Control software for the TOTEM DAQ

Carles Garcia Cabot

Supervisors:
Francesco Cafagna
Emilio Radicioni
Michele Quinto

August 26, 2016

1 Introduction

The main goals of the TOTEM[1] experiment at the LHC are the measurements of the elastic and total p-p cross sections and the studies of the diffractive dissociation processes. The TOTEM DAQ has been upgraded for the LHC's Run 2 with the Scalable Readout System (SRS), a scalable, multi-channel and general-purpose readout platform[2]. The upgraded DAQ has increased the experiment's trigger rate up to 100 kHz in stand-alone mode. Furthermore, the new system allows the TOTEM and CMS collaborations to perform common data taking during combined measurements campaigns.

For my Summer Student project at CERN I have developed a new software application to control and monitor the upgraded DAQ platform in a distributed environment. The new software, named SRSController, is necessary to operate the SRS within the CMS DAQ infrastructure during common data taking.

The purpose of this report is to explain the design of the SRSController and to serve as additional documentation for future users and developers.

2 Requirements

The application to be developed has to comply with the following requirements:

- be written in C++
- be developed using the XDAQ[3] framework
- be based on the *state design pattern*, following the standard CMS' run control state transition diagram
- comply with the CMS' run control message passing protocol
- implement a web interface using the XDAQ facilities
- upon receiving a state transition command the application has to configure the SRS hardware
- the content of the configuration to be applied to the hardware has to be specified in a user-defined XML file

- expose to the user interface the state of the XDAQ FSM
- expose to the user interface the configuration applied to the hardware
- expose to the user interface the reading of hardware status information for monitoring purposes. The list of the registers to be monitored has to be specified through a user-defined XML file.
- be generic, separating the hardware access layer API from the rest of the implementation, such that it can be easily extended to hardware interfaces other than the SRS
- exploiting the concept of sequences, abstract groups of hardware operations (i.e.: read, write) defined in the XML files and applied to the hardware, the web interface has to be implemented in a fully dynamic way.

3 Design

The operation of the SRSController can be divided in three main parts.

3.1 Parsing input files

The most important input files are the *sequences* and the *systems* XML files. The first contains multiple sequences; each one represents a list of operations that can be applied to the SRS to read or write registers. The second contains a description of the registers. These files are loaded into memory and parsed as a DOM representation using the Xerces-C++ library included in XDAQ. The SRSController has classes which hide the details concerning the extraction of data contained in the DOM elements. These abstractions will allow to change the parsing library or the XML representation in the future without changing the API.

3.2 Sending sequences to the SRS

These XML sequences aren't directly understood by the hardware, they need to be translated. This is done with the help of the package SRSSlowControl, which converts the sequences to frames that are sent to the SRS and trigger a reply. Then, the SRSController collects the data from the reply frames (the current values of the registers in case of read operations) and updates the sequences file.

3.3 Displaying sequences

The webpage is built with the help of a CGI library provided by XDAQ and displays the sequences in tables. The interface is detailed in 5.

4 Running SRSController

4.1 Configuration

When the program is compiled, a binary object is generated that has to be loaded into XDAQ using the XML configuration file *xml/SRSController.xml*. This file contains seven parameters that should be modified by the user before running. Below there is the XML code with the parameters colored in red:

```
<?xml version='1.0'?>
<xc:Partition xmlns:soapenc="http://schemas.xmlsoap.org/soap/
   encoding/" xmlns:xc="http://xdaq.web.cern.ch/xdaq/xsd/2004/
   XMLConfiguration-30" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance">
<xc:Context url="http://hostname:1972">
<xc:Application class="SRSController::Main" group="gr1" id="40"
   instance="40" network="local">
    <properties xmlns="urn:xdaq-application:Main" xsi:type="soapenc:
       Struct">
        <sequencesPathDefault xsi:type="xsd:string">/path/to/
           OptoRx1_RP210.xml</sequencesPathDefault>
        <systemsPathDefault xsi:type="xsd:string">/path/to/
           SRSRegisters.xml</systemsPathDefault>
        <cssPath xsi:type="xsd:string">/path/TOTEMController/
           SRSController/html/css/style.css</cssPath>
        <monitoringFile xsi:type="xsd:string">/path/to/OptoRxDvCount
           .xml</monitoringFile>
        <outputDirectory xsi:type="xsd:string">/tmp</outputDirectory
           >
    </properties>
</xc:Application>
<xc:Module>/path/TOTEMController/SRSController/lib/linux/x86_64/
   libSRSController.so</xc:Module>
</xc:Context>
</xc:Partition>
```

Code 1: XML configuration

- In *Context*, the user has to specify the host name of the computer running SRSController.
- In *sequencesPathDefault*, the path to the default sequences XML file to be displayed statically. It's used only when a Configure command doesn't specify one.
- In *systemsPathDefault*, the path to the default systems XML file.
- In *cssPath*, the path to the CSS file for the web interface.
- In *monitoringFile*, the path to the sequences XML file to be used for monitoring.
- In *outputDirectory*, the output directory for the updated monitoring file.
- In *Module*, the path to the compiled package.

