ISSN 1846-6168 (Print), ISSN 1848-5588 (Online)
https://doi.org/10.31803/tg-20240514045737

Original scientific paper
Received: 2024-05-14, Accepted: 2024-07-18

Design and Implementation of a DLMS Server with a Multi-threaded Architecture for AMI Systems

Gwang Hyeon Kim, Yeong Rak Seong*, Ha Ryoung Oh

Abstract: In this paper, a DLMS server with a multi-threaded architecture for AMI systems is proposed and implemented. To achieve this, the operation procedures between the DLMS server and the clients are analyzed, and the necessary design requirements for the server are derived. The roles of the DLMS server are divided into multiple threads to meet these requirements, and the DLMS operation procedures are organized at the thread level. The proposed architecture is modeled using the DEVS formalism, a language for hierarchical modularization of discrete event systems. Subsequently, the general operational procedures and exceptional situations are simulated in the DEVSim++ environment. The simulation results show that the DLMS server with the proposed structure behaves correctly for each scenario and meets the DLMS standards and the given functional requirements. Finally, the DLMS server is implemented based on the validated simulation code. The implementation results confirm that the DLMS server operates correctly when connected to multiple smart meter devices. In the implemented DLMS server, unnecessary concurrency among threads is strictly limited, significantly reducing the costs associated with testing.

Keywords: Device Language Message Specification (DLMS); Discrete Event Systems; Multithreading; Simulation

1 INTRODUCTION

Smart grid is a next-generation intelligent power grid that provides improved power services and maximizes energy efficiency by upgrading the power grid through the use of information and communication technology (ICT) for the existing power grid. In general, the components of the Advanced Metering Infrastructure (AMI) which serves as the core of a smart grid, include smart meters, data collection units (DCUs), and meter data management systems (MDMS). In an AMI system, smart meters collect data and transmit it in real-time to the DCUs and MDMS. The DCU is responsible for transmitting the data collected by the smart meters to the MDMS, and the MDMS monitors and optimizes energy usage patterns based on the transmitted data [1]. Communication among the smart meter, DCU, and MDMS is important for the application of the AMI technology. However, the communication protocols vary depending on the smart meter manufacturer, which causes various problems [2].

The Device Language Message Specification (DLMS) protocol [3-5] was established to address these issues. DLMS protocol is a key communication standard in the AMI system of the smart grid, enabling efficient interoperability between smart meters, DCUs, and MDMS. The DLMS protocol uses a server-client structure based on TCP-IP or HDLC. In this structure, the server responds as the client requests the service. In a DLMS system, the upper application acts as the client, and the smart meter acts as the server. The server in a DLMS system is responsible for communicating with the clients and processing messages, managing various events, reporting results, and handling errors. These operations require concurrency, efficient resource utilization, fast response times, and stability. To handle these behaviors effectively, it is better to design the DLMS server as a multi-threaded structure that can run in parallel rather than a single-threaded structure that runs sequentially [6, 7].

This paper proposes a DLMS server with a multi-threaded architecture. Based on the operation procedure of the DLMS system, the requirements for the design on the DLMS server are identified and the threads needed to fulfill these requirements are derived. The general behavior of the DLMS server is specified on a separate thread level and the roles of each thread are clearly defined based on it. The designed multi-threaded structure is verified using the Discrete Event System Specification (DEVS) formalism [8-10], a language for describing discrete event systems. The multi-threaded structure, designed according to the roles of each thread, is modeled using the atomic and coupled models of the DEVS formalism. Subsequently, it is simulated using DEVSim++. The reason for using the DEVS formalism in this paper is that it can easily model multi-threaded systems, strictly control the system's behavior through a synchronization mechanism based on virtual time, and finally, easily simulate the modeled system using an abstract simulator algorithm. The validated DLMS server is directly implemented based on the simulation code and tested in a virtual environment.

This paper is organized as follows. Chapter 2 gives a brief introduction to the DLMS system, and Chapter 3 builds on Chapter 2 to propose a multi-threaded architecture for the DLMS server. In Chapters 4 and 5, the multi-threaded structure proposed in Chapter 3 is modelled and simulated using the DEVS formalism. In Chapter 6, a DLMS server is implemented based on the simulation results. Finally, Chapter 7 is the conclusion of this paper.

2 DLMS PROTOCOL

The DLMS protocol is a protocol in the energy and utilities sector, which was specified by the International Electrotechnical Commission (IEC) and the DLMS US. It is also known as IEC 62056 and is used for communication between smart meters and upper applications, as shown in Fig. 1.

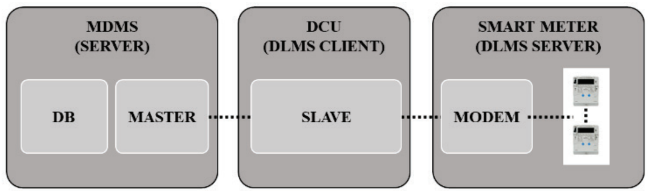


Figure 1 Interface between the smart meter and upper application

The protocol is designed to communicate not only with electricity meters but also with other devices, such as water and gas meters. The DLMS protocol defines three processes: modeling, messaging, and transport. Modeling phase is defined through Companion Specification for Energy Metering (COSEM) and Object Identification System (OBIS). COSEM is a standard data model for the interface and data identification of smart meters, structuring and defining data and services related to various metering functions. Each COSEM object performs operations such as reading, writing, and executing data through specific attributes and methods. OBIS is a system for identifying COSEM objects, uniquely identifying each data item. OBIS consists of six groups of numbers, each representing specific data items and functions. Messaging phase defines messages for accessing the modeled data. Messages include invoke ID and type, service parameters, priority, and so on, which determine and execute the operation mode based on these parameters. The main message types in DLMS are GET, SET, and ACTION, each having specific parameter values depending on the operation performed. Each message contains one or more service parameters, based on which the smart meter processes detailed operations. Finally, in transport phase, communication profiles are defined, which specify how the DLMS protocol can be used over various standard communication media. The DLMS protocol uses Transmission Control Protocol/Internet Protocol (TCP-IP) or High-Level Data Link Control (HDLC)-based communication, which basically works by exchanging messages [11-13].

