The Software for the CERN Detector Safety System

G. Morpurgo, R. B. Flockhart and S. Lüders
CERN IT/CO, Geneva, Switzerland

Abstract

The CERN Detector Safety System (DSS) has been developed to monitor the state of the LHC detectors and protect their components against emergency situations which other less reliable systems could possibly fail to handle, or which were not covered by any other system. The DSS consists of two main components: a PLC-based Front-End and a supervisor Back-End. The two parts communicate via the OPC mechanism.

The Front-End is the core of the DSS, to which all the runtime safety-critical actions are delegated. These are encoded in an "Alarm-Action Matrix", that can be seen as a set of user-defined rules relating the Input Channels (analogue and digital sensors) to the Output Channels (actuators, typically on/off interlock switches). One of the user requirements is the possibility of modifying these rules at any time, without stopping the system. New Input and Output Channels can also be incrementally added to the system without perturbing the previously existing part. To fulfil these requirements without detriment to the safety-critical part, the PLC software has been designed in a completely data-driven way. This approach allowed the PLC code to be kept as simple as possible and independent from the user-defined rules. The rules, as well as the parameterization for the Input and Output Channels, are stored into PLC "datablocks", whose contents drive the execution of the PLC code.

The supervisor Back-End, based on the CERN-chosen PVSS SCADA system, represents both the “Configuration Interface” and the “Operator Display” for the DSS. The User is able to define and parameterize Input and Output Channels and to modify the set of rules defining the Alarm-Action Matrix. These modifications, after passing some consistency checks, are accepted by the Back-End, permanently stored in PVSS "datapoints", and finally mirrored into the PLC datablocks via the OPC mechanism. In addition to this, the Back-End is constantly informed, again through OPC, of any changes in the status of the monitored system. The Back-End then displays all the relevant information to the User, both via an alarm screen and via an interactive synoptic system. This paper describes in greater details some of the ideas behind the design and the implementation of this system.

Introduction

The CERN Detector Safety System (DSS) was developed to protect reliably the four LHC experiments against potentially harmful situations which either escaped the control of other less reliable systems (like the Detector Control System), or were not covered by any other system. The protection offered by the DSS is encoded in the so-called "Alarm Action Matrix" (AAM), which specifies the Alarm Conditions (logical expressions depending on the values of the Sensors monitored by the DSS), and, for each Alarm, the protective Actions to be executed (typically, setting some Actuator to switch off the electrical power in some part of the experiment). The Alarm-Action Matrix is periodically evaluated by the PLC-based DSS Front-End, so that, should an Alarm Condition occur, it will be detected within seconds, and the related Actions will be executed.

The DSS system has to fulfil several requirements: one of them in particular had a significant impact on the way the system has been designed: "At any time the User can add new Sensors, Actuators or Alarm Conditions, or modify their parameters, without stopping the system". This required flexibility had to be combined with high reliability, and with the fact that at least one DSS system for each of the four LHC experiments had to be built and maintained over twenty years. The simplest way to match these requirements was to design the DSS to be a completely data-driven system. The only things that logically differentiate two DSS systems are the definitions of the Sensors, Alarm Conditions and Actuators. In this data-driven approach, the DSS software does not depend on the data peculiarities of the different DSS systems; the same identical software runs on all DSS systems. The overall reliability and availability of the system benefits a lot from this approach; in fact the Front-End software, which is responsible for the evaluation of the AAM and ultimately for the safety of the detectors, does not need to be modified when the AAM changes. This eliminates the possibility of introducing software bugs with any modifications.

