REPLACING CORE COMPONENTS OF THE PROCESSING AND PRESENTATION TIERS OF THE MEDAUSTRON CONTROL SYSTEM

A. Höller[†]*, S. Vörös[‡]*, A. Kerschbaum-Gruber, C. Maderböck, D. Gostinski, L. Adler, M. Eichinger, M. Plöchl, EBG MedAustron GmbH, Wiener Neustadt, Austria

Abstract

MedAustron is a synchrotron-based ion therapy and research facility in Austria, that has been successfully treating cancer patients since 2016. MedAustron acts as a manufacturer of its own accelerator with a strong commitment to continuous development and improvement for our customers, our users and our patients. The control system plays an integral role in this endeavour. The presented project focuses on replacing the well-established WinCC OA [1] SCADA system, enforcing separation of concerns mainly using .NET and web technologies, along with many upgrades of features and concepts where stakeholders had identified opportunities for improvement during our years of experience with the former control system setup for commissioning, operation and maintenance, as well as improving the user experience. Leveraging our newly developed control system API, we are currently working on an add-on called "Commissioning Worker”. The concept foresees the functionality for users to create Python scripts, upload them to the Commissioning Worker, and execute them on demand or on a scheduled basis, making it easy and highly time-efficient to execute tasks and integrate with already established Python frameworks for analysis and optimization. This contribution outlines the key changes and provides examples of how the user experience has been improved.

MEDAUSTRON

MedAustron is a synchrotron-based ion beam therapy and research centre located in Austria. The facility has been treating cancer patients since 2016, currently with the use of protons and carbon ions. In parallel to medical and research operation, MedAustron is also working on development and improvement projects, like commissioning the accelerator for use with a helium beam. MedAustron acts as a manufacturer of not only its own particle therapy accelerator, but also of further ion beam centres.

In consequence of being a manufacturer of multiple facilities, MedAustron has taken the decision to exchange some components of the MedAustron Control System (MACS) and enhance it with components more suitable to conform with the challenges of operating and further developing multiple facilities and the additional use case of initial commissioning (component commissioning, accelerator commissioning, beam commissioning) of new facilities.

* These authors contributed equally to this work
† angelika.holler@medaustron.at
‡ sandor.voros@medaustron.at

MEDAUSTRON CONTROL SYSTEM

The MedAustron accelerator delivers proton and carbon ion beams for cancer treatment and research to four irradiation rooms including 3 horizontal, 1 vertical and a gantry beam line. In order to deliver beam from the source to the treatment room, the particle accelerator consists of:

- ~300 power converters controlling ~350 magnets
- ~270 vacuum devices
- ~80 beam diagnostics devices
- 2 RF systems and amplifiers
- 3 ion sources

The MedAustron Control System (MACS) creates a framework for all accelerator devices by providing a standardized set of essential “services” and interfaces in various tiers [2].

Architecture

The architecture extends the industry best practice, 3-tier model [3, 4] in accordance with [5]: (1) presentation tier, (2) processing tier, (3) equipment tier and (4) frontend tier. Components in separate tiers that are distributed over a number of processing devices communicate with each other through a dedicated Ethernet network. Communication between equipment tier and frontend tier may also be achieved through dedicated field-bus and custom links, depending on the imposed constraints. [2] (see Fig. 1)

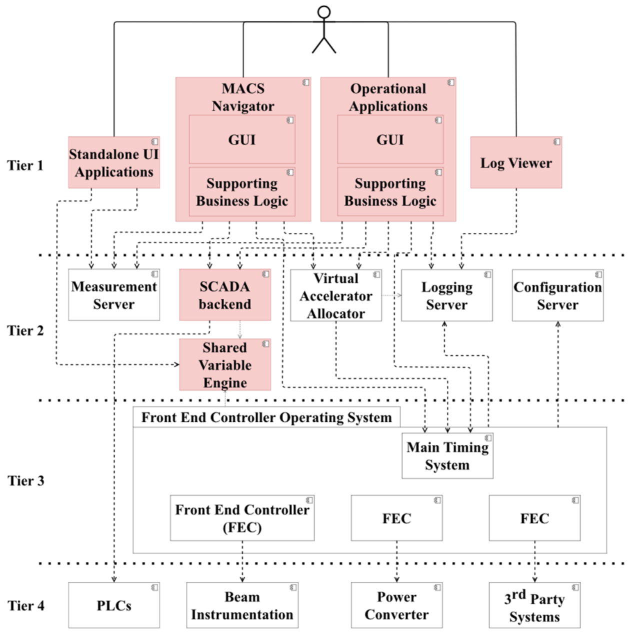

Figure 1: Architecture of the former MedAustron Control System (MACSv1).

