# Advanced Visualization System for Monitoring the ATLAS TDAQ Network in real-time

Batraneanu Silvia, Campora Daniel, Martin Brian, Savu Dan, Stancu Stefan and Leahu Lucian

*Abstract* – The trigger and data acquisition (TDAQ) system of the ATLAS experiment at CERN comprises approximately 2500 servers interconnected by three separate Ethernet networks, totaling 250 switches. Due to its real-time nature, there are additional requirements in comparison to conventional networks in terms of speed and performance. A comprehensive monitoring framework has been developed for expert use. However, non experts may experience difficulties in using it and interpreting data. Moreover, specific performance issues, such as single component saturation or unbalanced workload, need to be spotted with ease, in real-time, and understood in the context of the full system view. We addressed these issues by developing an innovative visualization system where the users benefit from the advantages of 3D graphics to visualize the large monitoring parameter space associated with our system. This has been done by developing a hierarchical model of the complete system onto which we overlaid geographical, logical and real-time monitoring information.

This article briefly describes the network monitoring framework and introduces the system's visualization challenges and previous limitations. The functionality, design and implementation of the 3D visualization system are then described in detail, with a focus on the model design, user interaction, navigation mechanisms and methods used to achieve scalability when rendering and updating many objects in real-time.

ATL-DAQ-PROC-2012-013
12 June 2012

## I. INTRODUCTION

The architecture of the ATLAS TDAQ networks has already been described in [1] and the operational model in [2]. There are three distinct networks, all based on the Ethernet technology: a *control network* and *two dedicated data networks.* Each network has two chassis based devices at the core and approximately 100 aggregation switches. Network traffic and device statistics are gathered in real time, at a 30 seconds interval, to assess the performance of the network, to assist in system analysis and to provide reliable status information to the experiment's run control. Gathering relevant statistics and system status information is only part of a much larger problem: making it available to different types of users and deciding in what form to present it.

Manuscript received April 28, 2012. (Write the date on which you submitted your paper for review.)

S. M. Batraneanu is with the University of California, Irvine, USA (e-mail: silviab@cern.ch).

D. H. Campora Perez, is with the University of Seville, Seville, Spain (e-mail: dcampora@cern.ch).

B. Martin is with CERN, Geneva, Switzerland (e-mail: bmartin@cern.ch)

D. O. Savu is with CERN, Geneva, Switzerland (e-mail:dsavu@cern.ch)

S. N. Stancu is with CERN, Geneva, Switzerland (e-mail:sstancu@cern.ch)

L. Leahu is with University "Politehnica" Bucharest, Romania (e-mail: lleahu@cern.ch)

## II. SPECIFIC VISUALIZATION REQUIREMENTS

Unlike general purpose networks, the ATLAS TDAQ system has very demanding requirements in terms of speed and performance. For example, some areas of the network have to operate at very high rates and even a low rate of discarded packets might cause unacceptable application delays. Both health and performance monitoring tools are needed to gather and analyze historical and real-time statistics for the purpose of insuring proper operation and performing post-mortem troubleshooting. In order to have a complete image of the overall system status, the information cannot be restricted to network-specific statistics, so other data categories are collected, such as environmental statistics, physical and logical grouping information, etc.

The existing centralized monitoring framework offers access to different data sources, both internal and external, and also provides front-end Web applications which allow the intuitive display and analysis of historical data. However, no single frontend application allows the intuitive display of all needed real time and historical monitoring information for all user categories. There are at least three different consumer groups, each with their own visualization requirements:

- **Networking experts** need to analyze, monitor and troubleshoot the network. They need to have access to detailed knowledge about which ports or devices are reporting chronic or sporadic failures in order to perform maintenance operations,
- **System analysts** need to understand how subsystems interact and to have a global view of the system, its traffic overloads, communication flows and lost packets. They use this information to determine if the TDAQ system is well balanced and to tune it for optimal performance,
- **Operators** need to have a less detailed view over the system that allows them to identify if all or most of the system is working within the design limits. This way they can adjust the event rate to accommodate to the current situation.

Taking all these into account, an effective network visualization system for the ATLAS TDAQ system needs to:

- display the network in an intuitive way by using a logical grouping that follows the architecture of the system and the flow of data inside it,
- display both traffic and status information in real time,

- offer top-level, intermediate and detailed views of the network,
- provide fast navigation between views to avoid missing the big picture while searching for detailed information.

With such a network to monitor, the developers face several important challenges. First, the system's scale brings a technical visualization problem which is augmented by the overlapping of data and control networks. Although some of the requirements have been addressed by two bi-dimensional real-time monitoring displays, there are still limitations with respect to the specific visualization requirements of the TDAQ system. Logical grouping is important in understanding how subsystems interact and this prevented us from using automatic placement algorithms to ameliorate visual clutter inside the existing monitoring displays.

