The Online Monitoring API for the DIALOG Library of the COMPASS Experiment

Ondřej Šubrt²,⁴,*, Martin Bodláčk³, Vladimír Frolov⁵, Stefan Huber¹, Matouš Jandek⁴, Vladimír Jarý⁴, Antonín Květoň⁴, Dmytro Levit¹, Josef Nový⁴, Dominik Steffen¹, Jan Tomsa⁴, and Miroslav Virius⁴

¹Technical University of Munich, Germany
²CERN, Geneva, Switzerland
³Charles University, Prague, Czech Republic
⁴Czech Technical University in Prague, Czech Republic
⁵Joint Institute for Nuclear Research, Dubna, Russia

Abstract. Modern experiments demand a powerful and efficient Data Acquisition System (DAQ). The intelligent, FPGA-based Data Acquisition System (iFDAQ) of the COMPASS experiment at CERN is composed of many processes communicating between each other. The DIALOG library covers a communication mechanism between processes and establishes a communication layer to each of them. It has been introduced to the iFDAQ in the Run 2016 and improved significantly the stability of the system. The paper presents the online monitoring API for the DIALOG library. Communication between processes is challenging from a synchronization, reliability and robustness point of view. Online monitoring tools of the communication between processes are capable to reveal communication problems to be fixed in future. The debugging purpose has been crucial during introduction period to the iFDAQ. On the other hand, based on the measurement of communication between processes, the proper load balancing of processes among machines can improve the stability of the system. The online monitoring API offers a general approach for the implementation of many monitoring tools with different purposes. In the paper, it is discussed its fundamental concept, integration to a new monitoring tool and a few examples of monitoring tools are given.

1 Introduction

The Common Muon and Proton Apparatus for Structure and Spectroscopy (COMPASS) [1, 2] is a multipurpose experiment at the Super Proton Synchrotron (SPS) in the CERN’s North Area. The purpose of the COMPASS experiment is to study the structure of gluons and quarks and the spectroscopy of hadrons using high intensity muon and hadron beams. In 2010, the experiment entered its second phase COMPASS-II [2] focusing on the Drell-Yan (DY) effect [2], the Primakoff scattering [2], and the Deeply Virtual Compton Scattering (DVCS) [2].

The paper presents the online monitoring API for the DIALOG library. In complex softwares, such as the iFDAQ, having tens of processes communicating with each other, the

*the corresponding author, e-mail: ondrej.subrt@cern.ch
Inter-Process Communication (IPC) is absolutely essential to satisfy correct synchronization and a proper data taking. The DIALOG library is a new communication library for IPC. The online monitoring API for the DIALOG library provides an easy use interface for the monitoring of an entire communication system.

The paper is organized as follows. In Section 2, an overview of the iFDAQ is presented from a hardware and software point of view, respectively. Section 3 is concerned with a brief description of the DIALOG library. Section 4 deals with the online monitoring API for the DIALOG library. It presents how to easily develop own monitoring tools and gives a few examples of monitoring tools. In Subsection 4.1, the online monitoring of communication among processes via the DIALOG GUI is stated. The DIALOG GUI allows a visualization of all processes involved in the application.

2 iFDAQ Architecture

The original DAQ system was based on DATE DAQ software of the ALICE experiment [3]. Current COMPASS DAQ system – the intelligent, FPGA-based Data Acquisition System (iFDAQ) [4–6] – is new and was designed in 2014 to improve its scalability, reliability, and performance; however, now, the process of event building is executed completely in hardware and the number of readout computers was reduced to only four (up to eight in full setup).

![COMPASS iFDAQ topology](image)

**Figure 1.** The COMPASS iFDAQ topology.

The COMPASS iFDAQ topology is shown in Figure 1. The first layer of the iFDAQ is called frontend electronics. Its task is to capture signals directly from the detectors and convert them to digital values. There are approximately 300,000 data channels coming from this layer. Then, the data are readout by roughly 250 of CATCH [7], HGeSiCA [8], and Gandalf [9] concentrator modules based on VME standard and grouped into subevents. The assembly, buffering and readout of these subevents is then performed by two layers of field programmable gate array (FPGA) cards, which replaced the original server-based event
building and buffering system. The resulting full events are then processed by readout engine computers, stored at first on local hard disks and subsequently transferred to CASTOR [10]. The iFDAQ enables configuration and control of hardware, monitors data taking, controls data flow, logs information and monitors errors and messages from the system. This is ensured by six main processes [6, 11]: Master, Slave-control, Slave-readout, Runcontrol GUI, MessageLogger and MessageBrowser. The Master process is used for communication between Runcontrol GUI and slave processes, the Slave-control process controls and monitors FPGA cards, the Slave-readout monitors a readout process and Runcontrol GUI monitors a state of the entire iFDAQ. The MessageLogger receives informative and error messages from other processes and the MessageBrowser enables access to these messages.

