Journal of Marine Science and Technology
https://doi.org/10.1007/s00773-025-01072-0

ORIGINAL ARTICLE

System design and evaluation method for practical development of autonomous ships

Suisei Wada[1] · **Ryohei Sawada**[2] · **Atsuo Maki**[1]

Received: 3 February 2024 / Accepted: 7 June 2025

Abstract
To develop a sophisticated ship system, such as an autonomous ship, experiments using a real ship are essential to evaluate its performance and behavior of the developing system. However, it is difficult to evaluate it installed on various ships, which have different configurations of actuators and sensors. This paper describes a design method of evaluation platform for autonomous ships that easily adapts the target system to different configured ships. The method is based on a distributed architecture, and the system consists of an abstracted hardware driver and target system connected through the communication bus. This paper explains a design method and an example module design. The authors implemented the system using the proposed architecture and showed an example of porting the system to another ship. In addition, this paper discusses the advantages and points to note associated with certification processes, such as classification approvals, when applying the proposed design method to actual development.

Keywords Autonomous ship · Ship experiment · Model ship · System design

1 Introduction

Autonomous ships are highly expected that one of the effective solutions for preventing ships from marine accidents. Also, it is considered that autonomous ship technology is an effective solution to the problem of seafarer shortages [1]. For this reason, some projects to develop autonomous ships have been started around the world recently [2–4]. Not only fully autonomous technologies but also technologies aimed at assisting crews, such as watch-keeping functions [5] have been also developing. In the development of these advanced ship control functions, evaluation is an essential process. Particularly, experiments using a real ship are considered to be essential.

Figure 1 shows the typical process of development of a ship control system. The process consists of three parts: the research process, the development process, and the product process. The last step in every process of development is a test or evaluation using a real ship. In the research process, the function test is a proof of concept (PoC) in which the performance of a unit function is evaluated. Subsequently, the system evaluation of a system that integrates several unit functions, which are verified by the functional test, brings the system from the development process to the production process. In the product process, the product will be installed on the real ship through an adaptation process involving parameter tuning specific to each individual ship.

As described above, real ship experiments play a significant role in the development of ship control systems. The system evaluation is reported in recent autonomous ship development projects in Japan [6, 7]. Conducting experiments with an actual ship yields precise results; however, it also incurs expenses and requires a significant amount of time. Thus, the earlier step of development consists of experiments with a model ship instead of an actual ship. For example, Ahmed et al. used a free-run model ship to evaluate their autonomous berthing algorithm [8]. In other cases, there is some research to evaluate the dynamic positioning function [9], and the course tracking function [10] with model ships. While model tests can verify the hydrodynamic behavior of the ship, it is difficult to verify the hardware-induced failures of the implemented system. Hardware in the Loop (HILS) is often performed for this purpose. Huijgens

✉ Suisei Wada
suisei_wada@naoe.eng.osaka-u.ac.jp

[1] Osaka University, 2-1, Yamadaoka, Suita, Osaka, Japan

[2] National Maritime Research Institute, 6-38-1, Shinkawa, Mitaka, Tokyo 181-0004, Japan

Published online: 12 July 2025

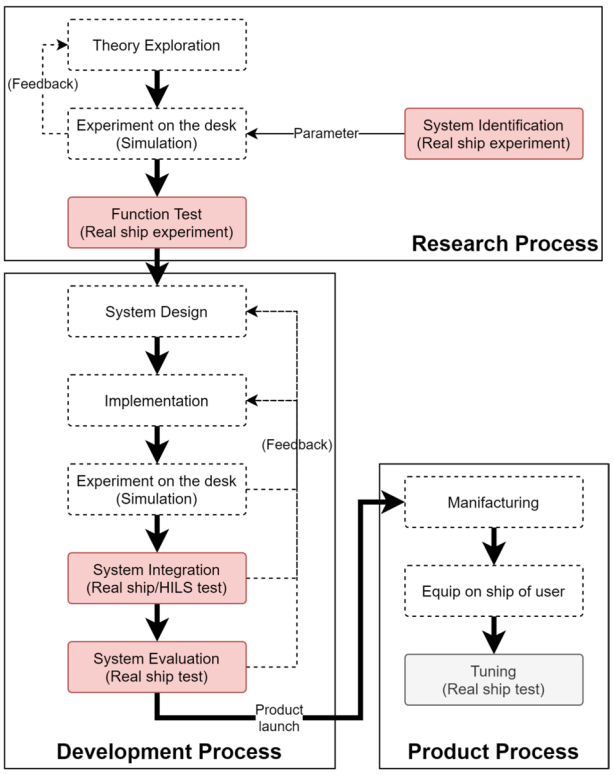

Fig. 1 Typical process of development of ship control system

et al. combined HILS and a towing test [11] to evaluate the electronic propulsion system [12].

Many methods are employed in the development of these systems, which leads to further cost increases. Every ship has different hardware configurations, so it has been difficult to use the same test platform for many types of ships. To be worth the cost of development, the system to be developed must be versatile enough to accommodate different hardware configurations of ships. In this context, during system development, hardware drivers responsible for managing the equipment installed on the ship can potentially pose challenges to achieving this versatility.

Then, the author presents here a method of design of development and evaluation platform that can be applied to different types of ship configurations. The proposed method is realized by utilizing the distributed architecture of abstracted hardware drivers.

This paper is composed of six sections. In the next section, an overview of the proposed method is provided. In Sect. 3, detailed design elements such as device drivers and application software are described. In the following section, an example implementation of an experimental system with the proposed architecture is presented. The following Sect. 5 provides a practical discussion on the benefits of applying the proposed methodology in the development of real–world marine systems, along with considerations for points to note in the design and development. In the final section, we conclude the paper.

