THE CONFIGURATIONS DATABASE CHALLENGE IN THE ATLAS DAQ SYSTEM

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

* On leave from PNPI

Abstract

The ATLAS trigger and data acquisition system (TDAQ) uses the database to describe configurations for different types of data taking runs and different sub-detectors. Such configurations are composed of complex data objects with many inter-relations.

During the TDAQ system initialisation phase the configurations database is simultaneously accessed by a large number of processes. Such processes can be notified about database changes that happen during or between data-taking runs.

The paper describes the architecture of the configurations database. It presents the set of graphical tools, which are available for the database schema design and the data editing. The automatic generation of data access libraries for C++ and Java languages is also described. They provide the programming interfaces to access the database either via a common file system or via remote database servers, and the notification mechanism on data changes.

The paper presents results of recent performance and scalability tests, which allow a conclusion to be drawn about the applicability of the current configurations database implementation in the future DAQ system.

INTRODUCTION

The Online Software [1] is a part of the ATLAS TDAQ project. It encompasses the software to configure, control and monitor the TDAQ system but excludes the processing and transportation of physics data. It does not contain any elements that are detector specific as it is to be used by all possible configurations of the DAQ and detector instrumentation. It must co-exist and co-operate with the other sub-systems, in particular, interfaces are required to the data-flow, triggers, processor farm, event builder, controllers of detector read-out creates and Detector Control System (DCS).

The Configurations Database is the essential part of the service to configure the TDAQ system. It is extensively used by the control service to setup and to diagnose the whole TDAQ system and detectors.

The paper describes what users expect from the configurations database and why we are talking about the challenge, it's design and implementation, the results of this year performance and scalability tests [2] and the experience from on going combined test beam.

REQUIREMENTS

The initial user requirements were collected within the framework of the ATLAS data acquisition and event filter prototype –1 project [3]. Later they were reviewed and updated by the competent working group including representatives from all TDAQ systems and detector [4].

The users would like to have a comprehensive and flexible configurations database, as it is described below.

The database is accessed simultaneously by many applications during TDAQ initialization. Each application can read different set of parameters in accordance with it's needs (varying from several bytes up to tens of Mbytes) and can be notified about data changes if subscribed. The total number of applications including high-level trigger may reach in the final system several thousands.

To speak on a common language across different TDAQ systems and detectors it is desirable to have a database schema describing common data types. Such schema can be extended as required by the systems and detectors. The relations between TDAQ components and complexity of the TDAQ hardware require many cross-relationships between data items describing them. The most appropriate data model satisfying above requirements is the object data model.

The developers would like to have a mechanism to access database information via data access library

(DAL), which maps database schema to classes of used programming languages (C++ and Java) and instantiates objects of such classes from database data. In addition, the DAL's application programming interface (API) should be completely independent from the database implementation and the application's code should not be modified in case of the database technology change. Since TDAQ systems and detectors often update the database schema during development, it is necessary to automate as much as possible the DAL generation procedure.

The configurations database should provide a graphical editor, which is capable to work with database information of any type and be customizable by database developers to present dedicated data views.

The database has to provide a way to describe a set of parameters (i.e. "configuration"), which is specific for given type of run and used hardware. Each detector and TDAQ system may have several configuration descriptions for physics data taking, calibration or debug purposes. It should be simple to merge such configurations into a combine one.

The database has to describe the current state of the DAQ system. The old configurations should be stored in archives and can be accessible for example from the offline conditions database [5].

DESIGN AND IMPLEMENTATION

Common Schema

The data structure is described by the common schema, which is agreed across all TDAQ systems and detectors [6]. The common schema includes several groups of classes to describe:

- the *software releases* including programs (a binary or a script), libraries and supported platforms;
- the *hardware* including racks, crates, modules, computers, interfaces, links, cables, networks and their composition into hardware systems;
- the *control* including applications (an application corresponds to a process to be started under certain conditions; it points to computer and program and defines their parameters), resources (a hardware object or an application which can be temporary enabled/disabled) and segments (an individually controlled container object, that is composed of resources, applications and nested segments);
- the *configuration* including partition object (lists parameters and segments for given run).

