EXPERIENCE WITH CORBA COMMUNICATION MIDDLEWARE IN THE ATLAS DAQ

S.Kolos[*], University of California, Irvine, USA
D.Burckhart-Chromek, J.Flammer, M.Dobson, R.Jones, D.Liko, L.Mapelli, CERN, Geneva, Switzerland
I.Alexandrov, V.Kotov, S.Korobov, M.Mineev , Joint Institute for Nuclear Research, Dubna, Russia
A.Amorim, N. Fiuza de Barros, D.Klose, L.Pedro, Universidade de Lisboa, Faculdade de Ciencias (FCUL- CFNUL), Lisbon, Portugal
E.Badescu, M.Caprini, National Institute of Physics and Nuclear Engineering, Bucharest, Romania
A.Kazarov, Y.Ryabov, I.Soloviev, Petersburg Nuclear Physics Institute, Gatchina, Russia

Abstract

As modern High Energy Physics experiments require more distributed computing power to fulfil their demands, the need for efficient distributed online services for control, configuration and monitoring in such experiments becomes increasingly important.

This paper describes the experience of using standard Common Object Request Broker Architecture (CORBA) [1] middleware for providing a high performance and scalable software, which will be used for the online control, configuration and monitoring in the ATLAS [2] Data Acquisition (DAQ) system. It also presents the experience, which was gained from using several CORBA implementations and replacing one CORBA broker with another.

Finally the paper introduces results of the large scale tests, which have been done on the cluster of more then 300 nodes, demonstrating the performance and scalability of the ATLAS DAQ online services. These results show that the CORBA standard is truly appropriate for the highly efficient online distributed computing in the area of modern HEP experiments.

INTRODUCTION

ATLAS is one of the four experiments in the Large Hadron Collider (LHC) [3] accelerator at CERN. The ATLAS detector consists of several sub-detectors, which can be operated in parallel and fully independently from each other.

The ATLAS Data Acquisition (DAQ) system transports event data from the 1600 detector read-out links to mass storage. In order to provide the required functionality and to handle the physics data rate, the DAQ system will use several hundreds processors connected altogether over a high-speed network with each of them running several DAQ software applications.

The Online Software project is part of the ATLAS DAQ, which is responsible for the control, configuration and monitoring of the ATLAS DAQ system. This is a challenging project which has to meet a number of strong requirements. The size and distributed nature of the ATLAS DAQ implies that the Online Software has to be scalable and high performance distributed software. The long life time of the ATLAS experiment requires that this software has to be easily extendable and maintainable. In order to meet these requirements it has been decided to use CORBA based communication middleware as a basis for the DAQ Online Software implementation.

The Online Software provides a number of software services, which are grouped into three categories: configuration, control and monitoring services. The services are clearly separated one from another and have well defined boundaries. For each service there is a low-level software component, which provides both the service implementation and API.

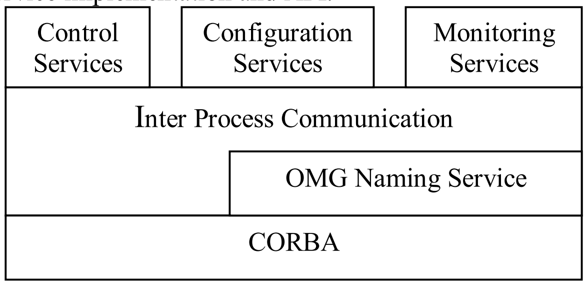

Figure 1: Online Software architecture

The distributed facilities of the Online Software services are implemented on top of the Inter Process Communication (IPC) software package, which is a wrapper above CORBA and the OMG Naming Service [4] (see Fig. 1).

RATIONALE OF CORBA

CORBA is an open standard for distributed object computing, which has been proposed in 1991 by the Object Management Group (OMG) [5]. It standardises many common network programming tasks such as object registration, location and activation; parameter marshalling and demarshalling; operation dispatching and so on.

CORBA supports most of the widely used programming languages, like for example C, C++, Java;

[*] On leave from Petersburg Nuclear Physics Institute

as well as a number of scripting languages like for example Python. This gives to software developers a large flexibility in choosing the most appropriate programming language for the distributed systems implementations.