Secondly, every component is characterized by several descriptive parameters and monitoring variables leading to a large variable space that needs to be visualized. Aggregation and relevant error propagation need to be used in order to help the user interpret the data with ease. Last but not least, many of these variables need to be updated in real time (30 seconds) and, to achieve this, efficient data structures and update mechanisms need to be implemented. The rendering of the whole 3D scene, including the dynamic objects, needs to be done at a frame rate of over 30fps, frame rate considered to be the lower limit for real-time rendering and which avoids image flickering.

Although 2D visualization has its advantages, there are good reasons to use 3D visualization, especially for large scale and complex models. The most obvious advantage is the additional dimension to encode information, which also creates a large visible workspace for holding visualizations.

Also, in 3D we can represent overlapping networks without causing visual clutter. Intuitive navigation paradigms [3 ] such as "walk" or "fly" can be used to navigate through our network model with ease from the top level view to the most detailed views. Another important benefit is the evaluation of the camera-object distance which can trigger different behaviours such as the levels of detail [4] or object update. Taking into account these benefits, we opted for a 3D visual structure to represent our system.

From a technical perspective, 3D implementations bring greater challenges because they require more processing power, involve many more parameters and six degrees of freedom for movement.

## III. MONITORING SOFTWARE PLATFORM

A comprehensive framework, depicted in Figure 1, has been deployed for monitoring the TDAQ networks health and performance. It contains a mixture of commercial software and in-house developed tools.

CA Spectrum [5] has been chosen for health monitoring. It maintains a model of the full network and polls the network devices and connections for their status every 60 seconds via the Simple Network Management Protocol (SNMP). Its most useful feature in our case is the alarm management mechanism: it generates alarms based on both synchronous and asynchronous events (SNMP traps) and forwards them to different types of clients.

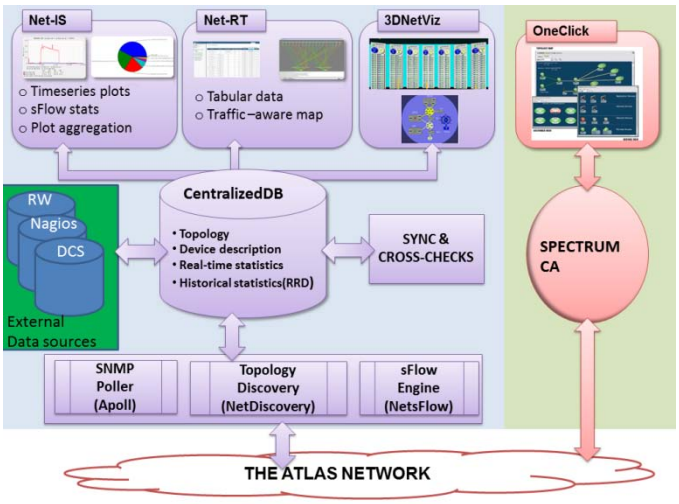


Fig. 1. Monitoring software framework for the ATLAS TDAQ networks

Performance monitoring is achieved using a set of in-house applications. In order to provide a complete performance monitoring solution, both back-end and front-end applications have been developed. Some of them have been developed in order to overcome the scaling issues of the aforementioned commercial package when performing high rate network statistics gathering and display. Others have been developed to access additional data sets which are relevant for troubleshooting.

The backend applications which are used to gather dynamic and static network-related information are:

- *APoll* [6]- a high-performance SNMP polling engine that efficiently gathers statistics from all the network ports and store them as RRD files, with a polling interval of 30 seconds or less.

- *NetDiscovery* - a network topology discovery engine which takes into account the particularities of the TDAQ network, such as aggregated links and physical links shared by VLANs, to create an up-to-date network map.

- *sFlow engine* [7] - an sFlow based flow collector and analysis engine, which stores statistically sampled data about flows. This is extremely useful for a post-mortem or detailed troubleshooting.

At the core of the monitoring framework lies a Central DataBase (CDB) which is used to store data and metadata from various sources and provide it on request to several

front-end clients. The data sources are internal, i.e. gathered by above-mentioned backend applications, or external, which are gathered and stored by applications managed by other groups but which are of major interest when performing network monitoring and troubleshooting. Several script-based tools connect to external and internal data sources, retrieve the latest updates and perform consistency checks. In this way, the CDB repository is kept consistent and up-to-date.

Historical time series statistics are stored in RRD [8] files and are split into internal repositories, network device and traffic statistics, and external repositories, containing various environmental and PC specific statistics. A set of MySQL databases are used to store both *static information*, such as topology and descriptive device information, and *dynamic data*, such as a current snapshot of important variables (i.e. SNMP counters) for devices and interfaces.

