PVSS Statistics

CERN Summer Student Report 2022

Prepared by:

Christian Maalouly (El)

Supervised by:

Wassef Karimeh

Table of Contents

1. Background Information

The Detector Control System (DCS) is in charge of ensuring that the CERN experiments are safely operable without interruption so that high-quality physics data is always available to be collected by the data acquisition systems.
At CMS, control systems are distributed, and every sub-detector has its own DCS. A central DCS system takes care of monitoring and managing the sub-systems DCS.
The distributed control system of the CMS experiment meets challenging requirements that reflect the scale and complexity of the detectors. Some of its hardware, namely the PLC and power supplies, operate in a hostile environment with radiation and a high magnetic field. In addition, many locations inside the detector are inaccessible without moving sections of the detector weighing several tons.

Simatic WinCC Open Architecture, originally known as PVSS II, is a commercial Supervisory Control and Data Acquisition (SCADA) software that was chosen as the detector controls development package at CERN in 1999. It is intended to make it simple for clients and developers to deploy customized industrial control systems. The system is efficient for large and complex applications and projects.
The engineering controls CERN group together with the LHC experiments established the Joint Controls Project (JCOP), which is in charge of identifying common needs across experiments and providing common solutions for them. The JCOP framework, which is the product of this research, is a set of tools and templates written in PVSS to simplify the effort of connecting and establishing the PVSS infrastructure required to control and monitor the hardware usually utilized by the experiments.

The day-to-day operations of the Large Hadron Collider (LHC) require a large number of highly specialized control systems. WinCC OA is used for example to control the extensive LHC cryogenic system, which cools super-conducting electro-magnets down to a temperature below that of outer space and operates over the 27-kilometre ring of the LHC.

2. Project Introduction

At CMS, the central DCS team is in charge of over fifty DCS systems and billions of real-time readouts. One method of optimizing the system is to examine the existing structures described in the WinCC-OA projects and report back to specialists who may be able to minimize or improve the structures.
WinCC-OA allows performing an ASCII dump to get the data of a running project into a .dpl file, this file represents a structure that reflects the WinCC OA project. However, the .dpl file is difficult

to read and analyze, which is why this project focuses on developing a dpl file parser to extract meaningful information from the offline files and write it into a database. Some basic statistics will also be conducted and preserved on a regular basis, and users can extract data from the database and run further required statistics if necessary.

3. Work Conducted

3.1. Solution Overview

The project is programmed using C++ for efficiency as it will be working with very large amounts of data. It is composed of multiple header files:

- parser.h: the parser code, users can specify a block inside a .dpl file they want parsed and it will return a matrix with the parsed data.
- dptBuilder.h: a builder for datapoint types and their elements, given a matrix of the datapoint type block in a .dpl file, it will build the datapoint types based on a specific model, each consisting of a name and a datapoint element list which contains the element name, type, and line number.
- configBuilder.h: a builder for the configs, given a matrix of a config block in a .dpl file, it will build the configs following a specific model consisting of a reference to the config table's metadata, the element name, the type name, and the value of its attributes.

There is also the main.cpp file, which is a sort of example of a user program that parses a specific file and builds the datapoint types and configs of the file, by changing the file and the blocks of data specified to match the file's blocks, it can perform the desired task for any .dpl file.
Another code that stores the data in the given database is still being worked on due to some problems encountered that will be elaborated on later.

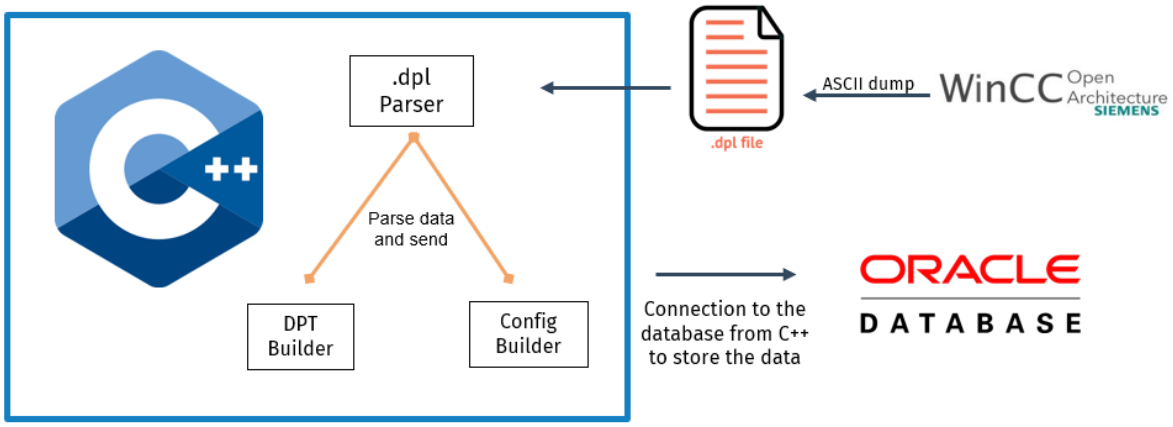

Figure 1: Solution Schema

3.2. Project Development

3.2.1. DPL Parser

The dpl parser code includes a parser class that allows users to pick a dpl file to work with using a setFile() method that takes the path of the file as a parameter. The method additionally locates the different blocks in the file so that they may be quickly accessed when needed; if the new file matches the previously specified file, nothing is done (this could be changed in the future if needed in case a file can be updated and keep the same name).
A parseDPLfile() method takes a string matrix and a table name as parameters. The matrix stores the data, and the table name determines which table is to be parsed. A method to parse all accessible tables might be introduced, however, some tables are not necessary and can be fairly large, thus specifying the required tables from the main code is more efficient.