2 System overview

Real ships have been used in the research and development of a sophisticated ship control system, such as an autonomous ship system [6–10]. Each real ship experiment platform, which is used in these examples, was designed for their research and development. There is also a report to develop a ship experiment platform [13]. These systems are considered to be helpful in evaluating the performance of the function or the system in their research. However, in these researches, the evaluated systems are not so much assumed to be deployed on different ships. Generally, ship configurations are different from vessel to vessel. And then, it has been mentioned in the previous section, that a design method is required to separate the evaluated software and the module of input and output, to apply it to various types of ship configurations.

Before the explanation of system structure, the term"application software"is defined here. In this context, application software is defined as an evaluated object software, which contains one or more functions that realize some integrated ship control system, such as an autonomous berthing controller, a track control system, etc. These functions are sometimes realized using multiple application software. In case the application software includes the communication part, such as the top figure in Fig. 2, the experiment system works for a particular system or ship.

However, when application software has to be ported to another type of ship, this structure has some problems as next:

- Generally, hardware has its own original communication scheme. In case the application software contains a device communication part, a communication scheme has to be implemented or removed every time when changing the structure of the hardware. In other words, application software cannot be ported to other ships without modification.
- The application software evaluation should not include the communication portion with the device. In other words, what we want to evaluate is the algorithm that the application software has, such as autonomous control, and it should be separated from the problem that the output commands are not properly input to the device. It sometimes induces harmful behavior in the communication part, and possibly misunderstands the result of the evaluation. In order to avoid troubles in an experiment, it is considered that the original code and the other functional code should be separated.

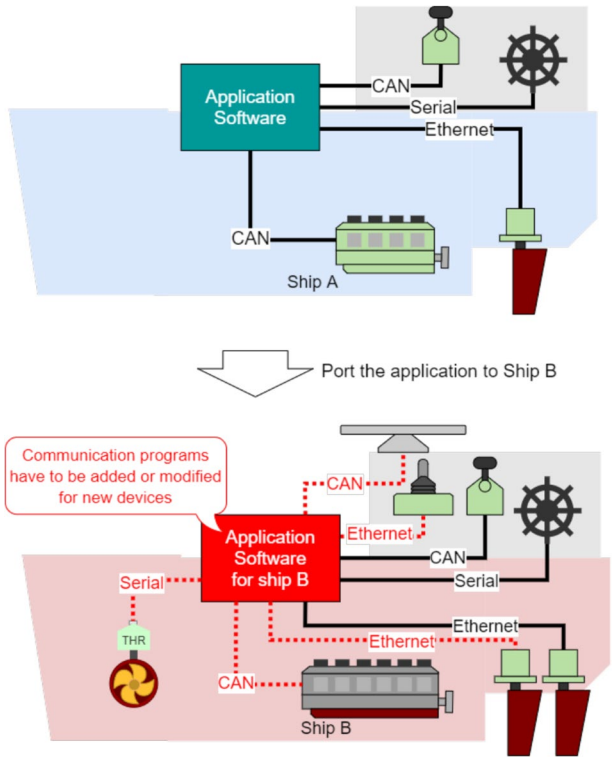

Fig. 2 Porting application to another ship without driver layer

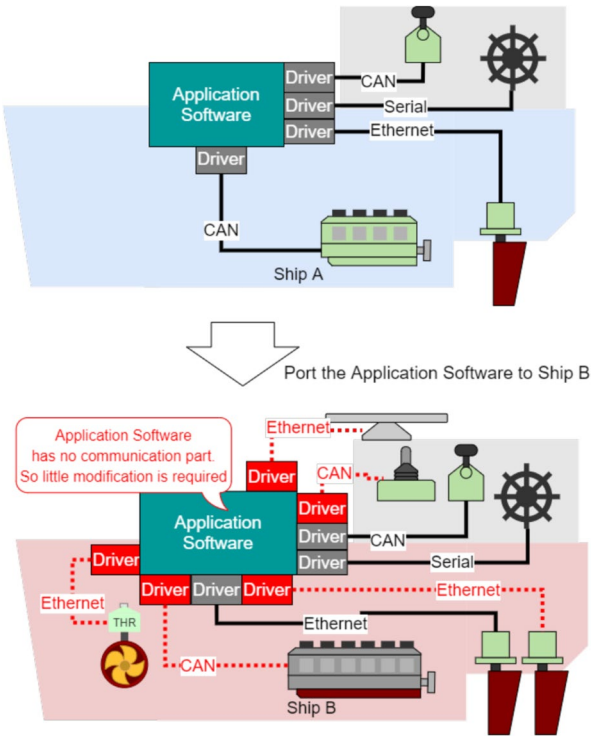

Fig. 3 Porting application to another ship with driver layer

– If a ship has the same devices, communication implementation on all application software is redundant.

Hence, the authors propose a design of system structure in which the application software and communication part are separated by introducing a distributed architecture. Distributed architecture [14] is considered to be the more appropriate architecture for a customizable system. For example, the distributed architecture is applied for vehicle systems [15]. This architecture has a feature that all hardware is modularized and connected by a common bus. In the proposed structure design, the authors introduce a device driver that abstracts the hardware and connects with application software by a common communication bus along with this distributed architecture. The device driver is defined as a bridge software between application software and a particular device. In this structure design, application software has the only way to communicate with hardware via the device driver. Introducing this structure, the different communication protocols specified by each device are absorbed by a device driver, and application software and the device driver communicate with each other by only using a standardized communication protocol. In other words, hardware is hidden and abstracted from the application software side. Then, it helps application software port to different ships. Using this design, communication parts are established as independent execution units, and application software is not affected by the difference in the system's consistency (Fig. 3). Moreover, if there are devices that use the same communication protocol, the device driver can be reused by multiple executions.

In the next section, it is explained the details of application software and device driver design.

3 Detailed design

In the previous section, we described the advantages of introducing a device driver that abstracts real hardware from application software. In this section, it is described that the detailed design of the system.

