ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEAL02

A FRAMEWORK FOR HIGH LEVEL MACHINE AUTOMATION BASED ON BEHAVIOR TREES

G. Gaio†, P. Cinquegrana, S. Krecic, G. Scalamera, G. Strangolino, F. Tripaldi, M. Trovò, L. Zambon, Elettra-Sincrotrone Trieste S.C.p.A., Trieste, Italy

Abstract

In order to carry out complex tasks on particle accelerators, physicists and operators need to know the correct sequence of actions usually performed through a large number of graphical panels. The automation logics often embedded in the GUIs prevents its reuse by other programs, thus limiting the level of automation a control system can achieve. In order to overcome this limit, we have introduced a new automation framework for shifting the logics from GUIs to server side, where simple tasks can be easily organized, inspected and stacked up to build more complex actions. This tool is based on Behavior Trees (BT) which has been recently adopted in the gaming industry for in-game AI player opponents. They are able to create very complex tasks composed by simple decoupled self-contained tasks (nodes), regardless how they are implemented. The automation framework has been deployed in the Tango-based control systems of Elettra and FERMI to implement autonomous operations. A dedicated Qt GUI and a web interface allow to inspect the BTs and dynamically go through a tree, visualize the dependencies, monitor the execution and display any running action.

INTRODUCTION

In Elettra and FERMI, a synchrotron light source and a free electron laser located in Italy, it is usual for control room operators or physicists to manually perform long sequences of operations to configure the accelerators in the desired state. These procedures are prone to errors and heavily dependent on the skills of the operators. Over time, many institutes have developed frameworks to automate these lengthy procedures [1,2,3,4]. Therefore, the framework we are going to present in this article is not an absolute novelty.

To be successful a framework must be easy to use, robust and adopted by as many people as possible. These concepts were the basis for the development of this new framework. The originality lies in inheriting the modularity, flexibility and robustness of the Behavior Trees (BT).

Behavior Trees

BTs are very efficient in modelling what an Artificial Intelligence (AI) algorithm can do. They allow designers to define very low-level tasks and combine them to create the set of high-level tasks that the developer wants available to the AI application.

The Behavior Tree is a directed rooted node tree where the internal nodes are called “control flow nodes” and leaf nodes are called “execution nodes” (see Fig. 1). Briefly, the execution flow starts from a root node and go through the tree down to the leaves. The main control flow node is the *sequence node* that can run in parallel or in series to other *sequence nodes* or *action nodes*. An *action node* executes a task and returns to its parent a *success*, *running* or *failure* state. The *sequence* node returns *success* to its own parent if all its children return *success*. The *sequence node* can be configured as *fallback node* to return *success* when at least one of its children return *success*. A *condition node* returns *success* or *failure* depending on the evaluating condition. A *decorator node* can invert a *failure* state into *success* and vice-versa.

There is no canonical implementation of BTs. They are very flexible and suitable to be customized for any application, whether it is an AI algorithm in a computer game [5] or in an Unmanned Aerial Vehicle [6].

Each node executes an instruction after receiving a tick from its parent. In our implementation this aspect has been neglected. The *Action* node start to executes the task at the first tick and completes procedure detached from any external timing signal.

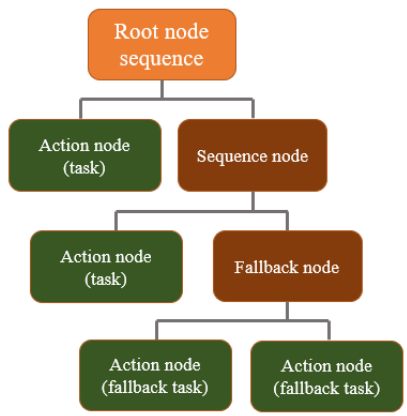


Figure 1: Example of Behavior Tree structure.

