IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Mathematical Modelling of Engineering Problems
Vol. 11, No. 12, December, 2024, pp. 3420-3430
Journal homepage: http://iieta.org/journals/mmep

The Use of Discrete-Deterministic Models in the Development of Software for Controlling Autonomous Electric Power Plants

Mahmoud M.S. Al-Suod[1,2], Mohammad S. Zannon[3*], Oleksandr Ushkarenko[4], Olha Dorohan[4]

[1] Electrical Engineering Department, Al-Ahliyya Amman University, Amman 19328, Jordan
[2] Electrical and Mechatronics Engineering Department, Tafila Technical University, At-Tafilah 66110, Jordan
[3] Department of Mathematics, Tafila Technical University, At-Tafilah 66110, Jordan
[4] Electrical Engineering Department, Admiral Makarov National University of Shipbuilding, Mykolaiv 54050, Ukraine

Corresponding Author Email: zanno1ms@gmail.com

https://doi.org/10.18280/mmep.111220

Received: 7 August 2024
Revised: 13 October 2024
Accepted: 20 October 2024
Available online: 31 December 2024

Keywords:
software development, electric power station, digital automata, UML diagrams

ABSTRACT

The paper presents an analysis of tasks performed by the software for automated control systems of autonomous electric power plants, leading to the identification of functional requirements. This analysis establishes operational modes, such as scheme designer mode and autonomous electric power plant monitoring and control mode, defines the component library, and outlines requirements for each component. The use of discrete-deterministic models in the form of digital automata, the research formalizes the problem of analyzing and synthesizing control algorithms. The novelty of this methodology lies in integrating digital automata with UML diagrams to develop adaptive software. By linking UML state diagrams directly with digital automata models, the system ensures consistency between conceptual design and code implementation. The research contributes to best practices in software engineering for complex, distributed control systems. The approach, proposed in the paper, allows developers to conceptualize the user interface of automated operator workstations as interconnected systems with defined relationships and communication types.

1. INTRODUCTION

At present, there is a tendency to move from centralized control systems of power plants to distributed control systems, which reflect a growing need for enhanced flexibility, scalability, and reliability of the systems that have a few hierarchical levels. Existing software frameworks for power plant automation, such as Supervisory Control and Data Acquisition (SCADA) systems, have limitations when applied to Autonomous Electric Power Plants (AEPPs). While SCADA provides reliable monitoring and remote control capabilities, it lacks the adaptability needed to support dynamic structural changes within AEPPs. Furthermore, most existing control frameworks struggle to provide effective solutions for real-time synchronization of diesel-generator units (DGUs) and the efficient distribution of active and reactive loads. Traditional algorithms for generator synchronization and load management are often hardware-dependent and difficult to modify, which limits their applicability in distributed power networks. This lack of adaptability is particularly problematic for AEPPs, where multiple generators operate in parallel and require constant monitoring and reconfiguration based on network conditions.

Although papers [1, 2] emphasize the importance of distributed systems in modern automation, the specific challenges related to AEPPs (e.g., real-time synchronization of DGUs, load sharing, and adaptability to dynamic configurations) are not addressed. Additionally, the works lacks a formalized framework for transitioning from design models to executable software. In the context of AEPPs, distributed control systems minimize hierarchical layers while improving system responsiveness. However, the adoption of these systems presents several critical challenges. Specifically, developing software for automated control systems (ACS) that can reliably monitor, manage, and synchronize complex generator networks in real-time remains a key issue. This is particularly important given that AEPPs often feature multiple DGUs with varying load demands and power distribution structures, which must be reconfigured dynamically during operation. One of the most difficult stages in the implementation of automated control systems (ACS) is the development of software that not only controls the operational parameters but also adapts to the power plant's structural changes [3, 4]. These papers explore predictive control for microgrid systems, analyzing key factors like reliability and controllability. It provides insights into load balancing and control under uncertain conditions, which align with some of the challenges in AEPPs. However, microgrids and AEPPs differ in their operational structure, particularly in the synchronization of multiple generators and dynamic load sharing. Those papers do not cover modular software verification or real-time control algorithm synthesis. Operators require precise, real-time information about both plant parameters and system structure, typically visualized

through mnemonic diagrams. These diagrams, which use visual metaphors to represent key components, facilitate intuitive control and monitoring. The core challenges this study addresses is the lack of formalized methods for designing ACS software that ensures both adaptability to structural changes and long-term reliability under AEPP-specific conditions.

In the study conducted by Kabbara et al. [5], several advances and future directions for virtualized control in power systems were considered, but it emphasized the need for research into scalable architectures that can handle diverse power system components without performance degradation. Existing methods, described by Kabbara et al. [5], don't sufficiently cover the real-time interaction between virtual components and hardware, leading to uncertainty in operational performance under stress conditions.

A critical review of related work highlighted additional limitations. SCADA systems, traditionally used for top-level control [6, 7] offer reliable monitoring and control but are not designed to support dynamic structural changes in AEPPs. With regard to the AEPP, the ACS should be able to automatically or remotely start and stop DGUs, synchronize the diesel-generators (DGs) with the buses of the main switchboard (MSB), share the active and reactive load between the parallel operating DGs, monitor the status of the AEPP main equipment and others. However, existing solutions for automating DGUs, such as synchronizing them with MSB and balancing active/reactive loads [8, 9] often lack the flexibility needed for the dynamic nature of AEPP operations.

Some researches explored IoT technologies for automating ship operations [10] and vessel maintenance [10], but he application of such technologies in AEPPs remains underexplored. The study of Kamolov and Park [10] is relevant as it demonstrates the potential of IoT to enhance automation. To analyze the reliability of IoT the technique described in the study [11] can be used. However, it leaves open questions about how these technologies can be integrated into AEPP ACS software to support real-time control and monitoring. This study aims to fill that gap by exploring how advanced software design approaches, informed by IoT frameworks, can improve AEPP automation.

As reported by Bayer et al. [12] and Koc et al. [13], Unified Modeling Language (UML) diagrams are widely used in the software engineering to visualize system interactions, but their use in AEPP automation software is under-researched. UML diagrams provide a structured way to model software components, but their application in AEPPs presents unique challenges, such as the need to adapt to evolving plant structures in real-time. However, Koc et al. [13] demonstrated that the application of UML to real-time control systems like AEPPs is not covered. In this case UML, which is a visual language to define and document a system [13, 14], can be used to describe scenarios that express how users use a system, and how different parts of the system interact with each other. Due to the fact that software development for ACS AEPP is a complex process, and there are a number of methods for assessing software complexity [15]. Every aspect of a system or application should be determined to develop software. Existing publications [16-18], highlight the use of UML in various industries, from electromechanical control systems to railway diagnostics, but do not address AEPP-specific requirements. UML also offers cost-saving advantages in software development [19]. But existing researches lacks practical frameworks for using UML in conjunction with discrete-deterministic models to handle the specific requirements of AEPPs. This paper addresses these gaps by proposing a systematic methodology that integrates both tools, offering a more formalized approach to designing reliable, adaptive ACS software.

