

# A MODULAR SOFTWARE ARCHITECTURE FOR APPLICATIONS THAT SUPPORT ACCELERATOR COMMISSIONING AT MEDAUSTRON

M. Hager, M. Regodic

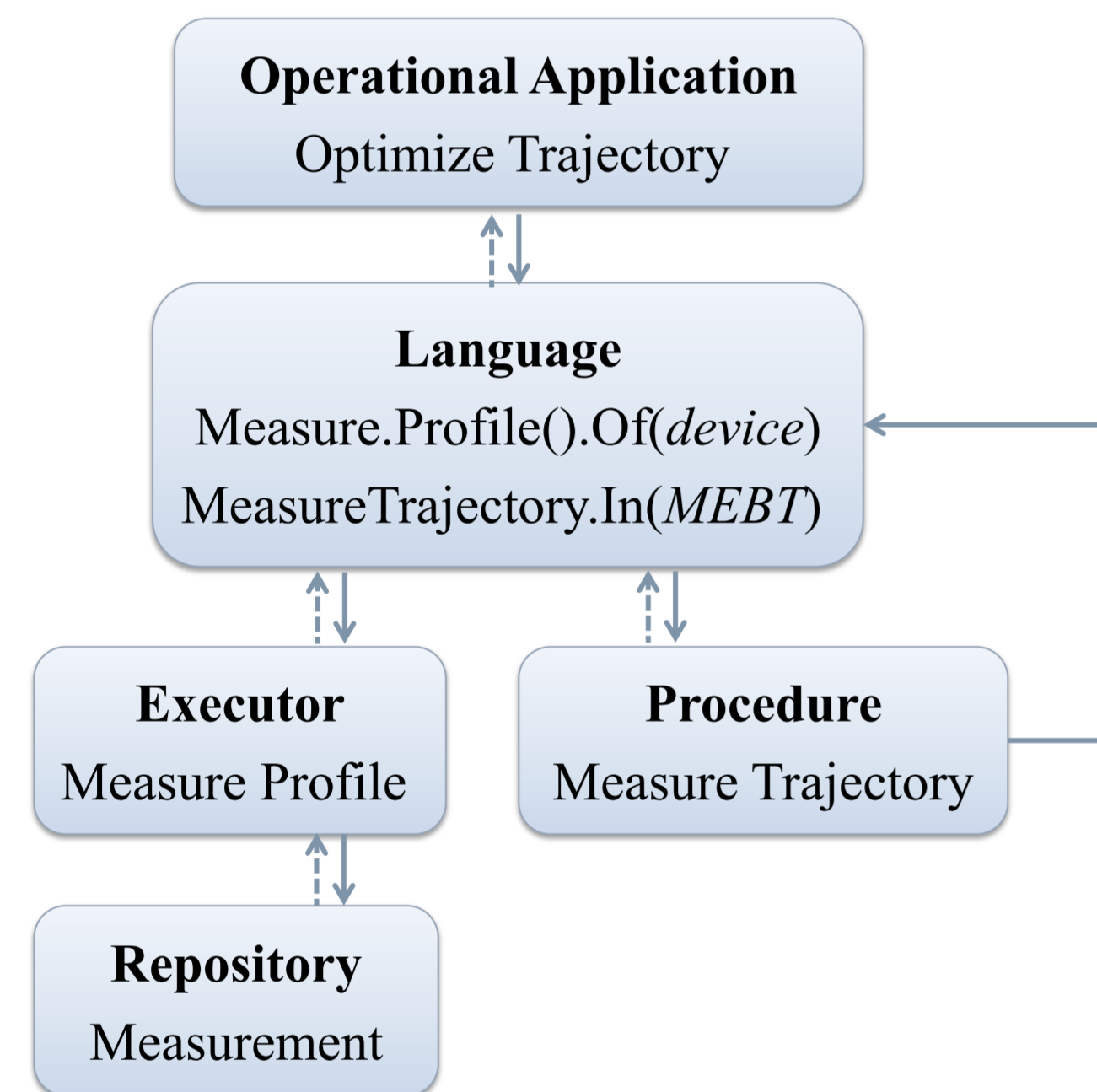
EBG MedAustron, Wiener Neustadt, Austria email: markus.hager@medaustron.at, milovan.regodic@medaustron.at

## Abstract

The commissioning and operation of an accelerator requires a large set of supportive applications. Especially in the early stages, these tools have to work with unfinished and changing systems. To allow the implementation of applications that are dynamic enough for this environment, a dedicated software architecture, the Operational Application (OpApp) architecture, has been developed at MedAustron. The main ideas of the architecture are a separation of functionality into reusable execution modules and a flexible and intuitive composition of the modules into bigger modules and applications. Execution modules are implemented for the acquisition of beam measurements, the generation of cycle dependent data, the access to a database and other tasks. On this basis, Operational Applications for a wide variety of use cases can be created, from small helper tools to interactive beam commissioning applications with graphical user interfaces. This contribution outlines the OpApp architecture and the implementation of the most frequently used applications.