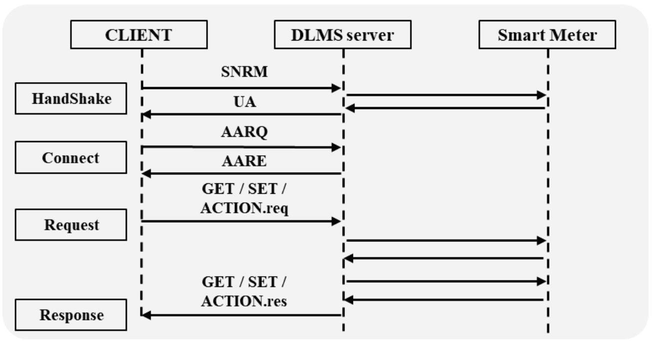


Figure 2 General operation procedure of the DLMS protocol

Fig. 2 shows the general operation procedure of the DLMS protocol. In the HandShake and Connect phases, when a client sends a connection request to the server to initiate communication with a specific smart meter, the server accepts it and securely establishes the connection through authentication and encryption. Subsequently, in the Request and Response phases, when the client sends a request message, the server gathers information from the smart meter regarding that request and delivers a response message back to the client. Further information on the DLMS protocol can be found in [4] and [5].

3 DESIGN

This chapter designs a DLMS server. The DLMS protocol documentation only describes the basics of DLMS, which allows DLMS servers to be designed in a wide variety of ways [14-17]. The design of the DLMS server considers the following requirements:

(i) The system should be able to handle request messages that come asynchronously from the client and exceptional situations that may occur unexpectedly in the smart meter.
(ii) The system should be able to record and manage the results and status information of the communications with the smart meter, and it should also be able to report the contents upon the client's request.
(iii) The system should be able to store and manage the parameters of the different request messages received from the client to the system.
(iv) The system should be able to communicate with the smart meter both wired and wirelessly.
(v) The system should support smart meters from different manufacturers to ensure interoperability.

To meet all of the requirements above, it is appropriate to design a DLMS server with a multi-threaded architecture that can handle multiple tasks in parallel. This paper divides the DLMS server into six types of threads. To satisfy requirement (i), the thread responsible for sending messages and the thread that receives messages are isolated from other threads. By separating the sending and receiving parts, communication with the client can be handled asynchronously, which makes communication more responsive. In addition, if any modifications to the communication interface are required, these can be kept to a minimum. In this paper, the sending and receiving parts of the message are separated from the other parts for the reasons mentioned above by using the SEND thread for the sending part and the RECV thread for the receiving part. To meet the requirement (ii), the REPORT thread is separated. The REPORT thread records and manages the processing results of the various request messages and the status of the smart meter and converts the data into the DLMS message format for reporting to the client. To fulfill the requirement (iii) and manage the overall behavior of the DLMS server, the MANAGER thread is also separated. The MANAGER thread stores and manages the messages received from the client and performs processing operations on them when they are ready for processing. To manage multiple smart meters and satisfy the requirements (iv) and (v) at the same time, the HANDLER threads and the uHANDLER threads are separated and employed. Depending on the type of command requested by the DLMS client, the DLMS server may need to communicate with the smart meter multiple times while processing a single command. Therefore, the HANDLER thread is responsible for splitting one DLMS message into a series of detailed operations according to the service parameters of the message and passing it to the uHANDLER thread, which is responsible for communicating with the smart meter according to the application programming interfaces (APIs) of the different manufacturers. Since several smart meters can be connected to one DLMS server, there are generally several HANDLER threads running on the DLMS server, and each HANDLER thread is connected to one uHANDLER thread.

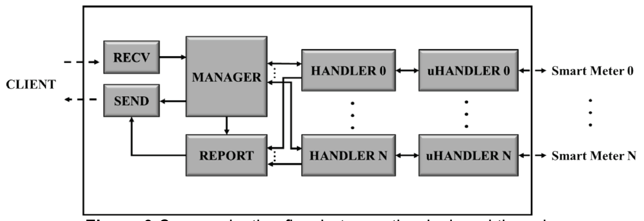


Figure 3 Communication flow between the designed threads

Fig. 3 shows the threads designed in this paper and the communication flow between the threads. As mentioned earlier, the RECV thread and SEND thread handle interactions with the client, establishing and maintaining connections with them. The HANDLER thread and uHANDLER thread, on the other hand, manage interactions with the smart meters, establishing connections with them. Between them, the REPORT thread stores data collected from the smart meters via the HANDLER thread, and upon completion of processing request messages, it sends response messages to the client. The MANAGER thread oversees the overall operations of the other threads.

