The DAQ system for the AEgIS experiment

IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 032014 doi:10.1088/1742-6596/898/3/032014

The DAQ system for the AEgIS experiment

F Prelz[1], S Aghion[1,2], C Amsler[3,4], T Ariga[3], G Bonomi[5,6], R S Brusa[7], M Caccia[1,8], R Caravita[9,10], F Castelli[1,11], G Cerchiari[12], D Comparat[13], G Consolati[1,2], A Demetrio[14], L Di Noto[9,10], M Doser[15], A Ereditato[3], C Evans[1,2], R Ferragut[1,2], J Fesel[15], A Fontana[6], S Gerber[15], M Giammarchi[1], A Gligorova[16], F Guatieri[7], S Haider[15], A Hinterberger[15], H Holmestad[17], A Kellerbauer[12], D Krasnický[9,10], V Lagomarsino[9,10], P Lansonneur[18], P Lebrun[18], C Malbrunot[4,15], S Mariazzi[4], V Matveev[19], Z Mazzotta[1,11], S R Müller[14], G Nebbia[20], P Nedelec[18], M Oberthaler[14], N Pacifico[16], D Pagano[5,6], L Penasa[7], V Petracek[21], M Prevedelli[22], L Ravelli[7], B Rienaecker[15], J Robert[13], O M Røhne[17], A Rotondi[6,23], M Sacerdoti[1,11], H Sandaker[17], R Santoro[1,8], P Scampoli[3,24], M Simon[4], L Smestad[15,25], F Sorrentino[9,10], G Testera[10], I C Tietje[15], E Widmann[4], P Yzombard[13], C Zimmer[12,14,15], J Zmeskal[4], N Zurlo[6,26]

[1] INFN Milano, via Celoria 16, 20133, Milano, Italy
[2] Politecnico of Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy
[3] Laboratory for High Energy Physics, Albert Einstein Center for Fundamental Physics, University of Bern, 3012 Bern, Switzerland
[4] Stefan Meyer Institute for Subatomic Physics, Austrian Academy of Sciences, Boltzmanngasse 3, 1090 Vienna, Austria
[5] Dept. of Mech. and Industrial Eng., Univ. of Brescia, via Branze 38, 25123 Brescia, Italy
[6] INFN Pavia, via Bassi 6, 27100 Pavia, Italy
[7] Department of Physics, University of Trento and TIFPA/INFN Trento, via Sommarive 14, 38123 Povo, Trento, Italy
[8] Department of Science, University of Insubria, Via Valleggio 11, 22100 Como, Italy
[9] Department of Physics, University of Genova, via Dodecaneso 33, 16146 Genova, Italy
[10] INFN Genova, via Dodecaneso 33, 16146 Genova, Italy
[11] Department of Physics, University of Milano, via Celoria 16, 20133 Milano, Italy
[12] Max Planck Institute for Nuclear Physics, Saupfercheckweg 1, 69117 Heidelberg, Germany
[13] Laboratoire Aimé Cotton, Université Paris-Sud, ENS Cachan, CNRS, Université Paris-Saclay, 91405 Orsay Cedex, France
[14] Kirchhoff-Institute for Physics, Im Neuenheimer Feld 227, 69120 Heidelberg, Germany
[15] Physics Department, CERN, 1211 Geneva 23, Switzerland
[16] Inst. of Physics and Technology, Univ. of Bergen, Allégaten 55, 5007 Bergen, Norway
[17] Department of Physics, University of Oslo, Sem Sælandsvei 24, 0371 Oslo, Norway
[18] Inst. of Nuclear Physics, CNRS/IN2p3, Univ. of Lyon 1, 69622 Villeurbanne, France
[19] Institute for Nuclear Research of the Russian Academy of Science, Moscow 117312, Russia and Joint Institute for Nuclear Research, 141980 Dubna, Russia
[20] INFN Padova, via Marzolo 8, 35131 Padova, Italy
[21] Czech Technical University, Prague, Břehovà 7, 11519 Prague 1, Czech Republic
[22] University of Bologna, Viale Berti Pichat 6/2, 40126 Bologna, Italy
[23] Department of Physics, Univ. of Pavia, via Bassi 6, 27100 Pavia, Italy
[24] Department of Physics "Ettore Pancini", University of Napoli Federico II, Complesso Universitario di Monte S. Angelo, 80126, Napoli, Italy
[25] The Research Council of Norway, P.O. Box 564, NO-1327 Lysaker, Norway

