ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-MOPV013

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

A DYNAMIC BEAM SCHEDULING SYSTEM FOR THE FAIR ACCELERATOR FACILITY

S. Krepp*, J. Fitzek, H. Hüther, R. Mueller, A. Schaller, A. Walter
GSI Helmholtz Centre for Heavy Ion Research, Darmstadt, Germany

Abstract

The new Accelerator Control System for GSI/FAIR is now being used productively for the GSI accelerator facility. As the central component for online beam orchestration, the Beam Scheduling System (BSS) is situated between the FAIR Settings Management System and the FAIR Timing System. Besides device settings, the Settings Management System provides timing schedules for beam production. The primary purpose of BSS is to define which of the beam schedules are executed by the Timing System, how often and in which order. To provide runtime decisions in pre-planned execution options (e.g. skipping of a particular beam), it processes external signals like user input, experiment requests or beam prohibits provided by the interlock system. More recently, advanced features have been added that allow for dynamic execution control required by Storage Ring Mode features such as breakpoints, repetitions, skipping and manipulations. This contribution gives an overview of the Beam Scheduling System including its interfaces.

INTRODUCTION

One of the major building blocks of the new Accelerator Control System for GSI/FAIR is the Beam Scheduling System (BSS). Residing in the middle tier of the FAIR Control System, its core functionality is the orchestration of beams based on user requests. Figure 1 shows how BSS is embedded into the overall Control System architecture.

In the FAIR Settings Management System LSA, beams are represented as Beam Production Chains and are put together to Patterns for defining an execution sequence. This includes both, settings for hardware devices, as well as an execution schedule defining which Timing Events are to be sent to which parts of the facility. Additionally, by assigning Patterns to Pattern Groups, LSA defines which Patterns must be executed sequentially and which Patterns may run in parallel. The schedule for each Pattern and the information about Pattern Groups is provided to the BSS system, which creates an overall schedule for the accelerator facility and sends it to the Timing System's Generator component. The Generator then translates this schedule to the low-level programming of the Data Master.

Operators and experimenters define, which beams they would like to have produced using applications and services, that in turn send these requests to BSS. At the same time, the Master Accelerator Status Processor (MASP) determines whether a certain Pattern can be executed, by collecting status and interlocks of all required devices and services. BSS then combines both of these inputs and sends commands to

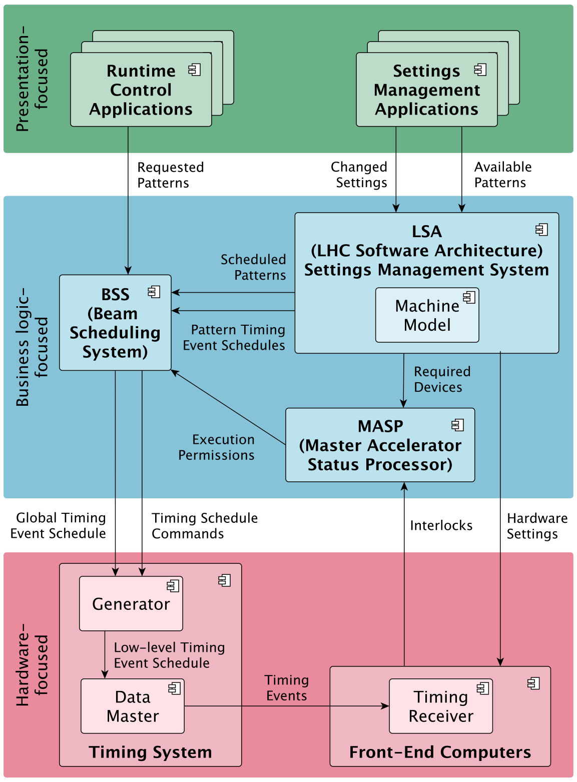

Figure 1: BSS in the FAIR Control System.

the Generator, that dynamically modifies the overall schedule accordingly. At runtime, the Data Master sends out timing events via the White Rabbit-based timing network. Those events lead to synchronous execution of settings in devices as calculated and provided by LSA.

