1. Piotr POWROŹNIK[1], 2. Filip OPŁOTNY[2], 3. Mateusz KOMAROWSKI[2], 4. Igor KOROPIECKI[2], 5. Krzysztof TURCHAN[2], 6. Krzysztof PIOTROWSKI[2]

University of Zielona Góra, Institute of Metrology, Electronics and Computer Science (1), IHP - Leibniz-Institut für innovative Mikroelektronik, Frankfurt (Oder), Germany (2)
ORCID: 1. 0000-0001-7485-9959; 2. 0009-0000-0889-1096; 3. 0009-0008-9002-073X; 4. 0000-0002-4405-5180; 5. 0000-0001-6080-2468; 6. 0000-0002-7231-6704

doi:10.15199/48.2024.12.25

Selection of database for efficient energy management in a power grid

***Abstract**. Database selection in a measurement and control system is based on criteria that ensure system stability, reliability, data security, scalability, and lower infrastructure costs. Efficiency is also crucial, guaranteeing GUI responsiveness and enabling complex data analysis. A data management solution utilizing smartDSM middleware was outlined in the article. This Java-based software facilitates CRUD operations and easy database engine addition. Reflectivity-based modularity streamlines implementation. The solution's effectiveness is confirmed by test results.*

Streszczenie.** Wybór bazy danych w systemie pomiarowo-sterującym opiera się na kryteriach zapewniających stabilność, niezawodność, bezpieczeństwo danych, skalowalność i niższe koszty infrastruktury. Istotna jest również efektywność gwarantująca responsywność GUI i umożliwiająca złożoną analizę danych. W artykule przedstawiono rozwiązanie do zarządzania danymi z zastosowaniem middleware smartDSM. Jest to oprogramowanie w Javie, ułatwiające operacje CRUD i łatwego dodawania silników bazodanowych. Modularność oparta na mechanizmie refleksji usprawnia implementację. Skuteczność rozwiązania potwierdzają wyniki testów. **(Wybór bazy danych dla efektywnego zarządzania energią w sieci elektroenergetycznej)

Keywords: energy management, middleware, database, graphical user interface.
Słowa kluczowe: zarządzanie energią, oprogramowanie pośredniczące, baza danych, graficzny interfejs użytkownika.

Introduction

The advancement of information technology (IT) is a dynamic and continuously evolving process that impacts nearly every aspect of our lives, including how we generate, distribute, and consume electricity. In the context of energy management in power grids, IT plays an increasingly crucial role, becoming an indispensable tool for ensuring the efficiency, reliability, and security of energy systems. IT enables the real-time collection and analysis of data from sensors deployed throughout the power grid. This data can be utilized to monitor the network's condition, identify potential issues, and optimize the energy flow. IT facilitates the implementation of automation systems that automatically respond to changes in the grid, ensuring the stability and reliability of energy supply. Furthermore, IT systems enable the development of smart grids (SGs), which integrate various energy sources, storage systems, and consumer devices, enabling intelligent energy management and adaptation to current needs [1]. They also play a significant role in designing demand management systems that encourage users to consume energy rationally based on the grid's current requirements. Energy management software facilitates easy monitoring of energy consumption, identification of wasteful areas, and the implementation of measures to eliminate them.

In today's data-driven world, nearly every process, program, application, and website generates data. Data management plays a critical role in computer science and related fields. The market offers a wide range of data management tools, and one of them is middleware [2, 3]. Middleware is software that connects different systems and applications, regardless of their origin, allowing them to exchange data smoothly. Middleware hides the complex technical details of communication and system integration from the user, providing simple and consistent APIs. Middleware can offer additional features such as user authentication, access authorization, transaction management, and event logging, which make it easier for developers to create applications. Middleware facilitates the integration of different systems and applications, enabling them to collaborate and exchange information in a consistent and efficient manner. Middleware allows for easy addition of new systems, applications, and components to existing infrastructure without the need to modify existing code, which increases the flexibility and scalability of solutions. These features make middleware a valuable tool for developers and IT system administrators who want to create efficient, scalable, and integrated solutions.

User-friendly access to energy management functionalities within middleware can be achieved through a well-designed graphical user interface (GUI). Research has consistently highlighted the benefits of investing in GUI development [4, 5]. Authors have emphasized the primary goal of GUI research as understanding user needs and behaviors to design interfaces that are easy to use, intuitive, and enjoyable to interact with. A crucial criterion for GUI design is to enhance user productivity. A well-designed GUI can empower users to perform tasks more quickly and efficiently, leading to increased overall productivity. Additionally, GUI research plays a vital role in identifying and eliminating potential user errors, thereby enhancing system safety and reliability. Furthermore, GUI design considerations can extend to accessibility, ensuring that interfaces are inclusive and usable by individuals with disabilities.

