PROCEEDINGS OF SCIENCE

Upgrades to the Slow Control of the IDMAR Junction Box

Emidio Giorgio,[a,*] **Fabrizio Ameli,**[b] **Rosanna Cocimano,**[a] **Lucio Mirko Coppolino,**[a] **Carlo Alessandro Nicolau,**[b] **Nunzio Randazzo,**[d] **Sara Pulvirenti,**[a] **Jan-Willem Schmelling,**[c] **Salvatore Viola**[a] **on behalf of the KM3NeT collaboration**

[a] *Istituto Nazionale di Fisica Nucleare - Laboratori Nazionali del Sud, Via Santa Sofia 62, Catania, Italy*

[b] *Istituto Nazionale di Fisica Nucleare - Sezione di Roma, Piazzale Aldo Moro 2, Roma, Italy*

[c] *NIKHEF Science Park 105,Amsterdam, The Netherlands*

[d] *Istituto Nazionale di Fisica Nucleare - Sezione di Catania, Via Santa Sofia 64, Catania, Italy*

E-mail: emidio.giorgio@infn.it

IDMAR is an underwater, multidisciplinary research infrastructure currently under construction in the abyssal depths of the Mediterranean Sea, off the southeastern coast of Sicily. It integrates innovative sensors for real-time studies in the submarine environment. One of IDMAR's primary use cases is ARCA, the Cherenkov detector being developed by the KM3NeT Collaboration as part of the KM3 Neutrino Telescope, designed to study cosmic neutrinos at TeV-PeV energies.

A key component of IDMAR is the Junction Box (JB)[2], which receives electro-optical signals transmitted from onshore via the Main Electro-Optical Cable and distributes them to underwater user experiments, such as the KM3NeT[1] Detection Units (DU).

The original JB design has been upgraded to support a greater number of detection units, optimizing costs in the process. Version 2 of the JB integrates two independent sub-units within the same mechanical enclosure, each featuring its own electro-optical connection. Despite their independence, a unified management approach has been maintained to align with operational practices.

Another major advancement in the slow control system is the enhancement of authentication and authorization mechanisms. In addition to local authentication based on usernames and passwords, external authentication providers are now supported. Furthermore, fine-grained user and group permission management has been implemented to improve access control.

This paper outlines the key decisions behind this significant architectural evolution and their implementation.

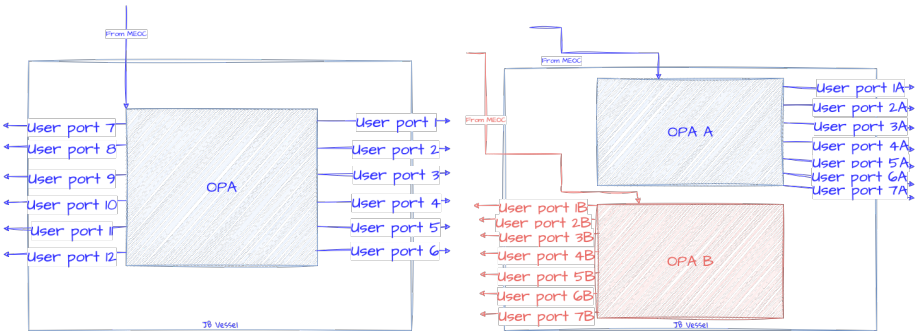
39th International Cosmic Ray Conference (ICRC2025)
15–24 July 2025
Geneva, Switzerland

*Speaker

PoS(ICRC2025)1043

1. Introduction

IDMAR is an offshore, multidisciplinary research facility located at a depth of 3500 meters and approximately 100 km off the southeastern coast of Sicily. It is connected to shore via electro-optical cables that provide both communication and power transmission. In addition to the first connection, a new Main Electro-Optical Cable (MEOC) was deployed in late 2022. This new MEOC terminates underwater at the Cable Termination Frame (CTF), which hosts fiber distribution systems and medium-voltage converters. The CTF is equipped with both electrical and optical ports to support second-level nodes, known as Junction Boxes (JBs) [2]. These JBs are connected to the CTF through seabed-laid cables arranged in a star topology. Junction Boxes distribute electrical power and optical signals to user systems such as the KM3NeT/ARCA Detection Units (DUs), which represent the main scientific use case for the IDMAR infrastructure. In addition, the JBs support instruments such as hydrophones for monitoring environmental noise, as well as acoustic and laser emitters used for the auto-calibration of long-baseline acoustic positioning systems deployed with the users' equipment. Leveraging the increased fiber count in the new MEOC cable, a significant hardware upgrade has been implemented, allowing two independent Optical and Power Assembly (OPAs)—each with 7 output ports—to be housed within the same JB vessel. As a result, the total capacity of each Junction Box has been increased from 12 to 14 output ports. For further details on the architecture and slow control system of the original JB implementation, refer to [10].