[26] Department of Civil Engineering, University of Brescia, via Branze 43, 25123 Brescia, Italy

E-mail: francesco.prelz@mi.infn.it

Abstract. In the sociology of small- to mid-sized (O(100) collaborators) experiments the issue of data collection and storage is sometimes felt as a residual problem for which well-established solutions are known. Still, the DAQ system can be one of the few forces that drive towards the integration of otherwise loosely coupled detector systems. As such it may be hard to complete with off-the-shelf components only.

LabVIEW and ROOT are the (only) two software systems that were assumed to be familiar enough to all collaborators of the AEgIS (AD6) experiment at CERN: working out of the GXML representation of LabVIEW Data types, a semantically equivalent representation as ROOT TTrees was developed for permanent storage and analysis. All data in the experiment is cast into this common format and can be produced and consumed on both systems and transferred over TCP and/or multicast over UDP for immediate sharing over the experiment LAN. We describe the setup that has been able to cater to all run data logging and long term monitoring needs of the AEgIS experiment so far.

1. Introduction

The AEgIS (AD6) experiment is set up at the Antiproton Decelerator (AD) Facility at CERN with the purpose of directly studying the free fall of antimatter in the Earth's gravitational field [1]. The physics process employed to produce and steer the excited antihydrogen atoms needed for the gravity measurement is briefly illustrated in Figure 1. One can see that the process calls for the interaction of various, diverse fields of expertise from various collaborating groups.

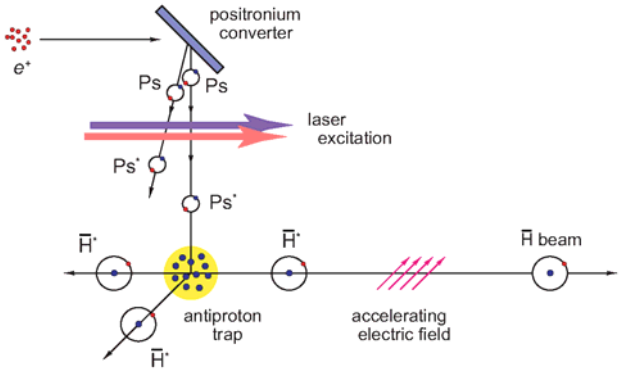

Figure 1. Process used in the AEgIS experiment for the creation of excited $\bar{H}$ atoms by resonant charge exchange with excited positronium [1]. The produced $\bar{H}$ is then accelerated towards a classical Moiré interferometer for direct gravity measurement [2]. Twenty groups bring to the collaboration the needed expertise in the specialised fields of laser technology, positronium production, atomic beam interferometry, plasma steering, etc.

Turning to the computing side of the experiment, the main challenge is not in the size or rate of data to be acquired[1], but rather in the lack of *common* technology familiar to the entire

[1] As detectors and other components were added to the experimental setup the raw data stored in each active 108-second beam cycle increased from an average of 4 MB compressed in 2012 (400k data atoms as defined in Section 2, or 3.7k atoms/s with average uncompressed payload size of 8.2 bytes) to an average of 29 MB compressed 2016 (1.3M data atoms, 12k atoms/s with average uncompressed data payload size of 71.0 bytes) in 2016. The compressed size of the full 2016 raw dataset is 534 GB.

collaboration: groups accustomed to running small test-stands tend to run various versions of LabVIEW™, saving data in local files of disparate formats; collaborators with data analysis experience at CERN tend to be trained in the use of the ROOT [3] analysis framework and the related forms of data object compressed storage; the integrated DAQ and data storage package most commonly used by the antimatter experiment community (and inherited by collaborators who are 'native' to the field) seems to be MIDAS [4]. The first effort was therefore spent in reducing the technology palette to a minimal set that would gain the confidence of most, if not all, collaborators.