Several front-end applications have been designed to visualize the gathered data. Some focus on historical data (Net-IS), while others on real-time data (Net- RT). A complete solution for viewing all sorts of historical data is Net-IS (Integrated System for Performance Monitoring) [9]. It has a powerful interface which provides convenient access to all historical data that is stored or referenced by the CDB. The information can either be browsed with the use of several predefined hierarchical trees, or it can be directly searched using regular expressions. Several time series values can be displayed at the same time both overlaid on the same plot, or as an array of mini-plots.

A different set of presentation clients is grouped under the name of Net-RT. In this category, real-time data regarding the component status and traffic values are provided both in tabular format and using a traffic-aware network map.

The main benefit of the current architecture is that all information categories, static and dynamic, are available to all visualization and analysis tools. Based on the stored logical and physical grouping information, statistics can be aggregated. As depicted in Figure 1, the 3D visualization is integrated into the framework. Its main role is to monitor in real-time both the health and the performance of the TDAQ networks.

## IV. TRIDIMENSIONAL MODEL

A hierarchical 3D model has been built using four layers of abstraction, as shown in Figure 2:

- *subsystem layer* - offers the overall picture of the network at a glance. It contains two types of containers: processor farms and network cores,
- *rack layer* - contains device groups formed using geographic proximity and/or function,
- *device layer* - contains processors and aggregation switches,
- *network interface layer* - offers information about network interfaces and their associated traffic throughput and errors, displayed in bi-dimensional panels.

Guidelines taken from the information visualization domain were used to map object attributes into graphical properties. At every level, individual and aggregated status information is color-coded based on predetermined thresholds which vary for each subsystem.

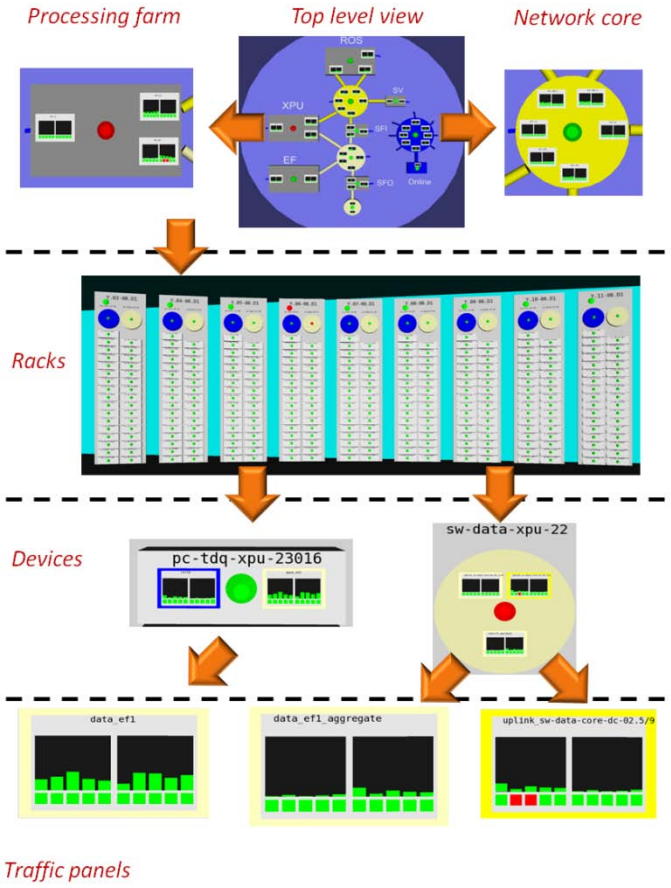


Fig. 2. Hierarchical 3D model of the TDAQ networks with four layers: network interfaces, devices, racks and subsystems

Three color levels are used consistently throughout the model whenever component status is represented: green - normal state; orange - warning state and red - critical state. This representation is frequently used in monitoring displays and hence the users are very familiar with it.

We equally used a consistent color schema at every level to display network affiliation: blue is used to represent the control network while yellow is used to represent the data network. Visual representations of components belonging to a particular network (network cores, aggregated connections, aggregation switches, network interfaces) are all colored using one of these colors. Components which are connected to both networks, such as computers, are colored using neutral colors such as white or gray. Color saturation is used to differentiate between the two data networks.

In the case of a network interface representation, since 2D areas are used, we decided to use borders for network

affiliation and the panel background for operational status. Shape is used to distinguish between component types: network devices (core and aggregation switches) are represented using cylinders while processor and processing farms are represented using boxes.

## V. System Implementation

### A. System and client architecture

The advanced visualization system is integrated in the monitoring platform described in Section II and is characterized by a three-layer architecture depicted in Figure 3. On the data layer lays the CDB repository containing all the metadata and the statistics of the underlying system. On the intermediate layer, a Web server is used to deliver, static and real-time content to client applications. Application logic, such as statistics aggregation and error propagation, is done at this middle layer and is embedded in a set of Python [10] scripts. On the presentation layer, a specialized 3D engine is used to display and interact with the 3D model.

