Application of Systems Theory in the Design of System Monitoring Service of Hardware-Software Complex

NATALIA MAMEDOVA[a], TIMOFEY BOLONIN
Basic Department of Digital Economy,
Plekhanov Russian University of Economics,
36, Stremyanny Lane, Moscow, 117997,
RUSSIA

[a]ORCiD: 0000-0002-8934-7363

Abstract: - In this paper, we propose a solution for applying the principles of structural analysis of systems to the design of a monitoring system for a hardware-software warehouse management complex. Taking into account the theoretical and methodological basis of systems theory for the choice of architectural solutions for systems engineering research and in the design of information systems is a non-trivial approach to design because it goes against the established practice of functional design within the boundaries of the market conjuncture of hardware and software architecture means. It is proposed to consider the principles of systems theory application as an approach to science-based information system design through the relationship between systems theory and system monitoring. The study consistently implements the approach to the design of the monitoring system of the hardware-software complex of warehouse management based on the principles of structural analysis of systems. The approach was applied iteratively to all components of the designed system and in describing the order of their interaction in order to find an efficient and resource-saving architectural solution. This allowed me to design the architecture of the system monitoring service. The principles of structural analysis of systems are shown in describing the functionality of the service, implemented for the internal logic of the server. The process and results of this study represent a positive practice of an orderly and informed approach to designing IT solutions without the prevalence of random or opportunistic factors characterizing the external environment of the design process.

Key-Words: - Systems theory, principles of structural analysis of systems, system monitoring, hardware-software complex, IT solution design methodology, system architecture.

Received: May 27, 2024. Revised: November 9, 2024. Accepted: March 7, 2025. Published: May 6, 2025.

1 Introduction

Different types of system theories are integrated into systems science, which includes general systems theory, branch and special systems theories, and systems engineering, [1]. Systems theory is a transdisciplinary field of knowledge that investigates the general principles and patterns that govern systems in different domains. Following, [2] we believe it is correct to apply to systems theory the category of 'transdisciplinarity', which goes beyond the meanings of multidisciplinarity and interdisciplinarity. At the same time, we consider it important to point out that the use of systems theory is possible not only as a philosophy of relations between monodisciplines, [3].

If it is used as an instrumental framework for the development of future-oriented IT solutions, it is appropriate to consider systems theory in the context of the category of 'interdisciplinarity' because there is a casual establishment of relationships between monodisciplines without feedback loops, [4]. And only when a promising IT solution is transformed into a projected IT solution based on systems engineering, it is expressed through transdisciplinary relations between mono-disciplines as part of the branch systems theory.

There are many examples of the application of systems theory to material objects, and it is on these examples that the general theory of systems was formed by its founders, including von Bertalanffy, Weinberg, and Miller. Their studies form the basis of the general theory of systems, [5], [6], [7]. The principles of systems theory application valid for intangible objects were proposed by Jordon, [8]. He labeled them as principles of structural analysis of systems, and their application opened up new avenues for systems engineering research, [9] including in the field of information systems design.

Systems research developing these principles has predominantly focused on creating useful and usable approaches to comparing and interconnecting domains, thus creating an interdisciplinary approach for describing and analyzing large and even complex systems, [10].

In this study, the principles of structural analysis are applied to the design of the service of system monitoring of the hardware and software complex of warehouse management in the example of a multimodal transport and logistics center. At the same time, it is suggested to consider this study as a general approach to design without focusing on the applied tasks of integrating the development result in warehouse logistics organizations.

From an organizational point of view, following, [11] project activity is seen as the material of organizational experience over a structurally stable system. Classifying the designed system in accordance with the regularities described in [11], we note that it functions on the basis of formative regularities that lead to the transition of the system to another quality. In addition, the designed system is in the state of complexion, because at this stage it is a purely mechanical union of elements, between which the processes of interaction have not yet begun. Completion of design and transition to software implementation will mean the transition to the conjugation stage.

Speaking in terms of the monitoring system design, the transdisciplinarity of systems theory application to it is manifested in the description of hardware and software parts of the complex as interrelated abstractions. Another manifestation is the establishment of the rules of system monitoring and interaction of the described abstractions within the boundaries of the general concept of the system boundary of the system boundary of the system theory, [12]. This is the basis for understanding how the components of the monitoring system interact according to the principles described in the [8], and to represent a hardware and software system as a complex system.

The management of complex systems is considered a related field of knowledge, for which systems theory provides the tools of analysis to understand complex systems, allowing the study of their structure, dynamics, and interactions in the context of broad scientific and practical problems.

The concept of a complex system algorithm is presented in the studies of [13], [14], [15]. Generalization of their materials and conclusions allowed us to reasonably call the hardware-software complex a complex system. This is also true in the aspect that the hardware-software complex combines software and hardware parts and constitutes an integral complex of hardware and software. Thus, in this study under the complex system, it is proposed to consider the hardware-software complex, which in the applied aspect functions in the form of access control systems, database systems, technological industry equipment, and complexes for production automation, [16], [17], [18], [19].

Like any automation product (whether it is an automatic or automated system), the hardware and software complex needs monitoring. This function is performed by system monitoring services. From the position of their place in the IT architecture, system monitoring services are IT systems that collect, process, store, and display information about the state of objects.

Traditionally, monitoring is conducted over the processes of object functioning in real-time. This implies the direct involvement of the operator to assess monitoring results and control monitoring events. However, promising solutions in the development of monitoring systems should go beyond this, offering the user solutions that will provide both remote monitoring and prediction of events in the behavior of the object and the environment external to the monitored object. These systems may not be fully realized or widely used, but their potential makes them important for future technology development.

The following artifacts of transdisciplinarity for the application of systems theory were identified in the ongoing research. From general systems theory, the inherited artifact was the systems approach. Since the research is aimed at finding the best IT solution, technical systems were selected among the artifacts of the branch systems theories. This conditioned the choice of the special systems theories artifact, as it could be the one that specifies the branch systems theory. The artifact was the principles of structural analysis of systems. Finally, systems engineering, as the most applied branch of systems theory, has an explicit material embodiment. The methodology of system monitoring has become an artifact.

The main objective of the study is to design a monitoring system that ensures the stability, availability, and performance of the warehouse management hardware and software system. The application of the systems approach in accordance with the systems theory for design means the development of a solution that implements the stated functionality in full, ensures the optimization of resources for design, ensures quality by applying the principles of structural analysis of systems to select

the architecture of the system monitoring service. The subject area of the study covers approaches to designing systems, in this case, a monitoring system, based on the concept of systems theory, applying its principles and practices to justify the architectural solution.