Each TDAQ system or detector can extend the common schema to introduce classes describing their specific data.

Data Organisation

The data are stored in folders. The folders are organized in the tree, which corresponds to the actual hierarchy of the TDAQ systems and detectors, as it is shown below:

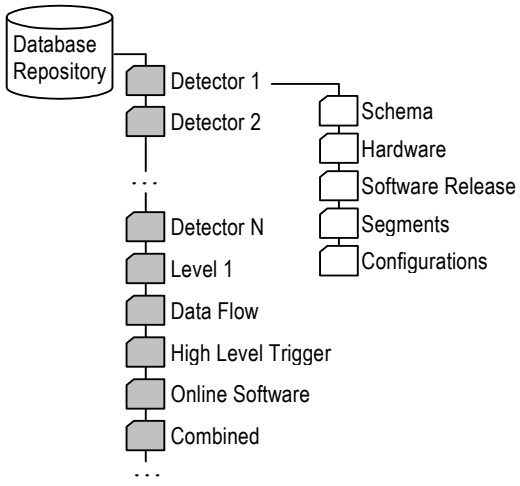

Figure 1: Example of database folders and files

Each folder is used to keep description of a TDAQ system or a detector. Each folder is organized in a similar way to easily identify the schema extensions, description of system or detector specific software and hardware, segments, and set of supported alone configurations. The *combined* folder keeps configurations, which are composed of systems and detectors segments.

Database Technology and Tools

The data themselves are stored as XML files using OKS persistent in-memory object manager [7]. One file can include another XML files. A tree of XML files describing some configuration can be build from single root file. The OKS provides the UML-like graphical editor to browse and to modify the database schema and the customizable graphical data editor.

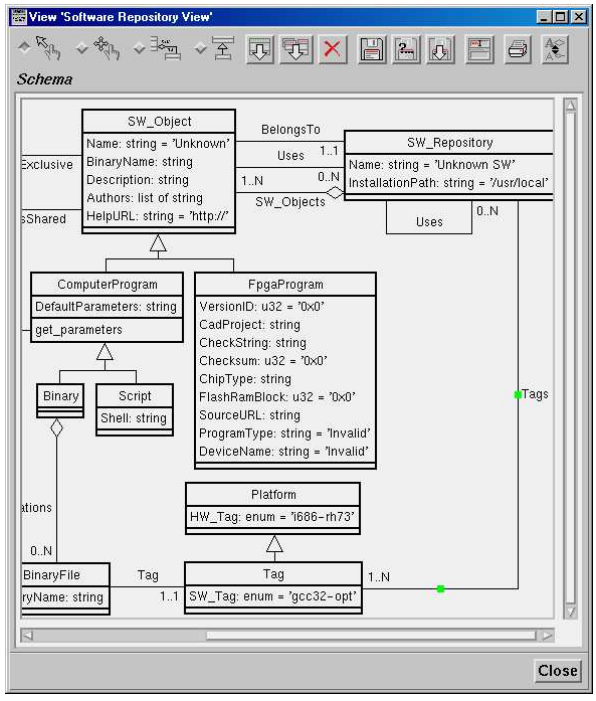

Figure 2: Example of OKS Schema editor view

The OKS data can be accessed either via native OKS C++ API using file system, or via remote database server (RDB) providing CORBA interface to OKS. Both OKS and RDB provide read-write access to data and allow subscribing on data notification.

Programming Interfaces

The user programs never use OKS or RDB directly. Instead there are two layers of database access to avoid dependency on database implementation.

The first layer provides abstract interface in C++ and Java to access database objects, to get and to put values of their attributes, to subscribe on objects modification and to get classes' meta-information. The implementations of such abstract interface are available for both OKS and RDB technologies and in future can be created for others.