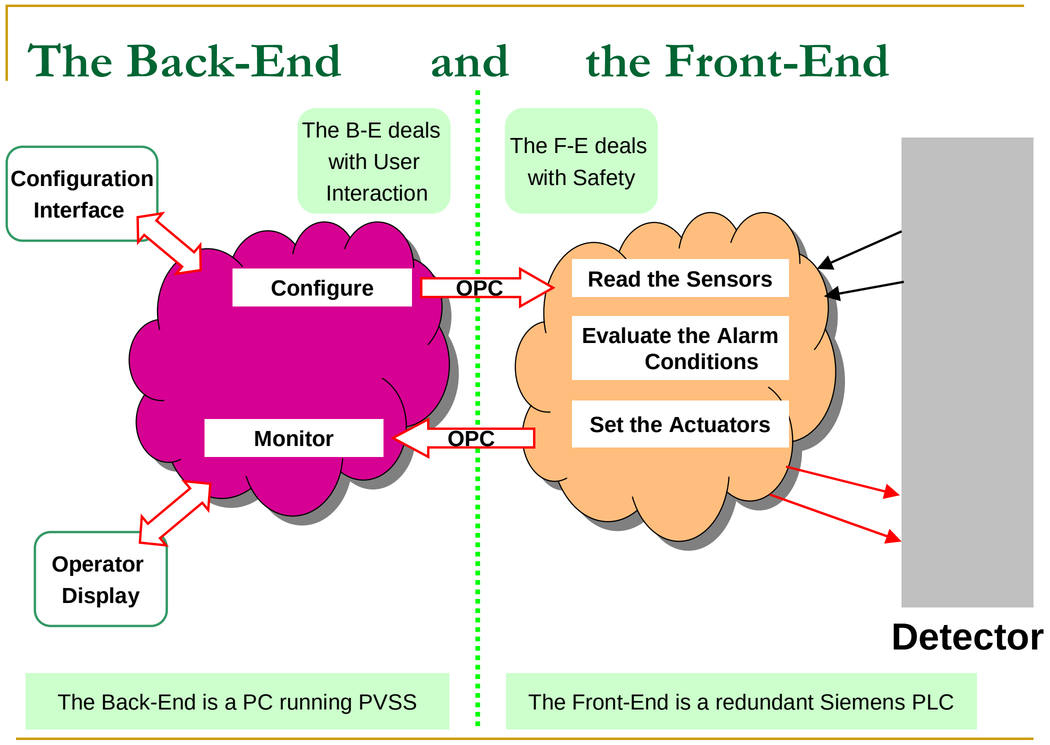
The Back-End and the Front-End
The B-E deals with User Interaction
The F-E deals with Safety
Configuration Interface
Configure
OPC
Read the Sensors
Evaluate the Alarm Conditions
Set the Actuators
Monitor
OPC
Operator Display
Detector
The Back-End is a PC running PVSS
The Front-End is a redundant Siemens PLC

Figure 1: The overall logical structure of the DSS.

Structure of the Front-End Alarm-Action Matrix Processing Software

The AAM processing software can be resumed by the following statement: "read the values from the Sensors, evaluate the Alarm Conditions, execute the Actions linked to those Alarms which became true". This software is organized as a sequence of several steps which process the data contained into seven PLC memory "datablocks". These datablocks contain all the necessary parameter and status information needed to run the system. The datablocks have room to contain a predefined number of entries. Each entry occupies a well defined position in the memory datablock, and can be addressed by the different processing steps via its Unique Channel Identifier (UCI), a number between 1 and 32767.

The first two datablocks contain parameters and status for digital and analogue sensors respectively. The sizes of each datablock are predefined (for instance, the first datablock can contain data for 2048 digital sensors, whereas the second one has space for 1024 analogue sensors). The position of a given sensor in the corresponding datablock is determined by its hardware address (see [1]). The datablock entries that do not correspond to an existing sensor are marked as "inhibited" by setting a bit in the parameter area of that entry. The PLC program that processes the analogue sensors loops on all the entries of the corresponding datablock. The inhibited entries are skipped, while the newly acquired values of the existing channels are compared with the "Too Low" and the "Too High" thresholds (contained in the parameter part of the datablock entry), to determine if the sensor is in the normal range, or not. If an abnormality is detected, the PLC program will set a unique bit (corresponding to that sensor and to that abnormal condition) in the status part of the datablock. This bit will be then used by the following steps (evaluation of the Alarm Conditions). The bit will also be mirrored to the Back-End through the OPC mechanism.

The next three datablocks store subcomponents of the Alarm Conditions, and the Alarm Condition themselves. In fact a complex Alarm Condition can be described as a two-level structure, where the first level can refer to up to eight "sub-components", each of which can be either a single sensor condition, or a combination of up to eight single sensor conditions, or the result of a comparison between the value difference of two analogue sensors and a threshold. Each of these sub-components is referred and addressed through its UCI. The Alarm Condition will become TRUE if at least m of its

n subcomponents became TRUE ("m" is specified by the User when he defines the Alarm Condition). The same applies to the second level structures. This generic "m-of-n" structure can be used to implement many commonly used structures, like AND (m=1), OR (m=n), 2-of-3 (m=2, n=3), etc.
Again, each datablock has room for a large number of entries, and the unused entries are signalled by the inhibit bit, but here the position of an entry in its datablock has to be explicitly handled by the Back-End software via an allocation mechanism.

