10-21-2023

Restructuring the Software Architecture: A Case Study of the CoolBiz Core Banking Platform

Jackson Júnior
Polytechnic Institute of Viana do Castelo, jacksonjunior@ipvc.pt

Pedro Coutinho
Polytechnic Institute of Viana do Castelo, Coollink, pedro.coutinho@coollink.pt

Restructuring the Software Architecture: A Case Study of the CoolBiz Core Banking Platform

Jackson Júnior, Polytechnic Institute of Viana do Castelo, Portugal, jacksonjunior@ipvc.pt

Pedro Coutinho, ADiT-LAB, Polytechnic Institute of Viana do Castelo, Portugal, p.coutinho@estg.ipvc.pt; Coollink, Portugal, pedro.coutinho@coollink.pt

Abstract

As the structural engineering underpins the resilience of a city built on an active geological fault, software architecture becomes crucial in an increasingly digital society. This paper investigates the challenges of rigid, low cohesion software structures through a detailed case study of the CoolBiz Platform, an integrated Core Banking solution. The platform currently faces significant issues in its service support framework, including low flexibility, unsatisfactory cohesion, non-adherence to SOLID principles, absence of unit tests, and lack of documentation. This study aims to describe the planning and implementation of a new event-driven architecture for the CoolBiz Platform. This architecture is expected to not only resolve current technical challenges but also bring significant business benefits, such as the implementation of language agnosticism, a strategy aimed at facilitating talent recruitment and retention by not limiting recruitment to expertise in a specific programming language.

Keywords: Software Architecture; Event-Driven Architecture; Framework Redesign; Case Study; Proof of Concept.

1. INTRODUCTION

Just as structural engineering defines the robustness of a city built on an active geological fault, software architecture becomes fundamental in an increasingly digitized society. The similarity lies in the need to anticipate and resist unexpected and inevitable movements. In the case of software systems, these "earthquakes" materialize as constant changes and challenges in the digital environment.

Despite the growing importance of software architecture, many organizations, especially those operating in highly complex and regulation-intensive industries such as banking, still face challenges with rigid, low cohesion software structures. These structures often fail to adhere to best design practices, such as SOLID principles (Martin, 2000), creating barriers to efficiency and innovation.

This research investigates these challenges through a detailed case study of the CoolBiz Platform, an integrated Core Banking solution. The platform is currently dealing with significant problems in its service support framework. These problems include low flexibility, unsatisfactory cohesion, non-compliance with SOLID principles, lack of unit testing, and even lack of documentation. The need

ISSN 2183-489X
http://dx.doi.org/10.18803/capsi.v23.348-363

to ensure uninterrupted service continuity often overshadows the urgency to modernize the software architecture, a common dilemma in highly complex, highly regulated environments.

The purpose of this study is to describe the planning and implementation of a new proposal for an event-driven architecture for the CoolBiz Platform. It is expected that this architecture will not only solve current technical challenges, but also bring significant business benefits. One such benefit is the implementation of language agnosticism, a strategy that aims to facilitate hiring and retention of talent by not limiting recruitment to expertise in a specific software development stack or a limited number of programming languages.

The remainder of the paper is organized as follows: Section 2 provides a review of the literature on software architecture and SOLID principles. Section 3 describes the methodology used for the case study. Section 4 investigates the current architecture of the CoolBiz Platform and identifies key areas for improvement. Section 5 presents the proposed new architecture in depth, explaining the design logic and presenting the results of a preliminary proof of concept. Finally, Section 6 concludes the paper with a summary of key points and suggestions for future research.

2. RELATED WORKS

Unit testing is an essential part of software development. (Palma, 2007) highlights the need for evolving unit tests to ensure software quality over time. (Petersen & Wohlin, 2009) also discuss the importance of unit tests in agile and incremental software development. (Nassif et al., 2020) discuss the generation of unit tests for documentation, highlighting the importance of consistent and verifiable tests to ensure software quality over time. Their research suggests that a lack of unit testing can lead to inconsistencies and degradation of documentation quality.

Software documentation is another crucial aspect in the development and maintenance of software systems. (Coelho, 2009) and (de Souza et al., 2007) highlight the need for proper documentation for effective software maintenance. Furthermore, (Souza & Moraes, 2018) discuss the importance of explicit knowledge in overcoming productive challenges in software development.

Event-driven architecture is an emerging paradigm in software architecture. (Clark & Barn, 2011) discuss modeling and simulation of event-driven architectures, while (Juric, 2010) propose extensions to Event-Driven Architecture in Web Services Description Language (WSDL) and Business Process Execution Language (BPEL). (Zhelev & Rozeva, 2019) discuss the use of microservices and event-driven architecture for processing large volumes of data, while (Mariani & Omicini, 2015) propose an event-driven architecture for situated multi-agent systems.

The transition from monolithic systems to microservices-based architectures is a topic of growing interest. (Valipour et al., 2009) provide an overview of the concepts of software architecture and service-oriented architecture. (Al-Debagy & Martinek, 2019) provide a comparative review of

monolithic and microservices architectures, highlighting the benefits and challenges of each approach. They suggest that while microservices architecture may offer greater flexibility and scalability, it can also present challenges in terms of complexity and management. (Laigner et al., 2020) discuss the transition from a monolithic big data system to an event-driven microservices-based architecture. Similarly, (Djogic et al., 2018) describe the redesign of an event-driven integration platform from a monolithic system to a microservices-based architecture.

These works provide valuable context for the case study presented in this paper, which focuses on restructuring the software architecture of the CoolBiz Platform, with a particular focus on implementing an event-driven architecture.

