EPJ Web of Conferences **245**, 05034 (2020) https://doi.org/10.1051/epjconf/202024505034
CHEP 2019

A multi-purpose user interface for the iFDAQ of the COMPASS experiment

Antonín Květoň[1,*], *Martin* Bodlák[1], *Vladimir* Frolov[2], *Stefan* Huber[3], *Vladimír* Jarý[4], *Igor* Konorov[3], *Josef* Nový[4], *Dominik* Steffen[3], *Ondřej* Šubrt[4], and *Miroslav* Virius[4]

[1]Charles University, Prague, Czech Republic
[2]Joint Institute for Nuclear Research, Dubna, Russia
[3]Technical University of Munich, Germany
[4]Czech Technical University in Prague, Czech Republic

Abstract. In HEP experiments, remote access to control systems is one of the fundamental pillars of efficient operations. At the same time, development of user interfaces with emphasis on usability can be one of the most labor-intensive software tasks to be undertaken in the life cycle of an experiment. While desirable, the development and maintenance of a large variety of interfaces (e.g., desktop control interface, web monitoring interface, development API...) is often simply not feasible, as far as manpower is concerned. We present a solution employed in the control software of the iFDAQ of the COMPASS experiment at CERN. Being a mix of a command-line and terminal tool, this interface can fulfill the roles of a dynamic monitoring interface, a control interface, and a scripting API simultaneously. Furthermore, it can easily be used as a remote access tool for operations experts, needing nearly no setup user-side and being compatible with smartphones. We also discuss the methodology and results of a concrete use case – automated run control for performance tests of the iFDAQ readout software.

1 Introduction

A new data acquisition system (DAQ) has been deployed at the COMPASS experiment [1] in 2014 [2] and used for data-taking during the period from 2015 to 2018. The system, referred to as the iFDAQ [3] is accompanied by a software framework [4, 5], which handles not only run control, configuration and monitoring (CCM), but also readout, processing and data storage.

Several interfaces have been an integral part of the framework since its conception, most notably the run control GUI. During the period from 2015 to 2018, an API and a remote control interface were also implemented, but have proven to be difficult to maintain, given the manpower limitations of the software development group. This gave rise to the need for a new approach to interfaces to the framework, that emphasizes manpower efficiency. This article describes the nature of the new interface architecture as well as the rationale behind the design decisions taken.

*e-mail: antonin.kveton@cern.ch

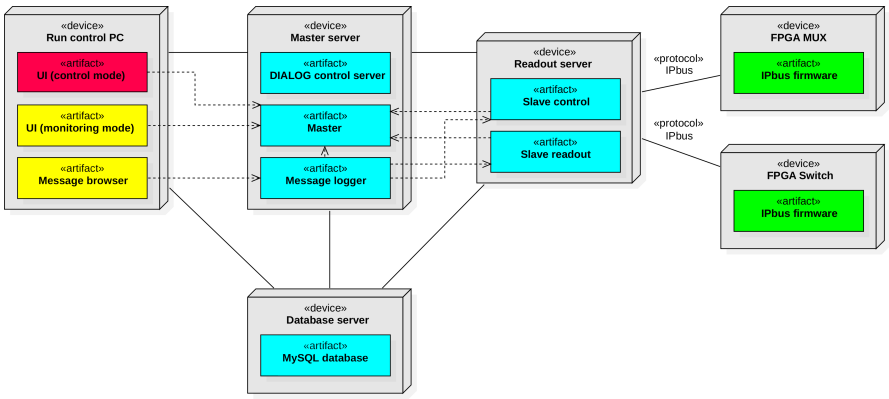

Figure 1. Software architecture of the iFDAQ software framework

2 Interface requirements

The types of interfaces required have been identified as the following:

- **Desktop GUI** – to be used in the control room; emphasis should be placed on user experience, as shifts are traditionally used to train new members of the collaboration
- **Remote access UI** – to be used by experts to solve problems remotely; to take form of a web interface or an X-forwarded on-site desktop GUI
- **Programming API** – to expose control and monitoring functionality to other systems
- **Scripting API** – to allow for quick and easy creation of custom user utilities

3 New interface architecture

In order to achieve maximum manpower efficiency, emphasis was placed on reusability of the interfaces for different purposes. In both the old and new architecture, the *Master* process/service, which effectively acts as a DIALOG server (cf. [6]), functions as a middleman between the interfaces and the rest of the system, namely the slave processes. While in the old architecture the clients would share only little communication-protocol code, the new architecture introduced the concept of a *Common client core*.