Worku et al. [20] focused on power management and synchronization strategies in microgrids using real-time control methods. It provides insights into the importance of maintaining voltage stability and load balancing to ensure system reliability. But it does not provide a comprehensive modular software framework or formalized verification process for control systems. This can be addressed by introducing UML diagrams and discrete-deterministic models to develop and verify AEPP software components. A hierarchical control system for reconfigurable solar power plants was considered by Debnath et al. [21]. The issues how power systems can self-adjust to changing operational conditions, aligning with the goals of AEPPs, were discussed. At the same time, that paper primarily focused on solar-based plants and lacks detailed software engineering approaches such as formal verification with UML models.

When developing software for specialized systems like AEPPs, it is crucial to use modeling approaches that facilitate the formalization and quality assurance of software components. As reported by Al-Suod et al. [22], the theory of digital automata offers a robust framework for defining discrete-deterministic software models. UML state diagrams, in particular, can represent the finite state behavior of software components. Then the software being developed will be a network of interacting digital state machines, obtained by composing individual discrete-deterministic models, which lends itself well to formalizing the description of the software operation process. Also, this deterministic modeling approach ensures that AEPP software can be implemented reliably in an object-oriented programming environment.

Existing scientific publications provide useful insights into distributed control, automation frameworks, and modeling techniques, but significant gaps remain regarding their application to AEPPs. In particular, there is limited research on software tools and methods that support the real-time, remote control of DGUs while accounting for AEPP-specific complexities. Currently, the issues of the processes of substantiating, evaluating, and selecting options for developing ACS for specialized purposes lack formalization and hinder the comprehensive consideration of several significant parameters when choosing solutions, as was stated by Lyaskovskiy et al. [23] and Garcia et al. [24], remain unresolved. Therefore, further underscoring the need for systematic, adaptable software design approaches.

The analysis of publications showed that the problems of creating specialized software for automated remote control of DGUs in real time, taking into account the specifics of work in AEPP conditions, are not sufficiently covered. UML is typically used for static design and documentation, and its integration with digital automata for real-time software implementation remains unexplored. Given the fact that information technologies are developing extremely rapidly, publications of other scientists 5-10 years ago on this issue can be considered to have lost their relevance and do not meet modern challenges.

The aim of this research is to address these gaps by developing a structured method for designing ACS software that meets the unique demands of AEPPs. The research

focuses on using UML diagrams and discrete-deterministic models to create a software architecture that can adapt to changing AEPP structures. UML state diagrams are used to graphically represent the behavior of each software component. They formalize how components respond to various events and how transitions between states occur. By leveraging these tools, the aim is to enable the seamless transition from conceptual models to code implementation while ensuring software reliability, quality, and cost-efficiency. The results will contribute to the development of more advanced, adaptive automation systems for AEPPs, enhancing the controllability and resilience of power generation processes.

2. SOFTWARE REQUIREMENTS AND COMPONENT LIBRARY COMPOSITION

The primary function of the top-level software is to remotely monitor parameters and control the AEPP. To develop the necessary functionality, the software must have tools for creating a diagram of an electric power plant, real-time monitoring and controlling of the AEPP, analyzing the modes of operation of DGs and the software.

Requirements for the software used to monitor parameters and control the AEPP can be categorized into three groups that are described in Table 1 based on the roles of the users involved.

Table 1. The roles of the users

Operator	Key Responsibilities
System Setup Operator	Creates mnemonic diagrams, displays the processes at several levels, configures component properties, and establishes connections between components (e.g., resizing, moving, or rotating elements).
Power Plant Control Operator	Monitors real-time processes, starts/stops DGUs, synchronizes DGUs with the main switchboard, and manages load distribution across units, DGUs protection, monitors the power quality.
Analyst	Analyzes data exchange packets, monitors communication channel loads, and evaluates DGU operation to optimize performance through iterative configuration and analysis, forecasts the changes in the AEPP load.

Below, the precedents are described in more detail.

- Graphic display of processes at several levels: display of the power plant block diagram; display of electrical parameters and states of discrete signals.
- Starts/stops DGUs requires the presence of a Button component (latched or pulsed).
- DGU protection is a general concept and consists of protection against reverse power, loss of excitation, maximum current flow, voltage changes beyond permissible limits, frequency changes beyond permissible limits, overload and insufficient or maximum output power.
- Synchronization system of the DGU with the MSB should be represented by a separate component, which contains means for initiating the process of automatic precise synchronization of the running DGU with the network at the hardware level, displaying the current values of voltages (RMS) and frequencies of the DGU and the network, the phase angle shift between the voltages, the possibility of changing synchronization settings and parameters of discrete signals for controlling generator excitation systems and diesel revolutions period.
- Changing the state of the load consists of connecting or disconnecting it from the main switchboard buses and can be done by changing the state of the Circuit Breaker component.
- A number of requirements are put forward for the distribution of loads (active and reactive) between parallel operating DGUs: dividing the DGU sets into separate independent sections; availability of means for specifying the distribution scheme and time of the transient process of load redistribution for each of the sections.
- To monitor the quality of electricity, a separate component can be created that will inform the Operator when the power quality indicators exceed normal and maximum permissible values, and also be able to monitor the state of the electric power plants in detail in a separate dialog box.
- Forecasting the results of connecting the load is a means of information support for the operator.

By analyzing the requirements associated with the performance of actions by the Power Plant Control Operator, it is possible to compile a list of components that should be in the software library. Using the software involves a three-stage iterative process – creating or modifying a mnemonic diagram of the power plant and configuring its components, operating the software in its primary mode, and analyzing system performance. The first and the third stages appear visually similar to the user, so the software usage can be categorized into two modes: the first one is the design mode of the mnemonic diagram (i.e., AEPP scheme), and the second one is the monitoring and control mode of AEPP.

3. DEVELOPMENT OF THE SOFTWARE STRUCTURE

A key challenge in developing ACS software is ensuring that all software components behave predictably under varying operational conditions. To address this, this research adopts discrete-deterministic models, which represent the behavior of each software component as a finite set of states. Each state encapsulates a specific operational condition of the component, while state transitions occur in response to events, such as power fluctuations or load adjustments. This modeling approach ensures that the system can respond deterministically – meaning that every input leads to a predictable output, crucial for AEPP reliability. To formalize these models, UML state diagrams are used. UML state diagrams are visual tools that depict the different states a system or component can occupy and define the transitions between them. In this research, each state diagram corresponds to a specific software module within the AEPP ACS.

The first step in the software development is to decide on the circuit components. Components are understood as graphic symbols and designations corresponding to physical (for example, a generator) or virtual (for example, a text field for displaying generator parameters) elements of the AEPP.

Components can be divided into three groups:

- Components that have a connection with other components ("Circuit Breaker", "Transformer", "Diesel", "Generator", "Induction motor" etc. for drawing up a block diagram).
- A component for communication ("Bus").
- Components that have no connection with other components.

Figure 1 is a UML class diagram that shows the relationships between core software classes, such as the DGU class, synchronization class, and control buttons, the key interactions between the system's components, data flows between modules and how commands propagate throughout the system. The software architecture for the AEPP control system follows a modular, layered design to ensure flexibility, scalability, and maintainability.

