

The educational simulator as a cyber-physical system instance

Marcin Bernaś, Marcel Gańczarczyk, Jakub Krawczyk, Kamil Kwiatkowski, and Mateusz Nawieśniak

Abstract—This article proposes a cyber-physical system (CPS) implementation that brings together computation, visualization, communication, and sensor layers. The implementation was developed as part of an educational program aimed at promoting science and presenting it in an engaging manner. Consequently, the system was designed according to the cyber-physical system paradigm and implemented as a space shuttle simulation featuring several physical consoles running their own pre-programmed routines. These consoles operate as autonomous CPS subsystems. The implementation details, along with the deployment process, are presented. Finally, remarks regarding future development are provided.

Keywords—Cyber-physical system; education; sensors; wireless communication

I. INTRODUCTION

IN an era marked by rapid technological advancements and global challenges, education must evolve to prepare individuals for a dynamic work environment. The integration of action-based learning and competence development in areas such as digitalization, sustainability, and human-centered design is crucial [1]. By doing so, educational programs can align with industry needs, fostering professionals who are adept at navigating the complexities of Industry 4.0 and beyond. Therefore, there is a growing need for interactive solutions that simulate specific environments to provide practical experience. One such example is an educational simulator teaching the basics of Information Science, created under a program funded by the Polish Ministry of Education. The simulator was designed to utilize physical consoles while also operating via digital touch screens. The system is capable of providing feedback in an attractive graphical form. To achieve this, the solution was developed as a complex cyber-physical system (CPS) [2]. CPSs are an essential component of the fourth industrial revolution. They manifest themselves through the use of edge and centralized processing technologies, where multiple sensors play a crucial role. These sensors can track the progress of trainees and adapt to their skill levels. Various solutions have implemented this concept, particularly by merging gamification with digital twins and CPS [3]. A CPS is characterized by the fusion of computational elements and physical entities. Typically manifested as embedded systems, CPSs are interconnected through networks and are predominantly used

to observe and manage physical operations within a feedback loop. In this setup, the physical operations provide the data necessary for computing control signals for the system. There are many approaches to CPS implementation. For instance, a solution based on grids, transformers, and balancing was proposed in a study by Platzer [4]. However, this approach requires a grid of sensors. The proposed application focuses on the core CPS system and its supporting subsystems, extending its overall capabilities. Computer-based simulations [5] offer opportunities to train procedural competence and enhance meta-competence. These simulations, while similar to human-based scenarios, focus on self-reflection and critical thinking. Therefore, this paper proposes the creation and deployment of such a system.

II. SYSTEM STRUCTURE DEFINITION

Building upon foundational cyber-physical system architectures [2], the proposed educational simulator is formally defined as a tuple of physical and cyber elements and subsystems denoted by Σ :

$$\Sigma = (P_1^\Sigma, P_2^\Sigma, P_3^\Sigma, P_4^\Sigma, C_1^\Sigma, \Sigma_1, \dots, \Sigma_5, \Sigma_6), \quad (1)$$

where:

P_1^Σ - represents the main processing unit with GPU support,
 P_i^Σ for $i \in \{2, 3, 4\}$ - display devices (one with touch capabilities),

C_1^Σ - main system cyber layer with a system loop and event-based state changes,

Σ_j for $j \in \{1, \dots, 5\}$ - embedded systems that allow extending the simulation functionality via physical consoles,

Σ_6 - the management system that allows for remote control of the system state. The system is autonomous and contains its own subsystems.

Each of the six subsystems ($k \in \{1, \dots, 6\}$) is defined as:

$$\Sigma_k = (P_1^{\Sigma_k}, C_1^{\Sigma_k}), \quad (2)$$

where:

$P_1^{\Sigma_k}$ is the physical layer containing sensors, the processing unit, and display devices, and $C_1^{\Sigma_k}$ is the cyber layer managing information exchange. The relations within the system are complex and comprise three main types. The first type connects the physical layer with the cyber layer of the main system: $\mathcal{R}_c(P_i^\Sigma, C_1^\Sigma)$ for $i \in \{1, \dots, 4\}$. The second type connects the physical layer with the cyber layer for specific

The authors are with the Faculty of Mechanical Engineering and Computer Science, University of Bielsko-Biała, Bielsko-Biała, Poland (e-mail: mbernas@ubb.edu.pl).



subsystems $j \in \{1, \dots, 5\}$: $\mathcal{R}_c(P_m^{\Sigma_j}, C_1^{\Sigma_j})$ for $m \in \{1, 2\}$. Finally, the third type enables information exchange between the main CPS and its sub-CPS: $\mathcal{R}_c(C_1^{\Sigma_j}, C_1^{\Sigma_j})$ for $j \in \{1, \dots, 5\}$.