THE SEQUENCER

A sequencer (corresponding to a node of the BT) is a Tango device. The core of the sequencer is the sequence which is written in a basic homemade language that has to implement the operations that are performed manually by operators. The language implements macros containing *if/else* conditions and *read/set* instructions, and support basic arithmetic and bitwise operations.

† email: giulio.gaio@elettra.eu

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEAL02

The sequence is stored in the sequencer Tango device property or in a text file (see Fig.2). The loading of the sequence is carried out during the Tango device initialization. After that, the sequencer device executes the sequence immediately or when a *start* command is received.

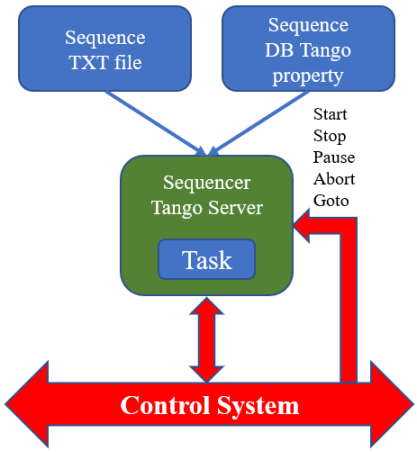


Figure 2: Sequencer device architecture.

The Tango server executes the sequence instructions (commands and read/write attributes) row by row. Each row is identified by a step number (see Fig. 3). For each step a mandatory *goto* statement specifies the next step to execute. The sequence ends after evaluating the last step or can last forever.

The sequencer states are:

- OFF: sequencer is not running (success state)
- RUNNING: sequencer is executing the sequence code
- FAULT: sequencer has terminated due to an exception
- WARNING: sequence parsing error
- STANDBY: the sequence is paused

Internal Variables

The sequence can instantiate internal variables (ex: a loop counter, a threshold, …) which are dynamically mapped into read/write attributes. For simplicity the attribute types are *bool*, *int32*, *int64* and *double,* and can support vectors and images.

Stop Conditions and Exceptions

The sequence stops after receiving an *abort* or *stop* command. While the *abort* command halts the sequence immediately, the *stop* command lets the sequence terminate the running step. In both cases the sequence ends in OFF state, which means *success*.

The sequence can be paused upon receipt of the *pause* command or can execute a single step specified in the *goto* command.

The instruction *goto(-1)* generates an internal exception that stops the sequence in FAULT state. The *goto(N+1)* instruction, where N is the last step number, terminates the sequence in OFF state.

There are four types of exceptions.

- *syntax exception*: it occurs during the sequence loading process in case of a syntax error. The sequencer state turns into ALARM and it cannot be restarted. The status attribute returns the part of the sequence containing the syntax error.
- *global timeout exception*: the sequence execution time takes more than the global timeout; the sequence ends and the device state is set to FAULT.
- *step timeout exception*: the step execution time takes more than its timeout; the sequence stops and the Tango device state changes to FAULT.
- *Tango exception*: when a Tango exception occurs in a command or read/write attribute, the sequence is immediately stopped and the sequence state is set to FAULT. Tango exceptions can be ignored; in this case all *if/else* statements involving Tango read attributes are evaluated as true.

Tango and step timeout exceptions can be caught and the execution flow redirected to a specific step. Similarly, the *Stop* command can be intercepted by the sequencer engine that, instead of ending the sequence, jumps to a predefined "exit" step.

The sequencer can be configured to have its error recognized by the operator. In this case the sequencer requires an *acknowledge* command before receiving a new *start* command.

stepN;[expression];[step description];[error message];[timeout ms.];[catch exceptions]

Example:
step1;read(sr/ps/ch_s1.1/State)==FAULT **?** command(sr/ps/ch_s11/Reset) && **goto(2) : goto(3)**;Reset PS;Error resetting PS;3000
step2;read(sr/ps/ch_s1.1/State) != OFF **?** sleep(1) && **goto(2) : goto(3)**;Waiting OFF state;Timeout waiting OFF state;6000
step3;read(sr/ps/ch_s1.1/State) != ON **?** command(sr/ps/ch_s11/On) && **goto(4) : goto(5)**;Turn PS ON;Error turning ON PS;3000
step4;read(sr/ps/ch_s1.1/State) != ON **?** sleep(1) && **goto(4) : goto(5)**;Waiting ON state;Timeout waiting ON state;6000

