Automation Tools in the Software Development of the TOTEM Detector Control System

Ivan Atannasov, Fernando Lucas Rodriguez, Member, IEEE, Paolo Palazzi, Federico Ravotti, Member, IEEE, Sami Stöckell, Janos Sziklai, Member, IEEE and Ville Tulimaki

Abstract—
The TOTEM experiment at the CERN LHC consists of three different subdetectors. Its Detector Control System (DCS) is built with the industry-supplied PVSS system and extended with components developed at CERN through the Joint Controls Project (JCOP). In view of minimizing the development effort and improving reliability, the TOTEM DCS team has developed various technologies to automate the software development process to a large extent.

A unified automatic development for all the three subdetectors of the experiment saves time, produces more reliable software, and makes maintenance less expensive. The automation process described here may be reused in other industrial automation applications.

Index Terms---Detector Control System, TOTEM, CERN, LHC, Finite State Machine, automation, heuristics, naming scheme.

I. Introduction

The TOTEM (TOTal and Elastic Measurements) experiment at CERN [1] [2] measures the total cross section of the proton and also monitors accurately the LHC luminosity [3]. To do this, TOTEM is able to detect particles very close to the LHC beams, after their collision in the interaction point.

TOTEM consists of three subdetectors. They are the “Roman Pot Stations” (RP), the “Cathode Strip Chambers” (CSC) Telescope 1 (T1) and the “Gas Electron Multipliers” (GEM) Telescope 2 (T2). The T1 and T2 detectors are located on each side of the CMS interaction point in the very forward region, but still within the CMS cavern. Two Roman Pot stations are located on each side of the interaction point, at 220 m and 147 m, inside the LHC tunnel. Each Roman Pot station consists of two groups of three Roman Pots separated by a few meters, as shown in Figure 1.

TOTEM is in its learning phase which will produce elaborated requirements for the Control System. As a first step, the inputs and outputs of the controlled system (the experiment) and the relation among them have been established [4] [5]. At a later step it will be evaluated under what exact circumstances actions have to take place.

II. Basic concepts and formalizations for the TOTEM DCS

A. Product Breakdown Structure of the Detector

A Product Breakdown Structure (PBS) is a hierarchical decomposition of a system. It is structured using nested levels that forms the main system. This decomposition can be applied to each detector or to the DCS itself resulting different PBS trees.

As an example, the TOTEM Roman Pot System develops on a hierarchical structure of eight levels as shown in Figure 2 which expands from the whole roman pot system (“system”) to its ultimate granularity (“strip”).

The mirror symmetry with respect to the CMS interaction point is used to define the names at the different levels. Each side follows the LHC sectors naming scheme (“sector 45” and “sector 56”). The distance from the interaction point (CMS) identifies the stations and the units (“station 147”, “station 220” and “unit near”, “unit far”). The pot name is derived from its position relative to the beam axis (“top pot”, “horizontal pot”…).

B. Naming scheme of the detector

A clear naming scheme is of prime importance in a system where many autonomous subsystems are integrated among them. Many sensors are installed inside each one, and every sensor needs its unique name. If this scheme were not existing, each group would use different conventions,
names or ordering. The resulting system would be very
difficult to develop and maintain.

Figure 2 is an extract of the requirements needed for
building the DCS of the TOTEM Roman Pots [6].

The naming of each piece of equipment of the Roman
Pot detector system is built by concatenating the naming
tags of its hierarchy in the PBS, following the order given
by the arrow on the left of Figure 2 and abbreviating where
possible to two letters (the first two consonants).

For example, the 4th VFAT (that is one of the readout
elements of the front-end electronics) in the 2nd Hybrid of
the top Pot in the far Unit of the Station at 147m of the
sector 45 is named as rp_45_147_fr_tp_02_004.

Following this set of examples it is possible to build a
Backus-Naur Form (BNF) grammar for the nomenclature
[6]. Having this grammar allows to validate the names
used in the software developments, and define algorithms
that only apply to specific PBS items, and this opens the
possibility for using heuristics based on the names.