2 Relationship between Systems Theory and System Monitoring

System monitoring occupies an important place within systems theory as it is a methodology and practice aimed at observing, analyzing and managing complex systems. This enables the identification of problems, the optimization of processes, and the effective functioning of systems in various domains. We emphasize the significance of the relationship between systems theory and systems monitoring in fields such as economics, logistics, information technology, and management.

As already mentioned, systems theory provides tools to understand complex systems. It is suggested that system monitoring is an effective tool for managing a complex system as a holistic object and for observing the environment of complex systems. Let us provide a justification for this assertion. According to the studies of scientists who have made a significant contribution to the development of systems theory, [1], [5], [6], [7], [8] in general terms, the system has the following characteristics - integrity, harmony, variability, and efficiency. Correlating the characteristics of the system with the characteristics of system monitoring, it should be noted the following.

System monitoring covers not only the individual components of the system but also the relationship between the components and their interaction with the external environment. In the context of general systems theory, this characteristic is identified as integrity and is investigated through the application of a systems approach.

System monitoring provides information for feedback, which is a key element in the management of dynamic systems. It ensures the harmony of the system and becomes the starting point for maintaining a balanced state of the system and defining the limits of its adaptability to changes in the external environment.

System monitoring provides data on the basis of which informed decisions can be made to adjust management processes. The resulting changes are possible due to the characterization of the variability of systems, while the assessment and direction of such changes are outside the scope of the variability characterization, as they relate to the characterization of the effectiveness of systems.

System monitoring provides a collection of data on the state of the system and its components, which enables performance analysis and identification of deviations from the parameters of effective functioning. The purpose of system monitoring services is to ensure the effective functioning of the system as a whole (system components, links between components), as well as effective management of system changes, and collection and processing of feedback. This suggests that the efficiency characteristic is the cornerstone of the list and describes the target state of the system as a whole.

The presented content of system monitoring in the relationship with the properties of systems can be called abstract, because it appeals to the provisions of the theory of systems about the universal regularities of structural transformation of systems, [11]. At the same time, a certain level of abstraction is inevitable because system monitoring inherits the principles of control in biological and technical systems, [20], Being a technical tool, designed by a human being and operated by a human being.

And yet, if we try to specify the mechanism of manifestation of system characteristics from the point of view of systems theory through the functions of system monitoring, we can identify the following dependencies.

In information technology, system monitoring is used for continuous observation of the state of servers, networks, and applications. Monitoring of servers and networks allows for rapid identification and troubleshooting of problems, ensuring stable operation of the IT infrastructure. System monitoring involves collecting metrics such as CPU utilization, memory usage, and response time. Performance analysis on this data helps identify system performance bottlenecks and optimize system performance. Request tracing also helps to identify performance issues and identify areas that need to be optimized: the trace method traces the path of requests through various system components. For hardware and software systems, system monitoring is used to diagnose the state of the hardware. Condition monitoring involves determining the current condition, predicting possible faults, and planning maintenance based on the actual condition. In project management, system monitoring allows you to track tasks according to plan, evaluate results, and adapt strategies to current conditions. These activities are enabled in system monitoring by the built-in program and project

evaluation functionality.

Thus, system monitoring is used in the practical application of systems theory theses to ensure effective management, analysis, and optimization of various systems. The representation of the relationship between system monitoring and systems theory in abstract and applied aspects can be fully transposed to the hardware and software system. As it has been shown, considering it as a complex system it is reasonable to proceed from the thesis of systems theory to design the architecture and formulate functional requirements, taking into account its purpose as a system. And, designating system monitoring as a tool for analyzing and understanding complex systems, we specify the service task of designing a monitoring system to ensure the effective functioning of the warehouse management hardware and software complex.

3 Approach to the Design of a Monitoring System based on the Principles of Structural Analysis of Systems

The principles of structural analysis of systems are formulated as follows, [8]:

Principle 1 - the more specialized or complex a system is, the less adaptable it is to changing conditions.

Principle 2 - the larger the system, the more resources are required to support it, and the increase is not linear but non-linear.

Principle 3 - systems often contain other systems or are themselves components of larger systems.

Principle 4 - systems grow over time, both in terms of size and structural complexity.

Their application at the design stage of the system monitoring service of the hardware and software warehouse management complex allowed us to justify the choice of architectural solutions, selection logic, and interaction schemes of the system components. In this way, a positive practice of an orderly and conscious approach to designing IT solutions was developed without the prevalence of random or opportunistic factors characterizing the external environment of the development process (technological hype, competition).

The designed system monitoring service contains the following components:

1. A server that receives and processes requests and performs all calculations.
2. A database that stores all the metrics obtained.
3. A query interface to retrieve specific monitoring information.
4. Agents that collect data on users' computers and send it to the server.

This means designing a layered architecture in which the implementation of each of the four layers will be independent of the others, and the exchange of information will be done by communication between the components.

Separating the logic (server and agents) became a necessary solution as the components will be on different physical devices. This is a modern approach to ensuring the information security of the IT infrastructure, [21], [22]. Therefore, the choice between multi-tier and three-tier architectures is driven by the specifics of the service being developed, although this choice reinforces the risks contained in the description of the fourth principle of structural analysis of systems, [8].

Obviously, all components in the system must functionally interact with each other. And the choice of interaction type includes two options - synchronous (blocking) and asynchronous (non-blocking). Keeping in mind that parallel work of system fragments is not a sufficient sign of asynchronousness, let's focus on the fact that during synchronous interaction the server ‘knows’ that the client's state will not change while processing the request. This ensures full synchronization of components, increasing the reliability and fault tolerance of the system.

However, we chose the non-blocking type of synchronization not because it provides better system performance, but because of the multiple functions of the server and agents (besides communication) in the designed monitoring system. Since the agent continuously sends data to the server, choosing a synchronous model, will result in no time to collect metrics on the device.

The second software component created entirely by the author is the server. The server consists of four main modules that implement the server logic. The UML diagram of the component is shown in Figure 1.

Network interaction with external applications and users is based on the TCP/IP model. Data exchange between applications, which are distributed on different nodes of the network, is implemented by the web service through the API programming interface. Among the many patterns for correct API implementation, the most common ones are SOAP API and REST API. SOAP is well-suited for complex and structured operations. The protocol also offers state persistence about the current state of the user or application on the server. While the advantages are obvious, the choice of SOAP API will increase the risks of non-linear

growth of resources to support the system (Principle 2).

