DOI: 10.32347/2412-9933.2025.64.225-230
UDC 004.05: 004.451.83:004.378

Tsiutsiura Mykola
DSc (Eng.), Professor, Professor of the Department of Software Engineering and Cybersecurity,
https://orcid.org/0000-0003-4713-7568
State University of Trade and Economics, Kyiv
Makoiedova Valentyna
Ph.D, Associate Professor of the Department of Digital Economy and System Analysis,
https://orcid.org/0000-0001-7518-894X
State University of Trade and Economics, Kyiv
Tsiutsiura Svitlana
DSc (Eng.), Professor, Professor of the Department of Software Engineering and Cybersecurity,
https://orcid.org/0000-0002-4270-7405
State University of Trade and Economics, Kyiv
Kryvoruchko Olena
DSc (Eng.), Professor, Professor of the Department of Computer Systems, Networks and Cybersecurity,
https://orcid.org/0000-0002-7661-9227
National University of Life and Environmental Sciences of Ukraine, Kyiv

ARCHITECTURAL TEMPLATES WITH BUILT-IN METHODOLOGY FOR DEVELOPING ERP SOLUTIONS

Abstract. *Choosing a service-oriented architecture for modern software applications provides a wide range of significant advantages that directly affect the quality, reliability, and sustainability of information systems. Among the key benefits of this architectural approach are simplified system maintenance, improved fault tolerance, increased reliability, accelerated development cycles, and enhanced flexibility of software solutions in response to changing business and technological requirements. By decomposing an application into independent services with clearly defined responsibilities and interfaces, service-oriented architecture enables teams to develop, deploy, and scale system components more efficiently and with lower operational risks. Despite its advantages, the implementation of a service-oriented architecture is associated with a number of challenges and potential issues. These include increased system complexity, difficulties in coordinating interactions between distributed services, latency in inter-service communication, and the need for reliable mechanisms to ensure data consistency and fault handling. Without proper architectural support, such systems may become difficult to manage, debug, and extend. The purpose of the work is to reveal the specifics of using event generation and processing mechanisms, which allow services to interact asynchronously and respond to changes in the system state in a timely and controlled manner. Experience with asynchronous and event-driven architectural approaches demonstrates that the application of well-established architectural patterns makes it possible to design software systems that operate smoothly even under high load and in dynamic execution environments. Event-based interaction models reduce tight coupling between services, enabling them to evolve independently and improving the overall resilience of the system. This approach also supports better scalability, as system components can be replicated or redistributed without significant changes to the core architecture. Furthermore, asynchronous service-oriented architectures provide favorable conditions for the incremental extension of system functionality. New services, business processes, or integration components can be added without disrupting existing system operations, which is particularly important in long-term software projects with evolving requirements. The use of standardized communication protocols, message brokers, and event generators ensures consistency, reliability, and transparency of interactions across the system. The results of the analysis confirm that the combination of service-oriented architecture principles with asynchronous event-driven mechanisms forms a robust foundation for building scalable, adaptable, and maintainable software applications. Such an architectural approach allows developers to balance system complexity with flexibility, ensuring high software quality and long-term effectiveness in rapidly changing technological environments.*

Keywords: software architecture; modification, extensibility, scalability, software quality; system development architecture

Introduction

Choosing a service-oriented architecture (SOA) for modern software applications provides a number of significant advantages, including simplified maintenance, increased reliability, accelerated development cycles, and improved flexibility of software systems in response to changing requirements. A well-designed architecture allows developers to decompose complex systems into independent services with clearly defined responsibilities, which facilitates parallel development and improves overall system stability [1; 5; 8 – 10].

At the same time, the implementation of service-oriented architectures is accompanied by a number of challenges and potential problems related to the coordination of distributed components, management of asynchronous interactions, and ensuring data consistency. These challenges can be effectively addressed through the use of appropriate event generators and orchestration mechanisms. Practical experience with asynchronous and event-driven architectures shows that, by applying proven architectural patterns, it is possible to design software applications that operate smoothly, are easily scalable, and allow new functions or business processes to be added without excessive effort or architectural restructuring [2; 5].

Objective of the work

The concept of service-oriented architecture

Today, a popular architectural approach for developing software applications based on individual modules is service-oriented architecture (SOA).