Each relation $\mathcal{R}_c$ allows information to flow in both directions, as presented in Fig. 1.

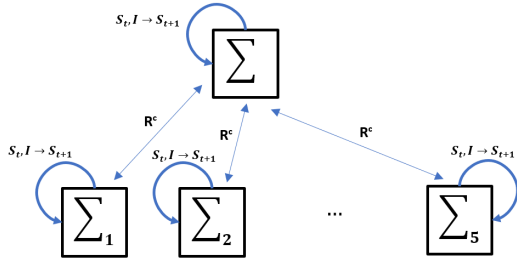


Fig. 1. CPS scheme overview.

The cyber elements utilize information I and state (S_t) at the current time step (t) to determine the next system state:

$$\Sigma : S_t, I \rightarrow S_{t+1}. \quad (3)$$

III. CYBER-PHYSICAL SYSTEM IMPLEMENTATION

The proposed system was implemented as a CPS. The core component is an educational application built on the Unity game engine [6], where physical panels are treated as supporting elements with their own functionalities. An overview of the implementation is presented in Fig. 2.

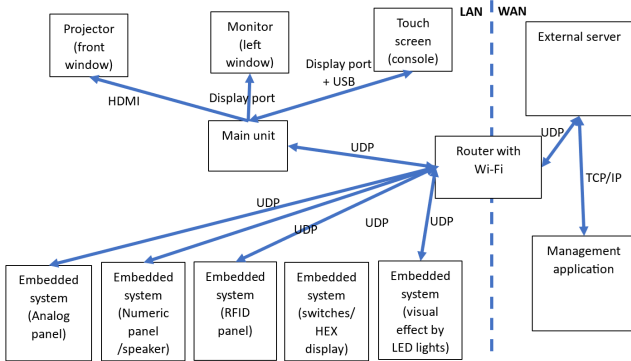


Fig. 2. Cyber-physical system implementation overview.

The main unit P_1^{Σ} is responsible for tracking the educational simulation and changing its states accordingly. It also receives information packages from other cyber-physical systems, processes them, and updates the application state. The main unit processes the simulation according to the following algorithm:

- 1) Set the current step of the simulation $S_c = S_0$.
- 2) Run routines for the current simulation step S_c and broadcast the state to $\Sigma_{1,5}^{\Sigma}$.
- 3) Obtain information I via the cyber layer from other cyber-physical systems.
- 4) Update the state S_c according to the received information and return to step 2.

The state of the simulation determines each step of the training scenario. Each state can trigger various routines,

such as presenting animations, launching tasks on a touch display, or executing separate consoles (embedded systems) with initial values. These routines monitor the information obtained either directly from sensors or via embedded systems, which determines the transition from the current state to the next one. This message loop is vital for the operation of the entire system. If the information is incomplete or the data frame is corrupted, it is instantly discarded. It is worth noting that changing simulation states can alter the set of active subroutines. Some subroutines may be present across multiple states, while others are specific to a given state. Fig. 3 illustrates the relationships between states and routines in a given scenario.

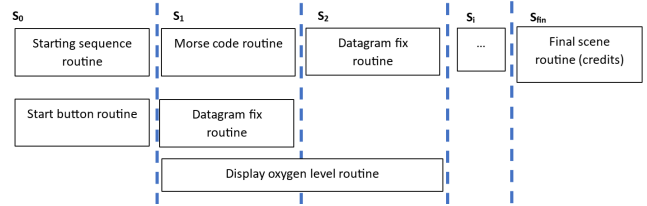


Fig. 3. Example of a scenario based on states.

