Hindawi
Complexity
Volume 2018, Article ID 7081501, 21 pages
https://doi.org/10.1155/2018/7081501

WILEY | Hindawi

Research Article

Interface Data Modeling to Detect and Diagnose Intersystem Faults for Designing and Integrating System of Systems

Kyung-Min Seo and Kwang-Phil Park

Naval and Energy System R&D Institute, Daewoo Shipbuilding and Marine Engineering, Seoul 04521, Republic of Korea

Correspondence should be addressed to Kyung-Min Seo; kmseo.kumsung@gmail.com

Received 25 April 2018; Revised 16 July 2018; Accepted 1 August 2018; Published 14 October 2018

Academic Editor: Zhiwei Gao

In system of systems engineering, system integrators are in charge of compatible and reliable interfaces between subsystems. This study explains a systematic solution to identify and diagnose interface faults during designing and integrating systems of systems. Because the systems targeted in this study are real underwater vessels, we first have anatomized 188 interface data transferred between 22 subsystems of them. Based on this, two interface data models are proposed, which include data sets regarding messages and inner fields and transition and decision functions for them. Specifically, a structure model at the message level evaluates how inner fields belong to a message, and a logic model at the field level assesses how each field is interpreted and if the interpreted value is understandable. The software that supports the modeling is implemented using the following concepts: (1) a model-view-viewmodel pattern for overall software design and (2) a computer network for representing sequential properties of field interpretations. The proposed modeling and software facilitate diagnostic decisions by checking the consistency between interface protocols and obtained real data. As a practical use, the proposed work was applied to an underwater shipbuilding project. Within 10 interfaces, 14 fault cases were identified and diagnosed. They were gradually resolved during the system design and integration phases, which formed the basis of successful submarine construction.

1. Introduction

A complex system such as automotive, marine, or aerospace system of systems (SoS) contains diverse subsystems that must be designed and integrated to work together [1–3]. In an underwater vessel, for example, an inertial navigation system (INS) receives speed over water and locational information from an electromagnetic log (EM log) and Global Positioning System (GPS), respectively [4]. They enable the INS to enhance the computational accuracy of its orientation and velocity. In this context, the input data of the INS have been utilized for computing the INS's outputs precisely; besides, the outputs also are a basis of estimating the geographic location of the vessel in water. Thus, understandable and reliable interfaces between the subsystems are the main prerequisite to organize the subsystems as an integrated system at the corporate level [5–7].

When designing and integrating the complex SoS, a system integrator has difficulty figuring out interface faults for the following reasons. First, both subsystems of an interface are commonly developed by different manufacturers, which gives rise to disparate implementations of the same interface protocols [8]. Furthermore, for easy modifiability and scalability, the manufacturers still prefer customized protocols to those that are standardized [9, 10]. In this regard, the protocols are occasionally revised during the system design phase as well as the integration phase.

This study suggests a systematic solution about how a subsystem successfully interacts with a counterpart one when they are designed and integrated for the whole system. Specifically, we have focused on resolving the interface faults (i.e., anomalies) during node-to-node delivery over the digital network. Our goal in this study is to ensure compatible and reliable interfaces by checking the

consistency between the interface protocols and obtained interface data.

In order to transfer sensitive information, a sending system encodes interface data (i.e., messages) so only authorized receivers can understand [11]. The encoding rules contain what payloads (i.e., fields in this study) are structured in a message or how each field is logically converted, which are described in the interface protocols [12]. Because the protocols including the structural and logical rules are diverse and complicated, they should be preferentially explored for overcoming the interface faults. To this end, we have analyzed interface protocols used in domestic ship systems that are already in operation or under construction. Messages transmitted via radar, navigation, acoustic, and optical sensor systems as well as several control systems were investigated in this study. We atomized 188 message types that interacted between 22 subsystems for message structures and field logics.

Based on the preanalysis, we proposed two modeling formulas, which contain data sets and functions [13]. A structure model at the message level formalizes how many fields belong to a message and what makes it to be structured, and a logic model at the field level assesses how each field is interpreted and whether the interpreted value of the field is understandable or not for receiving systems. The models fundamentally receive interface data as input. Then, they detect and diagnose the faults via the transition functions and output the results through the decision functions. In the proposed modeling, we practically classified five structure types at the message level and modularized several transition functions at the field level. With the proposed formalism, a modeler can specify the interface protocols and diagnose the faults containing messages and inner fields in a systematic rather than an ad hoc manner [14–16].

Over the last decade, several studies for fault detection and diagnosis methods have been developed for various systems and applications. Some researchers have developed system models for representing real systems by checking model-predicted outputs and obtained system outputs [17–19], and others have centered on output signals of the systems to analyze their features or patterns for fault detection and diagnosis [20–22]. This study combined these two methods. We focused on input/output (I/O) signals within digital interfaces; at the same time, the signals are explicitly formalized in the two-level models to detect and diagnose interface faults of an arbitrary interface. To the best our knowledge, no work has been reported toward focusing on fault detection and diagnosis during the system design and integration phases.

To realize the proposed models in an effective way, we have used the model-view-viewmodel (MVVM) design pattern in Windows Presentation Foundation (WPF) technology [23]. In the developed software, block libraries for modeling elements have been provided to illustrate the benefits of a graphical modeling environment. In addition, a computer network concept has been applied, which is based on the concept of using nodes and connections to create an overall logic modeling. Thus, it facilitates intuitive modeling via libraries regarding structural delimiters and logical operations and allows flexible modeling through their creation and revision.

As a practical use, the proposed work was applied to an underwater shipbuilding project, namely, a submarine renovation project [24]. Ten digital interfaces connected to improved subsystems were examined to resolve the interface faults. Seven tests were performed at sea to find the faults for various operational situations, and two tests that allow the ideal preparation for sea-trial tests were conducted in a harbor. The empirical results showed that 14 fault cases, which are either structural or logical, were detected and diagnosed during designing and integrating the renovated submarine. These incorrect patterns in the interfaces were successfully resolved during this project.

The study is organized as follows. Section 2 describes our focus of fault scope, and Section 3 analyzes previous works. Section 4 proposes modeling methods and realization of the modeling as a software tool. Section 5 explains and discusses an application for the shipbuilding project. Finally, Section 6 presents our conclusions.

2. Background

A fault is defined as an unpermitted deviation of at least one characteristic property or parameter of a system from the acceptable, usual, or standard condition [25]. Because the various cases of faults can occur when a system is under development as well as in operation, the fault scope interested in this study needs to be clarified here.

2.1. Interface Faults in Complex System Development. Figure 1 shows a simplified illustration of how a fault is identified and diagnosed in complex shipbuilding engineering. As explained in the introduction, the basic concept for resolving the fault is to evaluate the consistency by comparing with the interface data and the interface protocol including structural and logical rules.

The ship system as an SoS is generally composed of diverse sensing equipment and dynamic systems, which are incorporated into an integrated system [26, 27]. It has been noted that the majority of end systems such as sensors or actuators do not plug directly into the central network [28]. Instead, they connect to a local proxy with each point-to-point link, which in turn is distributed across a central bus network. The signal-processing unit, data integration system (DIS), and integrated management system in Figure 1 act as such proxies.