3. METHODOLOGY

The approach taken in this study was thorough and expansive, including a literature review, source code analysis, developer interviews, and comparative analysis.

To begin, a comprehensive review of the literature was conducted, with an emphasis on analyzing documentation and source codes found in various frameworks and architectures. This process aimed to comprehend current practices and determine optimal strategies for developing software architectures in contexts akin to the CoolBiz Platform.

Subsequently, the authors conducted a detailed needs assessment of the CoolBiz Platform. This analysis was conducted through interviews with company senior programmers and senior management members, as well as a review of existing code. The interviews provided a detailed comprehension of the challenges faced by developers, while the analysis of the code provided practical insights into the current solutions.

Following the collection of requirements, a comparative analysis was conducted. This stage involved comparing the different architectures and frameworks that were chosen in accordance with the requirements identified for the CoolBiz Platform. The comparative analysis enabled the authors to identify the pros and cons of each approach, aiding the selection of the most appropriate architectures for the new proposal.

Finally, based on the results of the comparative analysis, the authors designed the architecture for the CoolBiz Platform services framework. This proposal was designed to meet the non-functional requirements identified, such as scalability, high performance, resilience, and traceability.

The methodology used ensured a systematic and rigorous design approach for the architecture, ensuring that the final solution is strong, adaptable, and meets the requirements of the CoolBiz Platform.

4. CoolBiz Platform Analysis

The CoolBiz Platform, as it is currently structured, presents several challenges that need to be overcome. Among the main problems identified are low flexibility, lack of compliance with best practices and design patterns, violation of SOLID principles, lack of unit testing, and lack of proper documentation. In this section, we will examine these problems in detail.

4.1. Analysis of the current CoolBiz Platform structure

The CoolBiz Platform employs a robust service framework, developed in-house. The Coolbiz module communicates with a SQL Server database, in which the system's business logic is located (Figure 1).

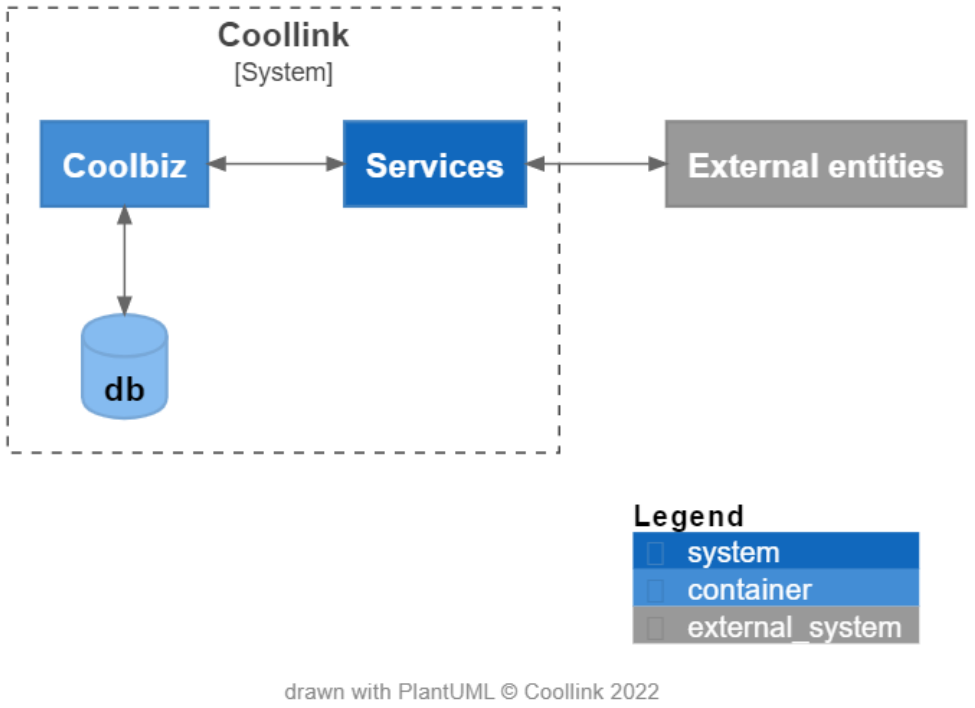

Figure 1 – CoolBiz ecosystem.

The monitoring and control of the services are performed by a controller, which updates the database with relevant information, which can be consulted in a webpage of one of the multiple CoolBiz Platform interfaces, as presented in Figure 2.