The embedded systems communicate with the main system via a dedicated network. This network allows for the transmission of information I through a secure infrastructure. The border router provides internet access, but in a very restricted manner. The security features are described in the following sections. Finally, each embedded system operates as a cyber-physical system in its own right. Despite the variety of sensors implemented, they all follow a default algorithm:

- 1) Initialize system sensors, the hardware watchdog, and network communication.
- 2) Listen for information I from the main CPS.
- 3) If information I is received, process it, change the internal state of the system, and execute the corresponding routine.
- 4) Every 100 ms, send a keep-alive signal to the main system.
- 5) Reset the watchdog routine and return to step 2.

In addition to the routines implemented for a given state, each system features a watchdog security mechanism that verifies if the network connection is active. Furthermore, the watchdog checks if the main loop is operating correctly. In the event of a failure or a library error, the system automatically reboots after 10 seconds. The keep-alive signal also allows the main system to monitor the condition of all subsystems and apply corrective measures if needed. The primary routines in the support systems periodically send their status to the main system to prevent the possibility of losing the current device state.

A. Main system implementation

To enhance the learning experience, the educational simulator was themed around a spaceship stranded in space. The primary cyber-physical system routine is responsible for rendering a digital model of the space surrounding the

spacecraft for users participating in the educational simulation. This was achieved by developing an application using the Unity game engine [6]. Leveraging modern special effects and GPU acceleration from the main system, the application renders two separate views of the simulation, presenting them as if viewed through a spaceship's windows. In addition to the constantly rendered graphics, the application includes interactive sound effects and music that dynamically react to events during the user's experience. To further enhance the visual immersion, an additional cyber-physical subsystem is responsible for controlling the color and intensity of LED lights, reacting to user actions in real time. This captivating display not only engages users but also facilitates learning in a fun and interactive manner. The visual overview of the simulation is presented in Fig. 4.

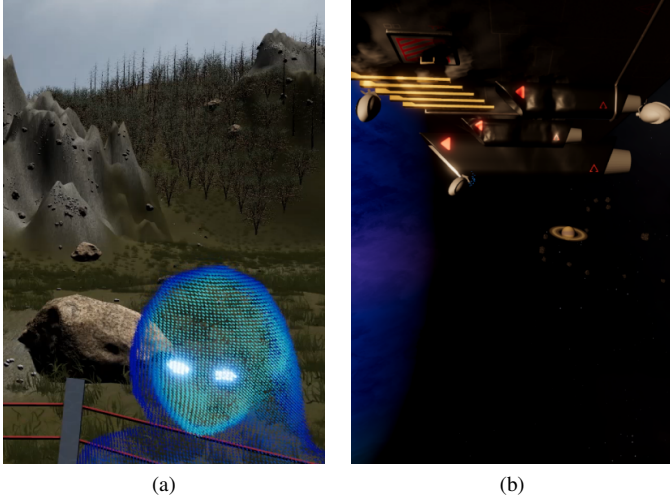


Fig. 4. Render of the spaceship's front (a) and side (b) windows.

The Unity application was divided into several puzzles (functioning as separate routines); each puzzle verifies the knowledge acquired by users from different sections of the educational simulator. Thanks to the system's modular structure, each puzzle can be disabled depending on the initial conditions or via a remote management system command, represented as a given state S_i . This ensures the stability of the overall application in case of a failure in any individual puzzle routine. Users' progress in solving each puzzle is tracked by the main CPS system based on inputs transmitted by other subsystems through the $\mathcal{R}_c$ relation. The application validates the data and updates the state by either marking the current task as completed or enabling new ones. When a new routine is enabled, the process checks the dependent sub-CPS puzzles and changes their states using dedicated commands. Each subsystem $\Sigma_{1,5}$ can be triggered by a "Start" command, followed by custom data if necessary. The information frame is sent directly to the corresponding sub-CPS. Similarly, the main routine of a sub-CPS can be disabled upon task completion by sending an "End" information frame. Apart from managing puzzle dependencies, the application updates the database of the remote management system Σ_6 . This update includes the

state of every puzzle, starting times, initial conditions, the current system time, and any system errors encountered. The update is committed whenever any of the aforementioned data changes (excluding the continuous system time). An example of a puzzle displayed on the touch screen is presented in Fig. 5.

