OPERATIONAL STATUS DISPLAY AND AUTOMATION TOOLS FOR FERMI@Elettra[*]

Claudio Scafuri, Sincrotrone Trieste, Trieste, Italy

Abstract

Detecting and locating faults and malfunctions of an accelerator is a difficult and time consuming task. The situation is even more difficult during the commissioning phase of a new accelerator, when the plants are not yet well known. Faults involving single devices are easy to detect, however a fault free machine does not imply that it is ready to run: the definition of malfunction depends on what is the expected behavior of the plant. In the case of FERMI@Elettra, in which the electron beam goes to different branches of the machine depending on the programmed activity, the configuration of the plant determines the rules for detecting malfunctions. In order to help the detection of faults and malfunctions and to display the status of the plant, a tool , known as the ”Matrix”, has been developed. It is composed by a graphical front-end which displays a synthetic view of the plant status grouped by subsystem and location along the accelerator, and by a back-end calculation engine. The graphical front-end gives also the possibility, once a problem is detected, to focus on its details. The calculation engine is composed by a set of objects known as Sequencers. The calculation rules have been determined by analyzing the various subsystems and global working of the accelerator with plant and operations experts. The Sequencer is designed so that it can also issue commands to the plant. This will be used in the next releases of the Matrix for actively switching from one accelerator configuration to another.

INTRODUCTION

FERMI@Elettra is the new 4th generation synchrotron light source currently under commissioning in Trieste, Italy[1]. It is based on a 1.5 GeV electron linac and seeded free electron laser photon generation. The FERMI@Elettra controls system [2] is based on Tango and is presently made up of about 2800 different Tango devices. The control room operator can access the plant by means of about 1600 different instances of control panels and tens of Matlab applications. These numbers show that the plant is large and complex and can be quite difficult to manage also for experienced users. Another source of complexity comes from the fact the the electron beam can be brought through different paths according to the task that must be performed: several spectrometers for measuring the energy and other characteristics of the beam at different acceleration stages, two bunch compressors, and finally two different undulator chains.

In order to help detecting faults and misconfigurations of the plant, we designed and implemented a tool - known as the ”Matrix” - to give a comprehensive and simple view of the state of accelerator to the control room operators. The Matrix analyzes many accelerator components - interfaced by means of Tango devices - and checks if they are ready to transport the beam to a certain destination. The analysis is done mostly utilizing the *State* of the tango devices and the desired *Scenario*.

States

Every Tango device exports a standard *State* variable. The *State* variable can assume a set of pre-defined values: ON, OFF, FAULT,STANDBY, etc.... By reading this variable we can know wether a device is working or not or if has any problem. Further diagnostics of possible problems is done by means of specific, device dependent informations. The value of the *State* variable is calculated from the operating conditions of the equipment that the Tango device is controlling. All the Tango devices in FERMI@Elettra use the *State* variable consistently.

Scenario

The *Scenario* is a coherent collection of rules used to check that the accelerator is ready to transport the beam to a desired destination, such as one of the various electron spectrometers, or one of the two undulator chains. Each rule is in charge of a well defined section and subsystem of the accelerator. Many of the rules are re-used in different scenarios, reflecting the fact the the electron beam must go through the same sections of the accelerator. The *Scenario* is selected and activated by the control room operator.

GRAPHICAL INTERFACE

The graphical interface (see Fig. 1) is horizontally divided in two areas. The upper area display the accelerator Matrix, the lower area shows a diagram of the accelerator and in formations about the selected scenario. The Matrix displays the state of the accelerator by means of a number of ”coloured lamps”. The lamps are organized in tabular format. Columns are associated to accelerator sections and ordered according to the beam path. Rows are associated to different sub-systems of the accelerator equipment: magnets or power supplies, diagnostics, vacuum, RF, laser. A lamp of a cell thus shows the status of a family of devices of a certain section of the accelerator. The colour of the lamp indicates if these devices are working as expected (green) or not (red). The lamp can be also grayed out when the