Synchronous SOA assumes that the system sends a request and expects an immediate response, so in this case, communication between services does not require this. This means that the client can send a request and move on to other tasks without waiting for a response.

Asynchronous SOA assumes that each service is autonomous and performs a separate task. In such an architecture, the interaction between services does not require an immediate response, which allows the client to send a request and continue other tasks without waiting for an instant response. For example, when developing an application for forming an educational trajectory, after forming one request, the user can continue to view resources or make other requests without waiting for the previous request to be formed and played back. When the request is processed, the user will receive a notification (Fig. 1) [6 – 10]. This ability to send a request and receive a response at an indefinite time is a characteristic feature of asynchronous SOA.

Summary of the main material

Challenges in implementing asynchronous service-oriented architecture

Developing asynchronous service-oriented architectures comes with a number of potential challenges.

Managing the state of each individual process. It can be difficult to track and manage the state of each individual process. After placing a request, it goes through various stages: confirmation, preparation, educational service, preparation, delivery. Each of these stages is an independent event, and some of them can occur simultaneously.

Accurately managing the state of each event in such distributed processes is a difficult task. For example, if the program shows that the request is only confirmed, when in fact it is already ready to ship, this can lead to confusion and customer user.

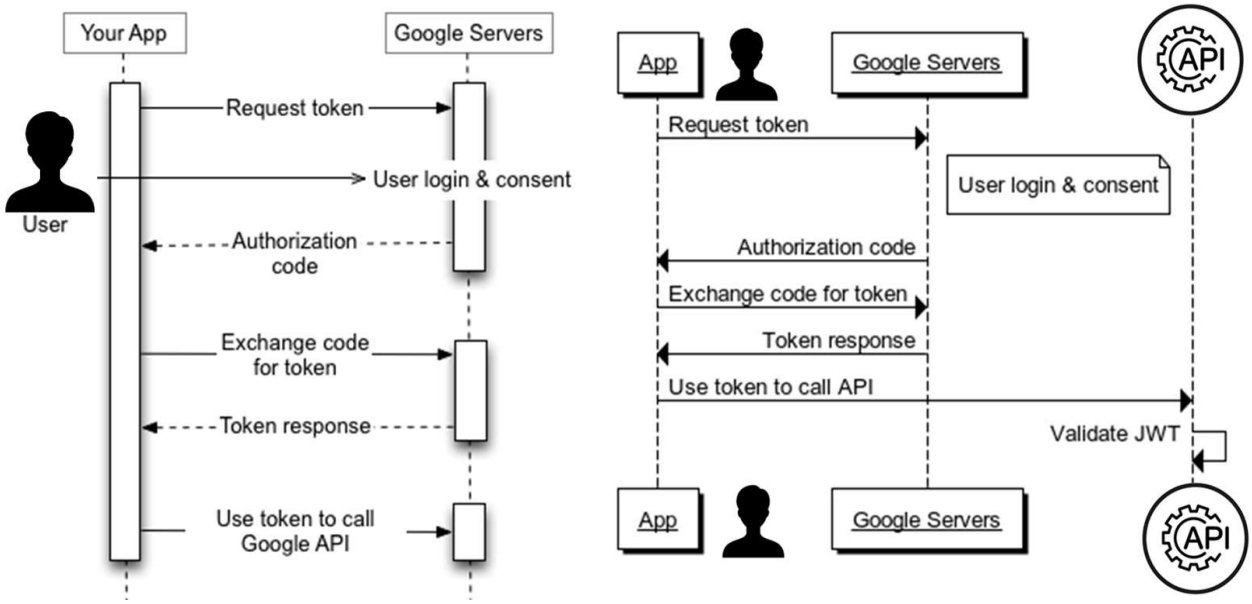

Figure 1 – Synchronous and asynchronous architectural patterns for building software applications by individual modules

The image below is an example of what the architecture of such event-driven software might look like. Whenever an event occurs, it will trigger a new operation. So, essentially, instead of a sequence of steps, the system executes events X, Y, and Z separately, without really understanding the sequence of steps that occur (Fig. 2). [1; 4].

