The object of the study is the architectural design process of a gas transportation automation system at the compressor station level as a multi-aspect heterogeneous computer-software system in the automation domain. This system integrates subsystems of different fundamental architectural natures – ranging from PLC programs to cloud-based analytical systems and information systems with operational logic.

The study is focused on solving the problem of completeness, relevance, and interoperability of architectural design artifacts for complex computer-software automation systems. Existing methods predominantly focus on individual aspects – design patterns, system integration and layering, optimal control, or the implementation of specific capabilities, such as supervisory control. They do not form a single, end-to-end architectural design process for the system as a whole.

The essence of the obtained results lies in the formation of an architectural model of the system through the execution of the architectural design process using a system-type-centric method. The constructed architectural model is distinguished by the traceability of architectural artifacts and the selection of architectural viewpoints depending on the fundamental nature of the systems, which ensures the coverage of fundamental architectural aspects and achieves the completeness of the architectural description. It is built through the sequential formation of system views from relevant viewpoints.

The results demonstrate the scientific and practical feasibility of the method for the comprehensive design of complex heterogeneous automation systems that combine operational technology (OT) and information technology (IT) levels. From a practical standpoint, the obtained results can be applied as a reference architectural model for the implementation of the method

Keywords: architectural design, automation, gas transportation, system-type-centric paradigm

UDC 004.414.2:681.51
DOI: 10.15587/1729-4061.2026.364578

ARCHITECTURAL DESIGN OF GAS TRANSPORTATION AUTOMATION SYSTEM BASED ON THE SYSTEM-TYPE-CENTRIC METHOD

Ihor Polataiko
PhD Student*
ORCID: https://orcid.org/0000-0002-9111-0162
Leonid Zamikhovskyi
Corresponding author
Doctor of Technical Sciences, Professor, Head of Department*
E-mail: leozam@ukr.net
ORCID: https://orcid.org/0000-0002-6374-8580
*Department of Information and Telecommunication Technology and Systems
Ivano-Frankivsk National Technical University of Oil and Gas
Karpatska str., 15, Ivano-Frankivsk, Ukraine, 76019

Received 20.03.2026
Received in revised form 01.06.2026
Accepted date 10.06.2026
Published date 30.06.2026

How to Cite: *Polataiko, I., Zamikhovskyi, L. (2026). Architectural design of gas transportation automation system based on the system-type-centric method. Eastern-European Journal of Enterprise Technologies, 3 (2 (141)), 111–138. https://doi.org/10.15587/1729-4061.2026.364578*

1. Introduction

Automation of the gas transportation process is one of the key areas of applied automation in the oil and gas industry. Modern compressor station (CS) automation systems are complex heterogeneous computer-software systems (CSS). They combine various levels of automation according to the Purdue pyramid [1] – from the level of field devices and control to the levels of supervisory control, planning, and enterprise management. Such complexity is driven both by the technological specifics of the gas transportation process and by the needs of modern business and production in the integration of the classic control systems with analytical and information systems within the framework of Industry 4.0 and Industry 5.0 concepts.

The technological process of gas transportation is carried out at the CS using gas pumping units (GPU), which ensure gas compression to the required parameters. Architecturally, such systems combine subsystems of a fundamentally different natures:

– GPU control systems based on programmable logic controllers (PLC);

– SCADA systems for supervisory control at different levels of the hierarchy;

– analytical data processing systems;

– web-oriented systems for the automation of business processes and data management;

– and others.

As a consequence of such heterogeneity and of the convergence of classical control systems (OT) with information and analytical systems (IT), subsystems that have a fundamentally different architectural nature are, in practice, designed according to different, mutually inconsistent engineering practices. This leads to the architectural artifacts of individual subsystems turning out to be disjointed and mutually incompatible, complicates integration at the IT/OT boundary, increases the risk of integration defects, and worsens the maintainability of the system as a whole. Therefore, there is a pressing need for a unified approach to architectural design that would make it possible to coherently encompass subsystems of different types within a single system.

Study of processes and models of architectural design of computer-software systems (CSS) is fundamentally important for forming standardized approaches. This becomes crit-

ically significant in the era of rapid development of AI-based development tools, since it formalizes architectural processes and models with interoperable artifacts that ensure the efficiency and direction of such intelligent systems' work.

In studies [2,3], the authors proposed a system-type-centric paradigm and method for the architectural design of automation CSS. Such an approach allows differentiating and structuring architectural design depending on the fundamental nature of each subsystem. On the one hand, this ensures the formation of relevant, traceable, and interoperable architectural design artifacts within the architectural model. On the other hand, it regulates a prescriptive architectural design process that adapts to the fundamentally different aspects of the architectural nature of various types of CSS in automation. However, in order to confirm the scientific and practical value of such a method, it is necessary to verify the expediency of its application for solving real engineering problems and to analyze how the resulting architectural models differ. Accordingly, study into the applied aspects of the method's application is fundamentally important as a study direction aimed at the establishment of structured architectural design approaches, which can constitute the basis both for more efficient design in general and serve as a foundation for AI-assisted systems for automating the design of CSS architecture.

The gas transportation automation system at the CS level is a representative object for such a study, since by its nature it requires the integration of various types of CSS [2]. It allows verifying whether the method is capable of ensuring the creation of a holistic, traceable, and interoperable architectural documentation package for a complex heterogeneous object. The relevance of applying the method specifically to the design of an automation system in the oil and gas industry lies in the need for a structured and interoperable architectural approach that ensures the efficiency and directedness of design and the interoperability of the resulting artifacts.

2. Literature review and problem statement

The issues of automating the gas transportation process and GPU control are considered in a number of works. Work [4] describes an optimal control system for the natural gas compression process at a CS equipped with a gas-turbine-driven GPU, which is integrated into the upper (supervisory) level as a component of the GPU control system. The proposed approach uses the apparatus of fuzzy algebra to account for uncertainties in the measurement of technological parameters and is based on a hierarchical structure of the control system. However, the work focuses precisely on control optimization algorithms, while the aspects of architectural design of a holistic CSS covering various automation levels are not considered.

Work [5] details the capabilities of SCADA systems for controlling pumping and compressor stations in the oil and gas industry, including SCADA architecture levels, principles of their interaction with sensors, as well as functions of technical diagnostics and predictive maintenance. The work focuses on the role of SCADA systems in ensuring industrial safety, energy efficiency, and reliability; however, SCADA is considered in isolation, without taking it into account as a subsystem within a broader heterogeneous automation system.

Work [6] is devoted to the problems of expanding the functionality of topologies of web-oriented process control systems based on "Open User Communication". Work [7] describes a GPU anti-surge protection system based on hardware-software means of vibration monitoring. Study [8] focuses on building a simulation model of a web-oriented system for frequency control of a servo drive based on the "Digital Twins" technology. Work [9] presents the results of developing a simulation model of a PID controller based on the Simatic S7 hardware-software means and the "Digital Twin" technology. The listed works contain important results in their subject areas; however, they focus on specific technological and technical aspects, without considering the issue of comprehensive architectural design of an automation system as a heterogeneous CSS.

In the field of architectural design of general-purpose automation CSS, industry reference models are widespread. The Purdue model [1] describes the hierarchical structure of industrial control systems, dividing them into levels from field devices to enterprise business planning. Its formalization is the ISA-95 standard [10], which defines the models and interfaces for integration between production control systems and corporate information systems. The Industrial Internet Reference Architecture IIRA [11] offers a general architectural foundation for IIoT systems with four viewpoints – the business, usage, functional, and implementation viewpoints. The Reference Architecture Model for Industry 4.0, RAMI [12–14], presents a three-dimensional representation of digitalized production that combines the life cycle, the hierarchical levels, and the architecture layers. Such models describe the structure and hierarchy of systems but do not define an architectural design process that takes into account the fundamental differences in the nature of systems of different types.

Among the widespread structural templates for documenting architecture are the arc42 template [15] and the C4 model [16], which provide structures for communicating architectural decisions to stakeholders. These tools, however, are system-agnostic and do not contain a prescriptive design process oriented toward the nature of the system. The classic "4+1" model [17] describes a specific set of architectural viewpoints, but remains too general – it does not take into account the fundamental differences between types of CSS.

In the context of software architecture design methodologies, several approaches are dominant. Views and Beyond [18] describes architecture through a set of coordinated views that correspond to the interests of various stakeholders, and offers a catalog of styles for documenting them. Attribute-Driven Design (ADD) [19] is an iterative design method in which architectural decisions are made on the basis of the system's quality attributes and business goals. The Viewpoints and Perspectives approach [20] extends the concept of viewpoints with cross-cutting perspectives – such as security, performance, or availability – that are applied to all views. These methodologies provide an important conceptual apparatus for multi-perspective design, but remain predominantly general. These methodologies do not offer concrete recommendations for adaptation to systems of fundamentally different architectural nature, such as heterogeneous CSS within automation systems that combine OT- and IT-level subsystems [21–23].

In the field of systems engineering, the Functional Architecture for Systems (FAS) approach [24] has been proposed for designing the functional architecture of a system independently of implementation details. Meanwhile, the System Architecture Framework (SAF) [25] is focused on model-

ing the architecture of technical systems in the context of Model-Based Systems Engineering (MBSE). Domain-Driven Design (DDD) [26] is widely used in IT systems for analyzing the subject domain and designing the system. However, these approaches do not encompass the fundamental differences in the architectural nature of different types of systems in heterogeneous automation environments.

In studies [2, 3], the authors proposed a system-type-centric paradigm (STCP) and a system-type-centric method (STCM) for the architectural design of automation CSS. The distinctive feature of these studies is that they take into account the fundamental differences in the architectural nature of systems of different types and ensure the formation of interoperable architectural design artifacts.

However, the issue of applying the STCM of architectural design to specific applied problems in the field of gas transportation automation remains not fully elaborated. The mentioned scientific works present the theoretical foundations of the method and its quantitative assessment, yet a detailed, elaborated example of the end-to-end application of the method to the design of a real heterogeneous automation system is absent.

The conducted analysis of literature sources reveals a substantial methodological gap between narrowly specialized engineering solutions and the needs for comprehensive design of heterogeneous systems. Existing applied studies in the field of gas transportation automation [4–9] are focused predominantly on isolated technological aspects (in particular, on control algorithms and principles or on dispatching aspects), which forms only a fragmentary, mono-aspect view of the automation system as an object of architectural design. At the same time, widespread industry standards and universal architectural design methodologies [1, 10–12, 15–20, 24–26] are for the most part system-agnostic: they operate with generalized categories that do not allow the fundamental differences in the architectural nature of various types of CSS within automation systems – be it industrial automation or robotics – to be taken into account.

The absence of works that would bridge this gap points to a pressing scientific and applied problem – the lack of formalized approaches for creating a holistic, multi-aspect architectural design of complex automation CSS.

The integration of the STCM into the practice of designing automation systems conceptually solves this problem at the theoretical level [2, 3]. However, the question of the practical applicability of the method to the design of a real heterogeneous automation CSS remains open. This gives grounds to assert that it is expedient to conduct a study devoted to applying the STCM of architectural design of automation CSS to the design of a gas transportation automation system at the CS level. It will allow to verify the practical applicability and expediency of the method by forming a complete architectural model of such a system in accordance with the proposed architectural documentation model [3].

3. The aim and objectives of the study

The aim of the study is the formation of a reference architectural model of a gas transportation automation system by applying the STCM of architectural design, and the verification of the method's applicability for designing complex heterogeneous systems. The scientific component of the aim is to expand the theoretical foundations of the STCP through analyzing the feasibility of applying this method and proving its ability to produce cohesive, traceable, and consistent architectural documentation for systems combining different architectural natures (from the PLC level to cloud platforms). The practical feasibility of the study lies in creating a benchmark (reference) architectural model of a CS automation system.

To achieve this aim, the following objectives have been set:

– to perform architectural design at the system-scale level through the formation of problem space specifications, system capabilities representations, and its decomposition into subsystems with classification by system types;

– to define a set of architectural viewpoints for each subsystem according to its architectural type based on the applicability matrix defined within the STCM of architectural design;

– to build architectural representations at the subsystem and component levels based on the defined viewpoints, thereby forming a holistic architectural model of the system;

– to evaluate the applicability of the STCM to the design of complex heterogeneous automation systems based on an analysis of the completeness and effectiveness of coverage of the system's architectural aspects in the designed architectural model.

4. Materials and methods

The object of the study is the architectural design process of a gas transportation automation system at the CS level as a multi-aspect heterogeneous CSS in the automation domain.

The main hypothesis of the study is that the STCM of architectural design [3] ensures practical applicability to the design of real complex heterogeneous automation systems. As well as the fact that applying the method allows forming a complete, interoperable, and traceable architectural documentation for complex heterogeneous CSS in the automation domain [2].

Accepted assumptions in the work: it is assumed that the problem space of the automation system – functional requirements, non-functional requirements, and constraints – is sufficiently formalized and stable. Aspects related to the definition of business goals of the system, domain exploration, and planning the implementation of architectural solutions are outside the scope of this study. The study also uses a simplified model of the gas transportation technological process to describe the control level, which does not cover all the details of the real technological process but is sufficient to demonstrate the applicability of the architectural design method.

The object of design is a typical gas transportation automation system at the CS level, which includes several compressor shops (CSh) with GPUs. According to the adopted system model, each GPU is equipped with a local control system based on a Siemens S7-1200 PLC (Siemens AG, Germany), and the CSh – with a local SCADA system based on WinCC Unified (Siemens AG, Germany). At the CS level, a centralized SCADA system is deployed. The cloud part of the system is deployed in the AWS Cloud infrastructure (Amazon Web Services, Inc., USA) and includes a subsystem for analytical data processing, equipment condition and maintenance registry, as well as a system for visualizing historical data based on Grafana (Grafana Labs, USA).

The study was conducted utilizing qualitative, constructive theory-building methods. Since the study focuses on

applying the developed architectural design method to a concrete applied task, laboratory equipment and hardware-software simulation tools were not used in the study. The results were obtained through the systematic application of the STCM of architectural design [3] to the designed model of the gas transportation automation system.

Architectural design within the scope of the study is carried out in accordance with the architectural documentation model and architectural design process defined in [3]. The design progresses hierarchically – from the system-scale level to the subsystem-scale level, and then – to the component-scale level, with a deepening to the level of detail for those aspects that require it. At each scale level, architectural viewpoints relevant to the corresponding type of subsystem or component are applied in accordance with the viewpoint applicability matrix described in [3].

The draw.io graphical tool was used to build architectural diagrams in the study. The AI agent tool Claude Design (Anthropic PBC, USA) was used to construct UI designs of the custom web application for the equipment condition and maintenance registry based on system requirements and the system capabilities representation. Standard notations of block diagrams of automatic control systems were used to construct architectural representations of control models.

5. Results of designing the automation system

5. 1. Design at the system-scale level

At the first stage, system requirements were formed and representations from the I-1 and I-2 viewpoints were built. These representations serve as input data for the entire architectural design process.

Table 1 presents the functional requirements viewpoint for the system.

Table 1

I-1: Functional requirements viewpoint

Requirement ID	User Story
FR-UC-01	As an automation engineer, I want to perform supervisory control at the GPU level in the SCADA system to manage an individual unit and monitor its current operating parameters
FR-UC-02	As an automation engineer, I want to perform supervisory control at the CS level in the SCADA system to see the overall picture of the technological process at the CS and coordinate the operation of all GPUs
FR-UC-03	As an analyst, I want to view charts of historical GPU data to analyze past equipment behavior and identify long-term trends
FR-UC-04	As an auditor/analyst, I want to review equipment condition assessment data to understand the current level of wear and efficiency of the GPUs
FR-UC-05	As an engineer/auditor, I want to manage equipment records (adding, editing passport data) to maintain an up-to-date digital registry of all station assets
...	

Table 2 presents the system's non-functional requirements (NFR) and constraints viewpoint.

Table 2

I-2: NFR and constraints viewpoint

Category	Requirement description
Security: Authorization (RBAC)	The level of functionality access to web applications must be bounded according to user roles: Automation Engineer, Auditor, Analyst, Administrator
Security: Authentication	Access to web applications is conducted with personal user accounts
SCADA authentication	Access control to SCADA is carried out using standard tools of the SCADA system
Security: Network communication	Telemetry transmission from the GPU to the cloud gateway via MQTT protocol must be carried out exclusively over an encrypted channel using certificates (mTLS). Interaction with the web application must occur over encrypted channels using TLS (HTTPS, WSS)
Security: Data at Rest	GPU data in the cloud must be stored in an encrypted format
Reliability: Uptime (Control) The capabilities «Supervisory control at the GPU level» and «Supervisory control at the CS level» (local SCADA) must have an availability level of no less than 99.99%	
Reliability: Uptime (Analytics)	Cloud capabilities (charts, registry, state assessment) can have a standard availability level of 99.9%. If the cloud becomes unavailable, local GPU control must not be interrupted
Performance: Near-real-time Analytics	Latency between data transmission via MQTT and the calculation of "Real-time GPU condition assessment" must not exceed 2 minutes
Data Retention (Preservation)	The «Historical GPU data storage» component must retain highly accurate raw telemetry for 3 months, and aggregated data for «Daily state assessment» for at least 5 years
Scalability	The cloud system must support a simultaneous connection to up to 1000 GPUs
Data locality	The data of the GPA operation must be stored and processed by cloud services (cloud computing)
Usability: Audit Trail	For the UCs «Equipment record management» and «Maintenance record management», the system must automatically maintain an immutable audit log (including users and their modifications)

At the second stage, based on I-1 and I-2, the SA-1 (system capabilities representation) was built (Table 3).

At the next stage, according to the method, the representation from SC-1 viewpoint was built, where subsystems are defined based on the capabilities and system types identified within the system-type-centric paradigm [2].

