The Online Histogram Presenter for the ATLAS experiment: a modular system for histogram visualization

Andrea Dotti[(1,2)], **Paolo Adragna**[(3)] **and Roberto A. Vitillo**[(4)]

[(1)] CERN , CH-1211 Genve 23 Switzerland
[(2)] Universtià degli Studi di Pisa, Dipartimento di Fisica "Enrico Fermi", Largo Bruno Pontecorvo 3, 56127 Pisa, Italy
[(3)] Physics Department, Queen Mary, University of London Mile End Road London E1 4RP United Kingdom
[(4)] INFN Sezione di Pisa, Ed. C Largo Bruno Pontecorvo 3, 56127 Pisa, Italy

E-mail: andrea.dotti@cern.ch

Abstract. The Online Histogram Presenter (OHP) is the ATLAS tool to display histograms produced by the online monitoring system. In spite of the name, the Online Histogram Presenter is much more than just a histogram display. To cope with the large amount of data, the application has been designed to minimise the network traffic; sophisticated caching, hashing and filtering algorithms reduce memory and CPU usage. The system uses Qt and ROOT for histogram visualisation and manipulation. In addition, histogram visualisation can be extensively customised through configuration files. Finally, its very modular architecture features a lightweight plug-in system, allowing extensions to accommodate specific user needs.

After an architectural overview of the application, the paper is going to present in detail the solutions adopted to increase the performance and a description of the plug-in system.

ATL-DAQ-PUB-2009-009
18 September 2009

1. Introduction

The challenging experimental environment and the extreme complexity of modern high-energy physics experiments make online monitoring an essential tool to assess the detector health and the quality of the acquired data. The Online Histogram Presenter (OHP) is the ATLAS [1] tool to display histograms produced by the online monitoring system [2]. The system uses Qt [3] and ROOT [4] for histogram visualization and manipulation, extensively customizable through configuration files. The Online Histogram Presenter unifies the approach to histogram visualization inside the ATLAS online environment in a general purpose, highly configurable, interactive application.

In the ATLAS online monitoring framework histograms are produced by Monitoring Applications that sample events at various level in the data-flow chain. Events are unpacked, decoded and histograms are filled with reconstructed quantities. Histograms are published in the OHS (Online Histogramming Service) servers. The OHP interacts with OHS to retrieve the histogram objects.

The OHP application is currently being used in the ATLAS control room for the commissioning of the detector with cosmic ray muons.

2. Overview of the application

OHP has been designed to cope with a large number of histograms. Histograms are presented to the shifters organized in a tree structure according to their name or in predefined windows that are automatically updated when histograms are published on the OHS server(s). If reference histograms are provided these are presented superimposed to the received histograms. All aspects of OHP behavior, including the drawing options, can be configured through XML configuration files.

OHP design is centered around the *Core* that serves as controller of the different sub-components, the most important being the *Histogram receiver*, designed to interact with OHS and the *Plug-in System*, used to load and control the graphical user interface (GUI) elements.

The application, written in C++, is part of the ATLAS TDAQ (Trigger and Data Acquisition) software framework. It relies on OHS API to interact with the OHS servers. This API allows for exchanging messages and objects between software applications deployed in a highly distributed environment. Inter process communication in TDAQ is based on the CORBA system [5]. OHP uses three functionalities of OHS API: *notifications*, *histogram getter*, *command sending*. Notifications are messages sent by OHS server to clients when a histogram is made available, while histogram getter permits a client application to retrieve a histogram object from OHS servers. Getters are also responsible to convert histogram objects from OHS internal format to the format requested by the client. In OHP case the histograms are converted to the ROOT format. In section 3.3 the details of notification propagation and histogram retrieval is presented. Command sending enables OHP with the possibility to send simple commands, in the form of strings, to the monitoring applications. Commands are exchanged between applications using the underlying CORBA layer, thus OHP can use the monitoring application name to identify the receiver of the command, OHS will forward this request to the actual component. It is worth to note that no acknowledgment of the command is possible in the current implementation and no guarantee is given that the receiving application is correctly interpreting the command string. For these reasons the commands are used to exchange simple, non critical and asynchronous messages between OHP and monitoring application (example of commands include: *Histogram Reset Request*, *Change Axis Limits Request*, *Force Publish Request*). Additional information on ATLAS online monitoring components and technical details can be found on the ATLAS Monitoring Working Group website [6].