Fault tolerance and maintainability. If a request is rejected during the work process or the system is not notified of a new request, this may cause the entire process to stop. Therefore, it is important to provide effective error handling mechanisms that will increase the reliability of the software. For example, the system can retry a failed transaction or notify the user of the problem and offer to return to the previous step. Asynchronous systems are inherently complex. Over time, as they develop and scale, this complexity only increases. For example, if you want to add a priority delivery feature for customers, a poorly designed system may require significant changes to many components, which will complicate the development and maintenance process [2 – 5]. Maintaining maintainability means that the system allows for easy changes, extensions, and scaling without disrupting its operation and without excessive effort.

To increase fault tolerance, various errors may occur during the request processing process, such as deviations in the formation of an educational trajectory or a notification that the system did not receive the request. Orchestrators have built-in mechanisms to automatically retry failed operations or apply alternative strategies, allowing the system to continue operating without disruption. For example, in the event of a problem with the lighting system the orchestrator can configure the number of retries and the intervals between them, ensuring flexibility and reliability of the process.

System tracing and monitoring. An educational environment system such as Uber Eats involves many components: the client application, lighting system and back-end services. Tracing a specific process or identifying performance bottlenecks can be a difficult task. For example, if a client reports that they have not received their trajectory, although the application shows that it has been formed, without proper monitoring it is difficult to determine at what stage the error occurred - processing and formation of the request. [3, 5]. In the diagram we saw a system that was completely event-driven, now orchestrators take care of the sequencing of all events.

The Role of Orchestration in Solving Asynchronous SOA Problems. Orchestration is a key element in ensuring the efficiency, reliability, and scalability of asynchronous service-oriented architectures. It acts as a central coordinator that manages all processes, similar to a conductor ensuring the smooth operation of an orchestra (Fig. 3) [3; 5].

Whereas in the previous diagram we saw a system that was completely event-driven, now orchestrators take care of the sequence of all events.

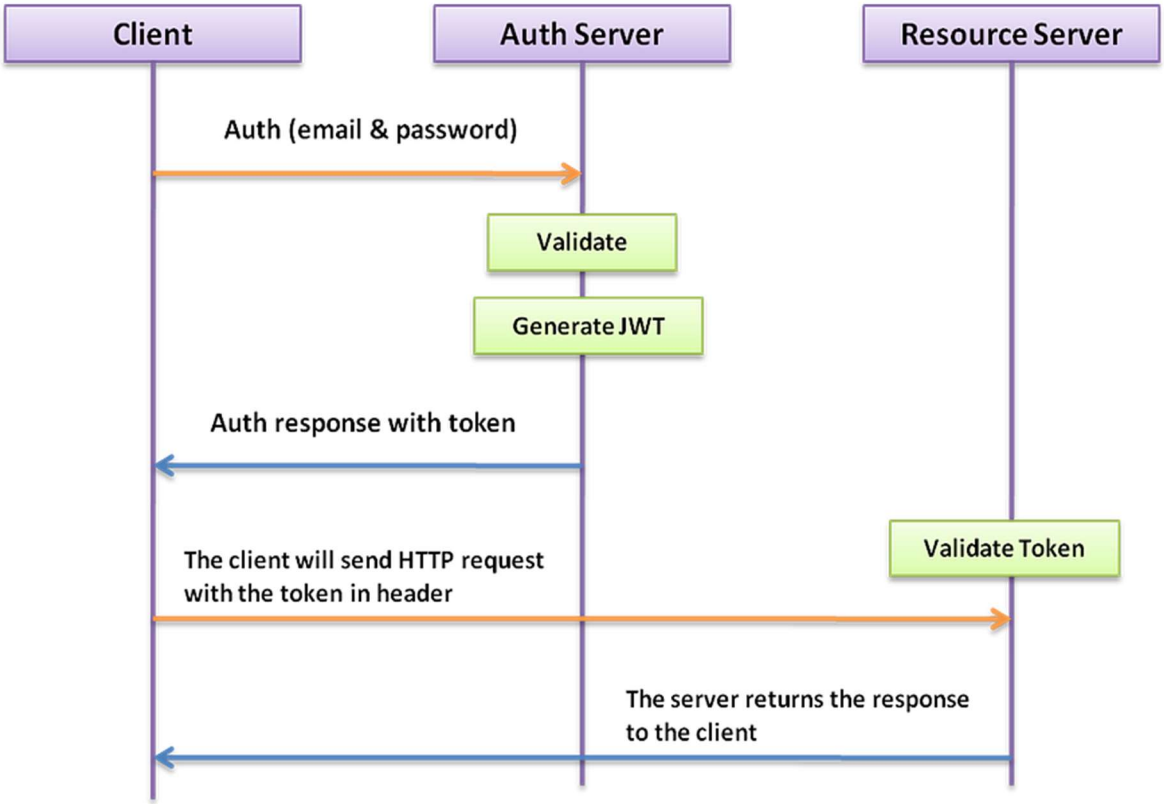

Figure 2 – Basic Authentication flow of JWT [1; 4]

