A MODULAR SOFTWARE ARCHITECTURE FOR APPLICATIONS THAT SUPPORT ACCELERATOR COMMISSIONING AT MedAustron

M. Hager*, M. Regodic#, EBG MedAustron, Wiener Neustadt, Austria

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.

INTRODUCTION

The heart of MedAustron is a synchrotron-based accelerator. The accelerator provides the possibility to generate a large range of different ion beams, for example Proton and Carbon beam with different beam sizes and hundreds of different energies. A backside of synchrotron-based accelerators is the high number of components and the complex control of the beam. The commissioning of such an accelerator is a laborious task that requires support by software to be executed efficiently.

One challenge in the development of such applications is the agility of the commissioning process. While the commissioning progresses, new components are integrated, existing ones change and the understanding of the accelerator and the beam behavior deepens. Software that supports the process has to be easily adaptable to the evolving environment. Previous architectural concepts turned out to be too restrictive for these demands. Therefore, a new, modular architecture has been developed at MedAustron, the OpApp architecture.

APPROACH

The OpApp framework is connected to the MedAustron Accelerator Control System (MACS), [1]. With the connection to MACS, OpApps can: A) Send commands to the systems and devices of the accelerator B) Apply device settings and retrieve state information. C) Request beam cycles D) Receive measurements and timing information.

The framework is also 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.

Functionality

Main domain of Operational Applications is the beam commissioning. 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 also apply the settings, request beam cycles and measure the characteristics of the generated beam. With this, OpApps can automatize many aspects of the measurement-based beam commissioning workflows. The role of Operational Applications in the workflow is depicted in Figure 1.

OpApps contribute to the processing and analysis of measurements but usually leave complex analysis tasks to dedicated tools. To provide data to other tools, OpApps can generate files in a variety of different formats.

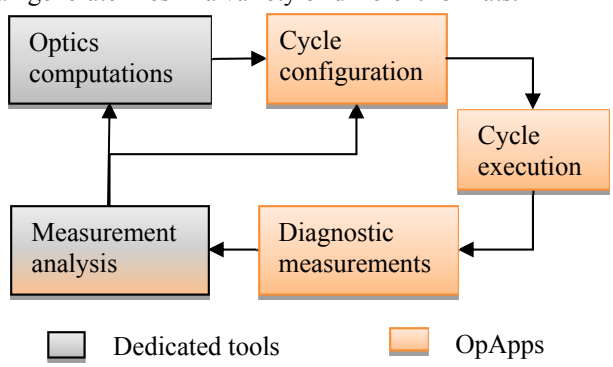

Figure 1: The measurement-based beam commissioning workflow

Although the OpApp concept was developed with beam commissioning in mind, OpApps are by far not limited to it. Operational Applications can also be used for:

- 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 the user to import device specifications into the database or export data for the Control System
- Accelerator and Beam Monitoring - OpApps can be used to acquire accelerator and beam data in the

* markus.hager@medaustron.at
# milovan.regodic@medaustron.at

ISBN 978-3-95450-148-9

background and log this data into the database. Additionally OpApps can analyse the stored data and generate reports, for example of the accelerator performance.

OPAPP ARCHITECTURE

Most 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 builds an activity that is executed as part of other commissioning activities. A trajectory measurement, for example, requires a series of cycle requests and beam position measurements. The trajectory measurement itself is used in a trajectory steering where it is combined with the application of device settings, i.e.: apply settings, measure the trajectory, (if necessary) apply better settings, and so on. Based on this realization, the OpApp architecture enforces a separation of the core commissioning tasks into dedicated modules and defines a mechanism that allows a flexible composition of the different modules.

Core Execution Modules

The core execution modules of the OpApp architecture 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.

An example for the separation into the two layers is the execution of beam diagnostic measurements. For each group of beam monitors an own Executor is implemented that handles the specifics of the measurement, for example a beam intensity measurement or a profile measurement. All measurement executors, however, use the same measurement repository. The measurement repository implements the generic access to the measurement interface provided by MACS. Figure 2 shows the main measurement executors and the repositories they use.

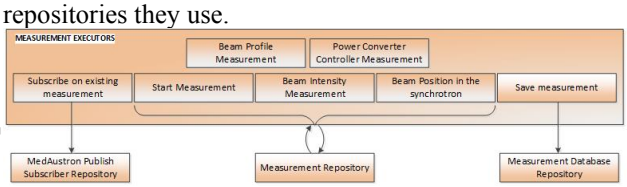

Figure 2: Measurement Executors and the Repositories they use

Changes in the interface of the connected systems only affect the repository layer but are not reflected into the OpApp business logic, which results in highly maintainable code. The centralization of the accessing logic also has another big advantage: It provides a way of testing operational applications without being connected to the real system. To test OpApps, "Demo" repositories are implemented that simulate the response of the different systems. The OpApp framework can switch, with a simple flag in the code, from the "live" repositories to the "demo" repositories. After the switch to demo mode the OpApps work as before but on simulated connections.