(a) (b)

Figure 1: Sketches of input and output connections of JBv1 (a) and JBv2 (b)

2. JB tasks, goals, and architectural differences from the original implementation

Each OPA inside a JB features one electro-optical input port connected to the CTF, and seven output ports dedicated to user systems. As previously introduced, the main architectural difference from the original design is the presence of two OPAs instead of one. While the first version of JB [10] had a single OPA with twelve user ports and one connection to the CTF, the new design includes two independent OPAs, each with seven output ports, increasing the total to fourteen.

Despite this upgrade, each output port still provides up to 1 A at 375 V DC and supports data transmission from the connected systems to shore. Interfaces for calibration instruments—such as hydrophones, laser emitters, and acoustic beacons—are maintained. However, the Instrumentation Control Electronics (ICE) board, which handles these devices, is present only in OPA B.

The following sections describe the JB subsystems in more detail, focusing on the architectural changes:

- **Optic:** This subsystem manages optical fibre data transmission and includes 1 physical fibre per OPA, used for both data streaming and command transmission. A key improvement lies in the simplification of the optical amplification system. Originally, two types of Erbium-Doped Fiber Amplifiers (EDFAs) were used: EDFA N (shore-to-JB) and EDFA P (JB-to-shore). Due to the increased number of fibres of the new cable, that allowed a newer design of the downstream KM3NeT readout [**?**], the EDFA N is no longer required. The optical module now consists of a pair of redundant EDFA P amplifiers, managed by an Optical Switchover Logic Board (OSL) that enables to switch between the primary and backup amplifier.

  Commands are sent offshore via the Master Control Unit (MCU). For each OPA, the MCU is composed of one onshore unit and two offshore counterparts, connected via separate links. These links are managed in failover mode by the onshore MCU. This JB version also includes monitoring of sensors to measure transmitted and received optical power. Each MCU has a dedicated IP address and encapsulates RS232 serial connections into TCP sockets, simplifying remote communication and control via standard network interfaces.

- **Power:** The power subsystem distributes electrical power received from shore through the Main Electro-Optical Cable (MEOC). Its architecture remains unchanged except for the increased number of output ports. It consists of two board types: the Power Supply Module (PSM) and the Switch Module (SWM). The PSM contains a DC/DC converter that steps down the 375 V input to the 5 V required by internal JB electronics. It also provides output isolation to protect against downstream short circuits. The SWM manages each user port individually, supporting port enable/disable operations, current monitoring, and automatic disconnection in case of persistent overload. Each output port has its own SWM. For reliability, the connection paths to SWMs are redundant and can be monitored independently through separate MCU ports.

- **Instruments:** Each JB includes sensors such as hydrophones, acoustic beacons, and laser emitters, forming part of the site's long-baseline acoustic positioning system. The hydrophone also streams acoustic data to shore, enabling environmental noise monitoring. The Instrumentation Control Electronics (ICE) board controls these devices and includes a White Rabbit module for sub-nanosecond synchronization [7].

Most electronic boards communicate natively via RS232 serial connections, which are encapsulated by the MCU into TCP sockets for network-based communication. The ICE board is an exception: it uses UDP protocol, communicates independently from the MCU, and has its own dedicated IP address.

The slow control system enables remote management of all JB subsystems. It continuously monitors the health of electronic components by acquiring operational parameters and recording functional values to support anomaly detection and proactive fault prevention.

3. Slow control implementation

Despite relatively minor changes to the hardware architecture, the implementation of the slow control system required several adjustments to preserve the user and system administrator experience as close as possible to that of the first version [, jbv1] while also addressing issues that emerged during three years of operation with version 1.

The three-tier architecture was preserved, with the JB Manager, Data Store, and Web GUI each deployed in separate containers and communicating exclusively over the network via RESTful APIs. This design also made it possible to represent the two OPAs within the Web GUI without fragmenting control across multiple servers, as it can be seen delving into details.