Different implementations of the CORBA standard are able to communicate with each other. This interoperability is one of the most important advantages of CORBA. This feature provides another level of flexibility by allowing using different CORBA brokers for example in C++ and Java applications.

All CORBA compliant brokers are compatible at the source code level. CORBA based user programs can be compiled with almost no changes using different CORBA brokers. This is very important in the context of the long lifetime of the ATLAS experiment. Using CORBA one relies to the widely used and mature standard, which allows replacing one CORBA broker with another with minimal changes to the software programs.

CORBA provides a high-level object-oriented paradigm for the distributed programming hiding low level aspects of the communication implementation. However the API and communication model, which are provided by CORBA, are considerably complex. This complexity is a result of the flexibility, which CORBA offers for the distributed application development. However the drawback of this flexibility is a significant learning curve for CORBA, which makes its use unproductive in the international scientific communities, in which people are changing each other very often. In order to overcome this issue, a light-weight software wrapper, called Inter Process Communication (IPC), has been implemented on top of CORBA in the scope of the ATLAS DAQ project. The IPC drastically simplifies the distributed programming interface by narrowing the very wide spectrum of features, provided by CORBA, to a reasonably small subset of functions, which are used by the ATLAS DAQ Online Software.

INTER PROCESS COMMUNICATION PACKAGE

The IPC is a software package, which provides several important functionalities to the other Online Software services:

- a simple API for the core CORBA facilities
- a simple API for the OMG Naming Service
- transparent cache for the remote object references

Core CORBA functionalities

There are several actions, which have to be done by any CORBA application: ORB initialisation, Root and specific Portable Object Adapters (POAs) creation, POA Manager creation, CORBA objects registration and initialisation, etc. IPC completely hides all these actions either by doing them implicitly (when it is appropriate) or by providing very simple function calls for them. For example the fragment of the C++ code, which is shown on Fig. 2, does the ORB initialisation as well as the Root POA and POA Manager objects creation. Parameters, which are passed to the IPCCore::init function, are used to transmit some ORB specific configuration attributes from the application command line to the IPC run time.

```
int main( int argc, char ** argv ) {
   IPCCore::init( argc, argv );
   …
}
```

Figure 2: IPC initialisation in C++

In order to develop a service using CORBA one has to provide the service description using the OMG Interface Definition Language (IDL) [6]. The IDL is a purely declarative language with the syntax very similar to C++, which makes it easy to learn and use. When the IDL description for the service is ready one has to invoke an appropriate IDL compiler, which translates the IDL declaration to the necessary programming language definitions in accordance with the OMG specification. Then the service can be implemented in that programming language. IPC respects this procedure and any IPC based service has to have an interface declared in IDL. Fig. 3 shows one of the real Online Software IDL interfaces.

```
module rc {
  interface controller : ipc::servant {
     void command( in string cmd );
  }
}
```

Figure 3: Run Control IDL interface

This is the interface to the Run Control (RC) service [7], which defines the rc::controller interface. This interface has a single method called command, which is used to send commands to any RC controller of the DAQ system.

Fig. 4 shows the implementation of the Run Control interface in C++. The POA_rc::controller class, which is used as template parameter of the IPCNamedObject class, is a class generated by the IDL to C++ compiler application. It has the same name for any CORBA compliant broker.

```
class RCController :
  public IPCNamedObject<POA_rc::controller>
{
  RCController( const IPCPartition & p,
                const std::string & name )
   :
IPCNamedObject<POA_rc::controller>(p,name)
                  { publish(); }

  ~RCController() { withdraw(); }

  void command( const char * cmd ) { … }
};
```

Figure 4: Run Control implementation in C++

IPC provides a very simple API for creating CORBA objects and registering them with the appropriate POA. This API completely hides the POA class and performs the necessary POA operations like POA creation, objects registration, objects activation and so on implicitly.