Within the SC-1 representation, the following subsystems were defined (Fig. 1):

1. "GPU control system" (Process Control System): Custom system / Continuous control system.

2. "SCADA system for GPU-level supervisory control": Off-the-shelf self-hosted.

3. "SCADA system for CS-level supervisory control": Off-the-shelf self-hosted.

4. "Analytical data processing system": Custom system / Data-flow-oriented system.

5. "Equipment condition and maintenance registry": Custom system / Behavior-oriented system.

6. "Historical data visualization system": Off-the-shelf self-hosted.

The SC-1 design step is crucial within the system-type-centric paradigm, as this is exactly where the decomposition into subsystems and the determination of subsystem types

occur [2]. Defining the type for each subsystem directly affects the subsequent design process and determines which artifacts will be built at the next stages.

Table 3

SA-1: Capabilities viewpoint

Capabilities group	Capability
Technical Process Control	Real-time GPU data processing and GPU control
Supervisory control	Supervisory control at the GPU level
Supervisory control	Supervisory control at the CS level
Data transmission and storage	Receiving data from the GPU and transmitting data for further processing (cloud gateway)
Data transmission and storage	Storage of historical GPU data
Equipment condition assessment	Daily GPU condition assessment (based on historical efficiency data
Equipment condition assessment	Real-time GPU condition assessment
Accounting	Accounting and planning of equipment maintenance
Accounting	Maintaining equipment records
Data visualization	Displaying charts of historical GPU data
Data visualization	Displaying GPU condition assessments
Operational functions	User profile management.
Email delivery	Delivery of Email messages

5. 2. Applicability of architectural viewpoints

For the "GPU control system" subsystem based on PLC (Continuous Control), the following set of representations has been formed:

– Control viewpoint (SSA-1) – in control block diagram format with mathematical models of control loops for GPU technological parameters;

– XA-1 – is not applied – the subsystem has no direct user interaction; the UX flow of the automation engineer when working with the GPU is provided via the GPU-level SCADA subsystem;

– High-level components viewpoint (SSA-3) and its extension with component classification (SSA-3-ext) – for the logical decomposition of the subsystem and designating the PLC program as a custom application-level component;

– Software components viewpoint (SSC-1) – representation of application and infrastructure level software components – in this case, the PLC program as an application-level software component along with the firmware (execution environment) of the Siemens S7-1200 PLC as a platform level;

– Infrastructure viewpoint (XC-2) – to describe the physical Siemens S7-1200 controller, GPU sensors and actuators (via I/O), the network interface for communication with SCADA, and the internet gateway for transmitting events to the cloud;

– Deployment viewpoint (XC-3) – to describe the deployment and update strategy of the PLC program via an engineering station based on Siemens TIA Portal (Siemens AG, Germany) as part of the supportive infrastructure.

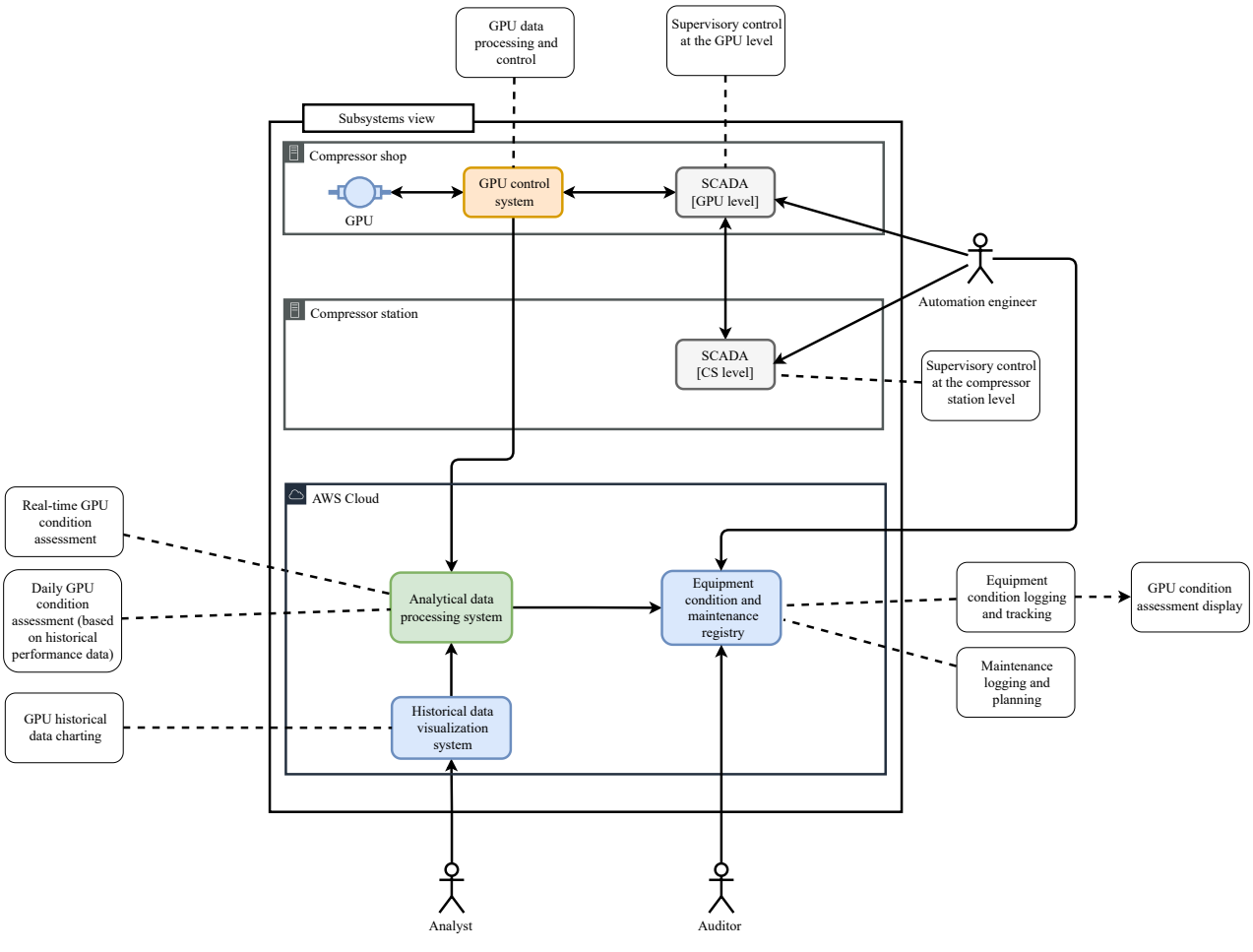

Fig. 1. System decomposition into subsystems (SC-1)

For the GPU-level supervisory control SCADA subsystem (off-the-shelf self-hosted, platform-like), full internal architectural design is not performed. According to the architectural method, a limited set of representations is applied for off-the-shelf self-hosted subsystems:

– XA-1 – for describing the UX flows of the automation engineer during supervisory control at the individual GPU level;

– SSC-1 – for describing exactly which software components form the subsystem (SCADA WinCC Server [WinCC Unified] together with MS SQL as a proprietary DBMS for storing tags, SCADA history, and events);

– XC-2 – for describing the hardware-platform environment of the SCADA server at the individual unit level – CS as an on-premises infrastructure;

– XC-3 – for describing the deployment and update strategy of SCADA servers and project-specific configuration via the Siemens TIA Portal engineering station.

For the CS-level supervisory control SCADA subsystem (off-the-shelf self-hosted, platform-like), a similar set of representations is applied with an emphasis on the station scale:

– XA-1 – for describing the UX flows of the automation engineer during supervisory control at the entire CS level, including the coordination of GPU operation;

– SSC-1 – for displaying the software components that form the subsystem (SCADA WinCC Server [WinCC Unified] and MS SQL as a proprietary DBMS);

– XC-2 – for describing the hardware-platform environment of the station-level SCADA server;

– XC-3 – for describing the deployment and update strategy of station-level SCADA servers via the Siemens TIA Portal engineering station.

For the "Analytical data processing system" subsystem (Data-flow-oriented, cloud-based):

– XA-1 – is not applied – the subsystem has no direct user interaction;

– SSA-3 and SSA-3-ext in data flow diagram format – for the logical representation of the telemetry processing flow: from the point of receiving MQTT messages from the PLC via the CS gateway to AWS IoT Core, through the input data reception component, then – in parallel – to the raw data storage component (Kinesis Data Firehose to S3) and the real-time GPU condition assessment component; with subsequent data aggregation and regular (daily) GPU condition assessment based on historical efficiency data;

– SSC-1 and XC-2 – for representing cloud components based on AWS, which includes the representation of the application and integration of cloud infrastructure services (SSC-1) and the infrastructure deployment and management strategy (XC-2).

– Deployment viewpoint (XC-3) – for describing the deployment strategy of application components via a CI/CD pipeline (Jenkins based on code from GitHub) to AWS Lambda and Amazon ECS, as well as AWS infrastructure management via Terraform.

For the "Equipment condition and maintenance registry" subsystem (Behavior-oriented: domain-data-model-oriented):

– XA-1 – for describing UX flows and detailed Web-UI designs;

– SSA-3 and SSA-3-ext – for the logical decomposition of the subsystem into abstract application and infrastructure level components (web application as a custom application-level component and a relational DBMS as the infrastructure level);

– SSC-1 – represents the software components of the subsystem: the custom registry web application (Java) as an application-level software component running in Amazon ECS, and Amazon RDS (Postgres, managed) as an infrastructure software component;

– XC-2 – for describing the cloud infrastructure of the subsystem based on AWS Cloud;

– XC-3 – for describing the deployment strategy of the registry web application via a CI/CD pipeline (Jenkins based on code from GitHub) to Amazon ECS and AWS infrastructure management (including RDS) via Terraform.

For the "Historical data visualization system" subsystem (off-the-shelf self-hosted, platform-like) – according to the method [3] – full internal architectural design is not performed. A limited set of representations is applied:

– XA-1 – for describing the UX flows of an analyst when viewing charts of historical GPU data and equipment condition assessments;

– SSC-1 – for describing exactly which off-the-shelf components form the subsystem (Grafana instance deployed in Amazon ECS);

– XC-2 – for describing the cloud infrastructure deployment environment of the subsystem within the AWS Cloud;

– XC-3 – for describing the deployment and update strategy of the Grafana instance, as well as the management of project-specific configuration of dashboards and data sources.

For each of the custom application-level software components of each subsystem, as well as for platform-like off-the-shelf components, relevant component-scale architectural representations were defined.

For the custom component "PLC – GPU control" (PLC program; FBD) as part of the GPU control subsystem (Continuous Control):

– Execution viewpoint (CC-1) – for describing the execution of PLC organization blocks and the distribution of tasks in the PLC execution environment;

– States / Stages viewpoint (CC-2) – for describing the discrete operating modes of the GPU (stopped, startup, normal operation, emergency shutdown) and the rules for transitioning between them; each mode corresponds to its own set of control rules, traced to SSA-1;

– CC-3 – is not applied – the modular structure of the PLC program in FBD format is trivial and is fully covered through CC-1 (execution of organization blocks) and CC-2 (discrete operating modes);

– CC-5 – description of MQTT events used for integration with the analytical data processing subsystem;

– Domain data model viewpoint (CC-6) – in PLC tags catalog format;

– I/O signal interfaces and pins viewpoint (CC-8) – for displaying the mapping of PLC inputs/outputs to physical GPU signals (measurements of pressures, temperatures, speeds, actuator control).

For custom application-level components of the "Analytical data processing" subsystem (Data-flow-oriented), which include the "Data aggregation component" (Spark-job on AWS Glue), the "Real-time GPU condition assessment component" (Real-time state estimate; Node.js on AWS Lambda), and the "Regular GPU condition assessment component" (Periodic estimate; Java on Amazon ECS), the following are applied for each component:

– CA-1 and CA-1-ext – for the catalog of processings indicating MQTT/Kinesis as the access method for real-time processings and scheduled triggers via EventBridge for regular processings;

– CA-2 – not applied – cross-behavioral dynamics between analytical processing components are fully described in data flow diagram format within SSA-3 at the subsystem level;

– CC-1 – not applied for the "Real-time GPU condition assessment component" and the "Data aggregation component"; applied for the "Regular GPU condition assessment component" – internal task concurrency during batch processing;

– CC-3 – not applied for the "Real-time GPU condition assessment component" and the "Data aggregation component" because the internal modular structure is trivial. For the "Regular GPU condition assessment component" (Java on Amazon ECS), CC-3 is applied to detail the modular structure of the Java application;

– Contracts / APIs / Actions viewpoint (CC-4) – for the specification of the input and output events structure (event schema) in AsyncAPI format;

– Integrations APIs viewpoint (CC-5) – for describing outgoing integrations between components – in this case, via AWS SQS (transmitting state assessment results to the registry);

– CC-6 – not applied – within the components of this subsystem, an internal domain model separate from the persistence structure is not introduced; the persistence structure is fully described in CC-7.

– Persistent data model viewpoint (CC-7) – for describing the structure of raw telemetry and aggregated data in S3 buckets and AWS Glue Data Catalog tables for Athena;

– Calculation viewpoint (D-2) – for mathematical models of real-time GPU condition assessment and daily state assessment based on historical efficiency data;

– Task / Transformation viewpoint (D-3) – for the rules of normalization, filtering, and enrichment of input telemetry, as well as aggregation rules when forming daily aggregations.

For the custom component "Equipment condition and maintenance registry web application" (Registry web app; Java) as part of the "Equipment condition and maintenance registry" subsystem (Behavior-oriented, hybrid of computation-oriented and domain-data-model-oriented):

– CA-1 and CA-1-ext – for the catalog of behaviors indicating API types (REST API, SSR/WebForm, event consumer from AWS SQS for receiving GPU condition assessment registrations, scheduled triggers);

– Behavioral Dynamic viewpoint (CA-2) – in sequence diagram format for cross-behavioral scenarios;

– Execution viewpoint (CC-1) – is not applied, since concurrency and execution aspects are standard for a backend web application with a given technical stack;

– Component structural viewpoint (CC-3) – in C4 component diagram format for the modular structure of the Java application;

– CC-4 – for specifying the REST API in OpenAPI format and AWS SQS (receiving real-time and regular GPU condition assessment registrations) in AsyncAPI format;

– Integrations APIs viewpoint (CC-5) – not applied, since the software component does not implement outgoing integrations via API with other systems/components;

– Domain data model viewpoint (CC-6) – in class diagram format for the domain model (Equipment, Maintenance record, State assessment, User, Role, Audit record);

– Persistent data model viewpoint (CC-7) – in ERD format for the Postgres schema in Amazon RDS;

– CC-9 – not applied – the registry does not use BPMN orchestration; stateful logic is limited to individual equipment and maintenance records in the persistence layer and does not form orchestration processes.

– Procedure viewpoint (D-1) – for detailing the logic of key behaviors.

– D-2 – not applied – the computational logic of the registry (validation of data consistency, formation of maintenance reminders based on MRO schedules) does not contain mathematical models per se and is fully covered through D-1.

For platform-like off-the-shelf subsystems – GPU-level and CS-level SCADA WinCC Server – at the component-scale design level:

– CA-1-ext and CC-4 – for documenting API extensions on each of the SCADA subsystems, if necessary;

– CC-5 – for describing SCADA system integrations (e.g., via MQTT or ModBus);

– CC-6 – for specifying the project-specific tag hierarchy of each SCADA subsystem (GPU-level SCADA tags cover the parameters of an individual unit; CS-level SCADA tags – a consolidated tag model of all station GPUs);

– CC-7 – not applied to SCADA subsystems, since the schema of their databases (MS SQL) is not a subject of introducing extensions or changes.

For the platform-like off-the-shelf component "Historical data visualization system" (Grafana instance):

– CA-1-ext and CC-4 – are not applied – custom API extensions in the Grafana instance are not implemented; project-specific configuration is limited to dashboards and data sources and is covered through XC-3 at the subsystem level;

– CC-5 – not applied (custom integrations are not implemented);

– CC-6 – not applied (a domain model is not introduced in the visualization tool);

– CC-7 – is not applied (the subsystem uses the persistence data of the analytical subsystem as an external source without introducing schema extensions).

For each applied architectural viewpoint, an additional governance document has been defined that describes the project-specific standardization of representations.

5. 3. Architectural design at the subsystem-scale and component-scale level

5. 3. 1. Control viewpoint representation – GPU Control System

According to the applicability matrix of architectural viewpoints [3], for the Continuous control system, the "Control viewpoint" (SSA-1) is relevant at this step. This representation is optional, as some continuous control systems can be simple enough and not require detailing the control logic.

The GPU control system is complex, since the control object itself is complex and depends on a significant number of parameters [27–29].

For this system, constructing a "control viewpoint" representation is necessary.

In general, the method allows designing control systems of various complexity levels. Within the scope of this study, for demonstration purposes, a simplified GPU control model was chosen.

For this control system, a control model was constructed, represented by a block diagram (Fig. 2). This control diagram allows understanding the control logic within the system and is necessary for the correct implementation of the logic of the designed CSS.