The second layer is the DAL itself. It only uses abstract interface, so an implementation is completely independent from underlying database technology. The DAL is generated from the database schema and optionally can include user-defined algorithms (methods written by the user for generated C++ and Java classes).

Below there is a schema describing relations between database interfaces and users:

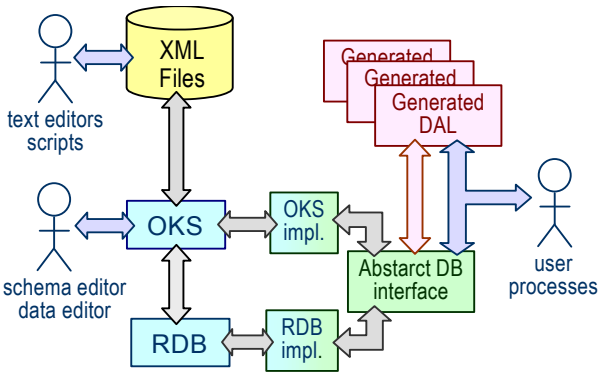

Figure 3: Database's interfaces and users

The user's code only sees generated DAL and abstract DB interfaces. An example below demonstrates their usage:

```
  // implementation object
::RdbImplementation impl;
  // database object using combined TEST configuration
::Configuration db("combined/partitions/test.data.xml", &impl);
  // find application (an object from generated DAL)
dal::Partition * p = db.get<dal::Partition>("TEST");
  // get partition default host
dal::Computer * host = app->get_DefaultHost();
  // find new host
dal::Computer * host2 = db.get<dal::Computer>("new-pc");
  // set new partition default host
if(host != host2) p->set_DefaultHost(host2);
  // commit above modification and print out result
if(db.commit()) {
   std::cout << "use host " << host2->get_Name() << '\n';
}
else { std::cerr << "ERROR: Commit failed" << std::endl; }
```

Figure 4: Example of code using DB interfaces

Database Objects

The database information is stored inside objects. An object is an instance of the database schema class. A class defines set of attributes and relationships with other objects. A class may inherit from another classes and have set of methods.

The possibility to set relations between objects simplifies the creation of full configuration description. There is special object of Partition class. It is a root of tree of other objects describing configuration. In particular, a partition object lists objects, describing top-level segments. In turn, a segment object lists nested segments, resources and applications. All objects describing configuration can be found by simple navigation starting from the root partition object. This significantly improves performance since there is no need to execute any queries, but instead to use direct links to database objects.

SCALABILITY AND PERFORMANCE TESTS

The requirements for the performance and scalability of the configurations database for the final system are very strong. It is clear that a single server is not capable to deal with thousands of clients reading different sets of configuration data at the same time. One of the possible solutions is to have single central database server providing access to the multiple RDB servers. The TDAQ and detector processes access such RDB servers and never get the data from the central server directly. The goal of recent tests was to find the performance and scalability characteristics of a single RDB server in order to be able to calculate required number of such servers during TDAQ commissioning and for the final system.

The results presented below show the time required to read from database a single object, a composite object and an array of data. The clients were run on 600 MHz PIII nodes with 100 Mbit/s Ethernet card, while the server was run on 2.4 GHz dual PVI node with 1 Gbit Ethernet card. All clients started simultaneously, established connection with RDB server, got required data and closed connection. The measured times are presented depending on number of clients from 10 to 210.

Read single object

Below there are results of reading small single object by unique identity and by query (no index). The average value linearly grows for both types of object access.

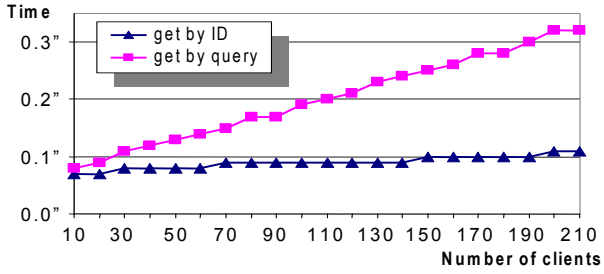

Figure 5: Single object read results

Read composite object

The composite object used for test was composed of 2042 small nested objects simulating control hierarchy of TDAQ for final system. Each small object was read by a separate network operation. Below there are average and maximum times for such test, which are quite close.