We started by putting together a test stand based on MIDAS. We quickly realised that the most noticeable shortcoming of MIDAS was neither code obsolescence nor lack of maintenance or support, but rather the fact that the package comes as a tightly integrated system. Developing new components that fit and morph into the existing MIDAS interfaces is a complex task, but it is required in case part of the needed functionality (front-end module support, storage data[base] format, access tools) is not found in the integrated package.

We then focused on identifying and designing a small set of simple building blocks built *only* around ROOT and LabVIEW™, providing the needed data acquisition, logging and access functions. The prototype data object was chosen to be a generic LabVIEW `Cluster`, including a unique name and a common format timestamp. The long-term storage technology was chosen to be ROOT `TTree`s. In the coming sections we describe in more detail this "AEḡIS data atom", the chosen serialisation format for network transfer and the related software libraries.

2. The AEḡIS data atom and its serialisation

One data object whose expressiveness is well understood by many people in the collaboration is the LabVIEW™ `Cluster`. We therefore chose to limit the allowable numeric formats to what LabVIEW™ can use and to model structured data types around the LabVIEW™ `Array` and `Cluster` types[2]. Two additional requirements were imposed:

(i) that all data that circulate in the DAQ system be identified by a unique name (string) and include a common format timestamp (see Table 1 for details).

(ii) that the recursiveness in data `Cluster` definitions be limited to what can be unwound into human-readable ROOT `TTree` definitions. The final data format for long-term storage can be thought of as a ROOT `TTree` translation of a LabVIEW™ cluster. In short, we allow clusters to contain arrays of scalar values, but we do not allow arrays of clusters or clusters containing other clusters.

This leads to the definition of the AEḡIS data atom shown in Table 1. It represents all data objects handled by the DAQ for both apparatus monitoring and physics data.

The issue of serialising data atoms for network transfer was addressed by scouting various options available within the LabVIEW™ software library. We looked for a format with no proprietary data content in order to preserve the interoperability with non-LabVIEW™ components. A text format was perceived as acceptable in the foreseen data throughput scale, so we examined the default "Flatten to/Unflatten from XML"[3] functions, but passed on them as the produced XML code was too verbose. We then settled on the GXML [5] reference library, that provided a reasonable balance between expressiveness and verbosity. An example of the GXML representation of a data atom with a cluster data payload is found in Figure 2. The main advantage of relying on a reference library provided by National Instruments is a guarantee that any internal component needed to unwind and serialise ("flatten" in LabVIEW™ jargon) data

[2] A LabVIEW™ `Array` is a collection of data of identical type, while a `Cluster` is a collection of data of different types, similar to a C language `struct`.

[3] The LabVIEW™ JSON helpers were still to come at the time (circa 2011) when the AEḡIS DAQ was being designed.

Name	Alphanumeric String containing a (possibly hierarchical) unique name for the data atom. The format of the data associated with a given name **should** not be changed.
Timestamp	Instant when the data was acquired, in three formats: 1) character string, parsable by `strptime(3)`; 2) `struct timespec` containing time since the UNIX epoch; 3) 64-bit unsigned integer with RF clock count, if applicable.
Data	Instance of a scalar, vector or structured (cluster) data type, compatible with LabVIEW™ types, and their conversion into ROOT `TTree`s.