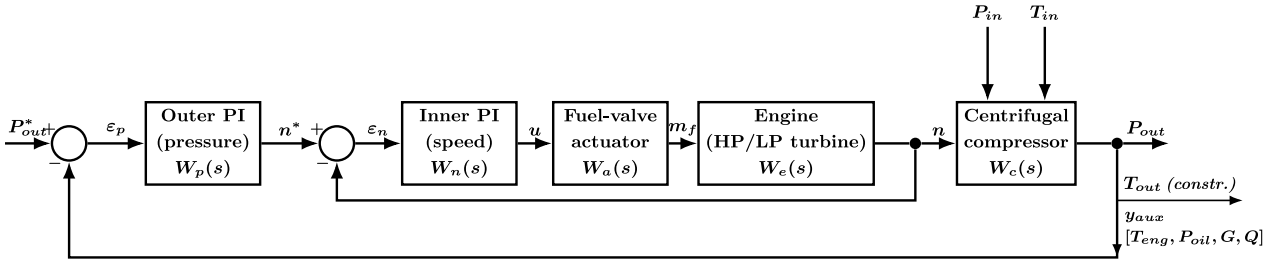

Fig. 2. GPU control block diagram (SSA-1)

The provided representation from the SSA-1 viewpoint describes the GPU automatic control system in steady-state operation mode (RUNNING state according to CC-2) in the format of standard notation of automatic control theory (ACT). The model is technology-agnostic: it captures the canonical control structure and dynamic characteristics in the form of typical transfer functions. Control rules can potentially differ for each individual state of the control system – in this case, an appropriate representation is built for each state of the control system.

The system is built according to a cascade two-loop SISO scheme with PI controllers.

The classification of control system parameters (within the control architectural viewpoint) is presented in Table 4.

Table 4

Parameter classification

Parameter	Designation	Parameter category
Target gas pressure at the compressor outlet (setpoint)	P^*_{out}	reference input
Gas pressure at the compressor inlet	P_{in}	disturbance input
Inlet gas temperature	T_{in}	disturbance input
Outer control loop error	$\varepsilon_p = P^*_{out} - P_{out}$	error signal
Target low-pressure turbine rotational speed	n^*	inner-loop reference
Inner control loop error	$\varepsilon_n = n^* - n$	error signal
Control signal to actuator	u	manipulated input
Fuel mass flow	m_f	actuator output
Actual rotational speed of the low-pressure turbine rotor	n	controlled internal variable
Actual gas pressure at the compressor outlet	P_{out}	controlled output
Actual gas temperature at the compressor outlet	T_{out}	constrained observed output
Vector of auxiliary outputs	$y_{aux} = [T_{eng}, P_{oil}, G, Q]^T$	observation outputs

The outer loop regulates the gas pressure at the compressor outlet P_{out} – the main technological goal of the unit's operation at a main gas pipeline CS. The $W_p(s)$ controller calculates the target rotational speed n^*, which serves as a setpoint for the inner loop.

The inner loop regulates the actual rotational speed of the low-pressure turbine (LPT) rotor n, responding to the target value n^*. The $W_n(s)$ controller produces the control signal u to the fuel valve actuator, which determines the fuel mass flow m_f into the combustion chamber.

The cascade structure is canonical for regulation and corresponds to the implementation in real GPU ACS. Cascading is motivated by the significant difference in the time constants of the gas dynamics of the compressor and its drive ($T_c \gg T_e$). The inner loop quickly responds to disturbances in the operation of the fuel valve and drive (pressure changes in the fuel gas line, temperature drifts) before they manage to manifest as disturbances in the regulated pressure.

5. 3. 1. 1. Controller transfer functions

The outer and inner loops are implemented by PI controllers in classical parallel form:

$$W_p(s) = K_p\,(1 + 1/(T_{ip} \cdot s)),$$

$$W_n(s) = K_n\,(1 + 1/(T_{in} \cdot s)).$$

The derivative component is not used in both loops due to the noisiness of the measured signals: pressure P_{out} contains gas-dynamic pulsations of turbulent flow, the speed signal n contains mechanical vibrations of the turbine. Introducing the D-component would amplify these noises and cause control instability.

5. 3. 1. 2. Linearized dynamic models of the plant

The fuel valve actuator is approximately described by a first-order aperiodic block

$$W_a(s) = 1 / (T_a \cdot s + 1).$$

The engine dynamics (HPT → LPT → compressor rotor linkage) relative to the fuel flow

$$W_e(s) = K_e / (T_e \cdot s + 1).$$

The compressor with the gas volume of the supercharger is described by a non-linear characteristic $P_{out} = \Phi(n, P_{in}, T_{in})$, the linearization of which around the operating point yields

$$W_c(s) \approx K_c / (T_c \cdot s + 1).$$

The exact non-linear model of a two-stage compressor taking into account losses due to gas leaks, disk friction, and the polytropic nature of compression is the subject of a separate study and goes beyond the linearized SSA-1 representation.

5. 3. 1. 3. Recommendations for implementation in the PLC program

The structure of this diagram maps directly to the Core Pump Control task from CC-1 (cycle time 20–50 ms):

– $W_p(s)$ and $W_n(s)$ controllers are discretized (Tustin or backward Euler) with a step equal to the task cycle. A sampling rate ≥ 100× relative to the dominant time constant

$T_e \approx 5$ s provides a sufficient margin according to the canonical recommendation $\geq 10\times$;

– output u is written via the PLC's AO module to the fuel valve actuator (XC-2: Process valves block);

– the actual speed n is read from the discrete speed relay or analog tachometer (XC-2: Speed / flame (discrete) block);

– the pressure P_{out} is read from a 4–20 mA transmitter via the AI module (XC-2: Pressures & levels block);

– the y_{aux} vector is written in parallel to the PLC data block and consumed by the Telemetry Preaggregation task (CC-1) for further publication via MQTT Publisher according to the CC-5 contract.

5. 3. 1. 4. Model limitations

SSA-1 describes exclusively the steady-state operating mode (RUNNING state, described in the viewpoint representation CC-2). Transient modes (startup, shutdown, emergency transitions) are controlled by the discrete logic of the finite state machine CC-2. Protective trips (overspeed, overpressure, lube oil pressure low, vibration trip, fire/gas detection, $T_{out} > T_{out}^{\max}$) are implemented in the Emergency & Protection task (CC-1) as discrete-event triggers initiating the transition of the FSM to the EMERGENCY_STOP or FAULT state.

Anti-surge protection, which in a real system is implemented via the compressor recycle valve (the second control channel), is not represented in the simplified model: the input parameter list of this SSA-1 contains only one control action (n via u).

The cross-coupling between pressure and temperature channels that arises due to the MIMO nature of a real turbine (a change in fuel flow affects not only the speed but also the exhaust gas temperature) is not modeled in the simplified model. The loops are depicted as independent; in a more detailed model, this coupling is described by a perturbation matrix and requires decoupling compensators, which is beyond the scope of the simplified GPU model representation.

5. 3. 2. Representation for the user experience viewpoint

The UX viewpoint (XA-1) is applied at the subsystem level, but is cross-subsystem (moreover, relevant only for subsystems that have user interaction).

Within the designed system for the user experience viewpoint (XA-1), a use-case representation is defined (Fig. 3), which is common to all subsystems of the system with which interaction occurs.

Furthermore, within the XA-1 viewpoint, UI designs have also been built for the "GPU equipment condition and maintenance registry" subsystem. An example of the GPU list screen is presented in Fig. 4.

These UI designs were built based on system requirements and the representation of system capabilities assigned to this subsystem, using Claude Design (Anthropic PBC, USA).

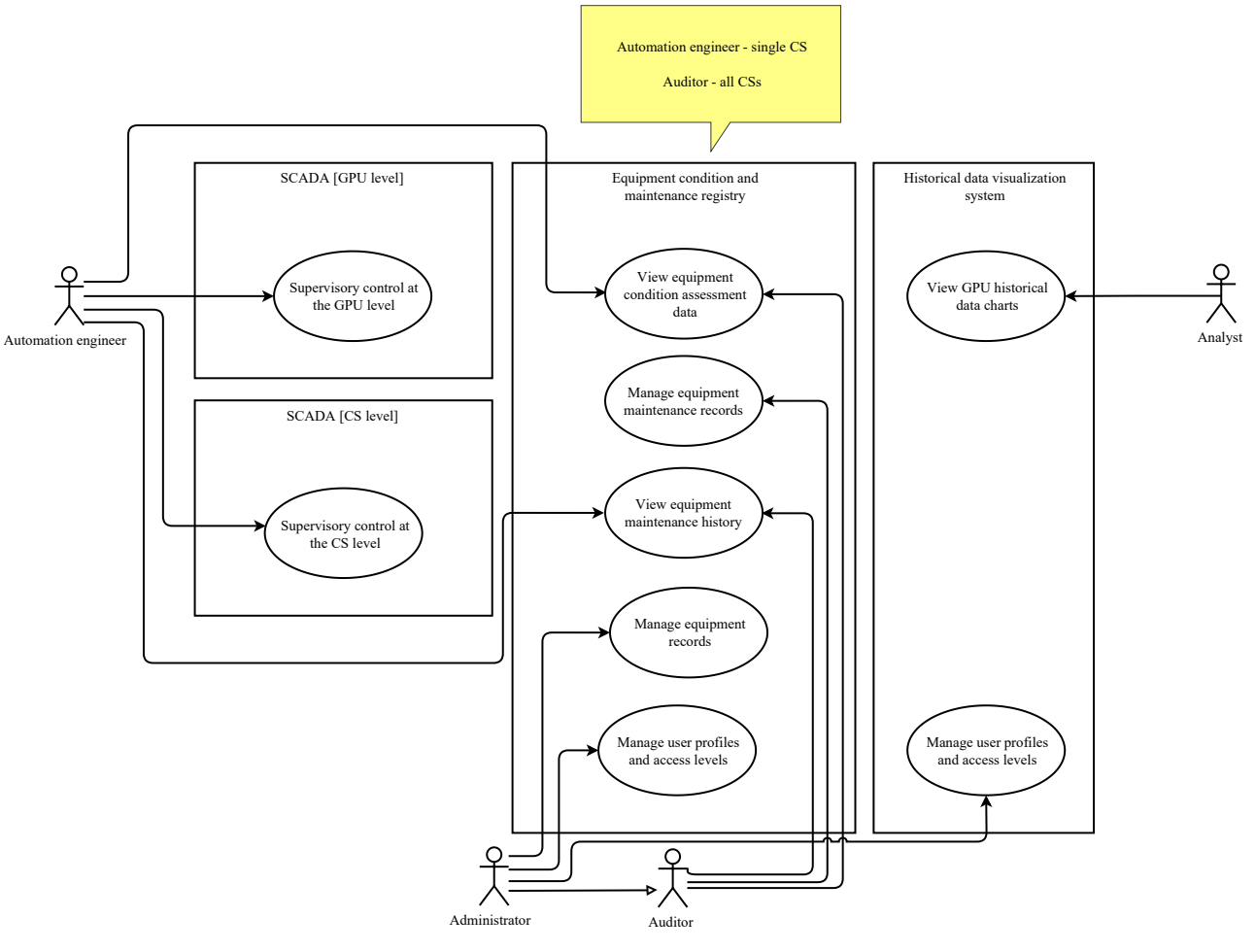

Fig. 3. Use-case diagram for the system (XA-1)

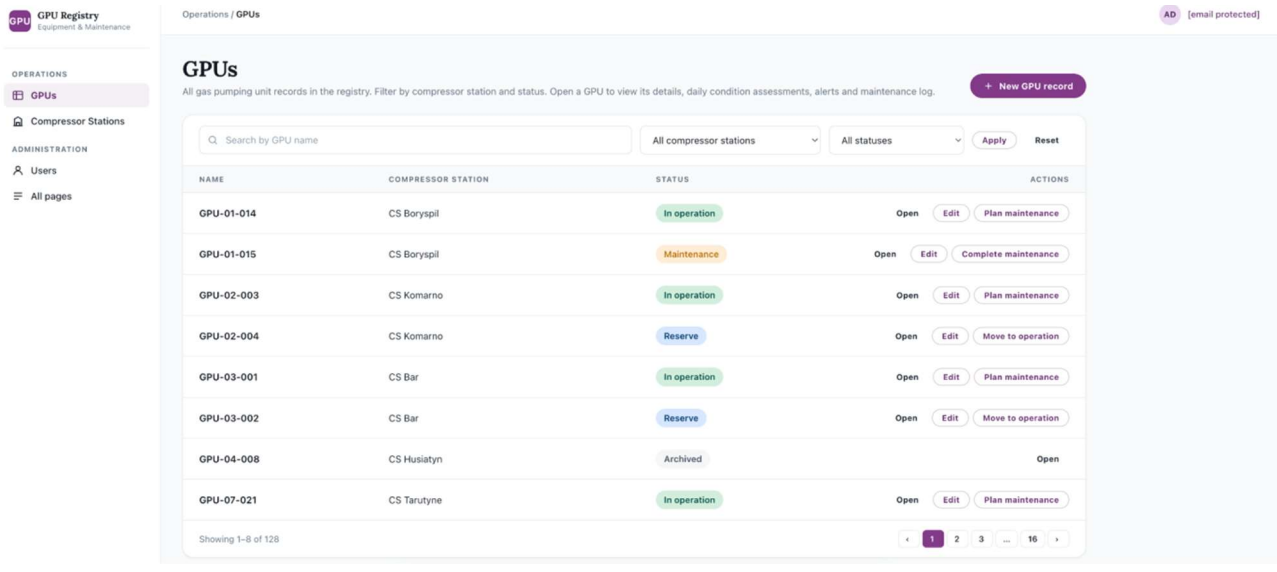

Fig. 4. UI design of the "GPU equipment condition and maintenance registry" subsystem: GPU list screen (XA-1)

5. 3. 3. Logical components representation (SSA-3, SSA-3-ext)

According to the method, the representation of logical components (SSA-3, SSA-3-ext) was built at the next stage.

For the sake of representation compactness within the scope of this article, the components of all subsystems are presented on a single diagram, with the subsystem boundaries highlighted.

Fig. 5 shows the representation of the logical components of the subsystems with the classification of components SSA-3-ext.

The created representations from the viewpoint of logical components highlight cohesive groups of functionality based on subsystem capabilities and serve as input for further design stages at the subsystem and component levels.

5. 3. 4. Abstract design at the component level

At this stage, architectural representations from the CA-1 viewpoint were built for the components of custom subsystems of "Behavior-oriented" and "Data-flow-oriented" types, for which this viewpoint is applied [3].

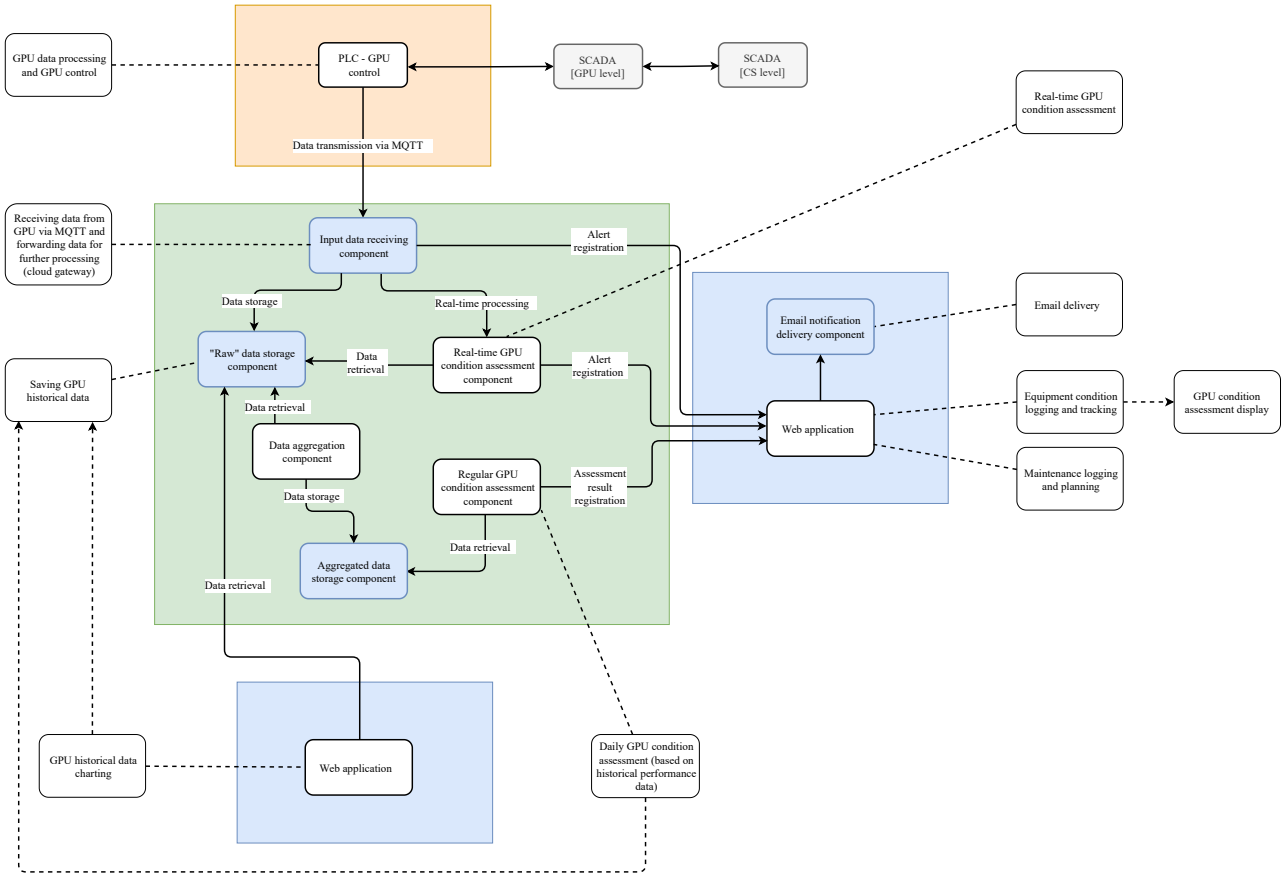

Fig. 5. Logical components representation of subsystems with component classification (SSA-3-ext)

Table 5 shows the representation from the CA-1 viewpoint for the "Real-time GPU condition assessment component" of the "Analytical data processing system" subsystem.