Description:
step1; If PS(sr/ps/ch_s1.1) is in FAULT state then Reset and go to step2, otherwise go to step3
step2; if PS(sr/ps/ch_s1.1) is not OFF then sleep one sec and check again (for max 6 sec.), otherwise goto step3
step3; if PS(sr/ps/ch_s1.1) is not ON turn PS ON and go to step4, otherwise go to step5 (exit)
step4;if PS(sr/ps/ch_s1.1) is not ON then sleep one sec and check again (for max 6 sec.), otherwise goto step5 (exit)

Figure 3: Description of the sequencer scripting language. Each code line is composed at least by 5 fields: step index, logical expression, step description, error message and timeout. The catch exception field is optional. The if/else statement is implemented with the ternary conditional operator (? :).

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEAL02

Logging

The sequencer device implements data logging on two levels:

- *high resolution buffer*: a circular buffer of 10000 entries records every instruction executed inside the sequence, any value read or set, and any error or conditional jump with microsecond resolution. It is mostly used by developers during the sequence debugging and, in the near future, analysed by software bots for malfunction prediction.
- *Long-term buffer*: in order to record any modification on machine parameters, each sequencer can store all executed Tango commands and written attributes in a circular buffer of 100000 entries.

The description and error messages encoded in each sequence step are also stored in the long-term buffer.

Currently both buffers are reset after device reinitialization.

Additional Info

In the header of the sequence code it is possible to specify a description of the sequence, the author and the release date.

The sequence device returns the last execution time, the elapsed time from the start, the remaining time till the global timeout and, in case of exception, the last faulty step. Moreover, during sequence execution, the description of the running step is pushed by the sequencer to any event subscriber.

In order to trace dependency between sequence devices, each device returns separately the list of the sequence devices and the list of other Tango devices to which it is connected. This information is useful for monitoring, at a higher (client) level, that the BT respects a Directed Acyclic Graphs (DAG) structure. In fact, no loops are allowed between parent nodes (ex: the root node) and the child nodes.

Templates

In order reuse the same sequence, the concept of template has been introduced. After loading the template from file and before checking its syntax, the Tango sequencer engine replaces keywords with the strings specified in a device property called *wildcards*. The replacing strings are usually Tango device names or constant values.

About fifty templates are available for sequencer developers. Some of them execute specific tasks (ex. recovering a faulty plant), other implement BT flow control, generic algorithms as scans/ramps, basic optimization algorithms (Extremum Seeking, Golden Search) or save/restore procedures. The most used template is the *launcher*. It is the main building block of the BT as it implements both a sequencer and fallback node.

The *launcher* can start up to 64 sub-sequences. Through the use of bit-masks (r/w attributes) it can select groups of sub-sequences:

- to be launched in series and/or parallel.
- to be restarted after a fault for a maximum number of times.
- to be ignored if producing errors and continue in launching the remaining sub-sequences.
- to be stopped at the first fault and abort the launch of the remaining sub-sequences.

Furthermore, a per sequence boolean attribute enables/disable its execution at runtime.

After all sub-sequences have been executed successfully, the *launcher* stops in OFF (or FAULT in case of errors) and its state is acquired by the parent node. When a *launcher* is going to terminate in FAULT state, it is possible to execute, as the last running sequence, a rollback sub-sequence.

Sometimes it is useful to stop, pause or abort the execution of an entire BT. In order to do so, the *launcher* sequencer, if explicitly enabled, can propagate all received commands directly to all child nodes.