In Table 2, the symbols used in Figure 1 and the classes designed to perform the functions of monitoring parameters and controlling the AEPP are described. The relationships between classes are inheritance and aggregation. Each component plays a defined role, and their interactions are modeled using UML diagrams to maintain consistency between design and implementation. Each block in the UML class diagram encapsulates a digital state machine whose behavior is determined by its functional purpose. Transitions between the states of the components are triggered by real-time data inputs from equipment sensors and control commands from the operator interface. By using discrete-deterministic models, the software ensures that each DGU follows a precise sequence of operations, minimizing the risk of misoperation.

The DGU class (K2.1.1), and other classes, inherits basic functions from a basic linkable class (K2.1), while the synchronization class aggregates real-time input data from DGUs and the MSB. The Control Button class (K2.2.2) sends commands to DGUs, initiating synchronization or load sharing. For example, when the Power Plant Control Operator triggers a synchronization request through the Control Button, the synchronization class retrieves real-time voltage and frequency values from the DGU and MSB (K2.4, K2.5). If the conditions match, the synchronization process completes, and the DGU transitions to the "Active" state.

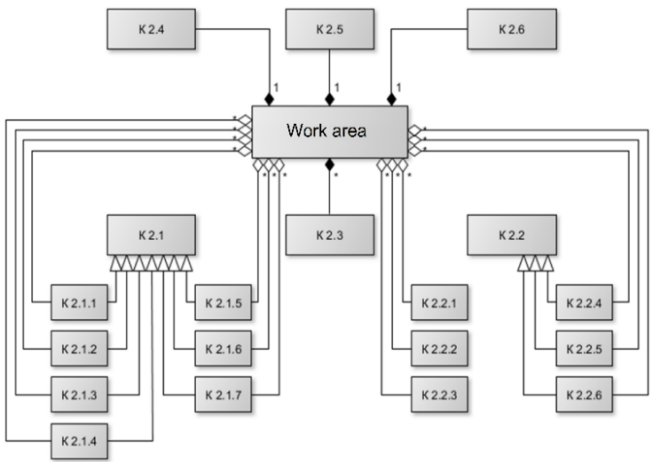

Figure 1. Basic class diagram

The interaction between objects in the system is carried out through messages, among which the following groups can be distinguished:

- messages between the work area and components regarding graphic actions (denoted conditionally as a set of actions $W = \{w1, w2 \ldots\}$; these actions are implemented using base classes, so the message group applies only to base classes and workspace.
- messages between the work area and components regarding data exchange (multiple actions $D = \{d1, d2\}$); these messages refer to components that provide data exchange with automation controllers in network mode ("Diesel-generator", "Generator", "Automatic switch", "Control button", "LEDs").
- messages between components (set of actions $V = \{v1, v2\}$); a group of messages refers to changing the color of the buses and related components when the Automatic switch state changes and displaying the parameters of the generator received during the data exchange.

In Table 2 the correspondence of markings in Figure 1 to components and classes is presented. The purpose of DGU is to generate electricity and maintains power levels during AEPP operation. The DGU component (K2.1.1) interacts with the synchronization component to connect to the main switchboard. The parameters of DGU are voltage (230V/400V), frequency (50/60 Hz), and active power output (100-500 kW). Synchronization system synchronizes DGUs with the MSB by matching frequency and phase angles. The component is represented by the Synchronization button (Control button, K2.2.2) on the user interface, settings panel (dialog box, K2.4, K2.5), and automated synchronization hardware.

Table 2. Correspondence of block names on the diagram to components and classes

Component	Block Name	Class Name	Basic Class
Working field	Work area	CTestView	CView
A basic linkable class	K 2.1	CStaticNoRsz	CStatic
DGU	K 2.1.1	Cd_gEx	CStaticNoRsz
Diesel	K 2.1.2	CD	CStaticNoRsz
Generator	K 2.1.3	CG	CStaticNoRsz
Induction motor 1	K 2.1.4	CAD	CStaticNoRsz
Induction motor 2	K 2.1.5	CADEx	CStaticNoRsz
Automatic switch	K 2.1.6	CSwitch	CStaticNoRsz
Transformer	K 2.1.7	CTrans	CStaticNoRsz
A scalable base class	K 2.2	CStaticEx	CStatic
Text field	K 2.2.1	CEditEx	CEdit
Control button, system button	K 2.2.2	CButtonEx	CButton
Progress bar	K 2.2.3	CProgressEx	CProgressBar
LED 1	K 2.2.4	CLedRound	CStaticEx
LED 2	K 2.2.5	CLedSquare	CStaticEx
Arrow indicator	K 2.2.6	CIndicatorArrow	CStaticEx
Bus	K 2.3	BUS STRUCT	
Generator parameters measurement dialog	K 2.4	CVoltCurDlg	CDialog
Diesel parameters measurement dialog	K2.5	CDieselDlg	CDialog
Data exchange process display dialog	K2.6	CCommunicationDIg	CDialog

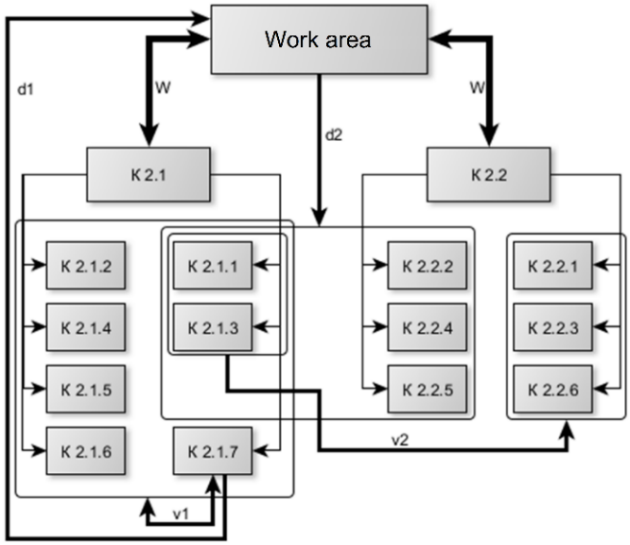

Figure 2. Interaction between elements of the software

The dialog boxes may contain various indicators (K2.2.3 – K2.2.6) are used to show the measured parameters in a user-friendly manner. The software displays parameter updates with a latency of ≤1 second. Updates include generator status, load conditions, and synchronization processes. Control Button (K2.2.2) components, which can be latched (holds state after activation) and non-latched (momentary action), initiates DGUs and induction motors (K2.1.4, K2.1.5) start/stop commands, synchronizations, and DGUs load sharing adjustments. The software should redistribute loads within 2 seconds following the disconnection of a generator, maintaining a load balance error below ±2%. Bus component (K2.3) and Automatic switch (K2.1.6) facilitates communication between DGUs, transformers (K2.1.7), induction motors and the MSB. The Bus component includes multiple color-coded connections to represent different states of communication and power flow.

Figure 2 shows the communication flow and state transitions among components.

The Work Area initiates requests to automation hardware via a communication bus. Data from the DGUs (e.g., power output and frequency) is transmitted to the Load Sharing System and visualized in the operator interface. Upon detecting anomalies (e.g., voltage fluctuations), the Load Sharing System sends corrective commands to the DGUs to redistribute the load dynamically. The system waits for real-time input from DGUs. If a DGU goes offline ("Inactive" state), the system initiates a load redistribution process through the Load Sharing System. Once load sharing is complete, the system returns to normal monitoring mode.