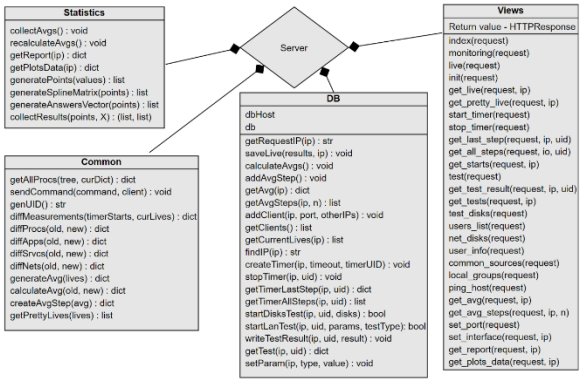

Fig. 1: UML diagram of a server

In addition, a simulation model of using both protocol protocols on the same data showed that the SOAP API protocol is more complex to implement and use, and its XML syntax is redundant for the operations performed (Principle 4). For the REST API protocol, unfortunately, there is no transactional support, and no client state records are stored. And also we have to put up with the necessity of a multi-tier system (principle 4), because the servers are located on different levels, but each server interacts only with the nearest levels and is not connected by requests with others. However, the simplicity of requests (it is not planned to send complex structured requests in the designed service) and high speed of data exchange for increased frequency of metrics updates were the determining factors in deciding that the network interaction of the web server will be based on the principles of RESTful services.

A separate decision was made to organized communication between the agents and the server. Unlike the server, agents exchange information only with the server. Since there is no need to comply with strict specifications, as there is no communication with third-party components, the use of low-level protocols (transport-level) TCP or UDP is justified. For communication at this level sockets based on the transport level protocol are used. The choice of the optimal solution was determined by the functional characteristics of the designed monitoring system - the network connection should be reliable and consistent, i.e. all sent information is guaranteed to reach the recipient in the same order in which it was sent.

In system monitoring, agents send information to a server (which is assumed to have an API), and the server sends commands to the agents (to run tests or change settings) in the form of small strings. The speed of interaction is not as significant for system monitoring as the accuracy and integrity of the data. In this case, the time difference in sending commands (which are sent not so often and have a small volume) will not be noticeable. However, any loss of data is critical. Therefore, the optimal choice is TCP protocol, including because UDP does not guarantee the sending of data and their order, but it guarantees that the packet will either reach completely or not reach at all. It is worth noting that the API call to the server will work using the HTTP protocol, which in turn is based on the same TCP.

All the above-described architectural solutions refer to the central component of the system - the server and ways of its communication with other layers. As for the database architecture, the following requirements were made: a) the database should work as a separate server; b) the database should be able to work with heterogeneous data that does not have a clear structure.

It was decided to organize the work with the database according to CRUD methodology:

1. Create (POST method in REST) - records are created in the database.
2. Read (GET method in REST) - data is read from the database.
3. Update (PUT method in REST) - the data in the database is updated.
4. Delete (DELETE method in REST) - data is deleted from the database.

CRUD is the basis for most modern web applications, it is used in many popular frameworks and programming languages, making it a universal solution for application development. The architecture solution is particularly valuable for the system being designed due to the fact that CRUD operations can be easily integrated into various application architectures, including RESTful APIs. This allows for scalable solutions that can adapt to changing requirements. This fact reduces the complexity of the system (Principle 1). For example, adding new functions or modifying existing operations is done using the CRUD methodology without significant changes to the application architecture.

The last component of the designed monitoring system is the internal work of data collection agents. Six subclasses of the Monitor control class are formed (discussed in detail in the next section).

The UML diagram of the agent classes is presented in Figure 2.

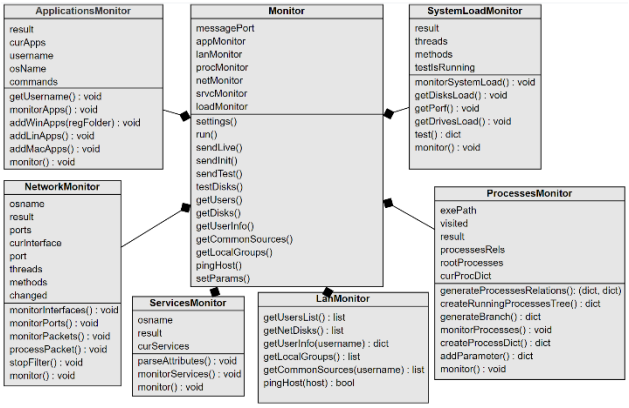

Fig. 2: UML diagram of an agent

For their efficient operation, it is assumed to collect metrics for each subclass in parallel (and to collect different data within the same subclass in parallel). There are three technologies for parallelizing computation: multithreading, multiprocessing, and asynchronous.

Since the agent sends to the server a packet with the state of all the sampling points of the system monitoring parameters at once, it must know about the state of the actual data. To realize such a thing in multiprocessing, processes have to send data to the main process using operating system techniques, which slows down the agent. Therefore, the main choice is between asynchrony and multithreading.

The asynchronous model involves working in a single thread in which the programprogram moves on to the next thread without waiting for the task to complete. This approach is effective in systems with a large number of requests (for example, a server), as it is inefficient to allocate system resources for each new request. However, in small components, asynchrony will work slower than multithreading (especially considering that different threads can be run on different cores), so for agents specifically, the most optimal option is to create a limited number of threads.

Summary, let us highlight the following key ideas of the architecture of the designed service of system monitoring of the hardware and software complex of warehouse management based on the results of applying the principles of structural analysis of systems:

1. The server is built on a multilevel client-server architecture. The component is a RESTful service accessed via HTTP protocol. Communication with agents takes place either via API or TCP connection. The server operation is asynchronous.
2. The database functions as a separate server that processes heterogeneous data. Interaction with the database is performed using CRUD methodology.
3. data collection agents collect metrics in parallel, being able to send all relevant information at any time.

4 Designing the Architecture of the System Monitoring Service

The implementation of data collection agents was carried out in the following logic. The agent algorithm programmatically collects data from the device on which it is located and structures it with processing. The algorithm sends the obtained metrics to the server. Also implemented is an option for the agent to listen to the server over a socket, wait for a command from it, execute, and send a response. The agent also periodically sends messages about ip addresses belonging to the client. This signal is required to initialize the agent on the server and to create a separate collection for it in the database.

The agent has a centralized control class Monitor, which consists of six subclasses. The agent program of the designed system monitoring service implements the operation of six subclasses for data monitoring:

1. ApplicationsMonitor - installed applications.
2. LanMonitor - local network.
3. SystemLoadMonitor - system load.
4. NetworkMonitor - network activity.
5. ProcessesMonitor - running processes.
6. ServicesMonitor - list of services.

All instances of subclasses are stored as Monitor attributes.

The principle of operation of all monitors is similar. Each class has a result attribute, which stores data to be sent to the server for each sample. The classes have a monitor method, which runs in an infinite loop the collector functions that update the result attribute upon completion of their work.