## Architectural Concept

Many beam commissioning activities involve the execution of the same core tasks, for example: "Request an accelerator cycle" or "Take a measurement with a beam diagnostic device". Often the combination of some core tasks is executed as part of other commissioning activities. Based on this realization, the **OpApp architecture enforces a separation of the core tasks into dedicated execution modules and defines a mechanism that allows a flexible composition of the different modules.**



In the OpApp architecture, all execution modules get registered in the OpApp framework. All modules also have access to this registry, to enable access from every module to every module. This allows a flexible composition of modules into bigger modules and applications.

The core execution modules are separated into two different layers. On layer, represented by *Executors*, is specific to devices and data structures. The other layer contains *Repositories* that encapsulates the interaction with connected systems, like MACS or the database. The repositories in this layer work with generic data structures.

## OpApp Language

To allow domain experts to contribute to the development of Operational Applications, the OpApp architecture specifies an own **language**. Via the language the different execution modules can be retrieved from the OpApp framework and be called in an **intuitive way, similar to a natural-language**. The OpApp Language is implemented with a fluent C# API.

```

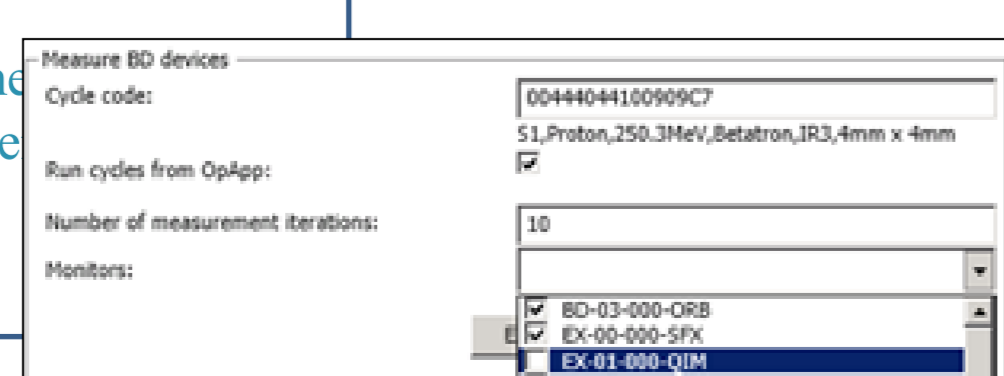
var trajectory = MeasureTrajectory.In(transferLine);
SetStrength.Of(magnet).To(milliard: 0.5);
var newTrajectory = MeasureTrajectory.In(transferLine);
Save(newTrajectory).ToDefaultPath();

foreach (var monitor in transferLine)
{
    var beamProfile = MeasureProfile.With(monitor);
    var position =
        CalculateBeamPosition
        .WithBias(percent: 15).From(beamProfile);
}
  
```

A **common user interface** has been implemented, together with a library of visualization modules for different input parameters. The common interface simplifies the development of OpApps by domain experts. OpApp authors mark the required input parameters in the code with special attributes. The common interface reads the attributes and automatically displays the according fields.

```

[ElementListParameter(AllChecked = false,
    DisplayName = "Monitors", Class = new ElementClass.QIM, ElementClass.ORB, ElementClass.SFX, {})]
public List<BDElement> Monitors { get; set; }
  