3. Requirements

The requirements that drove the design of the application have been collected during the experience of the ATLAS test beams. In 2004 a prototype of the application has been deployed and the current implementation of OHP (version number 3.5) is the result of the experience gained during the commissioning of the ATLAS detector.

3.1. Functional Requirements

- **FR 1** OHP shall allow the user to select a configurable set of histograms to be automatically retrieved from the OHS for further manipulations.
  Manipulations include displaying histograms in a graphical interface. Operating on a specific histogram is known as shifter mode.
- **FR 2** OHP shall allow the user to browse the whole content of the OHS server or a subset of it.
  Browsing histograms is known as expert mode.
- **FR 3** OHP shall interact with histogram producers.
  The interaction, mediated by the OHS, implies the possibility of changing parameters on the production side (such as histogram scales), resetting content, turning on/off production.

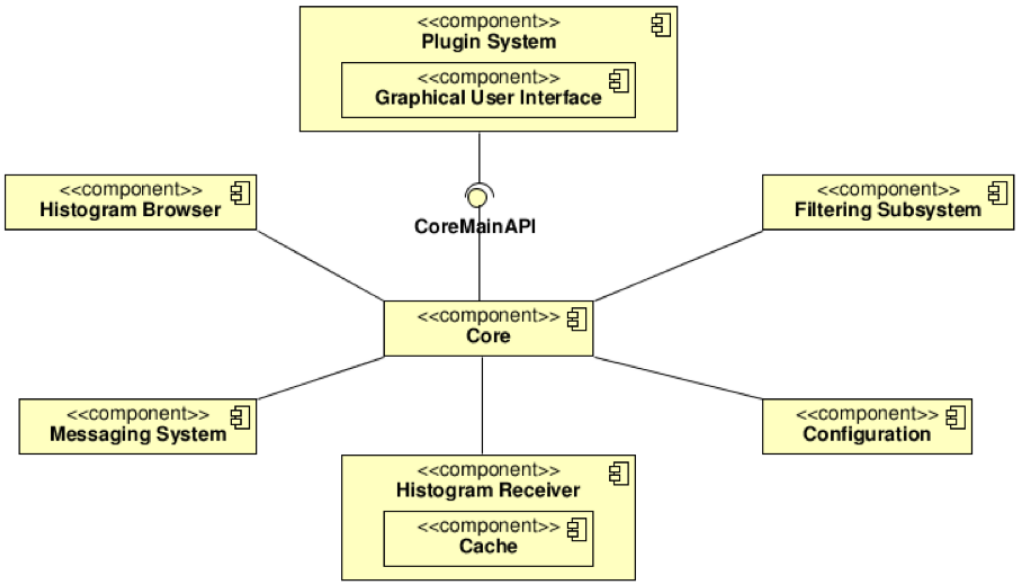

Figure 1. Component Diagram of OHP: the most important components of the application are displayed. OHP has a central component, the Core, responsible to provide services to the other components.

- **FR 4** OHP shall manage reference histograms.
  Reference histograms represent the trend of a monitored quantity as it should be in standard running conditions.
- **FR 5** OHP shall support the execution of plug-ins.
  Plug-ins provide additional features, helping OHP perform specific functions.
- **FR 6** The graphical interface of OHP shall allow the user to interact with the displayed histograms to perform operations.
  Example interactions are zooming the content, fitting the plot and changing the graphical appearance of the plots.
- **FR 7** The graphical interface of OHP shall allow the user to set the graphical attributes of the plots.
  An example of graphical attribute is the scale of an axis, which may be turned from linear to logarithmic.
- **FR 8** The graphical interface of OHP must automatically display histograms when they are available.
  The evolution of the plots is presented to the user in real-time.