NEED FOR A CHANGE

Although the accelerator's architecture is capable of covering all current use cases, new needs have emerged, requiring the redesign of upper-tier core components:

The **MACS Navigator**, which serves as the primary user interface, is implemented in WinCC OA and performs functions related to machine preparation, service, and basic operation. Although the initial concept aimed to separate the business logic from the presentation layer, certain essential scripts have become closely intertwined with the user interface.

From the outset, the control system was designed to facilitate archiving. However, the COTS **archiving** function into an Oracle DB [6] appeared somewhat problematic.

In the course of developing the MACS Navigator, **standalone user interfaces**, written in LabVIEW, had been implemented as an alternative. These interfaces communicate with the front-end controllers via the Shared Variable Engine, bypassing the SCADA backend.

The **Log Viewer** tool aids the user in monitoring log messages sent by underlying devices. In an unfiltered view the volume of log messages is overwhelming. Setting up a certain filter is a repetitive, manual task.

The **Operational Application** [7] is a highly complex framework that encompasses acquiring a single measurement from a specific device and executing fully automated tasks for specific user-defined purposes. The framework and applications are developed in .NET C# and necessitate in-depth programming expertise.

Tier 3 generates a uniform view of the diverse devices at tier 4 by introducing autonomously working front-end controllers (FEC) that all follow one design pattern. The Framework as well as the front-end controllers are written in LabVIEW. The front-end controllers are controlled and monitored via the SCADA backend. Due to technological constraints, the communication takes place via numerous different native protocols (NI-PSP [8] → **Shared Variable Engine** → OPC DA → WinCC OA Distributed Network Protocol), which leads to increased maintenance efforts and security flaws.

The architecture has been designed to address **cyber security concerns**, which have been exclusively included at the IT infrastructure level. As cyber security has gained significance, managing the overall close integration between tiers 1 and 2, the incorporation of identical interfaces for every upper-tier product, and ensuring security for specific communication protocols have become challenging. The same change in significance is valid for **remote** device or accelerator **commissioning**. While a secure VPN connection is a secure initial step, it cannot offer varied access control levels and potentially grants immediate entry to an entire network.

In addition to the above-mentioned required changes MedAustron has taken advantage of this redevelopment opportunity to improve the user experience with state-of-the-art technology and to simultaneously minimise maintenance costs by implementing in-house developed tools that increase customisability. Requisite core components, such as WinCC OA's SCADA, have been exchanged in close collaboration with Cosylab.

Overview of Architectural Changes

The **User Interface Client** is a web application built on top of a single page web application framework hosted on an IIS webserver [9]. The architecture of the user interface application is feature-oriented. This splits functionality into logical units that address different requirements. Each feature is composed of different reusable components, widgets and services that are provided by core and shared modules to the whole application. The client is either accessible via a standard web browser, or an installed Electron client [10], which unleashes its benefits when using a multi-monitor setup. The web application is written in Angular [11] and makes use of COTS components, e.g. Kendo-UI [12] and Apache ECharts [13]. The responsive design allows the usage of desktops and mobile devices.

Tier 2a exposes the MACS UI functionality to the end user clients in the internal and external network. It acts as a gateway between the user client and tier 2b, the data layer (see Fig 2).

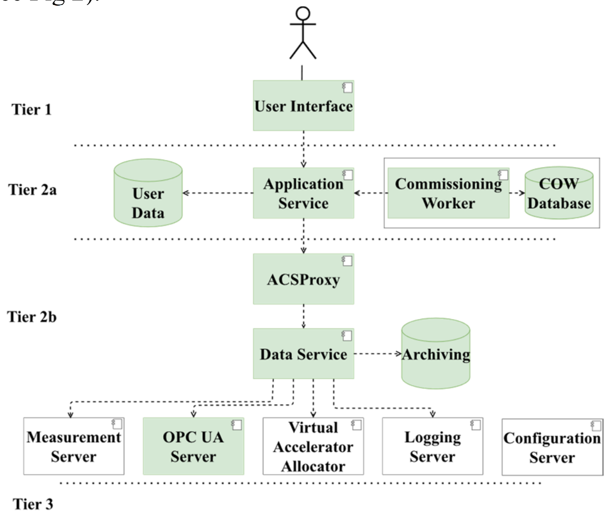

Figure 2: Overview of the architecture of MACSv2.