3.1 Communication protocol between application software and device driver

The main concept of the proposed system is to separate the hardware's unique communication part from the application software. In this design, the communication protocol used between application software and device drivers has to be defined. Application software and device drivers are independent programs from each other. In this case, the protocol used for a computer network is one of the typical methods to exchange data between two instantiated programs: such as UDP/IP or TCP/IP (UDP: User Datagram Protocol [16],

TCP: Transmission Control Protocol [17], IP: Internet Protocol [18]). These protocols are defined as the Internet Protocol Suite by IETF (Internet Engineering Task Force). These protocols are practically the standard for current networking technology and are so widely accepted that they are implemented in most OS (Operating Systems). Also, recent network infrastructures have usually adequate line bandwidth. Both application software and device drivers are expected to be executed on commercially supplied computers; thus, using these network-based protocols to exchange data is considered to be one of the practical options. There are many data exchange protocols based on UDP/IP and TCP/IP, such as MQTT (Message Queue Telemetry Transport) [19] defined by OASIS (Organization for the Advancement of Structured Information Standards), DDS (Data Distribution Service) [20] admired by OMG (Object Management Group), etc. These protocols are based on network technologies, and make it possible to execute application software and device drivers on different machines. On the other hand, if application software and device drivers are executed on the same memory, it is also available a way to use shared memory to exchange data. This method has the advantage of transport and calculation speed. Some DDS implementation has also a method using shared memory, such as FAST DDS [21]. Afterwards, the term"internal network"is defined as the network consisting of application software and device drivers using some protocol such as explained above.

3.2 Design of device driver

As mentioned in Sect. 2, the role of device drivers is to absorb the differences in communication protocols of each hardware, and it enables application software to communicate with hardware by a single method. Showing an example of a system which has LiDAR(UDP connection), GNSS(RS-422 connection), AIS(NMEA2000 connection), an engine (J1939 connection), and an embedded computing unit with application software that consisted of several processes. As shown in Fig. 4, processes are enabled to access to the hardware by a single communication method (e.g., MQTT) by including drivers for each device despite the different communication methods of each device.

Drivers themselves function as programs that relay communications with each piece of hardware and translate between protocols. When designing a driver, it is important to understand deeply the hardware-specific communication protocol and implement it correctly. It is also essential to recognize that protocol conversion requires a finite amount of time. In particular, when developing control-oriented application software, such as for autonomous systems, the real-time nature of data is often critical. Therefore, in the design phase, one must pay careful attention to the characteristics of each device's communication method so as not to inadvertently alter the original transmission intervals or introduce excessive conversion latency. If very high update rates are not necessary, however, techniques such as downsampling data may be helpful to conserve bandwidth. Additionally, assigning timestamps to each measurement, derived from a single, unified clock, can be an effective means of synchronizing time across multiple sensors.

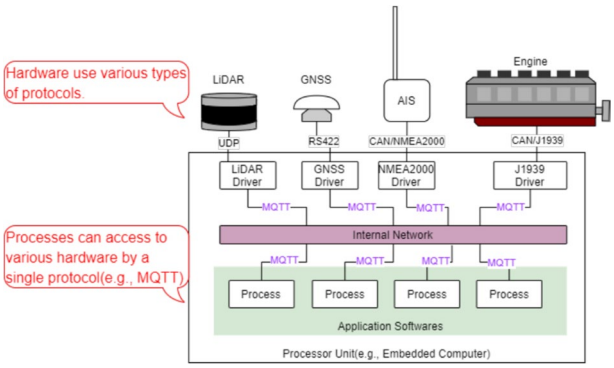

Fig. 4 Example of devices and process connection via drivers

Table 1 Example of a device that is equipped on ships

Device	Protocol(Example)
Engine	CAN, wire
Rudder	CAN, Ethernet, Analog, Serial
Hydraulic device	CAN, wire, Serial
Electric actuator	CAN, Ethernet, Serial
Satellite device (GNSS, internet, etc.)	CAN, Ethernet, Serial
Position measuring device (LiDAR, Radar, etc.)	CAN, Ethernet, Serial

CAN Control Area Network, *GNSS* Global Navigation Satellite System, *LiDAR* Light Detection And Ranging

The protocol of the internal network side is described in Sect. 3.1. On the other hand, on the hardware side, there are many types of devices, and each of them uses various protocols. Showing examples of devices equipped on ships, and communication protocols on Table 1.

In Table 1, a device actuated by some physical links is designed to be controlled by humans using some handles. For controlling such devices, it is required to equip some actuators to move electronically. In addition, computers cannot deal with analog signals directly. For using these devices, signals have to be converted by D/A and A/D converter (D/A: Digital to Analog, A/D: Analog to Digital).

Digital communication protocols mentioned in Table 1 have more various detailed protocols. For example, there are several standardized protocols in serial communication methods, such as EIA-232-D, and USB (Universal Serial Bus).

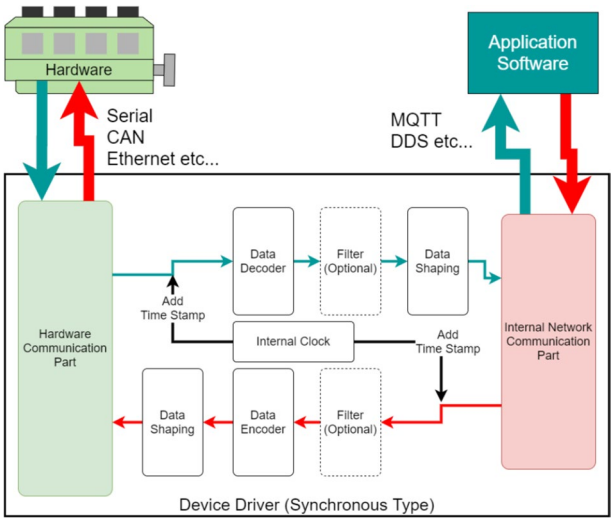

Fig. 5 Fundamental device driver structure (Sync. Type)