3.2. Non-Functional Requirements

- **NFR 1** OHP shall be integrated into the ATLAS TDAQ system.
- **NFR 2** OHP shall be independent from histogram producing, as different type of producers, developed independently, are expected.
- **NFR 3** OHP shall have the smallest possible impact on network traffic.

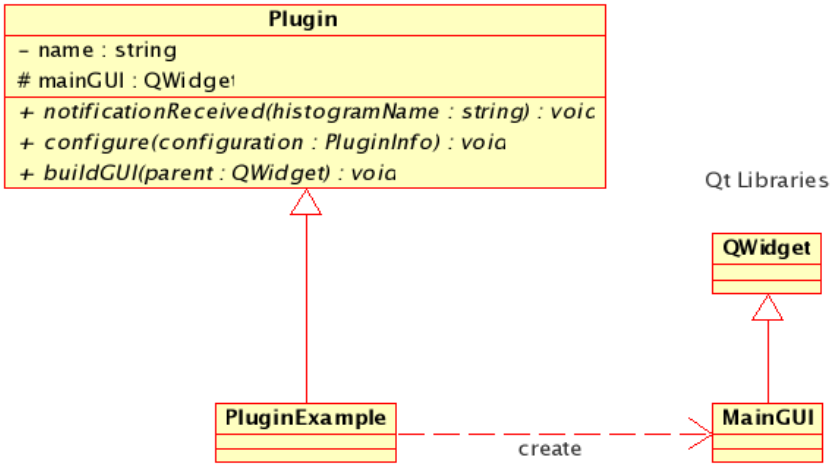

Figure 2. Class diagram for a generic plug-in. The user should implement a class inheriting from the base class Plugin. The user creates the layout and defines the behaviour of the GUI using the Qt libraries.

3.3. OHP Workflow and General Design

OHP works with a mixed push/pull mode in the interaction with the OHS server(s). OHP subscribes for set of histograms to the available OHS servers. Subscription can be defined with the help of regular expressions. When a provider publishes a histogram on one OHS server a *notification* (a concatenation of the server, provider and histogram names) is forwarded to OHP if it matches the subscriptions.

Notifications are pushed from the OHS server to OHP every time a histogram is updated, but only when the histogram is actually used (i.e. displayed) the histogram object is retrieved from the OHS server.

Notifications contain the name of the histogram that has been updated. The histogram name is searched for in the set of names of the on-focus histograms (histograms that are currently used). If found the notification is forwarded to the GUI for further processing, otherwise it is discarded. Only at the very end the GUI pulls the histogram object from the OHS server. This mixed push/pull approach optimizes the use of network resources since only when the histogram is going to be displayed on the screen the histogram object is retrieved from the OHS server.

To further optimize the use of network resources a cache system has been implemented. When a histogram object is retrieved a copy of it is stored in the OHP cache. When the GUI requires a histogram object, this is first searched in the cache and only if it is not found (or an updated version exists on the OHS server) the requests is forwarded to the OHS server.

3.4. Components of the Application

The Online Histogram Presenter is divided into the following components, shown in figure 1:

- **the Core** The Core is the central component of the application. Its most important task is to provide services to the other components, in particular the plug-ins, which use the Core to request histograms from OHS or to interact with the providers.
- **the Histogram Receivers** Receivers accept notifications and retrieve histograms from OHS servers on request. Multiple subscriptions are possible: each subscription creates a dedicated receiver.

- **the Cache** Caches are used to store received notification and histograms. Each histogram receiver owns a cache. The cache is a key component to minimise the network traffic.
- **the Filtering Subsystem** Filters sift notifications related to active plots. Active plots are the one on-focus, i.e. the one that the user is actually manipulating.
- **the Histogram Browser** The histogram browser is a special container which helps OHP to store notifications, allowing user the browsing of a subset of the OHS.
- **the Plug-in System** Plug-ins enhance the flexibility of the OHP providing the user with the possibility of including additional code at run time.
- **the Graphical User Interface** The graphical user interface (GUI), which is composed of one or more plug-ins, makes Qt and ROOT coexist nicely, providing the user with all the functionalities to interact with the histograms.
- **the Configuration** Both the core and the plug-ins can be configured via a text file in XML format. The configuration subsystem is in charge of parsing the file, checking its correctness, and preparing all the information necessary to bring the program up and running.
- **the Messaging Service** A simple messaging system allows plug-in to exchange some limited information in the form of strings.