[*] This work was supported in part by the Italian Ministry of University and Research under grants FIRB-RBAP045JF2 and FIRB-RBAP06AWK3

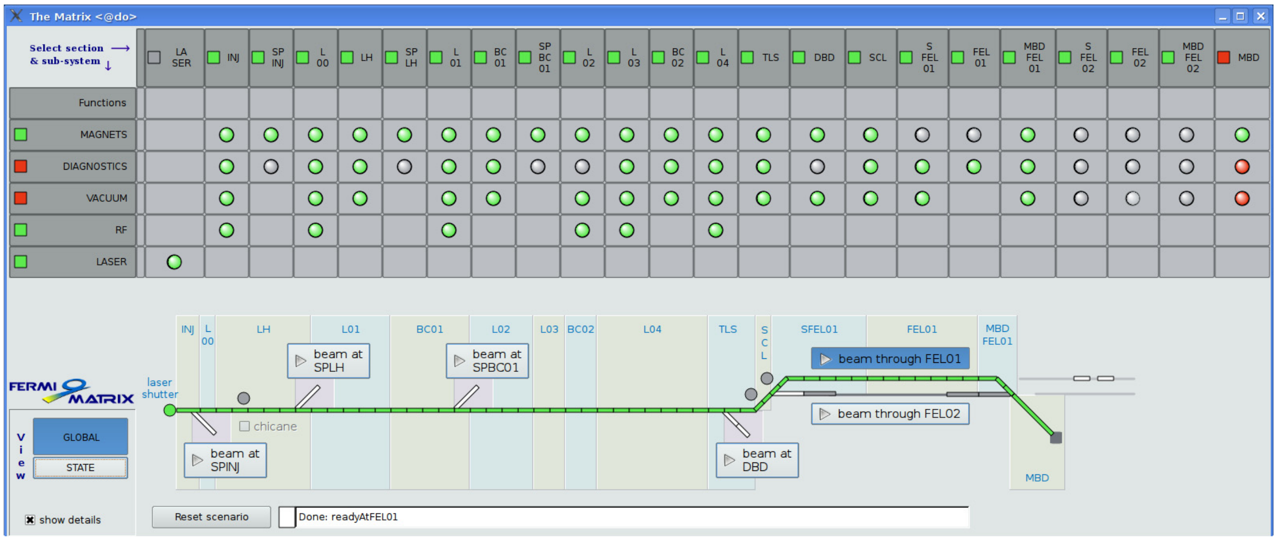

Figure 1: The Matrix panel.

cell is not taken into account in the active *Scenario*. Row and columns headings are also coloured. Their colour summarizes the state of the accelerator section or sub-system according to the active *Scenario*.

Diagnosis of Problems

In case of a problem, signaled by a red lamp, the operator can click on it and check which are the devices that do not behave as expected (see Fig. 2).

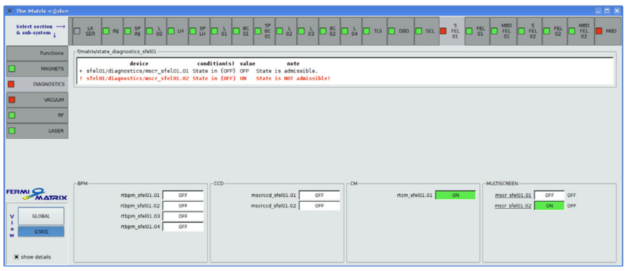

Figure 2: Diagnosis of a problem.

The view is switched to a table that emphasizes the rules which are not met. The state of all the devices involved in the calculations is also displayed in the lower part of the table. By clicking on the state label is then possible to start the control panel dedicated to the specific device. With just two mouse clicks most of the problems or misconfigurations are detected and explained.

Accelerator Diagram

The lower part of the graphic interface displays a schematic diagram of the accelerator and a number of buttons for selecting and activating the different *Scenarios*; the button corresponding to the active *Scenario* is highlighted. The schematic of the accelerator is divided in segments. Each segment is linked to a charge sensitive instrument (BPM, Charge Monitor), so that when the beam is transported through the instrument the corresponding segment is coloured in green.