Table 5

CA-1 viewpoint representation for the "Real-time GPU condition assessment component" of the "Analytical data processing system" subsystem

Processing	Description of processing logic
Processing of GPU telemetry event	Checking parameters for critical deviations: – "Compressor rotational speed" > *X*; – "Inlet gas temperature" > *X*; – "Outlet gas temperature" > *X*; – "Compressor temperature" > *X*; – "System oil pressure" > *X* || < *Y*. In case of detecting parameters going out of range, a message is sent to the queue

Table 6 shows the representation from CA-1 viewpoint for the "Data aggregation component" of the "Analytical data processing system" subsystem.

Table 6

CA-1 viewpoint representation for the "Data aggregation component" of the "Analytical data processing system" subsystem

Processing	Description of processing logic
Daily data aggregation	Reading detailed GPU events for the last day. Calculation of daily average values for each GPU. Saving aggregated data

Table 7 shows the representation from the CA-1 viewpoint for the "Regular GPU condition assessment component" of the "Analytical data processing system" subsystem.

Table 7

CA-1 viewpoint representation for the "Regular GPU condition assessment component" of the "Analytical data processing system" subsystem

Processing	Description of processing logic
Conducting GPU efficiency assessment	Assessment of historical trends: – comparison of the «Gas consumption» / «Volumetric capacity» ratio for the current day and the average over each of the past weeks; – comparison of the «Engine temperature» / «System oil pressure» ratio for the current day and the average over each of the past weeks; – comparison of the («Inlet gas temperature» * «Inlet gas pressure») / («Outlet gas temperature» * «Outlet gas pressure») ratio for the current day and the average over each of the past weeks. As a result of the processing, a «daily report» message is sent to the queue. The aggregated table of daily average values is used for the query

Table 8 shows the representation from the CA-1 viewpoint for the "Web application component" of the "Equipment condition and maintenance registry" subsystem.

The user password reset flow requires additional clarifications to explain the cross-behavioral logic that forms the flow itself. For this purpose, a representation from the CA-2 architectural viewpoint was built (Fig. 6).

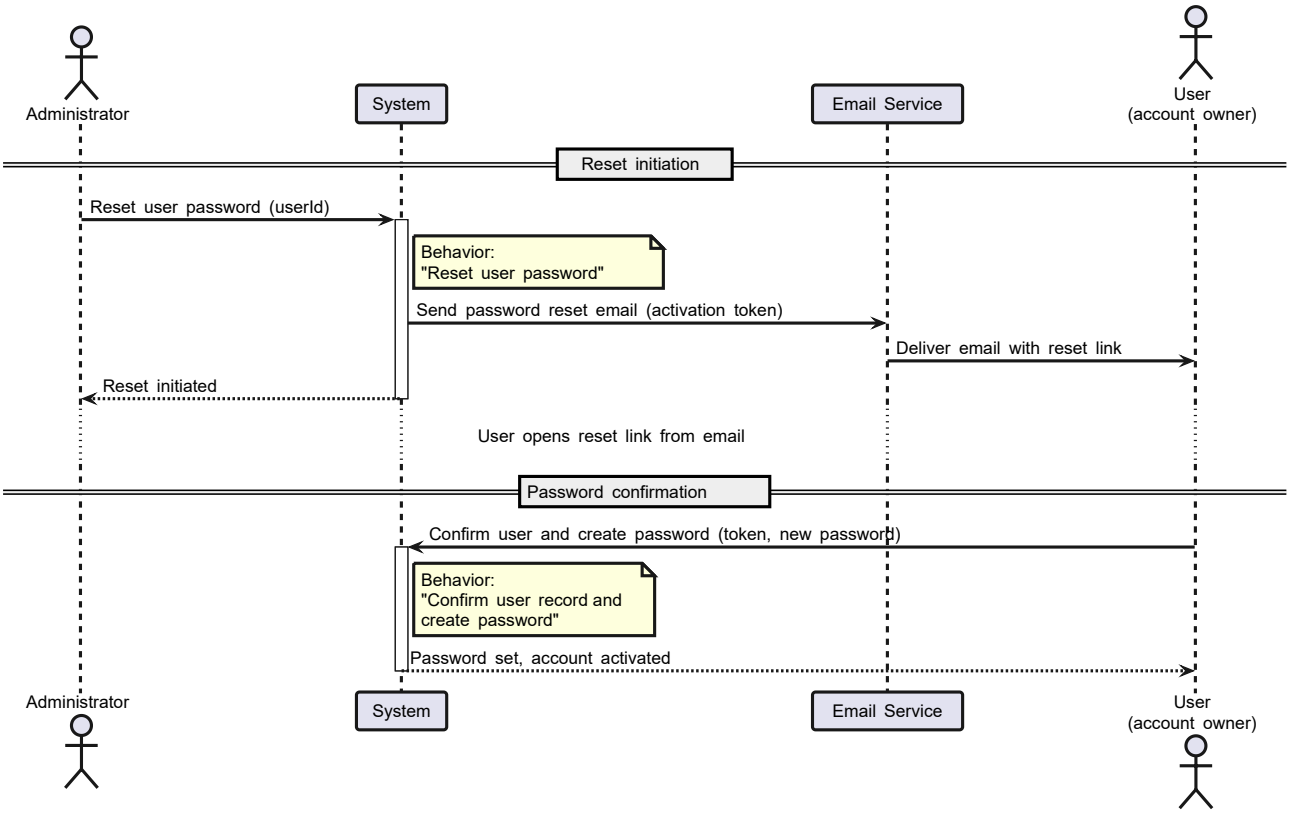

Fig. 6. Sequence diagram of the cross-behavioral user password reset flow for the "Web application" component of the "Equipment condition and maintenance registry" subsystem (CA-2)

Table 8

CA-1 viewpoint representation for the "Web application component" of the "Equipment condition and maintenance registry" subsystem

Behavior	Description of behavior logic	Trigger
Create CS record	Create CS record: status «active»/ «inactive», unique ID, name	Administrator
Update CS record	Set status «active»/ «inactive», unique ID, name	Administrator
View all CS records in the system	View the list of records about CS in the system	Administrator
User login to the system	Login with username and password	Unauthenticated user
User logout from the system	Terminate login session	Authenticated user
Obtaining a token for working with API	Obtaining a token for API access	Authenticated user
Create a user record	Creates a new user record. Input parameters: email, first name, last name, role, CS. User is inactive. Sends a confirmation email to the mail (with an activation token)	Administrator
Confirm a user record and create a password	Input parameters: activation token, new password	Unauthenticated user
Block a user record	Blocks a user record	Administrator
Update password	The user enters the old and new passwords	Authenticated user
Reset user password	Deactivates the user. Sends a password reset email to the mail (with a token for activation)	Administrator
...		
Record a daily GPU condition assessment	Received for a group of GPUs (multiple state assessments for different GPUs in one request)	Event / request from another component / subsystem
Record an alert for a GPU	Received for a specific GPU	Event / request from another component / subsystem

This representation describes the dynamic aspect of combining behaviors into a complete flow.

The representations formed at this stage serve as input data for concrete design at the subsystem level, taking into account the necessary components of external interaction with the system and interaction between components. They also serve as input data for the detailed design stage at the component level, since the behaviors and processings of abstract logical components serve as the basis for forming the API of application-level software components (or platform-like subsystems).

5. 3. 4. Concrete design at the subsystem level

Based on the architectural representations for viewpoints I-2, SSA-3-ext, XA-1, CA-1 [3], a representation of physical software components (SSC-1) was designed (Fig. 7).

It is important to note that the decision to use an off-the-shelf solution for SCADA was made at the system design level – SCADA was introduced as a subsystem [30–32]. The decision to use Grafana was made at the subsystem design level – Grafana was introduced as an off-the-shelf self-hosted application-level component. In both cases, the choice was made at a time when there was enough information for it – both cases are permissible from the perspective of the design method.

The next step was to build representations from the XC-2 viewpoint: Infrastructure viewpoint. For this system, a general representation of the system infrastructure was built (Fig. 8), which covers aspects of deploying a hybrid infrastructure in the physical environment of the CS (on-premises) and in the cloud environment (cloud) [33, 34].

In addition, for the "GPU control system" subsystem, a representation from the XC-2 viewpoint at the subsystem level was built (Fig. 9), which expands the description of the subsystem by indicating infrastructure details at the subsystem level, including specific sensors and actuators.

The deployment strategy aspects for subsystem components were designed and documented within XC-3: Deployment viewpoint (Fig. 10).

The presented representations for the architectural viewpoints of abstract and concrete design at the subsystem level capture most of the important architectural decisions for a given system. However, for the practical implementation of the system, designs at the component scale and detailed scale are also necessary to cover software implementation aspects.

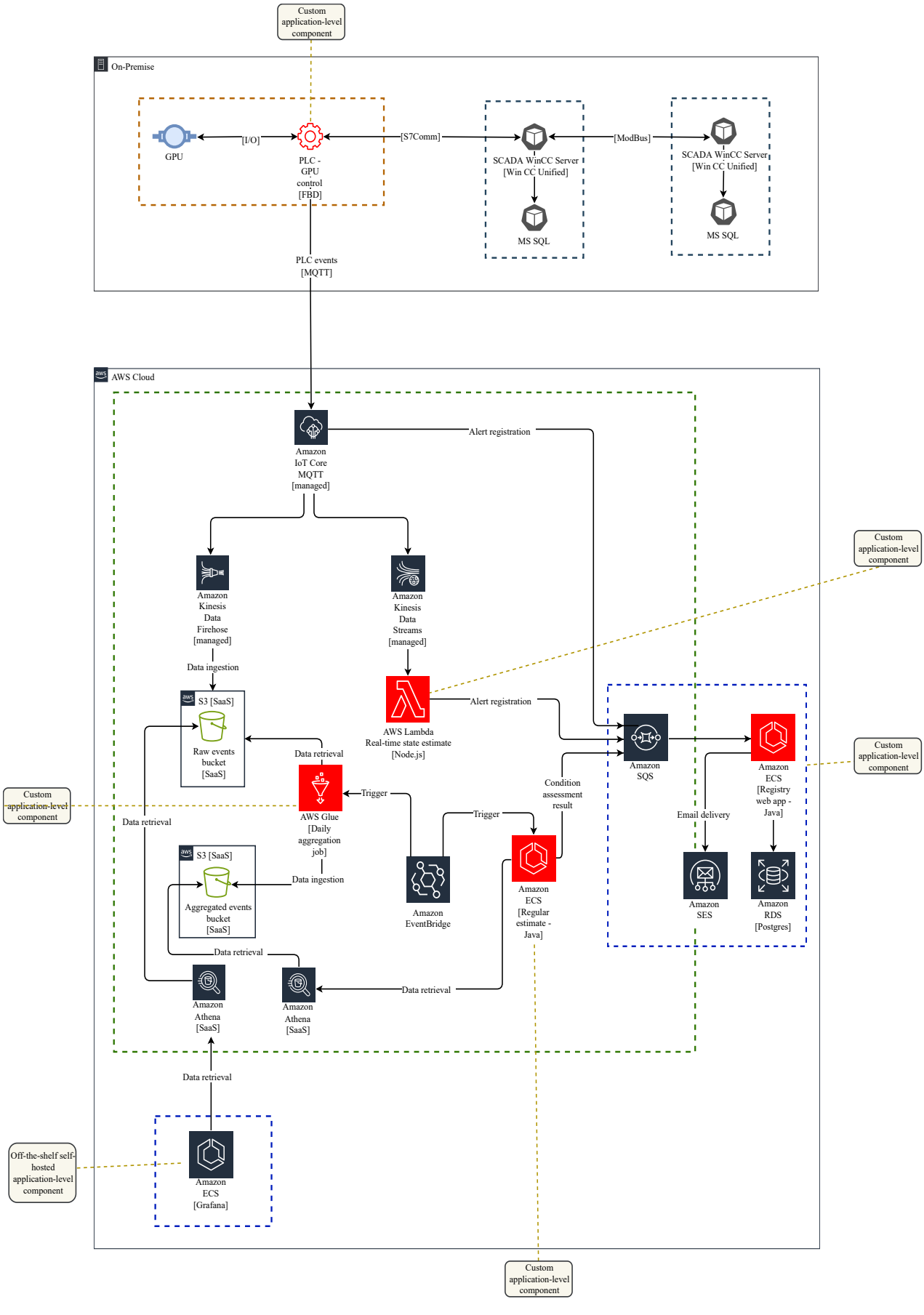

Fig. 7. Representation of physical software components of the system (SSC-1)

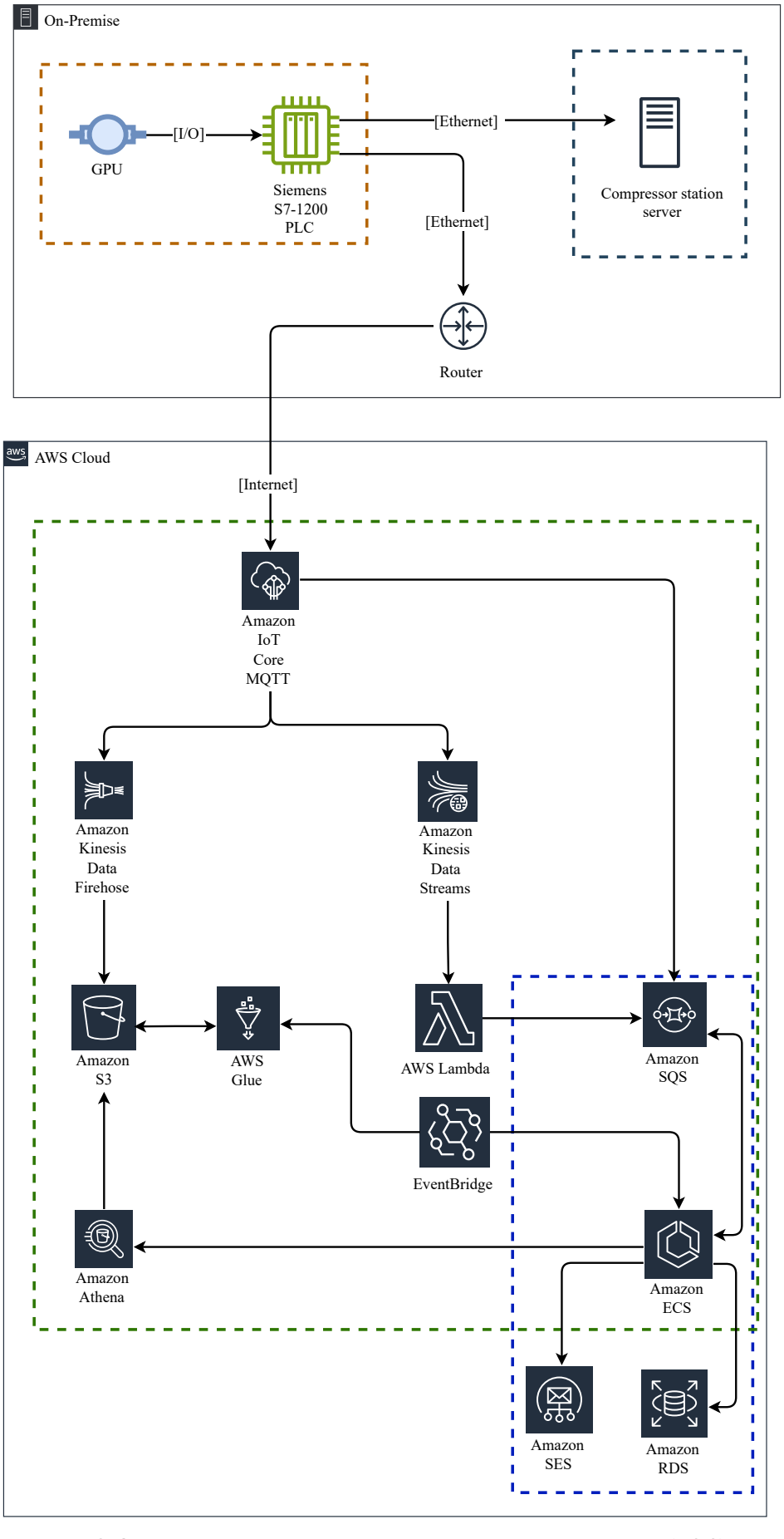

Fig. 8. General representation of the hybrid infrastructure of the system (XC-2)