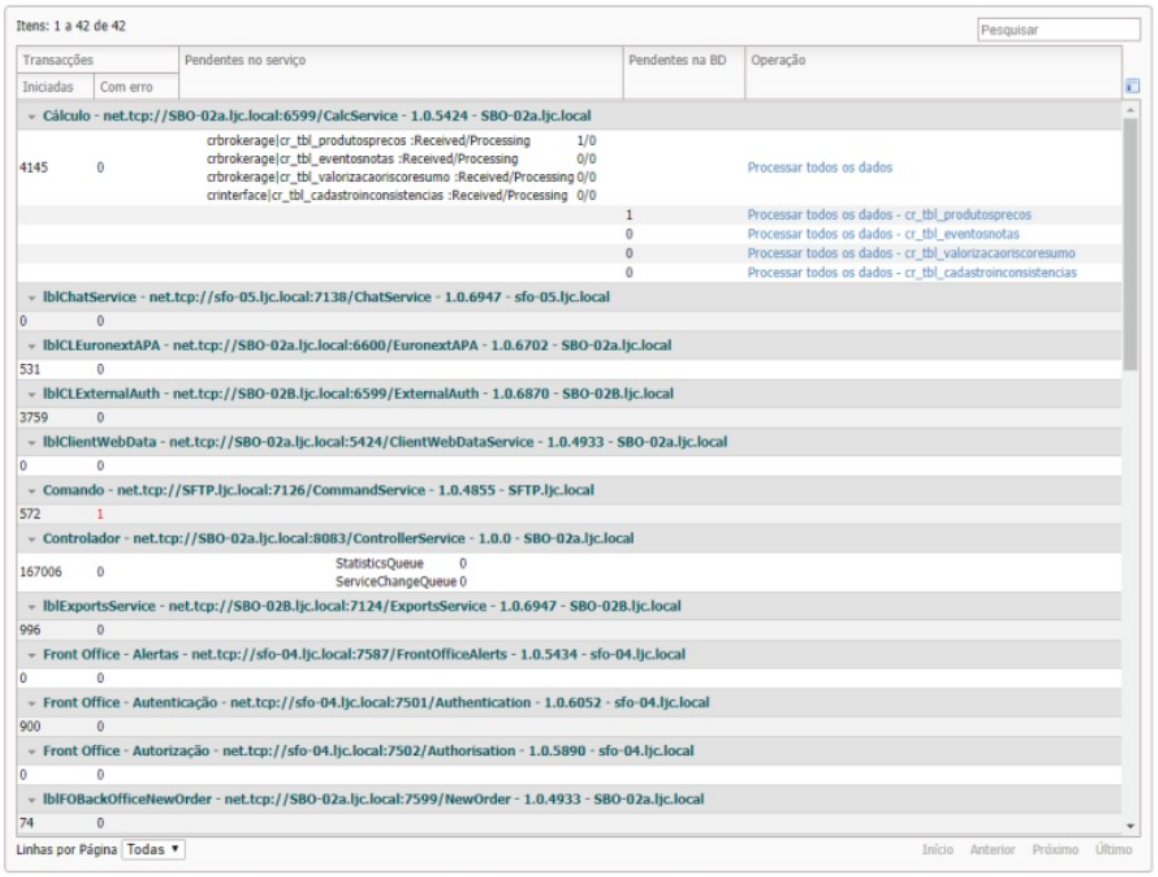

Figure 2 – Service monitoring screen.

The service library ensures communication with the controller, employing the inheritance technique of the object-oriented paradigm. Among the services offered is Log, a Windows Service, whose structure is detailed in Figure 3.

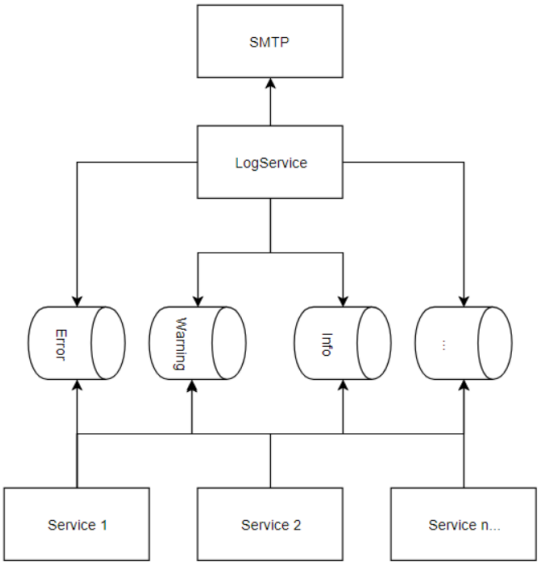

Figure 3 – Structure of the Log service.

4.2. Identified Issues

Through analysis conducted by the authors, multiple flaws in the current service framework were identified and categorized. These flaws indicate several challenges that require attention, including low flexibility, poor cohesion, non-compliance with good practices and design patterns, failure to adhere to SOLID principles, inadequate unit testing, and documentation.

4.2.1. Best Practices and SOLID

The review of the CoolBiz platform source code revealed an excessive use of inheritance, increasing the coupling of systems, which is not recommended in software development. In addition, SOLID principles, software design guidelines aimed at maintaining software comprehensibility, flexibility, and sustainability, are frequently violated.

For example, in the context of the Liskov Substitution Principle (LSP) (Martin, 2000), derived classes fail to adequately replace the base class, limiting the extensibility of the code. Also, violation of the Dependency Inversion Principle (DIP) (Martin, 2000) occurs when the implementation of services is directly bound to specific database information, compromising separation of concerns, and reducing testability.

4.2.2. Missing Unit Tests

Since much of the business logic is currently implemented directly in the database, through stored procedures and triggers, the platform lacks automated testing. This is a serious limitation, especially in critical systems. The practice of unit testing will bring significant benefits, such as increased productivity, reliability, maintainability, and rapid feedback.

4.2.3. High dependence on vendors

The business logic, when implemented directly in the database, ends up coupled strongly with the T-SQL language used in Microsoft's SQL Server. This represents an obstacle for strategic decisions, such as a change of database vendor.

4.2.4. Lack of Documentation

The absence of complete and up-to-date documentation is a significant barrier to developers understanding and maintaining the system. This can result in increased delivery time and a higher likelihood of errors.

In addition, this lack of documentation significantly hinders the onboarding of new team members. Without access to detailed and up-to-date documentation, new developers may face challenges understanding the business logic, code structure, and interactions between the various system

components. This can lengthen the required training period and increase the risk of introducing errors or inconsistencies in the code.

4.3. First Restructuring Attempt

Faced with the challenges of maintenance, onboarding new employees due to a lack of documentation, and difficulties with testing, an employee was assigned to restructure the service framework. The proposed changes, as illustrated in Figure 4, include self-hosting via the Open Web Interface for .NET (Smith & Anderson, 2023), using Swagger to document and test Web APIs, sending error logs by email, and other improvements.

