Article

Not peer-reviewed version

Composite Microservice Architecture of the Digital Twin

Eleonora Koltsova *, Maksim Pysin, Alexey Lobanov, Anatoly Antipov, Alexey Arkhipov, Anton Perekatov, Roman Krasheninnikov

Posted Date: 18 June 2026

doi: 10.20944/preprints202606.1372.v1

Keywords: digital twin; composite architecture; microservice architecture; 3D modeling; automated visualization; SCADA

 doi:10.20944/preprints202606.1372.v1

Article

Composite Microservice Architecture of the Digital Twin

Eleonora Koltsova [1,*], Maksim Pysin [1], Alexey Lobanov [1], Anatoly Antipov [2], Alexey Arkhipov [1], Anton Perekatov [3] and Roman Krasheninnikov [1]

[1] Department of Information and Computer Technologies, Mendeleev University of Chemical Technology of Russia, Moscow, 125047, Russia
[2] Department of Electroactive Materials and Chemical Power Sources, Mendeleev University of Chemical Technology of Russia, Moscow, 125047, Russia
[3] Department of Technologies of Inorganic Substances and Electrochemical Processes, Mendeleev University of Chemical Technology of Russia, Moscow, 125047, Russia
* Correspondence: koltsova.e.m@muctr.ru; Tel.: + 7-495-495-21-26

Abstract

This paper presents a study on the use of a microservices-based composite architecture for building a digital twin. It substantiates the importance of consciously choosing an architecture for a digital twin system. It proposes considering the twin as a high-order, multi-agent, distributed system that incorporates similar systems. It proposes using a microservices-based composite architecture as the foundation for building such systems. The key aspects and advantages of using such an architecture are described, noting its flexibility, scalability, and integrability with existing solutions. Potential for reducing cognitive complexity and building a conveyor-based organization for the production of the system itself is highlighted. A digital twin architecture is proposed, accompanied by a diagram and additional clarification of internal rules. The need for system components to comply with a number of responsibilities for the correct operation of such architectures is noted: contract persistence, environmental persistence, responsibility persistence, versioned changes persistence, and documentation completeness persistence. The possibility of using Domain-Driven Design (DDD) as a basis for dividing the system into subsystems is separately noted.

Keywords: digital twin; composite architecture; microservice architecture; 3D modeling; automated visualization; SCADA

1. Introduction

The Industry 4.0 concept represents a new stage in the development of industrial technological maturity. Its name alludes to scientific and technological revolutions and suggests the active phase of the fourth revolution [1]. The historical context of these processes is linked to the gradual transition from automation and the fragmented application of IT technologies in industry to networked multi-agent systems based on a comprehensive understanding of data collection and processing methods [2].

According to several authors, [3–5] the key factor in this revolution is the restructuring of value chains using digital technologies. Practical implementation of this concept utilizes deep integration of hardware and software, thereby creating various horizontal and vertical connections across production, logistics, and consumption. Through the use of modern data collection and processing solutions, autonomous decisions become the foundation of the process. The use of digital models of production processes enables greater understanding and transparency at every stage.

 doi:10.20944/preprints202606.1372.v1

Thus, the task is set to achieve such key properties as interoperability, information transparency and decentralization of decision-making, combined with their efficiency and reliance on technical means of assistance to operators at different levels [4].

Smart manufacturing, which integrates a range of modern technologies and architectures to enhance flexibility and efficiency, is considered the practical embodiment of Industry 4.0 principles [6]. These industries are based on cyber-physical systems and the Internet of Things, which enable the collection and processing of large amounts of data, which is then processed by AI and Big Data-based solutions [7,8]. From a practical perspective, they enable increased availability of physical and economic assets, as well as the adaptation of final products to the needs of a changing market. This is facilitated by the adaptability of production lines, the ability to flexibly and quickly adjust or redesign them in response to changing market demands, personnel autonomy, and integrated supply chains [9,10].

However, the widespread integration of Industry 4.0 technologies is hampered by a number of significant factors, the most significant of which are information security threats, a shortage of specialized competencies, and the lack of unified solutions for cross-standard interoperability [10].

A related and rapidly developing technology that applies the conceptual foundations of Industry 4.0 is digital twins at various levels. A digital twin is defined as a live virtual representation of a physical object or process with two-way data communication used for monitoring, simulation, and optimization [11–14]. It acts as a link between cyberspace and the physical world, actively leveraging the same technologies as smart factories [15]. The twin is often an integral part of a smart factory, while conceptually and technologically developing in parallel [16]. The key objective of digital twins is real-time modeling for identifying and predicting failures, as well as optimizing the entire product lifecycle, from supply chains and production to demand determination [11].

The implementation of Industry 4.0, smart factories, and digital twins is directly linked to their economic effectiveness. Thus, all of them are characterized by significant costs and potential economic benefits. The cost-benefit ratio is a key factor in deciding on their application and the selection of specific technologies and implementation strategies [5,17,18]. Using heavy industrial equipment as an example, one can see that the value of implementing new approaches is justified by reducing downtime and extending asset life. Looking at production as a whole, one can expect cost reductions through the use of digital services and enterprise-level ecosystems, ensuring control over personnel, products, and equipment while optimizing time and labor costs for non-core activities [19]. It should be clearly understood, however, that significant risks arise from the high level of initial investment and the lack of market expertise. The need to change business models for enterprises and companies, as well as safety issues, can also pose a barrier [20,21].