As shown, BSS is the central instance that decides which of the preconfigured schedules are executed. This contribution presents technical details of the BSS system and its interfaces to give an insight, of how beam orchestration is realized within the GSI/FAIR Accelerator Control System.

SCHEDULE REPRESENTATION

Schedules within BSS are represented as schedule graphs that can be executed by the Timing System [1]. The graph itself is represented in the .dot format [2]. The vertices in this directed graph can roughly be interpreted as executable units linked together by a set of edges. Once started, a control thread walks over those vertices one after another in order to execute them. The actual sequence of vertices is defined by a set of edges marked as default destinations. Alternative routes through the schedule are defined by a set of alternative destinations.

* s.krepp@gsi.de

To allow for dynamic schedule changes, the Timing System supports switching of default destinations in the schedule graph, thus allowing the beam schedule to be altered at runtime by changing the default path through the schedule graph. BSS is responsible for managing the configuration of the Timing System by providing beam schedules and for altering those schedules on the fly based on user requests.

SCHEDULE CONFIGURATION

As described, LSA supplies BSS with individual schedules per Pattern and it is up to BSS to integrate them into a global timing schedule. Doing so, the schedules are not just merged, but extended to later gain flexibility at runtime. Conceptionally, the resulting schedule is driven by three major requirements. Firstly, Patterns specified in one Pattern Group must be scheduled sequentially respecting their specified execution order, while Patterns in different Pattern Groups shall run in parallel. Secondly, BSS must be able to permanently enable and disable each individual Pattern schedule's execution to allow for beam prohibits and user requests. Thirdly, BSS must be able to schedule individual Patterns for being executed exactly once.

The first requirement is met, as BSS manages Pattern Schedules on a per Pattern Group basis. It combines the individual Pattern schedules to a circular Pattern Group schedule graph. Different Pattern Groups result in different Pattern Group schedule graphs and can be executed in parallel.

An example Pattern Group schedule graph is shown in Fig. 2. Each individual Pattern schedule has one explicit entry and exit node, e.g. A_ENTRY and A_EXIT. The Pattern schedule for Pattern A is shown here only as one node A for the sake of simplicity. However, in the real schedule graph, node A actually consists of many vertices and edges that represent the actual behaviour of the machine, e.g. the timing events for this Pattern. The currently active loop through the graph is drawn with solid edges, alternative paths trough the graph are drawn with dotted edges.

To meet the second requirement, and to be able to switch the execution of one Pattern on and off, BSS adds an alternative path directly from ENTRY to EXIT, i.e. the edge from A_ENTRY to A_EXIT. To prevent Patterns from being executed right away, once they are provided to the Timing System's Generator and execution is started, the edge that skips Pattern execution is in fact set as the default one. Consequently, it is drawn as a solid edge in the figure. The edge used for actually executing Pattern A is represented by a dotted line, as it must be explicitly switched on later by BSS.

In addition to the Pattern schedules provided by LSA, BSS adds a so-called Default Pattern schedule to the Pattern Group schedule graph. It is used for idle operation, i.e. when no beam is requested. Finally, each Pattern's exit node is extended with outgoing edges to each available Pattern's entry node which gives BSS the ability to build arbitrary loops of Pattern schedules.

To meet the third requirement of being able to execute a Pattern only once, it was necessary to keep the entry nodes