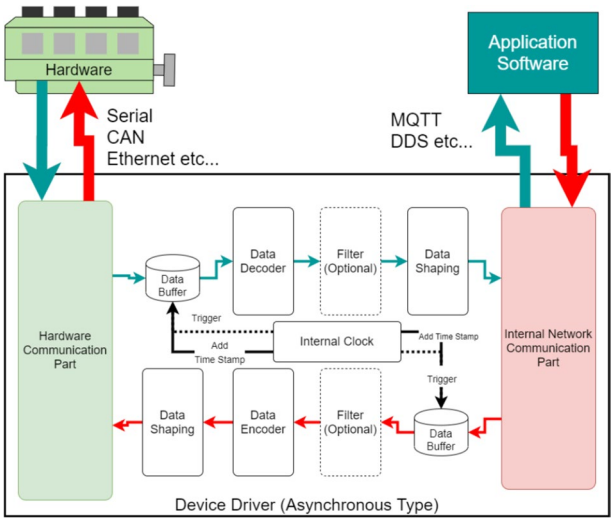

Fig. 6 Fundamental device driver structure (Async. Type)

To develop a device driver, it is required to adapt its design for each device specification, along with used communication protocol. In Fig. 5, a fundamental device driver structure is shown.

Figure 5 represents a synchronous type device driver. In this type, the device driver enters to procedure to convert and publish data just a time when it receives data(Synchronized Mode). In this mode, the time interval from hardware send until application software receiving, is substantially derived from only data transfer and conversion. So this type of device driver can reduce data delay as much as possible. If the hardware communication frequency and Internal Network communication frequency are the same, a synchronous device driver is considered to be one of the practical options.

However, in some cases, such as reducing data rate, Internal Network communication frequency has to be different from the hardware communication. In such cases, an asynchronous mode can be applied. The data diagram of the asynchronous type device driver is shown in Fig. 6.

The asynchronous type device driver receives data and stores it in a buffer at once. The rate of transferring data is regulated by the inner clock of the device driver, and data is published by a trigger generated by the clock. In this procedure, the newest data stored in the buffer is usually used, so some old data may possibly be abandoned. Also, it should be kept in mind that the published data was possibly received in the past. Figure 7 represents the case that a device driver receives data with a higher frequency than the publishing frequency. In asynchronous mode, published data is somehow delayed. Thus it is required to be designed to use a higher publish frequency than receiving frequency, not to skip data.

In any type of driver, the published data from the driver contains some delay. In case an application software requires severe on-time data, such as a control system, even nanosecond or millisecond delay might induce harmful effects. Thus, developers have to choose an appropriate way along with a specification of the application software.

3.3 Design of application software

In this paper, application software is defined as software for realizing a function of ship control. It includes a new algorithm or a method and behaves as a main component. It is usually an object to be evaluated in the experiment using an experimental system. In a huge sophisticated system such as an unmanned autonomous ship, such main software is sometimes described as separate programs. They are executed independently, and programs exchange messages with each other. Additionally, some utility software that helps experiment, such as a data logger or UI(User Interface), is also described as application software. Thus normally, multiple application software are executed on a system.

For evaluating such application software, it has to be implemented along with the proposed system structure. Next, we show an example to develop application software for a Zig-Zag test. A Zig-Zag test is one of the typical experiments to measure specifications of ship response by rudder control. The experiment is executed along with the next method:

1. Running in constant speed(rotating propeller in constant rotation speed.).
2. Steering rudder until a determined angle.
3. If the ship's heading reaches a determined heading, it steers the rudder until the inverse angle of a determined angle.

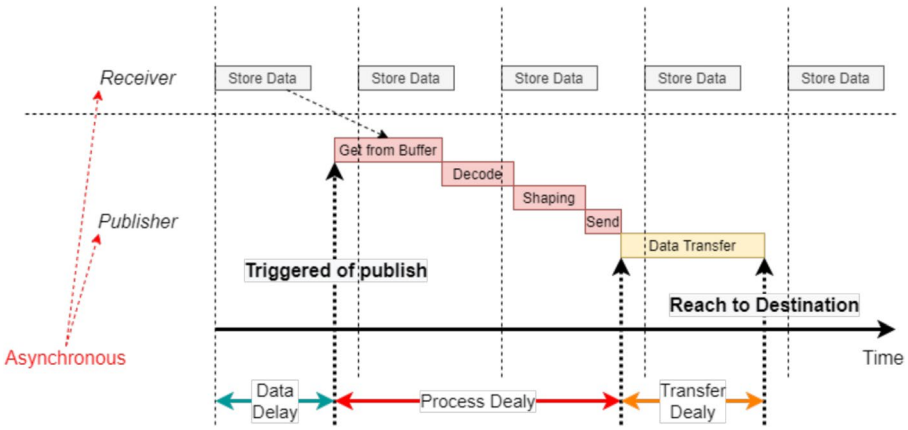

Fig. 7 Data Delay in asynchronous mode

4. Repeat the previous step and record the angle of the rudder, the ship's heading, and a time stamp.

To realize the method above, the application software needs the following information:

– Ship's heading(Sensing by gyroscope sensor)
– Rudder actual angle(Receiving from rudder driver)

Also, the application software needs to send commands to the next device:

– Propeller rotation number to the main actuator.
– Target angle of Rudder.

Then, the main application software for the Zig-Zag test can be designed as Fig. 8.

Similar to device drivers, application software can be developed in synchronous mode or asynchronous mode. In the Fig. 8 case, the asynchronous type is shown. The application receives the ship's heading from a gyroscope and the rudder angle from a rudder driver. Received data are stored in a buffer. The program cycle is regulated by the inner clock. In every cycle, the program fetches the newest data from the buffer and evaluates the next step's behavior. If the heading angle reaches to designated value, a steering command is generated and sent to the rudder device driver. In Fig. 8, a data logging tool is also represented as application software. It is not a main object for a control ship, but it is a utility tool.