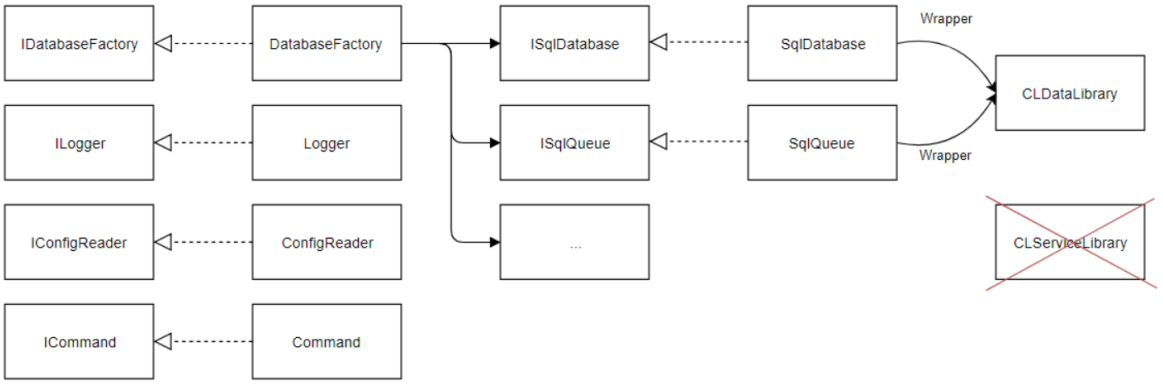

Figure 4 – First proposal for restructuring the service framework.

However, this initial attempt was not implemented due to a few obstacles. First, the company faced resource constraints, including time and personnel, that made it difficult to conduct a complete overhaul of the system. In addition, the restructuring attempt met resistance from internal stakeholders who were used to the existing service framework and were reluctant to adopt a completely new system. Finally, the complex nature of the existing business logic and the high coupling of the system components made the transition to a new framework extremely challenging.

The authors identified that even upon implementation of the proposed changes, the current structure's problems would not be fully resolved. While the proposed restructuring sought to modernize the service architecture and enhance code testability, it failed to adequately address the high coupling caused by the overuse of inheritance in place of composition. In addition, the proposal did not provide a comprehensive resolution to the issue of relying heavily on SQL Server and the T-SQL language. This dependence may still restrict the system's adaptability and pose challenges for incorporating alternative database technologies in the long run.

Moreover, the proposal did not include a clear plan to increase the documentation of the system, which could continue to make code maintenance and onboarding of new developers difficult.

Therefore, while the proposed changes had the potential to improve some aspects of the CoolBiz Platform's service framework, they would still leave several critical issues unresolved.

In the next section, a proposed new architecture for the CoolBiz Platform is presented. This new architecture seeks to effectively address the identified challenges and provide a more robust, scalable, and maintainable solution.

5. ARCHITECTURE PROPOSE

The objective of this study is to present an architecture proposal for CoolBiz Plaftorm's service framework, aiming to address critical non-functional requirements such as scalability, high availability, performance, resilience, and auditability. The proposal derives from a deep analysis of existing architectures, seeking a hybrid framework that can be phased in and adapted to CoolBiz Platform's legacy systems.

5.1. Technical Considerations

The suggested architecture incorporates two main inspirations: the Event-Driven Architecture (EDA) (Clark & Barn, 2011; Juric, 2010), and the Service-Based Architecture (SBA), whose main highlight is its characteristic of integrating with other paradigms, enabling a composition that can result in the architecture of event-driven microservices (Valipour et al., 2009; Zhelev & Rozeva, 2019).

EDA is an asynchronous paradigm composed of independent event processing components that operate in a decoupled fashion. SBA, on the other hand, is somewhere in between Service-Oriented Architecture (SOA) and microservices, facilitating the transition from systems with monolithic architecture (Djogic et al., 2018; Laigner et al., 2020).

EDA can be implemented using two different topologies: broker-based or mediation-based (Mariani & Omicini, 2015). In the broker-based configuration, the broker acts as an intermediary, facilitating communication between events and processors. This model is particularly suitable for events that do not require orchestration (Figure 5). However, this approach has certain limitations such as flow management and error handling that can become complex, making it difficult to recover and restart processors. In addition, there may be a risk of data inconsistency, an unacceptable situation in contexts such as banking sector.

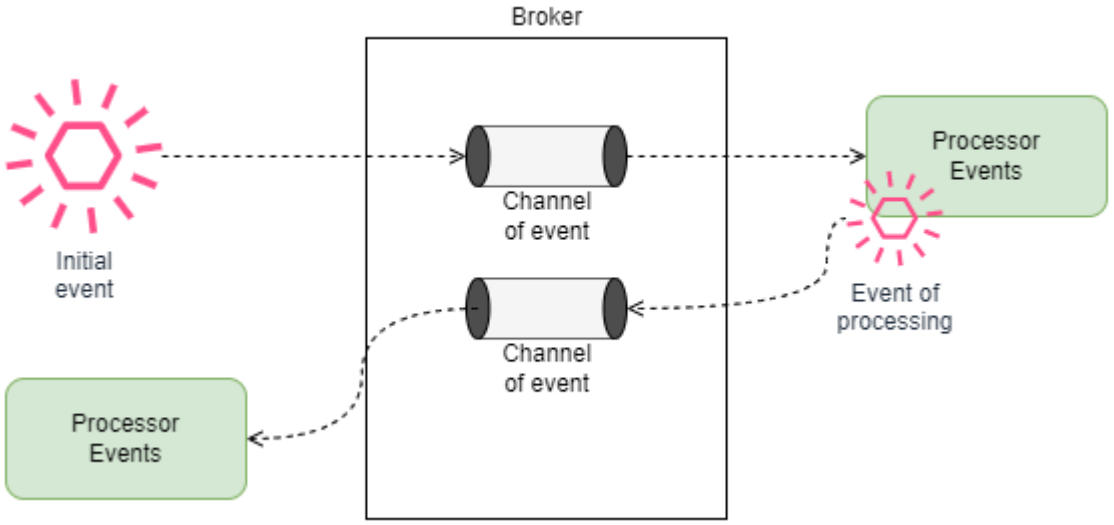

Figure 5 – Event-Driven Architecture - Broker-based Topology.