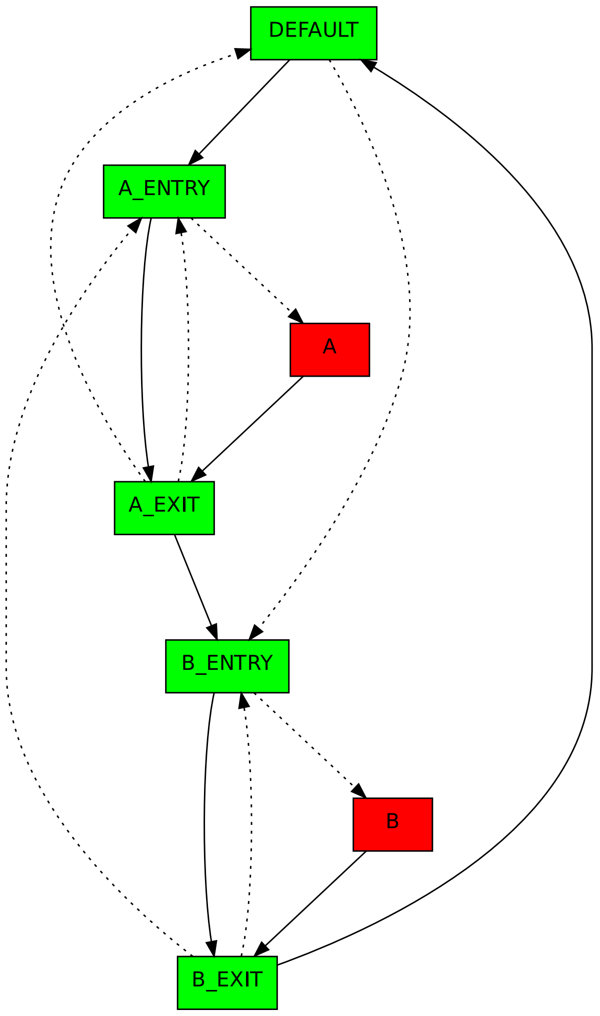

Figure 2: Pattern Group schedule with only the Default Pattern being active.

in the active loop for each Pattern also in the case, that this Pattern is not being requested at the moment. This is why in case no Pattern is requested, the Pattern Group schedule graph's active loop does not only consist of the Default Pattern pointing to itself, but contains all the entry and exit nodes as shown. The single shot request is described in more detail in section NON-PERMANENT OR SINGLE SHOT REQUESTS.

After the Pattern Group schedule has been prepared by BSS, it is added to the Timing System's global schedule. The Timing System supports multithreading by assigning parts of the global timing schedule to a certain thread. In order to support parallel execution of independent Patterns, each Pattern Group schedule is executed in its own thread. With a start command sent to the Generator, the execution begins right away. When the graph from Fig. 2 is executed, the Timing System loops over the nodes that are coloured green: a sequence containing the Default Pattern schedule and the nodes A_ENTRY, A_EXIT, B_ENTRY, B_EXIT. However, the Patterns A and B are not executed since their internal schedule subgraphs A and B are skipped.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

Whenever the Pattern Group gets changed, e.g. a Pattern is added or removed, BSS stops the corresponding thread and replaces the old Pattern Group schedule with the new one.

As shown, the overall timing schedule at FAIR includes schedules for multiple Pattern Groups, which itself integrate multiple Pattern schedules. The following sections describe how BSS interacts with the Timing Schedule and how this is triggered by external requests.

SIGNAL-BASED RUNTIME DECISIONS

The primary input for runtime decisions concerning which parts of the global schedule should be executed are dedicated binary signals, with valid values being either enabled or disabled. Signal states are exclusively managed by BSS, however it exposes a client API for reading and writing signal states. BSS defines various signal types which can be used for different use cases as well as for access control. Signals of type DISABLED_BY_LSA for instance, can only be set by the Settings Management System itself and not be overridden by client applications.

Basically, BSS performs two categories of runtime decisions. The first one are decisions on Pattern Group level, i.e. which Pattern should be running, which will be described in the next section. The second category are decisions on Pattern level where decisions on alternative paths within a Pattern schedule are made (see section STORAGE RING CONTROL).

PATTERN SCHEDULE EXECUTION CONDITIONS

In order to decide which Pattern schedules to execute, each schedule is bound to an execution condition. Each execution condition is expressed as a propositional formula using signals as statements and logical operators (AND, OR, NOT). Because of their binary nature, signals can be directly mapped to boolean variables. This way, complex execution conditions can be easily built as the following example illustrates:

$$(SIGNAL\_A \lor SIGNAL\_B) \land \neg SIGNAL\_C \qquad (1)$$

If a Pattern schedule is bound to the above condition, it is scheduled for execution if at least one of SIGNAL_A or SIGNAL_B is enabled and Signal_C is disabled.