With the separation of the DLMS server into six threads, the operation procedures of the DLMS server are refined and reorganized. Fig. 4 shows a schematic representation of the operation procedure of the DLMS server at a thread level for a GET message based on the general operation procedure in figure 2. When the client sends a GET message, ① it is first received by the RECV thread and ② passed to the MANAGER thread. The MANAGER thread is responsible for scheduling the processing of several received messages. Therefore, the MANAGER thread saves the received messages and forwards them to the ③ HANDLER thread and the ④ REPORT thread when they are ready to process the messages. The HANDLER thread then splits the received messages into specific detailed operations according to the service parameters and sends them to the ⑤ uHANDLER thread. Then, the ⑥ uHANDLER thread controls the smart meter directly connected to it to execute the directed detailed operations. Through the steps ⑦ - ⑨, the result is then passed to the REPORT thread, where it is saved. Finally, when all the detailed operations for the GET message have been completed, it is sent to the ⑬ client via the ⑩ MANAGER thread, the ⑪ REPORT thread, and the ⑫ SEND thread. As mentioned earlier, a single DLMS message requested by a client can be separated into a series of detailed operations. In Fig. 4, it should be noted that step ③, where the message is sent from the MANAGER thread to the HANDLER thread, occurs only once, whereas step ⑤, where the message is passed from the HANDLER thread to the uHANDLER thread, occurs twice.

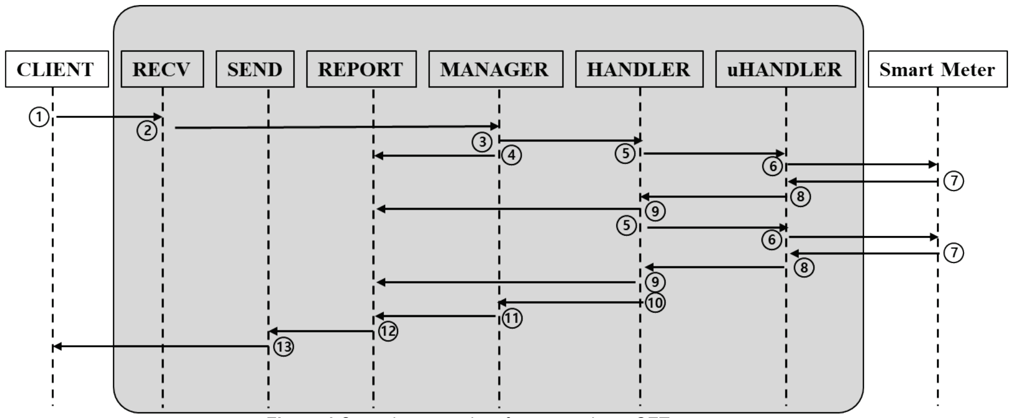


Figure 4 Operation procedure for processing a GET message

4 MODELING

This chapter models the proposed multi-threaded structure. Multi-threaded programs are more efficient than conventional single-threaded programs as they can execute multiple tasks simultaneously. They also have the advantage of being flexible enough to respond to events that occur outside the program. However, they are complex in design and difficult to verify compared to general programs. Multi-threaded programs do not always guarantee a consistent execution order as threads run in parallel and competitively with each other. Accordingly, even if the same events occur in the same order and at the same time, the operation result of the program may be different each time. The problems mentioned above make using the debugger program considerably more difficult [18].

To address these issues, the DEVS formalism is used, which describes discrete event systems in a hierarchical and modular way. The DEVS formalism is employed because (i) the behavior of a multi-threaded system can be modeled as a discrete event system, (ii) the behavior of thread structures can be tightly controlled by a synchronization mechanism based on virtual time, and (iii) the behavior of the modeled threads can be easily simulated using the abstract simulator algorithm of the DEVS formalism.

There are two types of models in the DEVS formalism: atomic models and coupled models. An atomic model describes the behavior of a component. An atomic model is defined by three sets of inputs (ports), outputs (ports), and state variables, and by four characteristic functions: external transition function, internal transition function, output function, and time advance function. The external transition function defines the change of state when an input is received, and the internal transition function defines the change of state over time. In addition, the output function defines the output when the state is changed by the internal transition function, and the time advance function defines how long it can remain in the current state. A coupled model, on the other hand, bundles several submodels into a single model and describes the hierarchical structure of the model and the connections between them. Further information on the atomic model and coupled model of the DEVS formalism can be found in [8].

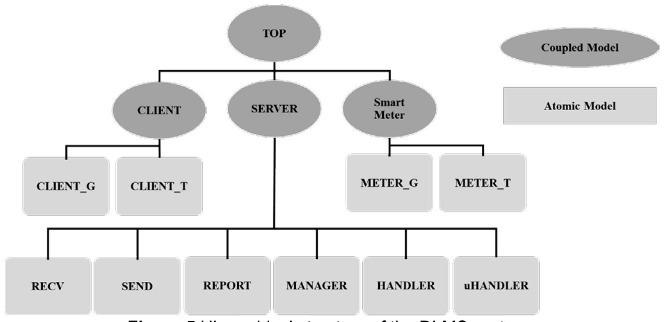


Figure 5 Hierarchical structure of the DLMS system

Each thread designed in Fig. 3 is modeled as an atomic model in the DEVS formalism. In addition, the message-passing relationships and the hierarchical organization between threads are modeled as a coupled model. The hierarchy of the models is shown in Fig. 5. The TOP model, which represents the entire system, consists of a CLIENT model, a SERVER model, and a SMART METER model. Of these, the SERVER model is a coupled model corresponding to the DLMS server proposed in this paper and contains the atomic models for the six types of threads described in Chapter 3. To model the asynchronous behavior of the DLMS client and the smart meter, the CLIENT model and the SMART METER model also consist of the CLIENT_G and METER_G models, which are responsible for generating and sending messages, respectively, and the CLIENT_T and METER_T models, which are responsible for receiving messages.

