ATLAS Qualification Interface Refactoring Strategy

Ana Clara Loureiro Cruz[1,*], *Rafaella* Lenzi Romano[1], *Carolina* Niklaus Moreira da Rocha Rodrigues[1], *Gabriela* Lemos Lúcidi Pinhão[2], *Leonardo* Mira Marins[1], *Pedro Henrique* Goes Afonso[1], *Rodrigo* Coura Torres[1], *José Manoel* Seixas[1], and *Natanael* Nunes de Moura Junior[1,**]

[1]Signal Processing Lab, COPPE/EE - UFRJ (Federal University of Rio de Janeiro)
[2]Laboratório de Instrumentação e Física Experimental de Partículas - LIP, Lisboa

Abstract. The ATLAS experiment involves over 6000 active members, including students, physicists, engineers, and researchers, and more than 2500 members are authors. This dynamic CERN environment brings up some challenges, such as managing the qualification status of each author. The Qualification system, developed by the Glance team, aims to automate the processes required for ATLAS members to achieve author status. Recently, ATLAS modified the policy behind it, and updates were necessary to put it into effect.
The system was developed on top of an outdated framework. In order to ease the transition to the new ATLAS authorship qualification policy, the former solution was updated to a Hexagonal architecture based on Domain Driven Design philosophy. The access to the database has shifted from ORM - Object Relational Mapper - to SQL repositories to align with the team's development stack. The system's quality is ensured with automatic tests as part of an effective refactoring process that is transparent to the end user. This refactoring strategy intends to enhance the system to improve code maintainability, efficiency and to increase flexibility to accommodate future changes in the qualification policy.

ATL-SOFT-PROC-2025-010
15 January 2025

1 Introduction

1.1 Context

The European Organization for Nuclear Research (CERN) is home to the largest particle accelerator in the world, the Large Hadron Collider (LHC). Established in 1954, CERN has been at the forefront of groundbreaking discoveries and technological innovations. One of the experiments conducted at CERN is ATLAS [1]. The primary goal of the ATLAS collaboration is to explore the fundamental particles and forces that make up the universe. The experiment involves over 6000 active members, including students, physicists, engineers, and researchers from numerous institutions across the globe [2]. Given this dynamic and multidisciplinary environment, the collaboration faces challenges in coordinating numerous working groups and efficiently managing the data related to the management of the experiment. To address these management complexities, the Glance project [3] was launched in

*e-mail: clara.cruz@cern.ch

2003, introducing an automated system designed to streamline operations and improve data management within the collaboration.

Currently, the Glance project implements and maintains systems designed to automate the management of multiple LHC experiments, including ATLAS, ALICE, LHCb and CMS. To minimize rework and ensure code uniformity across these diverse experiments, the project employs a common software development stack. This unified approach not only simplifies development but also allows collaboration between experiments by providing standardized components that can be shared between them.

1.2 The modernization of the Qualification System

To become an ATLAS author, a member must complete the qualification process: a structured workflow encompassing all necessary steps to be included on the ATLAS author list. This process is managed by the ATLAS Glance Qualification system.

After the system was developed in 2014, ATLAS revised the qualification workflow policy. Those updates required modifications to the system to meet new requirements. Following the Scrum methodology [4], the ATLAS Glance team proposed not only implementing the policy updates but also modernizing the codebase. Recognizing that the system was built on an outdated framework, the team identified this as an opportunity to align the system with the current Glance technology stack, ensuring greater consistency and maintainability across projects.

Refactoring is an important engineering practice within Scrum, enabling teams to improve code quality while ensuring it remains maintainable and adaptable to future changes. This approach minimizes technical debt, improves long-term maintainability, and simplifies future development efforts. To ensure the success of this refactoring initiative, automated testing was prioritized.

1.3 Project objectives

The primary objectives of the ATLAS Glance Qualification system's refactoring are:

- Incorporate unmet requirements by adding features like detailed history logs and new fields, addressing gaps in the system and enabling easier field expansion.
- Improve modularity, adaptability to support evolving requirements and system responsiveness.

2 Methodology

The development process was organized using the Scrum framework, which facilitates iterative progress, continuous feedback, and adaptive planning. The ATLAS Glance team is composed of cross-functional developers, enabling collaboration throughout the project lifecycle.

The project was divided into three distinct phases, illustrated in Figure 1. Each iteration represents incremental improvements and addresses specific aspects of the system.