SEQUENCER

The Sequencer is the building block of the calculation engine used by the Matrix. It is developed in Python starting from the *SequencerBlock* class which derives from standard Python threading.Thread class. The *SequencerBlock* class overrides the run() method (this is required by the Threading class; the run method is called indirectly by the the start() method of the Threading class), implementing the activity diagram shown in UML format in Fig. 3. The run() sequence codifies and formalizes the standard steps that must be done in order to verify equipment conditions or actively modify them; it is foressen in fact that Sequencers will be used also to automate the operations of FERMI@Elettra such as start-up, shutdown, changing of beam path, etc. . .

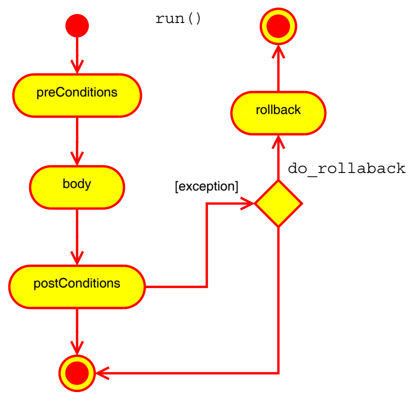

Figure 3: Calculation Engine: main sequence diagram.

The design of the Sequencer comes from many years of practice and experiences of accelerator operations automation [3], which have been thoroughly reviewed and upgraded to object-oriented programming concepts and techniques. The Sequencer has been designed from the start to be easily integrated with the Tango control system.

preConditions, body, postConditions

The Sequencer action is logically divided in three steps, which are called by the run() method. First, check if there are conditions to start the action; this is done in *preConditions* method. Then execute the effective action in the *body* method: calculate the State (that is colour) of a lamp of the Matrix, or issues commands to equipment to change the operating conditions of the plant. Finally check the outcome of the previous steps; this is done in the *postConditions* method.

Errors and Recovery

The three steps:*preConditions*, *body*, *postConditions*, notify errors by rising exceptions which are caught and handled by the *run* method. The Sequencer also provides a *rollback* method which can be used to try to restore the initial conditions in case the *postConditions* are not checked. The call to *rollback* must be explicitly enabled by the programmer of the Sequencer.

Implementation of Real Sequencer Objects in Python

The Sequencer provides a base class and a pattern; the effective knowledge (or intelligence) of the plant that must be checked by a real instance of the Sequencer must be coded by the programmer and is provided by machine and operations experts. The programmer must always derive new Sequencer object from *SequencerBlock* class. The new Sequencer implements directly the three main methods and can add other methods which can be useful for the specific situation. At the same time the programmer can use already existing Sequencer objects by aggregation. Sequencers are written in Python. With Python the programmer can very easily load end execute some Python module at runtime. In this way the programmer can modify the behavior of a Sequencer object at runtime. This feature has been extensively used in the Matrix for changing the behaviour of the program when a new *Scenario* is selected.

MATRIX SEQUENCERS

Several Sequencers have been developed for the Matrix. They perform two main tasks: first, calculate the state of Matrix cells, rows or columns headers: they are derived from *StatesBlock*; second, manage the setting of *Scenarios*: these objects are derived from *ScenarioBlock* (See Fig.4).

StatesBlock

The *StatesBlock* class is a sequencer that analyzes the Tango *State* variable of a group of devices and calculates a resulting *State*. A *StatesBlock* object is configured with a list of Tango devices and their respective admissible states. The *StatesBlock* body method periodically samples the configured devices and analyzes if the group is in an admissible state. The *StatesBlock* contains also some auxiliary methods to report in detail the result of the analysis. The report is used by the Matrix when the detailed view is selected. All the Matrix cell states are calculated by dedicated specializations of *StatesBlock* (e.g. diagnostics_dbd in Fig.4) *StatesBlock* is also the base class for *HeaderStatesBlock* objects which are used to calculate the State (that is the colour of the lamp) for a column or row of the Matrix. *HeaderStatesBlock* use simple syntactical rules to automatically create the list of Tango devices and admissible States.