Table 1. Structure of the AEḡIS data atom, representing *all* DAQ data objects.

```
<GXML_Root>
     <Name type='String'>a_test_cluster</Name>
     <Timestamp mems='4'>
          <str type='String'>16:18:09.220036 09/20/2016</str>
          <tv_sec type='U64'>1474381089</tv_sec>
          <tv_nsec type='U32'>220036174</tv_nsec>
          <Clock type='U64'>7856432</Clock>
     </Timestamp>
     <Data mems='3'>
          <double_val type='DBL'>1.2344999999999999307</double_val>
          <int_val type='I32'>12345</int_val>
          <float_array dim='[3]' type='SGL'>
               <v>1.1</v><v>2.2</v><v>3.3</v>
          </float_array>
     </Data>
</GXML_Root>
```

Figure 2. Example of GXML serialisation of an AEḡIS data atom containing a cluster of two numeric scalar values and one numeric array.

objects will be maintained and upgraded. The NI-supplied GXML helpers were supplemented with a (C++) library to code, decode, transfer, translate GXML on other platforms. They were also integrated into higher level LabVIEW™ Virtual Instruments (VIs). Both developments are described in the following sections.

3. C++ library and tools

All DAQ components and processes that are *not* run by LabVIEW™ create data atoms as instances of the `LVTClusterData` C++ class. The class can include a scalar payload, but is also provided with container-like properties to represent arrays and clusters: it can therefore represent *any* LabVIEW™ data type. A glance at Figure 3 shows that:

- All C++ classes developed for handling GXML data *can* inherit from a ROOT `TObject`, so they can be constructed and used in the interactive ROOT shell.
- Data arrays can also be represented by a specialised `LVTArrayData` subclass. While the base `LVTClusterData` class has the ability to represent array data types, it stores array contents

as a vector of `LVTClusterData`. On the other hand, `LVTArrayData` uses the ordinary, contiguous memory area storage for arrays and can therefore be made to point to existing array or `std::vector` storage, avoiding unnecessary memory copies. For increased data transfer efficiency (see Section 7) `LVTArrayData` can be serialised and deserialised in a compressed text format, in addition to the standard GXML format.

.

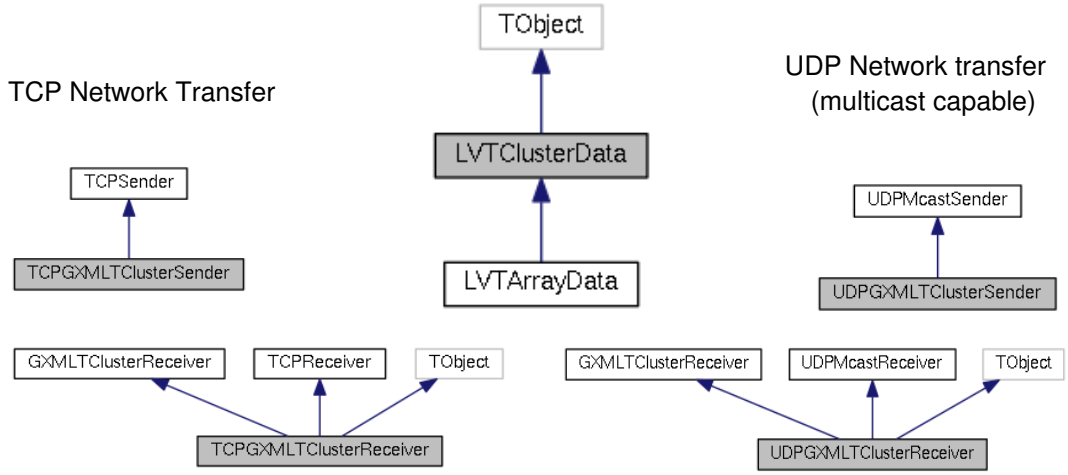

Figure 3. Inheritance graphs for the C++ support library allowing to code, decode, convert and transfer AEgIS DAQ data atoms. The `LVTCluster` class represents any object that can be expressed as a Cluster in LabVIEW™, and has the ability to serialise itself into GXML. IPv6-compatible classes to exchange data atoms via either TCP or UDP (including multicast) allow to assemble any needed data transport channel. All classes *can* (via a compile-time option) inherit from a ROOT `TObject` and be used from the interactive ROOT shell.