Signals used in execution conditions are either predefined signals that are independent from current scheduling-related settings in the facility (static signals) or signals defined by the Settings Management System (dynamic signals). Static signals for instance, are typically used for requesting beam, also from custom experiment control systems. It is up to the Settings Management System to supply BSS with execution conditions and dynamic signal definitions.

PERMANENT REQUESTS

With signals being used in execution conditions and BSS providing a simple API for changing signal states, users can enable and disable Pattern executions by simply sending permanent signal change requests. Signal state changes are persisted in a database. Afterwards BSS executes relevant execution conditions and determines which Patterns to enable or disable in the global timing schedule. Finally, it sends commands to the Timing System that switch schedule edges accordingly.

In order to permanently enable an actual Pattern for execution, the default destination from the Pattern's entry node is simply switched from the exit note to its actual timing schedule content (as represented by A in Fig. 3).

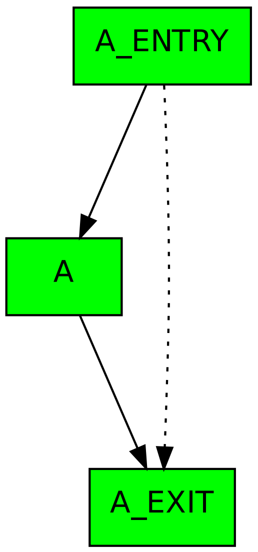

Figure 3: Part of a Pattern Group schedule with Pattern A being enabled.

NON-PERMANENT OR SINGLE SHOT REQUESTS

As described above, permanent signal requests lead to schedules being executed periodically since the resulting default execution path is always a circular subgraph. Therefore, those requests are not adequate for experiments requiring precisely a single shot or any other fixed number of Pattern executions. This is especially true for schedules of short duration, as reaction times of operators (and even automated systems) are threshold-bound and not arbitrarily short. It cannot be expected that an operator waits for a single execution of a Pattern and is able then to disable it before the next execution starts.

To allow for single shots, BSS provides so called non-permanent requests. Whenever a non-permanent request arrives, BSS does not apply the requested signal changes permanently, but builds a signal state snapshot from all existing signal states and applies the non-permanent changes to it. Afterwards, execution conditions of all schedules are evaluated against these snapshots. Schedules being enabled by this evaluation are scheduled to be executed exactly once (Fig. 4).

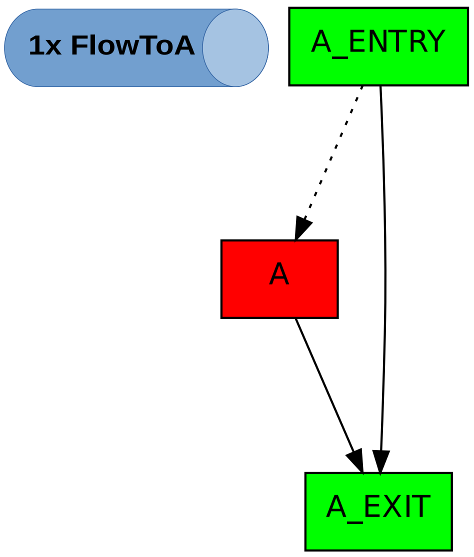

Figure 4: Queued Flow command in A_ENTRY leading to an exactly once execution of A when consumed.

To make this work, the BSS utilizes the ability of the Timing System to define command queues for schedule nodes. To prepare the schedule for non-permanent requests, a command queue is defined for each Pattern's entry node. Whenever a Pattern schedule should be executed exactly once, BSS writes a non-permanent flow command to the corresponding command queue specifying a temporary successor. When the Timing System executes this node, it will consume the command and will not go down the default execution path, but instead to the one specified in the command. Since the queued command is consumed, the prior default path is used again for the next execution.