Fig. 5. A puzzle verifying users' knowledge of an internet datagram.

B. Relation $\mathcal{R}^c$ implementation

The data transmission between the main CPS and its subsystems was achieved by sending UDP packets over a self-hosted local Wi-Fi network. For security reasons, this network was password-protected and its SSID was hidden, ensuring that only the educational simulator's cyber elements could send and receive information. To determine the packet destination, all systems maintain a lookup table mapping CPS devices to their respective IP addresses.

TABLE I. List of IP addresses for all CPS devices connected to the network.

Device	IP Address
Router	192.168.1.1/24
Main computer	192.168.3.35/24
Hex Controller	192.168.3.3/24
Potentiometers Controller	192.168.3.5/24
Morse Controller	192.168.3.2/24
RFID Controller	192.168.3.4/24
LED Controller	192.168.3.36/24
Mobile Devices with app	DHCP assigned

To standardize the communication, subsystems format their information frames I as JSON objects (defined standard):

```
1 {"ButtonStates":[], "PotentiometerStates": [],
2  , "\"AdditionalString\":" + CODE + "\""}

```

1) *The Main Server:* The main server acts as the heart of the entire system, operating on the main computer and serving as the central hub. It stores the configuration of each controller (IP, Name) and contains functions that allow it to execute specific actions regarding particular boards. When a user interaction triggers a function that initializes communication with a specific controller, the central hub sends instructions to the target station and then switches to listening mode. In listening mode, the main computer waits for response packets from the activated controllers. Upon receiving a packet, the

sender's IP is analyzed, and depending on the packet's content, the corresponding action is taken.

boolean flag. This flag informs the controller whether it should process the main script responsible for the puzzle logic. After initializing all variables, the station switches to listening mode.

```

1 ControllerInfo? deserializedMessage =
2   DeserializeMessage(message);
3 ControllerName? name = null;
4 name = Controllers.FirstOrDefault(controller =>
5   controller.Address.ToString().Split(':')[0] ==
6   endPoint.Address.ToString().Split(':')[0])?.Name;
7 if (name != null && deserializedMessage != null)
8 {
9   lock (ExecuteOnMainThread)
10  {
11    ExecuteOnMainThread.Enqueue(() =>
12      ReceiveMessage(deserializedMessage,
13        (ControllerName)name));
14  }
15 }

```

2) *Microcontrollers*: The microcontrollers are programmed so that immediately after powering on, they establish a connection with the Wi-Fi network using a pre-configured SSID and password. The communication is established by the following routine:

```

1 void setWiFiConnection() {
2   Serial.begin(19200);
3   Serial.print("Connecting to WiFi");
4   WiFi.begin(ssid, password);
5   while (WiFi.status() != WL_CONNECTED) {
6     delay(500);
7     Serial.print(".");
8   }
9   Serial.println(WiFi.localIP());
10  Udp.begin(port);
11 }

```

The embedded system is based on an ESP32-WROOM-32 microcontroller module (MCU). It is a powerful, versatile Wi-Fi, Bluetooth, and BLE (Bluetooth Low Energy) module designed for a wide range of applications, from low-power sensor networks to demanding tasks such as voice encoding, music streaming, and MP3 decoding [7]. A prototype of the module in a console is shown in Fig. 6.

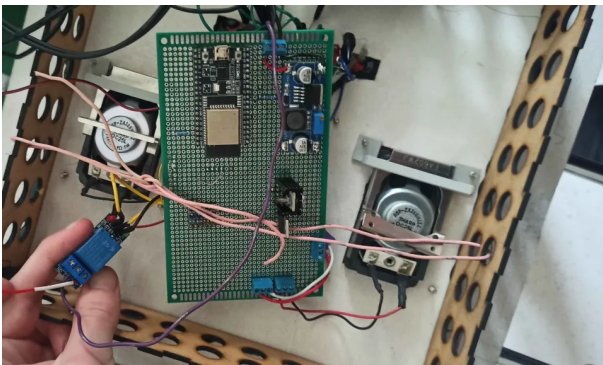


Fig. 6. Implementation of a sub-CPS controlling a locker, Morse code module, and numeric keypad.

