The integrated graphical user interface of the trigger and data acquisition system of the ATLAS experiment at the LHC

G Avolio\(^1\), M Caprini\(^2\), G Lehmann Miotto\(^3\)

\(^1\) Department of Physics and Astronomy, University of California, Irvine, USA
\(^2\) National Institute of Physics and Nuclear Engineering, Bucharest, Romania
\(^3\) CERN, Geneva, Switzerland

E-mail: Giuseppe.Avolio@cern.ch

Abstract. The ATLAS experiment at the Large Hadron Collider at CERN relies on a complex and highly distributed Trigger and Data Acquisition system to gather and select particle collision data at unprecedented energy and rates. The main interaction point between the operator in charge of the data taking and the Trigger and Data Acquisition (TDAQ) system is the Integrated Graphical User Interface (IGUI). The tasks of the IGUI can be coarsely grouped into three categories: system status monitoring, control and configuration. Status monitoring implies the presentation of the global status of the TDAQ system and of the ATLAS run, as well as the visualization of errors and other messages generated by the system; Control includes the functionality to interact with the TDAQ Run Control and Expert System; Configuration implies the possibility to give the current status and modify some parameters of the TDAQ system configuration. This paper describes the IGUI design and implementation. Particular emphasis will be given to the design choices taken to address the main performance and functionality requirements.

1. Introduction
The TDAQ system [1] of the ATLAS detector [2] is composed of a large number of distributed hardware and software components (about 3000 machines and more than 15000 concurrent processes) which in a coordinated manner provide the data-taking functionality of the overall system. The system is required to handle data coming in parallel from the detector readout over some 1600 point-to-point readout links.

The Online Software [3] encompasses the software to configure, control and monitor the TDAQ system. It is a framework which provides essentially the glue that holds the various sub-systems together. The IGUI, as a part of the Online Software, is the main interaction point with the operator and provides an integrated view of the TDAQ system.

1.1. The IGUI in the Online Software
As a component of the Online Software infrastructure, the IGUI interacts with several software services:

- **Information Service (IS):** it allows software applications to exchange user-defined information;
- **Message Reporting System (MRS):** it provides a facility which allows all software components to report messages to other components of the distributed TDAQ system. Messages are passed using a subscription-notification mechanism;
- **Expert System (ES):** it performs the analysis of the errors and decides what actions are needed;
- **Run Control (RC):** it steers the data acquisition by starting and stopping processes and by carrying all data-taking elements through well defined states in a coherent way (finite state machine pattern). The RC is organized as a hierarchical tree (run control tree) of run controllers following the functional de-composition into systems and sub-systems of the ATLAS detector;
- **Configuration database:** it provides the description of the TDAQ system configuration;
- **Process Manager (PMG):** it offers a service to create, control and monitor the status of all the processes in the TDAQ system;
- **Resource Manager (RM):** it provides management of hardware and software resources in the system.

All the components of the Online Software communicate with each other through CORBA\(^4\).

![Figure 1. The ATLAS TDAQ IGUI. The core part includes the main interface to the Run Control (the upper left pane), the interface to the Message Reporting System (the table at the bottom of the frame) and a tabbed panel reporting information about the ATLAS run (the region just beneath the main Run Control interface). All the additional panels are contained in a dedicated tabbed pane (in the picture the Run Control panel is shown).](http://www.corba.org)

### 2. Requirements

There is a number of main requirements that have guided the general design of the IGUI:

- **System status**
  - the IGUI shall present the global status of the TDAQ system;
  - the IGUI shall be updated when the information about the TDAQ system changes;
  - the IGUI shall display information about the status of an ATLAS run (*i.e.*, run parameters and run configuration);
  - the IGUI shall be able to receive and display MRS messages;
  - the IGUI shall show messages according to dynamically configurable MRS subscription criteria;
  - the IGUI shall display the configuration of the run controllers;
  - the IGUI shall display the status of all the run controllers;

- **System control**
  - the IGUI shall be able to send commands to the run controllers;
  - the IGUI shall be able to communicate with the Expert System (*i.e.*, sending and receiving commands);

\(^{4}\) [http://www.corba.org](http://www.corba.org)
concurrent running copies of the IGUI shall be allowed but system control shall be exclusive;

- **System configuration**
  - the IGUI shall provide the possibility to modify the system configuration;

- **Extendibility**
  - the IGUI functionality domain shall be extended with specialized custom panels;
  - the IGUI shall be able to create and load any additional panel at run-time.

3. Implementation

The IGUI (figure 1) is implemented in the Java programming language (currently version 1.6). This choice has been done taking into account the language native multi-threading support and the versatility and completeness of Swing and the availability of third-party libraries providing several additional graphical widgets.

The IGUI user interface can be coarsely divided into a core part (containing the main interfaces to the Run Control, the Message Reporting System and the run settings) and a modular part consisting of a tabbed pane which can host additional panels covering sub-system specific functionalities.