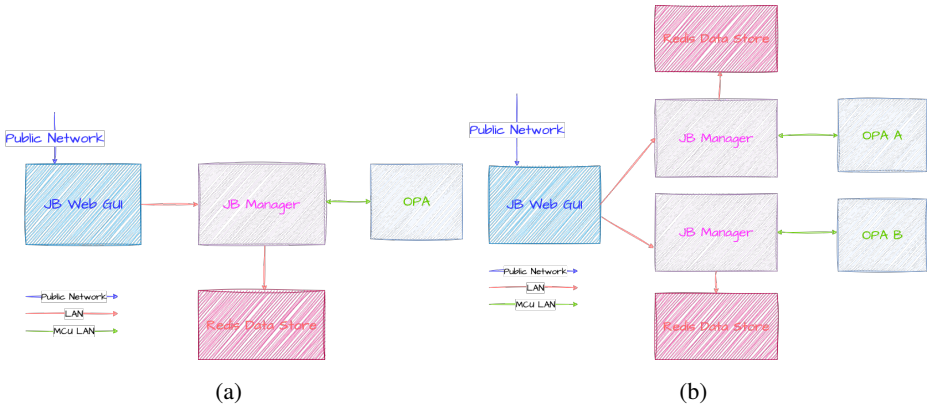


Figure 2: Sketches slow control subcomponents for JBv1 (a) and JBv2 (b)

3.1 JB Manager and Network Multiplication

The JB Manager is responsible for direct interaction with the hardware: it communicates with the JB subsystems and continuously reads out the parameters from the electronic boards. It is implemented using Flask [6], a web framework written in Python. The JB Manager features a dedicated software module for each JB subsystem. Each module is divided into two components, implemented as separate Python classes:

- **Low-level interface**: JB hardware devices expose only a serial interface, which poses several limitations—such as the use of cryptic commands and the lack of concurrent access. Although the MCU wraps these interfaces into TCP sockets, socket access remains exclusive, preventing multiple clients from simultaneously querying the same resource. The low-level classes manage connections to JB devices and provide intermediate-level methods that abstract raw serial commands for hardware configuration and readout. Commands (sent via TCP, or UDP in the case of the ICE board) follow a protocol defined by the board designers.
- **Upper-level adapter**: Each adapter class imports its corresponding low-level interface and runs a continuous refresh loop that invokes readout functions and stores the acquired parameters in the data store. These adapters also implement standard software interfaces as RESTful APIs [12]) to expose selected methods, allowing remote clients to perform readout

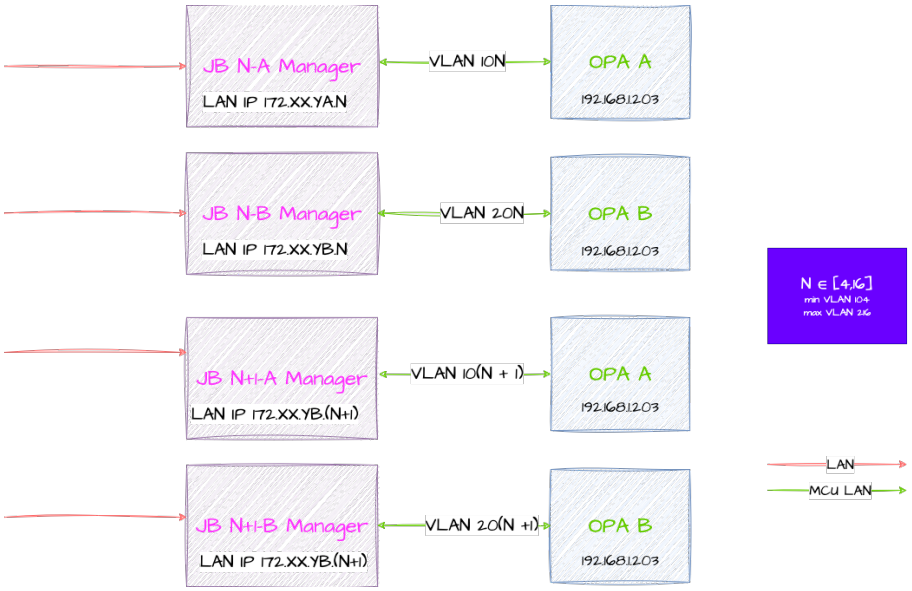
and operational tasks on the hardware. The API responses are formatted in JSON and follow standard HTTP conventions for status codes, ensuring compatibility with third-party clients that comply with these standards. To ensure consistency and prevent race conditions, each adapter class implements a locking mechanism that avoids concurrent writes to the same resource and prevents inconsistent callbacks.

While the software architecture described above remains largely unchanged from the previous version, a noteworthy update lies in the network configuration. The MCUs (both onshore and offshore) have predefined private IP addresses, retained to minimize the risk of offshore failures and to facilitate testing procedures.