The components for the first two layers run on servers placed inside the ATLAS Control Network (ATCN), which is a protected area, separated from the main General Purpose Network (GPN) of CERN. The front-end application is run both inside ATCN, for operators, and outside it, for remote users. Due to security reasons, a set of gateway machines, acting as an upper layer firewall, allow only certain HTTP conversations to be established based on known source and destination signatures.

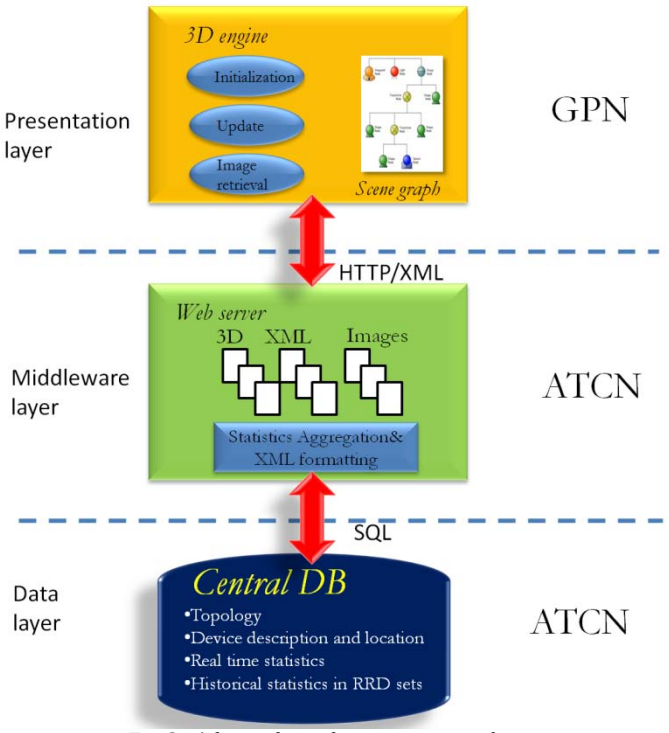


Fig. 3. Advanced visualization system architecture

Besides having to run the communication on top of the HTTP protocol, a data format that can impose a certain structure needed to be chosen. The XML format responded to both requirements and was used as the main technology for storing and transmitting data between the Web server and the client applications. Moreover, because XML is interoperable, it allows the communication between modules using different programming languages such as Python on the server side and C++ on the client side.

At the intermediate layer, Python scripts aggregate statistics, format and store them in XML files which are then served to the client applications. These files are updated at regular intervals: the static description of the system every 30 minutes and the statistics every 30 seconds. A set of multimedia files containing the top level 3D model or historical plots are also made available on request.

For the presentation layer, a 3D engine was built from scratch using the OpenSceneGraph (OSG) [11] platform to insure the full control over the creation, rendering and update of the 3D model and also over the user interaction and navigation mechanisms.

OSG is an open source, portable and high performance 3D graphics framework written in C++ and OpenGL [12]. The framework has rigorous data structures based on the Standard Template Libraries (STL) [13] and implements well-known design patterns [14], such as *visitor* and *callback*. It provides a thin wrapper on top of OpenGL calls which gives access to many of its rendering options and a statistics display, essential for performance tuning. Node selection based on bit masks and specialized event handlers can be used to implement powerful interaction mechanisms.

Figure 4 presents the internal architecture of the OSG-based 3D engine, which is designed using a 3 layer approach. The bottom layer of the 3D engine is in charge of retrieving and storing its XML and multimedia content.

The multimedia content retrieval has specific characteristics which didn't allow us to use the default modules provided by OSG. First of all, like in the case of XML content, the multimedia files should be obtained using secure HTTP connections and for this the underlying cURL [15] library was used. Secondly, to obtain the plot images, a series of HTTP GET parameters need to be sent to the server and hence the URL format became very specific. Also, in the majority of situations, more than one plot needs to be retrieved and displayed at the same time, so it is preferable to use a single connection for fetching multiple files (multi-download) in order to save time. The MediaIO module has been built to overcome these limitations and, in addition, to offer centralized access to all needed types of multimedia files.

The intermediate layer manages the scene graph and the construction of additional information on demand. The scene graph management module uses the information made available by the modules from the bottom layer to construct and update the scene. A scene manager class initiates the

creation process by iterating on the subsystem container list and generating the associated object hierarchy. This way, the static top level 3D model is populated progressively. At every hierarchical level, a dedicated manager class is used to create a set of objects of the same type.

