# AUTOMATED OPERATION OF THE MLS ELECTRON STORAGE RING

Thomas Birke[@], Michael Abo-Bakr, Jörg Feikes, Benjamin Franksen, Michael v. Hartrott, Godehard Wüstefeld – Helmholtz Zentrum Berlin für Materialien und Energie[†], Berlin, Germany

*Abstract*

The Metrology Light Source (MLS) [1] is in user operation since 2008. This versatile facility has to work at wide ranges of operating current and energy as well as different values for the momentum compaction factor according to user demands that vary even on very short notice.

In parallel to machine commissioning, a software system has been developed to control and coordinate the broad manifold of machine states and meanwhile has evolved into an indispensable operator tool acting by itself on demand of a few high level commands.

Actions range from plain device I/O to complex transactions and multiple device I/O. Design goal of the software is to keep and transfer machine and control system within well-defined and consistent states.

## MOTIVATION

The Physikalisch-Technische Bundesanstalt (PTB), a main customer of the BESSY II facility, is the owner of a low energy electron storage ring, the Metrology Light Source (MLS), located close to the BESSY II storage ring in Berlin. The MLS has been designed and built by HZB/BESSY[†] according to the specifications of the PTB and is also operated by HZB/BESSY staff. It offers user service since April 2008 and is now running in routine operation.

Table 1: Machine and Operating Parameters of the MLS

| | |
|---|---|
| Circumference | 48 m |
| Revolution Time | 160 ns |
| Injection Energy | 105 MeV |
| Operational Energy | 105-630 MeV |
| Beam Current | 1 pA-200 mA |
| Values for Momentum Compaction Factor α | $10^{-4}$ – $3x10^{-2}$ |
| Insertion Device | Electromagnetic Undulator 23x180 mm |

Table 1 shows that the MLS has a wide range of operating modes and parameter settings. Additional demands on operating the machine emerge from the use of an electromagnetic undulator. The strong nonlinear fields enforce compensation with correction coils using feed-forward systems, otherwise accumulating and storing beam would not be possible.

A ramping procedure was developed, that keeps the electron beam stored not only when ramping "up" (to higher energy) but also when ramping "down" (to lower energy). This way the energy ramp acts at the same time as a degaussing cycle. But as it does not drive the storage ring magnets into full saturation, some remanent fields cannot be cleared and strongly influence the machine dynamics. As a consequence any error in setting a magnet power supply amplitude or polarity can strongly deteriorate the machine performance and leave the machine in a different state even after completing the time consuming special designed degaussing procedure.

Therefore it is crucial to avoid any operating error when establishing the desired user state in the MLS, which is best realized with completely predefined and automatically performed set up procedures.

Another motivation for a high degree of automation originates from the fact that MLS commissioning work and operation is a service provided by HZB/BESSY to the PTB as a customer service. It should be as reliable and transparent as possible demanding user friendly interfaces and operation definitions.

Operating the MLS includes injecting beam up to a desired current, ramping the energy and adjusting the momentum compaction factor α. All these services require multiple actions to set up the machine for the mode requested by the users (PTB).

Since all signals that are required to determine the necessary steps are available as control system process variables, the decision was made to develop a software system performing the essential sequences of actions to get the machine into the desired states.

## SOFTWARE SYSTEM

### *Finite State Machine*

Based on experiences with smaller applications at BESSY as well as at the MLS, the described software is developed as a hierarchical set of state machines.

Finite state machines (FSM) are a well proven software concept to model and control behaviour of complex systems. A finite state machine – in this case a transducer that converts input (events) into output (actions) – consists of a set of all possible states of the modelled system along with all possible transitions between these states. The transitions are unambiguously performed on conditions associated with input events. Any transition as well as entering a state may initiate output actions. States describe possible situations of the whole system while transitions define when (condition) and how (action) to transform the system into another state.

In a controls application, the input of a finite state machine usually consists of events resulting from

[@] Thomas.Birke@helmholtz-berlin.de

[†] By the merger with the former Hahn-Meitner-Institut (HMI), BESSY became part of the new Helmholtz Zentrum Berlin für Materialien und Energie (HZB)

incoming changes of process variables from the underlying control system (EPICS, Experimental Physics and Industrial Control System), timer-events (esp. timeouts) and of course actions initiated by the user (e.g. using the graphical user interface). The output typically is any sequence of statements/operations limited only by the software environment, but particularly writing new values to the control system (modify process variables) and give feedback to the user.