IPC provides two base classes for CORBA object implementations, which perform all the necessary POA interactions. For example any instance of the class, which inherits the IPCNamedObject (as in Fig. 4), can be connected from any other IPC based application via the name, which is provided at the object construction. Fig. 5 shows how an instance of the IPCNamedObject based class can be created. After that any other application can get a reference to the "MyCtrl" controller in the "MyPartition" partition in order to invoke remote methods of that controller.

```
IPCPartition p( "MyPartition" );
RCController * rc;
rc = new RCController ( p, "MyCtrl" );
```

Figure 5: IPC named object creation

IPC provides also another class called IPCObject, which can be used as a base class for the anonymous CORBA object implementations. An instance of a class, which inherits the IPCObject, can be passed to another program only explicitly, as parameters of another remote operation. This facility is used for example to implement the subscription/call-back pattern.

IPC Partition

A DAQ Partition represents a self contained instance of the ATLAS DAQ system for the piece of the ATLAS detector, which can perform data taking activity independently and concurrently with the other detector parts.

The IPCPartition class puts a notion of the DAQ Partition into the programming language context. As one can see from the example on Fig. 5 any named IPC object may belong to one and only one IPC Partition. This allows having several concurrent instances of the same service for different DAQ Partitions. For example one can unambiguously create two RC controllers with the same name if they belong to different partitions. The IPCPartition class provides an isolated communication domain, which has no interference with any other IPC Partitions.

The IPC Partition implementation is based on the OMG Naming service, which provides a way of binding CORBA object reference with a name. Any application can interact with the Naming Service for getting a reference to any registered object by providing the object binding name. The IPCPartition class together with the IPCNamedObject provides a simple wrapper for the Naming Service API. The IPCNamedObject class has two related server side methods: publish, which binds the object to the name in the context of the specific partition; and withdraw, which removes that binding. Fig. 4 shows how these methods can be used in the constructor and destructor of the distributed service implementation class. Partition and the object name are provided as parameters of the IPCNamedObject constructor (see Fig. 4).

The IPCPartition class provides the client counterpart for the publish method, which is called lookup. Fig. 6 shows how to use this method for getting the remote object reference and invoking operation on that object.

```
IPCPartition p( "MyPartition" );
rc::controller_var rc =
        p.lookup<rc::controller>( "MyCtrl" );

rc -> command( "MyCommand" );
```

Figure 6: IPC named object creation

Note the use of the same partition and object names as in the Fig. 5. The rc::controller and rc::controller_var classes are generated by the IDL compiler. They have exactly the same signatures for any CORBA compliant broker.

Caching of remote object references

Establishing connection between client and server processes is one of the most expensive operations in the scope of remote communication. If a remote operation is invoked relatively often, then it is much more efficient to keep the connection open all the time. For this purpose the IPC library implements a cache for remote object references, which prevents connection from been closed and re-established in between subsequent requests to the same remote object.

In most cases this cache facility is completely transparent for a user application, but there is one case, in which user program assistance is required in order to handle this situation properly. Such a situation occurs if computer, on which one of the IPC based services is running, had died and that service is restarted on another machine. In this case the object reference, which is kept in the client programs cache, becomes invalid. An attempt to invoke any operation using this reference will result in the IPCCacheExpired exception. User has to catch this exception and call the IPCPartition::lookup function again in order to refresh the remote object reference. Fig. 7 shows how this has to be done.

```
  IPCPartition p( "MyPartition" );
again:
  rc::controller_var rc =
         p.lookup<rc::controller>( "MyCtrl" );

  try {
    rc -> command( "MyCommand" );
  }
  catch( IPCCacheExpired & )
  {
    goto again;
  }
```

Figure 7: Handling the IPCCacheExpired exception

Other IPC facilities

IPC provides some other facilities, which simplify distributed services development and maintenance. For example it provides a common base interface for all the other services specific IDL interfaces. This interface is called ipc::servant and must be inherited by any specific IDL interface (see Fig. 3). The ipc::servant defines several common operations for all the Online Software services. For the moment this interface provides a way of retrieving some general service parameters like the service owner, starting time and host machine. If necessary this interface can be transparently extended for providing more information.