Naming and Database Configuration

In the Tango database the naming of the sequences follows the rule:

seq/[*action*]/[*detailed_description*]_[*where*]

As example, the naming of a sequence which performs the power-cycling and communication recovery with a CCD in the Elettra booster is:

seq/reset/ccd_b1.1

Similarly, the sequence calculating the global orbit feedback response matrix in the storage ring is:

seq/calc/gof_rm_sr

A global free property in the Tango database called *sequencer* contains several sub-properties, each one containing a list of root sequence devices grouped by usage. For the Elettra storage ring the sub-property list names are Feedback, InsertionDevices, PowerSupplies, MSCR, Optimization, Recover, BeamOFF.

Another sub-property contained in the *sequencer* global free property called *SuperList* contains the name of all the sub-properties listed above. This information is used by graphical applications to dynamically build up the GUIs for configuration, monitoring and control of the sequence-based operations.

Executing External Programs

The sequencer language supports a restricted group of Tango APIs. For more complex tasks it is possible to execute in the sequence, by means of a c-style “exec” system call, Python, Matlab, Bash and any other type of script.

GRAPHICAL INTERFACE

One of the main problems of automation is making the operator aware of the operations involved in a procedure and understand what is the cause of an error. Both in FERMI and Elettra BTs are made of up to a few hundreds of sub-sequences and the number is constantly growing. In order to provide the operators with flexible command and control tools two graphical panels based on Qt-Cumbia libraries [7] have been developed.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEAL02

Sequencer GUI

The panel loads a list of predefined root sequences or, by means of Tango wildcards (ex: seq/*/* selects all sequences in Tango database), composes a dynamic list of all the sequences matching the search criteria. Thanks to the information gathered from the devices and the Tango database, the panel identifies the BT dependencies and builds the BT in a hierarchical list format. The user can inspect the entire BT from the root sequence node down to the leaf nodes (see Fig. 4). A tree structure view is also available. By browsing the hierarchical list, the user gets the description, the state (Tango *state* attribute), the last execution time and the last execution elapsed time of each sequence. In the hierarchical list the states of all non-sequence Tango devices logically connected to the BT are also reported.

The user can start/stop an entire BT or just a part of it. The source code of each sequence can be displayed in a read-only format.

A textual search can be performed simultaneously on the logging buffers of all the sequences belonging to a BT. Furthermore, the panel not only displays live, thanks to the Tango events, the list of errors and settings performed by one BT, but can also collect any action performed by all the sequencers configured in a Tango database.

SeqLauncher GUI

This panel read the *SuperList* free property from the Tango database and creates as many tabs as the number of lists found inside the *SuperList* property. Each tab contains all the root sequences belonging to the corresponding list (see Fig. 5). For each sequence an "OpenGUI" button starts preconfigured legacy applications that ease the monitoring of the BT execution. Similarly, the "CloseGUI" button kills all applications started by the OpenGUI button. This is the panel used by the operators in control room to access directly the sequencers.