**Figure 2.** The main interface to the Run Control: each button corresponds to a transition command that the user can send to the RC.

**Figure 3.** The operator can use this simple panel to set the run parameters.

3.1. Interface to the Run Control

As previously mentioned in section 1.1 the RC system follows a Finite State Machine (FSM) pattern. The IGUI holds an internal representation of the RC FSM states and transitions; this allows safer operations because the user is never allowed to send invalid transition commands to the RC system, making practically impossible for the operator to lead the system into an inconsistent state (figure 2).

The RC exchanges information with the IGUI through the Information Service (IS): as soon as the RC state changes, the IGUI is notified and detects whether a transition has taken place. A detailed view of the run control tree is provided by an additional panel (figure 1) which allows also sending commands to individual run controllers.

3.2. Interface to the Message Reporting System

The IGUI is able to receive MRS messages fulfilling some criteria set by the user (i.e., only messages with a defined severity or coming from a certain set of applications will be received).

Messages are reported in a table with columns showing different message qualifiers: time stamp, severity, name of the application producing the message, identifier and message text. The message severity is shown in different colors to help the user quickly identifying critical messages.

In order to avoid memory exhaustion messages are stored in a buffer whose size (within some bounds) can be specified by the operator.

[^5]: [http://java.sun.com/javase/6/docs/api/javax/swing/package-summary.html](http://java.sun.com/javase/6/docs/api/javax/swing/package-summary.html)
3.3. Interface to the Expert System
Whenever the Expert System (ES) needs the operator input (i.e., whether to disable a component causing troubles in the system), it can make remote calls to the IGU, which acts as a CORBA server. The IGU will show pop-up dialogs containing the message coming from the ES which, on the other side, may receive the user answer via a call-back mechanism.

3.4. Run information and settings
The Run Information and Settings pane (figure 3) reports important information about the on-going ATLAS run and allows the user to set some run-specific parameters (i.e., whether acquired data should be written to disk, a limit on the number of events to record).

Information exchange is performed through the IS system: the IGU is notified about any change in the run status and updates any run parameter modified by the operator.

3.5. Access control
Only one running instance of the IGU gives the user full control of the system, while others are executed in monitoring-only mode. This result is achieved via the Resource Manager (RM) component of the ATLAS Online Software: when the IGU starts, it tries to lock a resource in the RM; if that resource cannot be locked then the IGU disables any functionality which may cause a modification in the state of the system. Resource release and acquisition can be performed at run time allowing dynamic access control switching.

3.6. Additional panels
The IGU functionality can be easily extended with additional panels to cover different operational domains. Some panels are built-in because considered critical for safe ATLAS operations, while others can be loaded on-demand by the user. The IGU retrieves the list of additional panels from the Configuration Database and builds a check-box list containing their identifiers: the user can create or destroy a panel checking or clearing the corresponding box. Typical examples of additional panels are:

- **Data Flow**: it produces plots showing the time evolution of some critical system parameters;
- **PMG**: it shows the list of processes running on various hosts;
- **Segments & Resources**: it shows a hierarchical view of all the resources available in the TDAQ system, and allows the operator to dynamically enable or disable some of them (information is retrieved from the Configuration Database).

Panels are created at run-time using the Java reflection mechanism. Any panel has to extend the IGuiPanel class so that the only information to be retrieved from the Configuration Database is the panel full class name. The IGuiPanel interface is used for panels to be notified about modifications in the system state (e.g., the RC system has performed some transition or the Configuration Database has been changed), in the access control or just to signal a panel when it is selected by the user. Such methods can be overridden to customize the panel behavior.

3.7. Logging
The IGU offers a unified logging facility using the java.util.logging.Logger\(^6\) class. The aim of the provided logging facility is to have an uniform format of log messages produced by custom panels. For each log message the time stamp, severity and text are reported together with the signature of the method producing it. The user interface offers the possibility to apply a filter on the log message severity.

\(^6\) [http://java.sun.com/javase/6/docs/api/java/util/logging/Logger.html](http://java.sun.com/javase/6/docs/api/java/util/logging/Logger.html)
3.8. Threading model

When working with Swing components two rules must be honored:

- Swing is not thread safe and every Swing component has to be accessed in the Event Dispatching Thread (EDT);
- Time consuming tasks have to be executed in a thread different then the EDT to ensure user interface responsiveness.

Many tasks executed by the IGUUI are potentially time consuming because they perform calls to remote services (e.g., the RC, the IS, the RM). That means that a proper threading model is needed to not violate the Swing rules. The implemented model is based on thread pools and their implementation is provided by the `java.util.concurrent.ThreadPoolExecutor` class. Thread pools are organized in operational areas in order to avoid service starvation and saturation (i.e., all threads in the pool may be used to serve only one consumer). Each additional panel receives notifications through the `IguiPanel` interface in a dedicated thread: this makes panel development easy (from a panel point of view the whole application framework is single threaded) and exploits parallelism.