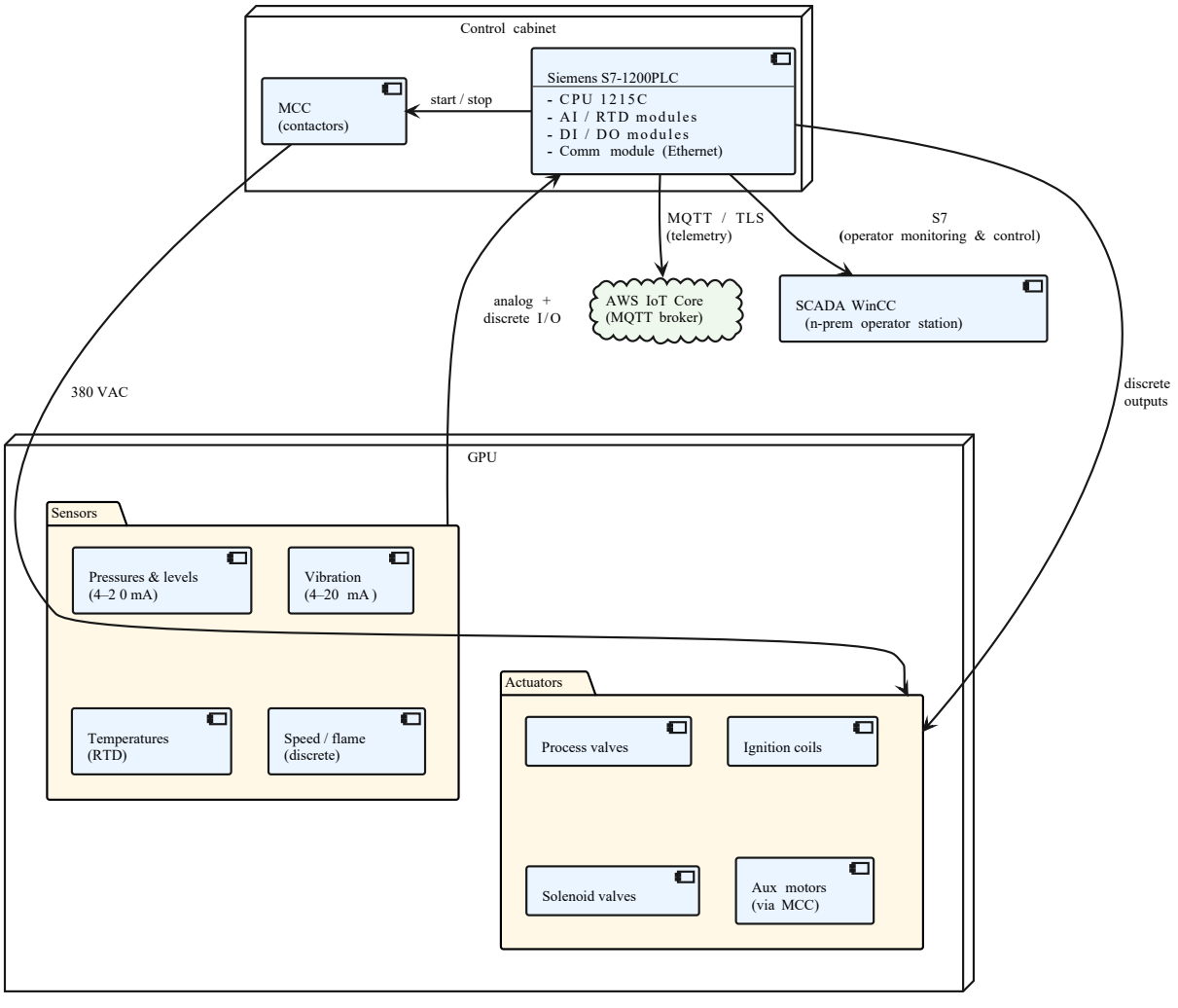

Fig. 9. Infrastructure representation of the “Gas pumping unit control system” subsystem detailing sensors and actuators (XC-2)

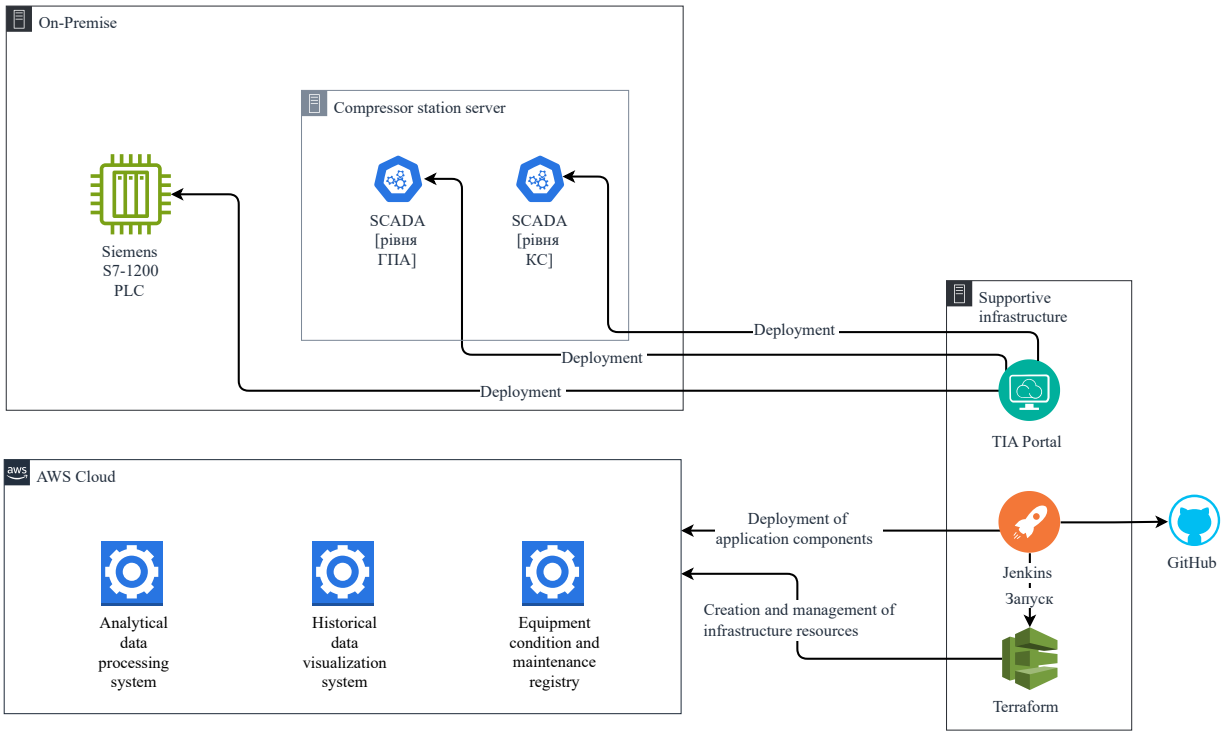

Fig. 10. Representation of the deployment strategy for subsystem components (XC-3)

5. 3. 5. Concrete design at the component scale and detailed scale level

Based on the designed architecture of the system scale and subsystem scale, in accordance with the method [3], component-scale representations were built.

Within this article, selective component-scale representations resulting from the design are presented.

To reflect the runtime tasks of the "PLC – GPU control" component of the "GPU control system" subsystem, a representation from the CC-1 architectural viewpoint was built, depicted in Fig. 11.

CC-2 models the internal finite state machine of the PLC program controlling a single GPU (component "PLC – GPU control") (Fig. 12). These states describe discrete system states that alter the controller's behavior according to the state.

This representation defines the following states of the PLC program's state machine:

– STOPPED – the unit is stopped (zero rotational speed) and awaits startup. Covers hot and cold standby;

– PRECHECK – checking pre-start conditions (power availability, no fire/gas contamination, readiness of protections and valves). Failure to meet conditions returns to STOPPED, successful check allows startup;

– STARTING – a composite startup state containing six sub-states^

– AUX_STARTUP – starting auxiliary mechanisms (pumps, fans), opening gas valves, checking systems and activating main protection;

– PURGE – purging the gas path at reduced speeds from the starting turbine. In "cranking" mode, the cycle ends here;

– IGNITION – ignition in the combustion chambers and monitoring the appearance of a flame within a limited time. In case of failure – fixing a failure or emergency stop (depending on the mode);

– WARMING – brief warming at a stable fuel supply; activating operating hours counters. In "ignition" mode, the unit is held in this state;

– ACCELERATION – accelerating the rotors by increasing the fuel supply with temperature control. The starting turbine is disconnected, the rotors reach idle speeds;

– SYNCHRONIZATION – switching the power supply to its own generator, stabilizing the rotors at nominal speed and transitioning to RUNNING;

– RUNNING – normal operation under load: continuous monitoring of sensors, telemetry transmission, active protective and anti-surge automation;

– NORMAL_STOP – planned shutdown: reducing speed, shifting valves, stopping fuel supply, switching to the external network and operating auxiliary pumps until the rotor completely stops;

– EMERGENCY_STOP – emergency shutdown triggered by a protection signal: instantaneous fuel cutoff and valve shifting for a controlled speed reduction to zero. Restart is blocked;

– COOLDOWN – cooling after shutdown: prolonged operation of the auxiliary oil pump by a timer. Early restart is possible. Upon timer expiration – transition to STOPPED;

– FAULT – critical malfunction with locking (latched) and safe positioning of outputs. Entry is possible from any state (including STOPPED). Transition out of the state – only through a manual reset by the operator.

To outline the architectural decisions regarding the internal structure of the "Web application" component of the "Equipment condition and maintenance registry" subsystem, a representation from the CC-3 architectural viewpoint was built, which is shown in Fig. 13.

Table 9 presents the representation for the CA-1-ext architectural viewpoint for the "Web application component" of the "Equipment condition and maintenance registry" subsystem.

Table 9

CA-1-ext representation for the "Web application component" of the "Equipment condition and maintenance registry" subsystem

Behavior	Description of behavior logic	API / Contract / Actions
Create CS record	Create CS record: status "active"/"inactive", unique ID, name	– WebForm; – REST API
Update CS record	Set status "active"/"inactive", unique ID, name	– WebForm; – REST API
Get CS record creation page	Web page	– WebPageRequest;
Get CS record update page	Web page	– WebPageRequest
View all CS records in the system	View the list of records about CS in the system	– WebPageRequest; – REST API
User login to the system	Login with username and password	– WebForm
User logout from the system	Terminate login session	– WebForm
Get login page	Web page	– WebPageRequest
Obtaining a token for working with API	Obtaining a token for API access	– REST API
...		
Record a daily GPU condition assessment	Received for a group of GPUs (multiple state assessments for different GPUs in one request)	– SQS message
Record an alert for a GPU	Received for a specific GPU	– SQS message

Table 10 presents the representation for the CA-1-ext architectural viewpoint for the "Regular GPU condition assessment component" of the "Analytical data processing system" subsystem.

As can be seen, the trigger for processing in this case is not an API request or user action, but the startup of the program – meaning the described processing is executed by the program after startup and throughout its entire lifecycle. After the processing is completed, the program stops. In this case, since the program has no processings called via an API, constructing a representation for the CC-4 viewpoint for this component is not relevant.

Based on the representation from the CA-1-ext viewpoint, the API design within the framework of the CC-4 architectural viewpoint was built for the "Web application component" of the "Equipment condition and maintenance registry" subsystem. The API for accessing behaviors available via the REST API is described in the OpenAPI format, while the API for behaviors triggered by SQS events is described

in the AsyncAPI format. The AsyncAPI specification of this component contains events sent by the “Real-time GPU condition assessment component” and the “Data aggregation component” of the “Analytical data processing” subsystem. For these components, these events are specified in the CC-5 architectural viewpoint representation.

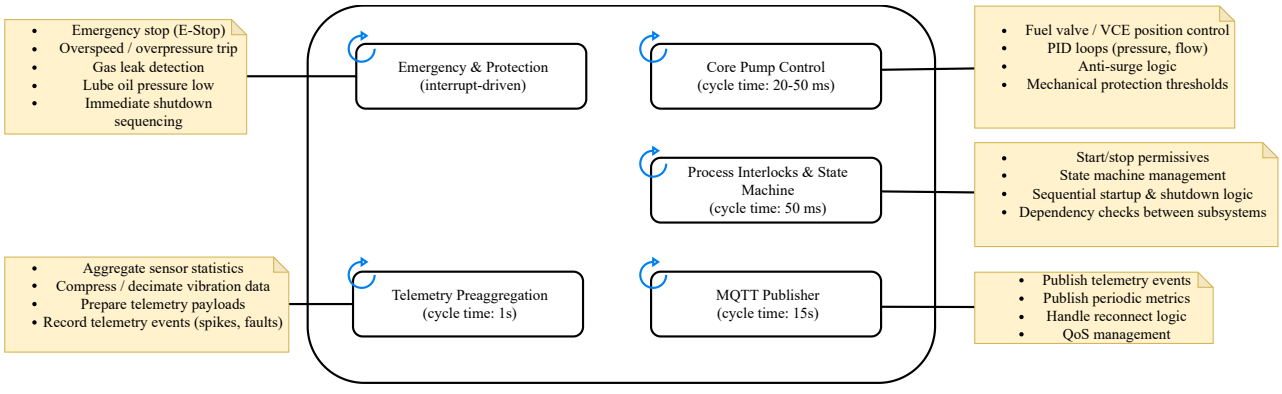

Fig. 11. Representation of runtime tasks for the “Gas pumping unit control” component of the “Gas pumping unit control system” subsystem (CC-1)

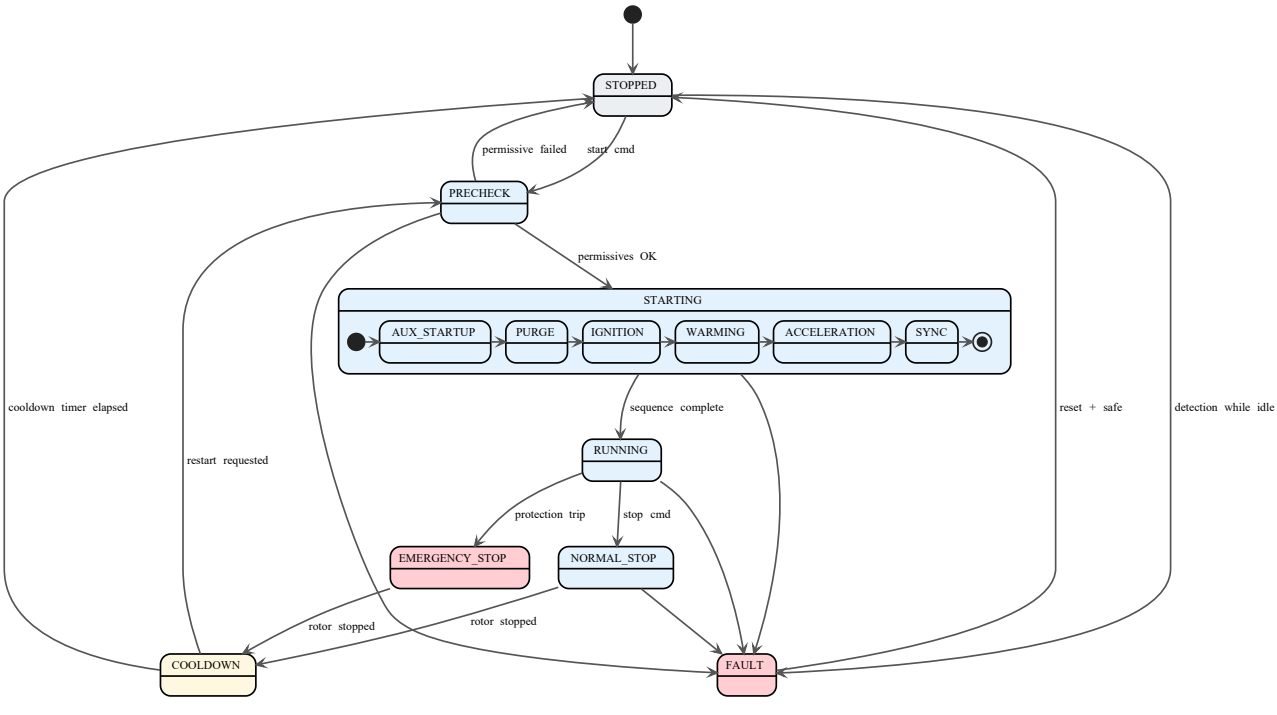

Fig. 12. Finite state machine of the component’s program “Gas pumping unit control program” of the “Gas pumping unit control system” subsystem (CC-2)

Table 10

CA-1-ext representation for the “Regular GPU condition assessment component” of the “Analytical data processing system” subsystem

Processing	Description of processing logic	API / Contract / Actions
Conducting GPU efficiency assessment	Assessment of historical trends: – Comparison of the “Gas consumption” / “Volumetric capacity” ratio for the current day and the average over each of the past weeks; – Comparison of the “Engine temperature” / “System oil pressure” ratio for the current day and the average over each of the past weeks; – Comparison of the (“Inlet gas temperature” * “Inlet gas pressure”) / (“Outlet gas temperature” * “Outlet gas pressure”) ratio for the current day and the average over each of the past weeks. As a result of the processing, a “daily report” message is sent to the queue. The aggregated table of daily average values is used for the query	Run at program startup

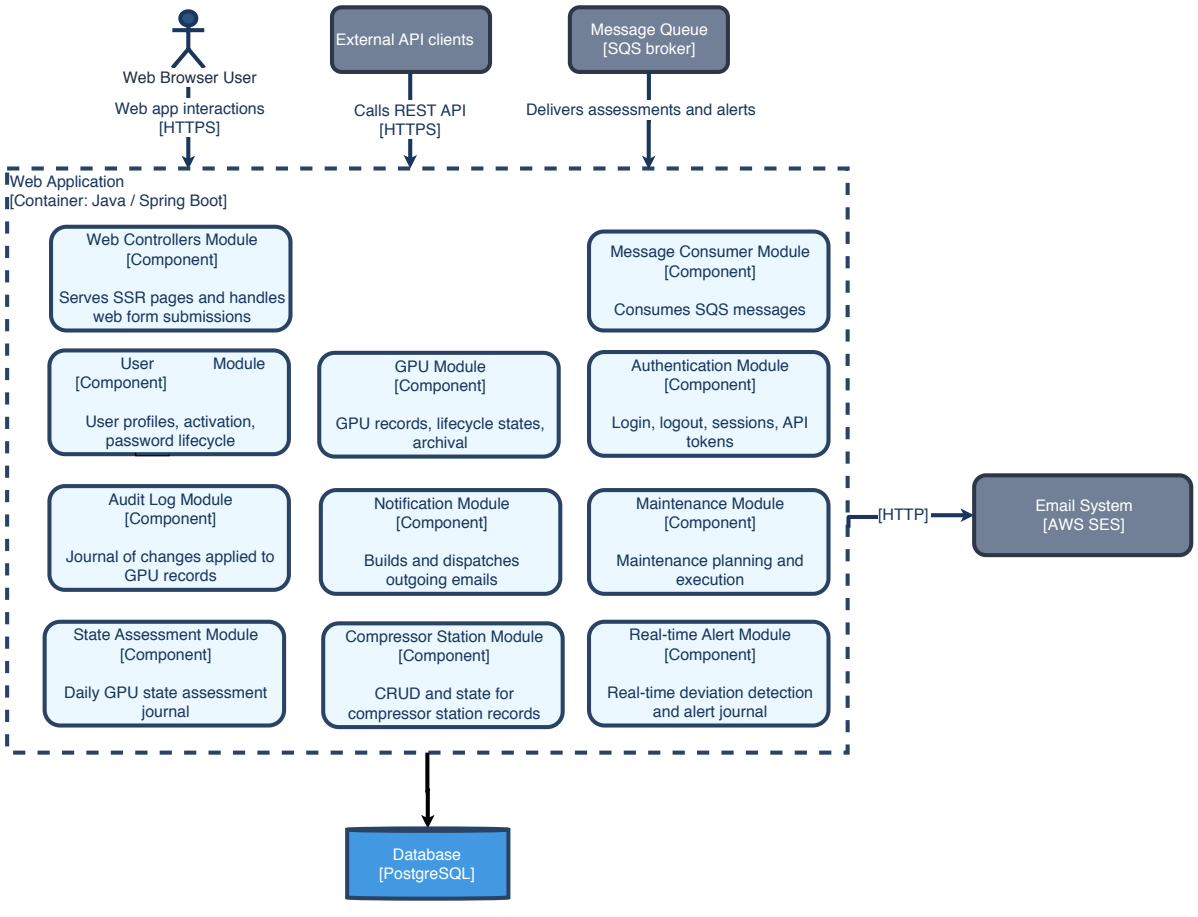

Fig. 13. Internal modular structure of the "Web application" component of the "Equipment condition and maintenance registry" subsystem (CC-3)

For the "PLC – GPU control" component of the "GPU control system" subsystem, a representation from the API integration architectural viewpoint – CC-5 – was built (Fig. 14). This representation specifies the integration contract between the "GPU control system" subsystem and the "Analytical data processing" subsystem, defining the structure of MQTT events and topics.