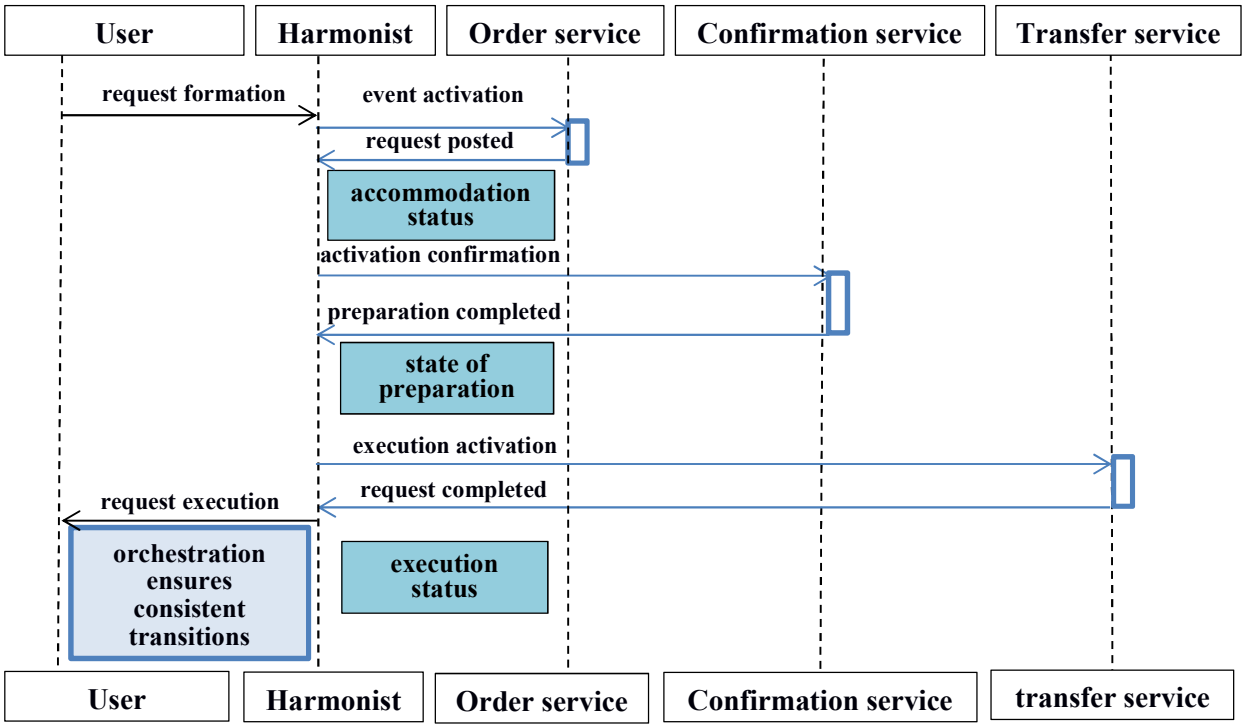

Figure 3 – Architecture Orchestrators take care of the sequence of all events [3; 5]

Sequence management Advanced system monitoring capabilities

In complex systems such as educational platforms, an order goes through various stages: confirmation, preparation, and delivery. Without proper control of the sequence of these stages, situations can arise when events are performed in the wrong order, leading to confusion and customer dissatisfaction. Orchestrators use predefined workflows that clearly define the order and dependencies between tasks, ensuring consistency and accuracy of information for all participants in the process.

In multi-component systems such as educational services, it is important to be able to track the status of each process. Orchestrators provide tools for detailed monitoring and tracing, which allows teams to quickly identify and fix problem areas. This provides transparency of processes and helps increase overall software efficiency.