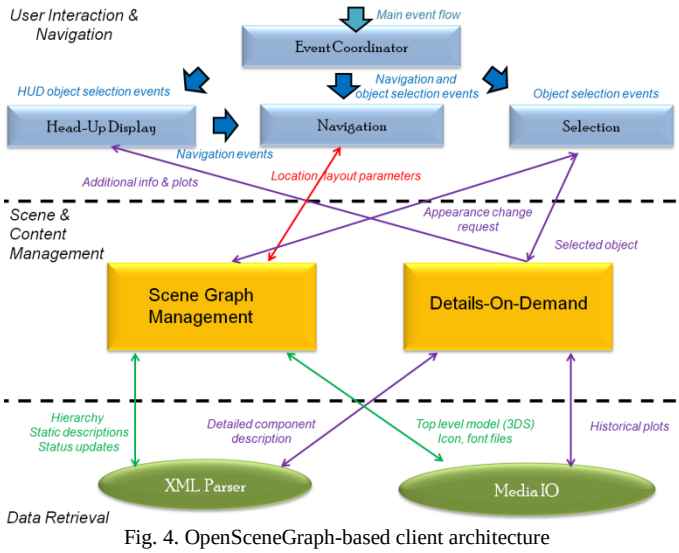


Fig. 4. OpenSceneGraph-based client architecture

Each of these classes contains a subclass dedicated to updating the dynamic components of the particular type, which is called *update callback functor* and is made available by OSG's traversal and callback mechanisms. Also, in order to update the visible objects only, a unique *cull callback factor* class is used in conjunction with the update callbacks. The second module (Details-On-Demand) uses the name of a selected object to retrieve its detailed description and associated plots.

The top level manages user interaction and navigation and contains four modules. The first one handles all navigation aspects, the second one handles object selection and highlight and the third one is dedicated to the Head-Up Display (HUD), containing navigation controls and additional information on demand. Since multiple event handlers are used to obtain different interaction behaviours, a fourth module plays the role of an event coordinator which enables/disables some behaviours at certain moments in time to avoid undesirable effects.

The selection module can trigger the display of additional information inside the HUD. These details are obtained and formatted by the intermediate level module Details-On-Demand. The module dedicated to the HUD can also trigger navigation to certain key locations in the scene, such as the top-level viewpoint or initial container viewpoints. In order to navigate to the close proximity of a selected object, the navigation module needs to obtain the object's specific position and dimensions from the scene graph management module.

### *B. Rendering and scene graph optimizations*

The goal was to create a true real-time rendering application which offers a frame rate of minimum 30 frames per seconds (fps). The OSG rendering statistics display allows precise rendering measurements and hence to evaluate the impact of each optimization on traversal times and on the overall frame rate.

Inside the client application, dedicated threads are used for XML and multimedia file transfers and, by doing so, the rendering performance is decoupled from the performance of these transfers. In any case, the transfer duration is less than one second and hence it is not affecting the 30 seconds update interval.

In terms of frame completion time, a limit of 30 fps translates into a maximum of 33.3 milliseconds (ms) in which a frame can be rendered. In order to render a frame, OSG performs at least three scene graph traversals: the UPDATE traversal flags the change made to the scene graph, the cull traversal keeps the visible nodes and the draw traversal sends the corresponding OpenGL calls to the GPU. More details about the scene graph traversals used in OSG can be found in [16].

The frame completion time represents the sum of all these traversal times. In this first phase of scene creation, we are mainly interested in the static behaviour of the scene, more precisely when real time updates are not performed, and we monitored closely the cull and draw traversal times. We had to decrease as much as possible these times and, according to these results, we adjusted the ranges used for the Level-Of-Detail nodes. Levels of details [4] represent a frequently used rendering acceleration technique which consists in hiding undistinguishable details when the camera is situated at certain distances from a given node.

We used the scene graph construction to perform a series of optimizations and structural changes. Tests were performed on three platforms running different operating systems (Windows XP, Windows 7 and Scientific Linux for CERN 5) , containing Intel Dual Core CPUs ranging from 1.86 to 3.4 GHz and three types of nVIDIA cards (Quadro FX 1400, GeForce GT 540M, GeForce 9800 GT) and the results were similar. In conclusion, in the case of our application, the behaviour was not influenced by the underlying platform.

OpenGL offers different vertex and primitive data definition and storage options. We evaluated these options in the context of our application to see which method offers the best performance for the different object types. For rendering geometry, the best solution was to use vertex arrays in combination with triangle primitives and color binding per vertex. It provided a 14% decrease in the draw time in comparison to the default. Also, some built-in OSG geometry classes such as osg::Box have been replaced by custom nodes optimized for fast rendering, leading to a decrease of 15% in the draw time.

Text rendering is very expensive and, for this reason, we disabled some built-in features and obtained a draw time 4 times smaller than the initial one. Also, we had to carefully choose how to display text and in what conditions, so we chose to temporarily augment the selected object or to display it in a different window on demand. Specialized OSG classes, such as Optimizer and Simplifier, were used to optimize complex objects and to reduce their sample ratio, hence creating simplified versions which can be used in levels of detail.