Module composition

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.

OpApp Language

Operational Applications are usually developed by the Controls team on request of domain experts. Often the requested functionality is an automatization of simple but time consuming routines, like the acquisition of an extensive set of measurements. If domain experts could write these routines themselves, they wouldn't have to request their development and wait for the implementation.

To allow domain experts to contribute to the development, the OpApp architecture specifies the implementation of an own OpApp language. Via the language the different execution modules can be retrieved from the OpApp framework and called in an intuitive way, similar to a natural-language. The trajectory measurement module, for example, can be executed with the following line of code:

MeasureTrajectory.In(*beamLine*).

Figure 3 shows the concept of the OpApp language and the composition of the execution modules.

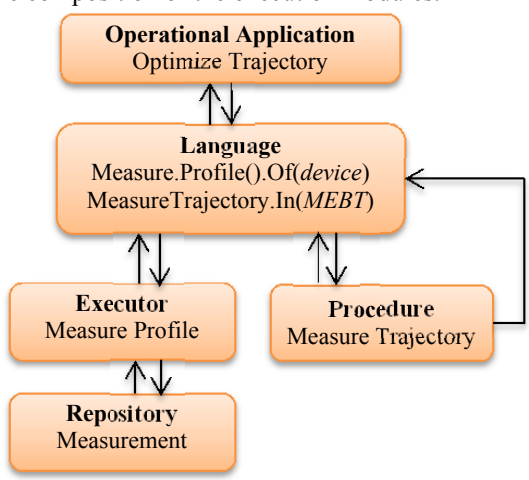

Figure 3: Execution modules and their composition via the Language

Accelerator Models

OpApps require information about the accelerator. They must know which elements are available and which properties of these elements they can influence. OpApps also require information about beam optical parameters, for the computation of device settings. To include this

ISBN 978-3-95450-148-9

information into the framework, the OpApp architecture uses several models. The code of all models is generated from the accelerator configuration, to allow an adjustment of the code when changes in the configuration occur.

One of the models is the accelerator model. This model contains the elements of the accelerator and their controllers, for example:

- S1-01-000-MBH (CS-03-031-PCC)
- S1-00-000-MCX
  - S1-00-000-MCH (CS-03-211-PCC)
  - S1-00-000-MCV (CS-03-246-PCC)

From the model, single elements can be retrieved but also all elements that belong to a certain part of the accelerator or a certain device group. The model information is available in the Language, which allows commands like: SetCurrentOf(S1-01-000-MBH).To(20);

IMPLEMENTATION

Operational Applications are developed in a Microsoft .Net environment with C#.

Language

The OpApp Language is implemented with a fluent API. A fluent API is a programming interface that employs mechanisms like method chaining to allow the creation of highly readable code.

Figure 4 shows an example for the implementation of the fluent API. In the example an OpApp sets the current for a magnet. The OpApp calls:

SetCurrent.Of(*magnet*).To(*current*).

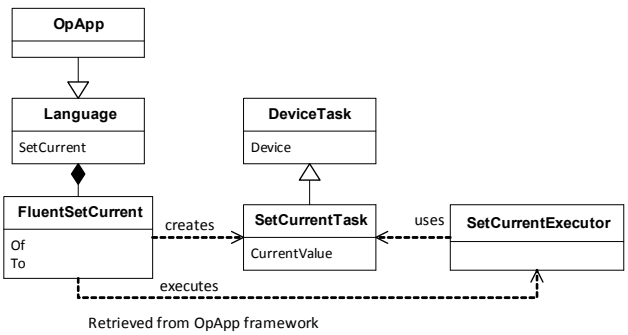

Figure 4: Implementation of the fluent API

The OpApp framework manages all execution modules in a Dependency Injection container. For the execution of the SetCurrent command, the Language retrieves the SetCurrentExecutor from this container. The name of the magnet and the current to be applied are stored in a SetCurrentTask object. This object is forwarded to the executor in the To() method. The executor uses an according repository to apply the current to the device via MACS.

User Interfaces

Graphical User Interfaces (GUI) for Operational Applications are developed with the .Net WPF framework.

In applications that require a high interaction with the user, the GUI can get quite complex. This means that a lot of interaction logic must be implemented. In addition, GUI-based OpApps require asynchronous mechanisms to interact with the execution modules. GUI-based OpApps are, therefore, developed by software engineers.

OpApps that execute simpler routines only require some input values but not a complex user interface. To allow the creation of simpler, language-based applications by domain experts, a common user interface has been implemented together with a library of visualization modules for different input parameters. OpApps authors mark the required input parameters in the code with special attributes. The user interface reads the attributes and automatically displays the according fields. Figure 5 and Figure 6 show an example of the code attributes and the resulting display in the user interface.