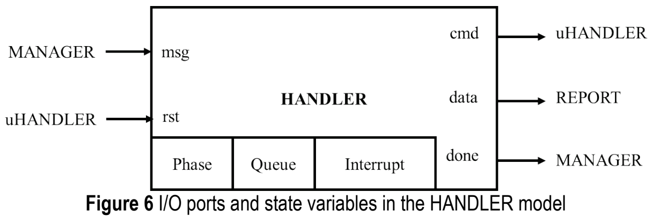


Figure 6 I/O ports and state variables in the HANDLER model

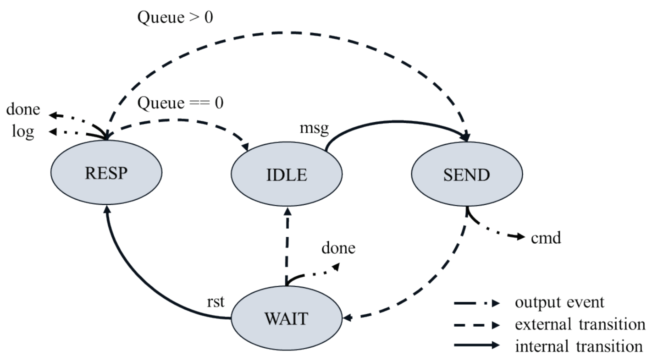


Figure 7 Phase transition diagram of the HANDLER model

Fig. 6 and Fig. 7 are simplified schematic representations of the HANDLER model. In Fig. 4, the HANDLER thread takes the input from ③ and ⑧ to provide the output to ⑤ and ⑨. As described earlier, the change of state due to input is defined as an external transition function, and the change of state over time and the output at this point as an internal transition function and output function. Initially, the HANDLER starts in the IDLE state. Then, when the message is received from the MANAGER through the "msg" port in ③, the external transition function of the HANDLER divides the message into detailed operation messages, stores them in the Queue, and changes the Phase to SEND. Then, after the time determined by the time advance function of the HANDLER in ⑤, the HANDLER’s output function delivers one of the detailed operation messages stored in the queue, which has not yet been processed, to the uHANDLER through the "cmd" port, while the internal transition function of the HANDLER changes the Phase to WAIT. Subsequently, if the result is received from the uHANDLER through the "rst" port within the time specified in ⑧, the external transition function of the HANDLER stores the result in the Queue and changes the Phase to RESP. Then, in ⑨, the output function of the HANDLER transfers the result received in ⑧ to the REPORT via the "data" port, and the internal transition function of the HANDLER changes the Phase to SEND to repeat the process from ⑤ if there are still detailed operations in the queue that have not yet been processed. Otherwise, it changes the Phase to IDLE to complete the processing of the message received in ③. If the message is not received from the uHANDLER after the specified time after sending the message in ⑤, the output function of the HANDLER sends an error message to the MANAGER through the "done" port, and the internal transition function of the HANDLER changes the Phase to IDLE. In addition, if the smart meter generates an alarm message indicating an urgent exceptional situation, the message is forwarded to the HANDLER through the "rst” port, although this is not shown in Fig. 4. In this case, the external transition function of the HANDLER changes the Interrupt to ON so that the output function of the HANDLER later informs the MANAGER of the alarm via the "done" port.

Fig. 8 shows the atomic DEVS specification of what is described in Fig. 6 and Fig. 7. In the figure, X is the input event set, which represents the input of messages through the "msg" and "rst" ports, and Y is the output event set, which represents the output of messages through the "cmd", "data", and "done" ports. S is the state variable set, which contains state variables such as Phase, Queue, and Interrupt. In addition, δ_{ext} and δ_{int} represent the external transition function and internal transition function, λ is the output function, and ta is the time advance function.

HANDLER = {X, Y, S, δ_{ext}, δ_{int}, λ ta}

X = {msg, rst}

Y = {cmd, data, done}

S = {Phase, Queue, Interrupt}	λ(SEND) = cmd
δ_{ext} {(IDLE, 0, -), msg} = (SEND, n, -)	λ(RESP) = done, log
δ_{ext} {(WAIT, -, -), rst} = (RESP, -, -)	λ(WAIT) = done
δ_{int} {(SEND, -, -)} = (RESP, n, -)	ta(WAIT) = 1000
δ_{int} {(WAIT, n, -)} = (IDLE, 0, -)	ta(SEND) = 0
δ_{int} {(RESP, n, -)} = if (Queue > 0) (SEND, n - 1, -)	ta(RESP) = 0
δ_{int} {(RESP, n, -)} = if (Queue == 0) (IDLE, 0, -)	ta(SEND) = ∞
δ_{int} {(-, -, OFF)} = if (msg.type == Error) (IDLE, -, ON)	

Figure 8 DEVS specification of the HANDLER model

5 SIMULATION

This chapter validates the proposed architecture from Chapter 4 through simulations. To this end, each of the models in Fog. 5, including the HANDLER model, is coded and simulated in DEVSim++ [19], a C++-based DEVS simulation environment. For more accurate verification, the behavioral procedures under normal circumstances, including those described in Fig. 4 and Fig. 5, as well as the operation procedures under various exceptional cases, are simulated. Below are the basic assumptions for the simulations in this paper:

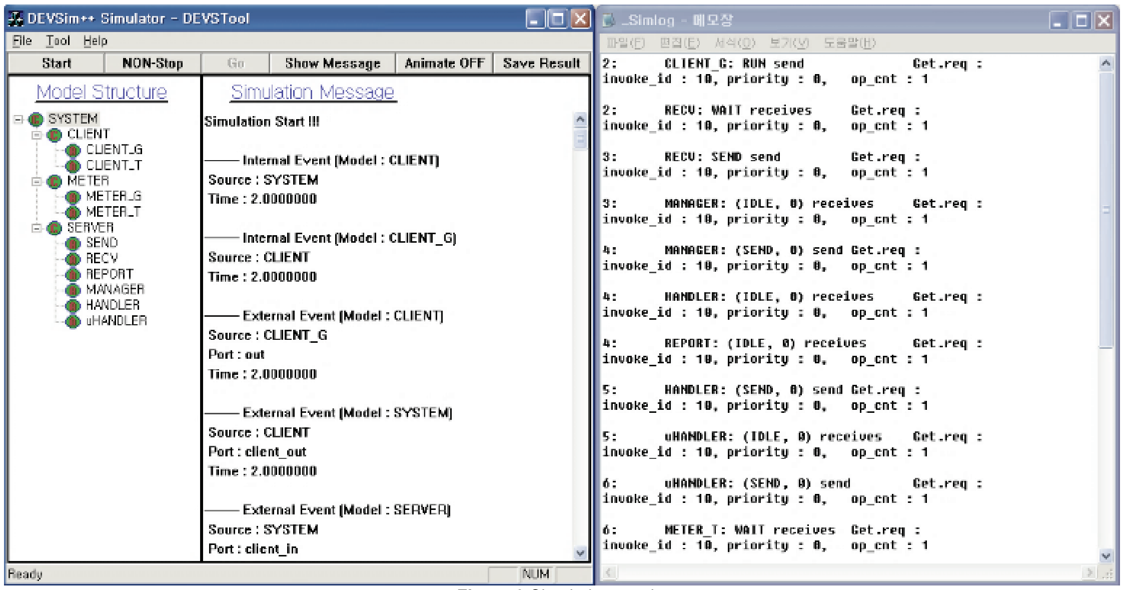
(i) There are no problems with the communication between the DLMS server and the client. Therefore, the simulation of the process of initializing and setting up communication is excluded.

(ii) The number of service parameters contained in one DLMS message is set to 3, i.e., it is assumed that the DLMS server always communicates with the smart meter three times to process a single DLMS message.

(iii) Two smart meters are connected to the DLMS server.

Based on these assumptions, simulates 7 scenarios. Among them, 4 scenarios simulate operational procedures in normal conditions, while 3 scenarios simulate operational procedures in abnormal conditions. In each scenario, the client generates 20 - 100 request messages.

As mentioned earlier, debugging in a multi-threaded environment can be difficult as the behavior of the threads is not sequential. Therefore, to overcome this difficulty, the synchronization mechanism of the DEVS formalism is utilized to strictly control the behavior of the thread structure. Fig. 9 shows a screenshot that simulates the general operation procedure of the DLMS server using DEVSim++ and the log messages generated during execution. By analyzing the log messages, the status change of each model over time can be checked and it can be determined whether the message is transmitted correctly.



```
Simulation Start !!!

—— Internal Event [Model : CLIENT]
Source : SYSTEM
Time : 2.0000000

—— Internal Event [Model : CLIENT_G]
Source : CLIENT
Time : 2.0000000

—— External Event [Model : CLIENT]
Source : CLIENT_G
Port : out
Time : 2.0000000

—— External Event [Model : SYSTEM]
Source : CLIENT
Port : client_out
Time : 2.0000000

—— External Event [Model : SERVER]
Source : SYSTEM
Port : client_in
```

```
2:      CLIENT_G: RUN send              Get.req :
invoke_id : 10, priority : 0,   op_cnt : 1

2:      RECV: WAIT receives     Get.req :
invoke_id : 10, priority : 0,   op_cnt : 1

3:      RECV: SEND send         Get.req :
invoke_id : 10, priority : 0,   op_cnt : 1

3:      MANAGER: (IDLE, 0) receives     Get.req :
invoke_id : 10, priority : 0,   op_cnt : 1

4:      MANAGER: (SEND, 0) send Get.req :
invoke_id : 10, priority : 0,   op_cnt : 1

4:      HANDLER: (IDLE, 0) receives     Get.req :
invoke_id : 10, priority : 0,   op_cnt : 1

4:      REPORT: (IDLE, 0) receives      Get.req :
invoke_id : 10, priority : 0,   op_cnt : 1

5:      HANDLER: (SEND, 0) send Get.req :
invoke_id : 10, priority : 0,   op_cnt : 1

5:      uHANDLER: (IDLE, 0) receives    Get.req :
invoke_id : 10, priority : 0,   op_cnt : 1

6:      uHANDLER: (SEND, 0) send        Get.req :
invoke_id : 10, priority : 0,   op_cnt : 1

6:      METER_T: WAIT receives  Get.req :
invoke_id : 10, priority : 0,   op_cnt : 1
```

Figure 9 Simulation results

In this paper, simulations of three representative scenarios out of seven scenarios are described. The first scenario is part of the simulation results of operational procedures under normal conditions. Fig. 10 shows the simulation results for the case where two new request messages (Message 2 & Message 3) arrive sequentially (ⓐ and ⓑ) while the DLMS server is processing a request message (Message 1). It should be noted that different types of lines are used in the figure to separate the messages. In this simulation, Message 2 has a higher priority than Message 1, and Message 3 has a higher priority than Message 2. In other words, Fig. 10 shows a case where a high-priority message arrives while a low-priority message is being processed. This paper treats the processing of each message as a single, indivisible atomic transaction. Thus, by default, higher-priority messages are processed before lower-priority messages. However, once a message has started processing, it is not to be preempted until it has completed processing. As a result, the simulation results shown in Fig. 10, the HANDLER thread continues to process the existing Message 1, even though higher priority messages have arrived at ⓐ and ⓑ. It can also be seen ⓒ that after processing Message 1, it starts processing Message 3, which has the highest priority among the messages stored in the queue of the MANAGER thread. As a result, it has been confirmed that messages sent by the client are appropriately processed according to their priorities.