2. Materials and Methods

Approaching the idea of implementing a digital twin and conducting a literature review in this area, we find a significant increase in the quantity and quality of publications on practical examples [22–27]. It is also worth noting the many existing unresolved issues related to design, architecture, models, and scalability [26]. Issues related to inconsistent definitions, data quality, and data integration are highlighted. The lack of standard architectures for building industrial digital twins is particularly noteworthy [27]. In our opinion, the latter issue is currently the most pressing, as it determines the solution to such problems as integration with existing systems, solution scalability, and data transparency and integration.

A digital twin is a multi-agent distributed system whose primary purpose is to facilitate interaction between its components [28]. The system's component composition may include facility control subsystems; a software and analytical complex for data processing and visualization; and virtual and augmented reality (VR/AR) tools for the interactive presentation of its virtual copy. The basis of a digital twin is the object for which the twin is being created.

An object can be:

- a single physically existing object, such as a machine, device, or even a person;

 doi:10.20944/preprints202606.1372.v1

- multiple objects linked into a single production process;
- a manufactured product and its lifecycle from production to disposal;
- a process occurring in reality or modeled using complex and resource-intensive algorithms.

Therefore, although an object is the basis of a twin, its presence does not characterize a system as a digital twin. The key to the definition, then, is the presence of a direct and feedback link between an easy-to-manipulate digital model, considered a twin, and a difficult-to-manipulate object. It follows that a digital twin must ensure the direct transfer of information from the object to the twin and the reverse transfer from the twin to the object. Moreover, by "twin," we mean not only a model or an individual system, but an entire complex of systems.

Designing such a system requires careful consideration of a variety of factors and should draw on the experience of existing distributed systems and the approaches developed during their creation. Over the past two decades, various web services and cloud applications have evolved from small, local systems to complex, multi-agent distributed systems [29–31]. Such systems can be considered multi-agent in the broad sense, as their behavior is determined not by a single software module, but by an interacting set of them. All components perform their own functions, possess a certain degree of autonomy, and contribute to achieving a common outcome. The creation and operation of such systems, designed for multiple users and intensive data flows, requires adherence to a number of architectural principles.

The first principle is flexible system scalability at the level of allocated resources. In modern conditions, this problem is solved primarily through cloud computing, containerization, and orchestration, which allow for the allocation of resources to be increased or decreased at any time depending on the load. Load balancing systems also play an important role, allowing for the redistribution of traffic and the deployment of additional server capacity when load increases and its decommissioning when it decreases.

The second principle is the evolutionary nature of systems at the architectural level. The complexity of a system will inevitably increase as it is developed, implemented, and operated. At the design stage, it is impossible to accurately predict the final system complexity. Therefore, the key risk factor is the inability to integrate new user and business requirements into the system. To address this, the system is typically built with a long support cycle and the continuous refinement of its components in mind.

The third principle is technological independence at the system component level. A key factor is the size of the system and the development, implementation, and operation teams. In such circumstances, tying the entire system to a single technology stack is a constraint. Any part of the system can be implemented and deployed using various technologies available at the time of its launch, and subsequently rewritten for completely different conditions. The main thing is that this part remains within its functional responsibilities and complies with the agreed-upon contracts.

Completing the chain is the fourth principle of standardizing data exchange and intercomponent interaction. As the overall adaptability of a system increases due to the autonomy of its components and their technological independence, the need for stability and rigidity of its rules grows. These rules include data formats, transport protocols, API agreements, version compatibility, identification mechanisms, verification, and error handling. Unlike the internal implementation of individual components, these agreements must be carefully designed, have a transparent chain of changes, and explicit enforcement mechanisms, as the integrity of the system depends on them.

Therefore, if we are to use the generalized development experience of large companies to create digital twins, we must propose an architecture that meets the above principles. Currently, technology companies themselves are addressing this issue by building a composite architecture, one variant of which is microservices. A composite architecture is one in which a software system is constructed from a set of loosely coupled, independently developed, deployed, and scalable components [32–34]. These components interact only through clearly defined interfaces. By using the idea of breaking a system into loosely coupled parts and standardizing the mechanisms for interaction between the

 doi:10.20944/preprints202606.1372.v1

parts, they build pipelines for processing user requests from the starting point, which the user directly contacts, to the end point, which no one outside the system is aware of.

As an example, consider the user authorization request path using the correct login and password pair in a corporate portal built on a composite microservices architecture:

- The user enters data on the login form.
- The login form accesses the login form storage server.
- The form storage server forwards the request to the backend request orchestration server.
- The orchestration server determines the target request subsystem, in this case, the authorization system.
- The authorization subsystem queries the user storage subsystem to identify the user by login.
- The authorization subsystem validates the password and generates an authentication token, which the user will use for subsequent requests.
- The authorization subsystem queries the logging subsystem to record the user's login.
- The authorization subsystem queries the signal distribution subsystem to send a signal about the user's successful login.
- The signal distribution subsystem sends a corresponding signal to all listeners.
- The authorization subsystem provides the user with an authentication token.

Taking the process further, the next step is requesting user information, which first involves the logging and authorization subsystems. The former will store all user actions, while the latter will verify access rights. Only after accessing these will the user storage subsystem be accessed to retrieve their information, which can be distributed across several other subsystems. This level of separation of responsibilities creates the impression of redundancy; why not simply store everything in a single database and process it with a single, unified system? This is true for physically small systems that rarely change and don't handle a large number of requests. If a system doesn't meet even one of these criteria, it will be fragmented, and this is determined not by the wishes of individual developers, but by experience. If it is created as a single, monolithic program, divided into several internal modules but launched as a single process, then if one of the modules experiences excessive load, multiple copies of the entire program will have to be launched. With proper fragmentation into subsystems running in separate, isolated processes, any performance deficits in an individual module are compensated for by increased resources allocated to it.