4. SOFTWARE OPERATING MODES

The designed software is a system with complex behavior, so it is advisable to use an object-oriented programming language for its implementation. The software contains a main window, consisting of two components – a frame and a work area. Since the designed software must have an additional menu (for switching program operating modes, setting connection parameters, etc.), support working with documents (save/restore mnemonic diagrams and settings), and at any time interaction can be carried out with only one mnemonic diagram, the SDI (single dialog interface) software was chosen as the base one.

The scheme designer mode combines two submodes of working with software: creating (or editing) a AEPP mnemonic scheme, setting the properties of its components and software environment; analysis of AEPP operation and software.

The actions of the User with the dialog windows are common to all scheme elements (components and Workspace). In the scheme designer mode, the requirement for the class is to encapsulate the processes of moving the component and creating a connection with it (creating the object itself is a task of the Workspace). The logic of the class in this mode can be represented as a Mealy state machine$A1 = \{Q, X, Y, f, g\}$ where $Q = \{q0, q1, q2, q3, q4, q5, q6\}$ - the set of states of the state machine: $q0$ – initial state (the object was created); $q1$ – the element is in focus and marked; $q2$ – movement of an object; the state is complex (the logic of behavior in this state is shown in Figure 3; the shift is the difference between the last remembered point and the coordinates of the cursor's current position, half of the cell is a distance equal to half of the grid step in any of the *Ox*, *Oy* directions or both at once); $q3$ – the link with the bus is possible; $q4$ – the starting of linking process with the bus; $q5$ – the finishing of linking process with the bus; $q6$ – the object was deleted, indicating the final state.

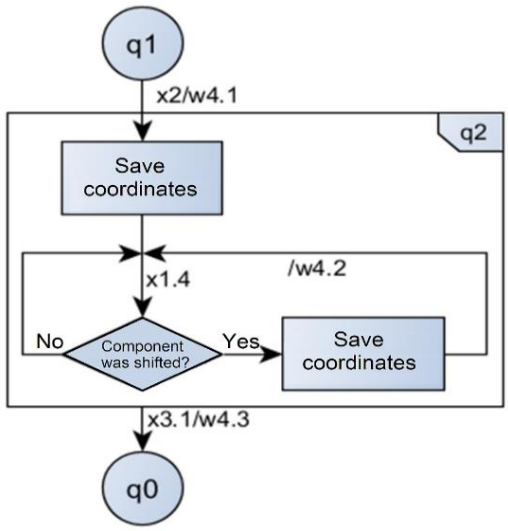

Figure 3. The behavior of the state machine in the state *q2*

The valid input actions set is:

$$X = \{x1.1, x1.2, x1.3, x1.4, x2, x3.1, x3.2, x4, w1, w2\}$$

where, $x1.1$ – moving the cursor over the object; $x1.2$ – moving the cursor over the object in the area of possible link; $x1.3$ – moving the cursor over the object outside the area of possible link; $x1.4$ – moving the captured cursor; $x2$ – a click of the left mouse button; $x3.1$– the left mouse button released; $x3.2$ – the left mouse button released in the possible link area; $x4$ – pressing the Delete key; $w1$– "Remove focus" message; $w2$ – "Change bus" message.

The set of initial actions:

$$Y = \{w1,\ w3,\ w4.1,\ w4.2,\ w4.3,\ w5,\ w6\}$$

where, $w1$ – "Remove focus" message; $w3$ – the message "Beginning of bus creation"; $w4.1$ – the message "Beginning

to move the object"; $w4.2$ – the message "Moving the object by one cell"; $w4.3$ – the message "End of object movement"; $w5$ – the message "End of bus creation"; $w6$ – "Delete object" message; functions f and g show states and output symbols, as depicted in Figure 4 [25].

Next, for each class, an automaton (final state machine) is designed that must implement the behavior of the corresponding component. The input and output actions for these automats will be receiving/sending messages from/to the listed sets.

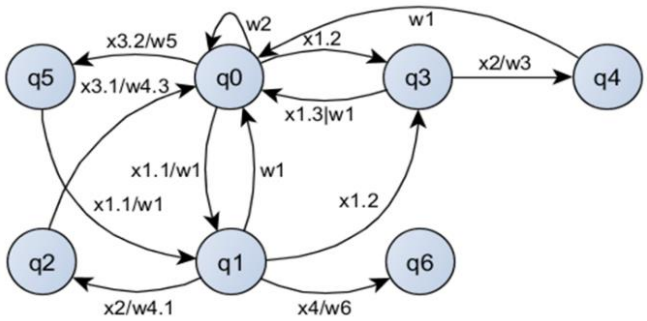

Figure 4. The state diagram of the class behavior

5. DISCRETE-DETERMINISTIC MODELS OF THE SOFTWARE COMPONENTS

In the mode of control and AEPP parameters monitoring, the software must perform several tasks:

– Data exchange with automation equipment includes transmitting operator commands and reading, displaying, and processing discrete and analog values captured by the hardware;

– Accumulation of statistical data on the load of the communication channel – the amount of bytes transmitted and received per unit of time;

– Accumulation of data about the diesel-generator units operating modes: the average values of the power they generate to the overall load during a specific period.