4.2 Sending commands

The interaction between the user and the application is done mainly via SOAP messages. Internally, the program is implemented with a Finite State Machine (FSM) that has the following states: Halted, Ready, Enabled, Suspended and Failed. To transition from one state to another, the SRSController needs to receive one of the following commands: Configure, Enable, Suspend, Resume, Halt or Reset.

The SRSController starts in Halted state. When it receives a Configure command, it parses the XML files and changes to Ready state. Then, when it receives an Enable command and the application is accessed via a web browser, the information is displayed.

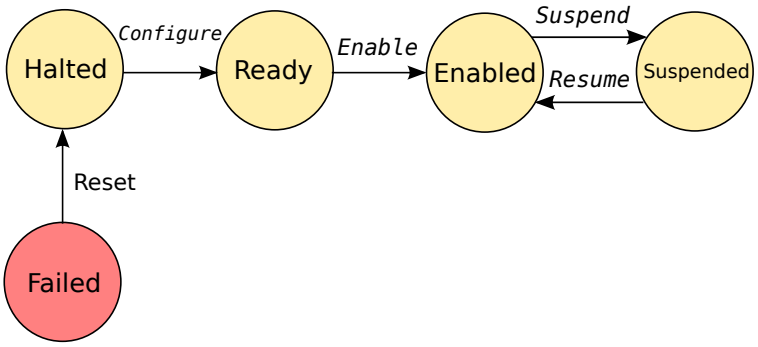

Figure 1: SRSController's states and transitions (commands).

When enabled, the user can send a Suspend command. To go back, the user can send a Resume command. In addition, the Halt command can be called from any yellow state.

If there's a failure during any transition the program changes to Failed state and shows an error message. Most of the failures are due to bad input, like specifying an incorrect path. To get more information about an error the user should read the log, which also keeps track of the state transitions and packets received from the SRS. When in Failed state, SRSController has to receive a Reset command to go back to Halted state.

The SOAP messages have the following format (in red the properties the user should modify):

```
<SOAP-ENV:Envelope SOAP-ENV:encodingStyle="http://schemas.xmlsoap.
   org/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/
   soap/envelope/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP
   -ENC="http://schemas.xmlsoap.org/soap/encoding/">
    <SOAP-ENV:Header>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <xdaq:Configure xmlns:xdaq="urn:xdaq-soap:3.0" >
                <xdaq:SequencesPath>/path/to/OptoRx1_RP220.xml</xdaq
                   :SequencesPath>
            <xdaq:SystemsPath>/path/to/SRSRegisters.xml</xdaq:
               SystemsPath>
        </xdaq:Configure>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Code 2: SOAP message containing a Configure command

These have to be sent to:

