaluation[10] showed that the CORBA standard as implemented by ILU satisfies the back-end DAQ communication requirements.

During setup and testing periods, many groups of people will work in parallel and require the use of various DAQ resources. The concept of a DAQ partition was introduced to provide a context in which DAQ software components can operate in parallel with other partitions without interference. The operator may create and modify multiple partitions.

The package called Inter-Process Communication was developed on the top of ILU in order to support partitioning of the software components communication. IPC implementations exist for both C++ and Java languages.

Most communication for the back-end DAQ is implemented on top of the IPC package. As an example of a back-end DAQ component that uses IPC, the Remote Database (RDB) access package[11] is described.

The DAQ Integrated Graphical User Interface (IGUI) is now under development. It is implemented in Java and communicates with C++ applications via CORBA/ ILU using the IPC package.

II. Overview of ILU System

The Inter-Language Unification (ILU) system is a multi-language object interface system. The object interfaces provided by ILU hide implementation distinctions between different languages, between different address spaces, and between different operating system types. ILU interfaces can be specified in the OMG's CORBA Interface Definition Language (CORBA IDL).

ILU 2.0 provides mappings for several programming languages including C++, ANSI C, Python, Java, and Common Lisp. ILU has been installed on most flavors of UNIX (SunOS, Solaris, HP-UX, AIX, OSF, IRIX, FreeBSD, Linux, LynxOS, SCO Unix, etc.) and MS-Windows (3.1, 95, NT). It supports both threaded (POSIX, Solaris, NT, etc.) and event loop (Xt, Tk, XView) operation.

The current release includes support for the CORBA Internet Inter-ORB Protocol (IIOP).

III. Overview of CORBA Naming Service

One of the first CORBA Service specifications proposed by OMG was the Naming Service. It defines a federated (hierarchical) naming service which is explained below.

A name-to-object association is called a name binding. A name binding is always defined relative to a naming context. A naming context is an object that contains a set of name bindings in which each name is unique. Different names can be bound to an object in the same or different contexts at the same time. To resolve a name is to determine the object

associated with the name in a given context. To bind a name is to create a name binding in a given context. A name is always resolved relative to a context, there are no absolute names. Because a context is like any other object, it can also be bound to a name in a naming context. Figure 3 in the next section shows an example of a naming graph.

The *CosNaming* Module is a collection of interfaces that together define the naming service. This module is described in CORBA IDL and contains two interfaces:

- The *NamingContext* interface - allows objects bindings and names resolution;
- The *BindingIterator* interface - allows iteration through the bindings.

IV. IPC PACKAGE

IPC provides a way to run several instances of components belonging to different contexts (partitions) in parallel. IPC simplifies the development process and minimizes the dependencies on a particular CORBA implementation.

A. IPC Partition definition

The term Partition has a complex explanation in the context of a full DAQ system. The IPC Partition is defined in relation to the software objects participating in communication. This definition is not in contradiction with the general one and does not impose any restrictions on it. The IPC Partition acts as a namespace for the communicating objects and satisfies the following rules:

- Each object belongs to one and only one partition;
- Each object has an identifier which is unique inside partition;
- Each object can be accessed from anywhere by it's identifier and the identifier of the partition it belongs to.

B. IPC Architecture

The IPC Object Model diagram is shown in Figure 1.

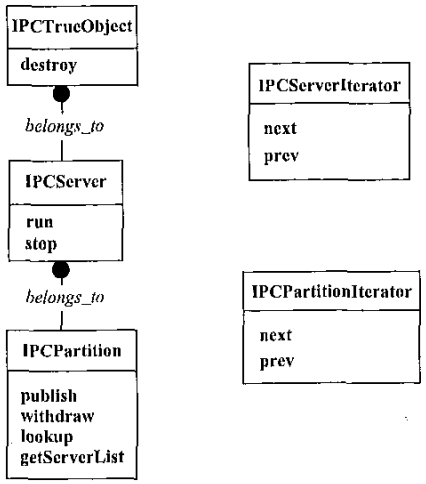

Figure 1: IPC Object Model Diagram (OMT notation).