When designing and integrating the subsystems for the overall system, faults can be found in the central network as well as the outside of the network, specifically in the point-to-point links between the local subsystems [29]. In this study, we focused on the local faults rather than the central faults for the following reasons. First, the local faults occur more frequently than the central faults due to disparate implementation of the same protocol. This problem accords with the current industrial tendency that interoperability testing for

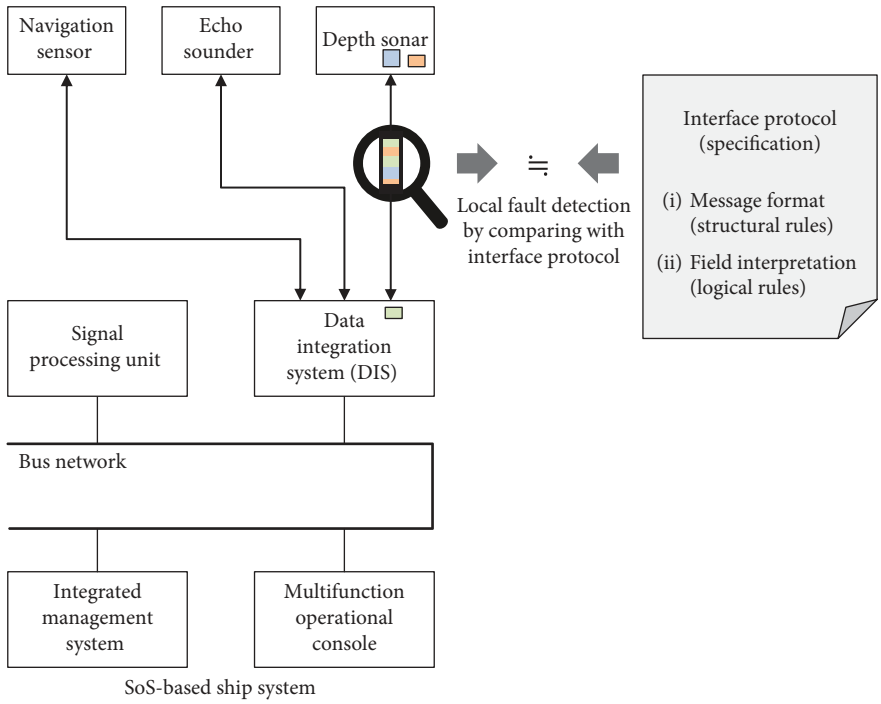


Figure 1: Local faults in SoS-based ship system.

communication between the connected systems becomes more important [30, 31].

Next, the local faults need to be preferentially identified, because these failures obviously influence the central part [32]. For example, if an EM log sends speed information with the wrong unit or resolution, other subsystems using the speed (e.g., navigation sensor or echo sounder) have abnormal behaviors sequentially. Thus, the local faults often have been ascribed to uncontrolled, unanticipated, and unwanted interactions between the subsystems [33, 34].

In this respect, this study focuses on the local faults during node-to-node delivery over digital interfaces when designing and integrating the SoS. The local faults were classified into structural and logical levels, which are explained in the following subsection.

2.2. Classification of Local Faults. Figure 2 illustrates exchanging digital data between two subsystems. In digital communication, the data are a sequential stream of bytes at the physical layer [35]. In this study, we assumed that the byte stream was already transformed into manageable data units (i.e., messages).

An individual message has a common format to be distinguished with different types of messages [36]. For example, a message may be determined to have a fixed-length structure or may include several fields for transmitted information as well as supplemental delimiters such as a header and a footer. Since the message is usually encoded for information security, it eventually needs a conversion process that returns into the original sequence of information [36]. These structural and logical rules are comprised in communication protocol, which have to do with an agreement between both-sided manufacturers and a system integrator.

As shown in Figure 2, the faults in an individual interface data are hierarchically classified into two levels: a structural and a logical fault. The structural fault is incorrectness about the exterior of a message. The wrong length of the message or different delimiters in the message are the structural fault. On the contrary, the logical fault occurs when a field conveyed in the message is semantically incorrect. Uninterpretable data, unmeasurable values, or unspecified status information in the field corresponds to this level.

Because the structural and logical rules are diverse and complicated, they should be preferentially explored to detect and diagnose their faults. To this end, we analyzed interface protocols for real messages transferred in domestic naval vessels. Table 1 shows the preinvestigation regarding two types of real submarine systems. The first-generation submarines have been in operation domestically, and those that are third-generation are under construction. In total, 188 message types in 22 subsystems were anatomized for message formats and field logics. We generalized how the messages were constructed to distinguish from others and which rules were required to interpret the fields for meaningful information.

In summary, this study introduces a practical concept for identifying and diagnosing the structural and the logical faults during the design and integration of the complex SoS. Based on the preanalysis, we formalize a structure model at the message level and a logic model at the field level. The proposed models are implemented to a software tool, which facilitates intuitive and flexible modeling to detect and diagnose the interface faults.

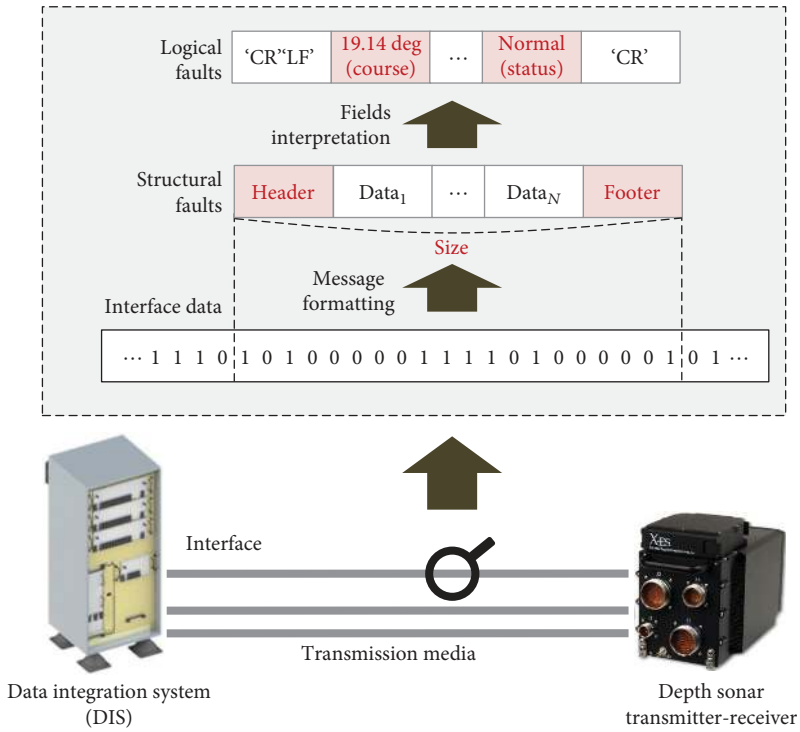


Figure 2: Classification of local faults in point-to-point link: structural and logical faults.

Table 1: Preinvestigated interface protocols in 2 types of submarines.

Submarine type	Number of subsystems	Number of interfaces	Number of messages
First-generation class	11	12	74
Third-generation class	11	31	114
Total number	22	43	188

Subsystems to be analyzed are (1) radar, navigation, acoustic, and optical sensor systems and (2) control systems such as plotting boards and weapons control systems.

3. Literature Review

Over the last decade, several studies for fault detection and diagnosis have been developed for various systems and applications. In this section, we have classified them into three approaches, which are summarized in Table 2.

In model-based approaches, system models are developed to describe the relationships among main system variables [39–41]. Based on the models, fault diagnosis algorithms have been developed to monitor the consistency between the measured outputs of the practical systems and the model-predicted outputs [42]. For example, Cai et al. [17] used object-oriented Bayesian networks (OOBN) to model complex systems. The OOBN-based modeling is classified into structure and parameter modeling that are built with sensor historical data and expert knowledge. Lamperti and Zhao [18] focused on the diagnosis of active systems, and the diagnosis of rules in the proposed finite system machine (FSM) has been specified based on associations between context-sensitive faults and regular expressions. Poon et al. [19] used a model-based state estimator to generate an error residual that captures the difference between the measured and estimator outputs. These model-based approaches require explicit models, whose accuracy determines the diagnosis performance.