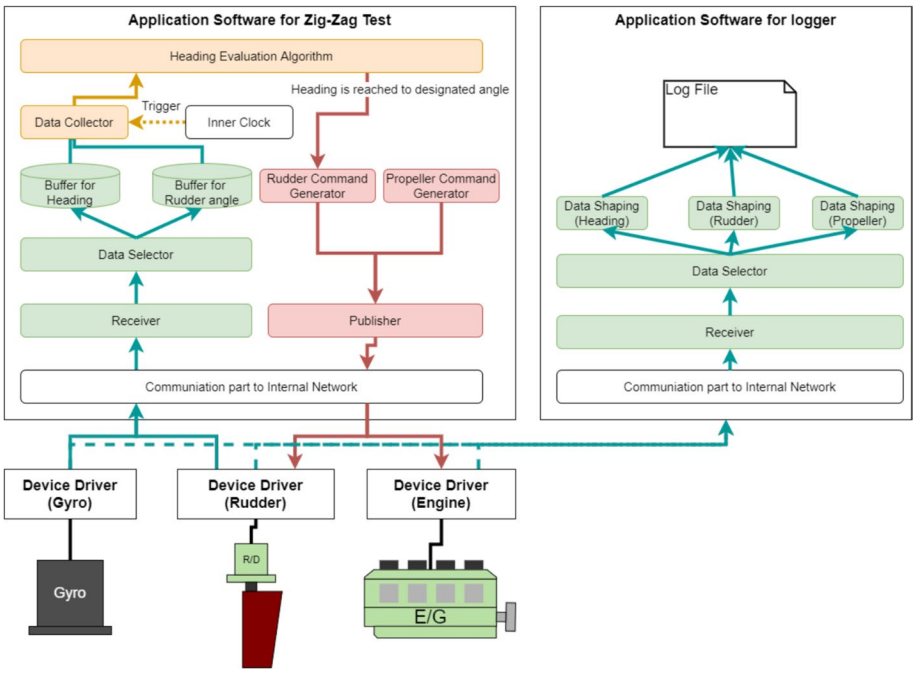

Fig. 8 Example application software Structure for Zig-Zag Test

4 Implementation example

In this section, the example of implementing a real ship experiment system is shown. We show the implementation method of our proposed structure and the result of one of the simple operations (Zig-Zag test) using the system. In addition, we show the system's portability by a trial to transplant the system to another type of ship.

4.1 Model ship specification

For this experiment, we used a model tanker ship as shown in Fig. 9, which was developed by authors in 2019 at Osaka University [22]. Its hardware structure is shown in Fig. 10. It had one engine (motor), one rudder, and several sensors: a rotation counter of the propeller, a rudder angle sensor, a gyroscope, two anemometers, and an RTK-GNSS (Real Time Kinematic-GNSS). It had also a main computer (NVIDIA Jetson TK1) to which Linux Ubuntu 14.04 is installed. All devices were connected to the computer directly or via a network hub.

Fig. 9 Exterior of the model ship

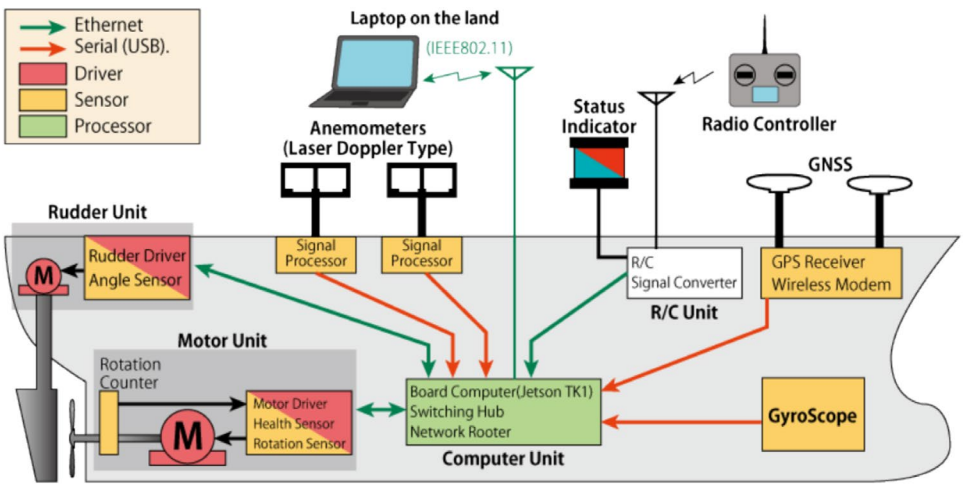

Fig. 10 Hardware structure of the model ship

4.2 System implementation

To implement the experiment system, the authors introduced ROS (Robot Operating System) [23]. It is one of the platforms for developing robots (robots in this context are represented as a kind of sophisticated system to behaves automatically). ROS contains the definition of a framework for building and executing multiple programs (ROS Nodes), communication protocols between ROS Nodes(ROS Topics, ROS Services, etc.), and surrounding ecosystems such as tools for logging, visualizing, package management, etc. ROS Nodes are defined as the execution unit of the program. In this implementation, all of the device drivers and application software that the authors proposed are implemented as ROS nodes. ROS Nodes exchange data with some defined message protocols: ROS Topic(Asynchronous communication), ROS Service (Synchronous communication), etc. These communication methods are implemented with TCP, which is one of the fundamental protocols of the Internet. TCP is a protocol that places relatively greater emphasis on reliable data delivery through mechanisms such as retransmission and congestion control. Although it offers lower real-time performance than UDP, leveraging the ROS ecosystem enables rapid development. Moreover, in a wired system without high-bandwidth sensors, communication remains stable, and since nanosecond-scale latency has virtually no impact compared to the poor inertial response of the control subject, we adopted this approach.