The ESP board connects to external input and output elements—such as switches, LEDs, analog potentiometers, keypads, speakers, and door drivers—via digital and analog ports. Once the controller successfully establishes a connection, it initializes its base variables, including the **isQuizWorking**

```

1 void listen() {
2   String unityRequest = "";
3   int packetSize = Udp.parsePacket();
4   if (packetSize) {
5     Serial.print("Received packet of size ");
6     Serial.print(packetSize);
7     Serial.print(" from ");
8     IPAddress remoteIp = Udp.remoteIP();
9     Serial.print(remoteIp);
10    Serial.print(", port ");
11    Serial.println(Udp.remotePort());
12
13    if (remoteIp.toString() != MainComputerIp) {
14      Serial.println("Wrong IP address.");
15    }
16    Udp.endPacket();
17    return;
18  }
19
20  unityRequest = Udp.readStringUntil('\n');
21  Serial.println(unityRequest);
22  Udp.flush();
23  Udp.endPacket();
24 }
25
26 if (unityRequest.startsWith("Start#")) {
27   vector<String> params = split(unityRequest, '#');
28   if (params.size() != 2) {
29     Serial.println("Received invalid request.");
30     return;
31   }
32
33   morseCode = params[1] + " ";
34   morseCodeLength = split(morseCode, ' ').size();
35   isQuizWorking = true;
36   isMorsePlaying = true;
37 } else if (unityRequest.startsWith("End")) {
38   Serial.println("End of puzzle");
39   isQuizWorking = false;
40
41   morseCode = "";
42   morseCodeLength = -1;
43   butValues[MAX_ANSWER_LENGTH] = {};
44   butNextIndex = 0;
45   theaterChase(strip.Color( 255, 0, 0), 25);
46 }
47
48 }

```

Listening is implemented in a way that the device accepts only UDP packets originating from the main computer's IP address. If any other device attempts to send a UDP frame, the controller ignores it and logs an error in the debug console ("Wrong IP address."). Upon receiving a UDP frame from the correct IP address and verifying that the payload length is greater than zero, the program parses the content. Depending on the prefix of the message, appropriate logic is executed. This data dictates which part of the code the controller should run. In the provided Morse code controller example, there are two primary state branches. The first one, triggered by "End", instructs the controller to terminate the execution of the main code block. The second one, "Start", commands the board to initialize necessary variables and begin executing the puzzle logic. Upon receiving the "Start" instruction, the microcontroller transitions to a state where it continuously reads analog inputs and sends the data back to the main computer. Data exchange between all sub-CPS units and the main CPS uses the unified JSON structure (defined standard), ensuring seamless integration with the models in the Unity application. This data sending process is implemented as follows.

```

1      String message = "{\"ButtonStates\":[";
2
3      message += "], \"PotentiometerStates\": []";
4      message += ", \"AdditionalString\": \"\" + CODE + \"\"";
5      sendToCheck = true;
6      Serial.println(message);
7      Udp.beginPacket(MainComputerIp, port);
8      Udp.println(message);
9      Udp.endPacket();
10     //isSendToCheck = true;
11     curK=0;
12     CODE = "00000";
13     theaterChase(strip.Color( 0,   0, 255), step);
14     clearLEDS();

```

Data is transmitted at specific intervals, mitigating the risk of information loss due to temporary network congestion or external factors.

3) *Communication using a dedicated router:* The communication layer was implemented using a dedicated router that manages the local network. This setup allows for isolated testing and validation of individual system components. Despite the Wi-Fi security features, the network firewall permits communication exclusively between explicitly allowed IP addresses. Given the specific requirements of the application, UDP datagrams were chosen for communication [8]. Although UDP is inherently unreliable and lossy, its primary advantage is that it does not require establishing and maintaining a persistent data stream between the microcontrollers and the main server. This prevents scenarios where the server might become blocked waiting for a response, thereby ignoring messages from other devices. A two-way protocol like TCP requires connection handshakes and acknowledgments; if a packet is dropped, the sender could be forced into a prolonged waiting state until a timeout occurs. With UDP (one-way communication), the sender simply dispatches a frame to the receiver's IP address without waiting for an acknowledgment, immediately returning to listening and processing subsequent incoming packets.