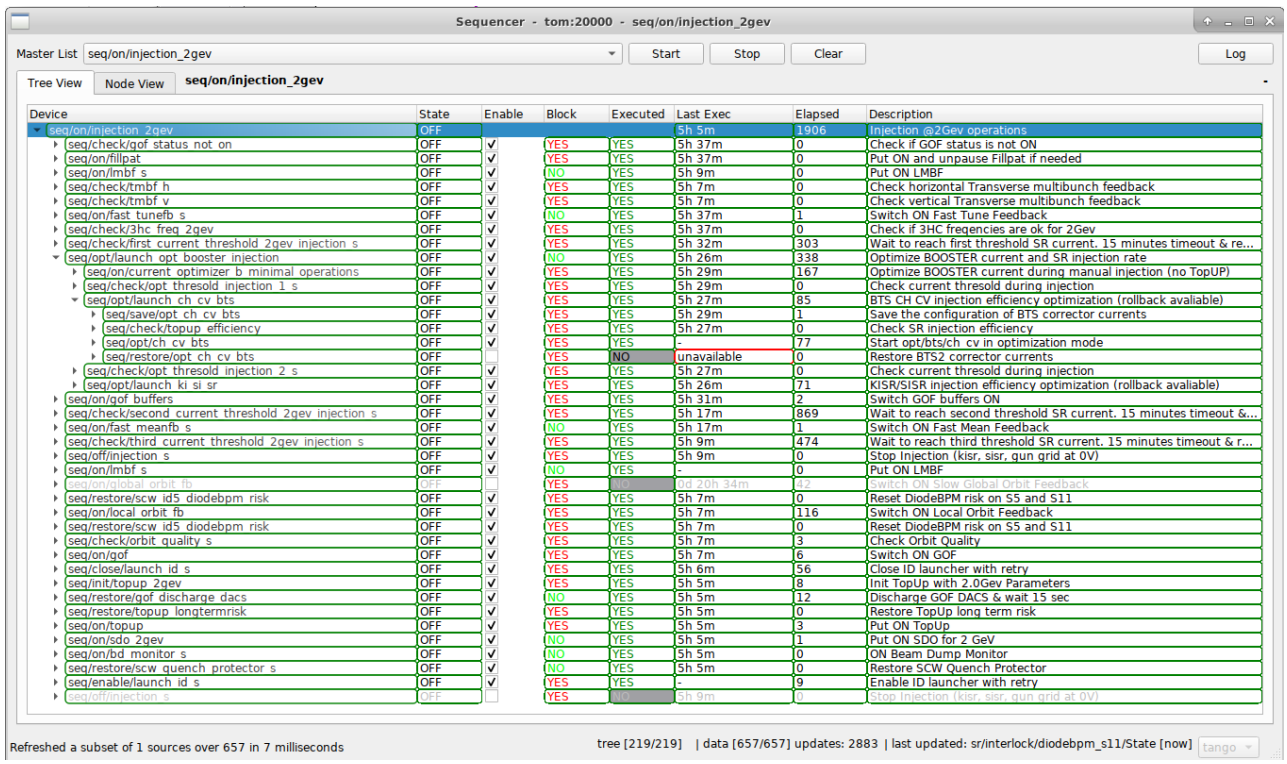
WEB interface

A web interface (see Fig. 6) based on PUMA [8] allows listing all sequence devices installed in FERMI and Elettra. Sequences can be remotely started by users after LDAP authentication.

APPLICATIONS

There are 958 sequencer devices installed in the Elettra Tango database and 995 in the FERMI Tango database.

In Elettra, most of the high-level applications used for the accelerator automation have been replaced by sequencers. There are two main sequencers, the first for recovering the machine from a faulty state, the second for injecting the beam into the storage ring and performing optimizations and orbit correction until a stable beam can be delivered to the beamlines. Thanks to the BT modularity both of them have been embedded in larger sequencer that is able to run the machine autonomously. The work to refine especially the recovery sequencers is ongoing with the goal to face any anomalous situation from which the machine could be automatically recovered.