4. The role of the Core

The central component of OHP architecture is the Core: its main functionality is to manage the communication with OHS servers. The Core is defined as the manager of all basic services (the cache sub-system, the histogram receivers, the plug-in system, etc.), the details of each service are hidden to the users (plug-ins): they interact with OHP sub-systems through a proxy artifact that exposes a uniform API: an instance of the *CoreMainAPI* class, implemented following the singleton design pattern.

The Core is organized as a finite state machine and some functionalities are available only in some states. For example: retrieval of histograms is allowed only when OHP is in *running* mode, while sending of commands is allowed in *running* and *paused* states. Upon request for a specific action, the Core checks that the particular action is allowed in the current state.

The Core is also responsible for loading the configuration file, parsing its content and loading the dynamic libraries containing the plug-ins. Once plug-ins have been instantiated they have access, through CoreMainAPI class, to all Core functionalities. These include:

- *Histogram retrieval*: the Core will first search histograms in local caches and if no valid entry can be found, the request is forwarded to the appropriate OHS servers.
- *Command sending*: the Core simply forwards the request to OHS server.
- *Histogram names query*: since the Core has access to the Histogram Browser components, plug-ins can query the existence of histograms which names match regular expressions.
- *Configuration query*: plug-ins can request specific information contained in the configuration file, the Core will retrieve this information from the Configuration sub-system and pass it to the plug-in. For example drawing options (style, logarithmic scales, color settings, etc.) can be passed to plug-ins in this way.
- *State transition requests*: some state transitions can be requested by plug-ins, for example a *pause/resume* or *stop/start* state transition can be requested automatically in case of errors or explicitly by the shiftr interaction with the GUI elements. Some transition requests are not allowed (e.g. *(un)load configuration file* and *apply configuration*) since these transitions could have major effects on the plug-ins themselves.

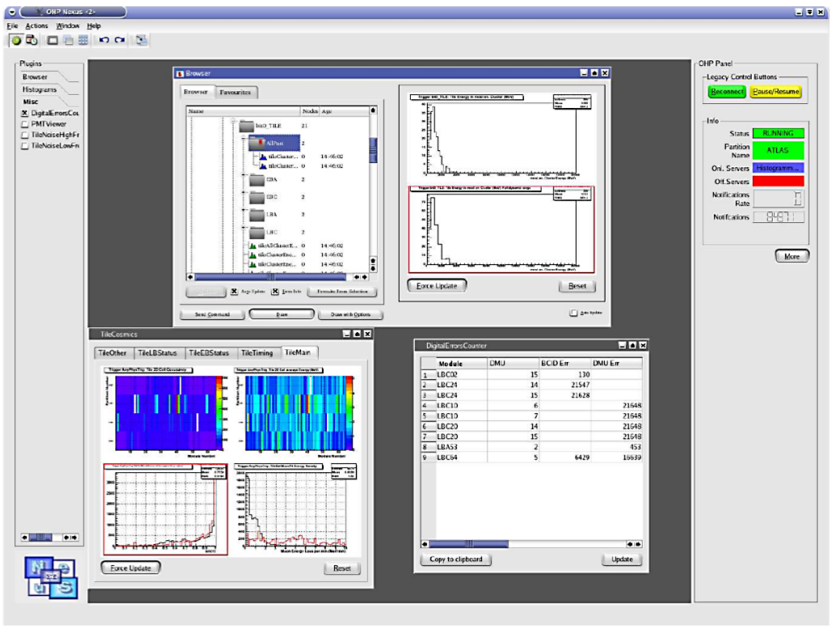

Figure 3. Some examples of plug-ins. The figure shows the MDI interface with several other plug-ins running. In addition to canvases a plug-in shows, in tabula form, the information extracted from histograms. Bottom left plug-in shows histograms in a canvas with several drawing options configured. A Browser plug-in is also active in the top area.