3 DIALOG Library

The DIALOG library [12] is a communication system both for distributed and mixed environments, it provides a network transparent inter-process communication layer. It is implemented in Qt framework and designed to meet following requirements:

- **Efficient communication mechanism** – Asynchronous behaviour, message sending and receiving as soon as possible.
- **Uniformity** – All processes use the same communication mechanism.
- **Transparency** – Any running process should be able to communicate with any process.
- **Reliability and robustness** – System recovery in a self-recoverable manner from error situations.

The DIALOG library uses a client/server mechanism and communication is based on a publish/subscribe method. It uses the TCP/IP protocol and sockets for message transmission. The information exchange between processes is handled by services when a process (server) publishes data to processes (clients) being interested in a particular service with a unique name. On the other hand, to control a particular process, a concept of commands is introduced – a process registers a command with a non-unique name it is willing to accept.

The communication layer of iFDAQ is based on the DIALOG library since 2016 and this integration improved the stability of iFDAQ significantly.

4 The DIALOG Online Monitoring API

The DIALOG library distinguishes three types of a process. The process type determines the purpose of a process and how the DIALOG library should deal with it.

- **ControlServer** – The Control Server keeps an up-to-date list of all processes, services and commands in the system. It receives registration messages from providers and request messages from subscribers. All processes send heartbeats at regular intervals so that the Control Server can be assured that they are working properly.
- **Custom** – It covers all processes they should communicate through the DIALOG library.
- **Monitoring** – Processes being responsible for the monitoring of the DIALOG library. Each of them represents a monitoring tool with a unique purpose. This general concept offers an easy to use API for a developer to implement any monitoring tool.

This section focuses on processes of type Monitoring. The Monitoring type identifies a process which does not contribute to the communication at all. It does not provide with any service. On the other hand, to be able to start listening to some/all communication, it can subscribe to a service or register a command it is interested in.
Firstly, a monitoring tool connects to the Control Server as well as anyone else. The Control Server recognizes the Monitoring type of a process and deals with it like with a monitoring tool during its whole life cycle.

The Control Server sends a monitoring info to a monitoring tool in XML format after successful connection of the monitoring tool to the Control Server. This monitoring info contains list of all connected processes of type Custom to the Control Server. The monitoring info in XML format also consists of all services being provided and subscribed by processes and all commands being registered in the Control Server. If something changes (e.g. a lost process, a new service, a new process, etc.), the Control Server will recognize it and re-send a new monitoring info immediately to all monitoring tools currently connected.

The monitoring info considers changes in the list of processes connected to the Control Server. Once a connected process terminates or crashes, the Control Server recognizes it and re-sends the monitoring info in XML format with the updated list of connected processes. If a new service is provided by any process or some service is subscribed by any process, the monitoring info is re-sent again to all monitoring tools. The last case is a registration of a new command. In this case, current monitoring info is again updated and re-sent. Based on the information from the monitoring info in XML format, a monitoring tool knows everything it needs to know. If a monitoring tool is interested in all communications it subscribes to all services and registers all commands and starts to listen to them. If a monitoring tool is interested in some portion of services, it subscribes and starts to listen to those services again.