Device drivers were implemented as ROS nodes that translate between each device's proprietary communication protocol and the TCP-based ROS Topic interface. The vessel is equipped with a motor (UDP communication), a rudder (UDP communication), a gyroscope (EIA232 communication), an anemometer (UDP communication), a GPS receiver (EIA232 communication), and an R/C transceiver (UDP communication). Taking into account the characteristics of each device, we designed

and implemented the drivers as ROS nodes. For the motor and rudder, rapid response to manipulate commands from the processor is essential, while the actuators themselves operate more stably when driven at fixed intervals regardless of the processor's polling rate. Accordingly, these drivers were implemented as the asynchronous programming model described in Sect. 3.2. The gyroscope and anemometer both transmit data periodically via serial interface and UDP, respectively, at rates that exceed the control loop frequency (10Hz order). Because serial communication involves sequential reading from a buffer, making tight synchronization with UDP streams problematic, these drivers were likewise implemented asynchronously. Although the GPS emits data at a fixed rate that approximates the control cycle-making a synchronous implementation technically feasible. Then, we chose to implement its driver asynchronously as well, to unify the programming model across the other device interfaces.

As application software, we designed and implemented the Zig-Zag test application software as ROS Nodes which was designed in the Sect. 3.3. As an experiment helper tool, data recording ROS Nodes was also implemented.

Finally, all nodes were deployed on the computer on the model ship as shown in Fig. 11.

4.3 Experiment with the implemented system

With using the system, the Zig-Zag test was executed in 2018 February, at Inukai Pond at Osaka University. In this experiment, the threshold of the heading angle was set as ±20 [deg], and the designated rudder angle was set as ∓20 [deg]. The result of the experiment is shown in Fig. 12. This figure is generated from the experiment log file which is logged by the logger node.

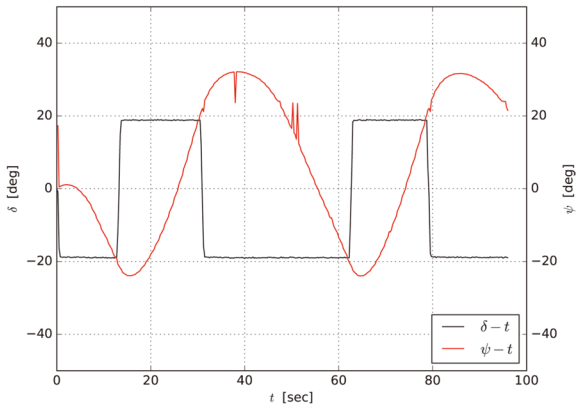

Fig. 12 Result of Zig-Zag experiment with the implemented system

4.4 Porting to another ship

To verify the scalability of the system implemented on Model Ship Type A, we attempted to port the Zig-Zag test system developed in this study to a different platform (Model Ship Type B). Although Type B shares the same overall system architecture as Type A, it is equipped with a single bow thruster and a Vec-Twin dual-rudder configuration. As shown in Fig. 13, we therefore added new device drivers for these components. The bow-thruster driver was adapted from the main motor driver, but the thruster itself was not used during the Zig-Zag test. The Vec-Twin rudders were implemented to operate in synchrony, yet were designed to subscribe to the same ROS Topic as the single rudder on Type A. Thanks to this driver-level abstraction, the Zig-Zag test application software ran unchanged on both Type A and Type B platforms – requiring 0% code modifications – and thus clearly demonstrates the advantage of isolating hardware differences within the device drivers. This approach makes it possible when deploying a feature under development simultaneously on vessels with differing

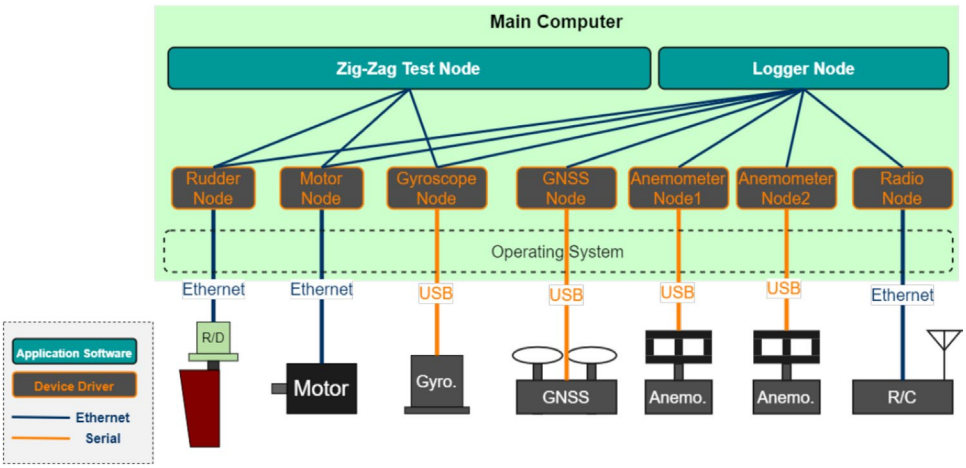

Fig. 11 Structure of Application Software and Device Drivers of the model ship

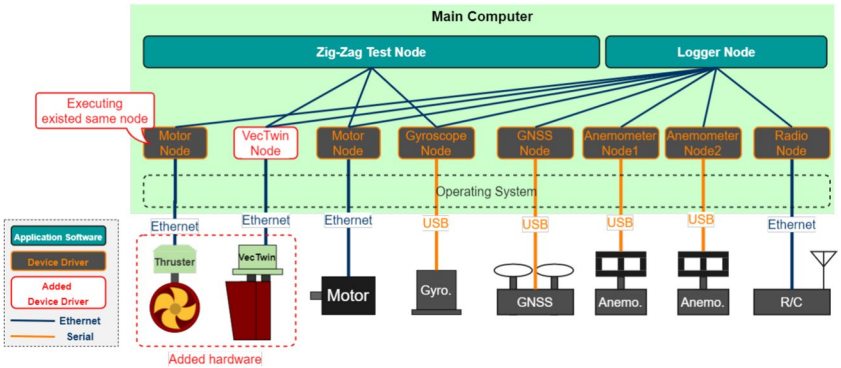

Fig. 13 Structure of Application Software and Device Drivers of another model ship

configurations, to support multiple ship types by developing only a single version of the application software, provided that the device drivers can absorb the hardware differences. This, in turn, reduces development costs and accelerates return on investment. Because each vessel may use different manufacturers or vary in detailed specifications to meet owners' requirements and technical constraints, even when offering the same functionality, the system modularization afforded by our proposal is significant. Of course, in more complex applications –such as autonomous navigation systems– where inputs and outputs change as functions are added or removed, some modification of the application software may still be required. The scope of those changes also depends on the device driver implementation, and function additions or deprecations may necessitate further adaptation. Nevertheless, the modular structure of the proposed system confines design changes to specific modules, and in particular for large-scale systems, it is expected to reduce the volume of code requiring retesting substantially.