The **Application Service,** written in .NET C#, is accessible by the User Interface Client over REST API using secured HTTPS and WSS protocols. SignalR is used as the main data exchange communication service. It provides real-time web functionality to the MACS UI Web Application. The **User Data Storage** is used for settings and configuration that has to be global to all users on all devices. A typical use case is to restore user configuration even if the application is accessed from a different machine.

The **Commissioning Worker** provides the possibility of writing procedures with the ability to directly control and monitor the accelerator, see the following section for more details.

Tier 2b is responsible for exposing the process data from the data sources and the stored data from the data storage. To simplify handling different types of data, a generic approach is used, where each piece of data, called a data fragment, can be accessed in the same manner, regardless of the data source specifics. Data access procedures are generalized as read, write, browse, subscribe, unsubscribe and invoke operations. Such a narrow set of operations significantly reduces the complexity while still exposing all required functionality of the data sources.

The **ACSProxy** is a component protecting central services of MACS from unintended access. It serves as a gateway and single point of communication between lower-level supervisory control system functionality, which enables it to generically monitor and control all commands and requests, and the upper tiers of the control system. As a gateway, it is also well suited to be placed at a network border which enables it to facilitate a secure network architecture. The ACSProxy receives generic requests from the Application Service components, which contain the service the component is trying to reach, the operation and its related payload. The ACSProxy then compares this generic request against a configurable set of rules and forwards it if granted. The ACSProxy is classified as medical software class C.

The central functional component of tier 2b, the **Data Service**, consolidates the information produced by other tier 2b, or tier 3 components to expose the data in the data layer with a unified interface. The Data Service directs requests to a specified component for processing and also publishes messages back to subscribers on a monitored property change. Measurements are sent from tier 3 to the upper tiers via the Measurement Server in form of data chunks, the same applies for log messages via the Logging Server. The Data Service receives all measurements and log messages and distributes them based on subscriptions to the Application Service and stores them in a TimescaleDB for archiving purposes [14].

One of the challenges during the exchange of WinCC OA was the search for a solution for middleware communication between front-end controllers and the upper tiers. The COTS product Atvise from Bachmann Visutec [15] was chosen, as its core is based on **OPC UA** and supports object-oriented configuration. Besides the OPC UA server capabilities, the Atvise connect subproduct was chosen for connecting PLCs. The communication in all directions is signed and encrypted. To correlate measurements and log messages with front-end controller states, a selected subset of OPC UA nodes is archived through the Data Service as well.

Overview of User Interface Changes

The changes on interface level have been introduced in close collaboration with the Operations team to enable a smooth transition between the two systems, while enhancing functionalities or adding missing functionalities which might have not been reported over the years. Figure 3 compares the Power Controller device expert panel: MACSv1 is visible in the top image, whereas the lower image depicts the new user interface. The following changes have been introduced:

- Incorporated logging for every panel with device dependent custom filter
- Introduced tabs in windows
- Last open tabs are automatically restored from local user data storage after panel restart
- Full text search for every tree view
- Full text search provided by web browser
- Text based content (controls and indicators as well) can be selected and copied to clipboard
- Archived measurement can be retrieved and displayed in the same panels as "live" measurement
- No hidden functionalities (right click, etc.) to support mobile devices
- Collapsed expert functionalities to set focus on basic, vital features
- Incorporated standalone UIs