IPC is also used for implementing many useful features transparently for the other Online Software services. For example the Access Management service is currently being implemented. It will provide any DAQ application with some sort of authorisation token. The IPC can transparently attach such a token to the context of a remote method invocation on a client side. At the server side the IPC can extract and verify this token (again transparently for the server program) and pass it to the Access Manager, which can make a decision of either to allow or refuse the requested operation.

CORBA BROKERS

Initially the Online Software has been using the implementation of CORBA, which is called Inter Language Unification (ILU) [8] for C++ and the JavaIDL[†] broker for Java. The results of using ILU have been found quite satisfactory, but in the middle of 2002 Xerox company, which implemented ILU, has dropped the ILU support. The ILU was not evolving any more and therefore missed several useful features, which appeared recently in the new versions of the CORBA standard. It has been decided to evaluate several other CORBA brokers in order to find a replacement for the ILU.

This evaluation has been reported in [9] and based on it's results, ILU has been replaced with another CORBA broker, called omniORB [10]. We have found also some drawbacks and problems for the JavaIDL broker and replaced it with JacORB [11].

Replacing the Java broker required no modification to the code of the Online Software services while transition to the new C++ broker was not transparent. Normally CORBA compliant C++ brokers differ only by the names of the header and library files. However migration from the ILU to the omniORB required more efforts because the ILU is not fully CORBA compliant. In fact the need of using a CORBA compliant broker was one of the reasons for moving to the omniORB. Any possible future transition to another CORBA compliant C++ broker will be straightforward.

PERFORMANCE AND SCALABILITY MEASUREMENTS

The unprecedented size of the ATLAS detector puts very strong performance and scalability requirements to the control, configuration and monitoring services, which are provided by the Online Software for the ATLAS DAQ system. In addition those requirements are not the same for different types of the services.

The Control services are dealing normally with short network messages, which can be sent for example to the Run Control controllers in order to change their states, or to the Process Manager agents in order ask them to start certain processes. Such operations are performed rarely and the total amount of data, which is transferred over network, is small. But the scalability aspect is very important because normally these operations are applied concurrently to a large number of the remote applications.

The situation is similar for the Configuration services, which provide configuration data for a large number of applications at the same time. The important difference is that the size of the data transferred over network is significantly larger than for the Control services.

The requirements for the Monitoring services differ because monitoring is permanently in use during data taking activity for transferring large amount of data between different DAQ applications.

In order to study the performance of different services as well as of the integrated DAQ infrastructure, a large test bed has been set up. The test bed consisted of more than 300 PCs, which were connected over 100 MB Ethernet. A number of independent tests have been performed for different services on that test bed. The next sections present some results of these tests.

Control services

Starting with booted but idle machines, the tests for the Control services simulate the start of data taking activities by creating all the necessary processes in the defined order and then cycling the system through the states prescribed by the Run Control to simulate a data taking activity. At the end of these cycles, the system is shut-down in an orderly manner and all the DAQ related processes are destroyed. Timing values were written to file and subsequently loaded into a spreadsheet. These tests have been done using a cluster of more than 300 computers. All the tests have been repeated a number of times and plots show mean times for each operation.

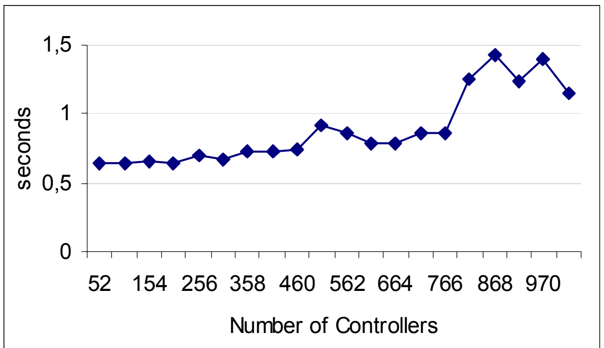

Figure 8: RC Start transition time

Fig. 8 shows the time for performing single Run Control transition, which is called Start. This is a simplest

[†] JavaIDL was a name of the CORBA broker integrated to JDK. Now it has different name.