On the contrary, signal-based approaches decide diagnostic decisions based on features or patterns of the extracted signals rather than the system models [43, 44]. For example, Loza et al. [20] proposed a nonhomogeneous high-order sliding mode observer to estimate sensor signal faults in finite time and in the presence of bounded disturbances. In Do and Chong's work [21], the vibration signal was translated into an image; the local features were then extracted from the image using scale-invariant feature transform for fault detection and isolation under a pattern classification framework. Pan et al. [22] proposed an acoustic fault detection method, which was addressed for the gearbox based on the improved frequency-domain blind deconvolution flow. The signal-based approaches generally extract the major features of the output signals for fault diagnosis, but they pay less attention to system inputs [45].

This study has combined the two approaches. To detect and diagnose interface digital signals, signal patterns under a normal status were generalized, which were known from the interface protocols. Input signals as well as output signals were targeted because the interfaces interested in this study generally had both-sided signals. Then, the patterns were formalized with mathematical models (i.e., interface data models). The proposed models had explicit sets and functions, and the fault diagnosis within the modeling was carried

Table 2: Summary of related works.

Approach	Previous work	Motivation	Method	Application
Model-based approach	Cai et al. [17]	To eliminate system faults immediately once they occur in complex systems	Structure and parameter models using object-oriented Bayesian networks are proposed.	Subsea production system
	Lamperti and Zhao [18]	To diagnose component faults in complex discrete-event systems during the system evolution	A finite-state machine model is built for fault diagnosis.	Military system
	Poon et al. [19]	To resolve faults in components and sensors in switching power converters	A model-based state estimator is proposed based on a library of fault signatures for possible component and sensor faults in all 4 converters.	Switching power converter system
Signal-based approach	Loza et al. [20]	To reconstruct signal faults as early as possible	A nonhomogeneous high-order sliding mode-based observation approach is proposed.	Aircraft transport system
	Do and Chong [21]	To detect and diagnose vibration signals of the inductor motor	The scale invariant feature transform algorithm is proposed to generate the faulty symptoms.	Three-phase AC motor
	Pan et al. [22]	To analyze acoustical signal for monitoring normal operation of gearbox	The complex-valued fixed point algorithm was used for frequency domain signal.	Power transfer system
Interoperability testing	Vijayaraghavan et al. [49]	To provide a common means for communication between devices	A data exchange standard was proposed.	Manufacturing system
	Shin et al. [38]	To analyze the operating situations of the systems at the system-integration phase	A message-description language was used to convert the raw interface data into the interpreted data format.	Ship system

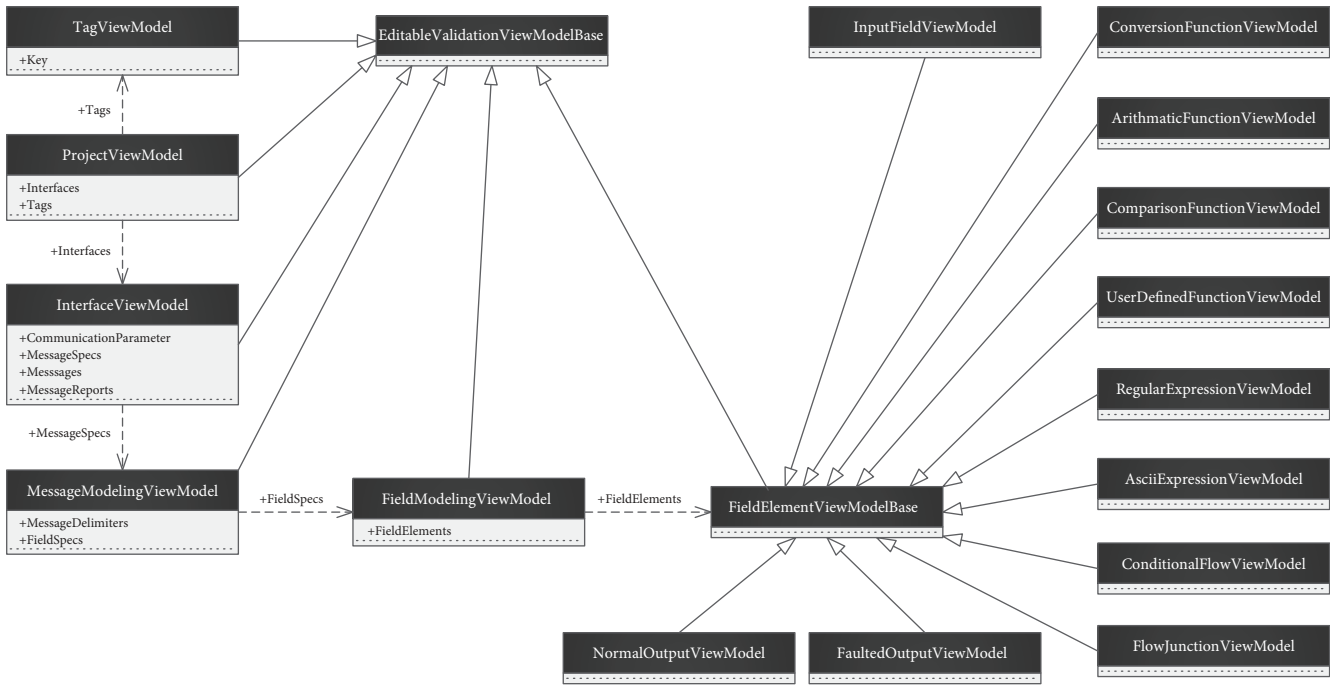


Figure 3: Simplified class diagram of developed software for fault detection and diagnosis.

out by checking the consistency between the structural and logical patterns and the measured signals. In short, we focused on the interface signals and developed explicit data models with deterministic criteria [46, 47].

Most of all, the above studies have been utilized in operating the complex systems. For system design and integration phases, the interoperability testing is a similar concept to our approach [37, 48]. For example, Vijayaraghavan et al. [49] proposed an open communication standard for data interoperability. The proposed open protocol provides the mechanism for process and system monitoring and optimization concerning resources. Shin et al. [38], which is similar to that of the present study, involved the development of an analysis tool to confirm the integrated performance of the complex system. To analyze the data, a message-description language was used to convert the raw interface data into the interpreted data format. Despite their practical contributions, however, they cannot diagnose the faults within the interfaces. To the best our knowledge, no work has been reported toward focusing on fault detection and diagnosis during the system design and integration phases.

4. Proposed Work

4.1. Software Architecture. Having introduced the concept of interface data and their faults, the overall design of the developed software will be introduced in this subsection. The focus of the software is to (1) represent hierarchies of the interface data (i.e., project, interface, message, and field) and (2) realize structure and logic modeling with graphical user interface (GUI).

To provide flexible GUI for modelers, a specific software design pattern was used. The MVVM design pattern in WPF technology facilitates to decouple the GUI from model logic and data [50, 51]. The model in the MVVM pattern is an implementation of the application's domain model that includes a data model along with business logic, and the view is responsible for defining the layout and appearance of what the user sees on the screen. The view model acts as an intermediary between the view and the model and is responsible for handling the view logic. Because the view model retrieves data from the model and then makes the data available to the view, in this subsection, we will focus on view models to realize the proposed modeling.

A class diagram for major view models of the developed software is described in Figure 3. Two *ViewModelBase* classes serve as base classes for other view model classes. The left part of Figure 3 indicates hierarchies for interface data modeling from a project to a field, and the right part shows specific elements to model a field logically.

The *ProjectViewModel* class takes charge of resolving interface faults for a particular project. After a project is determined, *TagViewModel* manages multiple tests in the project. In the following application section, nine tests for the submarine renovation project were managed by this class. The *InterfaceViewModel* ensures operations regarding evaluating an individual interface (e.g., loading interface data, analyzing them with interface protocols, and visualizing fault results). Therefore, it is mainly composed of the following properties: *Messages* for the interface data, *MessageSpecs* for the interface protocols, and *MessageReports* for the fault reports. Finally, the *MessageModelingViewModel* facilitates structure modeling at the message level, and the *FiledModelingViewModel* enables logic modeling at the field level. On this wise, the architecture fundamentally facilitates a hierarchical modeling: an interface provides the means for an arbitrary number of messages and a message also comprises multiple fields.