The very first version of the described program was a simple beam scrubbing automation, Fig. 1. After the MLS had been commissioned to a point where beam-accumulation and ramping to the highest energy was possible, it was important, to keep the beam-current at this high energy above a certain limit during nights and weekends in order to improve beam-conditions.

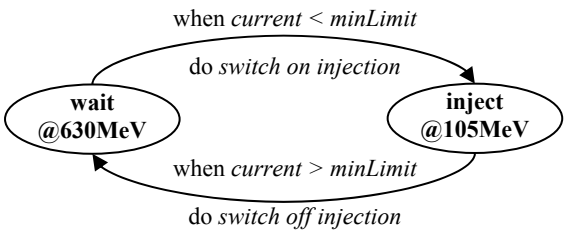


Figure 1: Simplified sketch of the first version.

This software has since then evolved into an important helper application. The state machine as it is currently used was not developed by design according to a full specification. It has undergone an evolutionary process influenced by experiences from machine commissioning as well as from daily use of the application itself. Only by using the application, it is possible to detect situations not yet handled by the state machine (often even by operating errors). The procedures described by the user to solve these problems are then implemented in the state machine and undergo a refinement phase based on the experiences using them. Numerous small development steps have been made, some of which were later removed in favour of alternative solutions or have simply proven obsolete during the commissioning process. A clear view of what actions are appropriate to setup a certain state often eventually arises from formally describing the solution in close cooperation of developer and users/scientists.

During this process, the application became an indispensable instrument performing all standard actions the operator has to take care of. It is an attempt to fill the gap between basic device control and the "one-button-machine".

By now, the main state machine (the *Operation Master*) consists of ~40 states and ~100 transitions. Other state machines, used to control sub-tasks add another ~20 states and ~40 transitions. These state machines not only describe the command-sequences necessary to set up a certain state but also include handling of otherwise unexpected conditions and implement solutions to maneuver out of these exception states into the desired or another safe state.

The whole system was running the MLS without any human intervention for about two weeks during holiday break 2008/2009 and performed well, Fig. 2. The only glitch was a microtron dropout that had to be cared for manually. This particular intervention is now part of the action-sequence to recover from microtron errors in the *Operation Master*, giving an example of how the system evolves by practical use.

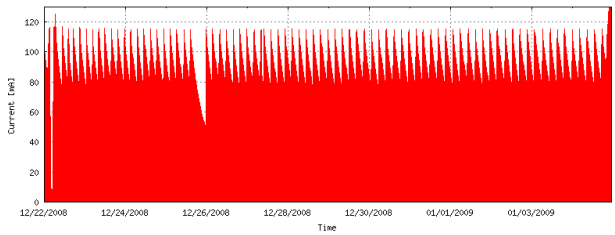


Figure 2: two-week run during holiday break 2008/2009.

## *Energy Ramp*

*Energy Ramp* is the application to ramp the energy of the stored beam. Injection always runs at 105 MeV, and after injection has finished, the machine has to be ramped to the desired energy (up to 630 MeV). The driving parameter is a software parameter that corresponds to the energy. This parameter is smoothly driven to the desired end-value and is used as the input-value to an interpolated breakpoint table for each participating device. The output of the table is the corresponding set point for this device. All devices are synchronously driven to the desired energy minimizing beam-loss.

The breakpoint tables are created with a semi-automated process. The machine is setup for a certain energy, all magnets are hand-optimized and when finished, a special command stores the current settings in the appropriate breakpoint tables.

Ramping usually is as simple as setting the target energy and hitting the "Go" button, but due to hysteresis in the magnets, ramping up and down have to use two different sets of breakpoint tables to not lose beam during ramp. These tables are only allowed to switch at the end-points of the energy ramp, where they map to the same values. Hence ramping to an arbitrary energy may require ramping to the endpoint in previous ramp-direction first before ramping to the desired energy (e.g. 105MeV ↗ 450MeV ↘ 300MeV won't work, must ramp 105MeV ↗ 450MeV ↗*630MeV* ↘ 300MeV instead).