There are several popular orchestration tools, each of which has its own features:

1. Apache Airflow. A flexible open-source orchestrator where tasks and their sequence are described in Python. Suitable for data processing tasks and scenarios with frequent workflow changes.

2. Argo. A native Kubernetes orchestrator designed for cloud environments. Each workflow step is executed in a separate container, providing flexibility and fault tolerance.

3. Temporal. A simple and reliable open-source orchestrator that supports multiple programming languages. Useful for complex business logic with many branches and built-in failure handling.

4. AWS Step Functions. A managed orchestrator from Amazon Web Services that simplifies the orchestration of complex multi-stage applications using visual workflows and easily integrates with other AWS services.

Embedded ERP solutions in system development

Changes occurring both in the economic life of the country and in the higher education system could not but affect the requirements that are imposed on the information system of a modern Russian higher education institution.

Changes taking place both in the economic life of the country and in the system of higher education institutions could not but affect the requirements for the information system of a modern Russian higher education institution.

The time of information systems, the sole task of which was the automation of the operational activities of functional units of higher education institutions, is a thing of the past. Such an approach, when the tasks of building an information system were formulated mainly by specialists in the field of information technology, cannot provide the management of higher education institutions with an effective tool for making management decisions

and ensure effective management of higher education institutions as a whole, relying not only on internal data of higher education institutions, but also on information available from external sources.

The purpose of the "Project Preparation" phase is preliminary planning and preparation of the software implementation project. ASAP contains numerous tools, such as: "how to do something" instructions, questionnaires, templates and checklists that save time and ultimately project costs. ASAP contains a detailed project plan, including task descriptions that explain in detail how to complete a specific task. The ASAP project plan gives an advantage during implementation and ensures that all important tasks are included in the plan. The management of higher education institutions implementing the system will be assisted in conducting all necessary checks in accordance with the provisions of the project plan, as well as in the development of project standards.

The purpose of the "Conceptual Design" phase is to collect requirements for the system's business processes that are necessary to support the management tasks of a specific HEI. The "Conceptual Design" phase completes the definition of the scope of the software implementation project [1 – 5].

This phase also includes the installation of software and the installation of development equipment. At this stage, in most cases, serious shortcomings in the organization of the work of a number of departments are revealed, especially in those processes where interaction with other departments occurs:

– Lack of process implementation regulations and responsibility for the final result.

– A number of functions are duplicated in several departments, which is often the main source of data inconsistency.

– Many logically related functions are performed in different units, and, conversely, a number of units perform functions that are not inherent to them.

Conclusions

1. A correctly selected orchestration tool helps create stable, scalable and easy-to-maintain applications. The choice of a suitable orchestrator depends on the specifics of the tasks, scalability requirements and integration with the existing infrastructure.

2. Orchestrators provide a clear structure of workflows, which simplifies understanding and modification of the software. As the system develops, there is a need to make changes or add new functions. This allows you to quickly adapt to new requirements without the risk of disrupting the operation of other components, ensuring flexibility and scalability of the architecture.

3. By abstracting the complexity of asynchronous processes, orchestrators allow developers to focus on the business logic and core functionality of applications. This reduces development and time-to-market by reducing the burden of infrastructure management and error handling.

References

1. Eeles, P. (2006). *What is a software architecture?* IBM Developer. https://www.ibm.com/developerworks/rational/library/feb06/eeles/index.html

2. Tecnovy. (2025). *Top 10 software architecture & design patterns of 2025*. https://tecnovy.com/en/top-10-software-architecture-patterns

3. Kralicek, E. (2016). *The accidental sysAdmin handbook: A primer for early level IT professionals* (1st ed.). Apress.

4. Plakalović, D., & Simić, D. (2021). Applying MVC and PAC patterns in mobile applications. *Journal of Computing*.

5. Richards, M. (2015). *Software architecture patterns*. O'Reilly Media.

6. Nesterenko, O. V. (2019). *Enterprise management information systems: Textbook*. UkrNC.

7. Pichkur, G., & Frolov, O. (n.d.). *How ERP for architects if.team became the basis for designing the future*. If.team. https://if.team/info/uk/cases/yak-erp-dlya-arhitektoriv-if-team-stav-osnovoyu-dlya-proyektuvannya-majbutnogo/

8. Tsiutsiura, M., Yerukaiev, A., & Lyashchenko, T. (2020). Balancing the educational space. Main elements of a comprehensive model for assessing the quality of education. *Management of Development of Complex Systems*, 43, 142–147. https://doi.org/10.32347/2412-9933.2020.43.142-147

9. Tsiutsiura, M. I., Tsiutsiura, S. V., & Kryvoruchko, O. V. (2019). *Information technologies for the development of the content of education* [Monograph]. CP «Comprint».