A unit of data exchange is a transaction comprising a request and its corresponding response – sending and receiving data packets using a serial port. The input influences associated with messages from the operating system during operator interaction with the software, the operation of timers and actions with the serial port, are marked as $e1$ – switching the software to the mode of power plant control; $e2$ – switching the software to the mode of the scheme design; $e3$ – the data packet received; $e4$ – place the cursor within the component area; $e5$ – the change in the structure of the power plant; $e6$ – the timer event; $e7$ – the load on at least one DGU exceeds permissible ranges; $e8$ – the synchronization window was opened; $e9$ – setting values received; $e10$ – the status of the generator circuit breaker was updated; $e11$ – the RMS of voltages, frequencies and phase difference between voltages were gained; $e12$ – changing reference point values; $e13$ – the control parameters were changed; $e14$ – the data packet was received; $e15$ – error when opening serial port; $e16$ – the synchronization dialog window was closed; $e17$ – the start of the synchronization process; $e18$ – the component received the data; $e19$ – mouse left button click; the input variables corresponding to the result of performing certain actions, as $x1$ – error in data packet; $x2$ – the error has been resolved; $x3$ – the error cannot be corrected without substituting the hardware; $x4$ – the connection is active: $x4.1$ – with the at least one display components, $x4.2$ – with one of the DGU protection components; $x5$ – the data has been generated; $x6$ – the request queue contains items; $x7$ – the data was forwarded to the component; $x8$ – the timer is active; $x9$ – for DGU protection, the data acquired by the Load Sharing System is in use; $x10$ – the data measured independently is used for DGU protection; $x11$ – the direct condition is met; $x12$ – the opposite condition is met; $x13$ – the DGU operates in parallel with other DGUs; $x14$ – the generator circuit breaker is in the close position; $x15$ – there are no items in the queue; $x16$ – database entry is permitted; $x17$ – displaying of the data packet contents is permitted; $x18$ – the database connection error; $x19$ – database connection error fixed; $x20$ – the system for correcting the diesel revolutions and/or the generator excitation is turned on, and the resulting operations that occur when specific combinations of input factors and variables marked as $z1$ – add request packets to the queue: $z1.1$ – Generator parameters (RMS values of voltage and current, frequency, and the power factor); $z1.2$ – the indicators needed to ensure the functions of DGU protection; $z1.3$ – close the discrete output; $z1.4$ – open the discrete output; $z1.5$ – sync parameters and control configurations; $z1.6$ – the data about the generator circuit breaker state; $z1.7$ – the current values of frequencies, voltages, and the phase shift angle between the voltage vectors of the MSB bus and the operating DGU; $z1.8$ – change synchronization configurations; $z1.9$ – change control configurations; $z1.10$ – the state of the section's generator circuit breakers; $z1.11$ – the data required to calculate the power generated by DGUs operating in parallel; $z1.12$ – enable corrective measures for frequency and voltage control; $z1.13$ – disable corrective actions for frequency and voltage regulation; $z1.14$ – activate the circuit breaker/relay; $z1.15$ – deactivate the circuit breaker/relay; $z2$ – queue request removal; $z3$ – data transfer to the component; $z4$ – the timer activated; $z5$ – the data displaying; $z6$ – the timer disabled; $z7$ – create a queue for data exchange; $z8$ – alter the bus color; $z9$ – dispatch the request data packet to the automation hardware; $z10$ – send data to a state that initiates the data exchange process; $z11$ – extract the request from the stack; $z12$ – initiate a permanent queue for requests; $z13$ – open the serial port; $z14$ – close the serial port; $z15$ – remove the request; $z16$ – save data about the state of loads and loads of the DGU; $z17$ – show the data packets content, then the explained behavior of the Work area class in the mode of the power plant control can thus be modeled by the finite state machine $A_8 = \{Q, X, E, Z, f, g\}$, where $Q = \{q0 - q8\}$ – automaton states set: $q0$ – initial condition; $q1$ – awaiting the initiation of the data exchange cycle; $q2$ – pending a reply to the request; $q3$ – pending the database entry; $q4$ – database entry; $q5$ – database connection error; $q6$ – waiting for synchronization of the diesel generator with the main switchboard buses; $q7$ – synchronization of the diesel generator with the main switchboard buses; $q8$ – waiting for control actions to be turned off for frequency and/or voltage control systems; X, E, Z – groups of permissible input influences, input parameters and resulting actions, described above; f and g – state functions and output symbols, presented as a UML state diagram in Figure 5 [25].

The Generator component behavior can be described as a

final state machine $A_9 = \{Q, X, E, Z, f, g\}$, where $Q = \{q0 - q7\}$ denotes the collection of automaton states: $q0$ – the initial condition; $q1$ – awaiting a response to a request; $q2$ – data packet check; $q3$ – there is no connection with automation tools; $q4$ – transferring data to the DGU protection component; $q5$ – transmission of data to display components; $q6$ – waiting for generator parameter values to be displayed; $q7$ – display of generator parameter values; f and g are state functions and output symbols, presented in the form of a UML state diagram in Figure 6.

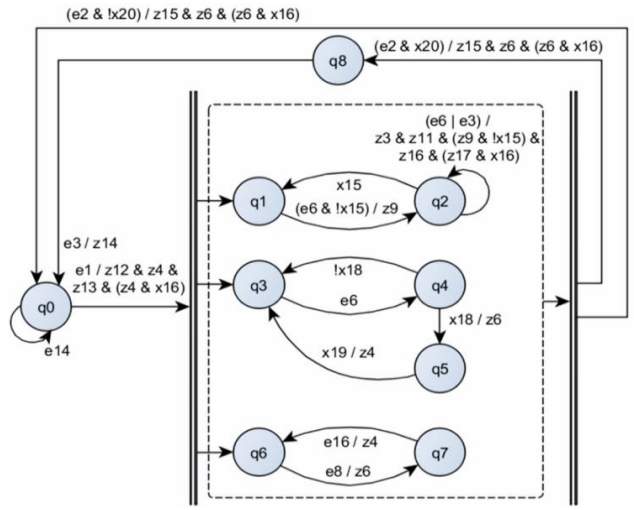

Figure 5. State diagram of the work area class in the AEPP parameters monitoring and control

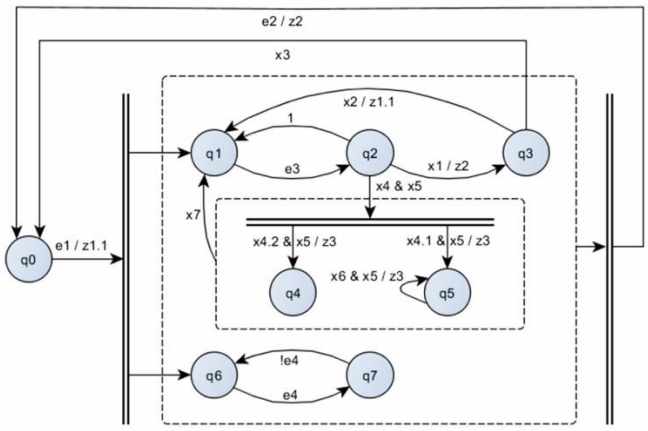

Figure 6. UML state diagram of the generator class

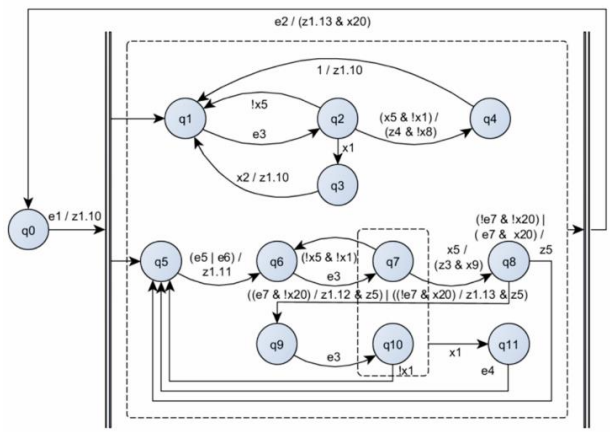

Figure 7. State diagram of the Load Sharing class

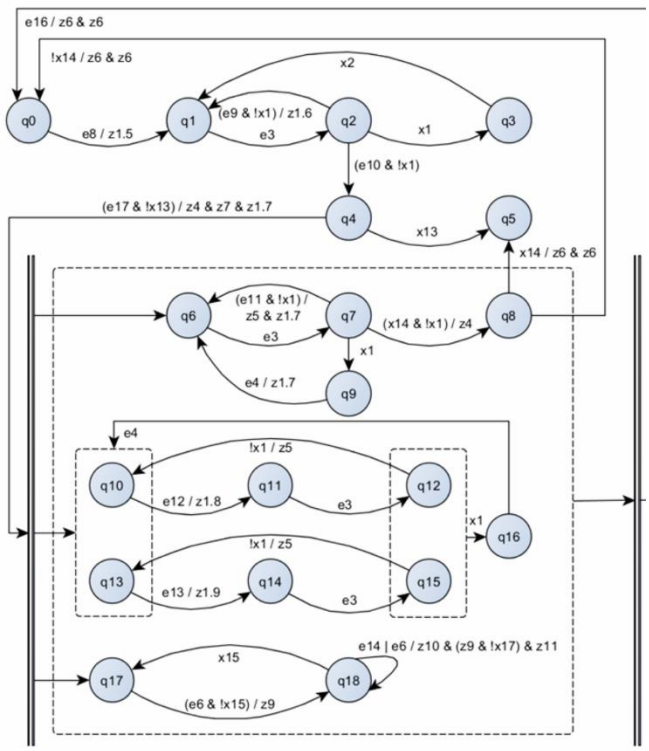

Figure 8. State diagram of the synchronization system class in the monitoring and control mode of AEPP parameters

In the initial state ($q0$) the component waits for the software to enter the power plant monitoring and control mode. When this event occurs, the parallel execution of several processes begins: data exchange with automation tools (states $q1$ – $q3$); simultaneous transfer of Generator parameters obtained during successful data exchange to the indication components ($q4$) and, if the information flow minimization mode is enabled, to the Protection component of this DGU ($q5$); displaying Generator parameters in the tooltip window when moving the cursor over the component ($q6$ – $q7$).