The monitor is the main controlling class of the agent, which is responsible for communicating with the server and managing the monitor classes. At initialization, instances of the previously described classes are created and stored as Monitor's fields. Among other attributes, the class stores its address (self.ip), a list of other addresses (self.ips), server address (self.serverAddr), and socket port (sel.messagePort).

The agent operation contains several parameters that can be configured manually. For this purpose, there is a config.json file consisting of four

parameters:

1. server_address - address of the remote monitoring service.
2. message_port - the socket port through which the agent receives commands from the server.
3. net_interface - parameter for setting the curInterface attribute of the NetworkMonitor class.
4. net_port - parameter to configure the port attribute of the NetworkMonitor class.

Therefore, before starting its work, Monitor reads the current settings written in the file and writes them to the corresponding variables. Then the main method of the class - run.

First, the method passes through all monitors and starts the monitor method in separate threads, starting data collection. Then a thread is created in which the agent continuously sends a 'live' document to the server. Next, a thread is created in which the agent continuously sends a packet with information about itself (ip, ips, messagePort) once every 10 seconds. This is necessary for several reasons. Firstly, initialize the agent on the server (the server learns about the new client and stores its address and port). Thanks to this periodic constant sending, it doesn't matter which component started first, the agent or the server, the latter still learns about the existence of the former after some time. Secondly, if the agent has new addresses or the user has changed the port earlier, the server will notice these changes and update the data in the database.

The sendLive method is responsible for sending 'live' packets. It passes through the result attribute of each monitor class, saves the received data, and sends it to the server. This is the advantage of running separate components in threads because the method does not need to wait for the end of the iteration of any builder, it immediately has access to the data area where the metrics are stored.

After seven threads have been created, the program goes into an infinite loop in which it waits for and processes messages from the server. There are only two types of messages received: running a command and changing attributes of the NetworkMonitor class (curInterface and port).

In general, sending any messages from the agent to the server is done via the HTTP protocol. The server address, including the port on which it is running, is specified in the config.json file (server_address key). As an alternative option, we considered message exchange via socket (the same way the server sends messages to agents). The choice in favor of the first option is justified by the type of packets sent. Messages sent by agents can have unlimited size and are stored in JSON format. Transmitting such messages as strings over a message socket would be problematic for two reasons:

- The server needs to add additional functionality to listen for messages from agents. Since it already processes HTTP requests, the best option is to send data to the port it already occupies, rather than create new processes that will occupy memory.
- Since the data is stored in JSON format, working with HTTP, which is adapted to work with this format, is more convenient. Data via sockets is sent as a binary string, which then needs to be correctly converted to JSON, which is just an additional step in communication between agents and servers.

Therefore, it was decided to send messages by agents by accessing the server API.

If a command comes from the server, the algorithm calls the necessary method of the monitor class for which the command came and sends the result to the server. For this purpose, the sendTest method is written, which, using the urllib3 library, makes 4 attempts to send the result to the /test address of the server. The results of the command are stored in JSON format, which is sent in the POST request. Figure 3 shows the implementation of two functions: testDisks (calls the SystemLoadMonitor class method and sends the result by calling sendTest) and the sendTest function itself.

```
def sendTest(self, result, testUID, attempt=1):
    if attempt == 5:
        return
    try:
        urllib3.request(method="POST", url=f"{self.serverAddr}/test",
                        json={"ip": self.ip, "result": json.dumps(result), "uid": testUID})
    except:
        time.sleep(5)
        self.sendTest(result, testUID, attempt+1)

def testDisks(self, testUID, params):
    res = self.loadMonitor.test(params.split())
    self.sendTest(res, testUID)
```

Fig. 3: Code fragment. Sending the test result to the server (PyCharm)

If the server sends a message about changing the NetworkMonitor parameter, the agent stops the sniff function method, changing the changed value to True, having previously updated the class attributes. Thus, a new packet search will be performed using the already-changed parameters. At the same time, the parameter change is also performed in the config.json file, so that when the agent is restarted, it remembers the last state. The setParam method is responsible for changing parameters. First, it reads the contents of the config.json file, then updates the necessary attributes (port or interface, depending on

the server request) of the NetworkMonitor class, and finally updates the contents of config.json by adding new attribute values. The implementation of the parameter change function is shown in Figure 4.

```
def setParam(self, sType, value):
    with open("config.json") as file:
        data = json.load(file)

    if sType == "PORT":
        self.netMonitor.port = value
        self.netMonitor.changed = True
        data["net_port"] = value
    elif sType == "INTERFACE":
        self.netMonitor.curInterface = value
        self.netMonitor.changed = True
        data["net_interface"] = value

    with open("config.json", "w") as file:
        json.dump(data, file)
```

Fig. 4: Code fragment. Changing filtering parameters (PyCharm)

All messages are processed in separate threads so as not to delay the other packets in the queue.

Defining the stack for the development of monitoring agents at the stage of designing the monitoring system architecture involved only selecting a programming language, since the agents do not use a database or other third-party software. The language selection was done experimentally - prototypes were written in Python, Java, C++ and C#. The main criteria were completeness of data sampling, speed of operation, and cross-platform.

Java and C++ could not collect all the necessary data without unlimited access to external libraries and insufficient inbuilt libraries to collect complete information for each section. Cross-platform problems occurred in C++ and C#, where it is difficult to adapt the program to different platforms while maintaining agent efficiency and code transparency. The optimal solution was Python, which has a wide set of libraries that allow thecollection of information on all sections in full. Python is also extremely convenient for cross-platform development, requiring almost no additional branching in the code for this purpose.

The minus was the speed of collecting data about running processes on Windows. For comparison: the speed of data collection on this parameter on C# is less than a second, on Python - about 30 seconds. That's why we decided to write the whole agent functionality in Python, calling the collected exe file written in C# under Windows. It was taken into account that under Windows the exe runs only in case of process data.

MongoDB was chosen as the database. Since the structure of packages, which can change dynamically, is not known in advance, the choice was made only from NoSQL DBMS. The reasons for choosing MongoDB were the following factors. First of all, it was taken into account that the server is a RESTful service, that is, it communicates using JSON packets. Further, the language of agent development was chosen Python, the main data structure which is a dictionary, which has the same structure as the JSON file. Therefore, a database convenient for working with this format was selected - Redis and MongoDB.

Redis stores data in RAM and requires about 4 GB, which is a heavy load on the system, especially when the stored data increases. Data in key-value storage can only be accessed by key. You cannot query records by multiple attributes, not to mention more complex queries, which is also a disadvantage when choosing. MongoDB, in its turn, does not have the mentioned disadvantages of its analog, but it works slower. At the same time, the DBMS allows you to create indexing by timestamps, which will help to speed up queries. That is why MongoBD was chosen as the database.