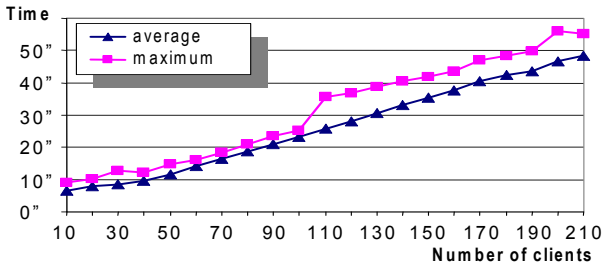

Figure 6: Composite object read results

For 210 clients the server processed around 430,000 requests in 55 seconds, or about 8,000 requests per second.

Reading arrays of data

Below there are results of reading 4 objects containing arrays of integers of different sizes. The figure presents maximum times only, the average ones are in two times smaller.

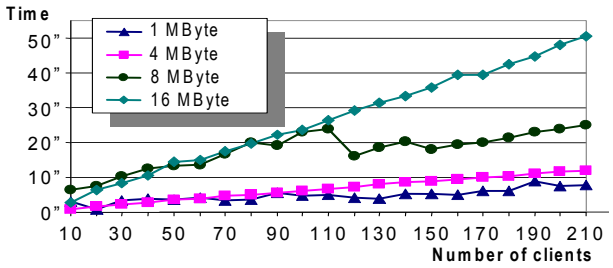

Figure 7: Arrays of data read results

With such test the performance of the RDB server reached more than 50% of raw network performance (3.36 Gbytes passed in 50 seconds, or 538 Mbits/s).

TEST BEAM EXPERIENCE

All TDAQ systems and detectors use the presented database to describe their test beam configuration data. The described test beam system includes more than 50 computers and several racks of electronics. Each detector and TDAQ system has set of segments, which are shared to build set of combined configurations.

During the test beam there were several major changes in the database schema making new data logically incompatible with old ones. This resulted creating several co-existent versions of databases.

Changes of the database files are stored in CVS. It is done by automatic cron job or by explicit user request. This allows easily checking out a configuration to see what parameters were used for past runs.

CONCLUSIONS

In order to provide required functionality for the final ATLAS TDAQ system the configurations database has to meet challenging requirements. It needs to have high performance for simultaneous access from multiple clients, to support object model for data description, to provide automatically generated data access libraries, to have graphical tools for the database schema design and the data modification.

The present implementation of the configuration databases is done on top of OKS. The testbeam experience has shown it provides required flexible data organisation. The recent performance and scalability tests have proved it can be used in combination with multiple RDB servers for implementation of the final system. At the same time the design of the configuration database includes clear separation between programming interfaces and database technology backend, so in future the OKS can be replaced by something also without disruption of the user code.

REFERENCES

[1] *Atlas Online Software*, http://cern.ch/atlas-onlsw

[2] *ATLAS DAQ/Event Filter Prototype -1 Project*, http://cern.ch/atlas-onlsw

[3] *ATLAS DAQ/Event Filter Prototype "-1" Project*, http://atddoc.cern.ch/Atlas

[4] *2004 Large Scale and Performance Tests of the ATLAS HLT/DAQ Software*, http://cern.ch/atlas-tdaq-large-scale-tests

[5] *Conditions DB*, http:// lcgapp.cern.ch/project/CondDB/

[6] *Common Configurations Database Schema,* http://cern.ch/Atlas-onlsw/components/configdb#Schema

[7] *OKS User's Guide, I. Soloviev, 2002*, EDMS note, http://edms.cern.ch/file/403060/2/