4. Quality control

Synthetic tools have been used during the development phase mainly to address any violation of the Swing rules and to check memory usage and thread starvation or dead-locks:

- **Swing Explorer**: it allows to explore the Swing component hierarchy and identify all the EDT violations;
- **Visual VM**: mostly used for profiling and Garbage Collector activity monitoring;
- **Find Bugs**: static code analyzer to find code bugs.

The usage of such tools allowed keeping the IGUUI code and functionality under control.

5. Performance Optimization

The IGUUI load is high during each RC FSM transition: all the controlled applications (about 15000 today) update their state and produce several MRS messages (with peaks of 1000 messages per second). Keeping the IGUUI responsive and, at the same time, promptly updating the status of the system has been the main challenge during development. In the followings some techniques used to reach the performance goal are described.

5.1. Run Control tree

The RC tree in the Run Control panel (figure 1) contains several nodes, each representing a run controller. The rate of state updates can be as high as 30000 per second. In order to deal with such a high rate, information is updated only for visible nodes (state information is lazily loaded when a node is made visible). Since the same node may receive several concurrent and out of order state update notifications, a thread safe lock-less algorithm based on the `Compare And Swap` technique has been developed.

Given the huge number of nodes in the tree, a search interface has been implemented to easily find a particular node. To improve performances all the tree paths are indexed the first time the search interface is used and every time the tree structure is modified (i.e., nodes are inserted or removed). The

---

7 [http://java.sun.com/javase/6/docs/api/java/util/concurrent/ThreadPoolExecutor.html](http://java.sun.com/javase/6/docs/api/java/util/concurrent/ThreadPoolExecutor.html)
8 [https://swingexplorer.dev.java.net/](https://swingexplorer.dev.java.net/)
9 [https://visualvm.dev.java.net/](https://visualvm.dev.java.net/)
10 [http://findbugs.sourceforge.net/](http://findbugs.sourceforge.net/)
11 [http://en.wikipedia.org/wiki/Compare-and-swap](http://en.wikipedia.org/wiki/Compare-and-swap)
tree searching feature has been implemented using a slightly modified version of the JIDE Common Layer\textsuperscript{12} TreeSearchable interface.

5.2. MRS messages
In order to absorb the peaks in message rate a producer/consumer pattern has been implemented: a thread receives the messages and puts them in a blocking queue, while a second thread regularly drains the messages from the queue and updates the user interface. This pattern allows to tune the consumer message processing time and to signal the user interface about new messages in bunches and not on a single message basis.

5.3. Busy state
Some operations performed by the IGUI are so critical (and of long duration) that the user should not interact with the system until they are completed. In this case the IGUI can be put in a busy state: the User Interface top frame is decorated in such a way all the mouse and keyboard events for its sub-components are caught, and messages about the status of the underlying job progression are shown. This functionality is provided by JXLayer\textsuperscript{13}: it acts like a glass-pane layer which can be made opaque or transparent on request. Timeouts are foreseen in order to avoid the IGUI to stay in a busy state for an indefinite period of time.

5.4. Creation and selection of panels
Several additional panels have been developed in order to add specific functionalities (\textit{i.e.}, operations related to an ATLAS sub-detector or sub-system) to the IGUI. The IGUI start-up procedure could be considerably slowed down with the increasing number of panels. In order to solve this performance issue, only a critical subset of all the available panels is created when the IGUI is started. All the other panels can be loaded on-demand by the operator, as already explained in section 3.6.

Moreover each panel is notified any time it is selected or unselected in the panel tabbed pane. This strategy helps improving the panel usage of resources and computing power (\textit{i.e.}, when a panel is not visible, it should stop receiving notifications from the Information Service; the needed information can be updated again when the panel is re-selected).

6. Conclusions
In this paper the design, implementation and performances of the TDAQ IGUI of the ATLAS experiment at the Large Hadron Collider have been presented. The aim of the software design has been to realize a component able to monitor and control a very complex and distributed system.

The strict quality control on the IGUI code and functionality helped to fulfill the requirements needed to guarantee stable and reliable operations. Nowadays the IGUI is successfully used to monitor and control the ATLAS TDAQ system during data-taking activities. Moreover the IGUI is also used by ATLAS sub-detectors for their stand-alone and calibration runs.

References
[1] Jenni P, Nessi M, Nordberg M and Smith K 2003 ATLAS high-level trigger, data-acquisition and controls : Technical Design Report (Geneva : CERN)
[2] The ATLAS Collaboration 2008 The ATLAS experiment at the CERN Large Hadron Collider \textit{J.Instrum.} 3 S08003
[3] Lehmann Miotto G \textit{et al}. 2010 Configuration and control of the ATLAS trigger and data acquisition \textit{NIMA} 623 549-551

\textsuperscript{12} http://www.jidesoft.com/products/oss.htm
\textsuperscript{13} https://jxlayer.dev.java.net