RC transition, which consists from two steps. At each step all the controllers are synchronised, i.e. they all receive a respective command, change their state according to them and report a new state to their parents.

Controllers, which have been used for these tests were a dummy ones, which did nothing during transition. The time, which has been measured for that transition, is twice of the time of distributing the RC commands to all the controllers and receiving a confirmation from them for the transition completion. More information about the results of the scalability tests for the Online Software Control services can be found in [7].

Configurations services

The Configurations service is used to keep the set of parameters describing TDAQ and to provide access to it. The database is read simultaneously by many DAQ applications during the data taking session initialisation. Each application may read a number of parameters from the database. A total amount of data to be read may vary from several bytes up to tens of Mbytes for different applications. The total number of applications in the final DAQ system will be at the order of several thousands. A single server is not capable of implementing the required database service for performance reasons and therefore it will be necessary to run several instances of them. The main goal of the configurations service tests was to find out the required number of Configuration servers for the final ATLAS DAQ system. There are several tests series, which have been performed using 220 computers from the test bed. All the tests have been repeated several times and plots show mean time for these repetitions. Fig. 9 shows mean time for reading single simple object from the database.

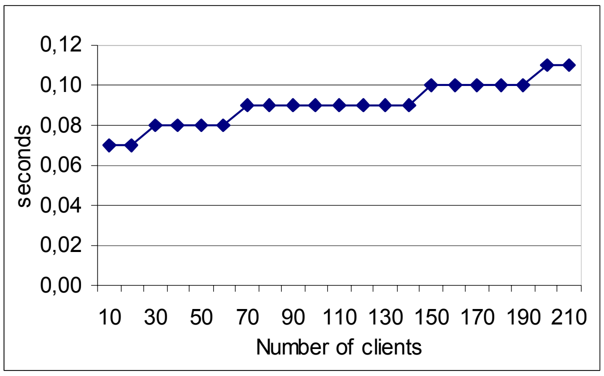

Figure 9: Time for reading single object

Fig. 10 shows results of the other tests, which have been performed to measure the time for reading composite objects from the database (consists of 2042 small nested objects) and for reading large objects, which have size of 1, 4, 8 and 16 MB.

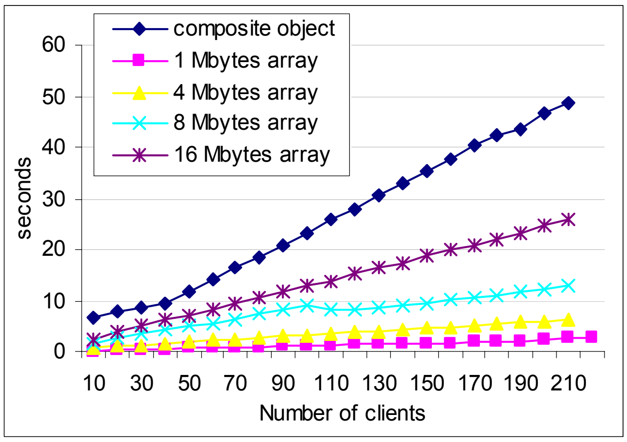

Figure 10: Time for reading composite and large objects

More information about the results of the scalability tests for the Online Software Control services can be fount in [12].

Monitoring services

The Information Service (IS) is one of the Monitoring services provided by the Online Software. It is used to share user defined information between DAQ applications. It is responsible for serving a large number of clients and sustaining a high rate of the monitoring data exchange. Two different test series have been performed for the IS. In both series Information Service has been represented by a single application running on the dedicated computer (dual PIV 2.2 GHz). All the clients for that IS were equally distributed over another 220 computers.

The first test series has been used to measure the mean time of a single information update operation for a single IS server in case it is used concurrently by a large number of information providers. In these tests every information provider performed one information update every second. Fig. 11 shows the results of these tests.