The behavior of the Load Sharing System component can be described in the form of an automaton $A_{10} = \{Q, X, E, Z, f, g\}$, where $Q = \{q0 - q11\}$, denotes the states of the automaton. The UML state diagram is shown in Figure 7.

In Figure 7 the states are: $q0$ – the initial state; $q1$ – pending a response to a request; $q2$ – data packet check; $q3$ – no connection to automation equipment; $q4$ – checking the section structure; $q5$ – waiting for the DGU workload to be checked; $q6$ – waiting to receive the power values generated by each DGU to the total load of the section; $q7$ – data packet check; $q8$ – calculation of deviations of real values of DGU loads from ideal ones; $q9$ – waiting for a response to the command to enable control actions; $q10$ – data packet check; $q11$ – no connection to automation equipment; f and g are state functions and output symbols.

When the software switches to the scheme designer mode, the Work area sends a relevant message to the Load Sharing System. When it is received, the component checks for the presence of enabled control actions and, if they are present, sends a command to disable them.

The synchronization system component is a control button and associated dialog box that is displayed when the user clicks it in the AEPP parameters monitoring and control operational mode. The UML state diagram of the synchronization system class presented in Figure 8.

As stated above, due to the peculiarities of the hardware implementation of the synchronization device, when opening a dialog box, information exchange with other automation tools must be temporarily blocked. Then the algorithm of the synchronization system can be represented as an automaton: $A_{15} = \{Q, X, E,\ Z, f, g\}$, where $Q = \{q0\ -\ q9\}$, represents the states of the automaton: $q0$ – initial state; $q1$ – waiting for data; $q2$ – data packet check; $q3$ – no connection with automation equipment; $q4$ – checking the condition of the generator circuit breaker; $q5$ – synchronization completed; $q6$ – waiting to receive values of voltages, frequencies and phase angles; $q7$ – data packet check; $q8$ – waiting for the feedback from the generator switch to operate; $q9$ – no connection with automation equipment; $q10$ – waiting for synchronization settings to change; $q11$ – waiting for data about the result of changing the settings; $q12$ – checking the data packet with the result of executing the command to change the synchronization settings; $q13$ – waiting for changes in control action parameters; $q14$ – waiting for data about the result of changing parameters; $q15$ – checking a data packet with the result of executing a command to change the parameters of control actions; $q16$ – no connection with automation equipment; $q17$ – waiting for process of data exchange; $q18$ – pending a response to a request; functions f and g define states and output symbols.

When the dialog box opens, the state machine goes into states $q1$ – $q3$, corresponding to polling the values of the synchronization settings, parameters of control actions for correcting the diesel revolutions period and excitation of the generator. Next, in state $q4$, the state of the generator switch is polled: if it is closed (the DGU is switched on for parallel operation with the main switchboard buses), the synchronization system displays this situation and goes to state $q5$, awaiting further user actions. Alternatively, the user can initiate the synchronization process at the hardware level by clicking the appropriate button in the dialog box, after which the processes in the synchronization system are divided into three components: data exchange with hardware ($q17$ – $q18$), changing synchronization settings or control action parameters ($q10$ – $q16$), generation of queries and display of current values of frequency and voltage differences, phase difference between the main switchboard bus voltage vectors and the generator ($q6$ – $q8$).

6. SOFTWARE IMPLEMENTATION AND VERIFICATION RESULTS

In Figure 9, the main window of the developed software to monitor the parameters and control the AEPP is shown. Software for automated control systems of AEPP is developed using modern tools and technologies for monitoring the parameters of power units, power quality and effective automated control of AEPP and the process of generating electricity. The AEPP control software was implemented using C^{++} programming language and object-oriented programming principles, with each software component mapped directly to UML classes. The development process followed a modular approach, ensuring that components like DGUs, load sharing systems**,** and synchronization modules could be independently developed and tested. UML State Diagrams described above were used for defining the behavior of software modules. Digital automata framework ensures discrete-deterministic control by assigning predictable state transitions to all modules.

The first stage before real-time AEPP control involves developing the power plant mnemonic diagram and configuring its components. To perform these tasks, the system setup operator can interact with the component library, using drag and drop technology to transfer components from the library to the workspace. Critical user interface elements, such as the synchronization button (Sync. 1 and Sync. 2), are accessible within 1 click from the main window. In case of communication failure, the developed software logs the error within 5 seconds and notifies the operator with visual and audible alarms.