```
http://hostname:1972/urn:xdaq-application:lid=40
```

Code 3: SRSController URL

The only thing that the user has to change is the command. The Configure command may optionally contain a SequencesPath tag specifying the path to the XML file containing sequences to display statically. It may also contain a SystemsPath tag specifying the path to the XML file containing systems and registers. If any of these aren't specified, the program will use the default paths specified in the XDAQ XML configuration file, as explained in 4.1.

5 Web interface

The web interface has been designed to be simple and functional so the interesting data can be comfortably read. It can be accessed directly through the URL specified in Code 3.

It is recommended to use the CSS file that comes with the package because the tabs are implemented with HTML and CSS (no Javascript), although the user can change the other parameters as he wants. Javascript is only used to get the current time.

The web always shows the program state in the header. In Enabled state, the sequences are shown. In Failed state, an error message is displayed.

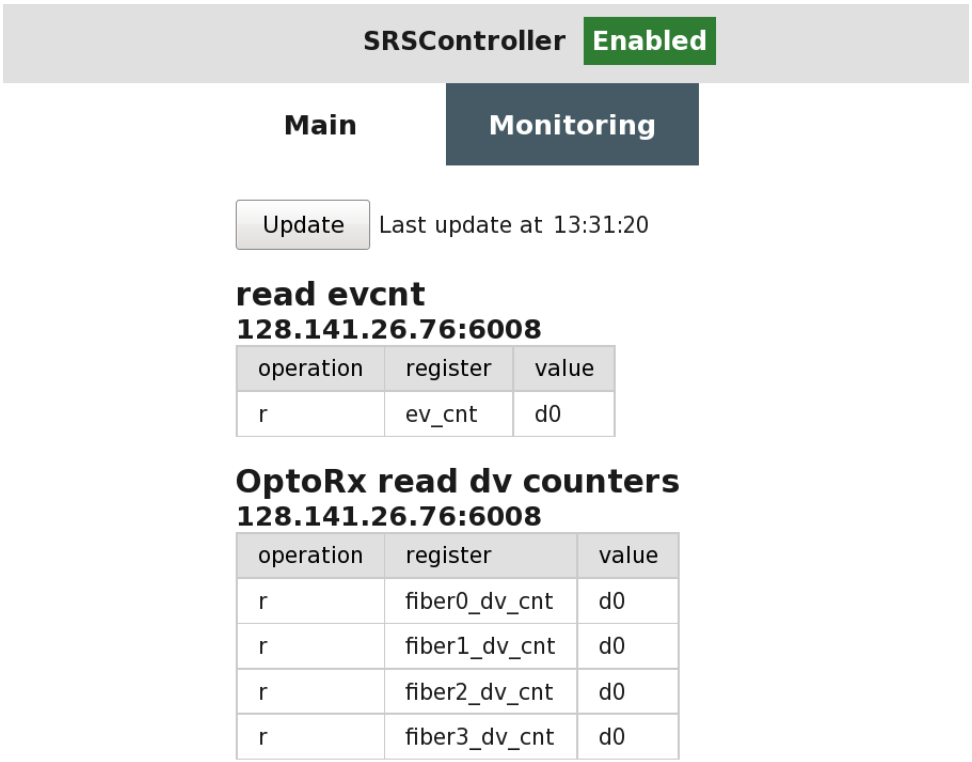

Figure 2: Web display in Enabled state.

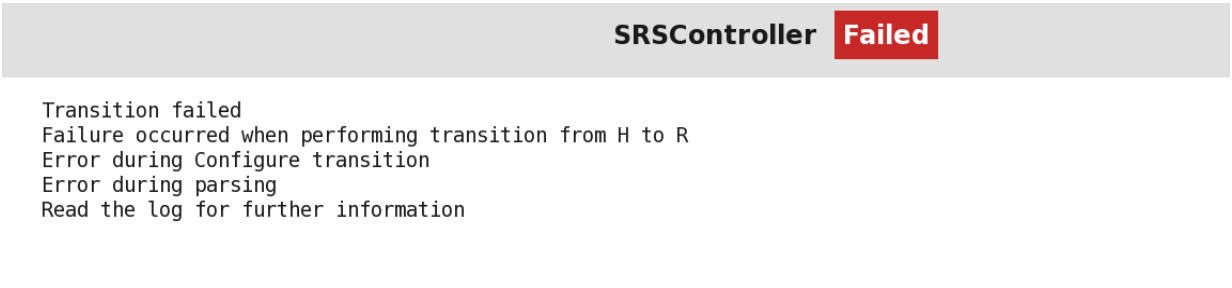

Figure 3: Web display in Failed state.

There are only two possible interactions between the user and the interface, and both appear in the Enabled state. The first feature is switching between the Main tab, which shows the main sequences file, and the Monitoring tab, which allows to monitor the values according to the sequences file loaded. The second feature is found inside the Monitoring tab and is the button to update the values. In addition, the last time updated is shown.

6 Documentation

The project is documented using Doxygen so that it can be read through a web browser. The classes and their member functions are documented in the header files. The implementation also contains comments to make the job easier to future maintainers and developers.

7 Conclusion

The development of the application has been successfully completed. I would like to thank my supervisors and the summer student team for offering me this project and for the help provided.

References

[1] TOTEM experiment webpage
`http://totem-experiment.web.cern.ch/totem-experiment/detectors/`

[2] Michele Quinto. *Design Development and Characterization of 2nd Level Trigger System for Very Forward Detector at LHC*. PhD thesis, 2014.
`https://cds.cern.ch/record/2008211?ln=en`

[3] XDAQ project webpage
`https://svnweb.cern.ch/trac/cmsos`