Server implementation can be divided into two components: backend and frontend. The choice of frontend implementation language is obvious - JavaScript. The language is the most popular and convenient tool in this industry. For the backend was chosen Python, or rather its framework Django. Python is convenient for cross-platform development, not requiring additional branches in the code. The choice is justified if both the agents and the test system into which the web service is to be integrated are written in Python - for better compatibility it is advisable to develop the server in the same language. The choice seems logical also when developing a cross-platform solution. Experiments with languages when writing agents revealed the problems of adapting the program in C++, and C# for different platforms - it turned out to be impossible to preserve the agent's efficiency and code transparency. Django was chosen among the frameworks, as it is the framework with the greatest functionality and is the main framework for the development of web services.

The last item is to select a web server. Among the options were Nginx and Apache (two of the most popular web servers serving more than 50% of all traffic on the Internet). Both servers have the necessary functionality and flexibility of configuration for the monitor. Therefore, the key criterion was speed of operation. The Apache architecture consists of a kernel and its associated modules: the kernel accepts connections, and the modules correspond to functions executed on request. As the number of requests increases, Apache's centralized architecture causes the web

server to slow down. Nginx was originally designed around asynchronous non-blocking event-driven algorithms. The server creates worker processes, each of which can handle thousands of connections. This approach to connection processing allows Nginx to scale with limited resources.

Because the server is single-threaded and does not create processes for each connection, memory, and CPU usage are relatively uniform, even under high loads. Even with a small number of requests, depending on the type of content (dynamic or static), Nginx is either as good as or better than Apache in terms of performance. Therefore, Nginx will act as the web server on which the monitor will be configured.

Thus, the entire technology stack for each layer of the designed monitoring system has been fully defined.

5 Description of System Monitoring Service Functionality

Before implementing the backend server, the API was compiled and described, including url, available methods, and input and output data. This was done in order to demonstrate what functionality the server performs and what queries it can answer.

Table 1. API Base Keys

№	Team	Request
1	Get live documents	GET/api/v1/get_live/<str:ip>
2	Start timer	POST/api/v1/start_timer
3	Stop timer	POST /api/v1/stop_timer
4	Request the last state of the timer	GET/api/v1/get_timer_last_step/<str:ip>/<str:uid>
5	Request all timer states	GET/api/v1/get_timer_all_steps/<str:ip>/<str:uid>
6	Request timer initializers	GET/api/v1/get_timer_starts/<str:ip>
7	Request all tests	GET/api/v1/get_tests/<str:ip>
8	Query a specific test	GET/api/v1/get_test/<str:ip>/<str:uid>
9	Get metrics averages	GET/api/v1/get_avg/<str:ip>
10	Get the last n documents with average values	GET/api/v1/get_avg_steps/<str:ip>/<int:n>
11	Change network interface	PUT /api/v1/set_interface
12	Change network port	PUT /api/v1/set_port
13	Request a report	GET/api/v1/get_report/<str:ip>
14	Get device evaluation	GET/api/v1/get_rate/<str:ip>

Since the task of designing a RESTful service was performed, according to its specification, the url starts with api/vx/, where x is the API version (at the time of designing the service v1). This will ensure backward compatibility when the API functionality changes. If some value needs to be passed in the URL itself, it is denoted as follows: <type:name>, where type is the type of the variable and name is the name. All APIs return status 200 if the request is correct, otherwise they return error code 40X.

The list with a description of available commands and their corresponding URLs is presented in Table 1.

The following are explanations of command usage for a number of non-obvious or ambiguous command values when using the system monitoring service.

Executing the start timer command (POST /api/v1/start_timer) outputs a string containing the uid of the created timer. Executing the stop timer command (POST /api/v1/stop_timer) sends an appropriate signal if the timer needs to be stopped early or if there is no time limit.

On the request timer initializers command (GET /api/v1/get_timer_starts/<str:ip>), the method is called to get a list of all previously started timers with information about them.

When executing the command to request a specific test (GET /api/v1/get_test/<str:ip>/<str:uid>), if the test has not yet completed execution (the ‘ready’ field is set to Flase), a JSON object {‘status’: False} with response code 400 will be returned.

For the command to get average values of metrics (GET /api/v1/get_avg/<str:ip>) the return value is the document ‘avg’ (JSON Object) in its original form (the metrics are not averaged).

For the command to get the last n documents with average values (GET /api/v1/get_avg_steps/<str:ip>/<int:n>) n is an integer value, the number of documents. If their number is less than n - all documents will be shown.
The change network port command (PUT /api/v1/set_port) uses the method called to change the port attribute of the NetworkMonitor class.

To request a report (GET /api/v1/get_report/<str:ip>), a JSON report object whose structure is specified by the ‘avg’ document serves as the returned document.

For the get device rating command (GET /api/v1/get_rate/<str:ip>), an integer value serves as the returned document.

According to the REST specification, the service exchanges information using the JSON format, so it is necessary to describe the format of input and

output data (keys), each of which is mandatory.

The representation of input and output data provides an understanding of the data structure required to execute a particular command (data types, mandatory and optional fields, data formats, and value constraints).

Table 2 describes the input data parameters. The first column contains the names of the keys, the second column contains their description, and the third column contains the number of commands in which they must be present.

Table 2. Input data for API commands

Input Data	Description	Commands
Ip	IP address of the device you want to retrieve information about (string)	2, 3, 11, 12
Timeout	The number of seconds the timer should run (number). If there is no set time, and the timer should wait for the end signal, the value 0 is specified.	2
Sections	An array of strings specifying the sampling sections for which metrics should be collected for the time interval (Array)	2
	proc - running processes	
	app - installed applications	
	load - system load	
	srvc - list of services	
	net - network activity	
Uid	Timer ID (string)	3
Interface	New interface name (string)	11
Port	New port value (number)	12

Table 3 describes the output data parameters. The structure of the table is similar to the structure of Table 2.

Table 3. Output data for API commands

Output Data	Description	Commands
Data	List of JSON objects with ‘live’ documents for each sampling point (array)	1
Data	Array with last received ‘timer_cur’ documents with specified uid (array)	4
Data	Array with all ‘timer_cur’ documents with the specified uid (array)	5
Data	Array with all ‘timer_start’ documents (array)	6
Data	Array of all ‘test’ documents (array)	7
Status	Status responsible for test readiness (boolean)	8
Result	The result of the test execution. Corresponds to the field ‘result’ of the document ‘test’ (object)	8
Data	Returns an array of the last n documents ‘avg_step’ (array)	10

The system monitoring service provides an interface that allows you to perform single operations on resources in the system without having to maintain state between calls. The APIs of one-time commands are shown in Table 4.