In contrast, the mediation-based topology offers more enhanced control over transactional flows (Figure 6), operating with a number of mediators equivalent to the number of domains present in the application (Souza & Moraes, 2018).

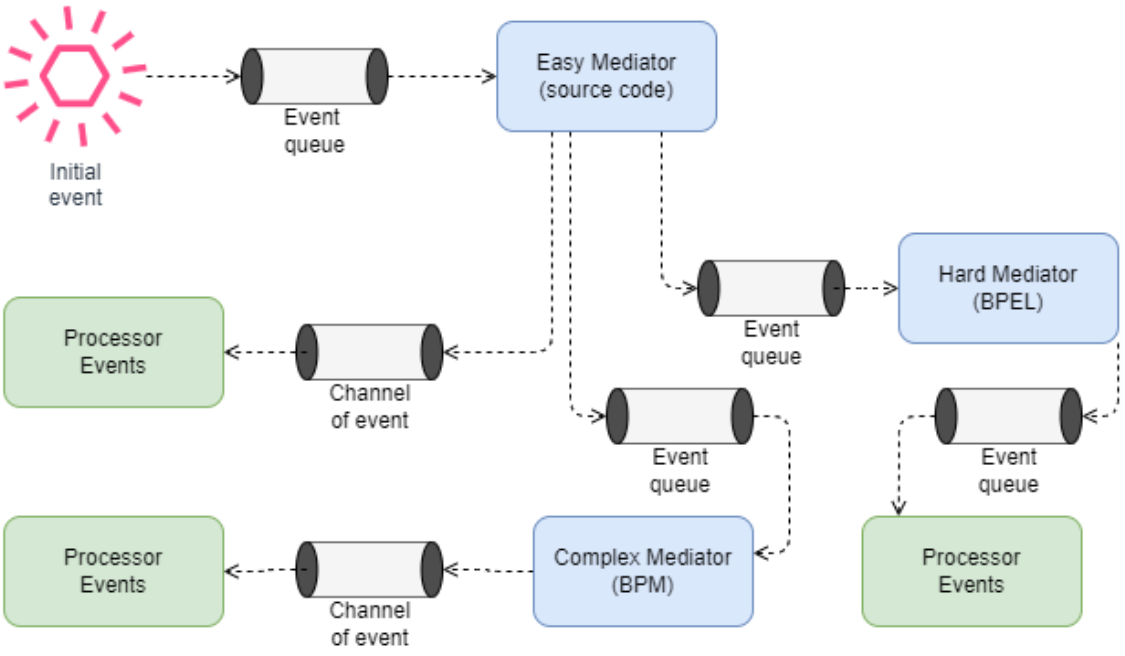

Figure 6 – Event-Driven Architecture - Mediation Topology.

Although this approach has some disadvantages, such as coupling between processors, the need to model complex flows, and potential reduction in performance and scalability, it is still relevant to the design of our architecture because of its flexibility in dealing with multiple domains.

5.2. Business Considerations

The architecture proposal considered more than just technical dimensions. The realities of modern business and the challenges inherent in the software development industry were crucial factors in shaping the solution presented.

One of the pressing problems in the technology industry is the difficulty in hiring and keeping qualified talent. The new generation of developers tends to be attracted not only by adequate compensation, but also by the opportunity to work with modern technologies and have creative freedom to solve problems. The perception of the CoolBiz Platform as a legacy system, as well as all the platforms and systems supporting the traditional banking sector, may not initially seem attractive. However, by employing a modern and robust architecture such as the one proposed, we believe that attracting talent will be made easier, as developers will have the opportunity to work with state-of-the-art technologies and participate in the modernization of an existing system.

Another significant business issue is the high dependency on one vendor, in this case Microsoft SQL Server and the T-SQL query language. During the lifetime of a project, it is crucial to have the freedom to make strategic choices and execute them efficiently. Migration from one vendor to another should be possible and preferably transparent. With the proposed architecture, we seek to minimize dependence on a single vendor by enabling the flexibility needed for smooth adaptation to changing business needs and circumstances.

In addition, the literature on productivity in software projects often discusses the impact of the number of developers in a project, from “The Mythical Man-Month” to Scrum development team culture (Brooks Jr, 1982; Rubin, 2010; Sutherland, 2014). Finding a balance in the allocation of human resources is critical to the success of the project. Therefore, the modularization allowed by the proposed architecture will allow the best distribution of tasks among the team and, consequently, the best use of the available talent.

Finally, a successful architectural proposal needs to consider all the elements that impact the business, not just the technology. Political, economic, social, and legal aspects can directly influence the success of a solution. The proposed architecture was designed to adapt to these variables, offering enough flexibility to face possible challenges in these aspects.

5.3. Architecture Proposed

Based on technical and business considerations, the proposed architecture is based on a combination of Event-Driven Architecture, Service-Based Architecture, microservices and Hexagonal Architecture (also known as "Ports and Adapters") (Cupac, 2023; Sharma, 2022). This combination will enable the overall decoupling of the system, both in terms of programming language and source code vendor or structure.