Device	State	Enable	Block	Executed	Last Exec	Elapsed	Description
seq/on/injection_2gev	OFF				5h 5m	1906	Injection @2Gev operations
seq/check/gof status not on	OFF	✓	YES	YES	5h 37m	0	Check if GOF status is not ON
seq/on/fillpat	OFF	✓	YES	YES	5h 37m	0	Put ON and unpause Fillpat if needed
seq/on/lmbf s	OFF	✓	NO	YES	5h 9m	0	Put ON LMBF
seq/check/tmbf h	OFF	✓	YES	YES	5h 7m	0	Check horizontal Transverse multibunch feedback
seq/check/tmbf v	OFF	✓	YES	YES	5h 7m	0	Check vertical Transverse multibunch feedback
seq/on/fast tunefb s	OFF	✓	NO	YES	5h 37m	1	Switch ON Fast Tune Feedback
seq/check/3hc freq 2gev	OFF	✓	YES	YES	5h 37m	0	Check if 3HC freqencies are ok for 2Gev
seq/check/first current threshold 2gev injection s	OFF	✓	YES	YES	5h 32m	303	Wait to reach first threshold SR current. 15 minutes timeout & re...
seq/opt/launch opt booster injection	OFF	✓	NO	YES	5h 26m	338	Optimize BOOSTER current and SR injection rate
seq/on/current optimizer b minimal operations	OFF	✓	YES	YES	5h 29m	167	Optimize BOOSTER current during manual injection (no TopUP)
seq/check/opt thresold injection 1 s	OFF	✓	YES	YES	5h 29m	0	Check current thresold during injection
seq/opt/launch ch cv bts	OFF	✓	YES	YES	5h 27m	85	BTS CH CV injection efficiency optimization (rollback avaliable)
seq/save/opt ch cv bts	OFF	✓	YES	YES	5h 29m	1	Save the configuration of BTS corrector currents
seq/check/topup efficiency	OFF	✓	YES	YES	5h 27m	0	Check SR injection efficiency
seq/opt/ch cv bts	OFF	✓	YES	YES	-	77	Start opt/bts/ch_cv in optimization mode
seq/restore/opt ch cv bts	OFF		YES	NO	unavailable	0	Restore BTS2 corrector currents
seq/check/opt thresold injection 2 s	OFF	✓	YES	YES	5h 27m	0	Check current thresold during injection
seq/opt/launch ki si sr	OFF	✓	YES	YES	5h 26m	71	KISR/SISR injection efficiency optimization (rollback avaliable)
seq/on/gof buffers	OFF	✓	YES	YES	5h 31m	2	Switch GOF buffers ON
seq/check/second current threshold 2gev injection s	OFF	✓	YES	YES	5h 17m	869	Wait to reach second threshold SR current. 15 minutes timeout &...
seq/on/fast meanfb s	OFF	✓	NO	YES	5h 17m	1	Switch ON Fast Mean Feedback
seq/check/third current threshold 2gev injection s	OFF	✓	YES	YES	5h 9m	474	Wait to reach third threshold SR current. 15 minutes timeout & r...
seq/off/injection s	OFF	✓	YES	YES	5h 9m	0	Stop Injection (kisr, sisr, gun grid at 0V)
seq/on/lmbf s	OFF	✓	NO	YES	-	0	Put ON LMBF
seq/on/global orbit fb	OFF		YES	NO	0d 20h 34m	42	Switch ON Slow Global Orbit Feedback
seq/restore/scw id5 diodebpm risk	OFF	✓	YES	YES	5h 7m	0	Reset DiodeBPM risk on S5 and S11
seq/on/local orbit fb	OFF	✓	YES	YES	5h 7m	116	Switch ON Local Orbit Feedback
seq/restore/scw id5 diodebpm risk	OFF	✓	YES	YES	5h 7m	0	Reset DiodeBPM risk on S5 and S11
seq/check/orbit quality s	OFF	✓	YES	YES	5h 7m	3	Check Orbit Quality
seq/on/gof	OFF	✓	YES	YES	5h 7m	6	Switch ON GOF
seq/close/launch id s	OFF	✓	YES	YES	5h 6m	56	Close ID launcher with retry
seq/init/topup 2gev	OFF	✓	YES	YES	5h 5m	8	Init TopUp with 2.0Gev Parameters
seq/restore/gof discharge dacs	OFF	✓	NO	YES	5h 5m	12	Discharge GOF DACS & wait 15 sec
seq/restore/topup longtermrisk	OFF	✓	YES	YES	5h 5m	0	Restore TopUp long term risk
seq/on/topup	OFF	✓	YES	YES	5h 5m	3	Put ON TopUp
seq/on/sdo 2gev	OFF	✓	NO	YES	5h 5m	1	Put ON SDO for 2 GeV
seq/on/bd monitor s	OFF	✓	NO	YES	5h 5m	0	ON Beam Dump Monitor
seq/restore/scw quench protector s	OFF	✓	NO	YES	5h 5m	0	Restore SCW Quench Protector
seq/enable/launch id s	OFF	✓	YES	YES	-	9	Enable ID launcher with retry
seq/off/injection s	OFF		YES	NO	5h 9m	0	Stop Injection (kisr, sisr, gun grid at 0V)

Figure 4: Overview of the BT used for the Elettra storage ring injection at 2GeV. The Root node launches 31 sub-sequences over a total of 219. The total procedure lasted 1906 seconds. Excluding the "waiting tasks", the longest ones during the injection are: automatic optimizations (338 sec.), orbit correction (116 sec.), undulator gaps closing (56 sec.).