STORAGE RING CONTROL

In addition to enabling and disabling whole Patterns, BSS also allows for a more fine-grained schedule control, that is extensively used for the Storage Ring Mode features breakpoints, skipping, repetition and manipulation for ESR and CRYRING [3]. To support these features, beam schedules for FAIR can contain sub-schedules with elements/edges that can be directly controlled by users utilizing BSS's signals. The mechanism is simple and is again based on switchable schedule edges which are based on the state of certain dynamic signals. A set of signal types has been added, that allows the definition of signals, from which BSS can directly infer which edges to switch.

To allow for breakpoints, where Pattern execution stops at defined points, the signal type BREAK_ENABLED is connected with a pre-defined break loop in the schedule graph. Setting the corresponding signal allows BSS to enable or disable the breakpoint. The name of the specific breakpoint is encoded in the signal's name, thus allowing for several breakpoints in the same schedule. Knowing the structure of a breakpoint sub-graph, BSS can switch the edges accordingly. This is demonstrated in Fig. 5. If BSS receives a signal with type BREAK_ENABLED and value==ENABLED, BSS firstly decodes the breakpoint Subchain from the signal's name. Afterwards it sets the default execution path from BREAK_ENTRY to BREAK_BLOCK_ENTRY. If the same signal is received with value==DISABLED, it sets BREAK_EXIT as the successor of BREAK_ENTRY, which means that the break loop is no longer executed.

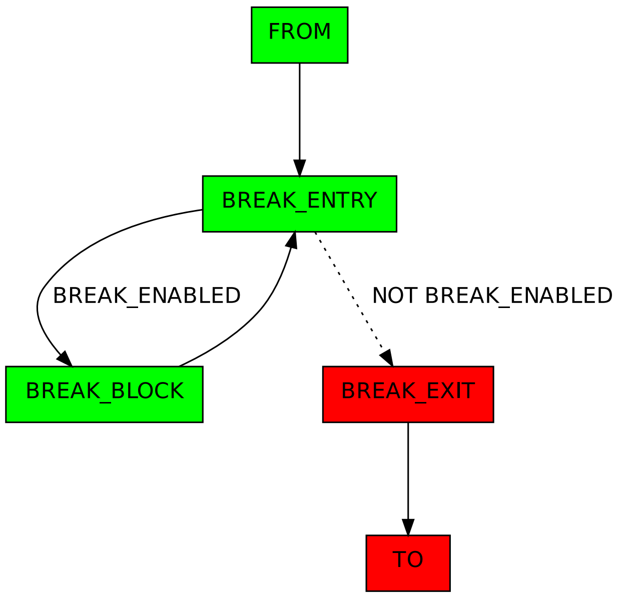

Figure 5: Part of Pattern schedule with an enabled breakpoint.

Skipping of Subchains has been realized using the signal type SKIP_SUBCHAIN. Similarly to skipping whole Patterns as described above, signals of this type are associated with edges that allow circumventing optional Subchains. As all of these features make use of dynamic signals whose initial value is set to be FALSE, the Subchain will be executed and not skipped by default.

Another basic feature of Storage Ring Mode is supporting Subchain repetitions, e.g. for performing a measurement for a defined number of times. With the beginning of Subchain execution in the Timing System, a command queue at an dedicated repetition node is filled with a command that defines the number of repetitions for this Subchain. BSS is responsible for cancelling these repetitions upon user request. This is done by flushing the corresponding queue using the Generator's command interface.

The most distinguished Storage Ring Mode feature is trimming (i.e. manipulation of settings) while a Pattern is running. This manipulation is realized as a loop in the graph that is connected with an exit signal that can be triggered by the user through BSS. While executing the same Subchain over and over again, the user can apply small setting changes to the devices.

For more information about these Storage Ring Mode features, please refer to [3].

