Visualization of historical data for the ATLAS detector controls - DDV

Visualization of historical data for the ATLAS detector controls - DDV

J Maciejewski[1] and S Schlenker[1]

[1] CERN, Geneva, Switzerland

E-mail: {Julian.maciejewski, Stefan.Schlenker}@cern.ch

Abstract. The ATLAS experiment is one of four detectors located on the Large Hardon Collider (LHC) based at CERN. Its detector control system (DCS) stores the slow control data acquired within the back-end of distributed WinCC OA applications, which enables the data to be retrieved for future analysis, debugging and detector development in an Oracle relational database. The ATLAS DCS Data Viewer (DDV) is a client-server application providing access to the historical data outside of the experiment network. The server builds optimized SQL queries, retrieves the data from the database and serves it to the clients via HTTP connections. The server also implements protection methods to prevent malicious use of the database. The client is an AJAX-type web application based on the Vaadin (framework build around the Google Web Toolkit (GWT)) which gives users the possibility to access the data with ease. The DCS metadata can be selected using a column-tree navigation or a search engine supporting regular expressions. The data is visualized by a selection of output modules such as a java script value-over time plots or a lazy loading table widget. Additional plugins give the users the possibility to retrieve the data in ROOT format or as an ASCII file. Control system alarms can also be visualized in a dedicated table if necessary. Python mock-up scripts can be generated by the client, allowing the user to query the pythonic DDV server directly, such that the users can embed the scripts into more complex analysis programs. Users are also able to store searches and output configurations as XML on the server to share with others via URL or to embed in HTML.

1. Introduction

The ATLAS Detector Control System (DCS) is a SCADA based in WinCC OA which is responsible for safe detector operation. DCS supervises all sensors, monitors all operational parameters, delivers information about online state and abnormal behavior and can also execute operator commands. All ATLAS DCS systems archive the detector parameters within the online database, while at the same time they can present a historical view of these parameters serving online monitoring needs. The database structure is defined by the proprietary schema of the WinCC OA RDB archiver application. For security and performance reasons, the online database is accessible only to systems within the ATLAS Control Network (ATCN). Nevertheless, a backup database - a complete replica of the online database - is in place to serve users within the CERN General Public Network (GPN). DDV[1] is connected to the backup database and aims to cover the needs for offline access of operational data worldwide. This backup database uses the Oracle Active Data Guard (ADG) mechanism and prevents data loss and downtime simply and economically by maintaining a synchronized physical replica (backup database) of a production database (online database) without impacting database performance. ADG uses a gateway between the ATCN and GPN to create a copy of the database. DDV accesses data by performing SQL queries to the backup database.

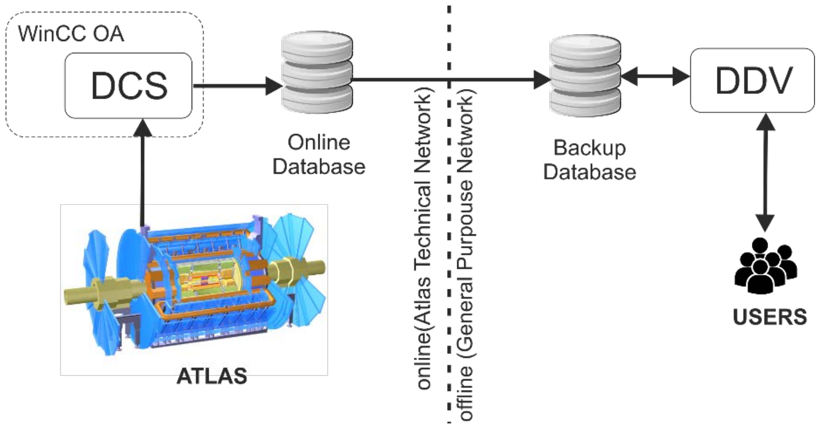

WinCC OA
DCS
Online Database
online(Atlas Technical Network)
offline (General Purpouse Network)
Backup Database
DDV
USERS
ATLAS

Figure 1. DDV data flow.