When the step responsible for the evaluation of the Alarm Conditions finds that one of them has become TRUE, the Alarm is triggered. As a consequence, all the Actions linked to the Alarm will be executed, either immediately or after a delay provided as a parameter. In the DSS context, to execute an Action means to set an Actuator (i.e. a Digital Output) to its "safe" (i.e. 0, for positive safety) state. As an Alarm can be linked to several Actions, and, conversely, the same Action can be linked to several Alarms, the sixth datablock is used to describe the Alarm-to-Action links. Each entry used contains the UCI of an Alarm, the UCI of one of the Actions to be executed when the Alarm triggers and the execution delay. Also for this datablock the position of the entries has to be handled by the Back-End software. The seventh datablock, finally, contains parameters and status information for the different Actions (i.e. Actuators). An Action's entry position is determined by the hardware address of the corresponding actuator. The "value" bit for an Action, set by the last AAM processing step, will be then used by the PLC system actually to set the corresponding actuator's physical value.

Graceful Degrading

When evaluating an Alarm Condition, it could happen that one or more of the Sensors involved are in inhibited state. A Sensor can be inhibited by a privileged User if he thinks that the Sensor is not working properly, or directly by the Front-End itself if a serious anomaly is detected. If the Alarm Condition under evaluation involves "n" single-sensor conditions, of which at least "m" must be TRUE for the Alarm to trigger, it is now possible that only "k" (< "m") Sensors are in a working state. In this case, the Front-End automatically evaluates the Alarm Condition like if it was of the "k-of-n" type, ignoring the inhibited Sensors. If all the Sensors are inhibited, the Front-End realizes that the Alarm Condition cannot be evaluated, and informs the Back-End about this fact.

The Communication between the Front-End and the Back-End

The seven Front-End memory datablocks have a common feature: all the parameter information comes from the Back-End, while all the status information is produced by the different processing steps executed on the Front-End. Once the parameters have been mirrored from the Back-End into the memory datablocks, the Front-End would be fully functional even if the Back-End were to stop running (or if the connection were to be lost). On the other end, the Back-End can perform correctly, displaying the updated status of the DSS system and letting the User modify its configuration, only if the connection is established. The mechanism over which this exchange of data is implemented is based on OPC [2] and on the PVSS [3] datapoint elements. Roughly speaking, a PVSS datapoint can be seen as a permanent instance of a C structure, and a datapoint element as a field in the structure. For every parameter that has to be sent from the Back-End to the Front-End and stored in the PLC memory datablocks, there exists a corresponding PVSS datapoint element that stores it in the Back-End. This datapoint element is also given the OPC address of the position in the memory datablock where the parameter has to be stored. In this way, when the parameter is changed on the Back-End, it will be automatically "mirrored" to the appropriate position in the Front-End memory datablock, where it will be taken into account next time it is read by one of the AAM processing steps. Similarly, every status value produced on the Front-End is mirrored into a corresponding datapoint element on the Back-End. In many cases the Back-End needs to react to the modification of the datapoint element value by executing an "event handler": PVSS provides mechanisms to implement this behaviour.

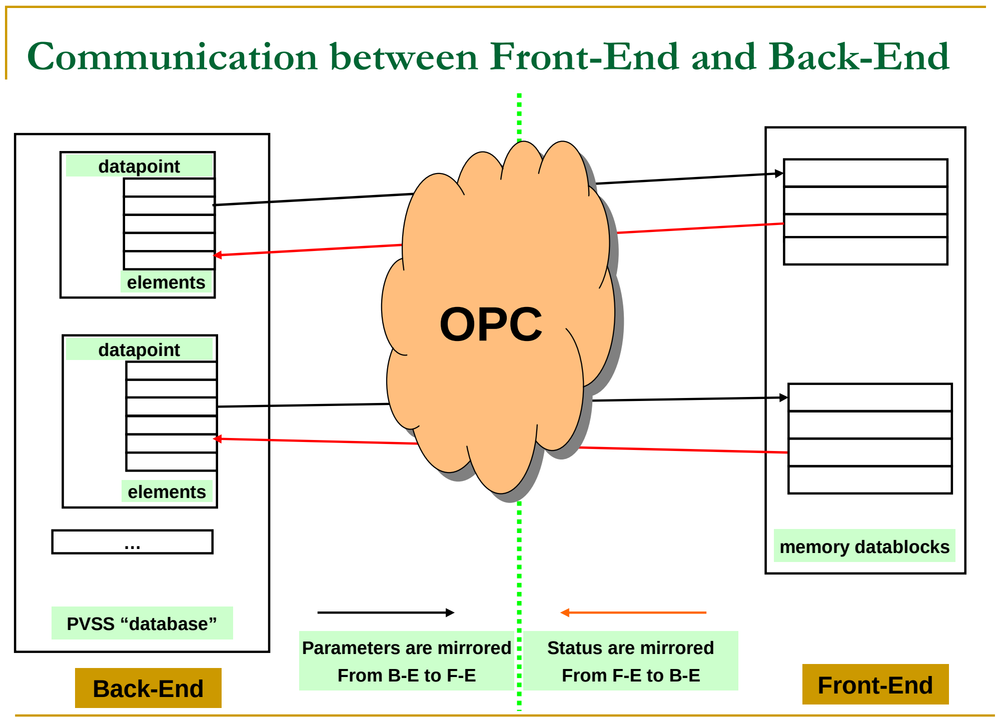
Communication between Front-End and Back-End
datapoint
elements
datapoint
elements
...
PVSS "database"
Back-End
OPC
memory datablocks
Front-End
Parameters are mirrored From B-E to F-E
Status are mirrored From F-E to B-E