The remainder of the C++ library is used to transfer `LVTClusters` across the network. It includes base transmitter and receiver classes and their specialisation for TCP and UDP transfer. The UDP classes support multicast transfer. All network code is IPv6 compliant. While `LVTClusters` can serialise themselves in GXML format, the de-serialisation code is kept in the network receiver class to avoid bringing in unnecessary dependencies from the external XML library used for XML parsing (see Section 7 for details). The receiver class is also able to cast received data into a semantically equivalent ROOT `TTree` format for long-term storage and subsequent data analysis.

The main users of the C++ library are:

(i) The core data logging executable. It is run in multiple instances to handle run and monitoring data for various subsystems: GXML data received by threads (serving one TCP and/or UDP port each) is parsed and converted into ROOT `TTrees`. These are continuously saved to disk (via `TTree::AutoSave("saveself")`).

(ii) The executable that is run on a VME embedded processor for front-end data collection via VMEbus (more details in Section 5).

4. LabVIEW library

Turning to the LabVIEW™ side of the design, while the basic code to serialise and deserialise data in GXML format is provided and maintained by National Instruments [5], we needed to:

- Add integrated VIs for network data transfer (a quick review of these is shown in Figure 4). Data can also be retrieved from the experiment long-term storage by querying a dedicated web server over HTTP.
- Work around performance shortcomings of the 32-bit LabVIEW™ memory allocator by developing a few Windows DLLs and executables to accelerate data conversion. More details on this specific issue can be found in Section 7.

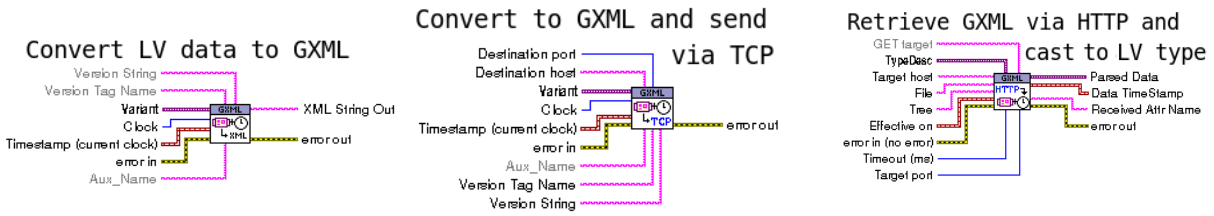

Figure 4. Fundamental LabVIEW™ Virtual Instruments for data atom exchange. They can be seen as equivalent to the C++ classes shown in Figure 3

One of the most noticeable practical advantages of building the DAQ design around the native LabVIEW™ data representation lies probably in the ease of sending data from any piece of code run by LabVIEW™ anywhere on the experiment network, including National Instrument embedded real-time processors running on the front-end. As long as data is tagged by a unique name, it can simply be connected to the appropriate VI and sent to the applicable TCP or UDP port for either run-data taking or long-term, continuous experiment monitoring. An example of how data can be sent to the DAQ is shown in Figure 5.

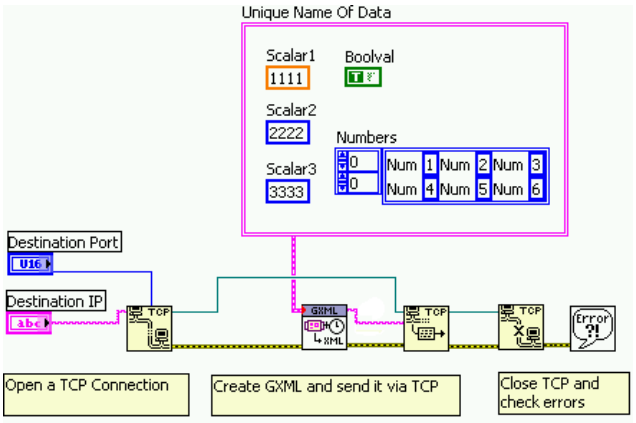

Figure 5. Sending a LabVIEW™ Cluster to the DAQ via TCP.

As the main run-control functions can also be driven from LabVIEW™, the palette of experiment-specific VIs was completed with run start/stop/status blocks (interacting with the main DAQ control node over the network).