Every line in the .dpl file is a row, with each column separated by a tab. A column can span multiple lines, which can be detected by looking for an odd number of quotation marks (") indicating a value that began but did not end. A column with several lines continues as long as there is an odd number of quotation marks. When we detect an odd number, we close the column and the rest of the line should contain the remaining columns if any are left (In the case of multiple multi-line columns, it will keep working since the uneven number from the current and the uneven from the new column would make an even number).

3.2.2. DPT Builder and Model

The dpt builder provides the user with a dpe (datapoint element) structure containing the name, type, and line number of an element, a DPTModel class (datapoint type) consisting of a name and a list of the dpe of that dpt, and lastly a DPTBuilder class that the user will mainly be working with.

The DPTBuilder provides the user with a buildDPT() method that takes a string matrix as a parameter which contains the parsed Datapoint Type table to work with. This method will build a list of DPTModel, meaning a list of the datapoint types. It uses the number of tabs preceding each line to figure out the path of each dpe, and to figure out whether it is reading a dpe or a dpt. If it finds an empty path, it adds it as a dpe and indicates on the console that there was an empty path at the given line. If a dpe was a reference, it adds the referred dpt's name to the element's name separated by a ":".

3.2.3. Config Builder and Model

The config builder provides the user with a configTableMeta struct that contains the config type and its attributes, a ConfigModel class that references its configTableMeta and contains an element name, a type name, and the list of attribute values (the same attributes in the references table meta), and a ConfigBuilder class for the users to work with mainly.

The ConfigBuilder provides the user with a buildConfig() method that takes a string matrix and a table name as parameters, the matrix containing the table's parsed data to work with. To differentiate elements with multiple configurations, certain tables feature an additional "DetailNr" field. Because this column is of little importance to us, the procedure simply ignores it when it is present; however, this may be altered to preserve it later if required. When finished, the method returns a list of configurations (list of ConfigModel).

3.3. Oracle Database, Work-In-Progress

The data built is intended to be saved in an Oracle database, after which statistics may be computed. However, certain issues with the database connection from C++ were encountered, resulting in this section not being completed yet.
The database, on the other hand, was designed and approved by the supervisor, and the majority of the SQL statements necessary for it were produced. The main issue remains the connection from C++, after which the rest of the work would be straightforward.
The configuration tables are static; nevertheless, they will be dynamically built from C++ to accommodate any modifications or differences that may occur for any configuration.

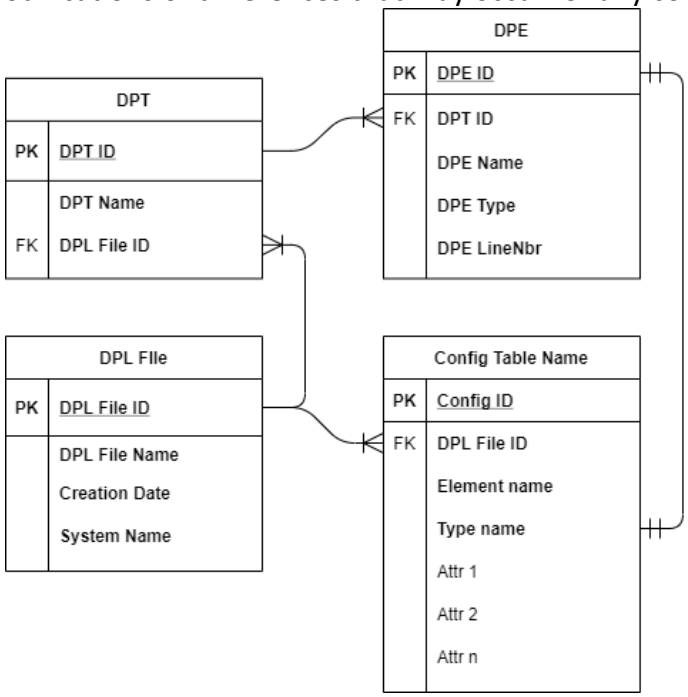

Figure 2: Database Schema

4. Conclusion

Overall, it was a great experience and a challenging project to work on. The work entailed studying the structure of the .dpl file to identify patterns and sequences that the code could detect and work with. Furthermore, the code had to be efficient while still being clear so that it could be modified and improved as needed. Furthermore, some problems encountered pushed for changes and alternative solutions, which in most cases significantly improved performance.

In the end, a great deal of knowledge and experience was acquired, as well as the satisfaction of completing a meaningful project.

When completed, it will give valuable data that will aid in the improvement of CMS's DCS.

5. References

[1] "iopscience.iop.org," 13 December 2012. [Online]. Available: https://iopscience.iop.org/article/10.1088/1742-6596/396/1/012023/pdf.

[2] A. Rassat, "CERN and Siemens subsidiary ETM sign knowledge transfer agreement | Knowledge Transfer," CERN, 5 October 2016. [Online]. Available: https://kt.cern/article/cern-and-siemens-subsidiary-etm-sign-knowledge-transfer-agreement.

[3] "accelconf.web.cern.ch," 14 October 2005. [Online]. Available: https://accelconf.web.cern.ch/ica05/proceedings/pdf/P1_062.pdf.