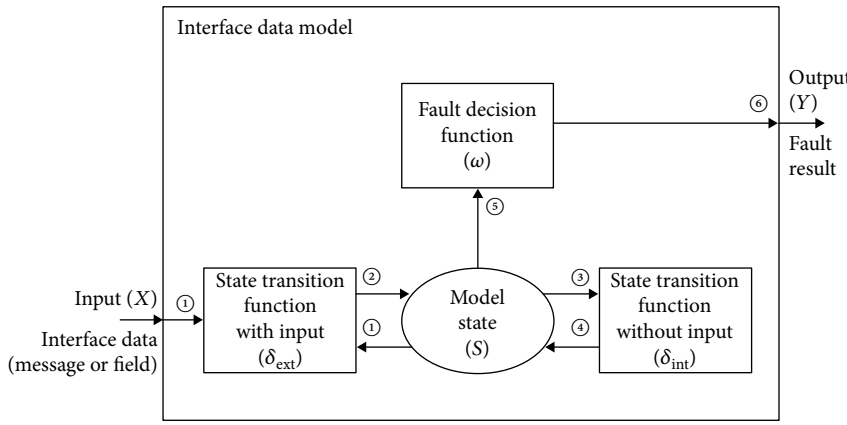


Figure 4: Elements of proposed interface data model.

In particular, the logic model at the field level recognizes logical rules and diagnoses faults by comparing received field data to the rules. The logical rules generally contain multiple steps to decode the raw data to interpretable information. Therefore, to model a field in stages, the developed software provides eight elements for the rules, which are illustrated in the right parts of Figure 3 (from *ConversionFunctionViewModel* to *FlowJunctionViewModel*). The remaining view models (i.e., *InputFieldViewModel*, *NormalOutputViewModel*, and *FaultOutputViewModel*) are for inputs and outputs of the logic model. Because these elements have their own views in the MVVM pattern, the developed software offers a graphical modeling approach that helps the modelers visualize every element to model a field. In the following subsections, we explain methodological aspects for structure and logic modeling at the message and the field levels, respectively.

4.2. Structure Modeling at Message Level. Figure 4 illustrates elements of the proposed model that is either a structure or a field model. The model fundamentally receives interface data (X in Figure 4) as input and sends fault results (Y) as output. Inside the model are one total state and three functions. These are depicted with circle and squares in Figure 4. The model state (S) is updated after performing two transition functions, which contain received data, conditions for the rules, and functional results.

If an input occurs, δ_{ext} interprets the input data with the rules and updates S (① and ② in Figure 4). As explained in the previous subsection, the logical rules need multiple interpretations. In this instance, after δ_{ext}, δ_{int} updates S without any input. Note that δ_{int} is carried out sequentially until the interpretations are completed (③ and ④). This is a general situation for logic modeling at the field level, which will be explained in the following subsection. Finally, when all δ_{ext} and δ_{int} are fulfilled, ω decides fault results based on the updated state and outputs the decision (⑤ and ⑥).

The proposed models are derived from the discrete event system specification (DEVS), which is a general mathematical representation for discrete event systems [52–54]. The main difference between the DEVS and our models is targets to be modeled. The DEVS focuses on the system itself. Thus, it should represent behaviors of the system as time passes. On the contrary, because our models aim at the interface data rather than the system, they have no concept of time. In other words, our models are static in which the output depends on the input at the same time.

According to Figure 4, the proposed structure model is 5-tuple consisting the following:

$$SM_{\text{Message}} = <X, Y, S, \delta_{\text{ext}}, \omega>$$

X is an input set of n fields comprised of a message, where

$X = \{(f_i, l_i) \mid 1 \le i \le n\}$,

f_i is the value of the i-th field within the message,

l_i is the length of f_i;

Y is an output set of structural faults, where

$Y = \{(d_j, r_j) \mid 1 \le j \le k, d_j \in D, r_j \in \{true, false\}\}$,

d_j is the j-th delimiter in D,

r_j is the fault result for d_j;

$S = D \times R$ is a total state set, where

D is a set of delimiters for structural rules,

$R = \{true, false\}$ is a set of transition results;

$\delta_{\text{ext}} : 2^X \times D \to S$ is the message transition function;

$\omega : S \to Y$ is the fault decision function.

Components of $\text{SM}_{\text{Message}}$ are contained within $\langle\rangle$. Every notation in $\text{SM}_{\text{Message}}$ is based on set theory. For example, $\{\}$ means a set; $\times$ indicates the Cartesian product (i.e., all possible ordered pairs); 2^X means the power set of X; and $\to$ means the function mapping.

The structure model evaluates how many fields belong to a message and what distinguishes the message. Specifically, the external transition function, δ_{ext}, receives all the fields comprising a message and appropriate delimiters.

Table 3: Examples of X and D depending on type of message structure.

Structure type	$x \in 2^X$	$d \in D$
Header and footer	(f_1, l_1)	d_{hdr}
	(f_n, l_n)	d_{ftr}
Header and length	(f_1, l_1)	d_{hdr}
	X or (f_i, l_i)	d_{length}
Header	(f_1, l_1)	d_{hdr}
Footer	(f_n, l_n)	d_{ftr}
Length	X or (f_i, l_i)	d_{length}

f_1 and f_n mean the first and last fields, respectively. (f_i, l_i) means a specific field containing the length information of the message.

And it updates the transition result, indicating if the message has a correct format. Because one or more fields are mapped into one delimiter, δ_{ext} needs X in the form of the power set. Note that the internal state transition function is not required in the structure model. Finally, the fault decision function, ω, checks the current state and produces a fault result.

Based on the preinvestigation in Table 1, the message structures were divided into five types, which is shown in Table 3. The structure types are classified depending on how to use the following delimiters: a header, a footer, and a message length. In case of types with the header and the footer, ω checks whether the first and the last fields are satisfied with the header and the footer, respectively. The length is known in two ways: (1) it is computed by adding length of all the fields, or (2) it can be found within a specific field, for example, (f_i, l_i) in Table 3. The structure model is relatively simple to design because it decides the correctness of the message exterior. As explained in the previous subsection, the *MessageModelingViewModel* in Figure 3 realizes the structure model.

The following specifications are a modeling example for a message in the GPS that will be explained in Section 5.3:

$$SM_{GPS-1} = <X, Y, S, \delta_{\text{ext}}, \omega>$$

$$X = \{(f_1, l_1), (f_2, l_2), (f_3, l_3), (f_4, l_4), (f_5, l_5), (f_6, l_6), (f_7, l_7), (f_8, l_8), (f_9, l_9)\}$$

$$Y = \{(d_{hdr}, r_{hdr}), (d_{ftr}, r_{ftr})\};$$

$S = \{d_{hdr}, d_{ftr}\} \times \{true, false\}$, where

d_{hdr} = 0x3A58 as a hexadecimal number,

d_{ftr} = 0x0D0A as a hexadecimal number;

$$\delta_{\text{ext}} : (f_1, l_1) \times (d_{hdr}) \rightarrow \begin{cases} (d_{hdr}, true) \text{ if } f_1 = d_{hdr}, \\ (d_{hdr}, false) \text{ if } f_1 \neq d_{hdr}; \end{cases}$$

$$(f_9, l_9) \times (d_{ftr}) \rightarrow \begin{cases} (d_{ftr}, true) \text{ if } f_9 = d_{ftr}, \\ (d_{ftr}, false) \text{ if } f_9 \neq d_{ftr}; \end{cases}$$

$$\omega : \begin{cases} (d_{hdr}, true) \rightarrow (d_{hdr}, true); \\ (d_{hdr}, false) \rightarrow (d_{hdr}, false); \\ (d_{ftr}, true) \rightarrow (d_{ftr}, true); \\ (d_{ftr}, false) \rightarrow (d_{ftr}, false). \end{cases}$$