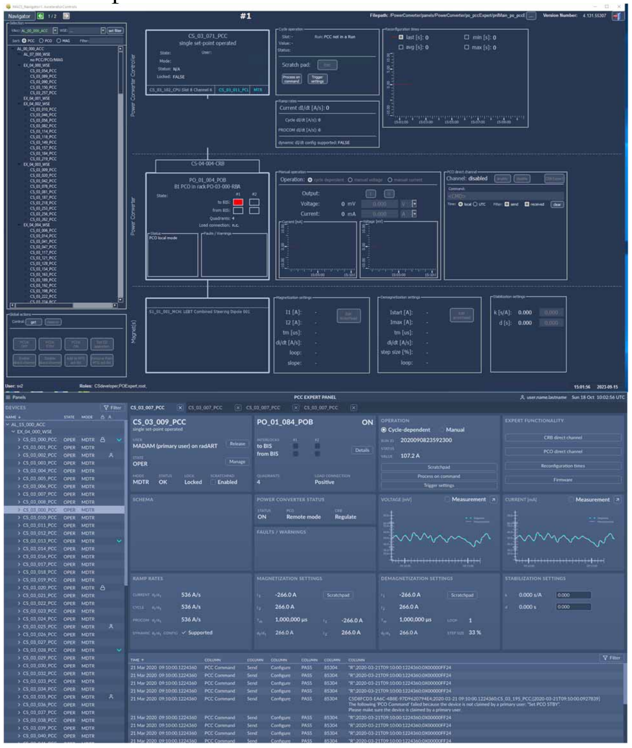
Figure 3: Power Converter Expert panel in MACSv1 (top) and the equivalent panel in MACSv2 (bottom).

COMMISSIONING WORKER (COW)

The Commissioning Worker is an application that interprets Python and is smoothly integrated into the control system architecture and its user interface. COW provides the possibility to write Procedures with the capability of directly controlling and monitoring the accelerator and integrates additional Python libraries and applications (e.g. for measurement data analysis or calculation of machine parameters). It provides the possibility to either write and execute Procedures via the UI, or integrate existing Procedures and Frameworks from a Git repository.

Table 1 describes important COW-related terminology.

Table 1: Definition of COW Terms

Term	Definition
Frame-work	A Framework is a version-controlled collection of Python and support files. An example of a MedAustron Framework is PACMAN (Python Algorithms Coded for Measurement data ANalysis) [16].
Proce-dure	A Procedure is a Python script that can be executed in the COW. Procedures make use of the functionality provided by Frameworks or other Procedures.
Task	A Task can be executed/scheduled/triggered in the COW. It is the combination of a Procedure with input parameters and settings.
Task Ex-ecution	A Task Execution is an instance of a Task.
Result	A Result is an artifact produced as an output of a Task Execution.

Taking the following facts into consideration, the decision was taken to pursue the development of COW:

COW connects two worlds:

- The development of MACS as part of a medical product requires software development according to the standard for medical device software [17]. This results in high-quality documentation and thorough validation, but it also entails a prolonged Time-to-Deployment, strict change management and development within a dedicated Control System development team.
- The creation of Procedures for various purposes during operation, commissioning, or ad-hoc-troubleshooting demand more flexibility in the development process, its use cases ranging from one-time use Procedures assembled during a commissioning shift to fully validated Procedures executed during a daily quality assurance (QA). Development can ideally be done by the expert user (e.g., a physicist responsible for a commissioning task).

In the previous version of the control system, the concept of the Operational Application Framework [7] had the aim to bridge this gap, but in the end proved to be too complex in its implementation. Consequently, development of additional Operational Applications still ended up in the software development team.

The replacement of core components of MACS in combination with the initial commissioning use case of additional facilities would have required a complete rework of the Operational Applications Framework and the existing Operational Applications.

Multiple Frameworks in Python are already in use at MedAustron, and the experience from the last years showed that the manufacturer team at MedAustron has both the motivation and the knowledge to further extend those Frameworks.

With the MACS upgrade the control system already provides the Application Service's API gateway, an interface to the lower tiers of the control system, foreseen to be used by panels and scripts.

Architecture

The **COW web client** is implemented as part of the existing MACS UI Angular web app. Existing patterns and Frameworks are reused to implement the new COW components.

The **COW Gateway** distinguishes incoming requests by type and forwards them to internal services. The execution of Tasks is delegated to the COW Execution Engine.

The **COW Execution Engine** manages the execution of Procedures. It receives execution requests from the COW Gateway, either for scheduled, event-based or manual executions. It collects all required Frameworks and scripts for Task Execution, then creates the so-called execution environment which runs the COW Python.NET. The COW Execution Engine monitors all running Tasks and reports state changes to the COW Gateway. The COW Gateway is responsible to inform subscribers, e.g. web clients, about the state change.

COW Python.NET is a .NET application for interacting with Python. It initializes the Python Engine of the Python.NET package. The package uses the locally installed Python version to execute scripts inside a .NET context. The Python.NET process waits for the execution to finish and forwards possible exceptions to the governing process. Any other communication of the script runs via PAAPI and the API Gateway.