In addition, the architecture patterns selected will allow the effective division of teams between the services to be developed. With this, the proposed solution will allow the choice of the most appropriate programming language or technology for each situation, which should help overcome the challenges in hiring and retaining talent.

Another fundamental aspect of the proposed architecture is its ability to be implemented in a phased manner. With this approach, it is possible to start with the service framework, maintaining interoperability between all components, and eventually expand to encompass the entire CoolBiz Platform.

In the new architecture, each processing unit can consume events at its own pace and according to its own specific quality metrics. Because this architecture was designed with monitoring and metrics in mind, it provides a native infrastructure for monitoring and managing the performance and availability of the processing units.

In a more detailed view, the proposed new architecture also enables the orchestration of services through the Message Broker. The orchestrator is responsible for receiving the event that requires orchestration and initiate a sequence of events to be processed by the different processors. When a processor finish processing an event, it fires a completion event, which is consumed by the orchestrator, which continues its sequence of operations until it finishes the transactional flow.

Event persistence is managed by the Message Broker, making the system completely auditable and allowing accurate reconstruction of the system state at any time. Figure 7 describes the proposed architecture, as well as exemplifies some possible technologies to be used.

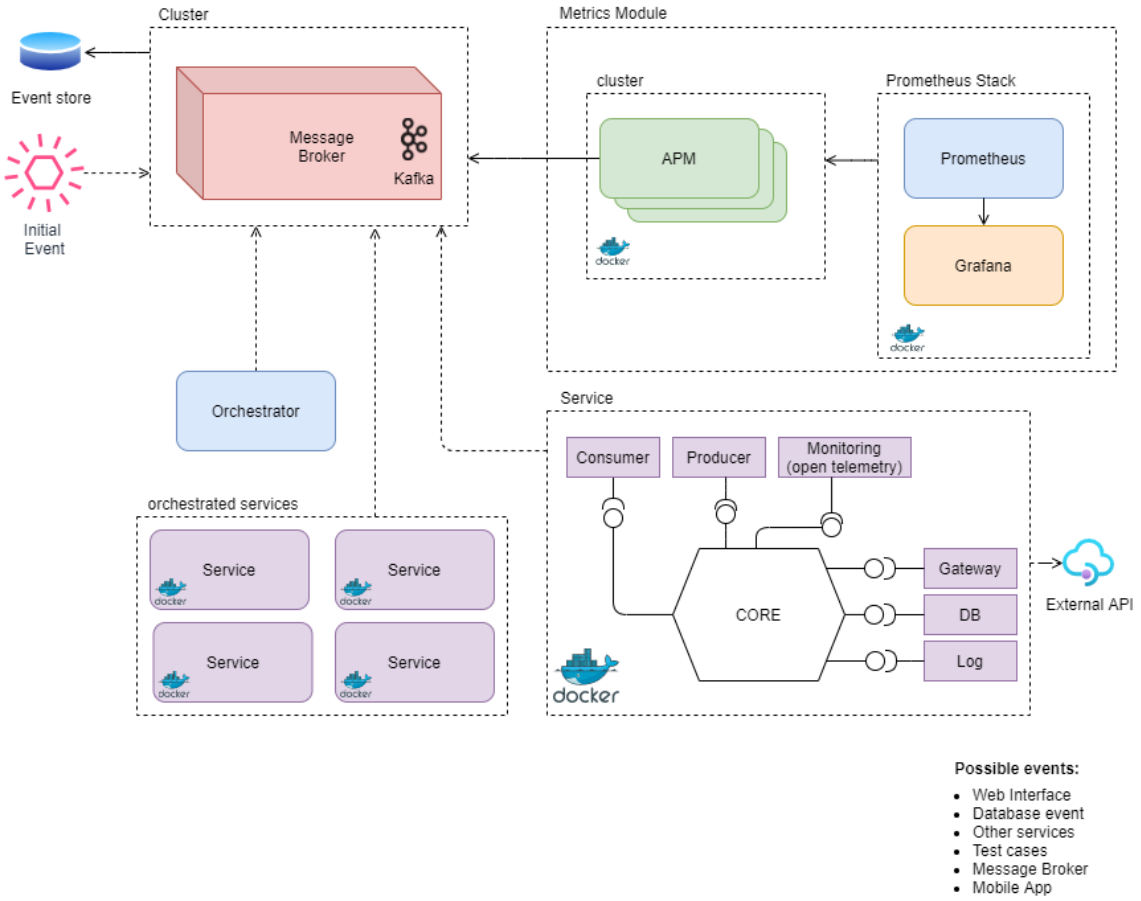

Figure 7 – The proposed architecture.

5.4. Proof-of-Concept

The Proof of Concept (PoC) (Figure 8) was developed with the purpose of testing the effectiveness of the proposed architecture. As an integral part of this process, it was decided to implement an existing service within the system - VatEuCheck. This service has the specific function of checking the validity of a TIN (Tax Identification Number) in any of the member states of the European Union.

This test was performed in an environment that replicated the production one, allowing an accurate evaluation of the service performance regarding the efficiency and effectiveness of the proposed architecture. By implementing an existing service, we were able to better understand how the proposed architecture interacts with existing systems, as well as providing an assessment of its ability to integrate it successfully.

The testing process began with the registration of a new TIN verification request in a specific database table, simulating a routine request made by a system operator. Once the request was registered, the Kafka Connect connector was strategically programmed to capture this information through a non-intrusive approach by monitoring the database logs.