Figure 2: How data are mirrored between the Front-End and the Back-End

OPC Optimization

The Front-End produces a lot of detailed information that the Back-End would ideally like to have regularly refreshed. However, throughput limitations in the OPC mechanism between the PLC and the OPC Server set a limit to the amount of data that can be monitored if the actual refresh rate has to be kept decent (~1 sec.). These limitations do not actually depend on how often the data changes, but only on the memory size over which the data that could possibly change is distributed. The reason for this observed behaviour is that the OPC server (to which the Back-End OPC Client is connected) needs to know if the data in the Front-End memory datablocks have changed. Therefore it has to constantly communicate with the Front-End. It appears that the maximum throughput of this communication mechanism is of the order of 22 Kbytes of data per second. Furthermore, data is transmitted in blocks of a few hundreds of bytes covering contiguous areas of the memory datablocks. Therefore it makes a lot of difference if the data to be monitored are contained in contiguous memory areas, or if they are spread over the entire size of the large memory datablocks. The solution to this optimization problem was obtained by restructuring the datablocks such that a) the most critical status information, always to be refreshed, is now contained in contiguous data arrays, and b) the more detailed status information, whose changes are anyway signalled by a change in the above mentioned arrays, are distributed among several "OPC groups", normally passive. Each group would cover about 2 Kbytes of data, and it would be automatically activated only when a change in the permanently refreshed data indicates that some of its elements have changed.

The Back-End Software

The Back-End software has to fulfil two main goals: to configure a DSS, and to display the status (and the evolution) of the monitored system. These two goals implement the two main data flows between the Back-End and the Front-End: configuring the DSS ultimately means filling the parameter part of the Front-End memory datablocks, while displaying the DSS status means translating the status information coming back from the Front-End into a format understandable by the Users. Being entirely written in PVSS, the Back-End software runs around the PVSS database (the ensemble of the PVSS datapoints that constitute a DSS system), and it consists of several interactive panels, and of several PVSS scripts. Among the panels we can distinguish two main classes: the Configuration panels and the Display panels. The scripts are mainly composed by small "event handlers", invoked whenever the value of a datapoint element of a given type changes.

Operator Display (Data from Front-End to Back-End)

The DSS Back-End implements two complementary ways of showing the Operator what is going on. The first way displays the different DSS events on the (almost) standard PVSS Alert-Event panel. This panel is divided into a top and a bottom part, each consisting of a table. The top part shows the different Warnings and Alarms detected by the DSS, while the bottom part shows the Actions taken by the DSS to protect the detector. The implementation of this part is relatively effortless, as PVSS provides tools to generate automatically a new entry for these tables when a specific datapoint element assumes a given value. By looking at the Alert-Event screen the Operator can immediately see all the abnormal situations detected by the DSS.

The second way is more geographically oriented. The entire experiment site is subdivided into a hierarchical tree of “locations”, and every DSS object (Sensor, Alarm, Actuator) is attached to one of these locations. A synoptic panel is used to display a location, together with the status of all the attached objects (represented as traffic lights). The locations descending from the current one are also represented as traffic lights. The colours of a location’s traffic light reflect the presence or absence of Alarm and Warning situations among the objects contained in the entire sub-tree having the location as root node. The Operator can therefore identify easily the areas of the experiment site where problems are detected, and he can navigate through the location tree to reach the appropriate level of detail. More explanations on this can be found in [4].