The physical independence of the codebase is also worth highlighting. Any change to it doesn't require a full compilation of the entire program, nor does it require searching for dependencies of internal functions on other modules or, conversely, maintaining a dependency on the current module. If processes implemented by modules have common dependencies on utility code, such blocks are isolated either as libraries or as independent modules with their own processes. Parallel to physical independence comes technological independence; there's no need to implement everything on a single technology stack, and decisions can be made tailored to a specific process and task.

Modularization also results in a fragmentation of the system's cognitive complexity. Any initially simple system that exists for a sufficiently long time and changes during operation inevitably becomes complex enough to make it difficult to grasp with a unified understanding at all levels of detail. But in composite systems, the module developer doesn't need to understand how everything works; any dependencies are an external black box that receives and returns data.

- Technical and physical independence, coupled with the fragmentation of cognitive complexity discussed earlier, allows for the organization of software development on a production line with a deep division of labor. Responsibility and awareness, following complexity, are fragmented into layers:
- Module layer – at this level, the developer knows everything, or almost everything, about their module. They also have limited information about the interconnected modules necessary to solve the problem assigned to their module. They may also know only a portion of the shared or specialized libraries they use. The developer can make technical and technological decisions by aligning them with the architecture layer. This layer includes both user interface modules

 doi:10.20944/preprints202606.1372.v1

designed for various devices and server applications that store the process logic and process its data.

- Library layer – at this level, developers know primarily everything about their library, which they maintain. Even knowledge of where and how their library is used may be redundant, since decisions about changing it are not made at their level. This layer has the least information about the actual software being released by the team and is often redundant.
- Infrastructure Layer – At this layer, the physical characteristics of all used resources and the resource and technology requirements of individual modules are known. However, any details regarding the functions performed by modules are redundant. This layer collects resource utilization statistics and provides information to other layers about failures, problems, and excessive resource consumption, as well as solutions for these problems if they are purely infrastructure-related and not caused by internal subsystem issues.
- Integration and Orchestration Layer – At this layer, all interactions between microservices are known without details about their implementation and interaction goals. At this layer, transport mechanisms are organized, information delivery is controlled, and issues of traffic balancing and redistribution are determined. Like the library layer, it is redundant for small and sometimes even medium-sized systems, but will definitely appear for a large and complex system.
- Architecture Layer – This layer contains information about the conceptual design of processes, the fundamental structure of their data, the general hardware, and the overall integration and orchestration structure. Here, they define data exchange formats, the overall stack, the general implementation scheme for modules and their interactions, but essentially have no detailed knowledge of any of the layers. This layer receives information about the business requirements that the system must meet and produces technical requirements and a general implementation framework. It monitors the correct functionality of the resulting product, but lacks complete and comprehensive information about all the technical decisions made in the other layers.
- Data layer – this layer contains users, operators, and system and subsystem administrators. Depending on their level of authority, they will have varying levels of awareness of the business process, but will never know any technical implementation details, even if they have the right to directly interact with data in databases and file storage for in-depth analysis.
- Software product or business function layer – this is where information about the system as a commercial entity resides. This includes what the software product does, how it is financed, how many people work on it, and the cost of its support. This is where the future functions needed for the software product are determined, and this is where its future is determined.

As you can see, all these layers imply varying degrees of awareness of the software product's core business, the activities of the company developing it, its customers, and so on. This separation also provides advantages in terms of controlling the integrity of product information and its trade secrets. This is also a problem, as only a few people will truly have completed and utter knowledge of everything related to the product. This division of labor allows for the organization of the development and maintenance of a complex system, and this can be considered a consequence of composite architecture. Modular monolithic programs can be developed in a similar manner, and some companies employ this approach, but this only addresses the cognitive complexity issue, leaving all other challenges untouched.

Therefore, it is reasonable to assume that in the case of a digital twin, which is a heterogeneous and inherently complex system comprising many independent parts connected by information flows, a composite architecture, including microservices, is a reasonable choice.

3. Results

Based on the above, we propose a digital twin architecture, the fundamentals of its subsystem interactions, and the rules for its development.

Figure 1 shows a digital twin architecture, which we propose as an example of building a twin based on existing systems.