Fig. 15 shows the representation for the CC-6 architectural viewpoint for the "Web application" component of the "Equipment condition and maintenance registry" subsystem.

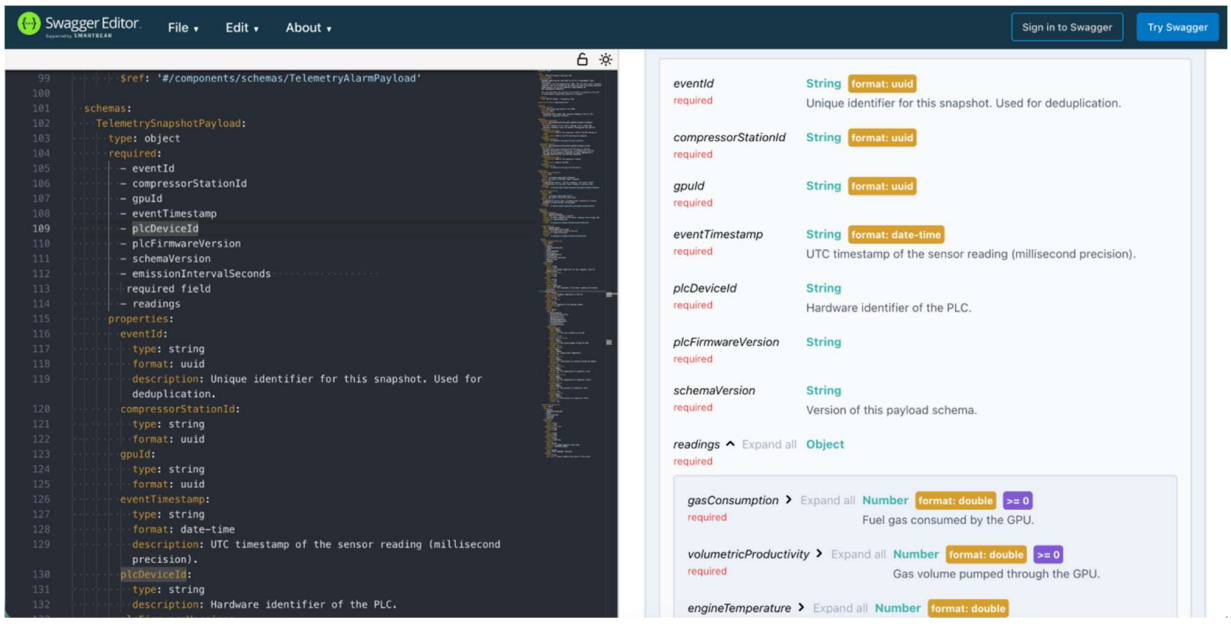

Fig. 14. Representation of API integrations (CC-5) of the "Gas pumping unit control program" component of the "Gas pumping unit control system" subsystem

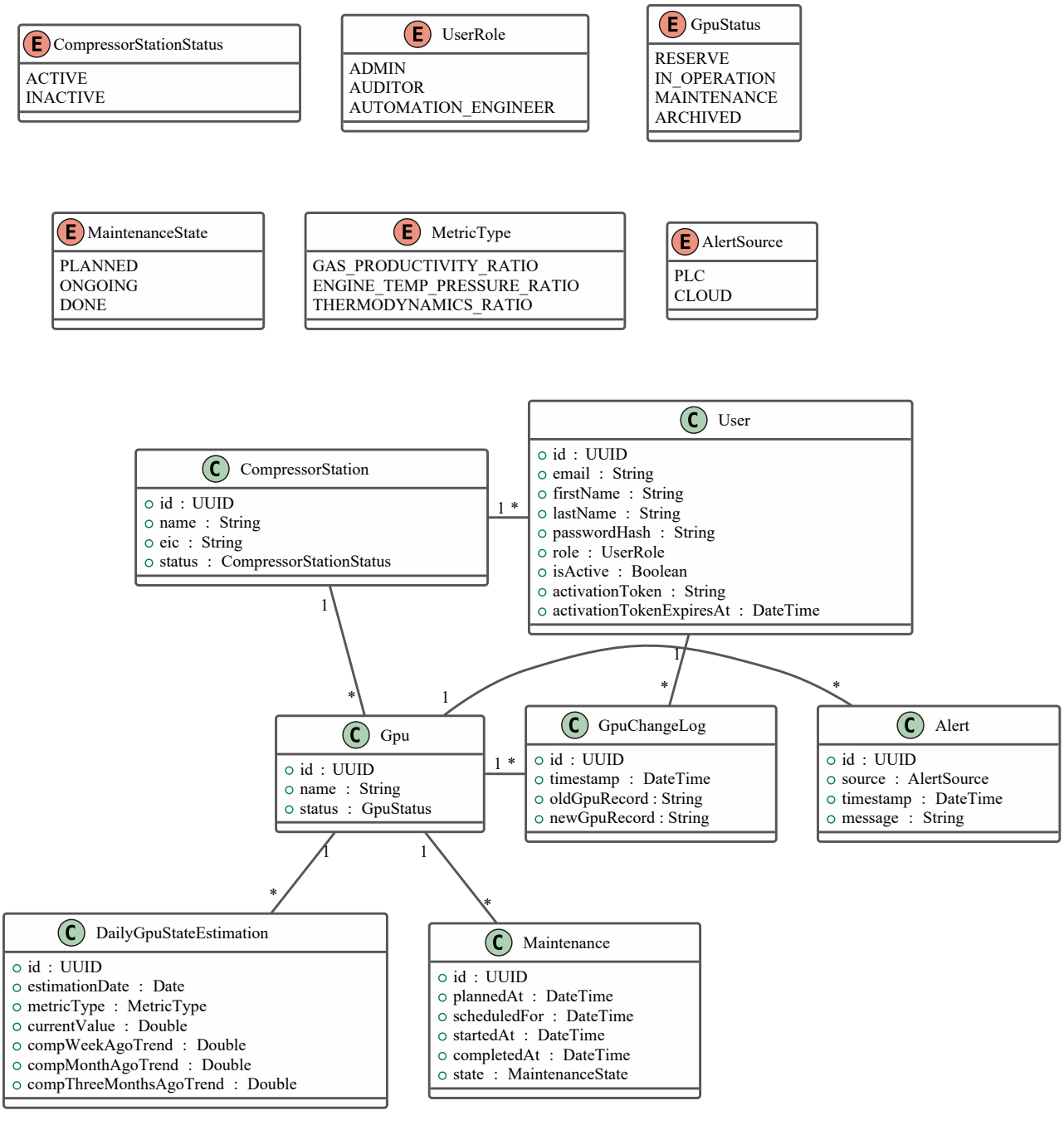

Fig. 15. Domain data model for the "Web application" component of the "Equipment condition and maintenance registry" subsystem (CC-6)

In the scope of the representation from the CC-6 viewpoint for the "PLC – GPU control" component of the "GPU control system" subsystem, the designed domain data model was described to determine the fundamental tags of the component (Table 11).

It is important to note that tags forming interaction contracts (with SCADA and HMI), as well as those tags describing important concepts during the design of the PLC program, must be specified at this stage. At the same time, tags that are implementation details do not have a significant impact on the design of the component and are not mandatory for display in this representation. The complete tag catalog may be overly detailed at the design stage and can be supplemented directly during the PLC program development phase [35].

Fig. 16 shows the representation for the CC-7 architectural viewpoint for the "Web application" component of the "Equipment condition and maintenance registry" subsystem.

Table 11

CC-6 representation for the "PLC – GPU control" component of the "GPU control system" subsystem

Tag name	Data type	I/O address
P_out	Real	IW68
T_eng	Real	IW84
n_LP_actual	Real	IW66 (HSC)
u_fuel_valve	Real	QW64
R_14LS	Bool	I0.5
EStop_PB	Bool	I8.0
...		

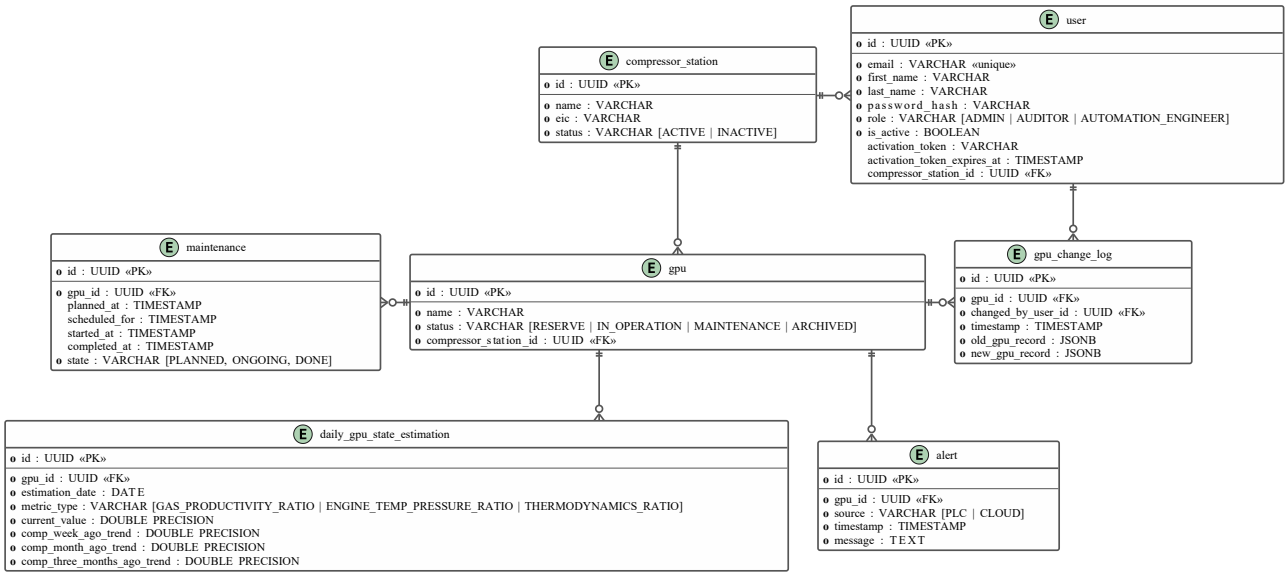

Fig. 16. Persistent data model for the "Web application" component of the "Equipment condition and maintenance registry" subsystem (CC-7)

Table 12

CC-8 representation for the "PLC – GPU control" component of the "GPU control system" subsystem

Module	Pin / Address	Signal type	Tag	Voltage / Current	Purpose
CPU onboard	I0.5	DI, 24 VDC	R_14LS	24 VDC, sinking	LPT speed relay (idle)
CPU onboard	I1.0	DI, 24 VDC	Flame_1	24 VDC	combustion chamber flame sensor
CPU onboard	Q0.0	DO, 24 VDC	Sol_20HD	24 VDC, 0.5 A	hydraulic drain valve
CPU onboard	Q0.4	DO, 24 VDC	Cmd_88QA	24 VDC → MCC contactor	auxiliary oil pump start command
CPU onboard	IW66	AI (HSC)	n_LP_actual	0–10 V / імпульсний	LPT rotor tachometer
CPU onboard	QW64	AO, 4–20 mA	u_fuel_valve	4–20 mA	fuel valve actuator
SM 1231 RTD IW80 (ch0) AI, Pt100, 4-wire			T_out	RTD	gas temperature at the compressor outlet
SM 1231 RTD	IW84 (ch2)	AI, Pt100, 4-wire	T_eng	RTD	drive temperature
…					

For the "Analytical data processing" subsystem, the representation from the point of view of the persistent data model (CC-7) is defined at the level of the entire subsystem (Fig. 17), which is permitted by the method. The representation is common to the "Data aggregation component" and the "Regular GPU condition assessment component."

The PLC program is deployed on a Siemens S7-1200 controller with a CPU 1215C and expansion modules (SM 1231 RTD, SM 1231 AI, SM 1223, SM 1221). Table 12 shows a fragment of the I/O catalog – one signal from each module and each signal type – illustrating the hardware configuration and its binding to the CC-6 software tags.

At the detailed scale level, a representation from the perspective of the Procedure viewpoint (D-1) was built for the behavior "Decommission GPU – Maintenance" of the "Web application component" of the "Equipment condition and maintenance registry" subsystem. This representation is shown in Fig. 18.

To detail the calculations carried out within the processing "Conducting GPU efficiency assessment" by the "Regular GPU condition assessment component" of the "Analytical data processing system" subsystem, a representation from the perspective of the Calculation viewpoint (D-2) was built.

The calculation viewpoint for the processing "conducting GPU efficiency assessment" describes the mathematics of calculating daily metrics, historical baseline values, and trend deltas published in the "daily report" message.

For GPU g and day d, the processing reads the following daily average aggregated values from the aggregation table (Table 13).

Table 13

Description of variables within the D-2 representation for the "Regular GPU condition assessment component" of the "Analytical data processing system" subsystem

Variable	Description
$G_{g,d}$	average gas consumption
$Q_{g,d}$	average volumetric capacity
$T_{g,d}^{eng}$	average gas turbine engine (GTE) temperature
$P_{g,d}^{oil}$	average oil pressure in the GTE
$T_{g,d}^{in}$, $T_{g,d}^{out}$	average inlet / outlet gas temperature
$P_{g,d}^{in}$, $P_{g,d}^{out}$	average inlet / outlet gas pressure

Gas capacity ratio: fuel burned per unit of pumped gas. A higher value means lower efficiency

$$R_{g,d}^{GP} = G_{g,d} / Q_{g,d.}.$$

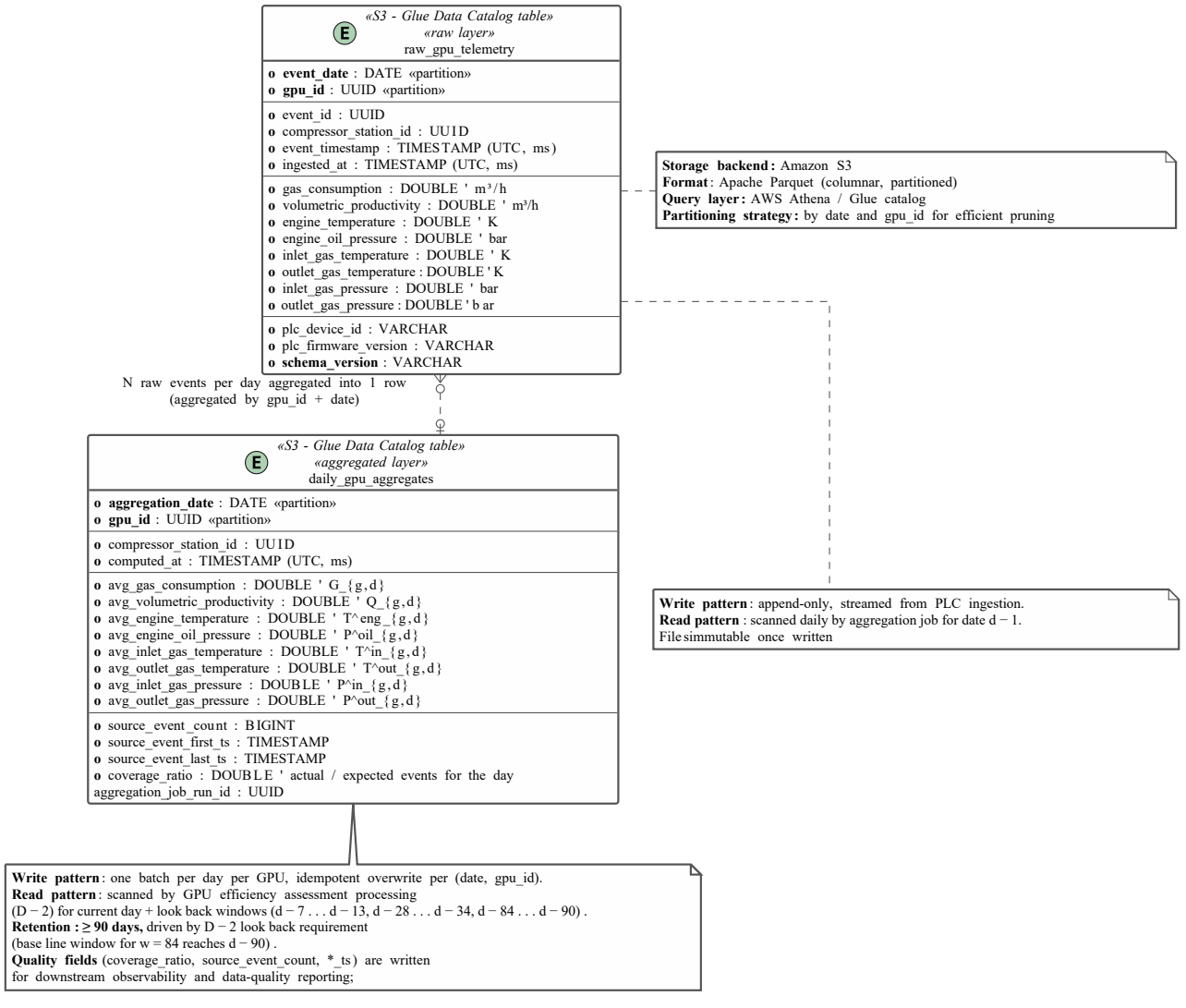

Fig. 17. Persistent data model for the components of the "Analytical data processing" subsystem (CC-7)

Ratio of gas turbine engine (GTE) temperature to oil pressure: thermal load relative to lubrication pressure. An increase in this value may indicate degradation of bearings or the cooling system

$$R^{TP}_{g,d} = T^{eng}_{g,d} / P^{oil}_{g,d}.$$

Thermodynamic compression ratio: deviation of the gas state at the compressor from the expected thermodynamic profile

$$R^{TD}_{g,d} = \left(T^{in}_{g,d} \cdot P^{in}_{g,d}\right) / \left(T^{out}_{g,d} \cdot P^{out}_{g,d}\right).$$

For each metric, a 7-day arithmetic mean preceding the lookback offset w is calculated

$$\bar{R}^{m,w}_{g,d} = (1/7) \cdot \Sigma_{i=0..6} R^{m}_{g,\,d-w-i},$$

where:

– $m \in$ {GP, TP, TD} – metric type;

– $w \in$ {7, 28, 84} days – lookback offsets (1 / 4 / 12 weeks).