2. Project architecture

Figure 2 illustrates the different building blocks that DDV is composed of. A SQlite[3] database is used at the server side and caches all the DCS metadata updated daily from the Oracle database. Furthermore, SQLite is configured in memory mode which means instead of hard disk files, information can be stored in memory file objects, optimizing the response time of the requests. At the other end, the user’s web browser represents the DDV client which fetches the data from the DDV server via HTTP requests. The client has been developed using Vaadin[4] - a Java web application framework for creating rich and interactive web applications running on a Tomcat web server and supporting all common web browsers. The aim of the selection component is to allow easy navigation among the metadata (e.g. column tree view widget or search panel) as well as constructing metadata requests to the server. The core component is responsible for building proper data requests to the server acquiring data and sending it to the chosen output. Those requests combine selected metadata and time periods. Due to its flexible interface and its generic and modular approach, DDV could be easily used in any control system which uses the WinCCOA RDB Archiver.

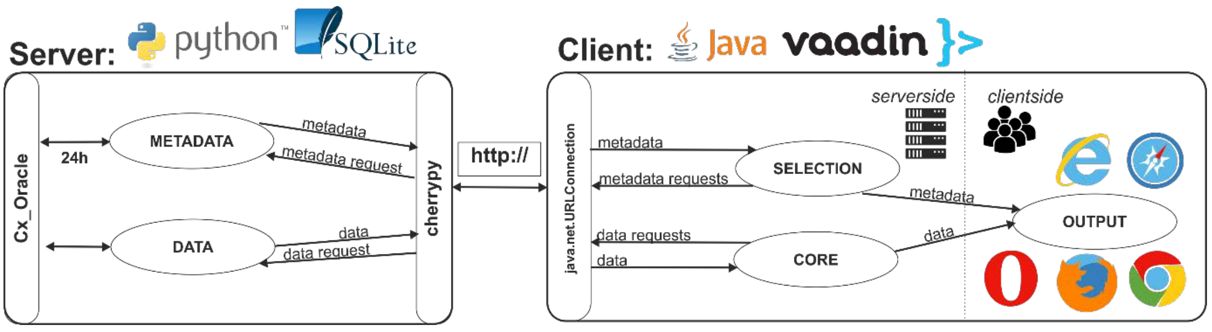


Figure 2. DDV architecture.

3. Server

The DDV server was written in the Python programming language and it was chosen for its readability and long term maintainability. Besides that, Python has a huge standard library especially useful for data manipulation. For data retrieval, cx_Oracle[5] was chosen which is a module that enables access to Oracle databases and conforms to the Python database API specification. The SQL queries are prepared as simple strings. There is no framework in use for mapping an object-oriented domain model to a relational database. This allows for better control of the query and enables easy optimization. The ROOT[8] Python library is used to prepare and serve data as ROOT files on request. CherryPy[6] – a pythonic, object-oriented HTTP framework – is used to serve data to any client within the CERN network.

3.1. SQLite DB metadata cache

The metadata consists of unique identifiers for each data point element (DPE) of the control system along with an alias and a description. These metadata values may change over time, e.g. when cabling changes the mapping of power supply channels to detector elements, and these changes are reflected by intervals of validity of the metadata entries. When a client queries for metadata or data, the evolution is automatically taken into account based on the requested time period and selected type of identifier.

Once per day, all metadata stored in the Oracle database is copied to the SQLite database cache inside the DDV server along with its evolution, i.e. also deleted identifiers are stored. The aim of the metadata cache is to reduce the number of queries to the database for this rather static data as well as accelerating the response time for client requests.

3.2. DB protection mechanism

Intending to serve a high number of users, DDV aims to be more than an interface to the database data. DDV validates and certifies each request with a minimum-response-time cost and finally propagates it to the DB. This protection mechanism is organized in three levels (diamonds in Fig. 3). Firstly, the total number of data point elements is limited. The second protection mechanism performs a light test-request with a time period significantly shorter than the one requested. The number of returned rows is extrapolated to the full query time and a decision is made on whether this request is acceptable – if not the query is canceled. In the third condition, the maximum duration of the query is limited to two minutes, beyond that the query is again canceled. In both cases the user is asked to change the query parameters (i.e. decrease query time, quantity of selected DPEs). The decision parameters were chosen empirically based on users' queries and database performance limits. Users who are not satisfied with the constraints can write their own scripts which split the query on parts and question the server directly. According to our observations, these are very rare cases.