The importance of conducting thorough GUI research has been firmly established in the literature [6, 7]. This research encompasses a range of methodologies, including user observation, interviews, usability testing, and other data collection techniques, to gain insights into how users interact with interfaces. Psychological experiments and other scientific methods are also employed to investigate the impact of various factors on user behavior. Computer models that simulate user thought processes and actions are implemented to predict their behavior within the interface. Additionally, a set of heuristics is employed to identify potential usability issues.

Databases are the primary data sources for GUIs. The literature emphasizes that database efficiency is a crucial criterion when selecting a specific database version. This criterion is paramount in ensuring faster GUI operation and responsiveness, enabling the scalability of the IT solution to

handle growing data volumes and user traffic [8]. This, in turn, leads to reduced IT infrastructure costs. A well-chosen database also enhances the reliability and stability of IT systems. Particularly in commercial applications, it elevates data security and enables more sophisticated data analysis.

This article presents a data management solution utilizing smartDSM middleware [9]. SmartDSM middleware is a Java-based software module designed for data management. The module's primary objective was to establish a codebase centralizing all CRUD (CREATE, READ, UPDATE, DELETE) operations within a well-defined set of classes and interfaces. This facilitates the effortless implementation of support for additional database engines. The module's modularity is built upon a reflection mechanism, eliminating the need for developers or users to recompile code. The article further delves into the presentation of energy management-related data on a GUI.

Description of the system based on smartDSM middleware

Figure 1 illustrates an exemplary system architecture built upon smartDSM middleware. Each smart appliance (SA) device runs a local instance of smartDSM middleware. This software is responsible for implementing measurement and control functionalities. SmartDSM middleware on the SA collects data related to energy consumption, for instance. It can also receive control data, such as commands to turn on, off, or reduce energy consumption.

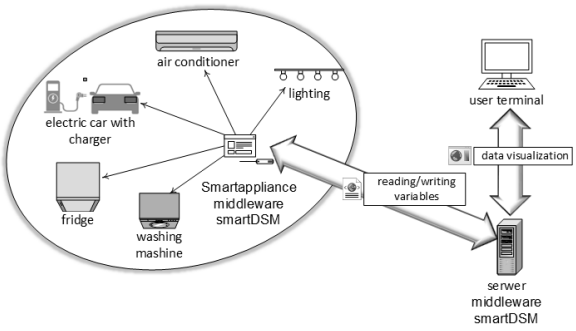

Fig.1. Architecture of the system based on smartDSM middleware

Data pertaining to device power consumption, for instance, is transmitted to the higher-level smartDSM middleware server via designated variables. Figure 2 illustrates example variable values in JSON format.

```
{   "VOLTAGE": [
      230.12
    ],
    "ACTIVE_POWER": [
      2026.02
    ],
    "REACTIVE_POWER": [
      205.04
    ],
    "FREQUENCY": [
      50.34
    ],
    "name":
"WASHING_MACHINE",
    "POWER_FACTOR": [
      0.99
    ],
    "CURRENT": [
      8.85
    ],
```

Fig. 2. Example variable values for a system based on smartDSM middleware

Monitoring the operation of individual SAs and the overall power grid is facilitated by visualizing various energy parameters on a user terminal using a GUI. SmartDSM middleware enables the implementation of a universal service that can operate within the smartDSM middleware server environment. This service communicates with a designated server by invoking smartDSM middleware server layers. The universal service for intelligent devices facilitates communication between devices and the main server. A key feature of the smartDSM middleware environment is its ability to seamlessly integrate new devices into the existing infrastructure. This necessitated a configurable and dynamic interface to accommodate newly added devices. Data from these devices is then channeled to the created user interface.

Data management in smartDSM middleware

Relational databases are among the most widely recognized types of databases. NoSQL databases [10] have emerged as a popular alternative to relational databases. The vast array of database engines available in the market can make selecting the right one a challenging task. This process necessitates careful consideration and understanding of the key factors that influence effective data management. Therefore, it is crucial to focus on various aspects such as performance, security, and ease of use. However, the abundance of database solutions and their diverse specifications necessitate narrowing the scope to a select few engines. This article delves into an analysis of three database engines: MariaDB, SQLite, and MongoDB. All three are open-source software, which implies that they share the inherent benefits of open-source software, including enhanced security, reliability, transparency, and performance. Table 1 shows a comparison of selected features of the three database engines. The basic features that were compared were the database type, data structure, transaction support, license and server.