For each metric m and each lookback offset w, the dimensionless relative difference between the current value and the historical baseline is calculated:

$$\Delta^{m,w}_{g,d} = \left(R^{m,w}_{g,d} - \bar{R}^{m,w}_{g,d}\right) / \bar{R}^{m,w}_{g,d}.$$

Interpretation for UI: $\Delta = +0.12$ means the current value is 12% higher than the historical baseline; $\Delta = -0.05$ – 5% lower. On the UI, values can be multiplied by 100 for percentage display, but they are stored in the database as a fraction.

As part of constructing the structure saved to daily_gpu_state_estimation, one row is created for each metric m (three rows per GPU per day) – Table 14.

Table 14

Calculation of time trends for periods

Column	Formula
current_value	$R^{m}_{g,d}$
comp_week_ago_trend	$\Delta^{m,7}_{g,d}$
comp_month_ago_trend	$\Delta^{m,28}_{g,d}$
comp_three_months_ago_trend	$\Delta^{m,84}_{g,d}$

After calculating and saving all metrics, the processing publishes a message to the queue with the structure:

$$\text{Report}_{g,d} = \left\{ g, d, \left(R^{m}_{g,d}, \Delta^{m,7}_{g,d}, \Delta^{m,28}_{g,d}, \Delta^{m,84}_{g,d} \right)_{m \in \{GP,TP,TD\}} \right\}.$$

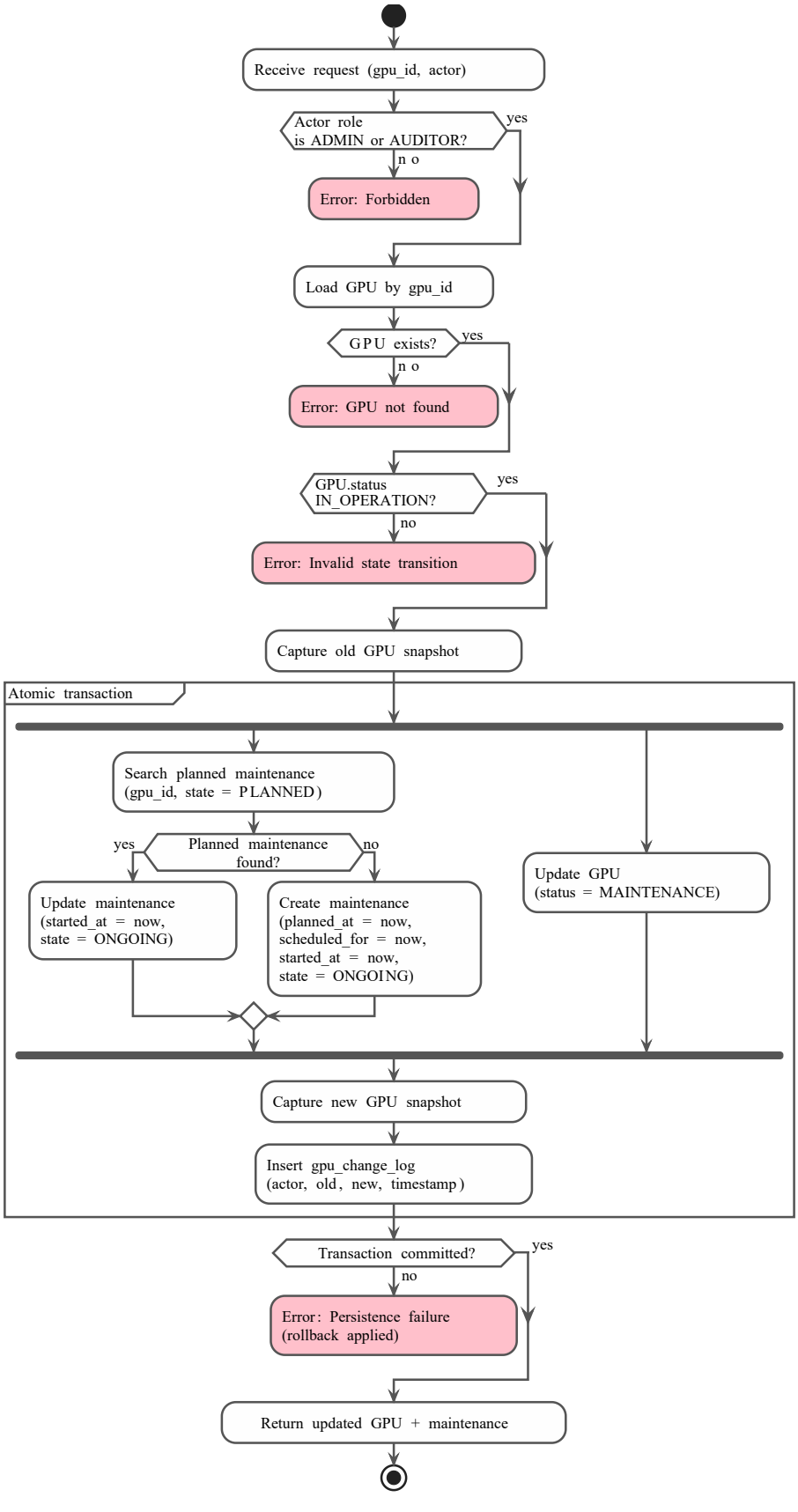

Fig. 18. Details of the logic of the behavior "Decommission gas pumping unit — Maintenance" for the "Web application" component of the "Equipment condition and maintenance registry" subsystem (D-1)

To detail the data transformation logic within the processing "Daily data aggregation" by the "Data aggregation component" of the "Analytical data processing" subsystem, a representation from the perspective of the Task / Transformation viewpoint (D-3) was built (Fig. 19).

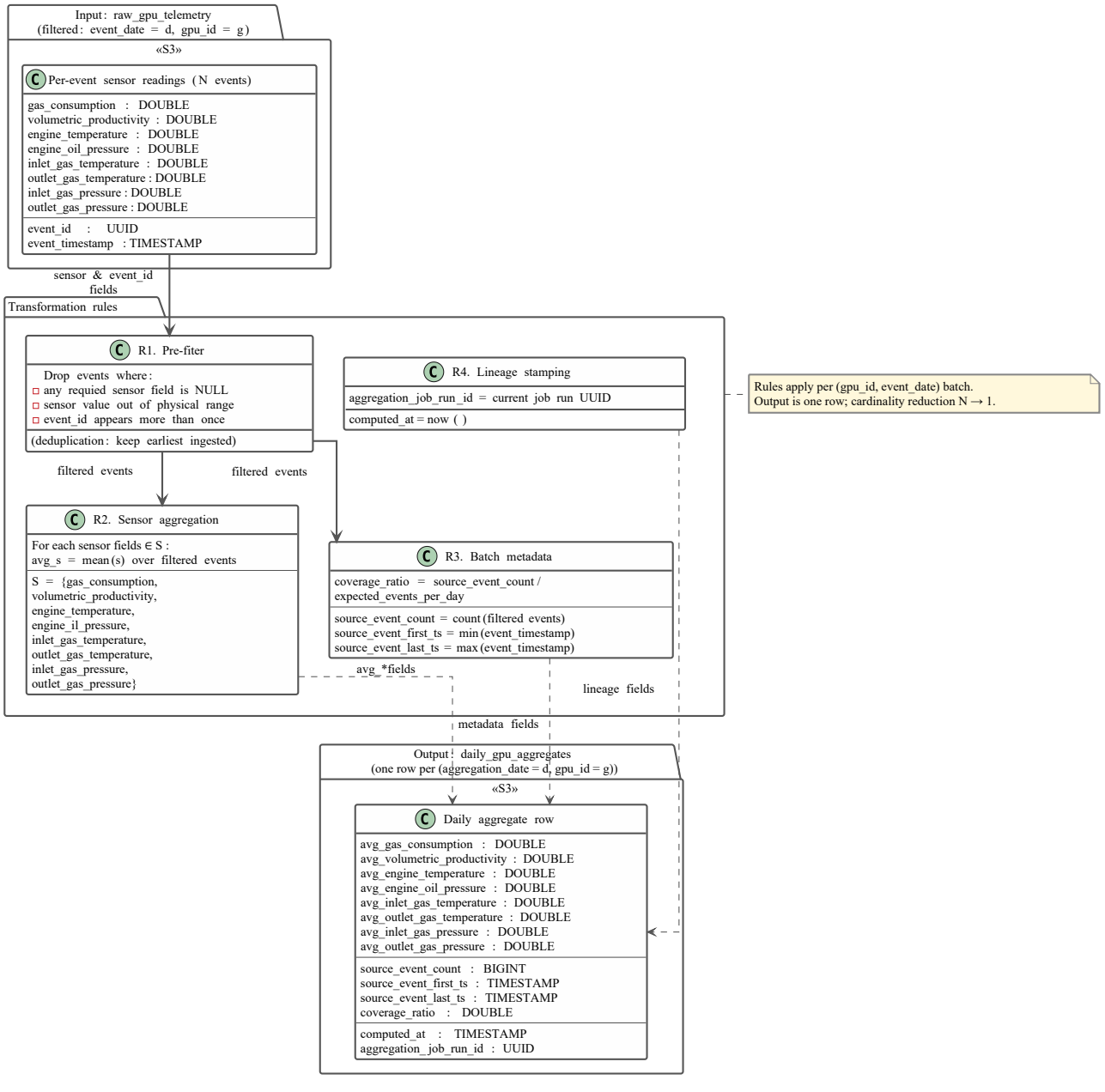

Fig. 19. Representation of data transformation rules for the "Daily data aggregation" processing of the "Data aggregation component" of the "Analytical data processing" subsystem (D-3)

The presented architectural viewpoints of the detailed scale describe aspects of implementation details. Representations of this scale are necessary only for those behaviors, processings, or stages for which low-level aspects are architecturally significant. That is, where at the architectural design level, to achieve architecturally significant requirements, it is necessary to specify the details of logic, calculations, or data transformations.

5. 4. Evaluation of completeness and effectiveness of covering system architectural aspects

A comparison of the STCM with other methods, approaches, and models is provided in [3], where comparative metrics are defined and a quantitative evaluation is conducted. In particular, the quantitative evaluation is carried out based on the following metrics: index of process prescriptiveness, completeness of architectural decomposition, adaptability to the heterogeneous nature of systems, interoperability of artifacts, and readiness for automation. These metrics relate to the method itself.

In this study, the feasibility of applying the method to a heterogeneous industrial automation system is important. For this purpose, it is crucial to perform an additional analysis regarding the coverage of diverse aspects within the architectural design process according to the method. To systematize this analysis, fundamental architectural aspects required for the comprehensive design of automation systems were mapped to the corresponding applied architectural viewpoints in Table 15.

Based on the obtained results of constructing the architectural model, a general characteristic of the method's applicability for real-world projects in the field of industrial automation was formed (Table 16). This format allows for an objective reflection of both the strengths of the approach and its operational costs.

Table 15

Coverage of fundamental architectural aspects of the system

Aspect	Description	Coverage
Input data for the design stage (architectural drivers)	Input data that serves as the foundation for starting the design process	Covered in the requirements and constraints viewpoints: I-1, I-2
Aspects of system logic and essence	Aspects describing the tasks and logic that the system must perform: this includes descriptions of control tasks, data transformation rules, system functions, etc.	High-level system capabilities are covered in SA-1; SSA-1 describes control models and rules; CA-1 and CA-2 describe aspects of operational/processing logic of systems; D-1, D-2, D-3 describe low-level implementation details of logic
Structural aspects	Aspects of decomposing the system into structural elements: subsystems, logical and software components, internal structure	SC-1 defines decomposition into subsystems and their types; SSA-3 / SSA-3-ext describe the logical structure of subsystem components; SSC-1 describes software components; CC-3 describes the internal modular structure of components
Technological stack and execution environment	Aspects of technological implementation, infrastructure, and deployment	SSC-1 captures decisions regarding the technological stack of components at the application and software infrastructure levels; XC-2 describes system infrastructure at the platform and hardware levels; XC-3 describes deployment and update strategy
Runtime execution aspects (execution, states)	Aspects of runtime operation of components and their discrete operating modes	CC-1 describes the execution of tasks and organization blocks of the PLC program implementing control; CC-2 describes discrete states of the component and transitions between them
Object of manipulation (domain model, signals)	Aspects of data and signals manipulated by the system	CC-6 describes domain data models (including the PLC tag catalog); CC-7 describes persistent data models with which system components interact; CC-8 describes I/O interfaces with which the system component – the PLC program – interacts
Integration aspects	Aspects of inter-component and inter-system interaction. Interconnection is one of the key aspects of Industry 4.0	CA-2 describes cross-behavioral interaction dynamics; CA-1-ext describes the behavior catalog with design decisions regarding access methods and protocols; CC-4 describes contracts and API specifications; CC-5 describes outgoing integrations
Human interaction	Aspects of user interaction. Highlighted as a separate aspect, as human-centricity is one of the key aspects of Industry 5.0	XA-1 describes UX flows and UI designs
Additionally: support for evolutionary component and governance	Aspects of supporting architecture evolution and artifact management	Architecture decision records capture architectural decisions; Governance documents describe project-specific standardization of representations

Table 16

Characteristics of the results of applying the method

Characteristic / criterion	Evaluation
Completeness of coverage of architectural aspects	Excellent – end-to-end coverage of fundamental architectural aspects (Table 15)
Presence of components that cannot be designed within this method (exceptions)	Not found – within the investigated gas transportation automation system, no elements were identified that go beyond the method or paradigm
Readiness of the model for transfer to development	Complete – the set of designed artifacts contains all necessary information (at different scale levels) for the direct development and implementation of the system
Labor-intensiveness of manual documentation generation	High – involves creating a significant number of detailed diagrams and specifications
Complexity of keeping artifacts up to date	High – requires effort for constant maintenance of artifacts in an up-to-date state, given changes in input data to the design stage (requirements, constraints)
Potential for AI automation	High–the structured nature of the method is ideally suited for automating the design process using AI agents

As can be seen, applying this method to system design allows for building a complete architectural model that covers fundamental architectural aspects. At the same time, the construction of such a model is done within a structured process that ensures process standardization while, thanks to its system-type-centric orientation, providing the necessary adaptability to the varying architectural nature of systems.

However, there are also disadvantages: forming a full package of architectural documentation for a system is a very labor-intensive process, especially at the stages of component-scale design and detailing, since a set of representations from the respective viewpoints is built for each of the custom application-level components or platform-like off-the-shelf self-hosted and externally integrated subsystems and components. This is partially resolved by the optionality of representations, which involves building only relevant representations from the set of applicable (based on the subsystem type) component-scale level representations. In addition, component-scale level representations are formed based on system and subsystem-scale level representations, as well as a set of governance documents describing project-specific rules for constructing representations from the correspond-

ing viewpoints. Such representations belong to the level of low-level design. It is completely permissible and recommended to form representations of this scale level with the partial assistance of artificial intelligence tools, providing a set of higher-scale level representations as part of the instructions. This can solve the problem of the labor-intensiveness of building a large number of representations, especially in aspects formed according to certain clear rules and requiring repetitive manual work. For example, forming component API specifications based on component behaviors. The investigation of aspects of applying this method in combination with an AI assistant is beyond the scope of this study. Furthermore, the use of basic template documents for representations built on the basis of governance documents can also accelerate the production of identical representations for large systems. However, it is important to note that decisions formed within representations of all scales are necessary for the correct implementation and execution of the architecture. These decisions must be made for each of the projects. Traditional approaches do not place enough emphasis on the standardization of the design process itself within which such decisions are made. Nor do they define a standardized set of representations that would be adaptive to the fundamental nature of the system and fully cover the necessary aspects. This leads to making such decisions implicitly, which can have a negative impact on the quality of the system architecture in terms of maintainability and traceability.