5. DAQ front-end components

The complete data flow for the experiment, leveraging all components described so far, is shown in Figure 6. The system includes so far three categories of front-end equipment:

(i) Regular PCs running any operating system, sending data either via the LabVIEW™ or the C++ support libraries.

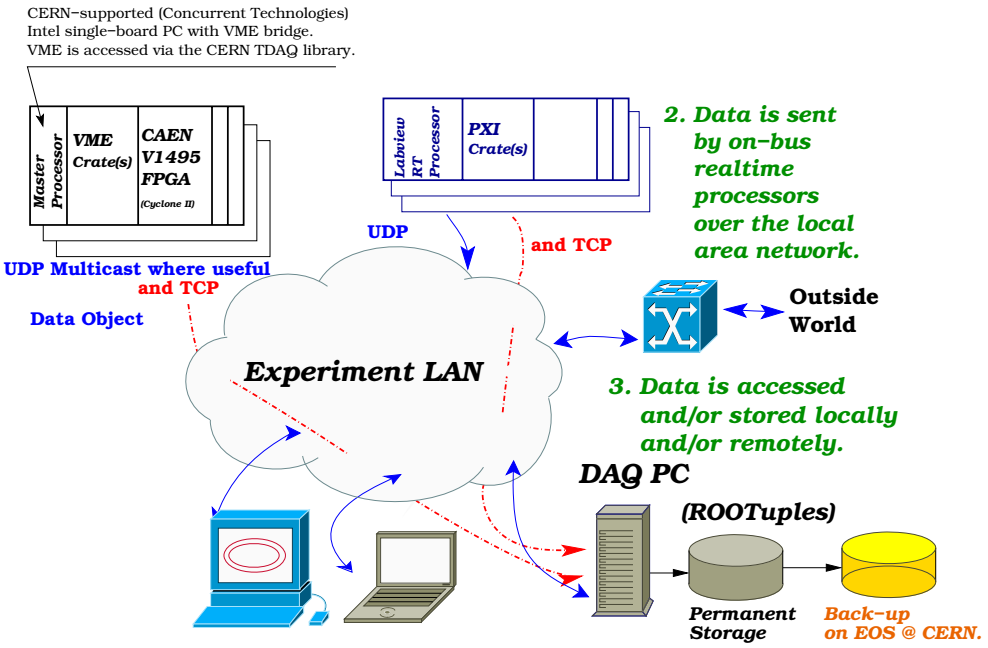

Figure 6. Interaction of front-end, data logging, storage and monitoring components on the experiment LAN.

(ii) CERN-supported VME PCs. After a few years on an EL Pool Concurrent Technologies VP110, we are currently running a VP717, also by Concurrent Technologies. Low-level code to drive respectively the Tundra Universe II or TSI148 VME bridge was inherited by the ATLAS TDAQ project at CERN [6], extracted from the ATLAS environment and adapted for AEḡIS. All data-taking chores can so far be handled by running one 3-thread process that gathers data on VMEbus and sends it out through one 1 Gbit/s Ethernet interface with the help of the C++ library described in Section 3.

(iii) Various Run-Time processors manufactured by National Instruments and running LabVIEW RT™ natively. All VIs developed for the experiment (and the reference GXML library) are compatible with LabVIEW RT™.

6. Web-based run control and data monitor

All data acquired by the AEḡIS DAQ system end up on disk in the form of ROOT files containing a number of `TTree`s, each representing a collection of data atoms unwound into ROOT native data types (object storage is used for `std::vector<std::string>`s *only*). This includes calibration data, physics run data as well as data from continuous apparatus monitoring. Alongside analysis code accessing directly the ROOT files, data are also served through an HTTP server. A custom back-end serves queries for specific data series by properly accessing and decoding stored ROOT files. This service is accessed by three classes of clients:

(i) LabVIEW™ code, where a VI is provided to retrieve and reconstitute data in the original data cluster format.

(ii) A Javascript data browser with simple stripchart display capabilities. Generated graphs can be annotated and sent directly to the experiment ELOG [7] server.