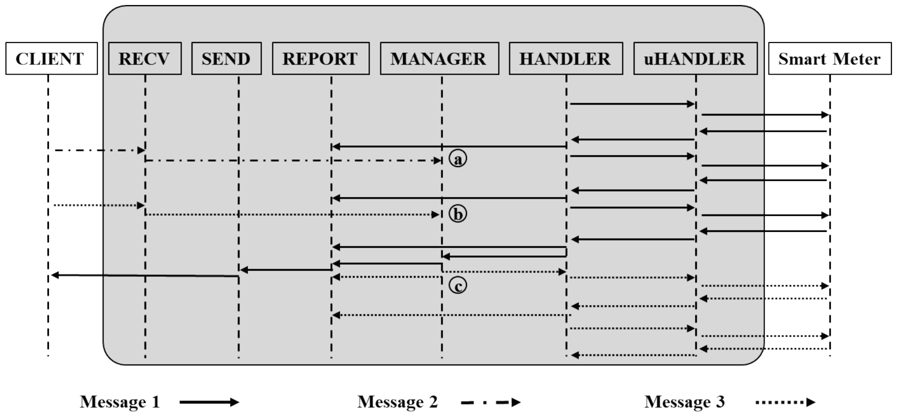


Figure 10 Processing of multiple requests

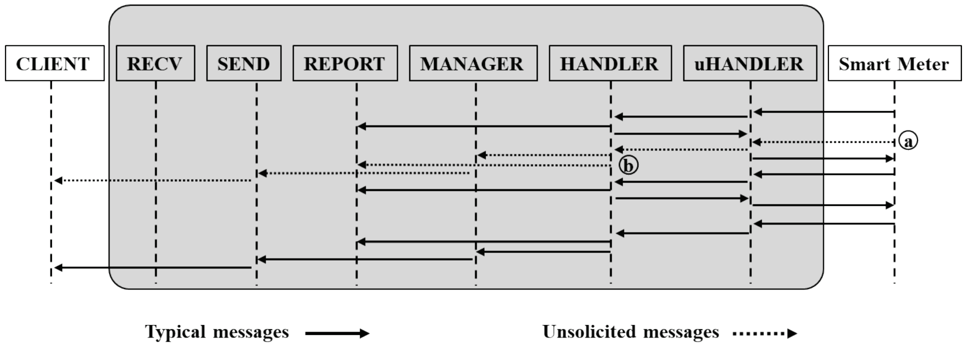


Figure 11 Processing of an unsolicited message generated by smart meters

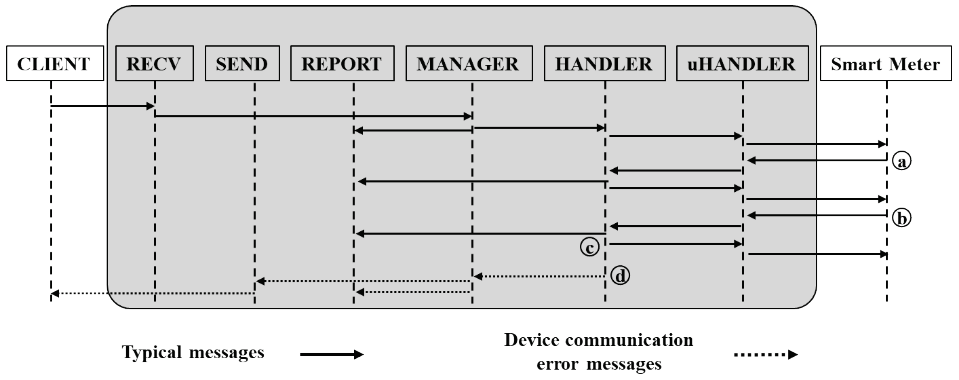


Figure 12 Processing of a communication error with smart meters

The second scenario is part of the simulation results of operational procedures under abnormal conditions. Fig. 11 shows the simulation results when the HANDLER thread receives an unsolicited message from the smart meter while processing the request. In the figure, the process of the HANDLER thread's request is represented by the solid line, while the processing of the unsolicited message generated by the smart meter is represented by the dotted line. There are two reasons why the smart meter may generate unsolicited messages. The first is when the smart meter encounters an abnormal situation, such as a device failure, and the second is when the smart meter needs to report status periodically. The DLMS protocol stipulates that when an unsolicited message occurs, the record is immediately stored on the DLMS server and reported to the DLMS client. According to the simulation result in Fig. 11, the smart meter has generated an unsolicited message in ⓐ. The message is then passed to the HANDLER thread via the uHANDLER thread in ⓑ. The

HANDLER thread then recognizes that it is an unsolicited message and immediately forwards it to the REPORT thread and the MANAGER thread. In addition, once the unsolicited message has been processed, the HANDLER thread continues to process the request that it was previously processing. As a result of the simulation, it is confirmed that the unsolicited message generated by the smart meter was delivered to the client prior to the processing of the existing message.