3.1 Common client core

The common client core, implemented as a shared object library with a C++ API, handles communication with the *Master* service. Upon initialization, it subscribes to the Master's DIALOG services and periodically receives messages containing information concerning system state as well as monitoring information. The core contains the logic to parse, store and expose this information to the user in the form of C++ functions. Being event-driven, the events (such as the on value update event) are also available to the user, implemented as exposed Qt signals.

The common client core also unifies communication being sent to the *Master* service, specifically configuration and run control commands, internally implemented as DIALOG commands.

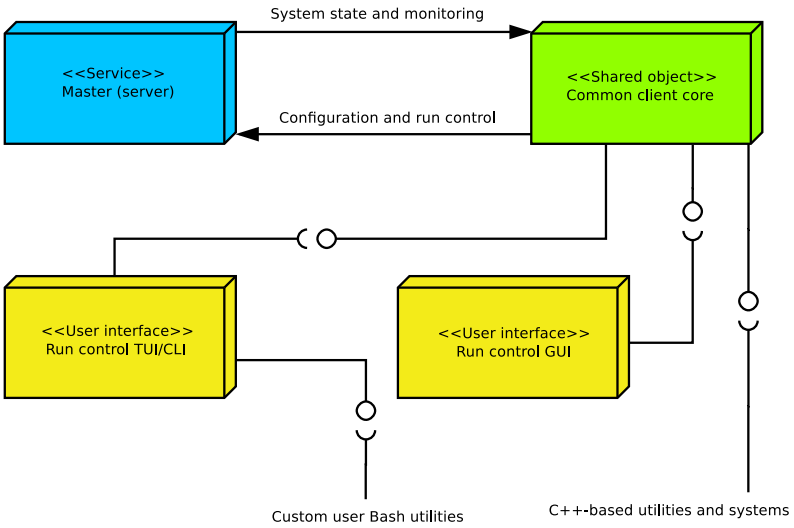

Figure 2. New iFDAQ interface architecture

Containing the system state information and exposing configuration and run control functionality, the common client core can therefore be used as both the basis for any user interfaces internal to the framework as well as what is effectively a C++ API to the iFDAQ system, which can then be used by outside systems. As DIALOG does not support communication over different subnets, applications making use of this API have to be run on the same network as the *Master* service.

3.2 Run control GUI

The first package building on the common client core is the Run control GUI. This is a heavy-weight interface to be used in the control room, taking up three 1920x1080 screens in the default state. The system state is represented in the form of GUI elements, with extra emphasis on usability.

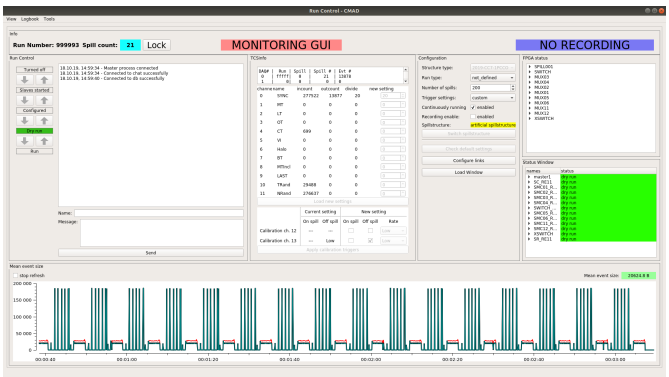

Figure 3. The main window of the Run control GUI. The main features present are buttons to initiate state transitions of the DAQ system, trigger and event rate display, DAQ state machine state display and FPGA registry display.

3.3 Run control TUI/CLI

The scale of the run control GUI makes it very impractical for remote usage, given the limitations of the DIALOG library. The remote user has to resort to X-forwarding the application over several SSH hops, which is not a satisfactory solution, as the performance of a large GUI forwarded in this way is extremely poor. While alternative solutions that improve performance exist (such as X2Go and Xpra), it was decided that a more dedicated solution would be needed for remote access.

A web interface was one of the proposed solutions, but was determined to be in conflict with the focus on manpower-efficient solutions, as this would require development of an iFDAQ API in a programming language better suited for web development (such as PHP or Python), effectively doubling the common client core workload.