C. Remote system management

The CPS was designed to allow administrators to manually configure, control, and monitor the uptime of all components, ensuring that the cyber-physical systems remain fully operational. The management tool was developed as a mobile application using the Flutter framework [9]. This framework was selected because it enables cross-platform compilation, making the application readily available for both iOS and Android devices.

Fig. 7a: The main screen allows the user to initiate or restart the game, enter the player's name, and select the difficulty level.

Fig. 7b: The puzzle management screen provides monitoring and control over active puzzles, offering options to bypass tasks, remotely trigger electronic locks, and skip the introductory cutscene.

Fig. 7c: The health check screen displays an overview of the operational status (uptime) and the assigned IP addresses of all active controllers.

The application architecture enables administrators to check the status of the controllers even when located outside the local LAN. Health check telemetry was periodically sent to a remote server located in Warsaw, Poland, and retrieved by the mobile app via an API running on NGINX. However, for security reasons, active control functionalities (like skipping puzzles or unlocking doors) remained strictly accessible only to authenticated users connected to the local Wi-Fi network.

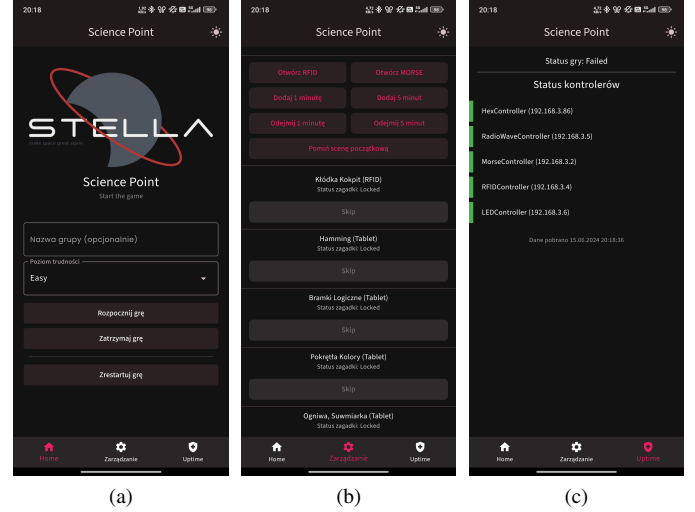


Fig. 7. Screenshots illustrating the user interface of the management application.

IV. DEPLOYMENT OF THE CPS SYSTEM

The CPS was deployed at the Gemini Park shopping center in Bielsko-Biała, Poland. It was constructed as an escape room-style puzzle experience, available to the public from January 26, 2024, to March 24, 2024. A view of the installation is presented in Fig. 8. To accurately mimic the interior of a futuristic space shuttle, the circuit boards were embedded into custom cabinets and the main desk, which also featured a central touchscreen. In addition to being aesthetically pleasing, the cabinets physically secured the electronic components, ensuring users could not accidentally damage them. The room was equipped with four physical puzzle controllers, a central touchscreen, a projector rendering the front window view, and an LCD monitor displaying the side window view. The use of a wireless network for microcontroller communication allowed for swift and flexible deployment of the hardware. However, this approach presented distinct challenges, as maintaining a highly stable connection was critical for seamless operation. Due to the site's network infrastructure, the primary router had to be placed in a separate room to gain internet access, forcing the main processing unit to connect via Wi-Fi. This proved to be a point of failure during the final two weeks of the project when the main computer's network adapter malfunctioned, rendering the CPS unresponsive. To restore functionality, the router was relocated near the main unit and connected via an Ethernet cable. While this brought the local CPS back online, it severed the internet connection, disabling the remote health check API. Another issue encountered with the wireless setup

was occasional packet loss between the microcontrollers and the main processing unit. This was mitigated by implementing the aforementioned watchdog timers, which monitored connectivity and automatically reset stalled network interfaces. User reception varied based on age, prior experience with escape rooms, and group size. The immersive, multi-device setup successfully kept most users engaged, even in larger groups. Younger audiences were particularly captivated by the visual effects, while older, more experienced participants appreciated the logical difficulty and technical craftsmanship. The overall completion rate was relatively low, which encouraged some participants to return for a second attempt, though it also left a few feeling overwhelmed. Despite these minor technical and balancing challenges, the project received overwhelmingly positive feedback. In total, the simulator was tested by over 1,000 people across various group configurations (Fig. 9).