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEAL02

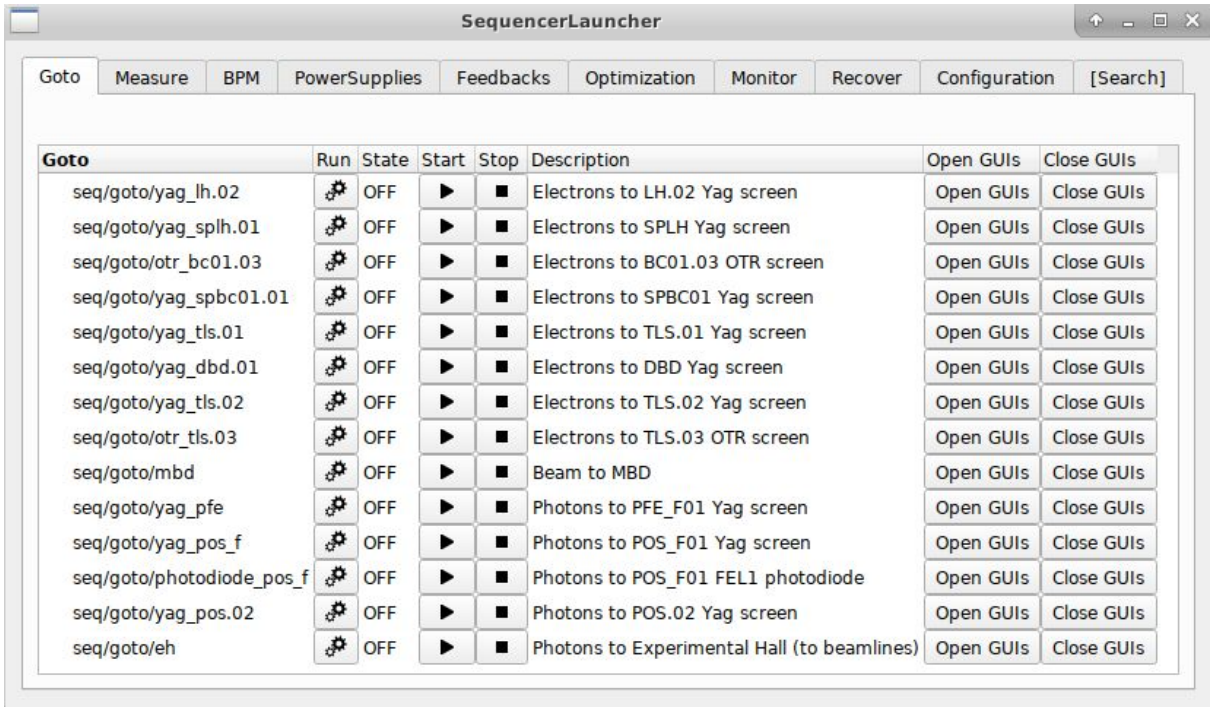


Figure 5: Control panel for driving electron and photon beams to diagnostic stations and beamlines in FERMI.