5 Discussions for practical development

This section discusses the practical application of the proposed design methodology in the development of integrated systems for ships, such as those presented thus far. In particular, when developing systems that are deeply involved in ship operations, such as autonomous navigation systems, it is critically important to ensure system reliability. The challenge of assuring the safety of complex systems has long been a subject of debate. However, in recent years, the result of the Regulatory Scoping Exercise(RSE) on standards for autonomous ships [24] has been reported by the International Maritime Organization (IMO), and classification societies in various countries have begun to formulate guidelines for the design and evaluation of autonomous ship systems [25–29]. As a result, system requirements, development processes, and related aspects are gradually being clarified. In anticipation of the future developments where system design and evaluation will be expected to align with these emerging standards, this section discusses both the advantages that our proposed methodology can offer developers and the future efforts needed for its effective implementation.

In addition to autonomous navigation functions, maritime systems are required to possess a high level of safety and reliability. Equipment certification systems have been established by various national governments to ensure the reliability of maritime equipment. Examples include the Type Approval system in Japan (Regulations for Type Approval of Vessels and Other Equipment) and the Marine Equipment Directive (MED, Directive 2014/90/EU of the European Parliament and of the Council) in the European Union. Maritime systems that have obtained such approvals are allowed to omit certain tests during classification inspections. However, if systems are developed individually for each vessel, safety testing would be required for each case, resulting in excessive costs and extended development timelines. Therefore, it is essential in maritime system development to design systems that are applicable to as many types of vessels as possible and to obtain type approval.

The authors propose a method to modularize the core software (application software) of maritime systems by absorbing communication protocol differences of peripheral devices through device drivers, thus enabling software commonality across different vessels. If type approval can be granted separately for application software and device drivers, developers may be able to combine pre-approved software components, limiting the scope of testing required for each development while maintaining reliability and improving development efficiency. In particular, device drivers, which interface with hardware components such as sensors and propulsion systems can be developed and type-approved by manufacturers who possess detailed knowledge of their own products. By packaging hardware and corresponding driver software together, this approach is expected to reduce the certification burden on maritime system developers (e.g., system integrators) and promote the development of safer and more diverse maritime systems.

However, to realize this kind of distributed development, it is essential to standardize and unify the communication

protocol that connects application software with device drivers. While application software and device drivers are generally installed on the same computing unit, modularization of application software by function may lead to the deployment of components across different computational platforms, requiring inter-device communication over a network. In either case, Ethernet appears to be the most suitable physical-layer protocol, as it facilitates network construction, supports a variety of protocols, and offers both high transmission reliability and sufficient bandwidth. Ethernet supports high-bandwidth communication –up to several hundred Gbps–via metallic cables or optical fibers, making it capable of transmitting large volumes of sensor data such as images or point cloud data of a LiDAR or a Radar. Furthermore, the bandwidth at the physical layer enables the integration of various higher-layer protocols, making it easy to implement features such as encryption and mutual authentication. Nonetheless, several considerations must be kept in mind when constructing networks based on Ethernet, which will be discussed in the following sections.

5.1 Throughput and delay

Because Ethernet allows all nodes to share part or all of the communication path, it is essential to implement countermeasures against network failures and delays. In particular, control communications necessary for ship maneuvering and sensor data must ensure sufficient data throughput and prevent malfunctions caused by delays or data loss. Therefore, the overall network traffic should be minimized as much as possible, and for critical communications, network separation from other communication systems and prioritization through Quality of Service (QoS) mechanisms are required. It is also considered necessary to implement measures such as redundancy for critical communication systems, including those related to navigation, to ensure continued operation in the event of transmission line failures.

On the software side, systems utilizing the network must take into account the possibility of some degree of latency or data loss. It is important to implement mechanisms such as timestamps to detect data losses or delays beyond expected levels. While the reference time for timestamps is often based on the Real-Time Clock (RTC) of the computer on which the software is installed, when multiple computers engage in mutual data communication, synchronization of clocks becomes necessary. This can be achieved using protocols such as Network Time Protocol (NTP) or Precision Time Protocol (PTP). On land, it is possible to access public time synchronization services via the internet. However, ships are often isolated from terrestrial networks, and thus it is necessary to consider incorporating dedicated time synchronization servers onboard as part of the system design.

5.2 Network security

As ship systems become increasingly sophisticated, the potential impact of cybercrime also grows. Therefore, robust measures must be taken to ensure cybersecurity. This is particularly critical in systems such as autonomous ships, where unauthorized access or malicious interference with communications could lead to hijacking or acts of terrorism. Accordingly, there is a pressing need to establish standardized guidelines for cybersecurity countermeasures.

Cyberattacks may include eavesdropping on communications, tampering with data, or unauthorized acquisition of administrator privileges. To counter these threats, technologies developed for terrestrial networks, such as encrypted communication, tamper detection, and segregation between closed and open networks, can be generally adapted for maritime systems as well. In closed systems that are not connected to the Internet, the primary risks for unauthorized access or malware introduction occur during software updates. However, in systems designed for remote ship operation, which require connections to public networks, extra caution is required. In such cases, a thorough verification process must be implemented during both the outfitting and operational phases to ensure that all security configurations are complete and no vulnerabilities remain.