Table 4. APIs of one-time commands

Request	Command
Conduct a speed test of the discs	POST /api/v1/disks_load
Provide a list of users in the domain	POST /api/v1/users_list
List network drives	POST /api/v1/net_disks
Give out the user information	POST /api/v1/user_info
List shares with a user	POST /api/v1/common_sources
List local groups	POST /api/v1/local_groups
Check host availability	POST /api/v1/ping_host

The peculiarity of the API of one-time commands of the projected service is that they include only non-muting commands that read data. All APIs related to ‘test’ documents as a response return a string with the identifier of the created test in case of a correct request. After that, it is necessary to address the system address (API endpoint to which requests are sent to get the results of operations).

6 Server Backend Implementation

This section describes the development of the server's internal logic, how it performs calculations, and communicates with other components.

Five main functional modules can be distinguished in the server structure:

1. urls.py - contains all the main URL paths that the server processes, correlated with handler methods.
2. views.py - contains the implementation of all handler methods.
3. db.py - implements methods of interaction with the database.
4. statistics.py - module responsible for the section with reports.
5. common.py is a module that implements general-purpose functions for packet conversion, identifier generation, etc.

In urls.py, all the necessary objects that bind addresses and methods of views.py are created. The django.urls.path object on the server is responsible for linking the URL to the method it calls. The object's constructor takes two mandatory parameters as input: the path itself and the function that will

handle the request (Figure 5).

```
path('', views.index, name="index"),
path('monitoring', views.monitoring, name='monitoring'),
path('live', views.live, name='live'),
path('init', views.init, name='init'),
path('get_pretty_live/<str:ip>', views.get_pretty_live, name="pretty_live"),
path("api/v1/get_live/<str:ip>", views.get_live, name="get_live"),
path("api/v1/start_timer", views.start_timer, name="start_timer"),
path("api/v1/stop_timer", views.stop_timer, name="stop_timer"),
path("api/v1/get_timer_last_step/<str:ip>/<str:uid>", views.get_last_step, name="gtls"),
path("api/v1/get_timer_all_steps/<str:ip>/<str:uid>", views.get_all_steps, name="gtal"),
path("api/v1/get_timer_starts/<str:ip>", views.get_starts, name="gts"),
path("api/v1/disks_load", views.test_disks, name="test_disks"),
path("test", views.test, name="test"),
path("api/v1/get_test/<str:ip>/<str:uid>", views.get_test_result, name="get_test"),
path("api/v1/get_tests/<str:ip>", views.get_tests, name="get_tests"),
path("api/v1/users_list", views.users_list, name="users_list"),
path("api/v1/net_disks", views.net_disks, name="net_disks"),
path("api/v1/user_info", views.user_info, name="user_info"),
path("api/v1/common_sources", views.common_sources, name="common_sources"),
path("api/v1/local_groups", views.local_groups, name="local_groups"),
path("api/v1/ping_host", views.ping_host, name="ping_host"),
path("api/v1/get_avg/<str:ip>", views.get_avg, name="get_avg"),
path("api/v1/get_avg_steps/<str:ip>/<int:n>", views.get_avg_steps, name="avg_steps"),
path("api/v1/set_interface", views.set_interface, name="set_interface"),
path("api/v1/set_port", views.set_port, name="set_port"),
path("api/v1/get_report/<str:ip>", views.get_report, name="get_report"),
path("get_plots_data/<str:ip>", views.get_plots_data, name="plots_data"),
path("api/v1/get_rate/<str:ip>", views.get_rate, name="get_rate"),
```

Fig. 5: Code fragment. Processed addresses (using PyCharm)

It is accepted that each method in views takes as input at least one mandatory argument - HttpRequest, the request that came to the corresponding address. This object contains the necessary information about the request (method, JSON, address, etc.). The system monitoring service implements the logic that first each method compares the received method with the expected method, if they differ, error 405 (Method not allowed) is returned. A total of two objects returned by the functions are provided:

1. HttpResponse. A normal HTTP response, used if the response does not contain a JSON file, and you only need to send the response code and, if necessary, a string.
2. JsonResponse. A response that is sent if the result contains a JSON object. This also specifies the response code.

We assume that methods that accept methods other than GET have the @csrf_exempt decorator. Normally, when a request is made to a server, it is expected that the packet being sent comes from the current website and not from some other domain (for security). In the designed system monitoring service, to ensure that this happens, the server looks for a CSRF token in the request that would indicate that it is an allowed request. The server must send a unique token to the browser and check whether the browser sends it in response to the request. If the tokens match, the request is valid, if not, it is rejected. Since the work of the system monitoring service involves interaction with external components, it is necessary to allow them to use any methods (that's why all functions check requests for correctness).

Views methods themselves do not perform calculations, they accept a request, call db.py methods and return a response, forming data packets via API. The db.py module is responsible for connecting to the database and sending queries. The database creation form is shown in Figure 6. Let's also add that the server with the database by default is running on localhost:27017.

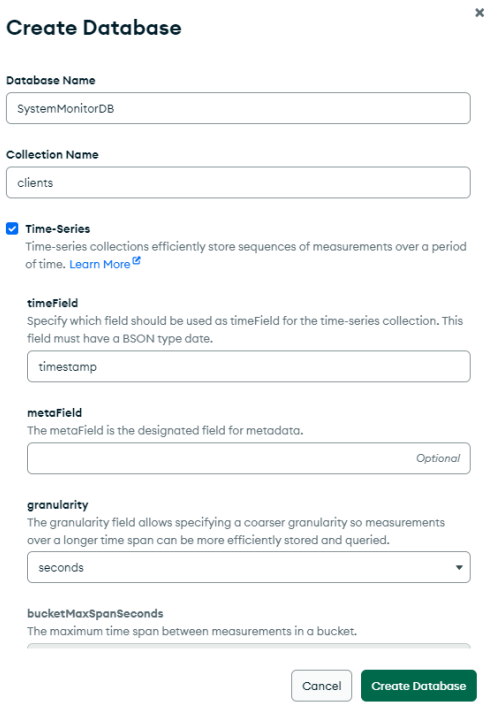

Fig. 6: Creating a database (using MongoDB)

The solution used is to access the db.py module asynchronously, variables are declared globally so that they can be accessed from any area of the system at any time. The connection to the MongoDB server is 'dbHost = pymongo.MongoClient('mongodb://localhost:27017 /')'. Connection to the monitor database (Db is an object whose keys are collection names and values are collections themselves) - 'db = dbHost['SystemMonitorDB']'.

The logic of the service provides that the module functions include searching and creating documents and updating their contents. The main methods of collection are:

1. find_one. Takes as input the dictionary by which the documents in the database are searched and returns the first matching one. For example, in the agent socket port change method, the required document in the client's collection is searched by ip using the command: 'db['clients'].find_one({'ip': ip})'.
2. find. Find and output a list of all documents that

match the filter. It takes the same argument as input. For example, to find all ‘live’ documents, the following command is executed: ‘db[ip].find({’section": “live”})’.