The more economic final solution was to use a dynamic text-based terminal interface whose commands could also be run in command-line mode, and thereby also effectively creating a bash API to the system that could allow for a CGI implementation.

3.3.1 Terminal mode

The terminal part of this interface offers dynamic visualization of the state of the iFDAQ system, as well as the possibility of control. Once launched, commands, categorized into sub-menus, are made available based on the current state of the system. Monitoring commands utilize ANSI escape sequences to provide dynamic output, in which displayed values can be seamlessly refreshed on update. Commands requiring input from the user can be either run with switches and parameters, or, if the required information is not provided in the form of switches and parameters, "interactively" (the user will be asked to provide the required values). Conveniently, this mode is also compatible with smartphones which allow for installation of a SSH client.

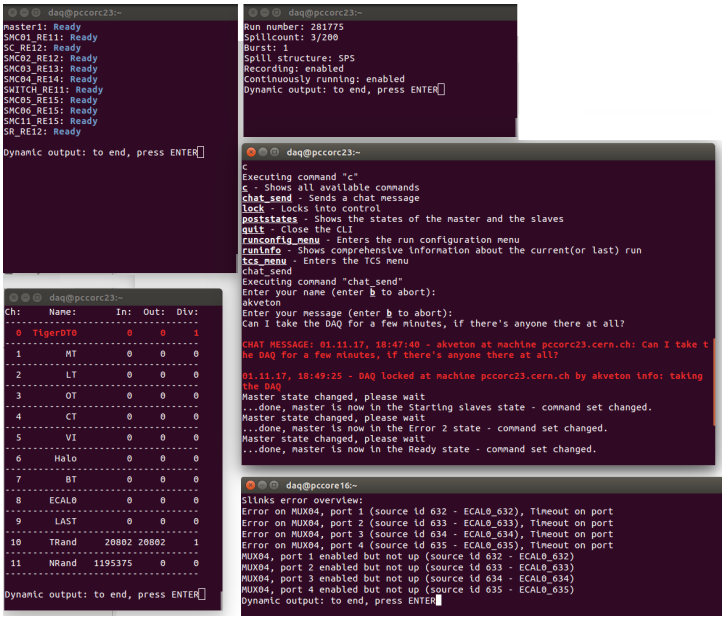

Figure 4. Several instances of the terminal mode running in parallel, effectively replacing the need for the Run control GUI. From top to bottom and left to right: state machine states, trigger rates and prescaler settings, current run metadata and configuration, interactive argument input, FPGA error registry overview

3.3.2 Command-line mode

The command-line mode allows the user to pass one or more commands as a parameter to an executable binary, and they will then be run, as they would be in the terminal mode. While the control commands are identical to their respective counterparts of the terminal version, monitoring commands provide machine-readable output – strings are substituted for enumerated integer values, the output follows a pre-defined format and ANSI escape sequences are no longer used.

The values of monitoring variables are not cached and the process has to reconnect to the DIALOG server and wait for an update, thereby causing approximately a 1-second execution time, making the command-line mode unsuitable for high-performance applications in the current state.

```
CLI -c lock -n akveton -m "Locking in to exclude a port" + setportm -m 1 -p 12 -d
```

Listing 1. Example of a command call in the command-line mode from a bash shell. Two commands are chained using the plus sign and executed sequentially. The first command establishes an exclusive lock on the system, preventing any other running instances of the common client core from sending control commands to the Master service. A user-input message describing intent behind the lock is also sent to all instances. The second command disables port 12 on multiplexer 1.

4 CLI use case – automated run control

One of the first real extensive use cases of the CLI mode of the Run control TUI/CLI was automated run control in order to measure software readout performance of the iFDAQ software in 2019. An artifical data generator that can generate raw data files with a specified number of events of a given event size was created. This data was then to be injected into the memory of a single readout server, and subsequently, the readout performance of the Slave readout process was to be measured in dependence on the event size. Multiple runs were planned to be collected per event size increment – the first measurement cycle required the collection of 3 runs in each of the 93 event size increments. It was therefore very desirable to automate this task and a wrapper Python script which handled the logic of the measurement cycle was created, using the CLI mode for run control. The duration of a single cycle was ~3 days.