Another way of improving rendering time is by restructuring the scene graph. In our case, this implied the elimination, where possible, of Transform nodes and embedding the object coordinates into the geometry description. By eliminating a Transform node at the panel level we obtained a cull time 3 times smaller than the initial one. In order to have a maximum of flexibility when using levels of detail, the positioning of the LOD (Level-Of-Detail) nodes needed to be chosen carefully.

During the scene creation process, we performed multiple evaluations of the static scene. After building the panel geometry, we evaluated the rendering time and the frame rate for an increasing number of panel instances. Some general conclusions were drawn. First of all, the rendering times provide a linear increase with the number of instances and the frame rate is inversely proportional with the number of instances. Second of all, the cull time represents 10% of the draw time, so optimizations regarding this time are not very important.

Regarding the number of panels that can be rendered and still remain above the 30fps limit, this was situated at around 3000. After the creation of all other objects, the rendering time was measured again. The frame rate dropped under 30 fps when rendering more than 15 fully detailed racks or 40 simplified racks, number corresponding to the maximum number of racks needed to be displayed inside a container.
Based on these measurements, the LOD ranges have been defined so as to maintain a frame rate of over 30fps in all conditions.

### C. *Targeted update mechanism*

The construction of the scene graph is expensive and is often done as a pre-process. The preferred approach was to create the majority of objects at initialization time and to update them in real-time during scene rendering. The synchronous update of all objects in the scene strongly interferes with the rendering. We found methods to perform only partial updates and to insure a minimum frame rate that will not affect the user's navigation experience.

OSG enables update based on proximity by introducing the option of only traversing the nodes which are rendered by their parent LOD nodes but, by default, it updates all active nodes regardless of their presence in the camera's view frustum. A more efficient method, depicted in Figure 5, is to use the cull traversal in addition to the update traversal to update only the nodes which are inside the view frustum *and* rendered by their parent LOD node. More precisely, the use of the cull visitor guarantees that only the objects that are visible (not culled by any scene graph culling mechanism) are updated.

As in any scene-graph-based API, in OSG the cull traversal is always performed after the update traversal and hence the list of visible objects cannot be modified during the rendering of a single frame. The targeted update mechanism exploits temporal coherence in rendering, implying that adjacent frames are usually very similar, and updates the nodes which were visible in the previous frame.

For flagging the visible nodes bit masks were used. A traversal mask has been set for the update visitor and after each update traversal every node mask is set so as to exclude it from the next update traversal. In the cull traversal, every visible node has its mask set to include it into the next update traversal.

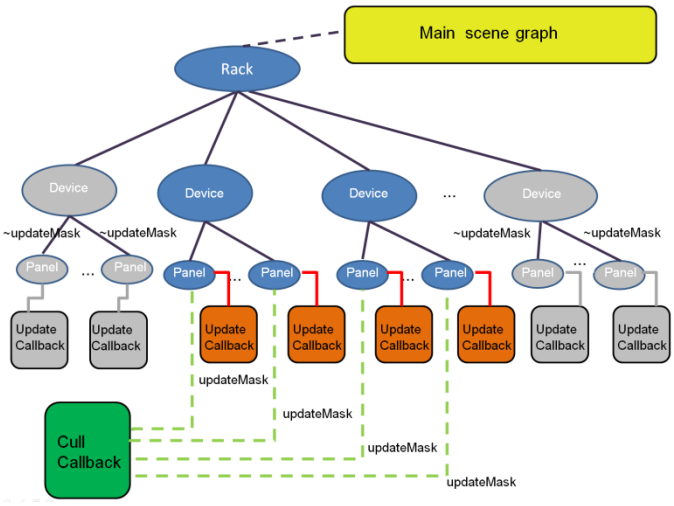

Fig. 5 – Targeted update mechanism

The real-time update is done with the help of a dedicated update callback functor class. An instance of this class can be attached either to a leaf (or geometry) node with the purpose of updating the contained geometry or to a grouping node (Group, LOD, Transform) with the purpose of updating the geometry contained by its children nodes and thus updating more objects in the same callback.

Using a callback of each object is not a viable option because the frame rate was dropping under 30 fps. Using callbacks at the device level led to an increase of 26% in the frame rate while at the rack level led to a 38% increase. The downside of this grouping was an increase of the maximum frame completion time which has been decreased by spreading the updates over multiple frames.

### D. *Interaction and navigation*

In the case of our system, the interaction and navigation needed to be adapted to the specifics of 3D graphics. The navigation was carefully planned to provide the user with intuitive movements and to fulfill one of the main requirements: navigating with ease from the top level view to the most detailed and back. Viewport controls, such as panning and zooming, were used in combination with guided navigation movements. We also aimed at a very intuitive interaction mechanism and managed to obtain the most detailed information in only a few clicks.