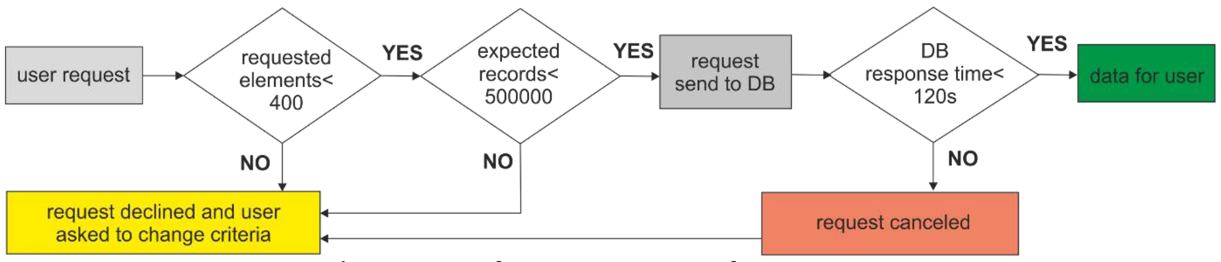
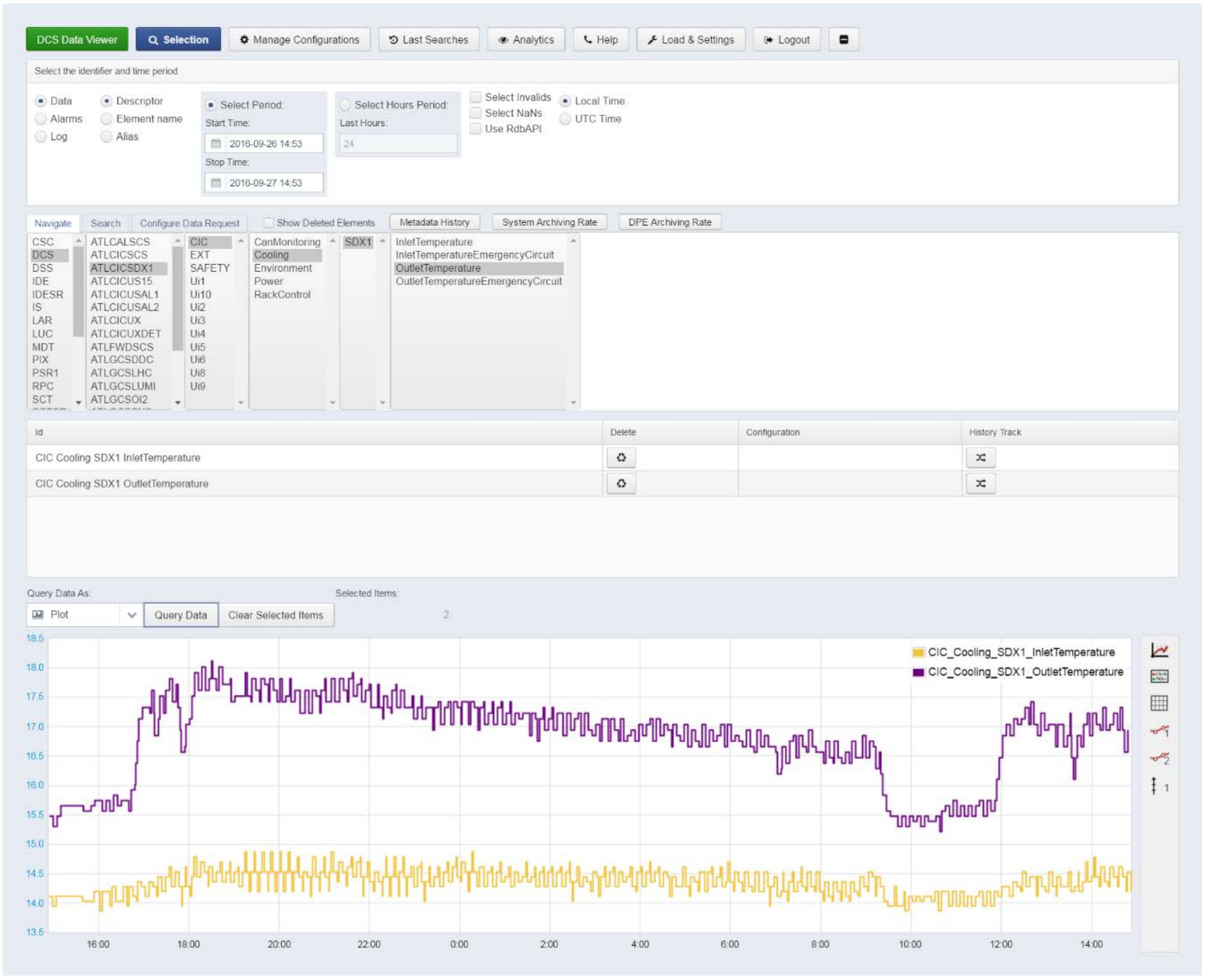