Table 1. Comparison of selected properties for three database engines

Feature	*mariaDB*	*SQLite*	*mongoDB*
Database Type	Relational	Relational	non-relational
Data Structure	Tables	Tables	Documents
Transaction Support	Yes	Yes	Yes, since version 4.0
License	Open Source	Open Source	Open Source
Server Requirement	Yes	No, local base	Yes

The selection of a specific database engine is driven by the project's unique characteristics and requirements for versatility, performance, and functionality. Each of the shortlisted engines possesses distinct features and operational nuances.

The choice of these engines was guided by their diverse attributes and application possibilities. The primary objective was to enhance the versatility and performance of smartDSM middleware. The selected database engines fulfill these criteria by offering both serverless and server-based operation, as well as NoSQL database capabilities. Additionally, the chosen engines enjoy widespread popularity and meet performance expectations. In the context of databases, a Database Abstraction Layer (DAL) serves as an intermediary between an application and various database systems. It enables developers to write code independent of the specific database type, facilitating application portability and enhancing flexibility.

The database module with an abstraction layer centralizes all database operations in one location,

improving code readability and simplifying developer tasks. A key challenge was ensuring consistent operation across relational databases (SQLite and MariaDB) and a NoSQL database (MongoDB). These database types differ in their query languages, data schemas, and Java drivers, necessitating the implementation of specialized programming solutions. The database abstraction layer proves to be an invaluable tool for developers striving to create portable, maintainable, and flexible database systems. Data normalization in a database is also a significant issue. By eliminating redundancy, normalization can significantly decrease the size of a database. However, this reduction may come at the cost of increased read and write times. In the database implementation discussed in this article, adding new data to MariaDB, SQLite, and MongoDB led to respective database size increases of 16.39kB, 4kB, and 4.1kB.

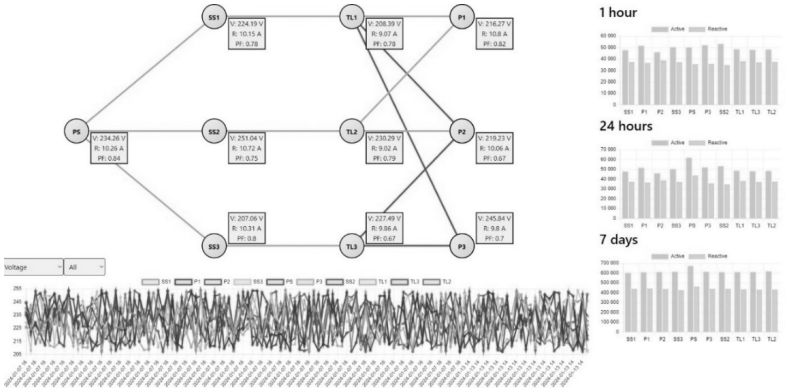

Fig.3. GUI Implementation

Data visualization with smartDSM middleware

To ensure the full functionality of the GUI, a well-designed and implemented backend is essential. In the context of smartDSM, these backend programs, referred to as services, enable connection to the smartDSM middleware server and execution of predefined functionalities. Leveraging Java and the Spring Boot framework for microservice implementation on the backend facilitated rapid and reliable service creation and data transmission logic handling. The combination of Vue.js and Node.js proved instrumental in implementing a responsive user interface. Vue.js, a popular JavaScript framework, enabled the creation of a dynamic and interactive UI, while Node.js, a JavaScript runtime environment, facilitated server-side rendering and real-time communication capabilities. The implemented GUI serves as a central hub for monitoring and controlling the energy network for connected devices. It provides users with a comprehensive overview of energy parameters, enabling informed decision-making and efficient energy management.

The designed GUI provides the required functionalities [11], including a graphical representation of the network topology, a clear presentation of energy data, and the ability to configure the network topology. The applied technologies ensure the ease of use, performance, and scalability of the interface. The Chart.js library is used for data visualization, Bootstrap ensures a responsive design, and Axios facilitates communication with the server. The Mitt and vue-native-websocket libraries are responsible for component reactivity and WebSocket communication, while the Simple Text Oriented Messaging Protocol JavaScript (STOMP.js) enables communication using the STOMP protocol. Figure 3 depicts the Vue.js component-based GUI, designed to display real-time energy data. Each component serves a specific function, fostering a user-friendly interface.