This non-intrusive methodology has considerable advantages. It ends up with the need to configure triggers or stored procedures in the database to capture changes, which could potentially impact database performance and require additional maintenance.

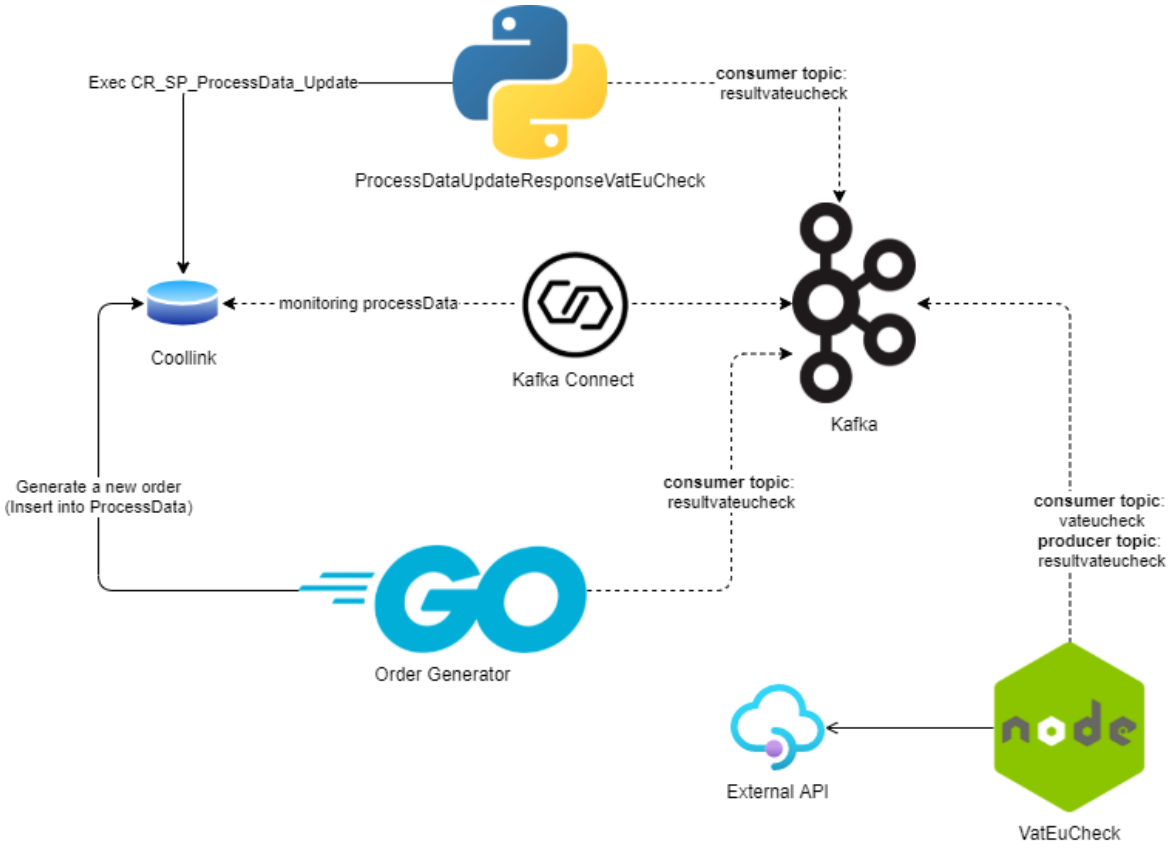

Figure 8 – PoC Architecture.

When it finished processing the result, the VatEuCheck service issued a completion event, containing the processed response. This result event was subsequently consumed by two other services - the Order Generator and the ProcessDataUpdateResponseVatEuCheck.

In this context, the Order Generator was designed to simulate a TIN verification request event by replicating the actions of a system operator. In its modus operandi, this service makes a record in the database, simulating the initial event. In doing so, it not only simulates the actual operating environment but also validates the interoperability of the architecture.

To test the system response in different scenarios, the Order Generator was programmed to create verification requests with a valid TIN (belonging to a Member State) and an invalid TIN (not belonging to a Member State). This cycle was intended to analyze the behavior of the system under high load requests. For each identified response, the Order Generator created two requests,

simulating a high demand environment, and allowing a rigorous stress test. The system performance during this test was exceptional, with no failures even under high stress.

On the other hand, the ProcessDataUpdateResponseVatEuCheck had the task of persisting the query results in the database through a stored procedure. This step was fundamental to validate the ability of the proposed architecture to integrate with existing data processes and to confirm the complete auditability of the system.

The Proof-of-Concept implementation was deliberately modularized, even when it would have been possible to consolidate the functionality into a single service. This decision was made in order to demonstrate that the solution is capable of maintaining exemplary performance even when broken down into multiple components.

Crucially, the modular approach not only proved the efficiency of the architecture in more complex situations, but also confirmed its agnosticism with respect to the programming language used. This was evidenced by the fact that it was possible to integrate multiple programming languages within the same project, thus allowing the selection of the language best suited to handle each specific task.

In the end, the PoC successfully demonstrated the ability of the proposed architecture to be implemented without the need to change the legacy system. This confirms the feasibility of a phased migration, as mentioned earlier, thus validating the proposed architecture for the CoolBiz Platfotm's service framework.

6. CONCLUSION

This study describes the overhaul of the software architecture for the CoolBiz Platform, an integrated Core Banking solution. The evaluation of the previous architecture revealed challenges such as limited flexibility, inadequate cohesion, non-compliance with SOLID principles, inadequate unit testing, and insufficient documentation. Although typical in complex and regulated sectors like banking, these challenges hinder efficiency and innovation.