 doi:10.20944/preprints202606.1372.v1

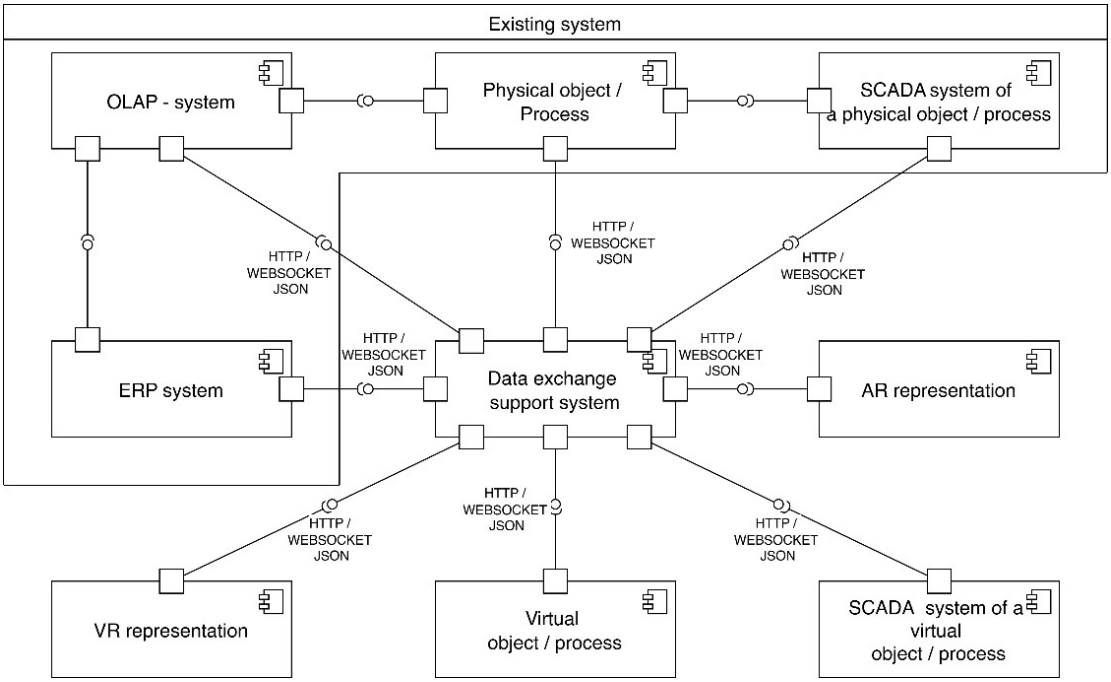

Figure 1. Schematic diagram of the digital twin architecture.

For this reason, we see a large circled block in the image, labeled "existing systems." These are subsystems typical for many production facilities, providing a general idea of the degree of integration of existing systems into digital twins. The proposed implementation of a unified data exchange system will allow for the abstraction of all disparate systems and the introduction of persistent contracts, supporting and guaranteeing their versioning, and ensuring the consistency of the exchange environment. In this context, such a system will act as a kind of API gateway. Its main difference from typical implementations of such solutions is that it does not directly communicate with other systems, but rather waits for them to supply information and caches it. As can be seen in the component diagram, all interactions with the data exchange system are conducted via the JSON protocol, while it allows any system to supply and consume data.

The proposed architecture and the outlined design principles were applied to the implementation of a digital twin of a production facility, presented in [35]. This paper examines the process flow diagram of a real chemical plant for the coproduction of methanol and ammonia. The main stages of the process include gas preparation (desulfurization, reforming, carbon dioxide removal), syngas compression, methanol synthesis, rectification and stabilization, ammonia synthesis, and finished product shipment. A dynamic model of this process flow diagram was created using Honeywell's UniSim Design software package; a portion of it is shown in Figure 2.

 doi:10.20944/preprints202606.1372.v1

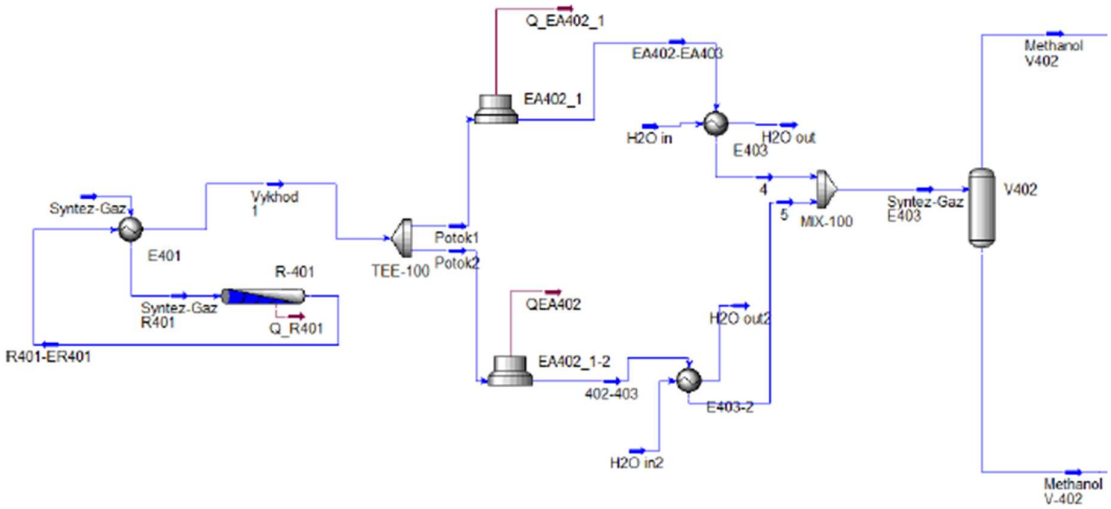

Figure 2. The first stage of methanol synthesis, modeled in UniSim Design.

The diagram (Figure 1) shows two SCADA systems for controlling a physical facility and for managing a model. The first system is deployed in a real-life production facility and was used as our primary example of visualizing operator control of production. The second system could be a complete copy of the first, but due to restrictions on access to commercial software, it is not. On the other hand, since a digital twin is not a simple simulator, it can and should provide more opportunities for exploring and manipulating the virtual model, so a custom implementation has several advantages. In our case, the two-dimensional production model is represented by a SCADA display in mnemonic diagram format, implemented as a web application. This application includes an editor for creating mnemonic diagrams and a tool for visualizing them. It receives current simulation parameters and allows them to be modified, controlling processes and interacting via a data exchange system. The diagram (Figure 1) also shows VR and AR representations that are fully part of the twin. They visualize the physical object or process and allow for interactive interaction with both the object and its virtual twin. The 3D representation is implemented using the Unity game engine, allowing it to be transformed into a full-fledged VR representation of the production facility, as shown in Figures 3 and 4.