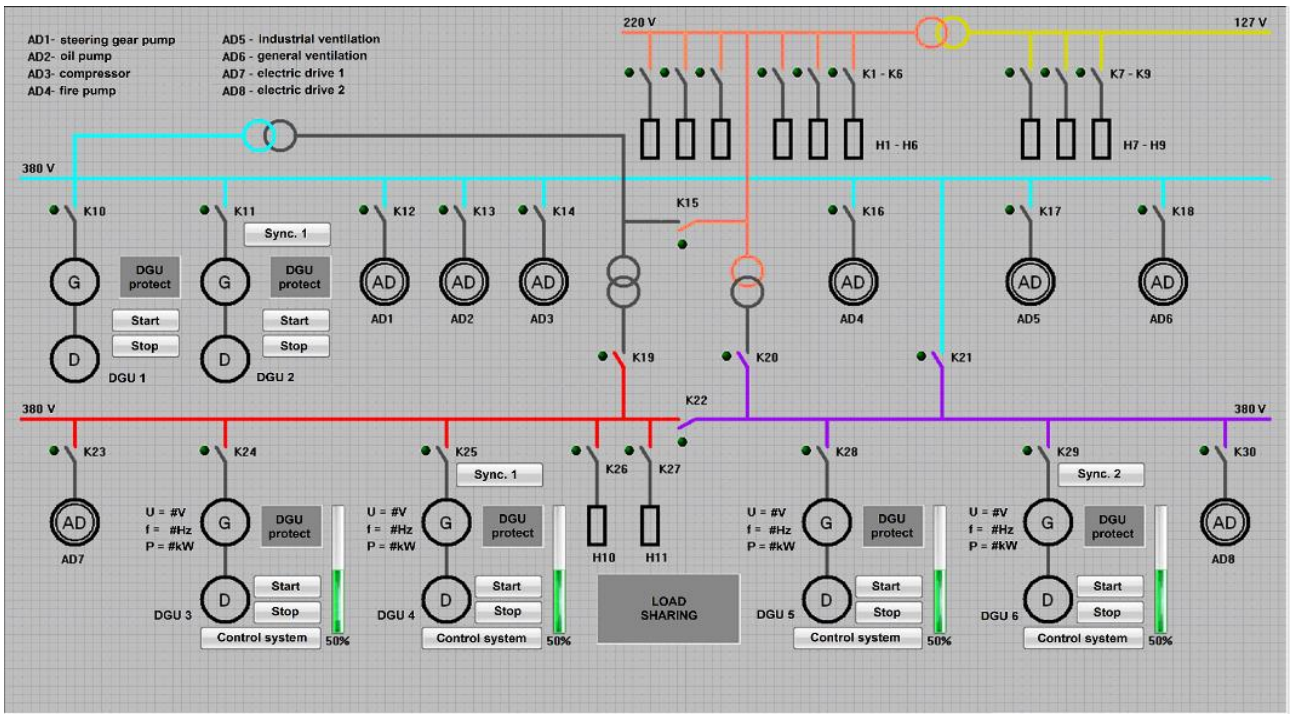

Figure 9. The main window of the software for AEPP control

One of the most important stages is software verification (testing). Software quality requirements [26, 27] is regulated by the square software quality model, based on ISO/IEC (25000-25099) standards. Functional testing is typically conducted in multiple stages, including unit, integration, and system testing. The testing process follows a structured sequence of actions: first, the criteria for test coverage are defined; next, a comprehensive set of test scenarios is developed; finally, a report is prepared documenting the test results. Unit testing is focused on verifying the individual behavior of components (e.g., DGU start/stop commands). Each unit was modeled as a finite-state machine, and its functionality was tested by simulating all possible state transitions. During the integration testing it was verified that modules interacted correctly (e.g., DGU integration with the synchronization system). Testing was conducted under both normal and failure scenarios to ensure robustness, such as communication failure recovery. System testing serves to verify the tools required by the Analysts to perform tasks such as analyzing data exchange packets, monitoring traffic statistics of communication channels, and reviewing the operational schedules of power units and electrical loads. This verification can only be conducted after the software has been in operation for a certain period in the power system control mode. Test metrics such as latency (response time), synchronization time, and load sharing efficiency were measured. Suitability is the sole functional attribute that can be assessed through this stage of testing.

The user interface shown in Figure 9 was tested for usability. The methods for evaluating practicality metrics are outlined by Hanaa et al. [28], with testing, peer review, and surveys identified as the most commonly used approaches. For the software responsible for monitoring parameters and controlling AEPPs, the survey method was selected as the primary tool for assessing usability. The survey involved 10 users to assess practicality indicators. Using the collected data, the normalized average values of these indicators were calculated through the method of summarizing and grouping the results from statistical observations [22]:

$$S_j = \frac{1}{k} \cdot \frac{\sum_{i=1}^{m} \alpha_{ij}}{m} \quad (1)$$

where, k – normalization coefficient (maximum value is 10); S_{ij} – evaluation of the j-th indicator by the i-th user; m_j – number of surveyed users. The attribute values are then calculated by applying weighting coefficients of importance and performing additive convolution [22]:

$$A_j = \frac{1}{\sum_{i=1}^{m_j} p_{ij}} \cdot \sum_{i=1}^{m_j} \left(S_{ij} \cdot p_{ij}^{(n)} \right) \quad (2)$$

where, $p_{ij}^{(n)}$ – the weight coefficient of the i-th indicator of the j-th attribute; S_{ij} – normalized average statistical value of the i-th indicator of the j-th attribute; m_j – the number of indicators of the j-th attribute. The quantitative software quality characteristics obtained using Eq. (1) and Eq. (2) are presented in Table 3.

Table 3. The software quality characteristics

Characteristics	Value
Suitability	0.91
Interoperability	0.86
Accuracy	0.81
Overall functionality score	0.86
Ergonomics	0.84
Clarity	0.89
Learning efficiency	0.83
Overall usability score	0.86

Table 4. Additional test metrics

Test Metric	Result
DGUs synchronization time	≤10 seconds
Load sharing time	≤2 seconds after DGU connection/disconnection
Parameters monitoring latency	≤1 second
Accuracy of load sharing	±2% error

The results of the developed software testing and verification carried out by Al-Suod et al. [22]. The developed software was also implemented at the enterprise LLC NVP "Inter Electro" (Mykolaiv, Ukraine), and in the specialized laboratory of the Admiral Makarov National University of Shipbuilding. Additional test metrics obtained experimentally during the use of the software are shown in Table 4.

The results confirm that the developed software met its design aims for real-time control, accuracy, and modularity. The modular verification methodology, using UML-based state modeling and digital automata, ensured the predictability of operations under dynamic conditions. The used testing approach validated both the discrete-deterministic models and the software's ability to meet the functional requirements of AEPP control. The quantitative results demonstrate the system's reliability, adaptability, and performance, addressing key gaps in existing solutions and contributing to the development of more advanced automation frameworks for AEPPs.

7. CONCLUSIONS

This research contributes to the development of adaptive ACS software for AEPPs by combining UML diagrams with discrete-deterministic models. The proposed methodology ensures predictable software behavior, smooth design-to-code transitions, and adaptability to changing system configurations.

Based on the analysis of the tasks solved by the ACS AEPP software, the requirements for its functionality were described, which made it possible to determine the operating modes of the software (scheme designer mode and the AEPP monitoring and control mode), the composition of the component library and the requirements for each component. The discrete-deterministic models of ACS AEPP software components were developed using the concept of digital automata. These models enabled the formalization of the analysis and synthesis of control algorithms, ensuring that the behavior of each software component is predictable and systematically defined. The adoption of UML state diagrams as a specification language facilitated a seamless and formal transition from system requirements to algorithmic behavior and

corresponding program code, ensuring consistency throughout the development process.

The scientific novelty of this research lies in the integration of discrete-deterministic models with UML state diagrams for AEPP control software. While previous studies have employed these methods individually, their combined application to formalize both algorithm synthesis and software verification in the context of AEPP automation is novel. This approach not only ensures a more reliable transition from design to code but also introduces a methodology for hierarchical and modular software verification, significantly improving the efficiency and accuracy of testing processes for complex, distributed systems. The resulting framework provides a robust, scalable solution that aligns with the latest advancements in control theory and automation, addressing critical challenges in the development of ACS for AEPPs. By addressing the challenges of software reliability, modularity, and real-time control, this study lays the foundation for more advanced automation solutions in AEPPs, meeting the demands of modern power generation. The methodology proposed in this work goes beyond AEPPs and has the potential to influence other domains that require adaptive control systems, such as microgrids, industrial automation, and smart grids. By demonstrating the practical use of UML in tandem with digital automata, this research contributes to best practices in software engineering for complex, distributed control systems.

ACKNOWLEDGMENT

This research supported by the Ukrainian Ministry of Education and Science and was carried out within the scientific and technical development project “Development of energy-efficient systems for generating and converting electric power for demagnetization systems of small ships” (Grant number: 0124U001522), stage 2 “Development of scientific and technical provisions for the organization of an automated control system and software for managing the processes of generation and conversion of electric power of the autonomous power plant to ensure the operation of demagnetization systems of small ships”.