5. The plug-ins system

OHP very modular architecture features a lightweight plug-ins system, allowing extensions to accommodate specific needs. Plug-ins are compositions of graphical objects (based on the Qt graphics libraries). A (non-graphical) object, inheriting from OHP base-class Plugin creates the graphical widgets of the plug-ins GUI. One of the widget (in general the main window) is elected to be the plug-ins GUI interface 2.

Plug-ins can instantiate OHPCoreInterface objects through which they have access to OHP functionalities. A plug-in can: retrieve OHP configuration, retrieve histograms, send commands to histogram providers, send messages to other plug-ins.

5.1. General Purpose Plug-ins

A set of general purpose plug-ins have been developed by the authors: a histogram browser, different canvases to display histograms, a Multi Document Interface (MDI) to organize and activate other plug-ins. Users have also developed specific plug-ins to show sub-system specific information. Figure 3 shows an example of a OHP configuration currently used by the ATLAS collaboration.

The general purpose plug-ins include:

- **The Histogram Browser** According to *ATLAS online monitoring histogram naming convention* the histogram names are composed of multiple strings divided by the "/" character in a way similar to a Posix path. Histogram names are thus visualized in a tree structure. In addition the user can group folders or single histograms in *favourites*, in analogy with the concept of the bookmarks of modern web browsers.
- **The Canvases** Histograms are displayed in *canvases*, it is possible to fully customize histogram visualization through the configuration file. Not only the number and the

organization of the histograms in each canvas can be specified, but also attributes such as drawing styles and reference histogram location. ATLAS software include a system to automatically test histogram content and provide a result in a form of a quality flag (*green* for success, *red* for failure, *yellow* for cases that require user attention) [7]. Canvases can be configured to display the color code representing this quality flag.

- **MDI** A special plug-in has been developed to organize and control other plug-ins. It has been developed as a Multi Document Interface: a virtual desktop in which all other plug-ins are displayed. Through this interface it is possible to show or hide the GUI of the plug-ins, organize them in categories and visualize the status of the application and of the OHS servers. Through menus it is also possible to save histograms in ROOT files, print canvases and access online manuals.

5.2. User Specific Plug-ins

Several plugins have been developed independently by sub-system experts. Examples of sub-system specific plug-ins include:

- **TriP** The Trigger Presenter displays quantities and histograms related to the performances of the trigger farms. Rates of the three trigger levels are also shown in the TriP GUI.
- **Noise Tables** The hadronic calorimeter sub-detector prepared plug-ins that summarize the content of several histograms in tables. Histograms are scanned for bins above threshold and summaries are converted to a tabular form that can be copied to the clipboard and send to the experts as plain text.
- **ROOT Macro interface** The high level trigger experts prepared plug-ins that can show summaries of the trigger information. The plug-ins delegate to ROOT macros, the calculation of summary information, in this way even non OHP experts can prepare user code for histogram analysis and manipulation.
- **Python scripts interface** In a similar way electromagnetic calorimeter experts prepared a plug-in that re-use already implemented python code to manipulate histograms. In this way code prepared in other contexts can be re-used in OHP.

Examples of the different plug-ins can be seen in figure 3.

6. Optimization

As discussed in section 3.3 the application has been designed to minimize the use of network resources. A mixed push/pull approach in the interaction with the OHS servers and the use of caches reduces the demand of network bandwidth.

In addition the application has been optimized to reduced memory and CPU usage footprint.

6.1. Memory usage optimization

OHP internal cache has a fixed size, when the cache is full elements have to be discarded. OHP typical work sessions suggests the use of a deletion policy with both characteristics of LFU (Least Frequently Used) and LRU (Least Recently Used) deletion policy. This mixed behavior has been obtained implementing the high performance Adaptive Replacement Cache that can self-tune internal parameters to dynamically respond to changing access patterns [8].