3. update_one. Search for the first matching document and update its fields. For example, when the test has finished its execution, you need to change the value of the result field to True and add a field with the result. This is achieved by the command: ‘db[ip].update_one({‘section’: ‘test‘, “uid”: uid}, {’$set": {‘ready’: True, ‘result’: result}})’. The first argument is the search filter, and the second argument is the fields to be changed and their new values. The $set operator makes the DBMS understand that the old value should be completely replaced with the new one.
4. update_many. The operation scenario is the same as update_one, but all the found documents are changed. Work example: ‘db[ip].update_many({’section‘: “timer_start”, “uid”: uid}, {’$set‘: {’finished‘: True}})’. When the timer finishes, all of its ‘timer_start’ start documents change the value of the ‘finished’ field to True.
5. insert_one. A method that takes only one argument as input - a document (in Python it's a dictionary) that writes to a collection. Example: collection.insert_one(avg_step). When a new copy of the current document ‘avg’ is created, the document ‘avg_step’ is inserted into the collection.
6. insert_many. A similar method, but which takes as input a list of documents, each of which is inserted into the collection. For example, live documents are usually updated, but if the collection is new, they need to be created first. In the service, this is achieved with the command: collection.insert_many(liveDocumentsList).

It should be clarified that all the functionality of the system monitoring service of the module function is performed by the described methods.

The backend of the server also implements a module - common.py. Its functions are heterogeneous, so its implementation is the result of choosing one of several service design scenarios.

The first function getAllProcs converts the process tree into a list (implemented through a dictionary, but without nesting). When the timer is started, the processes are displayed as a list, as this type of monitoring only collects changed processes, so there is no need to save the whole branch structure.

The second function sendCommand sends a command as a string to the agent. It takes as input the command itself, a tuple of socket address and port. The service provides several commands in total:

1. ‘[TEST] [DISKS] [disc_list] [uid_test]’ - test the input and output speed of data on the disc.
2. ‘[TEST] [LAN] [test_type] [additional parameters with a space] [uid_test_type]’ - run a test related to the local network. The types of tests are:

a. User list. test_type = ‘USERS’, no additional parameters.
b. List of network drives. test_type = ‘DISKS’, no additional parameters.
c. User information. test_type = ‘USER_INFO’, additional parameters: username.
d. Shared resources. test_type = ‘COMMON_SOURCES’, additional parameters: username.
e. Local groups. test_type = ‘GROUPS’, no additional parameters.
f. Host availability. test_type = ‘PING_HOST’, additional parameters: host address.

3. ‘[SET] [INTERFACE] [new value]’ - change the value of the network interface.
4. ‘[SET] [PORT] [new value]’ - change the network port.

The third function of the genUID service generates a unique sequence of characters that plays the role of an identifier. The function selects a random character from Latin letters (both uppercase and lowercase) or numbers 21 times, producing a string of 21 characters long. The probability of getting two identical identifiers is $2.3e^{-28}$, i.e. almost indistinguishable from 0.

Next comes a group of functions that form packets for timers. Two ‘live’ packages are fed to the input, between which the methods find the difference (which elements were not present before, which appeared, which changed status, etc.). These are called by the diffMeasurements function, which finds the difference between ‘timer_start’ and the new ‘live’.

The functions that calculate the difference separately for a particular sample work on the same principle, first they use the first data structure to add to the result those items that are not in the new one (i.e. the timer will show that some items have been deleted/stopped). Then the program goes through the new structure, adding items that are not in the old one (some items downloaded/started while the timer was running). The final object is a dictionary with two keys:

1. ‘+": a list of items that appeared.
2. ‘-": a list of disappeared items.

An example of such a function (process packet difference) is shown in Figure 7.

```
def diffProcs(old, new):
    measurement = {"+": [], "-": []}
    oldAllProcs = getAllProcs(old)
    newAllProcs = getAllProcs(new)

    for pid in newAllProcs:
        if pid not in oldAllProcs:
            measurement["+"].append(newAllProcs[pid])

    for pid in oldAllProcs:
        if pid not in newAllProcs:
            measurement["-"].append(oldAllProcs[pid])

    return measurement
```

Fig. 7: Code fragment. Calculation of changed processes

The last three functions are responsible for creating ‘avg’ and ‘avg_step’ documents. The first ‘avg’ created in the collection copies the values of the current ‘live’ documents and adds the field ‘count’=1 (function generateAvg). Then, when a new ‘live’ appears, the function adds all numerical parameters of the latter to the values of the ‘avg’ and increases ‘count’ by 1 (function calculateAvg). Thus, ‘avg’ is the sum of all received ‘live’ packets. The last function createAvgStep receives an ‘avg’ packet as input and divides all its numerical values by ‘count’.

The described solution for selecting modules and assembling collection methods is the result of working with the second principle of structural analysis of systems, [8]. It includes only functionally necessary system monitoring components, which are interconnected and implemented in such a way as to avoid an uncontrolled increase of resources to maintain the service functioning.

Undoubtedly, there will be an increase in resource requirements, and the content of the principle is one of the proofs of this. However, the result of the system monitoring service design will ensure their proportional increase as a result of the growing number of users. Let us also mention that the configured server backend solution will not solve the problem of non-linear increase of resources as a result of device growth due to the necessity of load balancing. This problem is not solved at the software level, as the solution assumes a high level of IT infrastructure maturity (according to Microsoft's model - Rationalised; according to Gartner's model - service level).

7 Conclusion

As a result, the architecture of the solution was designed - a system monitoring server that works with the database, exchanges messages with agents and clients, responds to incoming requests according to API, and performs all the described functionality.

Although the designed monitoring system implements the required functionality in full, some limitations were reached during the process.

1. Work should be done to optimize the web interface. This element was developed in native JavaScript, which negatively affects the speed of work. Since there are libraries that simplify the code structure, bypassing unnecessary steps that slow down the work, it is possible to reduce the number of iterations, the size of messages sent, and the complexity of algorithms.
2. An unresolved limitation is that working with a list of applications on MacOS is slower than on other operating systems by about 2-3 times. This is explained by the fact that in MacOS there is no built-in mechanism that would immediately display information about the application, you need to search for a special file in the package folder and read information from it. All this takes a relatively long time. At the same time, no other way to cover this sample has been found.
3. When collecting process data on Windows, a limitation on the completeness of the information has been reached. Since process collection on Windows was written in C#, the data that can be obtained with it is presented in a smaller volume than in the psutil library on Python. All important parameters are nevertheless present in both implementations, but minor metrics (e.g., open files) are not implemented on Windows.
4. When performing disc speed tests on Linux and macOS, there are only two tests without separating between random data and sequential data. The four tests available on Windows are possible in the future.