The IPCPartition class incorporates the Naming Service facility. It assigns a name to the object that inherits from IPCTrueObject class and makes this name-to-object association publicly available. Later the reference to this object can be obtained by calling the *IPCPartition::lookup* method with the object name as a parameter. The *IPCServer* class plays two roles. It services the application's external connections and internal alarms using the ILU event loop. It also defines the namespace for the object names allowing additional flexibility for object identification. The *IPCPartitionIterator* allows sequential access to all the available partitions. The *IPCServerIterator* allows iteration through all the servers in the current partition.

C. IPC Implementation

The IPC implementation is provided in the form of a library based on the CORBA Naming Service. It exists in both C++ and Java.

An IDL definition provides a remote object destruction facility (Figure 2).

The ipc::freeable interface is implemented by the *IPCTrueObject* class. The virtual destroy method inherited from the *IPCTrueObject* class can be used to implement application specific termination.

```
module ipc {
   interface freeable {
       oneway void destroy();
   };
};
```

Figure 2: IPC interface declaration (CORBA IDL)

A partition is represented by the Naming Server application. Each back-end DAQ component has a unique context identifier which is bound in the partition with a *naming context* object. For each object belonging to this component there is a *name binding* in this context. Figure 3 shows an example of the internal layout of the partition server.

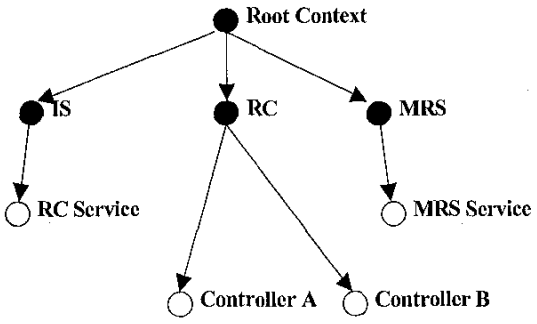

Figure 3: An Example of a Naming Graph for a Partition Server

There are three contexts for the DAQ components shown in Figure 3, namely:

- Run Control (RC) with two controller objects;
- Information Service (IS) with one object which supports Run Control dynamic information distribution;

- Message Reporting System (MRS) with one object which implements message distribution.

1) Java implementation

The Java IPC package implements all the classes shown on Figure 1. For the current implementation we have used JDK 1.2[12] and **idltojava** version 1.2[13], both products of Sun Microsystems.

The implementation is provided as a Java package called ipc. Figure 4 shows the classes in this package.

```
ipc.
        Partition
        PartitionIterator
        Server
        ServerIterator
        TrueObject
```

Figure 4: The structure of the Java IPC Package

D. IPC test results

The C++ IPC implementation was tested on all the platforms supported in back-end DAQ for functionality, performance and error recovery facilities. The overhead of using IPC against pure ILU was investigated.

The results of the performance tests show that there are practically no performance differences between IPC and pure ILU. The tests show that IPC works reliably with sufficient performance even for configurations with up to 250 clients communicating with the same server. The tests with larger numbers of clients were not performed because they are beyond the needs of the current DAQ prototype. It was discovered that even for the biggest configuration expected, the communication time grows linearly with the number of applications involved in communication with the same server.

The functionality and code coverage tests show the current implementation of IPC is reliable and stable enough to be used for communication purposes in the DAQ environment.

A detailed explanation of test items and results can be found in [14], [15].

V. RDB Package

Many components of the DAQ system require database access. Some of them access the database directly using its native API. But applications which are implemented in a programming language that is not supported by the database tool need another means of access. Another reason to have remote access to a database is to avoid the need for a common file system for all the applications in the DAQ system. The RDB package defines and implements an interface to read database information.

B. RDB Requirements

The RDB package is intended to provide applications with read- only access to a database regardless of their location: in the same process as the Persistent Object Manager, different processes on the same machine or different machines. The RDB's API is independent of a particular Persistent Object Manager and database schema.

Independence from the database schema means that RDB can provide an interface for database meta-information[1] access. It allows multiple clients implemented in different programming languages (C++, C, Java) to access the same database simultaneously.

A. RDB Architecture and Functionality

Figure 5 shows the Object Model of RDB. The Cursor class interacts with the database retrieving the necessary information which was requested via it's methods.