Figure 3. Database protection mechanism.

Connection pools of permanently active database connections are used to balance performance and resources for database queries on the server. Thus when a client ends a session, its assigned connection is not closed but only returned to the pool and can be reused for the next session.

4. Client

The main DDV user interface (see Fig. 4) is a web application accessible world-wide – the DDV client. It is implemented using Vaadin – a Java web application framework designed to facilitate the creation of high quality web-based user interfaces. Vaadin delivers a set of easily customizable user interface components/widgets and themes for controlling the appearance. It additionally hides asynchronous communication between the user browser and server. From the available features we profit from URL routing, calendar widgets, and Java indexed container mapping to table widgets.



Id	Delete	Configuration	History Track
CIC Cooling SDX1 InletTemperature			
CIC Cooling SDX1 OutletTemperature			

Figure 4. Client web interface.

The Model View Presenter (MVP) architecture has been used as software developing patterns. The pattern binds data (the Model) to the chosen view and it contains all the methods (Presenter) called when a user interacts with the view.

4.1. *Selection*

Navigation within the metadata is done by a column tree widget. This choice directly corresponds to the hierarchical naming conventions used for the parameter identifiers and minimizes the display latencies for searches within the metadata. To find and select specific DPEs, users can use a search engine widget implementing wildcard or regular expression matching.

4.2. Metadata evolution
The evolution of metadata is visualized in a graphical timeline or table showing the intervals of validity of the respective parameter mappings between identifier types (DPE=element, description, or alias), as shown in Fig. 5.