The found limitations do not reduce the significance of the obtained result. As a result, the service of remote monitoring of the functioning of the hardware and software complex of warehouse management was designed and developed. The monitor meets all technical requirements for data sampling and functionality. The advantage of the proposed architectural solution is the application of the principles of structural analysis of systems in the design. This allowed us to obtain a justified result, devoid of opportunistic distortions, at the end of the project.

Application of structural principles of systems analysis allowed to avoid risks:

- associated with excessive complication of the designed system in favor of functionality expansion;
- related to the choice of algorithms of the interaction of system components for its adaptation to changing conditions;
- associated with the choice of a stack of design and development technologies that provoke non-linear growth of resources required for system maintenance and support.

References:

[1] Wiener N. Perspectives in Cybernetics. In: Wiener N, Schadé JPBT-P in BR, editors. vol. 17, Elsevier; 1965, p. 399–415. https://doi.org/https://doi.org/10.1016/S0079-6123(08)60174-0.

[2] Hofkirchner W, Schafranek M. General System Theory. In: Hooker CBT-P of CS, editor. Handb. Philos. Sci., vol. 10, Amsterdam: North-Holland; 2011, p. 177–94. https://doi.org/https://doi.org/10.1016/B978-0-444-52076-0.50006-7.

[3] Rousseau D. General Systems Theory: Its Present and Potential. Syst Res Behav Sci 2015;32:522–33. https://doi.org/https://doi.org/10.1002/sres.2354.

[4] Anokhin P. *Nodal questions of the theory of functional systems.* in Russ. Moscow: Nauka; 1980.

[5] Bertalanffy L von, Sutherland JW. General Systems Theory: Foundations, Developments, Applications. *IEEE Trans Syst Man Cybern* 1974;SMC-4:592. https://doi.org/10.1109/TSMC.1974.4309376.

[6] Stallings W. Gerald M. Weinberg. A*n introduction to general systems thinking.* New York: Wiley, 1975, 279 pp. Behav Sci 1976;21:289–90. https://doi.org/https://doi.org/10.1002/bs.3830210409.

[7] Miller GR. The current status of theory and research in interpersonal communication. Hum Commun Res 1978;4:164–78. https://doi.org/https://doi.org/10.1111/j.1468-2958.1978.tb00606.x.

[8] Yourdon E. *Modern Structured Analysis.* Prentice-Hall; 1989.

[9] Wognum N, Mo J, Stjepandić J. Transdisciplinary Engineering Systems. Kenett RS, Swarz RS, Zo. A, Ed. Syst. Eng. Fourth Ind. Revolut. *Big Data, Nov. Technol. Mod. Syst. Eng.*, Wiley; 2019, p. 483–510.

[10] Wognum N, Verhagen WJC, Stjepandić J. Trans-Disciplinary Systems as Complex Systems. Vol. 5 Transdiscipl. Eng. A Paradig. Shift, 2017, p. 745–54. https://doi.org/10.3233/978-1-61499-779-5-745.

[11] Bogdanov A. *Tektologia: Universal organization science.* Book 1. Hull, UK: Center for Systems Studies Press; 1996.

[12] Caddy IN, Helou MM. Supply chains and their management: Application of general systems theory. *J. Retail Consum. Serv.* 2007;14:319–27. https://doi.org/https://doi.org/10.1016/j.jretconser.2006.12.001.

[13] Welch LR, Samuel AL, Masters MW, Harrison RD, Wilson M, Caruso J. Reengineering computer-based systems for enhanced concurrency and layering. *J. Syst. Softw.*, 1995;30:45–70. https://doi.org/https://doi.org/10.1016/0164-1212(94)00116-5.

[14] Ahmad A, Altamimi AB, Aqib J. *A reference architecture for quantum computing as a service.* J King Saud Univ - Comput Inf Sci 2024;36:102094. https://doi.org/https://doi.org/10.1016/j.jksuci.2024.102094.

[15] Beierling H, Richter P, Brandt M, Terfloth L, Schulte C, Wersing H, Vollmer AL. What you need to know about a learning robot: Identifying the enabling architecture of complex systems. *Cogn. Syst. Res.* 2024;88:101286. https://doi.org/https://doi.org/10.1016/j.cogsys.2024.101286.

[16] Loschi H, Nascimento D, Smolenski R, Lezynski P. Cyber–physical system for fast prototyping of power electronic converters in EMI shaping context. *J. Ind. Inf. Integr.* 2023;33:100457. https://doi.org/https://doi.org/10.1016/j.jii.2023.100457.

[17] Zhang T, Shi Y, Cheng Y, Zeng Y, Zhang X, Liang S. The design and implementation of distributed architecture in the CMOR motion control system. *Fusion. Eng. Des.* 2023;186:113357. https://doi.org/https://doi.org/10.1016/j.fusengdes.2022.113357.

[18] Yang L, Li Zhang X, Yaoming L, Liya L, Maolin S. Modeling and control methods of a multi-parameter system for threshing and

cleaning in grain combine harvesters. *Comput. Electron. Agric.* 2024;225:109251. https://doi.org/https://doi.org/10.1016/j.compag.2024.109251.

[19] Anistratov P, Golobokov Y, Pavlov V. Hardware-software Complex Prototyping for the Pulse Power Supply Control System of Tokamak T-15. *Procedia Comput. Sci.* 2015;66:546–55. https://doi.org/https://doi.org/10.1016/j.procs.2015.11.062.

[20] Wiener N. *Cybernetics or Control and Communication in the Animal and the Machine.* The M. I. T. Press; 2019. https://doi.org/10.7551/mitpress/11810.003.0010.

[21] Awad AI, Shokry M, Khalaf AAM, Abd-Ellah MK. Assessment of potential security risks in advanced metering infrastructure using the OCTAVE Allegro approach. *Comput. Electr. Eng.*, 2023;108:108667. https://doi.org/https://doi.org/10.1016/j.compeleceng.2023.108667.

[22] Villalón-Fonseca R. The nature of security: A conceptual framework for integral-comprehensive modeling of IT security and cybersecurity. *Comput. Secur.* 2022;120:102805. https://doi.org/https://doi.org/10.1016/j.cose.2022.102805.

Contribution of Individual Authors to the Creation of a Scientific Article (Ghostwriting Policy)

The authors equally contributed in the present research, at all stages from the formulation of the problem to the final findings and solution.

Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself

The research was funded by the grant Russian Science Foundation No.24-21-20089.

Conflict of Interest

The authors have no conflicts of interest to declare.