4.3. Logic Modeling at Field Level. The logic model assesses if each field is understandable for receiving systems. Because the interpretation process is complicated, the initial value of the field data needs a sequence of transition functions to decode to an understandable information. The proposed logic model is formalized as follows:

$$LM_{\text{Field}} = <X, Y, S, \delta_{\text{ext}}, \delta_{\text{int}}, \omega>$$

X is an input set of fields, where

$X = \{v_{tgt}, v_{ref}\}$,

v_{tgt} is the initial value of the target field,

v_{ref} is the initial value of the reference field;

Y is an output set of logical faults, where

$Y = \{(v, r) \mid v = v_{tgt} \in X, r \in \{true, false\}\}$,

v is the initial data of the target field,

r is the fault result of the target field;

$S = F \times P \times R$ is a total state set, where

$F = \{(f_{\text{int}}, f_{\text{dec}}) \mid f_{\text{int}} = v_{tgt} \in X\}$ is a set of values for the target field, where

f_{int} is the initial value of the target field,

f_{dec} is the decoded value of f_{int} after the previous transition function,

$P = \{p_j \mid 1 \leq j \leq n\}$ is a set of parameters for logical rules,

$R = \{(r_t, r_f) \mid r_t, r_f \in \{false, idle, true\}\}$ is a set of transition results, where

r_t is the transient result after the interim transition function,

r_f is the final result after the last transition function;

$\delta_{\text{ext}} : X \times P \times R \rightarrow F \times R$ is the field transition function with X;

$\delta_{\text{int}} : F \times P \times R \rightarrow F \times R$ is the field transition function without X;

$\omega : F \times R \rightarrow Y$ is the fault decision function.

A key element of LM_{Field} is the field transition functions, (i.e., δ_{ext} and δ_{int}); thus, we have summarized their practical

Table 4: Examples of δ classified into 5 operations.

Category	δ	Description
Conversion operation	Type conversion (δ^1)	Changing one data type into another, for example, integer to string
Arithmetic operation	Addition (δ^2)	Fundamental numerical functions, for example, $+, -, \times, \div$
	Subtraction (δ^3)	
	Multiplication (δ^4)	
	Division (δ^5)	
Comparison operation	Greater than (δ^6)	Inequality functions, for example, $>, \geq, <, \leq$
	Greater than or equal (δ^7)	
	Less than (δ^8)	
	Less than or equal (δ^9)	
	Equal (δ^{10})	Equality functions, for example, $=, \neq$
	Not equal (δ^{11})	
Special operation	User-defined equation (δ^{12})	User-defined equation in a complex form including arithmetical, logical, and bit-shift operations, for example, $(\sin x + 10) \times 30$, $(x < 90)\&(x > -90)$, [x] << 7
	Regular expression (δ^{13})	Comparison of standard textual syntax for representing patterns of text, for example, [p − s][0 − 5][0 − 9], [0 − 3][0 − 5][0 − 9]
	ASCII expression (δ^{14})	Comparison of ASCII codes for example: a specific field ≡ "STS"
Flow control operation	Conditional flow (δ^{15})	Conditional function, for example, If-else statement

δ is for either δ_{ext} or δ_{int}.

types in Table 4. In common with Table 3, the functions also were induced from Table 1.

The five categories in Table 4 show that three types of operations are basically provided to (1) convert the data type of the field to another, (2) compute it arithmetically, and (3) compare it with a criterion. To express the complicated patterns in the fields, we identified 3 special functions: user-defined equation, regular expression [55], and ASCII expression (δ_{12} to δ_{14} in Table 4). For example, we assume that a bearing field has three ASCII characters to represent a three-digit number. The field is additionally promised that the seventh bit of the third character is always assigned if the bearing is not newly updated. Otherwise, under normal states, the field is interpreted as the three-digit number from zero to 359. In this case, we simplified logic modeling by using two regular expressions: to compare the field with patterns, that is, $[0-3][0-5][0-9]$ for a normal state and $[p-s][0-5][0-9]$ for an abnormal state. Finally, these operations are parallel in using a flow control operation to make a conditional statement. Parameters for the operations are specified in P in LM_{Field}.

Let us explain how the transition functions in Table 4 are used in a real case. As explained previously, an Electronic Support Measure (ESM) sends bearing information to be encoded with three ASCII characters. The following steps show the overall procedures that the receiving system decodes the bearing information:

(1) The receiving system first identifies availability of the bearing field by checking another reference field within the same message (δ^{10}_{ext}, δ^{15}_{ext}).

(2) If the bearing field is not available, the system confirms that its hexadecimal values are all 0x20 (δ^{14}_{int}).

(3) If the bearing field is available, the system next identifies if the field is newly updated by comparing its values with predefined patterns ($\delta^{13.1}_{int}$).

(4) If the bearing field is proved to be newly updated, the system checks if the numeric value is within the valid range from zero to 360 ($\delta^{13.2}_{int}$).

(5) The system finally converses its current data type, that is, ASCII characters, into unsigned integers (δ^{1}_{int}).

These steps are performed sequentially according to the results of the previous step. In this case, for logic modeling, six transition functions were used including one conditional flow function. The overall specifications are as follows (the bold fonts in δ mean updated parts):

$$LM_{bearing} = <X, Y, S, \delta_{ext}, \delta_{int}, \omega>$$

$X = \{v_{brg}, v_{ref}\}$, where

v_{brg} is the initial value of the targeted bearing field,

v_{ref} is the initial value of the reference field for checking the availability of v_{brg};

$Y = \{(v_{brg}, true), (v_{brg}, false)\}$;

$S = \{(f_{int}, f_{dec})\} \times \{p_1, p_2, p_3, p_4\} \times \{(r_t, r_f)\}$, where

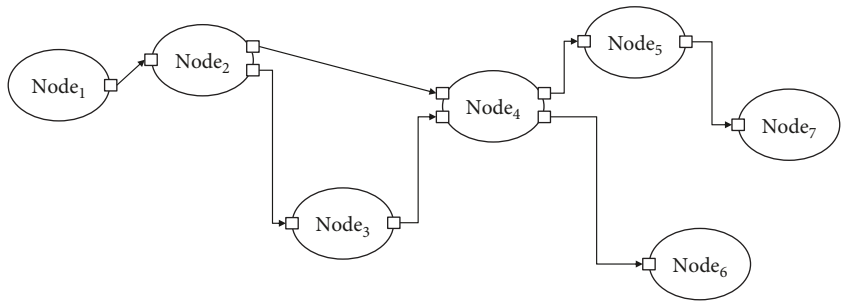


Figure 5: Sequential property for interpreting field.