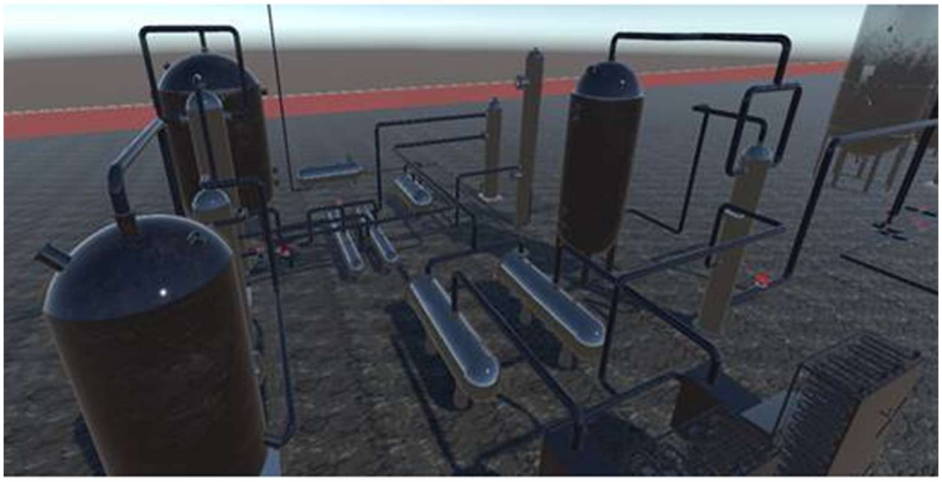

Figure 3. Virtual model in the Unity environment.

 doi:10.20944/preprints202606.1372.v1

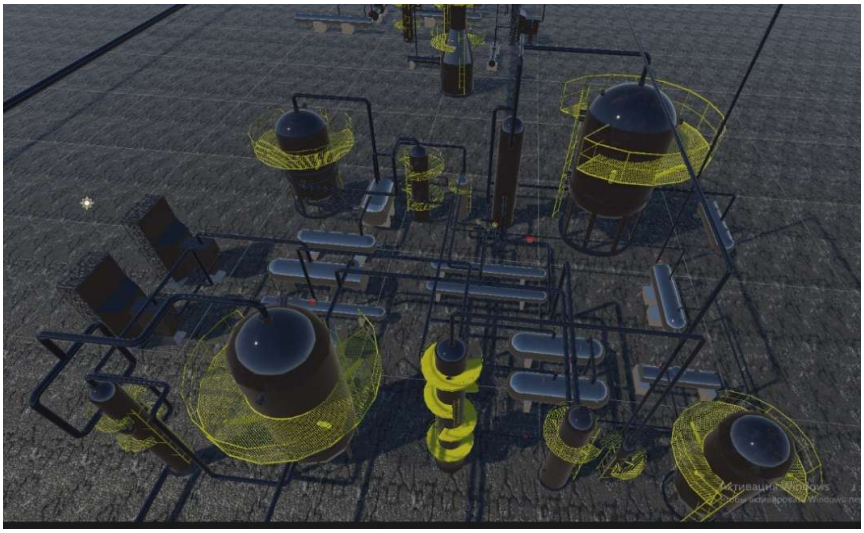

Figure 4. Simulated chemical-technological scheme.

Initially, all equipment was designed using CAD systems such as AutoCAD Plant 3D, 3ds Max, SolidWorks, or Blender. Next, a master plan for the placement of all objects was developed based on the process regulations and equipment specifications. The models were then imported into the Unity development environment in a unified format and positioned in the scene according to the master plan. Afterwards, materials, properties, and animations were assigned to the objects in C#, VR libraries (such as SteamVR) were connected, and the virtual environment became available for launch.

To better understand the technological processes in virtual reality, text fields were added to display reference information: the names of the equipment, their purpose, and key parameters such as temperature, pressure, and concentration.

An additional virtual model is an emergency simulation program, which was also implemented as part of the digital twin creation project. To handle emergency situations, a separate module described in [36] was integrated into the virtual solution. This module allows for visual representation of a pipe leak, a leak in a device, a stuck valve, and the entry of superheated refrigerant (water) into a shell-and-tube heat exchanger. A demonstration of an emergency situation associated with a leak in a device is shown in Figure 5.

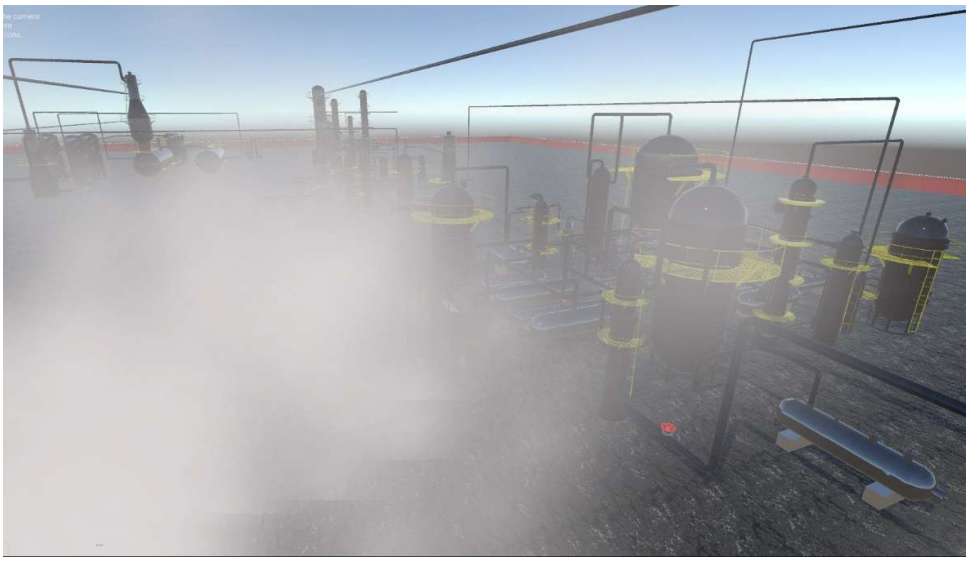

Figure 5. Emergency situation simulation in the Unity environment.