This constraint is not implemented in the *Energy Ramp* itself, but has to be followed by the caller. The *Operation Master* takes care of this, and switches tables properly.

The *Energy Ramp* is one example of a separate application that also uses a state-machine as the controlling entity. All interaction with this application is handled via control system process variables.

## *Optics Change*

Another tool that is currently being developed is an application to change the optics of the machine by modifying the momentum compaction factor α. It will be

very similar to the energy ramp application. The driving value corresponds to the synchrotron frequency.

### *Operation Master*

Both applications will be managed by the *Operation Master* so the operator just defines and sets the main parameters (current, energy, alpha …) and issues the initial command to start transition. The *Operation Master* takes care of all necessary steps to get from the current state to the desired parameters which may involve any subset of injection sequence, energy-ramping and optics-change. It is the central controlling instance keeping track of several decentralized non-linear processes.

## IMPLEMENTATION

The current implementation is a modular application written in Tcl/Tk, which is a proper choice for rapid prototyping and development of an exemplar application including a graphical user interface. But as the system settled and stabilized some drawbacks of a monolithic application became immanent. To avoid conflicts, the system has to ensure that only one instance is actively running, and the current status of the *Operation Master* is only visible on a single screen. Hence the current setup is using one dedicated operator-console to run the *Operation Master* and display the application main window.

### *State Machine*

The state machine module consists of a set of **states** and all **transitions** between these states as well as all possible actions to be performed during transition or when entering a state.

- A **state** is defined by its name, an optional action to be performed when the state is entered and a set of transitions.
- A **transition** is defined by a condition, an optional action and a target state. As soon as the condition gets true, the action is performed and the target state becomes the next active state.

As an example, a state may be coded as follows:

```
state "InjectionRunning" {
    when (current >= maxLimit) {
    } nextState "SwitchInjectionOff";

    when (microtronState == "ERROR") {
        ResetErrorsOnMicrotronPLCs()
    } nextState "RecoverMicrotron";

    when ( timeout(injectionTimeout) ) {
        msg("timeout during injection");
    } nextState "RecoverInjection";
}
```

### *State Engine*

The state engine is the actual processor that runs the state machine. On external events, it checks for all possible transitions of the current state and also manages timeouts. The state engine is just about 10% of the source code and has not been modified for more than a year now.

### *Graphical User Interface*

The current implementation features its own graphical user interface. The operator completely controls and monitors the software using this interface. These 20% of the source code are closely related to the state machine. It not only displays information about the current state, provides acces to all controlling parameters and permits operator interaction if appropriate, but also displays the history of actions and messages and logs these to a file for later analysis.

## FUTURE DEVELOPMENT

To overcome the shortcomings, the software has been redesigned and the application is separated from the graphical user interface.

Work-in-progress is currently to transform the *Operation Master* into a pure headless server process. All interaction with the *Operation Master* will happen through process variables using the control system infrastructure. The user interface will be a control display handled by the control system display manager that can be opened on all displays that have access to control system data (read/write as well as read-only). Control system process variables will be the only communication vehicle to control and monitor behaviour of the *Operation Master*.

Additionally, all other EPICS tools can be used to control and monitor activities. As an example, the alarm-handler can be used to alert/notify operator and others on unexpected events and the archiver may be used to log activity of the *Operation Master* for diagnostic and development purpose. This way, the application integrates very well into the existing control system infrastructure.

Although EPICS provides a powerful tool to implement finite state machines in an extended C syntax, the dynamic object-oriented programming language Python was chosen to implement new version of the *Operation Master*.

## CONCLUSION

The *Operation Master* minimizes errors due to inadvertences and avoids mistakes by taking the load of precisely following complex command sequences off the operator. It also implements standard mechanisms to recover from failure situations as long as no human interaction is necessary.

Experiences and the convincing success of the system are very encouraging to use the same system to develop new core control system components for other currently running as well as upcoming projects at BESSY/Helmholtz Zentrum Berlin.

## REFERENCES

[1] R. Klein et al., "Operation of the Metrology Light Source as a primary radiation source standard", Phys. Rev. ST Accel. Beams 11, 110701-1 (2008)