C. Product Breakdown Structure of the DCS

In the same way that exists a PBS for the detector exists
another one for the DCS itself. It is based on decomposing
the system by functionality: High Voltage (Hv), Low Volt-
age (Lv), Environmental sensors, Frond-End Electronics,
Cooling plant,…

Each functionality is controlled by independent pieces
of PVSS software [7]. The behaviour formalization for
monitoring and execution of the DCS commands uses Finite
State Machines (FSM).

Such decomposition is represented as graphical diagrams
as shown in Figure 3 [8]. These diagrams were inspired by
the ALICE DCS [9].

Further details about the DCS structure are given in [10].

D. PVSS datapoints

PVSS is structured around the concept of “datapoints”.
The value of the sensors is stored inside datapoints and the
commands to the actuators are sent by writing new values
into them.

Each datapoint name has to follow the JCOP [11] conven-
tions, so the JCOP framework can work properly. However
this name is related to the physical hardware connectivity.
For this reason it is usually referred as “hardware name”.

However each datapoint can also have an alias, and it is
usually referred as “logic name”. The alias of the datapoint
matches the naming schema defined in Section II-B. The
alias is later used for building the FSM, instead of directly
using the datapoint name. This allows changes in the
hardware connectivity without affecting the FSM logic.

E. Pinout formalization in Excel tables

The assignment of each wire from an ADC channel
or a voltage channel in the detector is called pinout. It
is necessary to map all the connectivity across all the
patch panels and connectors from the detector up to the
corresponding equipment in the experimental areas.

This information is structured as MS Excel pinout tables
in [12] according to the levels of the PBS of the detector.
Those tables are considered to be a “source” for the config-
uration of the DCS. The TOTEM DCS team keeps track of
the changes storing those files in Subversion [13], but they
could also be converted into a web interface and logging
the modifications.

Much of the innovation resides on this formalization. Not
only the basic cabling information, but all the connectivity
up to the DCS system inside the counting room are spec-
ified. The responsible for each part of the cabling chain
is also identified. This helps defining responsibilities and
allow the different groups to fill the MS Excel table by
their own following the given template. An example of
the pinout table of the environmental sensors for the RP
is given in Figure 4.

III. Life cycle of the TOTEM DCS

A control system development is an iterative process
[14]. Each new iteration of the development cycle can have
a huge impact on the initial requirements. Each software
release helps validating the initial requirements and as-
sumptions, and clarifies the next development cycles.
<table>
<thead>
<tr>
<th>Part name</th>
<th>Items</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product breakdown structure</td>
<td>sector, station, unit, pot, function</td>
<td>Identify univocally pots, units and stations</td>
</tr>
<tr>
<td>Motherboard pinout</td>
<td>RPMB pad, RPMB signal name, RPMB conn. pin</td>
<td>Signals and voltages between the motherboard and external cables</td>
</tr>
<tr>
<td>Short cables</td>
<td>19 pins conn. pot, short cable name, short cable wire, 19 pins conn. patch panel</td>
<td>Connect the motherboards into the station patch panel</td>
</tr>
<tr>
<td>Tunnel patch panel</td>
<td>19 pins conn. patch panel, patch panel wire, 48 pins conn. long cable</td>
<td>Connect all short cables of a station into the long cables</td>
</tr>
<tr>
<td>Long cables</td>
<td>48 pins conn. tunnel, long cable name, long cable wire, 48 pins conn. counting room</td>
<td>Signals and voltages from the tunnel into the counting room and alcoves</td>
</tr>
<tr>
<td>Patching into crates/climbers and control devices</td>
<td>48 pins conn. flat cable ID, flat cable pos</td>
<td>Discrete wiring into the counting room equipment</td>
</tr>
<tr>
<td>Configuration of hardware control devices</td>
<td>ELMB bus, ELMB ID (decimal), ELMB channel, ELMB comm. pin, ELMB channel polarity</td>
<td>Position of the boards into the crates, buses, IP addresses</td>
</tr>
<tr>
<td>DCS hardware and logic names</td>
<td>DCS hardware name, DCS logic name, PVSS datapoint names and aliases</td>
<td>PVSS datapoint names and aliases</td>
</tr>
</tbody>
</table>

Figure 4. Structure of a RP pinout table

Figure 5 shows the development sequence for developing new functionalities. There are four different kinds of blocks:

- **Green blocks**
  The requirements and the physical construction of the detector. Naming scheme, pinout tables, commissioning results (after development iterations),...