 doi:10.20944/preprints202606.1372.v1

This approach to creating a virtual 3D model proved ineffective due to its inherently slow implementation speed, which was associated with a high proportion of manual and semi-automated data processing. Therefore, the decision was made to develop a proprietary automated modeling system that would significantly accelerate the creation of similar 3D representations. To implement this system, software modules for constructing 2D chemical engineering plant diagrams [37] and automated 3D visualization of chemical engineering plant diagrams [38] were developed and subsequently used to refine the existing digital twin. These modules can be used to obtain both a finished 2D and 3D model for subsequent export to SCADA and VR environments. Modifications to the model can be easily implemented through automated model construction. The resulting model is shown in Figure 4.

It is important to note that all individual subsystems can and are as complex as the entire digital twin, implying that they themselves can implement any type of composite architecture. Therefore, we can say that a digital twin is a system built from other systems, and its architecture can be called macrosystemic. This approach might seem to preclude the advantages of the architecture described above, but in this case, the digital twin will act as a system of a higher order relative to its parts. In this context, each individual subsystem will be a microservice, with all the advantages of independence, decentralization, and scalability. Furthermore, each part will be loosely coupled and fits the description of domain-driven design concepts.

4. Discussion

When considering a microservice architecture as a basis for building a twin, it's worth specifically addressing its shortcomings and considering how to address them.

The first important drawback is the overall cognitive complexity of the architecture itself. Assuming the system is organized correctly, the challenges of its maintenance and development should not be neglected. Independent parts of the system, while providing a number of advantages, also impose a number of obligations that must be met. The main obligations include:

- Contract persistence, which states that once a participant has determined the format and content of the information provided, it must provide it in that format and content until the dependent party ceases using the contract. This is especially relevant in more typical systems, where an unaccounted missing value, a value of the wrong type, or an extra value can lead to the failure of a system unit. Contract persistence will require coordination of any data changes during development, allowing for early detection of points of failure.
- Persistence of the interaction environment implies a unified set of data exchange protocols, data encoding formats, and exchange rules that specify a single point of interaction concentration or, conversely, distributed and networked interaction. Crucially, where a single point of concentration exists, distributed interaction cannot exist, and vice versa. Violating this rule will lead to inconsistencies in the communication methods of system parts, which will lead to a gradual increase in chaos and will not allow for timely resolution of interaction problems.
- Permanence of responsibility ensures that, once a subsystem is identified as responsible for data, it does not lose this responsibility until all dependent systems cease using it. This obligation guarantees both the consumer and its provider that the provider is the sole owner of the data, responsible for and in control of it. This eliminates the possibility of diffuse responsibility and shared ownership of any data; the source of truth is always single, and others merely build on it.
- Consistent versioning requires separating all significant functional changes into versions, ensuring their parallel existence until the previous version is decommissioned by dependent systems. Versioning allows for permanent contracts to adapt system components to new needs and provides a flexible and rapid method for coordinated contract changes. A contract-dependent agent can always switch to a new version as it implements its functionality, and can also roll back to the previous version if unforeseen issues arise.

 doi:10.20944/preprints202606.1372.v1

- Consistent documentation completeness requires all system components, whether developed independently or related, to provide complete and up-to-date documentation on all possible interactions with them, their internal structure, and, in general, to cover the entire system with a set of artifacts describing its operation. This point may be perceived as insignificant or self-evident, but when developing complex, heterogeneous systems, documentation, its completeness, and its relevance are always at the greatest risk. If documentation is insufficient, the system will reach its expansion limits much earlier than it could have, thus documentation defines its growth limits.

These obligations are considered and managed at the system-wide level, since where variability and heterogeneity may be hidden in each individual component, it is imperative to build a homogeneous environment above these individual components.

Returning to the issue of the shortcomings of composite architectures, it's worth noting the lack of strict criteria for determining the appropriate degree of system partitioning. What constitutes a microservice and to what extent individual modules should be divided into submodules, separating them into services, is a question often avoided by proponents of this architecture. There are two main approaches. The first involves breaking the system down into subsystems as much as possible until a clearly indivisible unit emerges, i.e., a single logical entity within the system. This method of partitioning channels all interactions and minimizes the possibility of using optimizations associated with data aggregation at a single point, but offers immense flexibility in reusing subsystems. However, this approach is clearly incompatible with digital twins, as they contain a wealth of complex and existing software that cannot be broken down into components.

The second approach is to extract from the entire system so-called domains, or segments of business logic, essentially semantic or logically related units of the system. For example, in the case of a digital twin, the physical object itself is a logical unit, and we don't necessarily need to separate its parts from each other. However, in the case of a complex object, it can be broken down into segments that correspond to the logic of the process. Control and monitoring systems are also sufficient logical units and don't require further fragmentation. As can be seen from the examples given, the main factor in identifying a subsystem is its connectivity and participation in the main process. If a subsystem's internal connectivity is significantly higher than its external connectivity and it is also a participant in the main process around which everything is built, then this subsystem can be separated into a separate domain. Therefore, we can conclude that the second option is the most optimal for building a digital twin.