When deploying multiple JBs, identical IP addresses could lead to conflicts. This was mitigated by leveraging the fact that the JB Manager is the sole component communicating with the MCUs. Therefore, default IP addresses were preserved, and each JB Manager-to-OPA connection at the low level is placed in a dedicated VLAN.

Given that the infrastructure will eventually include up to fifteen JBs—resulting in around thirty OPAs—managing thirty VLANs remains practical within a private network.

On the upper level, each JB Manager is assigned a unique IP address, making it discoverable and accessible by the JB Web GUI and potentially other third-party client applications.



(a)

Figure 3: Sketches of internal network implementation for JBv2

Upper-level adapters are orchestrated by a manager class, which is invoked at the startup of the JB Manager container. Upon initialisation, the manager class reads the software configuration parameters—such as loop interval, debug level, and others—and then spawns a dedicated thread for each adapter. In this version, module-specific parameters for loop interval and debug level have been introduced, allowing per-module debugging and preventing unnecessary load on the MCU caused by excessively frequent queries. Each thread interacts with the corresponding low-level interface

to periodically update values in the relevant section of the data store and to handle incoming HTTP requests via the WSGI engine provided by Flask.

3.2 Data Store

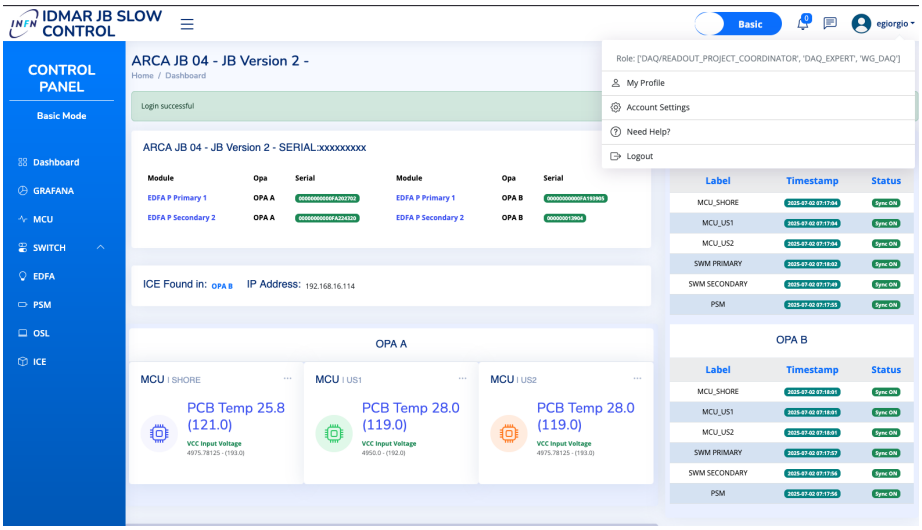
Data collected by the refresh loops are stored in a time-series database, RedisTimeSeries. Redis is a NoSQL (non-relational) in-memory key–value store designed for high-performance read/write operations. Although data resides in memory for speed, periodic snapshots to disk are supported to ensure persistence across server restarts.

The specific Redis flavor used in the JB slow control system supports time-series data, allowing each key to be associated with a (value, timestamp) pair. This makes it straightforward to construct historical time series for each key, where each key typically corresponds to a hardware parameter.

The relatively small and fixed number of keys (approximately 10^3) further improves Redis efficiency, as key access is effectively $O(1)$, and enables longer data retention within memory limits.

Thanks to these features, API requests for board parameters are not served by querying the hardware directly. Instead, they return the most recent value written to RedisTimeSeries by the adapter during its periodic refresh loop. In this context, the timestamp also serves as an indicator of data validity: values with outdated timestamps should be considered unreliable, and their lack of updates should be reported by the operators.

3.3 Web GUI



(a)

Figure 4: Landing page of JBv2, with overviews of both OPAs. It can be seen how the user is authenticated via KM3NeT SAML where it gets also attributes

Although in principle the Junction Box (JB) can be managed directly via the RESTful APIs exposed by the JB Manager—and users are free to develop their own client applications or even