PAAPI (Python Accelerator control system Application Programming Interface) is a Python Framework that acts as an abstraction layer of MACS UI's API Gateway, focussed on the end users' needs.

Python Frameworks and Procedures are stored in a Git repository. At Task creation the user has the opportunity to choose the version or branch of the Procedure and the involved Frameworks via the web client. Simple ad-hoc Procedures can be modified directly via the web client.

The **Procedure Store** is a collection of Git repositories that makes all the frameworks and procedures accessible to the Execution Engine

The **COW Database** is a relational database where the Result data is stored.

Sample Procedure for Multi-Energy-Steering

A common beam commissioning task is the alignment of the beam to pass through the optical centre of multiple consecutive quadrupoles for several extraction energies.

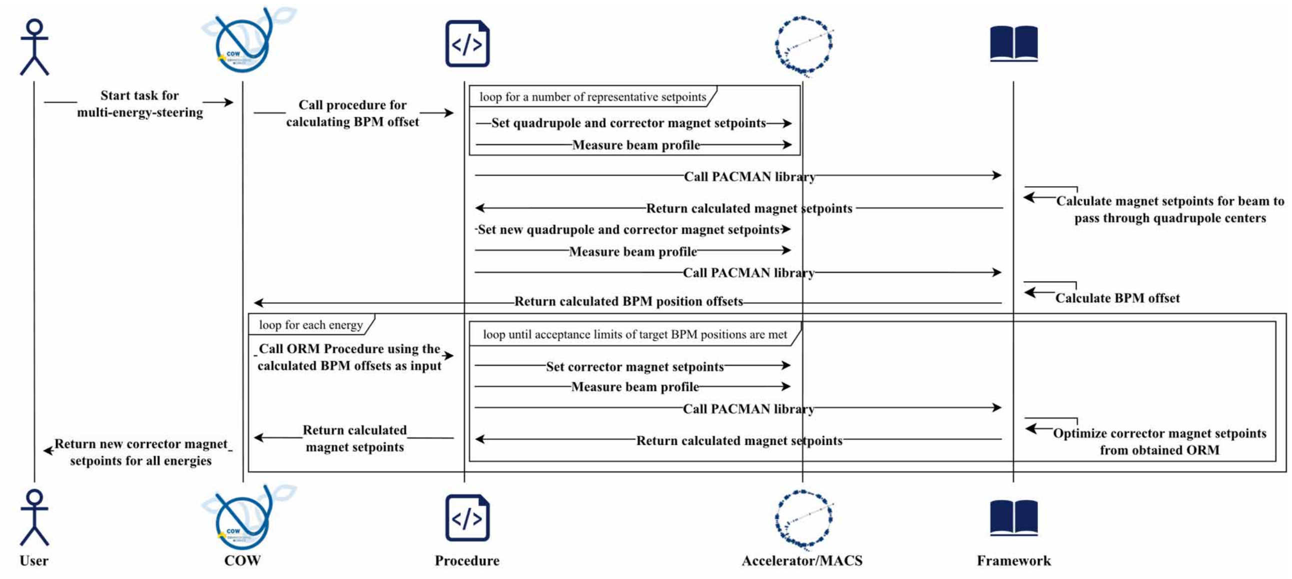
Start task for multi-energy-steering
Call procedure for calculating BPM offset
loop for a number of representative setpoints
Set quadrupole and corrector magnet setpoints
Measure beam profile
Call PACMAN library
Calculate magnet setpoints for beam to pass through quadrupole centers
Return calculated magnet setpoints
Set new quadrupole and corrector magnet setpoints
Measure beam profile
Call PACMAN library
Calculate BPM offset
Return calculated BPM position offsets
loop for each energy
loop until acceptance limits of target BPM positions are met
Call ORM Procedure using the calculated BPM offsets as input
Set corrector magnet setpoints
Measure beam profile
Call PACMAN library
Optimize corrector magnet setpoints from obtained ORM
Return calculated magnet setpoints
Return calculated magnet setpoints
Return new corrector magnet setpoints for all energies
User
COW
Procedure
Accelerator/MACS
Framework

Figure 4: COW sample Procedure “Multi-Energy-Steering”.