6.2. CPU usage optimization

OHP needs to perform several comparison between strings (histogram names). To increase the application performances all comparisons between strings have been changed into comparisons between integers (hashes). A large set of ATLAS histogram names have been used to test different hashing algorithms [9]. These can been classified by performance (time spent to

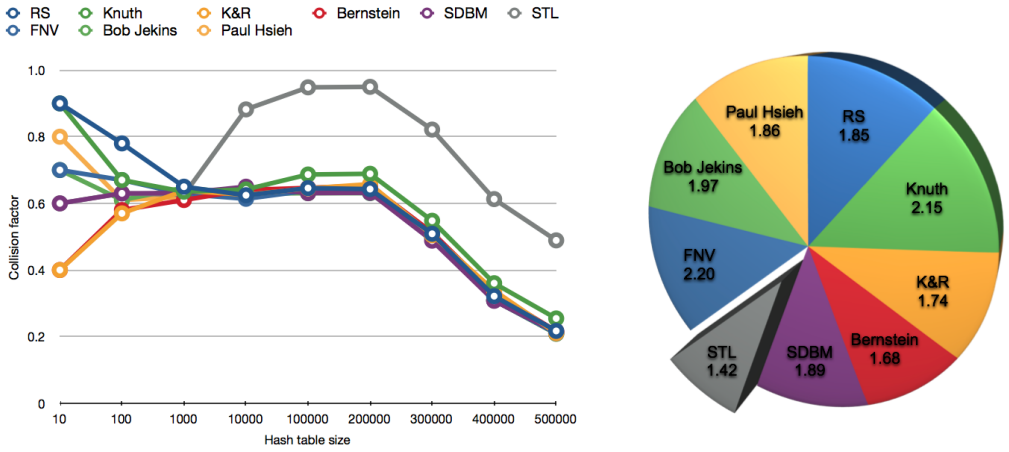

Figure 4. Test results for different hashing functions. The tests have been performed on a set of real ATLAS histogram names. Left: the collisions factor as a function of the hash table size for different hashing functions, smaller factor is better. Right: time spent (a.u.) by different algorithms, smaller numbers are better. The Bernstein algorithm has good performances while keeping small the number of collisions.

compute the hashes of the entire set) and number of collisions (distinct strings with the same hash). For a given hash function, the collision factor depends on the *size* of the domain (the set of the input strings) as well as the size of codomain (the set of all possible hash values). The domain has the infinite cardinality of natural numbers, while the codomain has a large, but finite, cardinality (the largest value that can be expressed with an integer variable). We expect smaller collision factors for larger codomains. We studied the number of collisions for different algorithms as a function of the hash map table size and searched a fast algorithm that minimizes the collision factor. The Bernstein algorithm shows the best results for ATLAS histogram naming convention (see figure 4).

References

[1] The ATLAS Collaboration 2008 *J. Instrum.* 3 S08003
[2] Vandell W *et al* 2007 *Nuclear Science, IEEE Transactions on* Vol. 54 Issue 3 **2** 609-615
[3] Trolltech Inc. , available online: ftp://ftp.trolltech.com/qt/pdf/3.0/whitepaper-a4-web.pdf
[4] Brun R and Rademakers F "ROOT: An Object-Oriented Data Analysis Framework" 1998 *Linux J.* 51
[5] CORBA webpage, http://www.corba.org/
[6] Monitoring Working Group webpage: http://atlas-tdaq-monitoring.web.cern.ch/atlas-tdaq-monitoring/
[7] Kolos S, Corso-Radu A, Hadavand H, Hauschild M and Kehoe R 2008 *Journal of Physics: Conference Series* 119 **2** 022033
[8] Megiddo N and Modh D S 2004 *IEEE Computer* 37 **4** 58-65
[9] B. J. McKenzie, R. Harries, T. Bell 2006 *Software: Practice and Experience* 20 **2** 209-224