3D navigation implies 6 degrees of freedom in comparison to only 2 degrees provided by the 2D navigation. For this reason, in a 3D world, free navigation without any restrictions is difficult especially for users which are not familiar with a 3D display. It constitutes established practice that in a 3D world the users are guided between two locations of interest and, when using viewport controls (panning, zooming, etc), their movements to be restricted to only the ones they really need [17].

We used navigation paradigms such as “fly” and “walk” to enable a more intuitive navigation. Inside the subsystem containers resembling rooms, we are using the “walk” paradigm. Outside the containers, the “fly” paradigm is used. Object layout parameters and dimensions were used to restrict navigation and to calculate locations of interest (ex: rack view, device view). We used guided movements between hierarchical levels and a combination of free and guided movements on the same hierarchical level. The guided navigation between hierarchical layers is done by drilling down via mouse selection events and returning to higher levels by selecting dedicated buttons on the Head-Up Display.

For bringing up additional information regarding a particular device, we used the details-on-demand technique [18] in two mechanisms. The first was to add additional objects such as additional text labels directly on the object upon its selection. Due to the limited space, adding too many elements creates the risk of visual clutter and hence we needed to display additional information in a separate window.

The main problem with 2D windows placed inside a 3D display is that they are not intuitive to display when the camera changes position or orientation. In 3D graphics, a common practice is to use a Head-Up Display (HUD) to visualize information of interest or to provide interaction controls. A HUD enables the usage of a 2D interface which is guaranteed to always be drawn on top of the 3D model, and hence the positioning and orientation problems are solved.

Besides providing *context-aware navigation* (depending on hierarchical level, position, object type, etc), we benefited from having full control of camera movements in order to introduce powerful enhancements such as radial navigation inside the farm containers, smooth interpolation for guided camera movements, speed and field of view control to accommodate panoramic and detailed views.

The viewpoint-based navigation and details-on-demand implementation rely on object selection. Different behaviours are triggered when single or double clicking on an object and the camera moves to different viewpoints based on these events.

The selection mechanism involves the use of a specialized Visitor node (IntersectionVisitor) that traverses the scene graph in the search for geometrical objects meeting different intersection criteria. Node masks are used to specify which nodes are selectable at a certain moment and to trigger different behaviours based on object type.

Figure 6 presents the overall interaction schema for the top to the bottom of the model hierarchy. When the camera is situated in the top level viewpoint, double clicking on a container will trigger navigation to the top level viewpoint of the selected container. By doing so, the user is able to see the top-level traffic panels and monitor the container’s overall state. When double-clicking on a container, the user will enter the container and arrive at an initial viewpoint from where all the racks are visible and any problematic racks may be discerned.

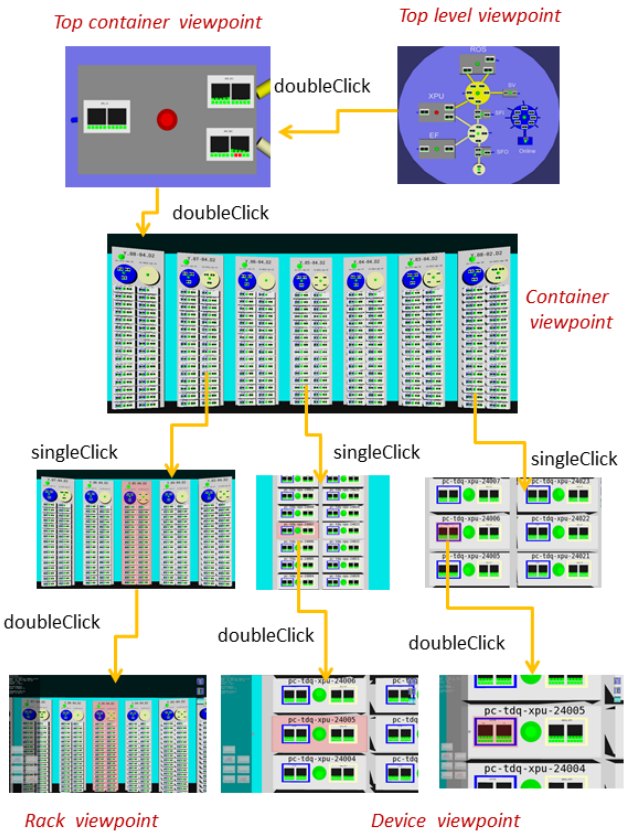


Fig. 6 – Overall interaction schema

From any location inside a container, the user is able to highlight either a whole rack, by clicking on the rack frame or a device, or an interface by clicking on the corresponding object. Double clicking on one of these objects will trigger navigation to a viewpoint from which the best view on the selected object is provided. Going further, additional information and mini-plots regarding the object appear in a dedicated HUD window.