(iii) DAQ front-end C/C++ code requiring access to the calibration data.

Long-term preservation of the acquired data is granted by staging files into the EOS [8] and CASTOR [9] systems at CERN. The current production of about half a TB of compressed data

per data-taking year fits in the minimum experiment quotas on those systems.

7. Lessons learned and conclusions

While the system described so far has been running and acquiring data continuously over the experiment lifetime (interrupted only by the occasional power outage), a few operational issues required adjustments to the implementation. We recollect them here for future memory:

- While the need to transfer large ($O(10^6)$ numeric members) arrays from LabVIEW™ to the DAQ did not emerge clearly at the time of system requirement collection, it imposed itself as a need at data-taking time. This required two corrections:
  (i) Most of the LabVIEW™ installations in the experiment are running the 32-bit package version. This includes a custom memory allocator that displays poor performance when copying arrays. As an update of LabVIEW™ was ruled out for fear of regression issues, alternative ways to funnel array data to the DAQ had to be explored. In addition to the native GXML code, both a custom Windows DLL (C-language source, to be invoked within a "Call Library Function" node) and a standalone executable were developed. The latter was needed because *the* most effective technique to avoid bottlenecks in LabVIEW™ was found to be dumping data on disk in "LabVIEW SpreadSheet" format and invoking an external executable to convert and transmit them.
  (ii) For these very large arrays, the default GXML representation of data is too verbose and data bloating is excessive. So an additional transfer format (i.e. compressed, base64-encoding of the array contents) was added into the system to handle large array data only. This led to the addition of a specialised class to model arrays in the C++ library (see Section 3).
- In order to limit the set of code dependencies, we initially based the (G)XML parsing code on the `Root::TDOMParser` class packaged with ROOT. To achieve acceptable performance under real data-taking conditions we had to replace this with RapidXML [10].

Applying these corrections, the DAQ system ran continuously for apparatus monitoring and for all data-taking campaigns so far. We were happy to have settled on one common format for all data, for having bridged cultural gaps by settling on a minimal set of well understood technologies (LabVIEW™ and ROOT) and for the flexibility achieved both in network transport arrangement and in data schema update/extension. These needs were addressed satisfactorily by the design we exposed.

References

[1] Doser M *et al.* [AEgIS Collaboration], 2012 *Class. Quant. Grav.* "Exploring the WEP with a pulsed cold beam of antihydrogen," **29** 184009. doi:10.1088/0264-9381/29/18/184009
[2] Aghion S *et al.* [AEgIS Collaboration], 2014 *Nature Commun.* "A moiré deflectometer for antimatter," **5** 4538. doi:10.1038/ncomms5538
[3] Brun R and Rademakers F, 1997 *Nucl. Inst. & Meth.* "ROOT - An Object Oriented Data Analysis Framework," **A 389** 81-86. doi:10.1016/S0168-9002(97)00048-X. See also `http://root.cern.ch/`.
[4] The main web page for MIDAS system can be accessed from `http://midas.triumf.ca/`. We were unable to locate a published overview paper on the system.
[5] The LabVIEW™"Reference Library for Converting Between LabVIEW and XML Data (GXML)" is documented and available for download at `http://www.ni.com/example/31330/`.
[6] Resources for the ATLAS TDAQ project can be currently located at: `http://atlas.web.cern.ch/Atlas/GROUPS/DAQTRIG/Newdaqtrig.php`.
[7] The home page for Elog is at `http://midas.psi.ch/elog/`.
[8] Peters A J, Sindrilaru E A, Adde G, 2015 *J. Phys.: Conf. Series* "EOS as the present and future solution for data storage at CERN," **664** 042042. doi:10.1088/1742-6596/664/4/042042
[9] Lo Presti G, Barring O, Earl A, 2007 *24th IEEE MSST Conference* "CASTOR: A Distributed Storage Resource Facility for High Performance Data Processing at CERN," doi:10.1109/MSST.2007.4367985
[10] The home page for RapidXML is at `http://rapidxml.sourceforge.net/`.