Experimental results

The database module was validated using 150 unit tests (50 per database) and 15 load tests (1000 repetitions each). Average response times for key methods are presented in Figure 4.

NoSQL databases, particularly MongoDB, demonstrate superior performance when handling large workloads, especially those involving the creation of numerous records. Their engines exhibit comparable efficiency when modifying large volumes of data with a single command. Consequently, NoSQL databases emerge as a compelling choice for applications demanding high performance under intense load, particularly when frequent record creation is a primary requirement. However, it is crucial to carefully consider the specific characteristics of the application and select the database that best aligns with its needs.

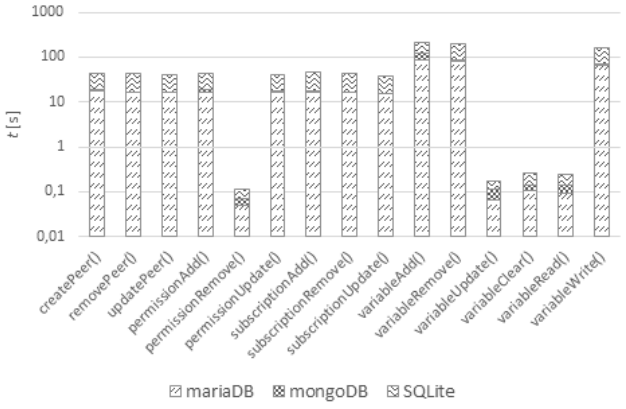

Fig.4. Average execution times of methods

The functionality of the energy network management application's backend has been verified through a wide range of tests. The backend components responsible for generating and saving energy data to the database as well as reading data from the database and making them available to the user interface were tested. Tests for the GUI were carried out using Jest.js and Babel.js libraries. The main areas of GUI testing included: correct component rendering, user interaction support, data processing and communication with the backend.

To further assess the performance of the smartDSM middleware server, the execution times of selected database access tasks were evaluated. Six test scenarios (sc) were defined, each focusing on specific data retrieval and processing operations. These scenarios (sc_1-sc_3) measure the response times for retrieving a single variable with varying numbers of readings: 10 readings (sc_1), 120 readings (sc_2), and 365 readings (sc_3). These scenarios

represent the retrieval of multiple consecutive variable values from the smartDSM middleware server. These scenarios (sc_4-sc_6) evaluate the time taken to calculate the total energy consumption for different time intervals: one hour (sc_4), one day (sc_5), and one week (sc_6). These scenarios represent internal operations within the smartDSM middleware server, involving the aggregation of energy consumption data over specified time periods. The performance tests were conducted for two cases: 8 SAs and 45 SAs in a household. This allowed for the evaluation of performance under varying levels of data complexity.

Figure 5 presents the average response times for operations executed on the smartDSM middleware server for households with 8 ($\overline{x}_{SA_8}$) and 45 ($\overline{x}_{SA_{45}}$) SAs, respectively.

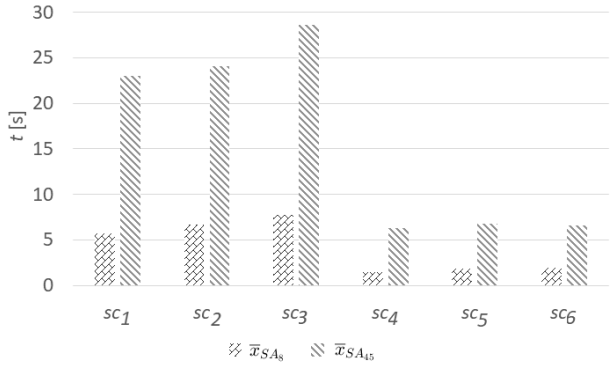

Fig.5. Average response times for operations executed on the smartDSM middleware server

Based on the results obtained for all scenarios (sc) depicted in Figure 5, it can be observed that the value of $\overline{x}_{SA}$ increases with a growing number of SAs. This suggests a correlation between the number of SAs and the overall processing time required for database operations. Furthermore, the conducted experiments revealed that performing mathematical operations directly on the smartDSM middleware server proves to be more efficient than retrieving individual variable values separately for subsequent mathematical computations. This highlights the advantage of minimizing data transfers and utilizing server-side processing whenever possible.