$p_1 = 0 \times 20$ as a hexadecimal number,

$p_2 = [\mathrm{p} - \mathrm{s}][0-5][0-9]$ as a regular expression,

$p_3 = [0-3][0-5][0-9]$ as a regular expression,

p_4 = unsigned integer as data type;

$\delta_{\mathrm{ext}}^{10} : (v_{ref}) \times (p_1) \times (idle, idle)$

$\rightarrow (f_{\mathrm{int}}, f_{\mathrm{dec}}) \times (\boldsymbol{true}, false)$ if v_{brg} is not available,

$(f_{\mathrm{int}}, f_{\mathrm{dec}}) \times (\boldsymbol{false}, false)$ if v_{brg} is available;

$\delta_{\mathrm{ext}}^{15}$:

$(v_{tgt}) \times \varnothing \times (true, false)$

$\rightarrow (\boldsymbol{f}_{\mathbf{int}}, f_{dec}) \times (\boldsymbol{true}, false)$,

$(v_{tgt}) \times \varnothing \times (false, false)$

$\rightarrow (\boldsymbol{f}_{\mathbf{int}}, f_{\mathrm{dec}}) \times (\boldsymbol{false}, false)$;

$\delta_{\mathrm{int}}^{14}$:

$(f_{\mathrm{int}}, f_{\mathrm{dec}}) \times (p_1) \times (true, false)$

$\rightarrow (f_{\mathrm{int}}, \boldsymbol{f}_{\mathbf{dec}}) \times (\boldsymbol{true}, \boldsymbol{true})$ if v_{tgt} is correct,

$(f_{\mathrm{int}}, \boldsymbol{f}_{\mathbf{dec}}) \times (\boldsymbol{false}, \boldsymbol{false})$ if v_{tgt} is not correct;

$\delta_{\mathrm{int}}^{13.1}$:

$(f_{\mathrm{int}}, f_{dec}) \times (p_2) \times (false, false)$

$\rightarrow (f_{\mathrm{int}}, \boldsymbol{f}_{\boldsymbol{dec}}) \times (\boldsymbol{true}, \boldsymbol{true})$ if the comparison is correct,

$(f_{\mathrm{int}}, \boldsymbol{f}_{\boldsymbol{dec}}) \times (\boldsymbol{false}, \boldsymbol{false})$ if the comparison is not correct;

$\delta_{\mathrm{int}}^{13.2}$:

$(f_{\mathrm{int}}, f_{\mathrm{dec}}) \times (p_3) \times (false, false)$

$\rightarrow (f_{\mathrm{int}}, \boldsymbol{f}_{\mathbf{dec}}) \times (\boldsymbol{true}, false)$ if the comparison is correct,

$(f_{\mathrm{int}}, \boldsymbol{f}_{\mathbf{dec}}) \times (\boldsymbol{false}, \boldsymbol{true})$ if the comparison is not correct;

$\delta_{\mathrm{int}}^{1} : (f_{\mathrm{int}}, f_{dec}) \times (p_4) \times (true, false) \rightarrow (f_{\mathrm{int}}, \boldsymbol{f}_{\mathbf{dec}}) \times (\boldsymbol{true}, \boldsymbol{true})$;

$\omega : (f_{\mathrm{int}}, f_{\mathrm{dec}}) \times (true, true) \rightarrow (v_{brg}, true)$,

$(f_{\mathrm{int}}, f_{\mathrm{dec}}) \times (false, true) \rightarrow (v_{brg}, false)$.

To realize the sequential characteristics of the logic modeling as a software, a computer network concept is applied. Figure 5 shows a schematic illustration of a network configuration, which is a collection of nodes and connections. The nodes are linked to each other by connections, and the connectors in the nodes are anchor points to attach connections between the nodes. For example, Node_1 corresponds to X of $\mathrm{LM}_{\mathrm{Field}}$, Node_6 and Node_7 are relevant to Y, and the others are represented by two transition functions: δ_{ext} or δ_{int}. A major difference from the typical computer network is that the network in Figure 5 is a one-way communication and not a two-way interaction; that is, all the connections have directions to pass the data to the node at right.

Figure 6 shows a class diagram for logic modeling based on the network configuration. The *DiagramViewModel* visualizes and edits the overall modeling of a field. *Nodes* and *Connections* as properties of this class specify the collections of nodes and connections to be displayed in the logic modeling. In *NodeViewModel*, *InputConnectors*, and *OutputConnectors* are the collection of connectors that specify the node's connection anchor points, and *AttachedConnections* retrieves a collection of the connections that are attached to the node. The *Element* determines the type of the node. The *ConnectionViewModel* describes a connection between both-sided nodes, specifically two connectors in each node (i.e., the *SourceConnector* and the *DestConnector*). This connection continuously monitors its source and destination connectors. Finally, the *ConnectorViewModel* indicates an anchor point on a node for attaching a connection. The *ParentNode* in this class references the node that owns the connector.

Figure 7 shows the modeling execution of the bearing field previously described, that is, $\mathrm{LM}_{\mathrm{bearing}}$. The developed software provides two views: a list view in the form of the ribbon command bar and a model view for building the model. The list view provides block libraries of modeling elements, in particular transition functions in Table 4 (the red box in Figure 7). Using the libraries, a modeler can

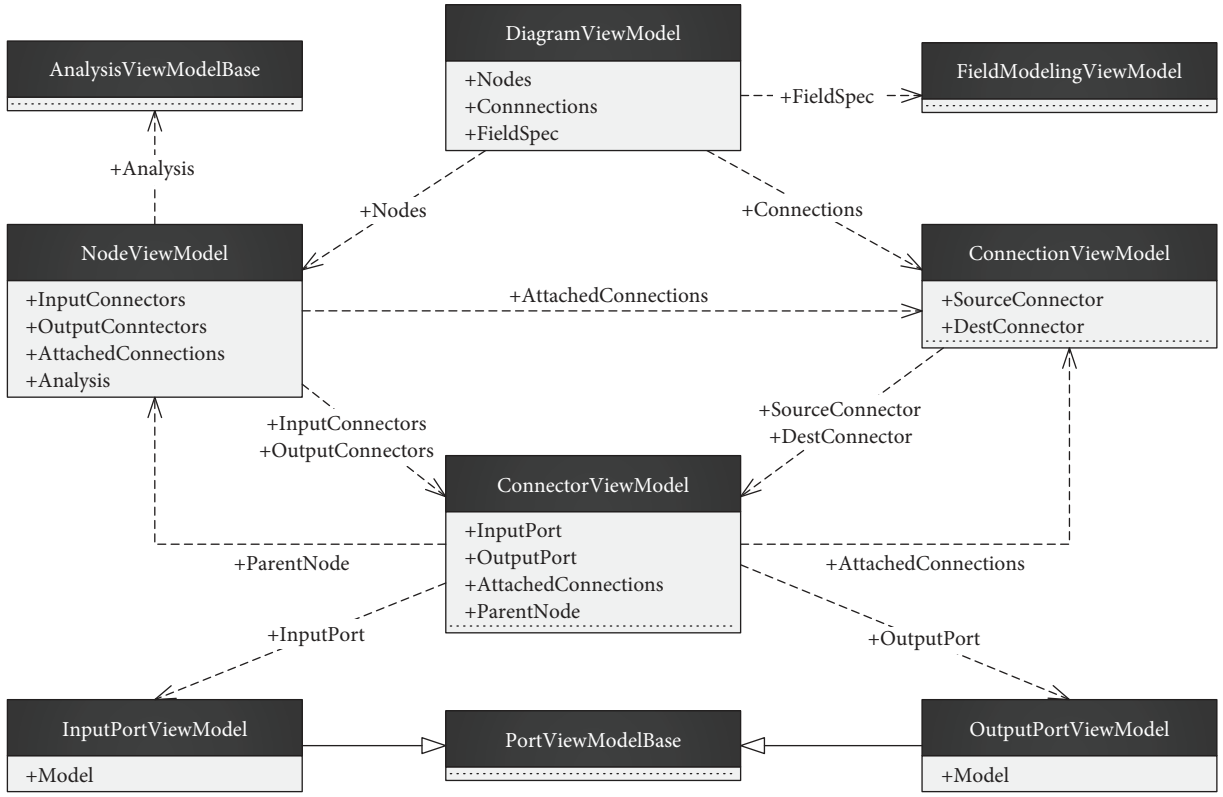


Figure 6: Class diagram for logic modeling using network configuration.

graphically build and edit the logic model in the model view. For example, he/she can choose an appropriate element from the list view based on the model design and drag it to the model view. By connecting the two-sided elements with lines, the modeler can easily build a sequence of transitions and decision functions. In Figure 7, yellow boxes are realizations of $LM_{bearing}$. In this manner, the developed software facilitates intuitive modeling via the block libraries and allows flexible modeling through the addition and deletion of the modeling elements.