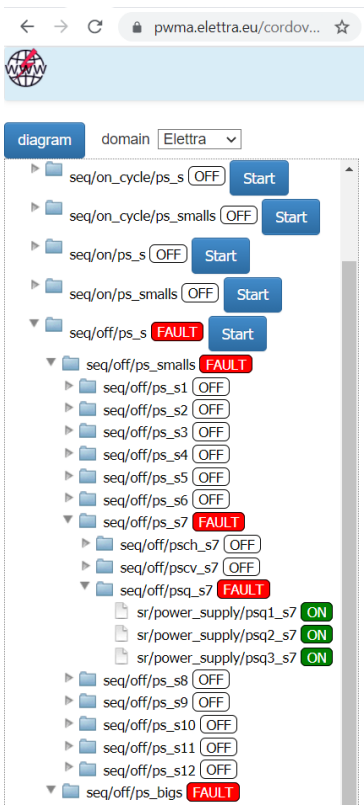


Figure 6: Web based sequence browser.

In FERMI the sequencers have been introduced quite recently. They supervise the online the FEL automatic optimizers and reconfigure the machine for delivering the electron and photon beams to diagnostic stations and beamlines.

CONCLUSION

With the adoption of sequencers, developing automation applications with very complex logics has become much easier than in the past. More people, even non-professional programmers, are actually working on automation, bringing machine autonomous operations to unprecedent levels of complexity.

Thanks to the sequencers, moving logics from GUIs to server applications is possible and convenient: the server-side logics can be recalled by any client in the control system without caring of its implementation. Moreover, encoding in a sequencer a procedure that is normally written in an operator manual or, even worse, known by just one expert, is a clear advantage. Once the optimal sequence is defined and encoded, everyone can perform that task at expert level.

Another advantage in the present Machine Learning era is that, by using sequencers, the machine operations become more deterministic. Machine settings are not polluted by the randomicity of the human actions. As a consequence, any Machine Learning algorithm designed to learn from logs any insight of malfunctions or weird behavior of the machine, will benefit from processing cleaner data.

To start playing with the sequences there is no need of external software except the sequencer Tango server and the Tango control system. For building up more complex applications, the use of the *Launcher* template and the Qt GUIs are strongly suggested.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-WEAL02

REFERENCES

[1] D.E. Johnson, R.P. Johnson, "The Colliding Beam Sequencer", in *Proc. 13th IEEE Particle Accelerator Conference (PAC89),* Chicago, IL USA, March 1989, doi:10.1364/OE.20.11396

[2] D. Bulfone, F. Potepan, C. Scafuri, "Automizing ELETTRA Operation with One Button Machine", in *Proc. 17th IEEE Particle Accelerator Conference (PAC97),* Vancouver, Canada, May 1997, doi: 10.1109/PAC.1997.751242

[3] J. van Zeijts, T. D'Ottavio *et al.*, "The RIHC Sequencer", in *Proc. of the 2001 IEEE Particle Accelerator Conference (PAC01)*, Chicago, IL USA, June 2001, paper MPPH008, pp. 782-784, doi: 10.1109/PAC.2001.986473

[4] V. Baggiolini, R. Alemany-Fernandez *et al.*, "A Sequencer for the LHC ERA", in *Proc. 12th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09),* Kobe, Japan, Oct. 2011, paper THC003, pp. 670-672.

[5] Y. A. Sekhavat, "Behavior Trees for Computer Games", *International Journal of Artificial Intelligence Tools*, 26(2), Jan. 2017, doi: 10.1142/S0218213017300010

[6] P. Ögren, "Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees", in *Proc. AIAA Guidance, Navigation, and Control Conference (MGNC12),* Minneapolis, Minnesota USA, Aug. 2012, doi: 10.2514/6.2012-4458

[7] G. Strangolino, "Cumbia: A New Library for Multi-Threaded Application Design and Implementation", in *Proc. 16th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'17),* Barcelona, Spain, Oct. 2017, paper TUPHA174, pp. 830-834, doi: 10.18429/JACoW-ICALEPCS2017-TUPHA174

[8] G. Strangolino, L. Zambon, "Canone3: a New Service and Development Framework for the Web and Platform Independent Applications", *presented at the 18th Int. Particle Accelerator Conf. (ICALEPCSC'21)*, FRAR02, Oct 2021, this conference