The last scenario also represents a partial simulation result of operational procedures under abnormal conditions. Fig. 12 is the simulation of an exceptional situation, in which the HANDLER thread has sent a request to the smart meter through the uHANDLER thread but has not received any response from the smart meter within the specified time. The DLMS protocol stipulates that in this case, the record is kept in the DLMS server and reported to the DLMS client, similar to the case of unsolicited messages. In figure 12, the smart meter sends a response to the request received in ⓐ and ⓑ, while in ⓒ, it does not send any response for some reason. Eventually, in ⓓ, the HANDLER thread realizes that an error has occurred during communication with the smart meter and sends a device communication error message to the MANAGER thread. The MANAGER thread then forwards the message to the REPORT thread and the SEND thread to log the occurrence of the device communication error and report it to the DLMS client to end exception handling. As a result, it is confirmed that when a communication error occurs from the smart meter, the DLMS server logs the error internally and promptly reports it to the client.

Based on diverse simulation results, including the aforementioned ones, it is confirmed that the proposed DLMS server effectively sends and receives messages asynchronously and processes them precisely as intended. It is also ensured that the DLMS server accurately accumulates state changes and results, while maintaining communication with the client for both general DLMS request messages and unexpected exceptional cases.

6 IMPLEMENTATION

This chapter implements a DLMS server with the proposed multi-threaded architecture. Although DEVSim++, which is based on the C++ language, is used to simulate the proposed architecture, the simulation code can only be executed in the environment of DEVSim++, and it is significantly different from the program code used in the real environment.

Most of the conventional programming languages are not developed on mathematical foundations. Therefore, many of the mathematical expressions used in modeling are often written in a slightly distorted form when translated into a programming language. The cumulative effect of these distortions is that the behavior of the implemented program differs significantly from what is formally modeled and validated by simulation. For this reason, it is usually considerably difficult to express the results modeled with a formal modeling method in a conventional programming language.

In a previous work [20], one solution was proposed to implement DEVS models for multi-threaded architectures written in DEVSim++ with real threads. In the proposed method, the hierarchical structure of the entire system model, which consists of multiple layers, is flattened into a two-layer structure in which all atomic models are subordinated to a single coupled model. Then, the single coupled model is implemented as a thread called SCHEDULER, and the DEVSim++ code of each atomic model is converted to an actual thread written in C++. This method reuses the expressions written in DEVSim++ with almost no modification while implementing threads, resolving the problem of expression distortion during the conversion process, which was a previous concern. The threads of the atomic models generated through this process have a characteristic that, unlike typical threads, they only execute actions corresponding to input or trigger messages received from external sources. Therefore, if no messages are received, they remain idle without performing any actions. Hence, even if the operating system schedules threads in a random order, the threads execute actual operations in the order of trigger messages generated by the SCHEDULER thread.

This paper implemented the proposed multi-threaded architecture DLMS server in the Linux operating system environment using the C++ language, following the method proposed in [20]. The implemented DLMS server consists of seven types of threads corresponding to six thread models, except for the CLIENT and the SMART METER models among the simulated models, and the SCHEDULER thread. As mentioned earlier, only one thread is created for each of the seven thread types except for the HANDLER thread and uHANDLER thread. However, several threads are created for the HANDLER thread and the uHANDLER thread to match the number of smart meters connected to the DLMS server.

Finally, the implemented DLMS server program is validated. For proper validation, it is required to connect the implemented DLMS server with real smart meters and examine its behavior while the server communicates with a real DLMS client. However, due to budget limitations, it is difficult to procure real smart meters and DLMS clients. Therefore, experiments are conducted by implementing independent programs for each device separately. For validation, two virtual smart meters are employed, identical to the simulation environment in Chapter 5. Therefore, the implemented DLMS server program communicates with one DLMS client program and two smart meter programs to process requests. Additionally, within the DLMS server, there are a total of 9 threads consisting of 2 HANDLER threads and 2 uHANDLER threads each.

Verification proceeds in two stages. In the first stage, to validate general operation procedure, DLMS clients generate requests identical to those simulated in Chapter 5, ensuring smart meters exhibit consistent responses. The operation of the DLMS server is logged and compared with simulation results. The comparison confirms identical operation between the observed simulation and the actual operation of the server. In the second stage, DLMS clients and smart meters are allowed to freely generate and respond to requests. Analysis of the generated logs confirms that the DLMS server operates in accordance with the requirements specified in the protocol and design.

7 CONCLUSION

This paper proposes a DLMS server with multi-threaded architecture for AMI systems. The proposed DLMS server must handle communication and message processing with clients, manage various events, and control multiple smart meter devices with different communication protocols. First, specific requirements are derived from the above needs. To meet these requirements, the DLMS server is designed with 6 threads. The proposed architecture is modeled using the DEVS formalism and simulated in the DEVSim++ environment for verification. The simulation results confirm that the proposed multi-threaded DLMS server operates according to the requirements identified in this paper. Ultimately, the DLMS server is implemented based on the simulation code.

As anticipated, there may be debates regarding performance issues with the DLMS server implemented in this paper. Specifically, the concurrency of thread operations is strictly limited by the scheduler thread, which may degrade the server's performance. Nonetheless, it is important to note that the concurrency of servicing DLMS requests is not restricted. The DLMS server implemented in this paper may experience slight performance degradation due to the strict limitations on concurrency of thread execution. However, this approach prevents race conditions caused by the disorderly execution of threads, significantly reducing the cost of implementing and testing the DLMS server. The method presented in this paper can be used in various fields where the verification of operations is more critical than system performance.