- **Blue blocks**
  Engineering formalization of all the requirements in a way that they can be processed automatically. However, they do not attempt to be a 100% formalization of the requirements.

- **Red blocks**
  PVSS developments; datapoints, datapoint types, FSM types, scripts, panels, ...

The automation tools described in this paper help in the steps between the blue and red blocks. Figure 6 shows more detail the steps where the tools and scripts are used.

**IV. Pinout and Finite State Machine preprocessor**

A specific tool has been developed in C# for the construction of the “hardware names” and “logic names” of the DCS datapoints. It uses the pinout information of the detector and some heuristics. It is also able to build a Finite State Machine of the detector.

By inspecting the final structure of the system it can do some estimations about the information exchanged the different levels and about the response time of the system.

This preprocessor is highly modular, and the output generation has eight steps:

1. Parse and clean the MS Excel tables.
2. Add additional columns in the tables with the heuristics of Section V.
3. Export into CSV the expanded pinout tables.
4. Validate the logic names using a BNF grammar.
5. Generate the FSM hierarchy by removing suffixes from the logic name.
6. Add extra attributes in the FSM hierarchy such as PBS, FSM type, ... with another set of heuristics.
7. Export into CSV the FSM hierarchy.
8. Execute correspondent algorithm for the information exchange calculations.

**V. Heuristics of the preprocessor**

The preprocessor of Section IV is generic enough to adapt to other possible experiments by defining a new set of heuristics. The heuristics are formalized using XML, as in Figure 8.

There are two sets of rules for the heuristics:

- **For the pinout**
  They generate the “hardware name” and the “logic name”. It also adds additional attributes as the PBS or FSM type.

- **For the FSM**
  When the intermediary nodes in the FSM tree are generated as explained in Section VII, they do not contain any PBS or FSM type information. For this reason dedicated rules are needed for them.
VI. DATAPoints GENERATION FOR THE PINOUT

The pinout tables described in Section II-E are exported with the preprocessor of Section IV into the CSV format. They include now the columns for the hardware and logic names, and they are processed by a custom made PVSS script.

This script checks the hardware and logic names; if they do not exist or they differ from what is stated in the table, they are created or properly updated. Also all the crates, CAN [16] buses, ELMB [17] and any other control function from Section II-C are set up automatically inside the PVSS project to match the table.

Those scripts compare the current situation with the new desired configuration, and informs about all the performed changes while they are taking place. This is a valuable feature for maintenance, by checking the logs of the system the developer can have confidence that the upgrade (new configuration) went well, and did not do anything more or anything less than expected.

VII. DATAPoints GENERATION FOR THE Finite State MACHINES TREE

The hardware related datapoints were previously generated by the script of Section VI. The preprocessor of Section IV also generates another CSV file representing the FSM hierarchy. This file table is processed by another PVSS script to generate the final datapoints.

Figure 9 is an extract of a CSV file defining the FSM tree (or hierarchy) of a detector.

This table has three zones:

1) **Hierarchy level**
   Here the name of the FSM node is written. Implicitly the column position (the number of empty cell before the first one filled) indicates the nesting in the hierarchy.

2) **Type of FSM node**
   They are defined in SMI [18]. This part determines if it is a Control Unit (CU), Logic Unit (LU) or a Device Unit (DU). It also specifies the PVSS datapoint type associated to the FSM type.

3) **DCS Product Breakdown Structure entry**
   It is related to the type of hardware under control. See Section II-C.