REFERENCES

[1] Tkacik, M., Jadlovsky, J., Jadlovska, S., Jadlovska, A., Tkacik, T. (2023). Modeling and analysis of distributed control systems: Proposal of a methodology. Processes, 12(1): 5. https://doi.org/10.3390/pr12010005

[2] Adebayo, D., Chinedu, U. (2022). An overview of distributed generation in power plants. International Journal of Frontline Research in Engineering and Technology, 1: 027-033. https://doi.org/10.56355/ijfret.2022.1.1.0001

[3] Romashkin, M., Vlasov, V., Moshev, E. (2024). Development of software for the management of the maintenance of equipment of thermal power plants. Applied Mathematics and Control Sciences, 3: 95-108. https://doi.org/10.15593/2499-9873/2023.3.07

[4] Han, F., Zio, E. (2018). Modeling an electric power microgrid by model predictive control for analyzing its characteristics from reliability, controllability and topological perspectives. Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability, 232: 216-224. https://doi.org/10.1177/1748006X17744382

[5] Kabbara, N., Mohand, O., Nait, B., Gibescu, M., Camargo, L.R., et al. (2022). Towards software-defined protection, automation, and control in power systems: Concepts, state of the art, and future challenges. Energies, 15(24): 9362. https://doi.org/10.3390/en15249362

[6] Narmania, D., Seturidze, R., Machitidze, M., Davitaia, S., Maghradze, M. (2023). The improvement ways of SCADA system management in power energy. Economics and Business, 15(3): 152-171. https://dspace.tsu.ge/handle/123456789/2460

[7] Sayed, K., Abo-Khalil, A.G., Eltamaly, A.M. (2021). Wind power plants control systems based on SCADA system. Control and Operation of Grid-Connected Wind Energy Systems, 109-151. https://doi.org/10.1007/978-3-030-64336-2_6

[8] Al-Suod, M.M.S., Alexander, U., Dorogan, O. (2014). Monitoring and automatic control for ship power plants based logical algorithms. International Journal of Advanced Computer Research, 4(4): 966-972.

[9] Kindjock, J. (2021). Design technique for load-sharing and monitoring of a power plant using an intelligent control technique. International Journal for Research in Applied Science and Engineering Technology, 9: 1687-1698. https://doi.org/10.22214/ijraset.2021.39057

[10] Kamolov, A., Park, S. (2018). An IoT based smart berthing (parking) system for vessels and ports. In Proceedings of the International Conference on Mobile and Wireless Technology (ICMWT 2018), pp. 129-139. https://doi.org/10.1007/978-981-13-1059-1_13

[11] Singh, K., Singh, Y., Barak, D., Yadav, M., Özen, E. (2023). Parametric evaluation techniques for reliability of Internet of Things (IoT). International Journal of Computational Methods and Experimental Measurements, 11(2): 123-134. https://doi.org/10.18280/ijcmem.110207

[12] Bayer, D., Aydin, Ö., Celik, M. (2021). An ICOR approach towards ship maintenance software development. International Journal of Maritime Engineering, 160(A1). https://doi.org/10.5750/ijme.v160iA1.1044

[13] Koc, H., Erdoğan, A., Barjakly, Y., Peker, S. (2021). UML diagrams in software engineering research: A systematic literature review. Proceedings, 74(13): 1-5. https://doi.org/10.3390/proceedings2021074013

[14] Al Rababah, A. (2024). Assessing the effectiveness of UML models in software system development. International Journal of Applied Science and Research, 7(1): 13-24. https://doi.org/10.56293/IJASR.2024.5703

[15] Kazimov, T., Bayramova, T. (2022). Development of a hybrid method for calculation of software complexity. System Research and Information Technologies, 2: 32-44. https://doi.org/10.20535/SRIT.2308-8893.2022.2.02

[16] Nowakowski, W., Ciszewski, T. (2023). The development of software solutions for diagnosis railway control systems. In Proceedings of the 27th International Scientific Conference Transport Means 2023, Palanga, Lithuania, pp. 4-6.

[17] Polyuschenkov, I.S. (2022). Development of electric drive software for coordinated control in electromechanical system. Vestnik IGEU, 53-63.

https://doi.org/10.17588/2072-2672.2022.4.053-063
[18] Yegül, U. (2023). Development of an embedded software and control kit to be used in soilless agriculture production systems. Sensors, 23(7): 3706. https://doi.org/10.3390/s23073706
[19] Altaher, A. (2021). Unified Modelling Language (UML) effect on the total cost of ownership (TCO) for a software development. Iraqi Journal of Science, 207-209. https://doi.org/10.24996/ijs.2021.SI.1.29
[20] Worku, M.Y., Hassan, M.A., Abido, M.A. (2021). Power management, voltage control and grid synchronization of microgrids in real time. Arabian Journal for Science and Engineering, 46: 1411-1429. https://doi.org/10.1007/s13369-020-05062-9
[21] Debnath, S., Xia, Q., Dong, Z., Marthi, P., Marti, S., Kondabathini, A., Chakraborty, S., Saeedifard, M., Pan, J., Arifujjaman, M. (2023). Control system of multi-port autonomous reconfigurable solar power plant (MARS) & HIL platforms for design. IEEE Transactions on Sustainable Energy, 15(3): 1423-1434. https://doi.org/10.1109/TSTE.2023.3346313
[22] Al-Suod, M.M.S., Ushkarenko, O., Dorohan, O., Abdullah Eial, A., Awwad, A., Al-Quteimat. (2023). Software quality assessment technique for the autonomous power plants automated control systems. Journal Européen des Systèmes Automatisés, 56(6): 1043-1051. https://doi.org/10.18280/jesa.560614
[23] Lyaskovskiy, V.L., Bresler, I.B., Alasheev, M.A. (2021). Methodological and software tools for selecting solutions for the creation (development) of automated control systems. H&ES Research, 13: 48-59. https://doi.org/10.36724/2409-5419-2021-13-3-48-59
[24] García, F.J.B., Sierra, J.R.A., Blanco, J.G., Lázaro, A.C. (2022). The importance of self-generation of electricity through controlled recycling: A case study in West Sub-Saharan African regions. International Journal of Energy Production and Management, 7(4): 338-350. https://doi.org/10.2495/EQ-V7-N4-338-350
[25] Al-Suod, M.M., Ushkarenko, A., Dorogan, O. (2019). Power planet control and monitoring using state space diagram and multi-αgent environment. WSEAS Transactions on Computer Research, 7: 85-93.
[26] Dhivya, D., Nirmala, K. (2018). Study on integration testing and system testing. International Journal of Creative Research Thoughts, 6(2): 794-798.
[27] Lotfi, Z., Khalifi, H., Ouardi, F. (2023). Efficient algebraic method for testing the invertibility of finite state machines. Computation, 11: 125. https://doi.org/10.3390/computation11070125
[28] Hanaa, B., Noura, A.S., Hesham, H., Khaled, W. (2022). Application-based usability evaluation metrics. International Journal of Advanced Computer Science and Applications, 13(7): 84-91. https://doi.org/10.14569/IJACSA.2022.0130712