8 REFERENCES

[1] Jung, S. M., Kim, T.-K., Seo, H.-S., Lee, S.-J., & Kwak, J. (2013). The prediction of network efficiency in the smart grid. *Electronic Commerce Research, 13*, 347–356. https://doi.org/10.1007/s10660-013-9124-1

[2] Feuerhahn, S., Zillgith, M., Wittwer, C., & Wietfeld, C. (2011). Comparison of the communication protocols DLMS/COSEM, SML and IEC 61850 for smart metering applications. *The IEEE International Conference on Smart Grid Communications (SmartGridComm2011)*, 410-415. https://doi.org/10.1109/SmartGridComm.2011.6102357

[3] Prasanth, G. (2009). Implementing DLMS Client and Server Protocols in Meters, IED's, MRI's and Meter Reading Applications – An Overview. Vinoo S Warrier, Vice President, Kalkitech (Internet). Version 1. *gopalakrishnanprasanth*. https://gopalakrishnanprasanth.wordpress.com/article/implementing-dlms-client-and-server-3bk0x32fl4sfh-15/

[4] DLMS User Association. (2019). DLMS/COSEM Architecture and Protocols. *Green Book Edition 8.1*.

[5] DLMS User Association. (2019). COSEM Interface Classes and OBIS Object Identification System. *Blue Book Edition 13*.

[6] Campbell, J. M., Ellis, R. K., & Giele, W. T. (2015). A multi-threaded version of MCFM. *The European Physical Journal*, C 75, 1-7. https://doi.org/10.1140/epjc/s10052-015-3461-2

[7] Wickramasinghe, M., & Guo, H. (2014). Energy-Aware Thread Scheduling for Embedded Multi-threaded Processors: Architectural Level Design and Implementatio. *The IEEE Computer Society Annual Symposium on VLSI*, Tampa, FL, USA, 178-183. https://doi.org/10.1109/ISVLSI.2014.55

[8] Zeigler, B. P. (1990). *Object-Oriented Simulation with Hierarchical, Modular Models*. Academic Press.

[9] Zeigler, B. P. (1984). *Multifaceted Modeling and Discrete Event Simulation*. Academic Press.

[10] Zeigler, B. P., Prähofer, H., & Kim, T. G. (2000). *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press.

[11] Štruklec, G., & Maršić, J. (2011). Implementing DLMS/COSEM in smart meters. *The 8th International Conference on the European Energy Market (EEM2011)*, Zagreb, Croatia, 747-752. https://doi.org/10.1109/EEM.2011.5953109

[12] Kheaksong, A., & Lee, W. (2014). Packet transfer of DLMS/COSEM standards for smart grid. The 20th Asia-Pacific Conference on Communication, 391-396. https://doi.org/10.1109/APCC.2014.7092843

[13] Ju, S. H., & Seo, H. S. (2018). Design key management system for DLMS/COSEM standard-based smart metering. *International Journal of Engineering and Technology (UAE), 7*, 554-557. https://doi.org/10.14419/ijet.v7i3.34.19380

[14] Burunkaya, M., & Pars, T. (2017). A smart meter design and implementation using ZigBee based Wireless Sensor Network in Smart Grid. *The 4th International Conference on Electrical and Electronic Engineering (ICEEE2017)*, 158-162. https://doi.org/10.1109/ICEEE2.2017.7935812

[15] Park, S. B., Ahn, Y. J., & Kim, J. H. (2010). Development of OPC Server Unifying Communication Profiles of DLMS/COSEM in Smart Grid. *The Journal of Korean Institute of Information Technology, 8*(9), 1-11. https://doi.org/10.14801/jkiit.2020.18.9.1

[16] Im, C. J., Jang, S. J., Hahn, K. S., Kim, B. S., & Jung, N. J. (2007). *The Development of IEC62056 based Energy Information Concentrator for DLMS Meters*. The Korean Institute of Electrical Engineers, 54-56.

[17] Biswas, S., Himanshu, Ghosh, S., Das, P., Saha, K., & De, S. (2023). Efficient Data Transfer Mechanism for DLMS/COSEM Enabled Smart Energy Metering Platform. SIGMETRICS Perform. *ACM SIGMETRICS Performance Evaluation Review, 50*(4), 14-16. https://doi.org/10.1145/3595244.3595250

[18] Tarvo, A., & Reiss, S. P. (2018). Automatic performance prediction of multithreaded programs: a simulation approach. *Automated Software Engineering, 25*(1), 101-155. https://doi.org/10.1007/s10515-017-0214-5

[19] Kim, T. G. (1994). DEVSim++ User's Manual: C++ Based Simulation with Hierarchical Modular DEVS Models.

[20] Kim, Y. H., Seong, Y. R., & Oh, H. R. (2014). Software Development Method Using the Concurrency Control Approach Based on DEVS Simulation. *Proceeding of the 2014 FTRA International Conference on ACS, 11*(7), 553-558.

Authors' contacts:

Gwang Hyeon Kim, Student
Department of Electrical Engineering, Kookmin University,
77 Jeongneung-ro Seongbuk-gu, Seoul, 02707, Korea
rhkdgus306@kookmin.ac.kr

Yeong Rak Seong, Professor
(Corresponding author)
Department of Electrical Engineering, Kookmin University,
77 Jeongneung-ro Seongbuk-gu, Seoul, 02707, Korea
yeong@kookmin.ac.kr

Ha Ryoung Oh, Professor
Department of Electrical Engineering, Kookmin University,
77 Jeongneung-ro Seongbuk-gu, Seoul, 02707, Korea
hroh@kookmin.ac.kr