5. Conclusions

The concept of Industry 4.0 and its application to digital twins is one of the stages in the evolution of production processes, potentially enabling significant increases in the efficiency and flexibility of all processes, from raw material supply to product completion. New technologies offer numerous valuable tools for addressing existing complexities in production chains and paving the way for their optimization, but they also create new challenges. Among the key challenges that will be relevant in the near future are cybersecurity and skills shortages. Furthermore, significant attention in current research is paid to the standardization and scalability of solutions. The foundation for addressing these challenges lies in the principles of digital twin architecture. One promising approach, proven effective in areas unrelated to digital twins, is a composite architecture based on microservices. This architecture enables the efficient management of complex systems while enabling their continuous evolution. It's important that the development and implementation of digital twin systems be accompanied by clear interaction standards, consistent contracts, and up-to-date documentation. This will ultimately ensure the desired flexibility, performance, and scalability. Therefore, further research and development of effective digital twin architectures will contribute to the successful implementation of Industry 4.0 across various industries.

 doi:10.20944/preprints202606.1372.v1

Author Contributions: Conceptualization, Eleonora Koltsova; Methodology, Maksim Pysin, Alexey Lobanov and Anatoly Antipov; Software, Alexey Lobanov, Alexey Arkhipov and Roman Krasheninnikov; Validation, Roman Krasheninnikov; Writing – original draft, Eleonora Koltsova, Maksim Pysin, Alexey Lobanov and Anatoly Antipov; Writing – review & editing, Maksim Pysin, Alexey Lobanov and Anton Perekatov; Visualization, Alexey Lobanov, Alexey Arkhipov and Anton Perekatov. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ledford, A.; Hyre, G.; Harris, G.; Purdy, G.; Hedberg, T. Origin of the Fourth Industrial Revolution: Manufacturing Predictions Preceding Industrie 4.0. J. Sci. Technol. Policy Manag. 2024. https://doi.org/10.1108/jstpm-03-2023-0040.
2. Hamad, B. M.; Jawad, M. K. The Fourth Industrial Revolution: A Historical and Conceptual Review. J. Econ. Adm. Sci. 2024, 30, 154–172. https://doi.org/10.33095/gh3a7g38.
3. Nosalska, K.; Mazurek, G. Marketing Principles for Industry 4.0 — A Conceptual Framework. Eur. Manag. J. 2019. https://doi.org/10.2478/emj-2019-0016.
4. L. Pullagura et al., Industry 4.0 Design Principles, Technologies, and Applications, Auerbach Publications, 2024. https://doi.org/10.1201/9781003581963-11.
5. Ćwiklicki, M.; Wojnarowska, M. Circular Economy and Industry 4.0: One-Way or Two-Way Relationships? Eng. Econ. 2020. https://doi.org/10.5755/j01.ee.31.4.24565.
6. Mosleuzzaman, M.; Arif, I.; Siddiki, A. Design and Development of a Smart Factory Using Industry 4.0 Technologies. AJBAIS 2024.
7. Loseto, G.; et al. Osmotic Cloud-Edge Intelligence for IoT-Based Cyber-Physical Systems. Sensors 2022, 22, 2166. https://doi.org/10.3390/s22062166.
8. Rathore. Next Generation Intelligent IoT Use Case in Smart Manufacturing. In Proceedings of the Name of the Conference; 2023. https://doi.org/10.1007/978-981-99-6553-3_21.
9. Kaur, N. Intelligent Manufacturing in Industry 4.0, Informa, 2025. https://doi.org/10.1201/9781032655758-2.
10. Pasupuleti, M. K. Smart Industry 4.0: Transformative Innovations and Advanced Technologies. 2024. https://doi.org/10.62311/nesx/77691.
11. Tao, F.; Zhang, H.; Liu, A.; Nee, A. Y. C. Digital Twin in Industry: State-of-the-Art. IEEE Trans. Ind. Inform. 2019. https://doi.org/10.1109/tii.2018.2873186.
12. Huang, Z.; Shen, Y.; Li, J.; Fey, M.; Brecher, C. A Survey on AI-Driven Digital Twins in Industry 4.0: Smart Manufacturing and Advanced Robotics. Sensors 2021. https://doi.org/10.3390/s21196340.
13. Zhang, Z. Digital Twin: Application and Prospect. 2023. https://doi.org/10.54254/2753-8818/12/20230477.
14. Caiza, G.; Sanz, R. Digital Twin to Control and Monitor an Industrial Cyber-Physical Environment Supported by Augmented Reality. Appl. Sci. 2023, 13, 7503. https://doi.org/10.3390/app13137503.
15. Wang, K.; Wang, Y.; Li, Y.; Fan, X.-P.; Xiao, S.; Hu, L. A Review of the Technology Standards for Enabling Digital Twin. Digit. Twin 2022. https://doi.org/10.12688/digitaltwin.17549.1.
16. Aziz, S.; Jung, D. W.; Zaman, U. K. uz; Bin Aqeel, A. Digital Twins in Smart Manufacturing. In Advances in Manufacturing; Publisher, 2023. https://doi.org/10.1201/9781003327523-6.
17. Abdullah, F. M.; Al-Ahmari, A. M.; Anwar, S. A Hybrid Fuzzy Multi-Criteria Decision-Making Model for Evaluating the Influence of Industry 4.0 Technologies on Manufacturing Strategies. Machines 2023. https://doi.org/10.3390/machines11020310.
18. Golovina, T. A.; et al. Digital Twins as a New Paradigm of an Industrial Enterprise. Int. J. Technol. 2020. https://doi.org/10.14716/ijtech.v11i6.4427.