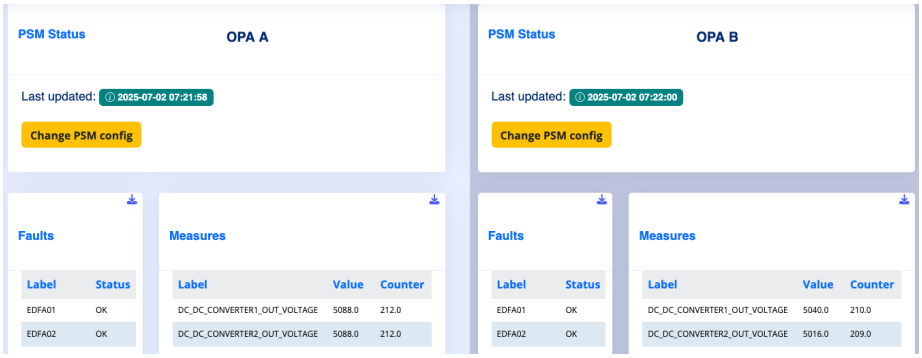
interact using command-line tools such as ‘curl‘—a Web GUI was developed to simplify operations for end users and system operators.

The Web GUI is implemented as a separate Flask web application and provides a responsive dashboard based on a customized free Bootstrap template. The Bootstrap framework was upgraded to version 4 to support a more modern layout, including improved components and interactive controls.

The Web GUI backend mirrors the JB Manager architecture and follows the Model-View-Controller (MVC) design pattern [8]. In a typical workflow, the user accesses a URL corresponding to a specific JB subsystem. The request is routed internally through the appropriate view, which invokes a model responsible for handling any user input and issuing the relevant API call to the JB Manager. The API response, formatted as JSON, is then rendered into HTML using predefined templates. This pattern is applied consistently across both read and write operations—whether retrieving subsystem status or modifying a parameter.

As mentioned earlier, the Web GUI fully supports the dual OPA architecture. A single Web GUI instance handles both OPAs within the same JB, presenting them side-by-side in the interface. JavaScript is used to preserve overall usability and interactivity. Importantly, support for single OPA configurations has been retained in order to avoid maintaining separate codebases and to accommodate legacy hardware and test environments, where only one OPA is typically in use.

Additional enhancements include the introduction of macro commands to automate repetitive tasks—such as disabling auto-rescue conditions or activating user ports—as well as a dashboard panel summarizing synchronization status across all modules.



(a)

Figure 5: Overview of PSM control page, showing both OPAs. On each card there is a download button enablig export of data to a csv

4. Authentication and authorization

While data readout is essentially a rework of the previous implementation—adapted to reflect hardware updates—the authentication and authorization framework has undergone a substantial redesign. Building upon the existing support for JWT-based access tokens, the authentication backend has been migrated from a local database to an external identity provider, with capabilities

of integration with various technologies and other identity providers, including the KM3NeT identity system.

This advancement enables several important capabilities, the most significant of which is fine-grained authorization. Access rights can now be restricted based on roles and permission levels associated with the authenticated identity. Additionally, control over specific subsystems can be delegated according to user roles defined in the identity provider.

For example, a KM3NeT Detector Operations Manager may be authorized to open or close user ports or adjust the emission rate of acoustic beacons, while a user with the Power Management role may be permitted to permanently disable ports.

5. Future outlook

The hardware is now stable, so major changes to the software are no longer expected, just minor improvements concerning usability, and bug fixed. Future efforts will rely on improving authorisation layers, in order to extend interoperability, in particular the possibility of managing the instruments via the KM3NeT Control Unit.

References

[1] S Adrían-Martínez et al. (KM3NeT Collaboration), *Letter of Intent for ARCA and ORCA* in *J. Phys. G: Nucl. Part. Phys.* **43** (2016), 084001

[2] F. Ameli et al, *Underwater node for electrical power distribution and optical data transmission for multidisciplinary science in the Mediterranean Sea*, *2022 IEEE International Workshop on Metrology for the Sea; Learning to Measure Sea Health Parameters (MetroSea)* 441–445

[3] F. Ameli et al, *The ICE board: a Hi-Rel electronics board for sub-nanosecond synchronization of submarine instrumentation* *2022 IEEE International Workshop on Metrology for the Sea; Learning to Measure Sea Health Parameters (MetroSea)* 429–434

[4] Redis Timeseries

[5] Use containers to Build, Share and Run your applications

[6] Get started with Flask

[7] White Rabbit for control and data acquisition system

[8] Pop, Dragos-Paul and Altar, Adam, *Designing an MVC model for rapid web application development*, *Procedia Engineering* (2014) **69** 1172–1179

[9] Introduction to JSON web token

[10] E. Giorgio et al, *Slow control of the IDMAR Junction Box* *ICRC2023*

[11] F. Benfenati Gualandi et al, *The data acquisition system for the complete KM3NeT-ARCA neutrino telescope* *ICRC2025*

[12] What is REST?