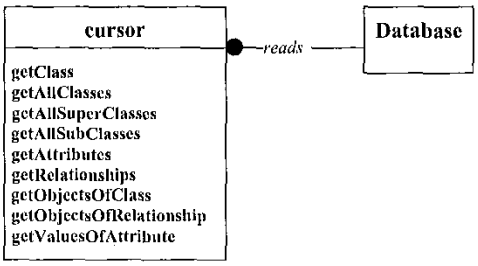

Figure 5: RDB Object Model Diagram (OMT notation)

The first six methods of the cursor object are intended to retrieve the following meta-information from the database:

- Meta-information for a specific class: name, description, number of attributes and relationships;
- Meta-information for a specific class attribute: name, description, type, multiplicity;
- Meta-information for a specific class relationship: name, description, cardinality constraints, class name it relates to.

Using meta-information it is possible to access the values of specific attributes and relationships of specific objects. The Cursor class provides three methods to access database information:

- Get a list of all the instances of a class;
- Get all the values for a relationship of a specific class instance;
- Get all the values for an attribute of a specific class instance.

B. RDBServer Implementation

The RDB implementation is based on the IPC package. It is provided in the form of a C++ library with an interface declaration in CORBA IDL (**rdb.idl**). It defines the module *rdb* with one interface called cursor which declares all the methods shown on Figure 5. The cursor interface inherits from the *ipc::freeable* interface. Figure 6 presents the general structure of the **rdb.idl** file.

[1] Meta-information is information about database schema, i.e. a description of classes and their relationships.

```
#include "ipc.idl"
module rdb{
    // various structures
    // are declared here
    .......
    interface cursor: ipc::freeable{
    // methods are declared here
    ....
    };
};
```

Figure 6: IDL declaration for RDB

The class *rdb_T_cursor* is generated by the IDL to C++ translator from **rdb.idl**. The *RDBServer* class inherits from the *rdb_T_cursor* class and provides implementations for all the methods. The class IPCPartition from the IPC package is used to provide a way of splitting the information between different partitions. Classes IPCServer and IPCTrueObject are used as base classes for *RDBServer* in order to support server animation and IPC object implementation respectively.

Figure 7 shows all the classes in the RDB library and their relationships.

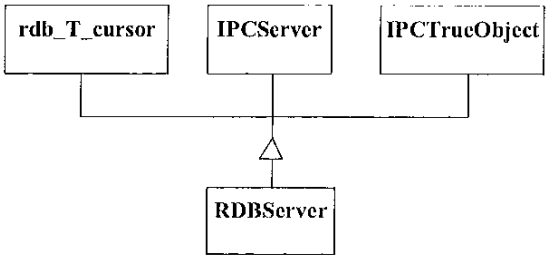

Figure 7: The RDB Implementation classes (OMT notation)

There is no hand-written code for the client part of the library. So in order to retrieve information from the remote database server the class generated by IDL compiler is used directly. This class is called *rdb_T_cursor* for C++ and *rdb.cursor* for Java. The instance of such a class can be obtained in the client program by using the lookup method of the *IPCPartition* class. Call to any method of this class causes its invocation with the same signature on the respective instance of the *RDBServer* class. It is done transparently via ILU.

VI. Integrated User Interface

We are currently using the OMG IDL language to specify component interfaces and the IDL compiler to generate C++ code from these specifications. Such interfaces are defined for the following components: Information Service (IS)[16], Message Reporting System (MRS)[17], Process Manager (PMG)[18], Run Control (RC)[19] and Remote Database access (RDB). All these components are implemented on top of the IPC package.

The Integrated Graphic User Interface (IGUI) requires interaction with all the components mentioned above. As it is implemented in Java, a Java version of IPC was provided. The IGUI application uses the Java IPC package and additional Java code generated by the IDL-to-Java compiler from the IDL specifications of the back-end packages.

A. IGUI Requirements

The IGUI[20] presents the status of the DAQ system to the human operator and provide a means to display the status of individual sub-systems and components. User identification and access control are applied before sending commands to the DAQ system.

B. IGUI Architecture

Figure 8 presents the basic context diagram for the IGUI showing the exchanges of information with the other back-end subsystems.