Deployment

The Matrix needs one *StatesBlock* object for each of its active cells. All these objects are managed by a dedicated Tango device server written in Python, which exports a simple and clean interface for interacting with a *SequencerBlock* object. This deployment scheme makes all the stuff needed by the Matrix generally available as standard control system components. The most important technical reason for this deployment scheme is that it ensures that there is exactly one instance of each of the configured

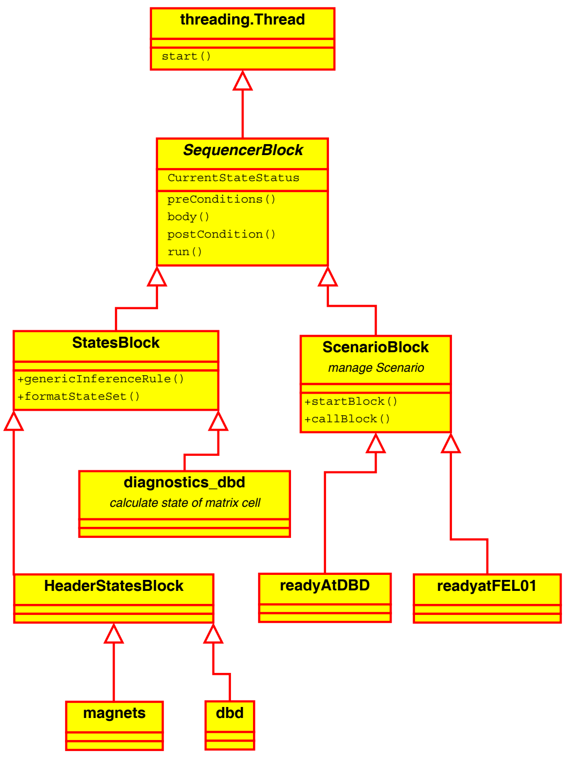

Figure 4: Calculation Engine: class diagram.

SequencerBlock running in the control system of the plant: in this way, when the *SequencerBlock* objects is be used to modify the plant configuration they will do it in an ordered and consistent manner.

ScenarioBlock

Objects of the *ScenarioBlock* class manage the activation of *Scenarios*. A *ScenarioBlock* is programmed with the list of Tango devices and the corresponding *StateBlocks* objects that must be activated for the given *Scenario*. The configured Tango devices are loaded with the chosen *StatesBlock* and started when the *ScenarioBlock* is activated. The *ScenarioBlock* class is derived from *StatesBlock* so that we can reuse the Tango State analysis and reporting features.

CONCLUSION AND OUTLOOK

The Matrix has been used during the commissioning shifts and has shown to be an effective tool for monitoring the general status of the plant. The effectiveness of the Matrix depends strictly on the quality and completeness of the Scenarios: a good collaboration and mutual understanding between machine experts and programmers is of paramount importance in order to correctly and completely analyze and define the various rules. We expect to improve and extend the Scenarios and the diagnostics capabilities of the Matrix with the knowledge gained during the commissioning of FERMI@Elettra.

The next important development of the Matrix will be the automation of operations. As the new accelerator will start operations for FEL light users, standard procedures will be defined for setting up the plant and guaranteeing optimal and repeatable operating conditions. These procedure will be analyzed and implemented by means of *SequencerBlock* objects and supervised by the Matrix. The ultimate goal is to make FERMI@Elettra operable by a team of two control room operators.

THANKS

The initial design and development of the system described in this paper was done by Fabio Asnicar.

REFERENCES

[1] S. Di Mitri, “Commissioning and Initial Operation of FERMI@Elettra”, IPAC 2011, San Sebastin, September 2011.

[2] M. Lonza et al, “Status Report of FERMI@Eelettra Control System”, ICALEPCS 2011, Grenoble, October 2011

[3] D. Bulfone, F. Potepan, C. Scafuri, “Automating ELETTRA Operation with ‘One Button Machine’ ” 17Th Particle Accelerator Conference, Vancouver, Canada, 12 - 16 May 1997, p. 2467