The monitoring info in XML format begins with an XML declaration describing XML version and encoding. The uppermost tag <processes> encapsulates the list of all processes. Each <process> tag is a child element of <processes> tag and has attribute name, address, port and pid. The tag <process> either has no child elements or has the list of services it provides and registered commands it is willing to accept. The tag <command> is considered to be an empty-element tag and has only attribute name, such as <command name="commandName"/>. The tag <service> has also only one attribute name and can contain child elements representing the list or receivers, i.e., the list of processes being subscribed to the service. The tag <receiver> consists of attributes name, address, port and pid. The sample of monitoring info XML format follows in Listing 1.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<processes>
  <process name="SR_RE11" address="pccore11.cern.ch" port="57143"
pid="22640">
    <service name="INFO_SERVICE_56">
      <receiver name="MSGBrowser" address="pccorc21.cern.ch"
port="33511" pid="24958"/>
      <receiver name="Master" address="pccore15.cern.ch" port="51523"
pid="17376"/>
      <receiver name="MSGLogger" address="pccore15.cern.ch"
port="56614" pid="17377"/>
    </service>
    <command name="RUN_CONTROL_56"/>
  </process>
</processes>
```

Listing 1. The monitoring info XML sample.

4.1 The DIALOG GUI

The DIALOG GUI is one example among many of the monitoring tools using the online monitoring API for the DIALOG library. A behaviour of complex distributed applications
can be very difficult to understand without a help of a dedicated tool for online monitoring. The DIALOG GUI allows a visualization of all processes involved in the distributed communication system as shown in Figure 2.

![DIALOG GUI](image)

**Figure 2.** The DIALOG GUI.

A development of a distributed system is quite challenging from a synchronization and robustness point of view. The DIALOG GUI helps with a debugging of misleading functioning so that the system start to work properly. It provides all information from the monitoring info in a well-arranged way. The main widget consists of two parts. The main part contains all necessary information about connected processes to the Control Server with the process type `Custom`. There is the name of each process, the machine where it runs, the port on which it listens to and its process ID. Moreover, each row corresponding to a single process offers the list of provided services, subscribed services and registered commands by the process. If the DIALOG system contains many processes, a user appreciates a filter in the DIALOG GUI for an easy searching.

The provided services widget shows the list of provided services by the selected process. A user can select a service from the list and see the list of all subscribers of the service. Moreover, a user can start to listen to the selected service and see what the service is providing, see Figure 3.

The subscribed services widget shows all services being subscribed by the selected process. Basically, it is a list of all services the process is interested in. In the subscribed services widget, a user can select a service from the list of services and see which process is providing the service. Moreover, a user can start to listen to the selected service again. In both provided/subscribed services widget, a user can start to listen to all services at once and see the entire process communication.

The commands widget is corresponding to the selected process and its registered commands it is willing to accept. In the widget, the list of registered commands is stated. A user can start to listen to the selected one or to all of them. Moreover, there is also possibility to send a command message directly from the DIALOG GUI using the selected command.
4.2 The DIALOG POST Daemon

Web-based applications offer a range of business advantages over traditional desktop applications. Web-based applications are:

- **Easier and more cost effective to develop** – Users access the system via a uniform environment (a web browser).
- **More useful for users** – Unlike traditional applications, web systems are accessible anytime, anywhere and via any device with an internet connection.
- **Easier to install, maintain and keep secure** – Once a new version or upgrade is installed on the host server, all users can access it straight away and there is no need to upgrade a device of each potential user.

Considering all above-mentioned aspects, the DIALOG library should provide a layer for communication between a desktop application and a web application. There is a wide range of possible solutions in terms of a system architecture. The paper considers and presents two possible solutions how to communicate between a desktop application and a web application – HTTP GET/POST methods and WebSockets, see in Section 4.3.

The DIALOG POST Daemon serves as a middleman between a desktop application side and a web application side. The DIALOG POST Daemon’s purpose is to catch all communication being sent among all processes with the process type Custom and transmit all data in JSON format using HTTP and its POST method. The POST request is received by a web application for a communication measurement. It is a measurement tool of overall communication and shows statistics and plots related to communication among processes. It can help with understanding of bottlenecks and better load balancing of processes among machines.

The main idea is that the DIALOG POST Daemon with the Monitoring process type subscribes to all services being provided and registers all commands being already registered on the Control Server. Once a new monitoring info comes, the DIALOG POST Daemon subscribes to new services and registers new commands. Then, all communication is sent to
the DIALOG POST Daemon. Basically, the DIALOG POST Daemon is capable to determine the communication between any processes.

A publish frequency for the POST method is set to 1 second and can be easily changed. During 1 second period, all received messages are gathered in JSON format and the collected JSON is published using POST method and freed every 1 second. Each message in JSON consists of information about a sender, i.e., a sender address, a sender port and its process name. To reduce JSON message size, messages are grouped by message itself. That means, if a message has more subscribers, the list of subscribers is only added. Therefore, the list of receivers is added (each entry with receiver address, receiver port and receiver name).

The rest of record is filled with the Date\(\text{Time}\), the Message\(\text{Header}\) and the Message\(\text{Body}\). The Message\(\text{Header}\) distinguishes a message type. Either it can be a service message or a command message. The Message\(\text{Header}\) contains also information about the name of service or command. The Message\(\text{Body}\) is filled with the transmitted information (message) itself. It is encoded using Base64 [13] since a message could be general indeed. It could contain also non-printing characters that would be lost during transmition. Base64 encoding allows to transmit not only messages but also files. The sample of JSON format being regularly sent using HTTP POST method follows in Listing 2.