Conclusions

This article presents the results of implementing a list of methods for database communication based on the smartDSM middleware source code. The purpose of these methods was to provide the ability to perform all CRUD (Create, Read, Update, Delete) operations.

Based on this list, the required functionalities of the database module were identified and implemented in Java. This interface, together with the implemented abstract class and database classes, forms the core structure of the database module.

Database connection for relational databases is realized using JDBC drivers. For the non-relational MongoDB database, a separate official driver is used. The MongoDB database class required the adaptation of some methods to the specific query syntax of MongoDB. The methods that required this were appropriately adapted, taking into account the requirement for uniformity between database engines.

The article also presents the results of implementing the frontend for the power grid management application. A client-server architecture was used, where the backend simulates the operation of devices and collects data. The GUI interface visualizes the data in charts and a connection tree. Unit tests were conducted. There is potential for further development of the application, including adding a function to view historical data and configure charts.

Acknowledgements

The results of the research presented in this article were partially realized during the first author's research stay at the IHP - Leibniz Institute for High Performance Microelectronics, Germany.

***Authors**: dr inż. Piotr Powroźnik, Uniwersity of Zielona Góra, Institute of Metrology, Electronics and Computer Science, Podgórna 50, 65-246 Zielona Góra, E-mail: p.powroznik@imei.uz.zgora.pl; inż. Filip Opłotny, Uniwersity of Zielona Góra, Institute of Metrology, Electronics and Computer Science, Podgórna 50, 65-246 Zielona Góra; inż. Mateusz Komarowski, Uniwersity of Zielona Góra, Institute of Metrology, Electronics and Computer Science, Podgórna 50, 65-246 Zielona Góra; mgr inż. Igor Koropiecki, IHP - Leibniz-Institut für innovative Mikroelektronik, Im Technologiepark 25, 15236 Frankfurt (Oder), Germany, E-mail: koropiecki@ihp-microelectronics.com; mgr inż. Krzysztof Turchan, IHP - Leibniz-Institut für innovative Mikroelektronik, Im Technologiepark 25, 15236 Frankfurt (Oder), Germany, E-mail: turchan@ihp-microelectronics.com; prof. dr inż. Krzysztof Piotrowski, IHP - Leibniz-Institut für innovative Mikroelektronik, Im Technologiepark 25, 15236 Frankfurt (Oder), Germany, E-mail: piotrowski@ihp-microelectronics.com.*

REFERENCES

[1] Szcześniak P., Powroźnik P., Sztajmec E., Selected voltage control methods in LV local distribution grids with high penetration of PV. *Przegląd Elektrotechniczny*, 99 (2023), nr 11, 62-65

[2] Luo Z., Li D., Wan J., Wang S., Wang G., Cheng M., & Li T., Component integration manufacturing middleware for customized production, *Advanced Engineering Informatics*, 59 (2024), 102317

[3] Souki O., Djemaa R. B., Amous I., Sèdes F., A Survey of Middlewares for self-adaptation and context-aware in Cloud of Things environment, *Procedia Computer Science*, 207 (2022), 2804–2813

[4] Hartson R., Pyla P. S., The UX Book: Agile UX Design for a Quality User Experience, *Holand: Elsevier Science*, (2018)

[5] Kantamneni S., User Experience Design: A Practical Playbook to Fuel Business Growth, *United Kington: Wiley,* (2022)

[6] Allanwood G., Beare P., User Experience Design: A Practical Introduction, *United Kington: Bloomsbury Publishing*, (2019)

[7] Marsh S., User Research: Improve Product and Service Design and Enhance Your UX Research, *United Kington: Kogan Page*, (2022)

[8] Taipalus T., Database management system performance comparisons: A systematic literature review, *Journal of Systems and Software/the Journal of Systems and Software*, 208 (2024), 111872

[9] Koropiecki I., Piotrowski K., SmartDSM: Towards User-Centric IoT Middleware Platform for Privacy-Focused Smart Systems, *IEEE International Conference on Internet of Things and Intelligence Systems (IoTaIS)*, Bali Indonesia (2023), 79-84

[10] Mehmood E., Anees T., Performance Analysis of Not Only SQL Semi-Stream Join Using MongoDB for Real-Time Data Warehousing, *IEEE Access*, 7 (2019), 134215-134225

[11] Xiao F., Lu T., Ai Q., Wang X., Chen X., Fang S., Wu Q., Design and Implementation of a Data-Driven Approach to Visualizing Power Quality, *IEEE Transactions on Smart Grid*, 11(5) (2020), 4366 - 4379