 doi:10.20944/preprints202606.1372.v1

19. Tostes, A. D.; Azevedo, A. A Value-Oriented Framework for Return Evaluation of Industry 4.0 Projects, 2023. https://doi.org/10.1007/978-3-031-38241-3_95.
20. Baxter, J. Maintenance Performance in the Age of Industry 4.0: A Bibliometric Performance Analysis and a Systematic Literature Review. Sensors 2023. https://doi.org/10.3390/s23031409.
21. Grabowska, S.; Gajdzik, B.; Saniuk, S. The Role and Impact of Industry 4.0 on Business Models. In Book Title; 2020. https://doi.org/10.1007/978-3-030-33369-0_3.
22. Kerrouchi, S.; Aghezzaf, E.; Cottyn, J. Production Digital Twin: A Systematic Literature Review of Challenges. Int. J. Comput. Integr. Manuf. 2024. https://doi.org/10.1080/0951192x.2024.2314792.
23. Kadam, A. A.; et al. A Comprehensive Review on Digital Twin Integration in Smart Manufacturing Technologies, Challenges, and Future Trends. JAICC 2025. https://doi.org/10.47363/jaicc/2025(4)470.
24. Yu, J.; Zhou, J.; Fu, J. From Digital Twin to Digital Twin Agent. In Proceedings; 2025. https://doi.org/10.1109/e-cargo65996.2025.11139170.
25. Arin, I. A.; Murad, D. F.; Hendric, H. L.; Warnars, S. A Systematic Literature Review of Recent Trends and Challenges in Digital Twin Implementation. In Proceedings; 2023. https://doi.org/10.1109/iciss59129.2023.10291219.
26. Younes, F.; Lahsen-Cherif, I.; El Ghazi, H. Toward a City Digital Twin: Design Principles, and Challenges. In Proceedings; 2024. https://doi.org/10.1109/commnet63022.2024.10793378.
27. Zaborowski, P.; et al. The Role of Standards in The Environmental Digital Twins Architectures. In Proceedings; 2024. https://doi.org/10.1109/igarss53475.2024.10640598.
28. Kalyani, Y.; Collier, R. The Role of Multi-Agents in Digital Twin Implementation: Short Survey. ACM Comput. Surv. 2024. https://doi.org/10.1145/3697350.
29. Dähling S., Razik L., Monti A. Enabling scalable and fault-tolerant multi-agent systems by utilizing cloud-native computing // Autonomous Agents and Multi-Agent Systems. — 2021. — Vol. 35, No. 1. — Art. 10. — DOI: 10.1007/s10458-020-09489-0.
30. Qusef A., Alshraideh M., Otoom A., AbuZaiter R. Design of Modern Distributed Systems based on Microservices Architecture // International Journal of Advanced Computer Science and Applications. — 2021. — Vol. 12, No. 2. — P. 175–185. — DOI: 10.14569/IJACSA.2021.0120220.
31. Salah T., Zemerly M. J., Yeun C. Y., Al-Qutayri M., Al-Hammadi Y. The Evolution of Distributed Systems Towards Microservices Architecture // Proceedings of the International Conference for Internet Technology and Secured Transactions. — 2016. — P. 318–325. — DOI: 10.1109/ICITST.2016.7856721.
32. Mallick, S.; Bhose, R.; Malaiyandisamy, G. Composite Applications Using Service Component Architecture Model and Open Virtualization Format. U.S. Patent US20120260228A1, 2011. https://patents.google.com/patent/US20120260228A1/en.
33. Stiehl, V. Process-Driven Composite Application Architecture. U.S. Patent US20130159062A1, 2011. https://patents.google.com/patent/US20130159062A1/en.
34. Philip, M. M.; Natarajan, K.; Ramanathan, A.; Balakrishnan, V. Composite Pattern to Handle Variation Points in Software Architectural Design of Evolving Application Systems. IET Softw. 2020, 14, 98–105. https://doi.org/10.1049/IET-SEN.2019.0006.
35. Lobanov, A.V.; Sidorenko, N.V.; Filippova, E.B. Razrabotka modeli tekhnologicheskogo proizvodstva metanola i ammiaka v virtual'noy srede. Uspekhi v khimii i khimicheskoy tekhnologii 2022, 36 (11), 70–73. (In Russian)
36. Lobanov, A.V.; Pysin, M.D. Otobrazheniya dannykh i protsessov v virtual'nom okruzhenie tekhnologicheskogo proizvodstva tsifrovogo zavoda. In Innovatsionnoe razvitie sovremennoy nauki: novye podkhody i aktual'nye issledovaniya; Sbornik materialov Mezhdunarodnoy nauchno-prakticheskoy konferentsii, Moscow, Russia, 30 November 2023; Alef: Moscow, Russia, 2023; pp. 158–164. https://doi.org/10.26118/1468.2023.32.42.010.
37. Arkhipov, A.M.; Lobanov, A.V.; Sidorenko, N.V. Razrabotka programmnogo modulya dlya konstruirovaniya skhem khimiko-tekhnologicheskikh proizvodstv v 2D predstavlenii. Uspekhi v khimii i khimicheskoy tekhnologii 2024, 38 (1), 69–71. (In Russian).

 doi:10.20944/preprints202606.1372.v1

38. Sidorenko, N.V.; Lobanov, A.V.; Arkhipov, A.M. Razrabotka programmnogo modulya dlya avtomatizirovannoy vizualizatsii skhem khimiko-tekhnologicheskikh proizvodstv v 3D predstavlenie. Uspekhi v khimii i khimicheskoy tekhnologii 2024, 38 (9), 80–82. (In Russian).