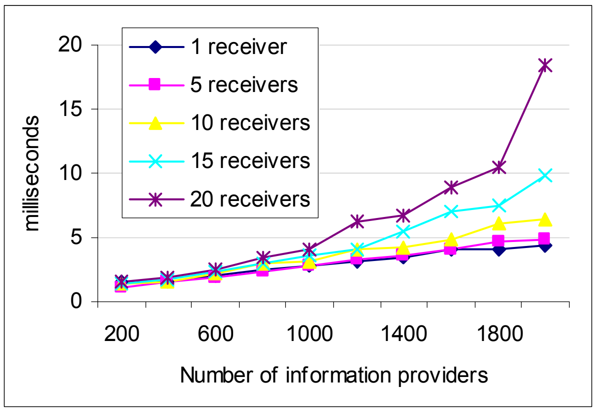

Figure 11: Mean information update time

In the second series of tests each information provider made a fixed number of information updates sequentially with no delay between them. Fig. 12 shows the results of these tests.

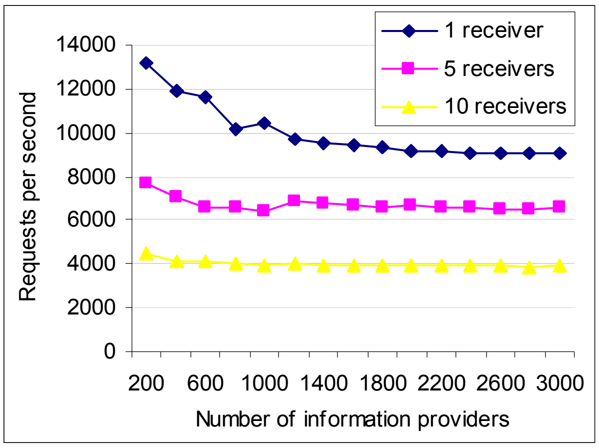

Figure 12: Number of requests per second, which can be handled by a single IS server

This test has been used to find out the maximum number of information updates, which can be handled by a single IS server depending on the number of concurrent information providers and receivers.

CONCLUSION

The control, monitoring and configuration facilities for the ATLAS DAQ system are implemented on top of the CORBA standard. CORBA is high-level object standard for distributed communication. CORBA supports most of the widely used programming and scripting languages. There is a number of free and commercial CORBA implementations in the market, which are able to interoperate with each other. The only drawback of CORBA is a complicated API, which implies a significant learning curve. In order to overcome this issue a software wrapper called IPC has been provided on top of CORBA. IPC dramatically simplifies the usage of CORBA in the scope of the ATLAS DAQ system. In addition IPC provides a very convenient way of implementing common generic facilities for the Online Software services.

The IPC API hides the details of the CORBA communication layer from the software developer. This speeds up the software development process and results in the production of the highly effective and reliable software. Our experience shows that even a beginning software developer can easily learn the IPC API in several hours.

Performance and scalability studies, which have been done on the cluster consisting of more then 300 computers, reassure the choice of CORBA as communication technology for the ATLAS DAQ control, configuration and monitoring. They show that CORBA provides the necessary performance and scalability for the distributed ATLAS DAQ.

REFERENCES

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

[2] ATLAS Technical Proposal, CERN/LHCC/94-43 ISBN 92-9083-067-0.

[3] Status of the LHC, R.Schmidt, CERN-LHC-Project Report-569, 02 Jul 2002.

[4] OMG Naming Service formal specification, http://www.omg.org/technology/documents/formal/naming_service.htm

[5] OMG Home page, http://www.omg.org/

[6] OMG IDL specification, http://www.omg.org/cgi-bin/doc?formal/02-06-07

[7] D.Liko at. al., Control in the ATLAS TDAQ system, this conference.

[8] ILU home page, ftp://ftp.parc.xerox.com/pub/ilu/ilu.html

[9] S.Kolos, Evaluation of CORBA implementations, https://edms.cern.ch/cedar/plsql/doc.info?cookie=2992923&document_id=403799&version=1.1

[10] omniORB home, http://omniorb.sourceforge.net/

[11] JacORB home, http://www.jacorb.org/

[12] I.Soloviev at. al., The configurations database challenge in the ATLAS DAQ system, this conference