5. Application

The objective of this application is to demonstrate how we can detect and diagnose interface faults when designing and integrating a complex SoS. The targeted system is an underwater vessel (i.e., a submarine system). The faults were (1) incorrect interface protocols during the design phase and (2) abnormal values in interface data during the integration phase.

5.1. Shipbuilding Project Overview. Due to budget constraints, the Navy can no longer afford to build new ships beyond its existing military force [28]. In this context, the product improvement program (PIP) is a good alternative. The PIP incorporates improvements of partial systems to enhance overall system performance. Because it reduces the procurement time and lowers the maintenance costs compared to the development of an entirely new system, PIP has become an industrial trend in several industrial fields [56–59].

Since late 2014, South Korean shipbuilder, Daewoo Shipbuilding and Marine Engineering (DSME), has undergone a PIP for three submarine systems that corresponded to the first-generation class of Table 1 [24]. The submarines' onboard subsystems including navigation, acoustic, optical, and radar sensors as well as combat systems have been renovated. The PIP will be finalized in 2019.

For this PIP, the compatibility of the improved subsystems with existing ones at the I/O level is a key consideration. Thus, two phases in the system development life cycle (SDLC) require interface fault-handling activities. To be specific, the validity of the interface protocols needs to be assured to accurately represent the interface data between the linked subsystems. Then, the developed subsystems should be verified at the I/O level by comparing the interface data with the valid protocols. In this respect, the DSME has carried out several tests to resolve interface faults using the proposed method and software at the design phase as well as the integration phase. More detailed descriptions for this PIP were informed in our previous work [12].

5.2. Design of Tests. As shown in Table 5, the proposed modeling and software have been utilized for nine shipboard tests over the last three years. Until the first half of 2017, preliminary and critical design phases had been proceeded for the first renovated submarine. During this period, eight tests were conducted. Thereafter, all the subsystems were

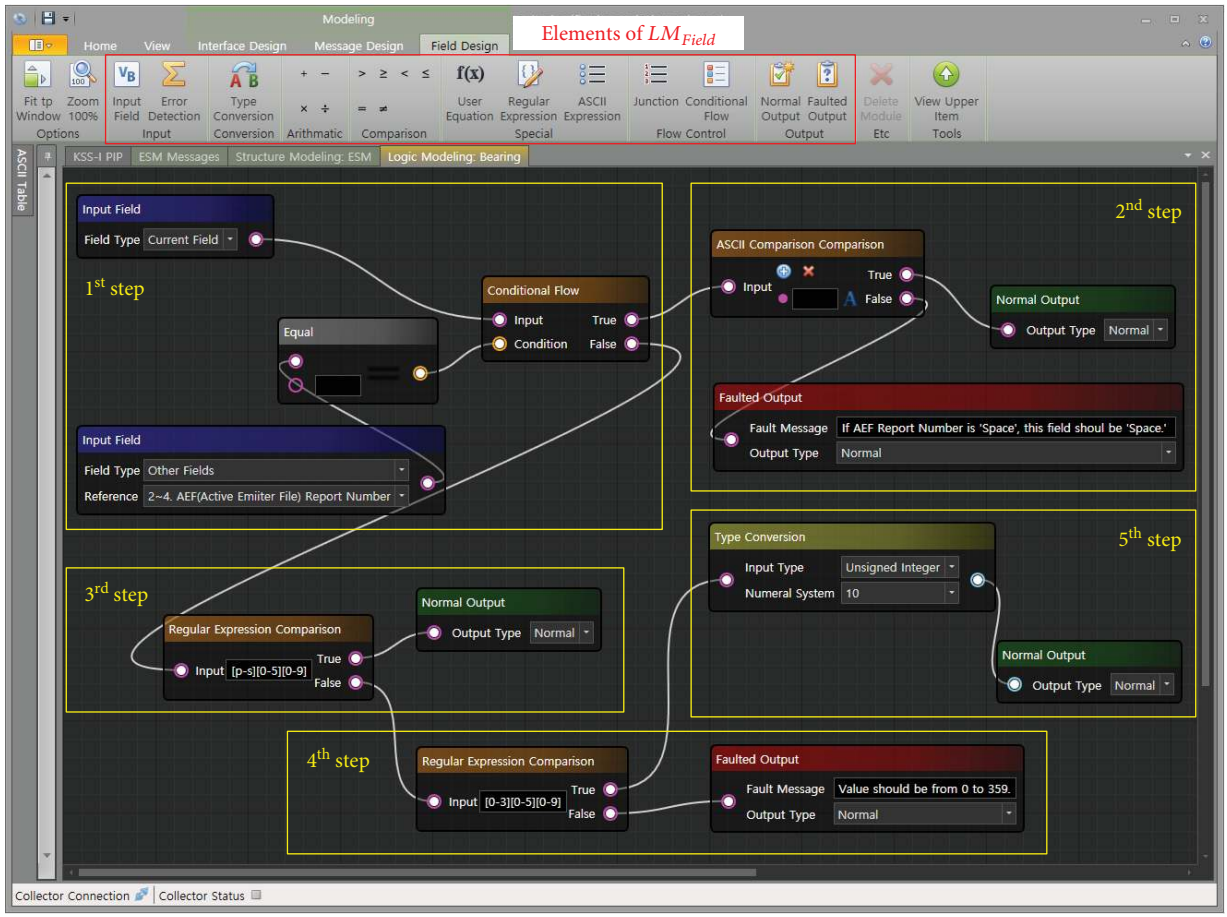
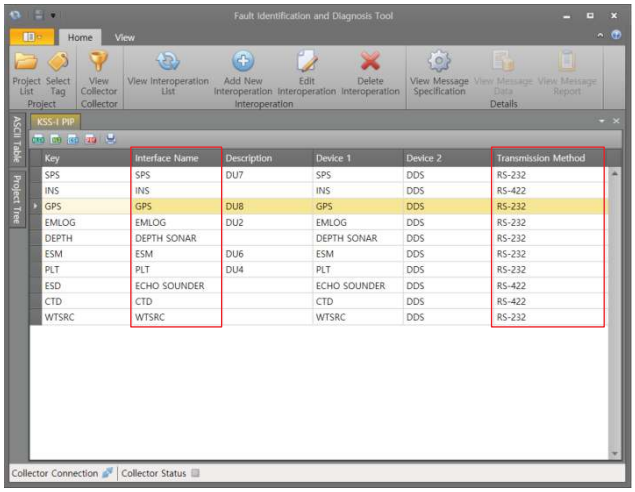


Figure 7: Logic modeling of bearing field in ESM message.

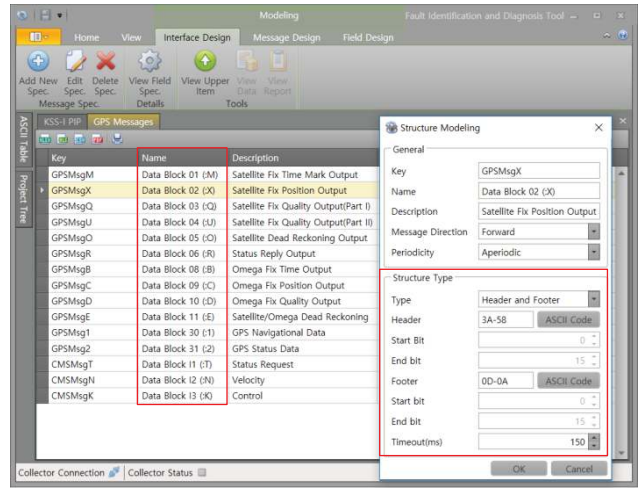
Table 5: Overall design of tests during system design and integration phases.