The FSM tree of Figure 9 is generated from the pinout information by taking the logic names and removing the suffixes. This will lead to reconstruct the hierarchy of Section II-A.

The preprocessor of Section IV also does this.

VIII. Finite State Machines Types

Figure 10 is an UML state diagram [19] representing the agreed FSM logic.

Those UML diagrams are developed with the tool “Visual Paradigm for UML” [20]. In a second step they are converted into XMI [21] (based on XML) with the same tool. Later the XMI file is parsed within PVSS, and it
generates inside PVSS the FSM types as defined by the JCOP framework.

The transition arrows are of two types:

- **Gray, thick and labelled**
  They represent the commands of the FSM type. The label is the “command” name and the transition is triggered manually by the operator or by FSM internal logic.

- **Black, thin and unlabelled**
  They represent autonomous transitions in the system. They take place without the DCS intervention as response of changes in the internal status of the hardware.

**Figure 9. Extract of the FSM hierarchy table**

A special BUSY (not shown in the UML diagram) state is introduced when converting from UML into PVSS. Its main purpose is that the operator notices that there is a transition in progress or there is not a “stable” situation. It allows to isolate the calculation of the current state from the propagation of commands. After every command, the FSM goes to the BUSY state. Being in that state, when the hardware reaches a “stable” situation, the new state for the FSM node is calculated. Then the FSM goes to the calculated state and this is propagated to the upper levels in the hierarchy.

For example, when sending a command OFF the FSM will execute the corresponding logic. But until the hardware reacts, the FSM will show the BUSY state, but neither ON nor OFF. The last command sent to a DU is also shown in the User Interface.

The FSM types names have to follow the JCOP naming scheme, that have a different philosophy from the one of Section II-B. However, they follow the TOTEM naming as much as possible.

Each state of the FSM type is colored according to the JCOP agreed colors; They are mapped from UML into JCOP framework agreed colors using a color proximity algorithm. Also each FSM type has an associated user panel to resume the status of the children nodes in a convenient way for the operators. Also the basic JCOP panels for direct hardware operation can be reused for the DU and fixed before the official releases takes place.

**Figure 10. UML state model for FwWienerMarathonChannelTot FSM type**

In PVSS/SMI the logic is divided in two parts.

- **Current state calculations**
  For the CU and LU it is done by SMI, based on the state of all the children.
  For the DU it is done within PVSS.

- **Execution of commands**
  For the CU and LU it is done by SMI, by propagating and translating the proper commands to each one of the children.
  For the DU it is done within PVSS.

Only the SMI code for CU and LU is automated from the UML diagrams. It is generated from PVSS by using the specific JCOP API. The DU logic is hardware dependent and cannot be automated.

IX. Conclusions

The control system must have a development methodology flexible enough to provide a new release of the system a few days after the new requirements have been defined. It must be also a well defined procedure, so the changes in the code can be traced back, and automated as much as possible to avoid human mistakes.

Those automation tools and scripts make the DCS developments and maintenance much faster than usual, but even more important is the confidence of what is agreed in the table is really implemented in the final system. The detector experts that provide the requirements can inspect the human readable representations of the pinout tables and UML diagrams.

The automation scripts written in PVSS for the PVSS datapoints generation, uses the JCOP API. If at some moment we would like to use another SCADA system, those scripts should be ported, but the preprocessor, and all the definitions agreed with the experts would remain the same.

This is the great advantage of having an independent preprocessor and automation tools. The system can be ported to another platform just porting the final step of the process.

Further improvements in the preprocessor could include automatic detection of polarity mismatches, wires not plugged, wires plugged twice, ... Also it could evolve into a webservice and integrate in the webpage for the pinout.
ACKNOWLEDGMENT

We would like to give thanks to the TOTEM collaboration colleagues. They have provided the information needed of how to decompose the system in the different levels and the behavior implemented in the FSM.

REFERENCES

[17] J. Cook and G. Thomas, ELMB128 documentation: Everything you wanted to know about the ELMB128 but were afraid to ask, EDMS 684947 rev 4.3, February 2005.