Configuration Tool (Data from Back-End to Front-End)

Configuring the DSS ultimately means to create and parameterize all the datapoints that constitute the Alarm Action Matrix (Sensors, Alarms, Actuators and their relations). The OPC links to connect the Back-End with the Front-End have also to be created. As already mentioned in the introduction, this configuration activity can be carried on “incrementally” at any time, and new items can be added to the system without perturbing the behaviour of whatever already exists. The (privileged) User is assisted in the creation operation by a series of tab-based “wizard-like” interactive panels (whose implementation required a lot of PVSS arm-twisting), leading him, one page at a time, through the input of the many parameters that must be provided. Whenever possible, no typing is required, and the parameters are selected from lists of possible options or inherited by special datapoints which define behaviours common to many Sensors. After the creation is completed, the underlying PVSS scripts automatically create and fill the datapoint corresponding to the new item. Now the same panels can be reused, this time with direct access to all the tab pages, to look at the detailed status of the new item, or to possibly modify some of its parameters.

Consistency Checks

One point not to be neglected is that if the User inputs incorrect parameters into the system, the DSS would not be able to protect effectively the detector. For instance, if the “Too High” alarm threshold for a temperature sensor was mistakenly set higher than the maximum value that can be measured by the sensor, the sensor would never report a “Too High” condition. Or, if an Alarm Condition required that a sensor was “Too High” AND that the same sensor was “Too Low”, the Alarm Condition would never be triggered. This is why many consistency checks have to be performed before accepting the User’s input, to minimize the error’s likelihood. The most sophisticated checks are performed on the definitions of the Alarm Condition, as the PVSS script has to ensure that, depending on the possible status of the different sensors appearing in the Alarm Condition’s logical expression, the Alarm Condition is neither always TRUE, nor always FALSE. The checks also detect redundant usage of the same sensors in the Alarm Condition definition, which is another sign that the User might not have input what he intended.

Auxiliary Tools

In addition to the main DSS software written to configure, process and monitor the Alarm Action Matrix, various other utilities have been developed, both on the Front-End and on the Back-End. On the Front-End PLC, several tasks perform periodical checks to verify the integrity of the system. The results of the checks are transmitted to the Back-End through the OPC mechanism. On the Back-End itself an “Access Control” utility ensure that the AAM parameterization can only be changed by entitled Users. Several functions help the User to simulate what would happen to the Alarm Conditions if he inhibited or re-enabled a Sensor, or if he modified a threshold value for a Sensor.

One of the potentially most useful utilities is the Oracle logging. The DSS optionally logs in an Oracle database all the configuration operations performed by the User, such as adding, modifying or deleting a Sensor, an Alarm or an Actuator, together with all the DSS events, such as a Sensor crossing a threshold, an Alarm triggering, an Action being taken. This can be useful to correlate configuration modifications with the appearance of problems. It is also possible to “Tag” the entire configuration at the current time and to save it into the database.

The DSS Back-End also includes tools that help the User in building and processing sequences of operation for the parameterization of the DSS Objects. Finally, a complete .html based online help system has been built and integrated with the system.

Conclusions

The basic ideas behind the design and the implementation of the DSS Software have been described and motivated. The two major benefits deriving from the adopted data-driven approach are an enhanced reliability and simplicity in the Front-End code, and a unified development completely decoupled from the particularities of the different LHC experiments. This allowed a very small development team to match a tight schedule, and indeed the four DSS originally foreseen have been installed at the LHC experimental sites and are fully operational. One of the experiments requested a fifth DSS, which is now used to protect electronic racks in an assembling hall. In fact, thanks to the data-driven approach, the DSS can be considered as a very highly reliable “general-purpose” safety system, and it could be effortlessly reused “as it is” in many other different environments, including in industry.

Acknowledgements

We would like to thanks the Siemens support team (and particularly R. Lafay and T. Deutschmann) for their continuous help concerning the PLC programming and OPC optimization, and J. Auer, previously with ETM, for the many discussions and good advice on how to do things with PVSS.

References

1) S. Lüders et al., “The CERN Detector Safety System for the LHC Experiments”, ICALEPCS’2003, Gyeongiu, Korea, October 2003.
2) Open Connectivity via Open Standards. See `www.opcfoundation.org` .
3) PVSS II (Prozessvisualisierungs- und Steuerungssystem) is a trademark of ETM.
4) G. Morpurgo, R. B. Flockhart, S. Lüders, “The DSS Synoptic Facility”, these proceedings.