```



## OpApp Functionality

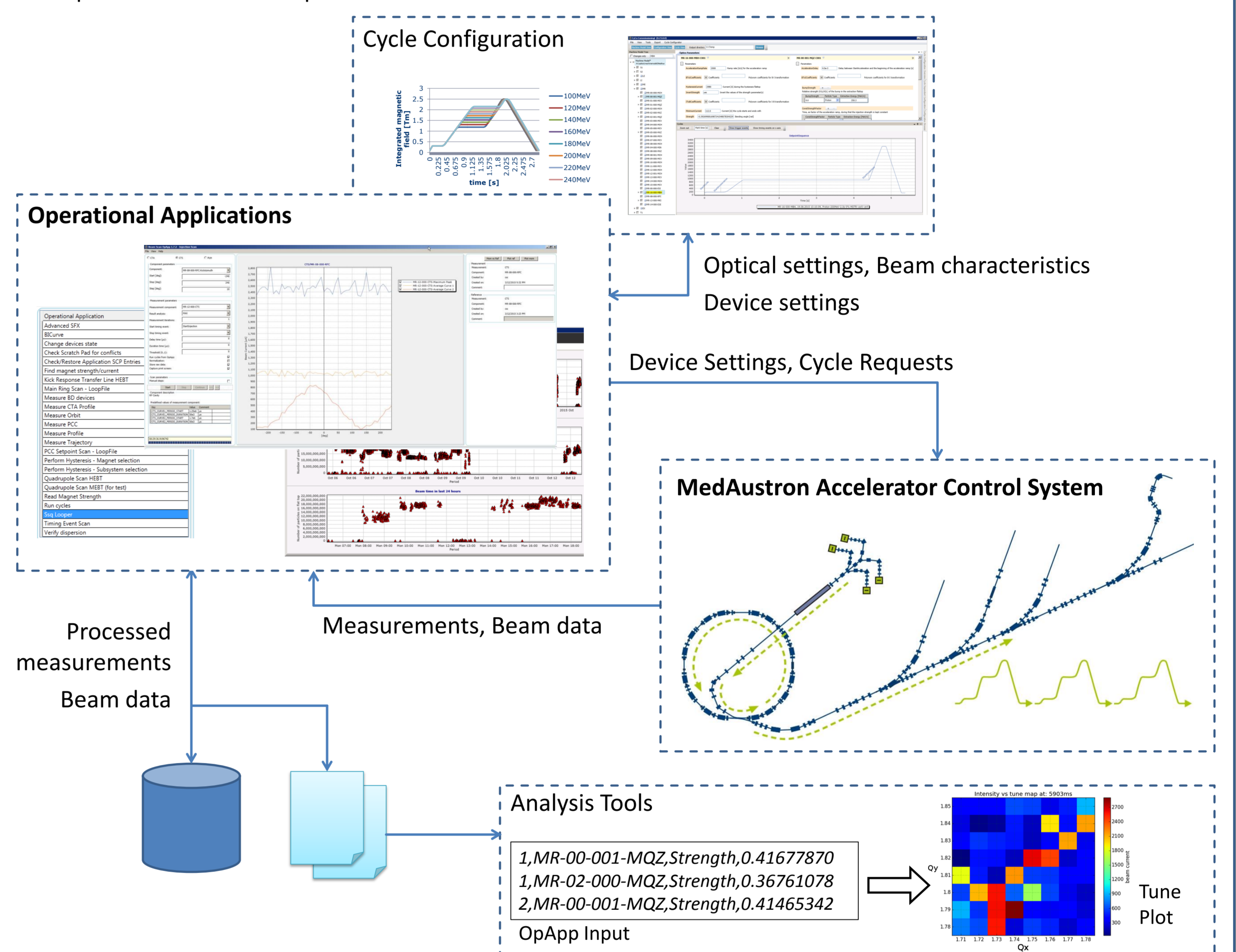
OpApps can compute settings, like currents and voltages, for all accelerator devices based on the optical setup of the accelerator and the desired beam characteristics. OpApps can apply the settings, request beam cycles and measure the characteristics of the generated beam.

The OpApp framework is connected to a database. OpApps use the database to store data related to the beam generation as well as acquired measurements and accelerator configuration. OpApps also retrieve and analyze stored data.

To provide data to other tools, OpApps can generate files in a variety of different formats.

OpApps are used for:

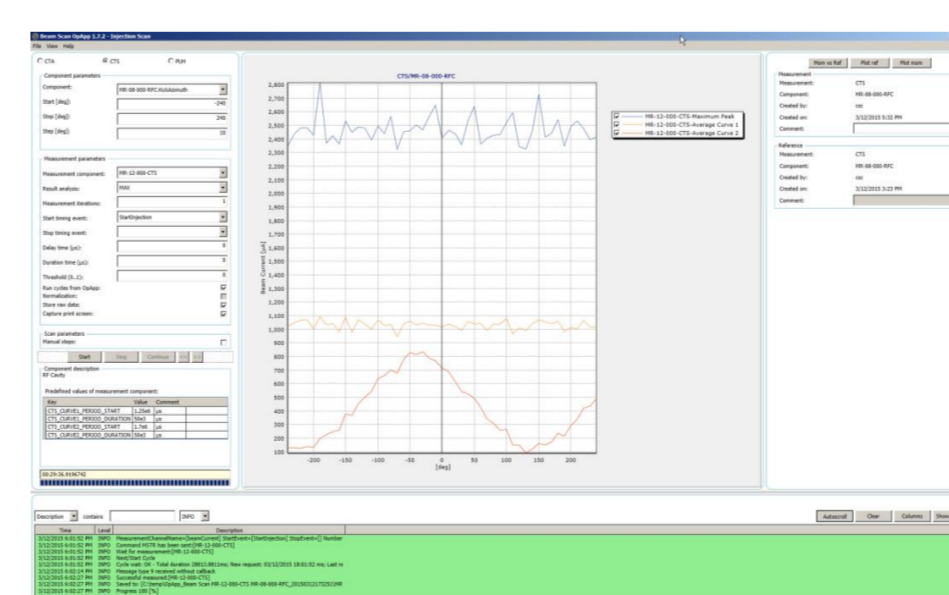
- **Beam Commissioning** - Main domain of Operational Applications
- **Quality Assurance (QA)** - OpApps that acquire measurements and compare them with stored reference data can be used for a regular QA of the beam characteristics
- **Configuration Management** - With a set of database related execution modules OpApps can, for example, help to import device specifications into the database or export stored data for the use in the Control System
- **Accelerator and Beam Monitoring** - OpApps can be used to acquire accelerator and beam data in the background and log this data into the database. Additional OpApps can analyse the stored data and generate reports, for example of the accelerator performance.



## Results

### Beam Scan

In the Beam Scan OpApp, first a device and a beam monitor are selected. Then a range of values for an optical parameter is set. The OpApp runs through all values, computes and applies the according device settings and then measures the beam. In this way the OpApp allows the **determination of the setting that results in the best beam characteristics.**



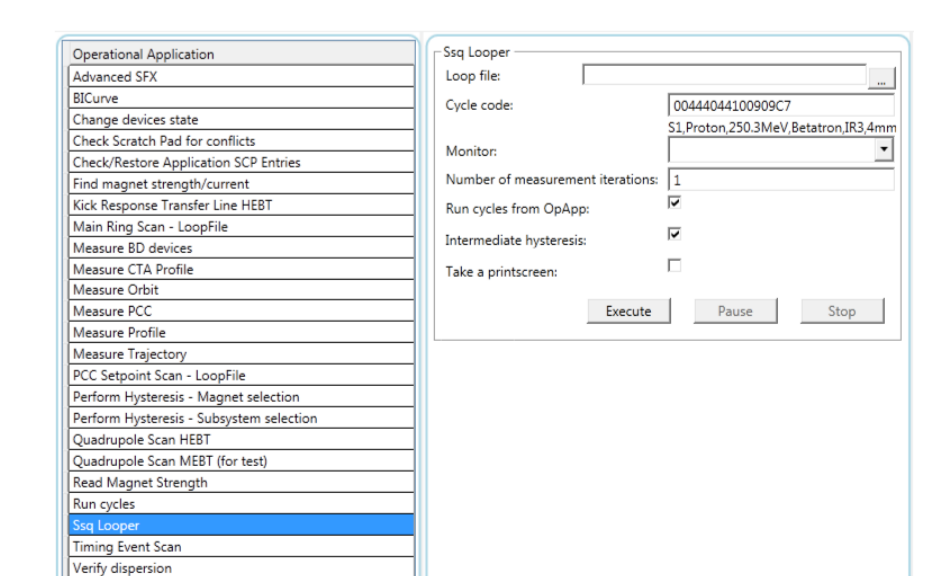
### Particle Logger

The Particle Logger is a Monitoring OpApp. It is composed of a service OpApp and an analysis OpApp. The service parasitically acquires measurements of the beam current and writes them into the database. The analysis OpApp retrieves the data, displays it and **generates performance indicators**, for example about the number of particles in every cycle.



### Atomic OpApps

Atomic OpApps are started via a common, generic user interface. Atomic OpApps use the OpApp language and can potentially be written by domain experts. Examples are: **Trajectory measurement, Kick Response measurement, Execution of beam cycles, ...**



## Conclusion

The OpApp architecture has proven to be an excellent basis for the development of software solutions that support the commissioning and operation of the MedAustron accelerator. The modular design, enforced by the architecture, has shown to allow a very quick adaptation and development of applications.

OpApps already build an indispensable set of tools for the commissioning and operation of the MedAustron accelerator – and their importance will continue to grow, as there are many supportive applications waiting to be developed.