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

Figure 5: Example of parameter attribute

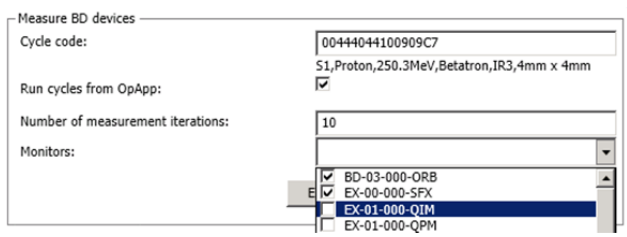

Figure 6: Automatically generated user interface

RESULTS

Around 1.5 man years of work have already been spent on the development of Operational Applications. During this time the framework has been developed, together with about 25 *Atomic* OpApps, one *Complex* application and one *Monitoring* application.

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 for routines executed by Atomic OpApps are trajectory measurements, kick response measurements and checks for incorrect device settings. Figure 7 shows an example for a trajectory measurement written with the OpApp language.

```
var trajectory = CreateEmptyTrajectory();
foreach (var monitor in ProfileMonitors.AllIn(transferLine))
{
    var beamProfile = MeasureProfile.With(monitor);
    var position =
        CalculateBeamPosition
          .WithBias(percent: 15).From(beamProfile);
    trajectory.AddPosition(postion);
}
```

Figure 7: Example for a fluent trajectory measurement

ISBN 978-3-95450-148-9

Complex OpApps

Complex OpApps are interactive applications that are operated via their own user interfaces. This type of applications uses the execution modules but not necessarily the language.

One Complex application has already been developed, the Beam Scan OpApp. Main functionality of the OpApp is the application of device settings in a given range and a subsequent measurement of the beam. In this way, the OpApp allows the determination of the setting that results in the best beam characteristics.

The Beam Scan OpApp has turned out to be extremely helpful. The application supports the scan of many optical parameters and a wide range of device, monitor combinations. 8 shows a scan of a synchrotron RF system setting with a beam current measurement.

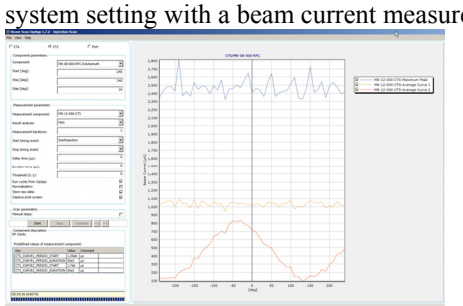

Figure 8: Screenshot of the Beam Scan OpApp

Monitoring OpApps

Monitoring OpApps continuously collect and store machine and beam related data. They run on dedicated machines and don't require any user interaction.

The Particle Logger is the first Monitoring OpApp. The application is composed of two parts. One part parasitically acquires measurements of the beam current in the synchrotron and logs this information together with beam cycle data into the database. The second part is an analysis application that retrieves this data, displays it and generates performance indicators. One indicator for the accelerator performance is the number of cycles in a given period in that more than a certain number of particles were measured in the beam. Figure 9 shows an example of a chart that contains the number of particles in the cycles generated over 24 hours.

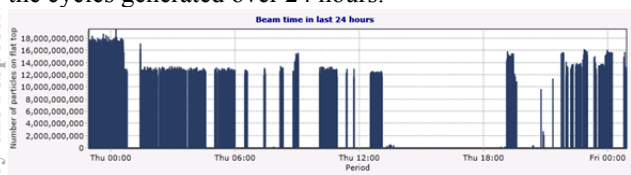

Figure 9: Screenshot of the Particle Logger Analysis application

Operational Applications have also been presented at the IPAC 2015. The according contribution, [2], contains further OpApp examples.

OUTLOOK

The OpApp development at MedAustron will continue. The OpApp framework will be extended and improved, for example, with a synchronization mechanism between asynchronously executed tasks.

Despite of the availability of the fluent language, OpApps have so far only been developed by software engineers. In order to encourage the participation of domain experts, an out-of-the-box development environment and user guides will be prepared.

Upcoming OpApps will focus on advanced beam commissioning tasks and on aspects of the accelerator operation, mainly: Quality Assurance, Configuration Management and Accelerator Monitoring.

With the integration of dosimetric measurement equipment and the Treatment Control System, OpApps could in the future also be used for treatment-related QA procedures and allow the tuning of the accelerator from a treatment perspective.

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.

ACKNOWLEDGEMENTS

We would like to thank A. Wastl (MedAustron) for his important contributions to the OpApp development and K. Fuchsberger (CERN) for his help in working out the initial OpApp concept. We would also like to thank J. Junuzovic (MedAustron) for her support.

REFERENCES

[1] J. Gutleber, et al., The MedAustron Accelerator Control System, Proceedings of ICALEPCS 2011, Grenoble, France

[2] A. Wastl, M. Hager, M. Regodic, Operational Applications – A Software Framework used for the Commissioning of the MedAustron Accelerator, Proceedings of IPAC 2015, Richmond, VA, USA

ISBN 978-3-95450-148-9