## VI. Conclusions

An in-depth study concerning the TDAQ system's visualization requirements has been performed. The system's large scale represented one of the most important visualization problems that we've faced, augmented by the fact that the networks overlap. Moreover, our monitoring tools gather a big volume of data and operate at a fast rate. These statistics need to be visualized in real-time and the display has to be updated very frequently. One of the main novelties regarding our approach is that it uses 3D visualization and guidelines from the information visualization domain to build an advanced visualization system which accommodates these requirements.

We used the OSG framework to build a dedicated 3D engine for the display, interaction and real-time update of our system model. Scene creation and assembly have been automated. From the rendering performance point of view, we achieved the goal of rendering at a frame rate of 30fps or better. For this purpose, we performed optimizations at every level: geometry, text rendering, scene graph restructuring and, depending on the results, we used and adjusted Level-Of-Detail nodes to obtain the wanted frame rate and created a targeted update mechanism based on visibility and proximity to minimize the impact of the real time updates of the overall rendering time.

We took advantage from having full control on camera movements to implement specialized mechanisms allowing intuitive and context aware navigation. Also, features such as selection bit masks, intersection visitors and event handlers allowed us to provide efficient user interaction via an intuitive selection and highlight mechanism. Additional information and statistics plots, related to a selected node, are displayed with the help of a Head-Up Display.

## VII. Future Work

The data taking process is characterized by a multitude of parameters such as trigger rates, event size, luminosity blocks, etc. This information needs to be integrated into the visualization system so as to help place the computing-related events into the context of the current data taking run.

A rule-based engine has been recently developed to collect and aggregate logs and alarms. It can be integrated with the visualization system to offer precise information about a certain problem and to perform more efficient error propagation inside the visualization system.

OSG offers the possibility of displaying multiple views in the same time. This facility can also be used to display multiple views of different containers which can be used to help finding correlations between behaviours of different elements.

## References


[1] S. Stancu M. Ciobotaru L. Leahu B. Martin and C. Meirosu. "Networks for ATLAS trigger and data acquisition". Proceedings of Computing in High Energy Physics (CHEP 06), Mumbai, India, February 2006.

[2] S.M. Batraneanu A. Al-Shabibi M. Ciobotaru M. Ivanovici L. Leahu B. Martin and S. Stancu. "Operational Model of the ATLAS TDAQ Network". Proceedings IEEE Real Time 2007 Conference, May 2007.

[3] 3D Navigation behaviours.http://web3d.org/x3d/wiki/index.php/General_X3D_Navigation_Behaviors

[4] Levels of detail technique : http://en.wikipedia.org/wiki/Level_of_detail

[5] CA SPECTRUM Infrastructure Manager. http:// www.ca.com/us/root-cause-analysis.aspx

[6] D. Savu A. Al-Shabibi S. Batraneanu B. Martin R. Sjoen S. Stancu. "Integrated System for Performance Monitoring of ATLAS TDAQ Network". Proceedings of CHEP 2010, Taipei, Taiwan, 2010

[7] R. Sjoen S. Stancu M. Ciobotaru S. Batraneanu L. Leahu B. Martin and A.Al-Shabibi. "Monitoring individual traffic flows within the ATLAS TDAQ network". Proceeding of 17th International Conference on Computing in High Energy and Nuclear Physics (CHEP09), Prague, Czech Republic, 2009.

[8] RRDtool - Round Robin Datafiles for storing time series. http://oss.oetiker.ch/rrdtool/.

[9] D. Savu A. Al-Shabibi S. Batraneanu B. Martin R. Sjoen S. Stancu. Efficient Network Monitoring for Large Data Acquisition Systems. 13th International Conference on Accelerator and Large Experimental Physics Control Systems(ICALEPCS) 2011 , Grenoble, France, 2011.

[10] The Python Programming Language. http://www.python.org.

[11] OpenSceneGraph - open source high performance 3D graphics toolkit. http://www.openscenegraph.org.

[12] OpenGL - The industry standard for high performance graphics. http://www.opengl.org/.

[13] The Standard Template Library : Introduction. http://www.sgi.com/tech/stl/stl_introduction.html.

[14] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, first edition, 1995.

[15] cURL library for enhancing Web content retrieval. http://curl.haxx.se/.

[16] P. Martz. OpenSceneGraph Quick Start Guide. Computer Graphics Systems Development Corporation, 2007.http://www.lulu.com/items/volume_51/767000/767629/3/print/OSGQSG.pdf

[17] Stephanie Houde. Iterative design of an interface for easy 3D direct manipulation. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, pp. 135-142, 1992.

[18] Details-On-Demand technique. http://www.infovis-wiki.net/index.php?title=Details_on_demand.