Fig. 8. Fully deployed room with circuit boards built into custom cabinets (view 1).



Fig. 9. Fully deployed room showcasing the simulator environment (view 2).

V. SUMMARY

The proposed cyber-physical system (CPS) implementation was rigorously tested during a two-month continuous deployment, followed by several one-day events across the Silesia region. Thanks to its modular and well-designed software

structure, the solution can be easily adapted for entirely new educational scenarios. Each cyber-physical node responds to state changes based on strict boundary conditions. The system's high potential was proven by its reliability, maintaining functionality throughout the lengthy exhibition. If a specific subsystem suffered a hardware failure, the software allowed administrators to skip the associated states, ensuring the overall simulation could continue uninterrupted. A key advantage of this architecture is its self-diagnostic routines, enabling subsystems to automatically recover from network drops or software hangs. Furthermore, damaged physical panels could be quickly swapped without rewriting the core logic. The isolated network infrastructure ensured communication reliability and prevented external interference during public operations. Ultimately, the proposed framework is highly adaptable for other educational or simulation environments that require modular, interactive hardware. The consistent engagement and positive user feedback underscore its value in modern learning contexts. For future iterations, enhancing the security protocols at the border router level should be prioritized. Additionally, further standardizing the embedded software could yield a universal framework for similar cyber-physical educational tools. This is particularly relevant in educational research, where interactive simulators and gamification represent highly effective teaching mediums [10].

VI. ACKNOWLEDGEMENT

This work was financed by the Polish Ministry of Education under the "Social Responsibility of Science II" program. The research and development were conducted by the RESET science club at the University of Bielsko-Biala.

REFERENCES

- [1] C. Pacher, M. Woschank, B. M. Zunk, and E. Gruber, "Engineering education 5.0: a systematic literature review on competence-based education in the industrial engineering and management discipline," *Production & Manufacturing Research*, vol. 12, no. 1, p. 2337224, 2024.
- [2] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for Industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.
- [3] FAITH EC Project, "Enhancing cybersecurity training efficacy: A comprehensive analysis of gamified learning, behavioral strategies and digital twins," *FAITH European Commission Research Framework*, 2024, accessed: Oct. 20, 2024. [Online]. Available: https://faith-ec-project.eu/wp-content/uploads/2024/09/Enhancing_Cybersecurity_Training_Efficacy_A_Comprehensive_Analysis_of_Gamified_Learning_Behavioral_Strategies_and_Digital_Twins.pdf
- [4] A. Platzer, *Logical Foundations of Cyber-Physical Systems*. Springer International Publishing, 2018.
- [5] G. Egonsson and M. Israelsson, "Computer-based simulations in social work education: a scoping review," *Research on Social Work Practice*, vol. 34, no. 1, pp. 41–53, 2024.
- [6] Unity Technologies. (2023) Unity. Game development platform. Accessed: Oct. 20, 2024. [Online]. Available: <https://unity.com/>
- [7] Espressif Systems, *ESP32-WROOM-32 Datasheet*, 2023, accessed: Oct. 20, 2024. [Online]. Available: <https://www.espressif.com/>
- [8] B. E. Bekele, K. Tokarz, N. Y. Gebeyehu, B. Pochopień, and D. Mrozek, "Performance evaluation of UDP-based data transmission with acknowledgment for various network topologies in IoT environments," *Electronics*, vol. 13, no. 18, p. 3697, 2024.
- [9] Google. (2021) Flutter. Accessed: Oct. 20, 2024. [Online]. Available: <https://flutter.dev>
- [10] L. Sun, Z. Guo, and L. Hu, "Educational games promote the development of students' computational thinking: a meta-analytic review," *Interactive Learning Environments*, vol. 31, no. 6, pp. 3476–3490, 2023.