6. Discussion of the results of automation system design

The architectural design of the gas transportation automation system at the CS level, proposed within the scope of this study, was carried out by the consistent application of the STCM of architectural design [3], which is based on the STCP [2]. This ensured both the methodical formation of architectural design artifacts and their interoperability and traceability at all scale levels.

The obtained result is explained by the consistent application of architectural viewpoints defined in the architectural documentation model [3] in accordance with the fundamental nature of each of the subsystems. Tables 1–3 and Fig. 1 show the results of architectural design at the system scale level, within which the system's capabilities were determined and the system was divided into subsystems. Subsystems from the resulting list (Fig. 1) were classified according to the STCP [2], and architectural viewpoints applicable to these subsystems were determined according to the STCM [3]. Based on the determined applicability of viewpoints, abstract and concrete design was performed for each subsystem at the scale level of subsystems, components, and the level of detailing, the results of which are presented in Fig. 2–19 and in Tables 4–14. The designed architecture, described by representing the system from a set of relevant architectural viewpoints, is coherent and provides end-to-end tracing between representations. This is due to the fact that, according to the design method [3], building representations for different viewpoints is performed within the architectural design process. The process involves the consistent construction of representations in a clear order and with dependencies between them. Based on the conducted architectural design process and its results – a set of architectural artifacts – an evaluation of the feasibility and effectiveness of applying the method was carried out, the results of which are presented in Tables 15, 16.

From the obtained result, it is evident that this method can be effectively applied to creation of a complete and relevant multi-aspect architectural model with interoperable and traceable artifacts for a complex heterogeneous industrial automation system. The architectural description of the system is formed within a clearly defined prescriptive architectural design process, which involves the consistent creation of artifacts describing the system architecture from a set of relevant viewpoints, while ensuring the traceability of artifacts and tracking. This distinguishes the result of this study from ad-hoc design practices, in which architectural decisions are formed only for certain aspects and are not standardized relative to the structure of their representation. Based on the obtained result, it can be concluded that this method should be used for designing complex heterogeneous automation systems where prescriptiveness and structural clarity of the process, completeness of the architectural description, and interoperability of design artifacts are important.

The main distinctive features of the obtained results lie in:

– applying the STCM to the design of a heterogeneous hybrid system covering levels from the field level of the CS to the management and analytics levels based on cloud infrastructure and including subsystems of basic types of CPS in the field of automation [2] – Continuous Control, Behavior-oriented, Data-flow-oriented;

– forming a complete, relevant, and interoperable set of architectural artifacts that are linked to each other through explicit viewpoint dependencies defined in the architectural documentation model [3].

The practical significance of the obtained results lies in building a reference architectural model of a gas transportation automation system, which is characterized by the completeness of the architectural description covering a set of fundamental architectural aspects, and the interoperability and traceability of architectural artifacts. The architectural representations built within the study can serve as a reference base for designing CS automation systems. In addition, the results of the study can serve as a guideline for applying the method to any other heterogeneous automation systems [35, 36], including industrial automation, communications [37], robotics, and IT systems for business process automation and data processing [38].

The limitations of the study are related to the fact that a simplified GPU control model was used in the study, which does not cover all the details of a real technological process. Such details include a non-linear model of a two-stage compressor, anti-surge protection via a recycle valve [7], and cross-coupling between pressure and temperature channels. Detailings and technological solutions of the scale of real GPU control systems in the context of these aspects require a separate study and more detailed architectural development, but go beyond the demonstration application of the method. Despite this, complicating the GPU model does not change the way the method is applied to system design – the impact of such a complication will only be reflected in the content of representations from the corresponding associated viewpoints.

Among the shortcomings of this study, it is worth noting the high labor-intensiveness of manually forming a full package of architectural documentation according to the STCM of architectural design. This shortcoming can be eliminated in the future by integrating the method into automated architectural design tools, including AI-assisted ones.

Prospects for further study lie in the following directions:

– investigating the aspects of implementing the method to a wide range of diverse CPS, which would allow for creating a catalog of reference architectures designed based on the application of the STCM [3];

– applying the architectural design method to build template solutions that could be integrated as elements of composite architectures (thanks to the property of design artifact interoperability) of automated control systems at various levels, for example, for simulation models based on "Digital Twins" [39–41];

– integrating the results of this study with automated AI-assisted architectural design systems. The STCM forms a structured set of instructions for AI agents, which makes it possible to automate the construction of the architecture of similar systems based on requirements as input data for the system. This is especially relevant in connection with the accelerated development of large language models and agent systems for engineering design tasks.

7. Conclusions

1. Architectural design at the system scale level was performed using architectural viewpoints of input specifications, system capability representation, and subsystem representation. A list of system capabilities grouped into functional groups was formed, and the system was decomposed into subsystems, with the classification of each subsystem in accordance with the STCP. This provided a holistic view of the system as a heterogeneous CPS combining subsystems of basic types.

2. A set of architectural viewpoints was determined for each subsystem according to its architectural type based on the applicability matrix defined within the STCM of architectural design. This ensured full coverage of architecturally significant aspects of each subsystem.

3. Architectural representations were built at the subsystem and component levels based on the defined viewpoints. The architectural representations are linked to each other through explicit dependencies defined in the architectural documentation model, which provides a holistic view of the system architecture and demonstrated the interoperability and traceability of artifacts. The designed reference architecture forms a holistic representation of a hybrid heterogeneous system combining the on-premises infrastructure of the CS and the cloud infrastructure based on AWS Cloud.

4. The applicability of the system-type-centric approach to designing complex heterogeneous automation systems was evaluated based on the analysis of the completeness and effectiveness of covering system architectural aspects in the designed architectural model. Based on the evaluation results, it was determined that applying this method to system design allows for building a complete architectural model covering fundamental architectural aspects.

Conflict of interest

The authors declare that they have no conflict of interest related to this study, whether financial, personal, authorship, or otherwise, that could influence the study and its results presented in this article.

Financing

The study was conducted without financial support.

Data availability

The manuscript has no associated data.

Use of artificial intelligence

The authors confirm that they did not use artificial intelligence technologies during the creation of the submitted article. Within the scope of the study, the agent-based AI tool Claude Design (Anthropic PBC, USA) was used to build UI designs based on system requirements and the system capability representation.

Authors' contributions

Ihor Polataiko: Conceptualization; Investigation; Methodology; Writing – Original Draft; Writing – review & editing; **Leonid Zamikhovskyi:** Supervision; Project administration; Writing – review & editing.

References

1. Williams, T. J. (1990). A Reference Model for Computer Integrated Manufacturing from the Viewpoint of Industrial Automation. IFAC Proceedings Volumes, 23 (8), 281–291. https://doi.org/10.1016/s1474-6670(17)51748-6
2. Polataiko, I., Zamikhovskyi, L. (2026). Development of system-type-centric paradigm of computer-software systems architectural design for automation systems. Technology Audit and Production Reserves, 1 (2 (87)), 43–56. https://doi.org/10.15587/2706-5448.2026.349943
3. Polataiko, I., Zamikhovskyi, L. (2026). Development of system-type-centric method for architectural design of computer-software automation systems. Eastern-European Journal of Enterprise Technologies, 2 (2 (140)), 65–84. https://doi.org/10.15587/1729-4061.2026.359147
4. Gorbijchuk, M. I., Lazoriv, O. T., Lazoriv, A. M. (2020). The computer system for optimal control of natural gas pumping units. Methods and Devices of Quality Control, 2 (45), 90–101. https://doi.org/10.31471/1993-9981-2020-2(45)-90-101
5. Damirova, J., Salmanov, V. (2026). Control of compressor and pumping systems in the oil and gas industry using SCADA. ETM Equipment Technologies Materials, 217. https://doi.org/10.36962/etm33022026-25
6. Zamikhovskyi, L., Nykolaychuk, M., Levytskyi, I. (2025). Extending the functionality of topologies of Web-oriented control systems for technological objects based on "Open User Communication". Eastern-European Journal of Enterprise Technologies, 6 (2 (138)), 94–115. https://doi.org/10.15587/1729-4061.2025.348728

7. Zamikhovskyi, L., Zamikhovska, O., Ivanyuk, N., Mirzoieva, O., Nykolaychuk, M. (2025). Development of an anti-surge protection system for gas pumping units based on hardware and software vibration monitoring tools. Eastern-European Journal of Enterprise Technologies, 4 (2 (136)), 117–132. https://doi.org/10.15587/1729-4061.2025.337736
8. Zamikhovskyi, L., Nykolaychuk, M., Levytskyi, I. (2025). Development of a simulation model of a WEB-oriented servo drive frequency control system based on "Digital Twins" technology. Technology Audit and Production Reserves, 6 (2 (86)), 76–90. https://doi.org/10.15587/2706-5448.2025.345825
9. Nykolaychuk, M., Zamikhovskyi, L., Levitskyi, I., Kopei, V., Ropyak, L. (2026). Development of a Simulation Model of a PID Controller Based on Simatic S7 Hardware-Software Tools and "Digital Twin" Technology. Automation, 7 (3), 74. https://doi.org/10.3390/automation7030074
10. IEC 62264-1:2013. Enterprise-control system integration - Part 1: Models and terminology. International Electrotechnical Commission. Available at: https://webstore.iec.ch/en/publication/6675
11. The Industrial Internet Reference Architecture (IIRA), Version 1.10: An Industry IoT Consortium foundational document (2022). Boston: Industry IoT Consortium. Available at: https://www.iiconsortium.org/wp-content/uploads/sites/2/2022/11/IIRA-v1.10.pdf
12. Megow, J. (2020). Reference architecture models for Industry 4.0, smart manufacturing and IoT: An introduction. Berlin: Begleitforschung PAiCE; IIT – Institut für Innovation und Technik in der VDI / VDE Innovation + Technik GmbH. Available at: https://www.digitale-technologien.de/DT/Redaktion/DE/Downloads/Publikation/PAiCE_Leitfaden_Reference_Architecture.pdf?__blob=publicationFile&v=1
13. Villafuerte, C., Moncayo, M., Oñate, W. (2026). RAMI 4.0 Architecture for Industrial Traceability with Artificial Intelligence and Integrated Security. Automation, 7 (3), 72. https://doi.org/10.3390/automation7030072
14. Folgado, F., Calderón, D., González, I., Calderón, A. (2024). Review of Industry 4.0 from the Perspective of Automation and Supervision Systems: Definitions, Architectures and Recent Trends. Electronics, 13 (4), 782. https://doi.org/10.3390/electronics13040782
15. Starke, G., Simons, M., Zörner, S., Müller, R. D., Losch, H. (2019). arc42 by Example: Software architecture documentation in practice. Birmingham: Packt Publishing.
16. Brown, S. (2015). The C4 model for visualising software architecture. Leanpub. Available at: https://leanpub.com/visualising-software-architecture
17. Kruchten, P. B. (1995). The 4+1 View Model of architecture. IEEE Software, 12 (6), 42–50. https://doi.org/10.1109/52.469759
18. Bass, L., Clements, P., Kazman, R. (2021). Software Architecture in Practice. Addison-Wesley Professional.
19. Cervantes, H., Kazman, R. (2016). Designing Software Architectures: A Practical Approach. Boston: Addison-Wesley Professional. Available at: https://repo.darmajaya.ac.id/3948/1/Designing%20Software%20Architectures_%20A%20Practical%20Approach%20%28%20PDFDrive%20%29.pdf
20. Rozanski, N., Woods, E. (2011). Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. Boston: Addison-Wesley. Available at: https://ptgmedia.pearsoncmg.com/images/9780321718334/samplepages/032171833X.pdf
21. Calderón, D., Folgado, F. J., González, I., Calderón, A. J. (2024). Implementation and Experimental Application of Industrial IoT Architecture Using Automation and IoT Hardware/Software. Sensors, 24 (24), 8074. https://doi.org/10.3390/s24248074
22. Villar, E., Martín Toral, I., Calvo, I., Barambones, O., Fernández-Bustamante, P. (2024). Architectures for Industrial AIoT Applications. Sensors, 24 (15), 4929. https://doi.org/10.3390/s24154929
23. Gourisetti, S. N. G., Bhadra, S., Sebastian-Cardenas, D. J., Touhiduzzaman, M., Ahmed, O. (2023). A Theoretical Open Architecture Framework and Technology Stack for Digital Twins in Energy Sector Applications. Energies, 16 (13), 4853. https://doi.org/10.3390/en16134853
24. Lamm, J. G., Weilkiens, T. (2013). Method for Deriving Functional Architectures from Use Cases. Systems Engineering, 17 (2), 225–236. https://doi.org/10.1002/sys.21265
25. System Architecture Framework. (GfSE). Available at: https://saf.gfse.org/
26. Evans, E. (2003). Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley Professional. Available at: https://fabiofumarola.github.io/nosql/readingMaterial/Evans03.pdf
27. Waqas, M., Jamil, M., Khan, A. A. (2024). Hybrid Power System Design and Dynamic Modeling for Enhanced Reliability in Remote Natural Gas Pipeline Control Stations. Energies, 17 (7), 1763. https://doi.org/10.3390/en17071763
28. Nazarenko, I. V., Nikolaychuk, M. Ya., Ferenets, V. D., Sukhanov, D. Ye. (2014). Construction and modeling of unified control systems of actuating mechanisms for objects of gas-transport system. Eastern-European Journal of Enterprise Technologies, 1 (2 (67)), 41–48. https://doi.org/10.15587/1729-4061.2014.21204
29. Waqas, M., Jamil, M. (2024). Smart IoT SCADA System for Hybrid Power Monitoring in Remote Natural Gas Pipeline Control Stations. Electronics, 13 (16), 3235. https://doi.org/10.3390/electronics13163235
30. Sayghe, A. (2025). Digital Twin-Driven Intrusion Detection for Industrial SCADA: A Cyber-Physical Case Study. Sensors, 25 (16), 4963. https://doi.org/10.3390/s25164963
31. Zamikhovskyi, L., Nykolaychuk, M., Levytskyi, I. (2026). Method for building a virtual simulator of the drowworks automated control subsystem of a drilling rig with a WEB interface. IOP Conference Series: Earth and Environmental Science, 1630 (1), 12068. https://doi.org/10.1088/1755-1315/1630/1/012068

32. Chen, S., Ebe, F., Morris, J., Lorenz, H., Kondzialka, C., Heilscher, G. (2022). Implementation and Test of an IEC 61850-Based Automation Framework for the Automated Data Model Integration of DES (ADMID) into DSO SCADA. Energies, 15 (4), 1552. https://doi.org/10.3390/en15041552
33. Cirani, S., Ferrari, G., Mancin, M., Picone, M. (2018). Virtual Replication of IoT Hubs in the Cloud: A Flexible Approach to Smart Object Management. Journal of Sensor and Actuator Networks, 7 (2), 16. https://doi.org/10.3390/jsan7020016
34. Rosioru, S., Mihai, V., Neghina, M., Craciunean, D., Stamatescu, G. (2022). PROSIM in the Cloud: Remote Automation Training Platform with Virtualized Infrastructure. Applied Sciences, 12 (6), 3038. https://doi.org/10.3390/app12063038
35. Dobaj, J., Riel, A., Macher, G., Egretzberger, M. (2023). Towards DevOps for Cyber-Physical Systems (CPSs): Resilient Self-Adaptive Software for Sustainable Human-Centric Smart CPS Facilitated by Digital Twins. Machines, 11 (10), 973. https://doi.org/10.3390/machines11100973
36. Solomchak, O., Nykolaychuk, M., Solomchak, A. (2024). Modeling and Simulation of Distribution Network with the D-STATCOM and Photovoltaic Power Stations Using MATLAB. 2024 IEEE 5th KhPI Week on Advanced Technology (KhPIWeek), 1–5. https://doi.org/10.1109/khpiweek61434.2024.10878105
37. Petrescu, I., Niculae, E., Vulturescu, V., Dimitrescu, A., Ungureanu, L. M. (2025). Transport and Application Layer Protocols for IoT: Comprehensive Review. Technologies, 13 (12), 583. https://doi.org/10.3390/technologies13120583
38. Christ, L., Milloch, E., Boshoff, M., Hypki, A., Kuhlenkötter, B. (2023). Implementation of Digital Twin and Real Production System to Address Actual and Future Challenges in Assembly Technology. Automation, 4 (4), 345–358. https://doi.org/10.3390/automation4040020
39. Zamikhovskyi L., Nykolaychuk, M., Levytskyi, I. (2026). Development of a simulation model and testing methodology for frequency control systems of an induction drives using Sinamics G220 and Digital Twin technology. Technology Audit and Production Reserves, 3 (2 (89)), 75–90. https://doi.org/10.15587/2706-5448.2026.360983
40. Al-Najari, B., Hen, C. K., Siaw Paw, J. K., Marhoon, A. F. (2025). Dynamic Tuning of PLC-Based Built-In PID Controller Using PSO-MANFIS Hybrid Algorithm via OPC Server. Automation, 6 (4), 83. https://doi.org/10.3390/automation6040083
41. Rahman, M. A., Shahrior, M. F., Iqbal, K., Abushaiba, A. A. (2025). Enabling Intelligent Industrial Automation: A Review of Machine Learning Applications with Digital Twin and Edge AI Integration. Automation, 6 (3), 37. https://doi.org/10.3390/automation6030037