This work highlights the significance of taking a holistic approach when discussing software architecture, beyond purely technical matters and including factors such as talent attraction and retention. Reflection on these aspects is crucial in academia, given their impact on professional training and discipline advancement. In addition, many organizations are faced with the reality of migrating legacy systems, necessitating the exploration of new perspectives to ensure business continuity and innovation.

To overcome these challenges, an architecture based on Event-Driven Architecture, Service-Based Architecture, microservices, and Hexagonal Architecture has been proposed. This suggested solution offers robustness and scalability while increasing flexibility, cohesion, adherence to SOLID

principles, test coverage, and documentation. The ability to implement in stages and language agnosticism are intrinsic benefits of the proposed solution.

The proof-of-concept, which included the transition of a service to the new architecture, validated the effectiveness of the solution. The findings suggest that the architecture not only meets the technical challenges, but also confers business advantages, such as scalability and reduced dependence on suppliers.

However, it is important to acknowledge that architectural restructuring is a complex process that requires significant resource investment and meticulous change management. Obstacles such as resistance to change and the high interdependence of components are inherent to the process.

A significant limitation of this study is its reliance on subjective perceptions due to the absence of objective metrics, particularly in the proof-of-concept. The absence of a metrics system in the legacy system can restrict a full understanding of the impact of restructuring.

Future research should include the development of a detailed implementation plan covering all stages and potential challenges. Investigating the impact of the new architecture on attracting and retaining talent would be valuable in providing insights for organizations seeking innovation and sustained growth.

In conclusion, this study enhances the literature on software architecture by clarifying the restructuring of a legacy banking system. The findings indicate that implementing a comprehensive and methodical approach can lead to improvements in the system's adaptability and maintainability, while also providing strategic advantages to the organization.

ACKNOWLEDGMENT

We would like to express our deep gratitude to the company Coollink, responsible for the creation of the CoolBiz system, which played a crucial role in the development of this work, in particular to Rui Silva whose insights on the actual and proposed architecture were extremely valuable. In addition, we would like to give special thanks to Leonardo Martinez Lopez, with whom Jackson developed the initial work that formed the basis of this paper, still in the scope of the curricular internship of the degree in Computer Engineering.

REFERENCES

Al-Debagy, O., & Martinek, P. (2019). *A Comparative Review of Microservices and Monolithic Architectures*. http://arxiv.org/abs/1909.02742

Brooks Jr, F. P. (1982). *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley Pub. Co.

Clark, T., & Barn, B. S. (2011). Event driven architecture modelling and simulation. *Proceedings of 2011 IEEE 6th International Symposium on Service Oriented System (SOSE)*, 43–54.

Coelho, H. S. (2009). Documentação de software: uma necessidade. *Texto Livre: Linguagem e Tecnologia*, *2*(1), 17–21.

Cupac, V. (2023). *Hexagonal Architecture - Ports and Adapters*. Optivem Journal. https://journal.optivem.com/p/hexagonal-architecture-ports-and-adapters

de Souza, S. C. B., das Neves, W. C. G., Anquetil, N., & de Oliveira, K. M. (2007). Documentação essencial para manutenção de software ii. *IV Workshop de Manutenção de Software Moderna (WMSWM), Porto de Galinhas, PE*.

Djogic, E., Ribic, S., & Donko, D. (2018). Monolithic to microservices redesign of event driven integration platform. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 1411–1414.

Juric, M. B. (2010). WSDL and BPEL extensions for Event Driven Architecture. *Information and Software Technology*, *52*(10), 1023–1043.

Laigner, R., Kalinowski, M., Diniz, P., Barros, L., Cassino, C., Lemos, M., Arruda, D., Lifschitz, S., & Zhou, Y. (2020). From a monolithic big data system to a microservices event-driven architecture. *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 213–220.

Mariani, S., & Omicini, A. (2015). Coordinating activities and change: An event-driven architecture for situated MAS. *Engineering Applications of Artificial Intelligence*, *41*, 298–309.

Martin, R. C. (2000). *Design Principles and Design Patterns*. Object Mentor Inc.

Nassif, M., Hernandez, A., Sridharan, A., & Robillard, M. P. (2020). *Generating Unit Tests for Documentation*. http://arxiv.org/abs/2005.08750

Palma, N. A. (2007). *Testes unitários evolutivos*. Universidade de Évora.

Petersen, K., & Wohlin, C. (2009). A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. *Journal of Systems and Software*, *82*(9), 1479–1490.

Rubin, M. (2010). *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley Professional.

Sharma, G. (2022). *Designing Hexagonal Architecture with Java: An architect's guide to building maintainable and change-tolerant applications with Java and Quarkus*. Packt Publishing.

Smith, S., & Anderson, R. (2023). *Open Web Interface for .NET (OWIN) with ASP.NET Core*. Microsoft Learn. https://learn.microsoft.com/en-us/aspnet/core/fundamentals/owin?view=aspnetcore-7.0

Souza, L. P. P., & Moraes, C. R. B. (2018). Conhecimento explícito, desafios produtivos e desenvolvimento de software. *XIX Encontro Nacional de Pesquisa em Ciência da Informação (XIX Enancib)*, *24*(2).

Sutherland, J. (2014). *Scrum: The Art of Doing Twice the Work in Half the Time*. Crown Business.

Valipour, M. H., AmirZafari, B., Maleki, K. N., & Daneshpour, N. (2009). A brief survey of software architecture concepts and service oriented architecture. *2009 2nd IEEE International Conference on Computer Science and Information Technology*, 34–38.

Zhelev, S., & Rozeva, A. (2019). Using microservices and event driven architecture for big data stream processing. *AIP Conference Proceedings*, *2172*(1), 90010.