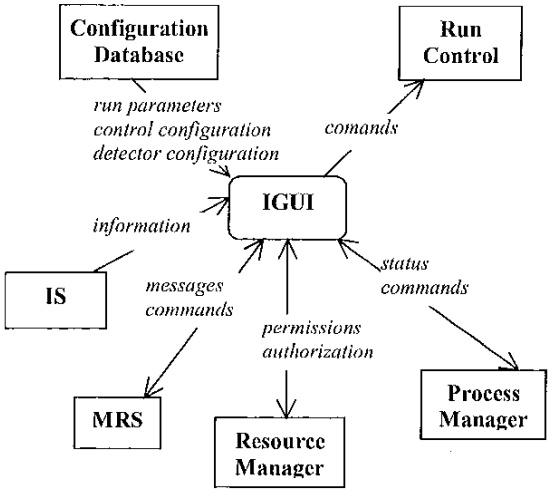

Figure 8: IGUI Context Diagram

A. IGUI Implementation

The current version of IGUI uses JDK 1.2 and **idltojava** version 1.2 from Sun Microsystems.

The IGUI application is able to:

- Show the current partition and lists others;
- Send commands to the DAQ supervisor and RC controllers;
- Show the RC tree with controller's status;
- Receive MRS messages;
- Show a list of Process Manager Agents and processes started by them.

VII. Evaluation of ORBacus

The purpose of the ORBacus evaluation was to compare an alternative CORBA implementation with ILU currently in use in the DAQ. The motivation for the evaluation was to gain experience with version 2.0 of the CORBA/IIOP standard since it is not fully supported by the current version of ILU. In addition, ILU lacks some CORBA services that potentially could be used within the DAQ. For example we have made our

own implementation of the CORBA Naming Service because it was not provided by ILU when we started the project.

A. ORBacus overview

ORBacus is a robust, full-featured object request broker provided by the OOC company. ORBacus is fully-compliant with the CORBA 2.0 specification[21], including the C++ and Java mappings[22]. ORBacus supports every feature of CORBA IDL with no omissions. ORBacus uses IIOP as its native protocol and also has an open architecture that allows the development of other protocols for use with the ORB. ORBacus includes a robust, fully featured Interface Repository. ORBacus includes C++ and Java implementations of three standard OMG services: Naming, Event and Property.

ORBacus has very good support and maintenance as well as excellent documentation. OOC sell the same ORBacus version as is available free for non-commercial use. It is distributed in source code and supports a number of popular operating systems and compilers.

B. Evaluation results

We installed ORBacus version 3.1.2 for the evaluation on LynxOS, Linux and Solaris using various compilers. Apart from installation problems on LynxOS with the older GCC compiler[23] (version 2.7.2 which is not officially supported by ORBacus), ORBacus worked well. It showed similar performance characteristics to ILU for basic communication using the IIOP. The evaluation will continue to explore the CORBA Services when the compiler situation on LynxOS improves.

VIII. Conclusions

Ten CORBA based interfaces have been defined for DAQ components including IPC, IS, MRS, Process Manager, Resource Manager, Remote Database Access and Run Control using the IDL language.

Based on the results of components implementations, we can conclude that CORBA is suitable as a means of providing inter-process communication for the ATLAS prototype DAQ back-end. CORBA provides a high-level standard means of communication. Following this standard simplifies the modeling and subsequent realization of complex communication systems as well as giving a higher level of code portability.

In comparison with direct socket programming CORBA gives many advantages, namely:

- Advanced object addressing schema;
- Communication protocol independence;
- Simple and efficient connection management mechanism;
- Transparent data marshalling.

CORBA implementations lack performance in comparison with using a TCP/IP interface directly. Generally they are about 30% slower[24]. None the less, it is possible to achieve with CORBA the performance necessary for DAQ control purposes.

The DAQ makes use of the IIOP protocol with ILU. This allows interoperability with other CORBA broker implementations. For example, currently we are using a CORBA implementation included in JDK1.2. We are able to use Java clients interfaced to existing C++ servers.

The Naming Service has proved very useful for object addressing in the DAQ. There are also some other services, specified in the latest CORBA standard revision, which could be useful in the DAQ: Event, Persistence and Relationship Services. We intend to study alternative CORBA implementations which include such services.