While the CLI mode proved to be sufficient, it proved that a C++ API would have been more efficient for this specific use case, as it takes up to two seconds to run a single command in the CLI mode, which eventually leads to a considerable overhead from state check commands. However, the common client core was not yet fully implemented at the time.

The results of the aforementioned measurement are shown in Figure 5. This measurement was the first of a series that aimed to optimize the readout performance of the process. Several major improvements have been implemented since the time of the initial measurement – as such, the results are not representative of the current state of the system, which will be published in the future. The two phenomena of note observed in the initial measurement (periodic processing speed fluctuation and overall speed falloff after having reached the plateau of the curve) were found to be systematic errors and eliminated in later measurements.

The total speed has been fit using the following function:

$$y = y_{min} - (y_{min} - y_{max}) \exp(-kx) \tag{1}$$

, where k is a free parameter.

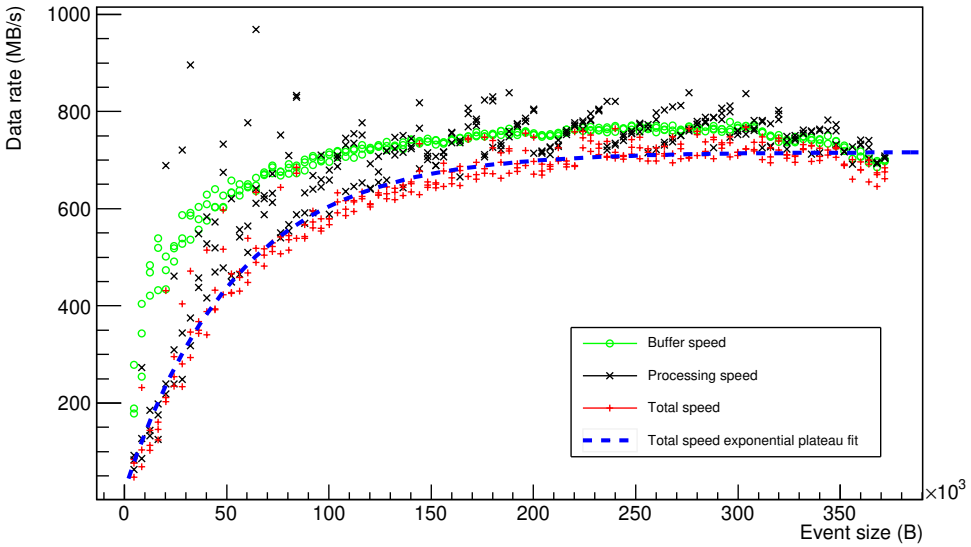

Figure 5. Readout performance results for a single Slave readout process running on a single readout server. Buffer speed represents performance of all logic leading up to the processing stage, processing speed represents performance of the processing stage (c.f. [7]), and total speed combines the two.

5 Conclusion

The new interface architecture of the iFDAQ aims to siginificantly reduce long-term developer workload and ease the unified addition of new interface features. The common client core has been implemented in 2019 and is nearing its testing phase as of February 2020. The Run control GUI is currently using its old back-end implemenentation, which is planned to be changed over to the common client core and tested before the COMPASS 2020 dry run. The Run control TUI/CLI has been implemented based on the GUI back-end, and therefore is also planned to be changed over to the common client core in 2020. Both the terminal and command-line modes having seen use among the members of the collaboration, the Run control TUI/CLI has proven to be a valuable addition to the iFDAQ software framework.

References

[1] COMPASS Collaboration (Gautheron, F et al.), **CERN-SPSC**, CERN-SPSC-2010-014
[2] J. Nový et al, Journal of Physics: Conference Series, **Volume 664**, 082042 (2015)
[3] D. Steffen et al, Proceedings, 38th International Conference on High Energy Physics, **Volume ICHEP2016**, 912 (2016)
[4] M. Bodlák et al, Acta Polytechnica, **Volume 53**(4), 338-343 (2013)
[5] M. Bodlák et al, Nuclear and Particle Physics Proceedings, **Volumes 273-275**, 976-981 (2016)
[6] Y. Bai et al, International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering, **Volume 11**, 372-381 (2017)

[7] J. Nový et al, Fifth International Conference on Communication Systems and Network Technologies, **Volume 2015**, 1303-1306 (2015)