As the absolute positions of the in-line beam profile monitors (BPMs) cannot be assumed to be accurate, a series of corrector magnet scans combined with quadrupole strength variations have to be performed to establish a beam trajectory that passes through the quadrupole centres. This trajectory can be used to obtain the position offsets of the BPMs which can then be used as targets in an orbit-response-matrix (ORM) based beam steering approach for the various extraction energies.

In the new MACS architecture, a single procedure in the COW would allow to autonomously perform the complete steering for all required energies and directly store the newly obtained corrector magnet setpoints in the machine physics database (see Fig. 4). In the current MACS architecture, there is no communication established between the accelerator and the analysis frameworks. Therefore, manual intervention is required every time measurements are taken, to be able to proceed with the analysis. On top of that, incoming measurements can be evaluated as soon as they are available, leading to additional reduction of analysis time, as part of the analysis can be parallelized with the ongoing measurement chain.

Outlook

The Commissioning Worker is currently under development. A proof-of-concept version is ready, including a few sample Procedures.

One simple Procedure, already tested through all layers of the COW and MACS UI, was bringing a single device into a different operational state.

In this simple example the Python Procedure has two input parameters: the device name and the target state. On Task Execution a PAAPI method is called, which translates to the according MACS UI’s API Gateway’s method.

This is the equivalent of manually navigating to the device in question via a standard UI panel, selecting the device and selecting the target status in the device’s state machine UI.

The first COW version targeted to the end user is planned to be ready in 2024, with a fully defined PAAPI, and the first available “real-life” Procedures.

The COW is designated to be used at the facility in Austria and other future facilities.

The expectancy towards the COW is:

- A great reduction in machine time required for commissioning tasks and reduction in error-proneness, by removing manual iterative steps and not requiring constant context switching.
- Flexibility, either to create quick one-time Procedures, or to create validated Procedures and Frameworks conforming with all the necessary development processes.
- Traceability (Who executed which Procedure, in which version, using which Framework versions, when, with what Result?).
- Smooth integration into the MACS architecture, therefore not opening additional entry points into the medical product, with obvious advantages regarding cyber security.

For this reason, COW is a highly anticipated extension to our control system by our commissioning, expert and operations teams.

REFERENCES

[1] WinCC OA, `https://www.winccoa.com`

[2] J. Gutleber *et al.*, “The MedAustron Accelerator Control System”, in *Proc. ICALEPCS'11*, Grenoble, France, Oct. 2011, paper MOBAUST03, pp. 9-12.

[3] W. W. Eckerson, “Three Tier Client/Server Architecture: Achieving Scalability, Performance and Efficiency in Client Server Applications”, Open Information Systems, 1995, 1:10.

[4] *Synchrotron Radiation Sources: A Primer*, H. Winick, Ed. World Scientific, 1994, pp. 218.

[5] L. Evans and P. Bryant (eds.) "The CERN Large Hadron Collider: Accelerator and Experiments", *J. Instrum.*, vol. 3, chapter 9, p. 98, 2008. doi:10.1088/1748-0221/3/08/S08001

[6] Oracle DB, https://www.oracle.com/database/technologies

[7] M. Hager and M. Regodic, "A Modular Software Architecture for Applications that Support Accelerator Commissioning at MedAustron", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 938-941. doi:10.18429/JACoW-ICALEPCS2015-WEPGF101

[8] Using the LabVIEW Shared Variable, https://www.ni.com/en/support/documentation/supplemental/06/using-the-labview-shared-variable.html

[9] IIS Webserver Overview, https://learn.microsoft.com/en-us/iis/get-started/introduction-to-iis/iis-web-server-overview

[10] Electron, https://www.electronjs.org

[11] Angular, https://angular.io

[12] Kendo UI for Angular, https://www.telerik.com/kendo-angular-ui

[13] Apache ECharts, https://echarts.apache.org

[14] TimescaleDB, https://www.timescale.com

[15] Atvise, https://www.atvise.com

[16] A. Wastl, A. Garonna, T. K. D. Kulenkampff, and S. Nowak, "PACMAN - the MedAustron Measurement Data Analysis Framework", in *Proc. IPAC'16*, Busan, Korea, May 2016, pp. 2774-2777. doi:10.18429/JACoW-IPAC2016-WEPOR045

[17] "Medical device software - Software life cycle processes," International Electrotechnical Commission (IEC), IEC 62304, 2015.