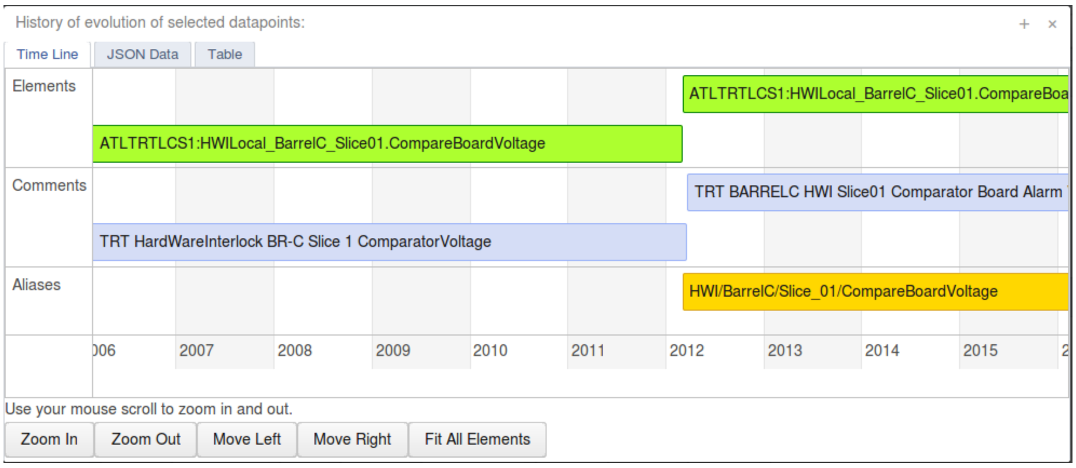

History of evolution of selected datapoints:
Time Line
JSON Data
Table
Elements
ATLTRTLCS1:HWILocal_BarrelC_Slice01.CompareBoa
ATLTRTLCS1:HWILocal_BarrelC_Slice01.CompareBoardVoltage
Comments
TRT BARRELC HWI Slice01 Comparator Board Alarm
TRT HardWareInterlock BR-C Slice 1 ComparatorVoltage
Aliases
HWI/BarrelC/Slice_01/CompareBoardVoltage
)06
2007
2008
2009
2010
2011
2012
2013
2014
2015
2
Use your mouse scroll to zoom in and out.
Zoom In
Zoom Out
Move Left
Move Right
Fit All Elements

Figure 5. Metadata evolution shown on the timeline.

4.3. Client- or server-based configuration
DDV offers the possibility to save the current configuration of selected metadata and output settings to an XML file. The file is stored on the server in either the user folder, a shared folder within a given subdetector or the global folder available for every user. The configuration file can then be downloaded when required or the user can share it as a URL. By accessing the URL, the configuration will be loaded and stored data queries are executed. Such URLs could then for example be embedded in inline frames (iframe) of HTML pages. The interactive control elements of DDV can be hidden on demand using additional URL parameters, e.g. to show only the resulting plot or table.

4.4. Quick history reload
The query history is stored for each user on the server and can be quickly reloaded to avoid repetitive configuration and selection tasks.

4.5. Insert rate evaluation
The calculation and table representation of database insert rates of all parameters for each subsystem was implemented to quickly spot parameters with excessive insert rate and simplify subsequent investigation.

4.6. Outputs
To satisfy the needs of users, DDV provides five different outputs. The default output of DDV Client is a JavaScript Plot implemented using the JsPlot library[7]. The user can customize plot settings like colors, axes and line types and then store those settings in the aforementioned configuration file.

The second output type is the Lazy Loading Filter Table. This widget provided by Vaadin stores all the data on the client's server side and loads only the chunk visible on the screen. It has filter fields on each column such that a user can quickly filter on output data such as time periods or parameter names. Apart from the web browser outputs, DDV also offers the possibility to download ROOT-structured output. Further, simple ASCII file format is available including information of the archived parameter, its timestamp and its recorded value.

Finally, the user can download a basic python script which includes direct queries to the server and a mock-up loop over the query period. The aim is to give the user hints on how to develop their own scripts and fetch more data over a longer period when used inside the CERN network.

Since the output module has a rather thin interface to the other DDV client modules, other output types can be added easily in the future according to user requirements.

5. Summary

The deliberately modular design of the tool in the initial phase of the development has paid back well in accomplishing stable service operation while continuing the development. Further, the strict separation of server-based database access within the CERN network and the user interface provided as web application client proved to be a good choice. The client provides easy navigation amongst the control system metadata and its evolution and presents requested data with a set of output plugins meeting most user requirements. The selected data representation can be accessed and shared easily worldwide. Stable service operation and maintainability of the client and server are achieved with proper software management tools, e.g. the project is hosted on a Gitlab[9] server instance exploiting its continuous integration solution to automatically generate deployment packages. Finally, building on the WinCCOA RDB Archiver database schema, DDV could be easily adapted to other WinCCOA-based control systems.

References

[1] The ATLAS Collaboration 2008, JINST 3, S08003
[2] Schlenker S et al ICALEPCS 2011 MOBAUST02
[3] SQLite, http://sqlite.org
[4] Vaadin, https://vaadin.com
[5] cx_Oracle, http://cx-oracle.sourceforge.net/
[6] CherryPy, http://cherrypy.org
[7] JsPlot, http://jsplot.sourceforge.net/
[8] Root, http://root.cern.ch
[9] Gitlab, http://gitlab.com