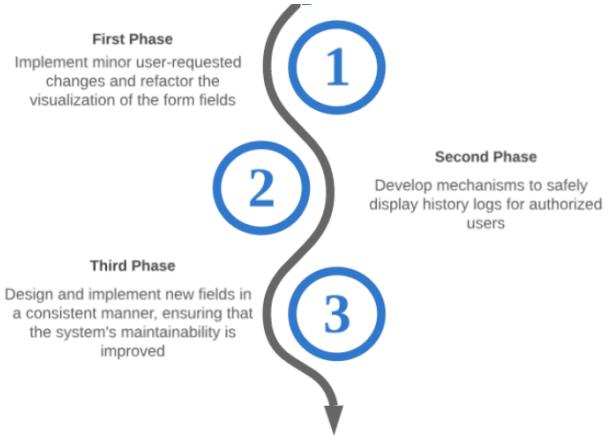

Figure 1. Phases of the Qualification refactoring project.

The first phase was focused on implementing minor user-requested changes and refactoring the visualization use cases. The refactoring effort was designed to improve the readability and maintainability of the system, setting a solid foundation for subsequent developments. The second phase is centered on the implementation of history tracking features. This phase involves developing mechanisms to safely display history logs for authorized users. The third and final phase will address the refactoring of write repositories (database adaptors [5]) and the implementation of new fields requested by users. The refactoring of the code will be simplified by the uniform structure established in previous phases, facilitating efficient integration of new features. This phase includes the design and implementation of new data fields in a consistent manner, ensuring that the system's flexibility and adaptability are improved.

Throughout the project, the Scrum framework provides iterative feedback loops and regular sprint reviews and retrospectives, allowing continuous refinement and alignment with user requirements. This phased approach not only ensures that each requirement is thoroughly addressed but also enables the team to adapt to evolving needs effectively. Currently, the team is developing the second phase.

3 Project Architecture

Before this project, the Qualification system was implemented with a fragmented architecture. Read use cases were implemented using the legacy Fence framework [6], while write use cases were managed through Doctrine Object Relational Mapper (ORM) [7]. This dual approach introduced inconsistencies, increased complexity in maintenance, and compromised system scalability. The refactoring process involves the migration from Fence to a hexagonal architecture [5], aiming at achieving a modular and testable structure that adheres to contemporary software development practices. Additionally, the project involves the replacement of Doctrine ORM with Structured Query Language (SQL) [8] repositories, standardizing data access, optimizing performance, and ensuring uniformity across the system.

3.1 Migration to hexagonal architecture

The replacement of the legacy Fence framework with a hexagonal architecture, as illustrated in Figure 2, implements Domain-Driven Design (DDD) principles [9]. This architecture

promotes a clear separation of concerns by isolating the business logic in the domain layer, positioned at the center of the structure. The domain is connected to external systems through adaptors, which act as intermediaries between the core logic and the port. In this case, one port interfaces with the Fence framework front-end, and another manages interactions with the database. This design not only ensures modularity and testability but also simplifies the addition of future features by decoupling dependencies, aligning with the team's standard development stack, and reducing technical complexity.

To provide a seamless transition for the end user, the existing Fence user interface was preserved. Figure 3 shows the system organization: the Controller mediates between the Repository and the Fence user interface. This design allowed the Controller to format data from the newly implemented architecture to match the structure of the legacy system.

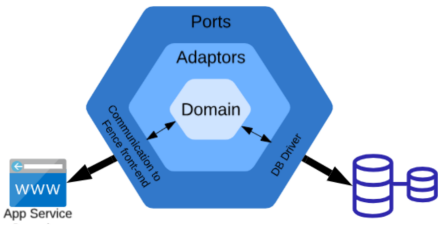

Figure 2. Hexagonal architecture of the Qualification System. The central domain layer encapsulates the business logic, surrounded by adaptors connecting the system to external components. Two primary ports are shown: one interfaces with the Fence framework for the user interface, and the other connects to the Oracle relational database [10], facilitating efficient data management.

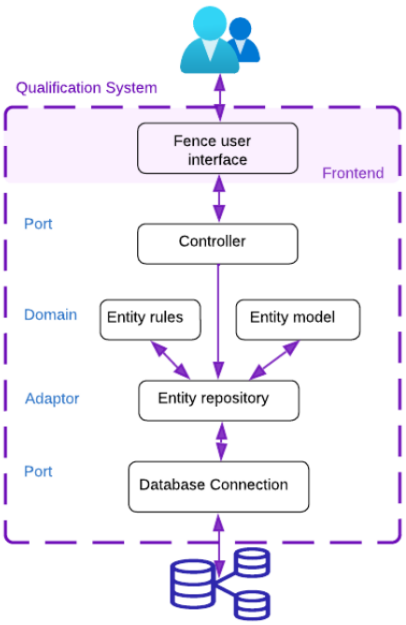

Figure 3. Qualification system's overall structure.