IX. Acknowledgment

The authors would like to thank all of their colleagues in the ATLAS prototype DAQ project for providing valuable input during the design of the back-end components. In particular we would like to thank Igor Soloviev for his help with the RDB Package interface definition, Mihai Caprini and Lorenzo Moneta for their work on the use of the IPC within the IGUI. Frederic Hogbe deserves special thanks for the ORBacus evaluation.

X. References

[1] CORBA, http://www.omg.org/corba /

[2] ATLAS DAQ/Event Filter Prototype -1 Project, CERN, http://atddoc.cern.ch/Atlas /

[3] ILU by PARC Xerox Co., ftp:// ftp.parc.xerox.com/pub/ilu/ilu.html

[4] CORBA Naming service specification, ftp:// www.omg.org/pub/docs/formal/97-12-10.pdf

[5] S.Kolos, ATLAS DAQ Prototype -1 Technical Note 75, Inter Process Communication package, http://atddoc.cern.ch/Atlas/Notes/075/Note075-1.html

[6] ORBacus home page, http://www.ooc.com/ob/

[7] ATLAS Technical Proposal, CERN/LHCC/94-43 (ISBN 92- 9083-067-0)

[8] ATLAS DAQ Back-end software User Requirements Document. http://atddoc.cern.ch/Atlas/DaqSoft /document/draft_1.html

[9] The Object Management Group, http://www.omg.org/ omg/

[10] S. Kolos, Inter-component communication in the ATLAS DAQ back-end software (evaluation of the ILU multi-language object interface system), ATLAS DAQ Prototype -1 Technical Note 3, http://atddoc.cern.ch/Atlas/Notes/003/ Note003-1.html

[11] S. Kolos, I. Soloviev, Remote Database Access library: Users Guide, ATLAS DAQ Prototype -1 Technical Note 122, http://atddoc.cern.ch/Atlas/Notes/122/Note122-1.html

[12] JAVA TM 2 SDK, Standard Edition, http://www.java-soft.com/products/jdk/1.2/

[13] Java IDL Documentation, http://java.sun.com/products/jdk/1.2/docs/guide/idl/index.html

[14] D. Burckhart, S. Kolos, Test Plan of the Inter Process Communication package for the Atlas DAQ Prototype -1, ATLAS DAQ Prototype -1 Technical Note 103, http://atddoc.cern.ch/Atlas/Notes/103/Note103-1.html

[15] S. Kolos, Test Report of the Inter Process Communication package for the Atlas DAQ Prototype -1, ATLAS DAQ Prototype -1 Technical Note 123, http://atddoc.cern.ch/Atlas/Notes/123/Note123-1.html

[16] S. Kolos, Implementation of the Information service, ATLAS DAQ Prototype -1 Technical Note 37, http://atddoc.cern.ch/Atlas/Notes/037/Note037-1.html

[17] D. Burckhart, M. Caprini, S. Kolos, R, Jones, A. Radu Users Guide and Implementation of the Message Reporting System for the Atlas DAQ Prototype -1, ATLAS DAQ Prototype -1 Technical Note 59, http://atddoc.cern.ch/ Atlas/postscript/Note059.ps

[18] P-Y. Duval, L. Cohen, Users guide for the Process Manager, ATLAS DAQ Prototype -1 Technical Note 81, http:// atddoc.cern.ch/Atlas/Notes/081/Note081-1.html

[19] Run Control User's Guide, ATLAS DAQ Prototype -1 Technical Note 107, http://atddoc.cern.ch/Atlas/Notes/107/Note107-1.html

[20] Integrated Graphic User Interface (IGUI) User Requirements document, http://atddoc.cern.ch/Atlas/DaqSoft/components/gui/GUI_ur.html

[21] The Common Object Request Broker: Architecture and Specification Revision 2.0, (OMG Document 97-02-25)

[22] IDL/Java Language Mapping, (OMG Document 97-03-01)

[23] The GNU C Compiler, http://www.gnu.org/soft-ware/gcc/gcc.html.

[24] A. Gokhale, D. Schmidt, Washington University, Measuring the Performance of Communication Middleware on High-Speed Networks, submitted to IEEE Transactions on Computing, http://www.cs.wustl.edu/~schmidt/ieee_tc-97.ps.gz