10. Nikolajchuk, O. A., Pavlov, A. I., & Yurin, A. Y. (2010). Component approach: Production expertise system module. *Software Products and Systems*, 3, 41–44.

The article has been sent to the editorial board 30.11.2025

Цюцюра Микола Ігорович
Доктор технічних наук, професор, професор кафедри інженерії програмного забезпечення та кібербезпеки,
https://orcid.org/0000-0003-4713-7568
Державний торговельно-економічний університет, Київ
Макоєдова Валентина Олександрівна
Ph.D., доцентка кафедри цифрової економіки та системного аналізу,
https://orcid.org/0000-0001-7518-894X
Державний торговельно-економічний університет, Київ
Цюцюра Світлана Володимирівна
Докторка технічних наук, професорка, професорка кафедри інженерії програмного забезпечення та кібербезпеки,
https://orcid.org/0000-0002-4270-7405
Державний торговельно-економічний університет, Київ
Криворучко Олена Володимирівна
Докторка технічних наук, професорка, професорка кафедри комп'ютерних систем, мереж та кібербезпеки,
https://orcid.org/0000-0002-7661-9227
Національний університет біоресурсів і природокористування України, Київ

АРХІТЕКТУРНІ ШАБЛОНИ ІЗ ВБУДОВАНОЮ МЕТОДОЛОГІЄЮ РОЗРОБКИ ERP-РІШЕНЬ

Анотація. *Вибір сервіс-орієнтованої архітектури (SOA) для сучасних програмних систем забезпечує суттєві переваги у якості, надійності та сталості ПЗ. Ключовими перевагами є спрощене обслуговування, висока відмовостійкість, прискорення розробки та адаптивність до змін бізнес-вимог. Декомпозиція додатка на незалежні сервіси з чіткими інтерфейсами дозволяє ефективно масштабувати компоненти та мінімізувати операційні ризики. Разом з тим, впровадження SOA супроводжується складністю координації розподілених сервісів, затримками в комунікаціях та потребою у надійних механізмах узгодженості даних. Без належної архітектурної підтримки такі системи стають складними для керування та налагодження. Метою роботи є дослідження специфіки механізмів генерації та обробки подій, що забезпечують асинхронну взаємодію сервісів і контрольовану реакцію на зміни стану системи. Застосування перевірених подієво-орієнтованих патернів дозволяє проєктувати системи, стійкі до високих навантажень. Моделі взаємодії на основі подій зменшують зв'язність сервісів, сприяючи їх незалежному розвитку та масштабованості без радикальних змін структури. Асинхронні SOA-рішення також створюють умови для інкрементального розширення функціоналу без зупинки існуючих процесів, що критично для довгострокових проєктів. Використання брокерів повідомлень і стандартизованих протоколів гарантує надійність та прозорість взаємодій. Результати аналізу підтверджують, що поєднання принципів SOA з асинхронними механізмами є надійним фундаментом для створення адаптивного та якісного ПЗ у динамічному технологічному середовищі.*

Ключові слова: архітектура програмного забезпечення; модифікація; розширюваність; масштабованість; якість програмного забезпечення; архітектура розробки системи

Link to publication

APA Tsiutsiura, M., Makoiedova, V., Tsiutsiura, S., & Kryvoruchko, O. (2025). Architectural patterns with built-in ERP system development methodology. *Management of Development of Complex Systems*, 64, 225–230, dx.doi.org\10.32347/2412-9933.2025.64.225-230.

ДСТУ Цюцюра М. І., Макоєдова В. О., Цюцюра С. В., Криворучко О. В. Архітектурні шаблони із вбудованою методологією розробки ERP-рішень. *Управління розвитком складних систем.* Київ, 2025. № 64. С. 225 – 230, dx.doi.org\10.32347/2412-9933.2025.64.225-230.