3.2 Migration to SQL Repositories

The second phase of the project includes the migration from Doctrine ORM to SQL repositories. This eliminates the additional abstraction layer used to interface with the database. This approach directly integrates SQL queries into the codebase, bypassing the need for intermediate configuration files. By doing so, data manipulation operations become more explicit and transparent for developers, facilitating both debugging and the implementation of future changes.

The adoption of the hexagonal architecture simplified this migration process. With the domain logic already encapsulated in the Domain Layer, the refactoring required only targeted updates to existing repositories. Reusing the pre-defined domain reduced the effort needed to adapt database operations to the new architecture. This change aligns the Qualification system with standardized Glance development practices, improving maintainability and long-term adaptability.

3.3 Automatic Tests

By validating that the system's functionalities remain consistent after modifications, automated tests were developed during each phase of the refactoring project. Comprehensive test coverage ensured that modifications did not compromise existing functionalities, improving the development process. The tests systematically evaluate each code change before integration, maintaining system integrity and stability.

The refactored system employs PHPUnit [11] to implement a comprehensive suite of automated tests, divided into unit tests and integration tests. Unit tests validate individual components, such as functions or classes, in isolation from the rest of the codebase. These tests execute quickly and provide precise feedback, enabling developers to identify and resolve defects at a granular level during development. Integration tests were used to assess the interactions between project components and the database, ensuring correct functionality in real-world scenarios. Those tests validate dependency management and data flow integrity, addressing issues that unit tests alone cannot detect.

Mocks and stubs [12] were used in unit tests to simulate dependencies and isolate testable units. Integration tests, on the other hand, directly interact with the database to verify system-wide behavior.

This dual testing strategy ensures thorough verification at both the component and system levels. The automated tests have been instrumental in maintaining system integrity during the refactoring process.

4 Conclusion and Next Steps

The team's domain-driven strategy combined with an iterative development, a cornerstone of Scrum, can be used to address immediate needs while simultaneously enhancing the overall quality and scalability of the software.

The Qualification system's fragmented codebase, resulting from inconsistent integration with new technologies, needed a transition to a unified, modular architecture. This project aims at consolidating the system by fully migrating to a consistent stack, enhancing maintainability and coherence. The first phase, which involved minor changes and refactoring of read use cases, has been completed. The remaining phases will focus on implementing history features and refactoring writing use cases.

Acknowledgments

The authors would like to thank CNPq, CAPES, FAPERJ, and RENAFAE (Brazil), as well as CERN and the ATLAS collaboration for providing financial support for this work.

References

[1] ATLAS Collaboration, JINST 3, S08003 (2008). DOI: 10.1088/1748-0221/3/08/S08003

[2] *ATLAS Institutions*(https://atlaspo.cern.ch/public/institutions/, Accessed 23-07-2024)

[3] C. Maidantchik, F. Grael, K. Galvão and K. Pommès (2008) Glance project: a database retrieval mechanism for the ATLAS detector. *J. Phys.: Conf. Ser.*. DOI: 10.1088/1742-6596/119/4/042020

[4] *Scrum Framework Guide* (https://www.scrum-framework.com/Library, Accessed 26-08-2024)

[5] M. Noback (2020). Advanced Web Application Architecture. LeanPub.

[6] B. Lange et al. (2015) An object-oriented approach to deploying highly configurable Web interfaces for the ATLAS experiment. *J. Phys.: Conf. Ser. 664 062026*. DOI: 10.1088/1742-6596/664/6/062026

[7] Doctrine Object Relational Mapper, URL https://www.doctrine-project.org/projects/orm.html [accessed 04-Dec-2024].

[8] Oracle SQL for Accessing, Defining, and Maintaining Data, URL https://www.oracle.com/database/technologies/appdev/sql.html [accessed 04-Dec-2024].

[9] E. Evans (2004). Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley.

[10] Oracle database, URL https://www.oracle.com/database/ [accessed 07-Jan-2025].

[11] PHPUnit, URL https://phpunit.de/index.html [accessed 04-Dec-2024].

[12] PHPUnit Test Doubles, URL https://docs.phpunit.de/en/10.5/test-doubles.html [accessed 04-Dec-2024]