6 Conclusion

This paper proposed a method for efficiently developing and evaluating shipboard systems by designing communication with hardware through device drivers, thereby absorbing hardware-specific specifications. Since the configuration of ships varies from vessel to vessel, thorough evaluation of each system is essential to ensure safety, and certification is also required in practical development processes. A system design that can be applied across multiple ship types is expected to improve development efficiency and contribute to the widespread adoption of more advanced systems, such as autonomous ships. To achieve this, it will be necessary to establish industry-wide standards not only for hardware configuration but also for certification at the software module level and for safe and efficient interconnection between software modules. Furthermore, although evaluation using real ships is indispensable, it is difficult to conduct such evaluations frequently due to constraints of cost and time. Therefore, the use of simulators will also be important to further enhance development efficiency. Future work will include investigating methods to integrate real-ship development systems with simulators.

Funding Open Access funding provided by The University of Osaka.

References

1. Witherby Publishing Group (2021) Seafarer workforce report: the global supply and demand for seafarers in 2021. Tech. rep, BIMCO and International Chamber of Shipping
2. Wariishi K (2019) Maritime autonomous surface ships: development trends and prospects -how digitalization drivies changes in maritime industry-. Tech. rep. Mitsui & Co. Global Strategic Studies Institute
3. The Nippon Foundation (2024) The Nippon Foundation MEGURI2040 Fully Autonomous Ship Program (2022) https://www.nippon-foundation.or.jp/en/news/articles/2022/20220118-66716.html. Accessed 24 Jan 2024
4. Autonomous Shipping Initiative for European Waters, Autoship (2024) https://www.autoship-project.eu. Accessed 24 Jan 2024
5. Kondo M, Shoji R, Miyake K, Zhang T, Furuya T, Ohshima K, Inaishi M, Nakagawa M (2018) The "Watch" Support System for Ship Navigation. In: Human Interface and the Management of Information. Information in Applications and Services. Las Vegas, U.S.A., pp 429–440. https://doi.org/10.1007/978-3-319-92046-7_36
6. Suzuki T (2021) Challenge of technology development through MEGURI 2040. Tech. Rep. 3 Class NK
7. Inoue S, Mori H (2021) Development of automated ship operation technologies. Tech. Rep. 3 Class NK
8. Ahmed YA, Hasegawa K (2013) Implementation of automatic ship berthing using artificial neural network for free running experiment. IFAC Proc. Vol. 46:25–30. https://doi.org/10.3182/20130918-4-JP-3022.00036
9. Lee S, Yu C, Hsiu K, Hsieh Y, Tzeng C, Kehr Y (2010) Design and experiment of a small boat track-keeping autopilot. Ocean Eng. 37(2):208–217. https://doi.org/10.1016/j.oceaneng.2009.11.005
10. Morawski L, Pomirski J (1998) Ship track-keeping: experiments with a physical tanker model. Control Eng. Pract. 6(6):763–769. https://doi.org/10.1016/S0967-0661(98)00082-3
11. Hirdaris S, Mikkola T (2021) Ship dynamics. J Mar Sci Eng 9(2):105. https://doi.org/10.3390/jmse9020105
12. Huijgens L, Vrijdag A, Hopman H (2021) Hardware in the loop experiments with ship propulsion systems in the towing tank: scale effects, corrections and demonstration. Ocean Eng 226:108789. https://doi.org/10.1016/j.oceaneng.2021.108789
13. Perera L, Moreira L, Santos F, Ferrari V, Sutulo S, Soares CG (2012) A navigation and control platform for real-time manoeuvring of autonomous ship models, pp 465–470. https://doi.org/10.3182/20120919-3-IT-2046.00079
14. Nunes U, Fonseca A, Almeida L, Araújo R, Maia R (2003) Using distributed systems in real-time control of autonomous vehicles. Robotica 21:271–281. https://doi.org/10.1017/S0263574702004770
15. Jo K, Kim J, Kim D, Jang C, Sunwoo M (2014) Development of autonomous car-part I: distributed system architecture and development process. IEEE Trans Ind Electron 61(12):7131–7140. https://doi.org/10.1109/TIE.2014.2321342
16. Postel J (1980) RFC-768 User Datagram Protocol. https://www.rfc-editor.org/rfc/rfc768. Accessed 24 Jan 2024
17. DARPA Information Processing Techniques Office (1981) RFC-793 transmission control protocol -DARPA internet program protocol specification. https://www.rfc-editor.org/rfc/rfc793. Accessed 24 Jan 2024
18. DARPA Information Processing Techniques Office (1981) RFC-791 Internet Protocol -DARPA Internet Program Protocol Specification. https://www.rfc-editor.org/rfc/rfc791. Accessed 24 Jan 2024
19. Organization for the Advancement of Structured Information Standards (2018) Index of /mqtt/. https://docs.oasis-open.org/mqtt/. Accessed 24 Jan 2024
20. Object Management Group Inc (2021) DDS Foundation Wiki. https://www.omgwiki.org/ddsf/doku.php. Accessed 24 Jan 2024
21. eProsima (2024) Fast DDS -The most complete open source DDS middleware. https://www.eprosima.com/index.php/products-all/eprosima-fast-dds. Accessed 24 Jan 2024
22. Wada S, Maki A, Umeda N (2019) General purpose free-running model ship with ROS -enhanced model ship experiments system. In: Conference Proceedings The Japan Society of Naval Architects and Ocean Engineers, vol. 28. Nagasaki, Japan, pp 587–594. https://doi.org/10.14856/conf.28.0_587
23. Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R, Ng A (2009) ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software, vol. 3
24. IMO MSC.1/Circ.1638 (2021) Outcome of the regulatory scoping exercise for the use of maritime autonomous surface ships (Mass)
25. Lloyd's Register (2016) Cyber-enabled ships ShipRight procedure: autonomous ships
26. DNV (2018) CLASS GUIDELINE Autonomous and remotely operated ships, DNVGL-CG-0264
27. Bureau Veritas Marine & Offshore (2019) Guidelines for autonomous shipping, guidance note NI 641 R01 (2019)
28. Class NK (2025) Guidelines for automated/autonomous operation on ships (Ver.2.0)
29. American Bureau of Shipping (2021) Guide for Autonomous and Remote Control Functions