```json
[  
    {
      "Sender": {
        "SenderAddress": "pccore15.cern.ch",
        "SenderPort": "12345",
        "SenderName": "Master"
      },
      "Receivers": {
        "Receiver0": {
          "ReceiverAddress": "pccore13.cern.ch",
          "ReceiverPort": "54698",
          "ReceiverName": "SR_RE13"
        },
        "Receiver1": {
          "ReceiverAddress": "pccore12.cern.ch",
          "ReceiverPort": "54879",
          "ReceiverName": "SR_RE12"
        }
      },
      "DateTime": "2017-09-13T07:20:25Z",
      "Message\(\text{Header}\)": "SERVICE\_MESSAGE\|STATUS\_OF\_SLAVES",
      "Message\(\text{Body}\)": "\text{MessageBody\|textEncodedInBase64} // base64 encoding"
    }
  ]
```

Listing 2. The DIALOG POST Daemon JSON sample.

Currently, the DIALOG POST Daemon is prepared and a web application is still under development. The development of web application should be finished in 2018 and deployment is planned in the late 2018.

### 4.3 The DIALOG WebSockets Daemon

The HTTP GET/POST method is suitable for a client-server architecture, when a client (the DIALOG POST Daemon) is publishing something to a server (a web application). On the other hand, to communicate in both directions efficiently, WebSockets can be used. WebSockets [14] is an advanced technology that makes it possible to open an interactive communication session between the user’s browser and a server. With this API, an application can
send messages to a server and receive event-driven responses without having to poll the server for a reply. WebSocket provides full-duplex communication channels over a single TCP connection. The idea is to develop the DIALOG WebSockets Daemon as a middleman between desktop communication system and a web application. Runcontrol GUI for monitoring and controlling of the iFDAQ as a web application is planned for year 2019.

5 Conclusion

The DIALOG library is a new communication library for iFDAQ of the COMPASS experiment at CERN. It is responsible for basically all communications inside the iFDAQ, in this environment it makes available around 100 services provided by 30 servers.

The DIALOG Online Monitoring API provides an easy use interface for the monitoring of an entire communication system. Its general implementation enables a development of various monitoring tools. The DIALOG GUI is found to be very useful not only for monitoring of the system state but also to determine available services at a given time. Furthermore, the DIALOG POST Daemon and the DIALOG WebSockets Daemon establish a connection between any desktop application and any web application. That would make any system based on the DIALOG library less dependent in terms of an operating system and environment.

6 Acknowledgment

This research has been supported by OP VVV, Research Center for Informatics, CZ.02.1.01/0.0/0.0/16_019/0000765 and by the Maier-Leibnitz-Labor, Garching and the DFG cluster of excellence Origin and Structure of the Universe.

References

[1] P. Abbon et al., Nucl. Instrum. Methods Phys. Res. A577, 455 (2007)
[2] V.Y. Alexakhin et al., COMPASS-II Proposal, The COMPASS Collaboration (2010)
[3] T. Anticic et al., CERN, EDMS 616039 (2006)
[4] M. Bodlak et al., Acta polytechnica: Scientific Journal of the Czech Technical University in Prague 53 (2013)
[5] M. Bodlak et al., Journal of Physics: Conference Series 513, 012029 (2014)
[6] M. Bodlak et al., Journal of Instrumentation 8, C02009 (2013)
[7] Electronic developments for COMPASS at Freiburg, http://hpfr02.physik.uni-freiburg.de/projects/compass/electronics/catch.html, accessed 2018-09-09
[8] IMUX/HGESICA module, http://wwwcompass.cern.ch/twiki/pub/Detectors/FrontEndElectronics/imux_manual.pdf, accessed 2018-09-09
[9] The GANDALF Module, http://wwwhad.physik.unifreiburg.de/gandalf/pages/hardware/the-gandalf-module.php?lang=EN, accessed 2018-09-09
[10] CASTOR — CERN Advanced Storage manager, http://castor.web.cern.ch, accessed 2018-09-09
[11] M. Bodlak et al., Nuclear and Particle Physics Proceedings 273, 976 (2016)
[12] M. Bodlak et al., International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering 11, 372 (2017)
[13] IETF, The Base16, Base32, and Base64 Data Encodings (Internet Engineering Task Force, 2006), RFC 4648
[14] IETF, The WebSocket Protocol (Internet Engineering Task Force, 2011), RFC 6455