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2022). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

BEAM SCHEDULING VS. MACHINE PROTECTION

The Beam Scheduling System is not part of the Machine Protection System of the FAIR facility as it operates on a different time scale than fast hardware protection systems. However, if something goes wrong in the machine, BSS is responsible for preventing Pattern schedules from being executed on a software level. To do so, it continuously processes status messages from the Master Accelerator Status Processor (MASP), which in turn monitors relevant devices and services. Whenever BSS detects, that a Pattern must no longer be executed, BSS disables the associated schedule in the global timing graph. In contrast to Patterns that are just disabled by user requests, for prohibited Patterns, even their entry and exit nodes are excluded from the schedule loop (see Fig. 6). This ensures, that parts of the global schedule which have been prohibited by interlocks, can no longer be enabled by user requests, as long as the causing interlocks remain active. In this regard, interlocks act as an overruling of the signal based execution conditions.

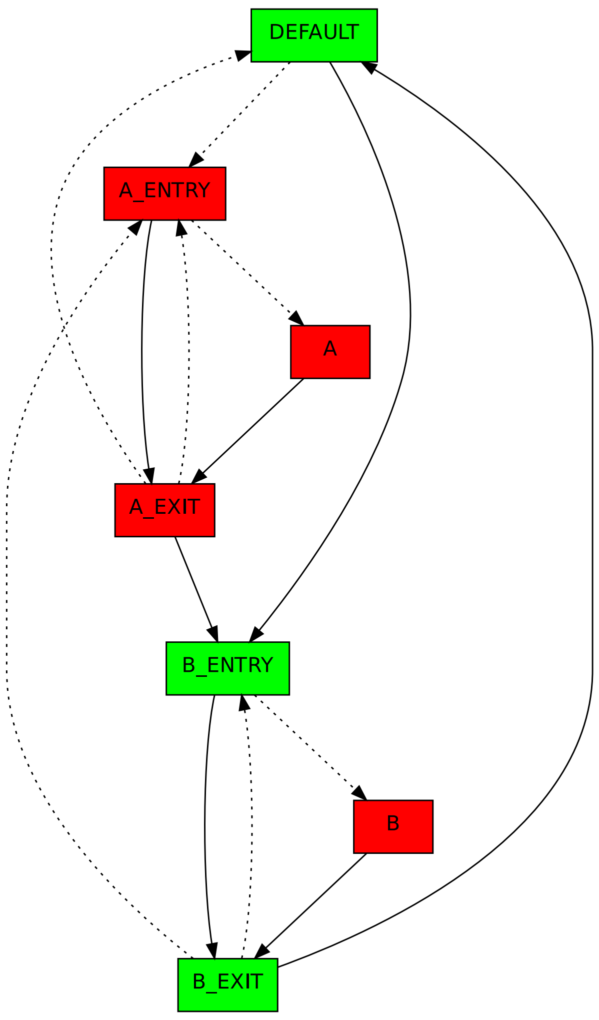

Figure 6: A Pattern Group schedule with Pattern A disabled by the Machine Protection System.

SUMMARY AND OUTLOOK

The Beam Scheduling System, as described in this contribution, has been used productively in several beamtimes. Newer features, like those for Storage Ring Mode operation have been added and it has proven to be a reliable and important building block of the FAIR Control System. Complete integration into the settings management system's data supply process, has allowed for an integrated solution, that makes LSA not only the place for calculating settings for devices, but also the place for schedule planning. BSS on the other hand, creates the global execution schedule and brings it together with runtime information, like user and experiment requests and beam prohibits, which makes it the central instance to decide, which of the preconfigured schedules are to be executed by the Timing System.

Next BSS developments will target requirements coming from the Injector Controls Upgrade project, which will bring the existing UNILAC into the new Accelerator Control System. Challenges are expected to arise from its 50Hz operation and a large number of pre-planned schedules.

REFERENCES

[1] DataMaster Manual, `https://www-acc.gsi.de/wiki/pub/Timing/TimingSystemDataMaster/FTN_dm_schedules.pdf`

[2] DOT Language, `https://graphviz.org/doc/info/lang.html`

[3] R. Mueller, J. Fitzek, H. C. Hüther, H. Liebermann, D. Ondreka, A. Schaller, and A. Walter, "Supporting Flexible Runtime Control and Storage Ring Operation with the FAIR Settings Management System", presented at 21th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'21), Shanghai, China, Oct. 2021, paper WEPV047, this conference.