SDLC	Test no.	When (yy.mm)	Test site	Subsystems of local interfaces	Objective	Test period
System design phase	$Test_1$	15.03	At sea	Eight sensor and one control systems	To train for detecting a surface target and evaluate target motion analysis	72 hours
	$Test_2$	15.05	At sea	Eight sensor and one control systems		72 hours
	$Test_3$	15.09	At the harbor	One additional control system	To train command and control for underwater weapon engagement	6 hours
	$Test_4$	15.12	At sea	Eight sensor and one control systems	To measure accuracy of passive sonars to detect a surface target	72 hours
	$Test_5$	16.08	At sea	Eight sensor and one control systems	To acquire navigation data	72 hours
	$Test_6$	16.11	At sea	Eight sensor and one control systems	To measure underwater radiation noise	72 hours
	$Test_7$	16.11	At sea	Eight sensor and one control systems	To measure self-noise of sonar systems	72 hours
	$Test_8$	17.04	At sea	Eight sensor and one control systems		72 hours
System integration phase	$Test_9$	18.01	At the harbor	Four sensor systems	To evaluate systems integration	6 hours

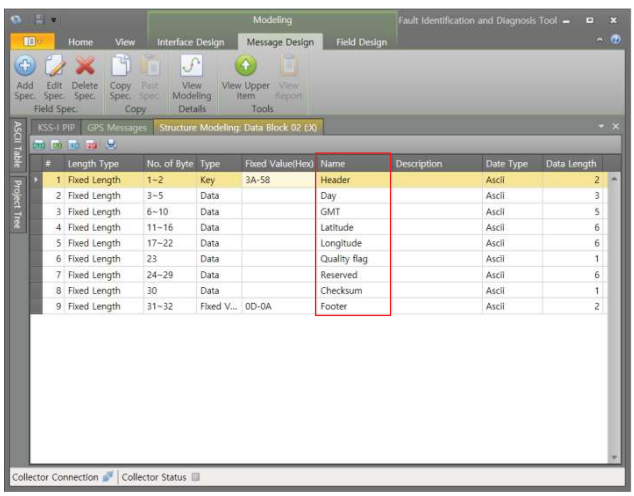
yy and mm in (yy.mm) mean year and month, respectively. Subsystems in the fourth column are connected to local proxies such as the DIS, the integrated management system, or the signal processing system. These tests are extension from our previous study [12].



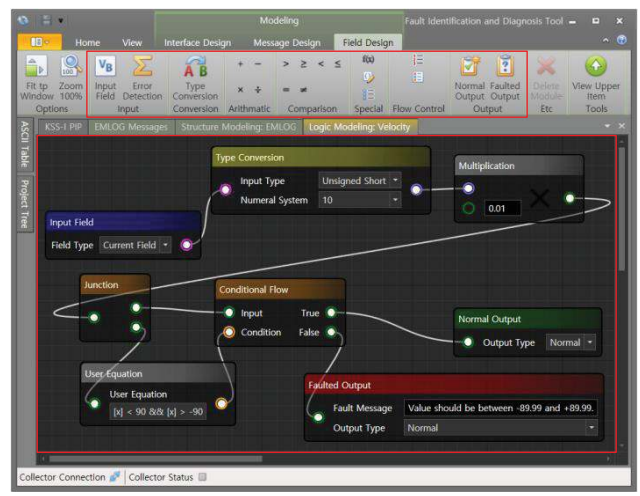
(a) Modeling of 9 groups of interfaces in shipbuilding system



(b) Modeling of 15 groups of messages transferred in GPS



(c) Modeling of 9 fields in GPS position message



(d) Modeling of velocity field in EM log message

FIGURE 8: Interface data modeling using developed software.

completely developed; thus, they have been integrated to the submarine system as of late 2017. During this time, we carried out one fault test. Most tests were conducted at sea to resolve the faults for various operational situations.

Among 10 subsystems to be tested, eight were sensor systems including acoustic and optical sensors (e.g., echo sounder, depth sonar, periscope, CTD device, EM log, and ESM) and the navigation suite of sensors (e.g., INS and GPS). Two control systems were a plotting board system and a weapon control system. The weapon control system was an additionally renovated system during the PIP. This means that test_3 in Table 5 was unexpected and belatedly determined just two months before the test.

5.3. Interface Data Modeling. For detection and diagnosis of the interface faults, we first modeled message structures and field logics in each interface. Figure 8 shows some modeling results using the developed software. Ten interfaces between the 10 subsystems and local proxies were targeted, which are based on serial communications such as RS-232 and RS-422 (Figure 8(a)). As an example of the interface between the GPS and the DIS, 15 groups of messages were transferred (Figure 8(b)). One message, Data Block 2, was modeled with nine fields including d_{hdr} and d_{ftr} (Figure 8(c)). These figures are the realization of SM_{GPS-1} described in Section 4.2.

Figure 8(d) shows logic modeling of the velocity field in the EM log message. In this field, two main interpretations are required: (1) to convert a hexadecimal number to a floating-point number and (2) to check the valid range of the value. Specifically, the field data has initially a hexadecimal number; thus, it needs to be first converted to a decimal number. Then the decimal number is multiplied by 0.01 to represent two decimal places. The value is finally checked whether it is within the valid range from −90 to +90. If the result is out of range, the field is diagnosed with a logical fault. The logic modeling for this process can be expressed in a combination of various transition functions. In Figure 8(d), four transition functions (i.e., δ^1_{ext}, δ^4_{int}, δ^{12}_{int}, and δ^{15}_{int},) were used to model the logic of the velocity field.

Table 6: Main results of interface data modeling.

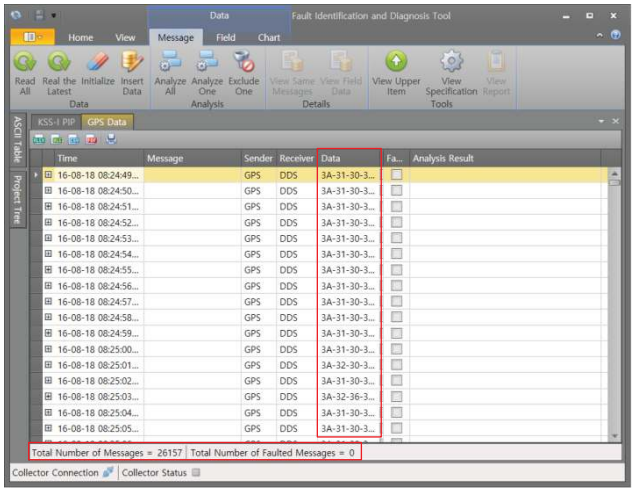
Interface type	Message type	Structure modeling (SM)					Logic modeling (LM)			
		$\sum_{i=1}^{n} l_i \in X$	$N(X)$	$d \in D$			Field division unit	$\frac{\sum_{i=1}^{N(\mathrm{SM}.X)}[N(\delta_{\mathrm{ext}}) + N(\delta_{\mathrm{int}})]}{N(\mathrm{SM}.X)}$	Decoded value	
				d_{hdr}	d_{ftr}	d_{length}			Number	Character
$\mathrm{Sensor}_{\mathrm{A}}$	$\mathrm{Msg}_{\mathrm{A\text{-}1}}$	14	6	O	O		Byte	4.33	O	O
	$\mathrm{Msg}_{\mathrm{A\text{-}2}}$	9	4	O	O		Byte	2.50	O	
$\mathrm{Sensor}_{\mathrm{B}}$	$\mathrm{Msg}_{\mathrm{B\text{-}1}}$	28	20	O	O		Bit	3.00	O	O
	$\mathrm{Msg}_{\mathrm{B\text{-}2}}$	30	27			O	Bit	3.44	O	O
$\mathrm{Sensor}_{\mathrm{C}}$	$\mathrm{Msg}_{\mathrm{C\text{-}1}}$	4	2	O	O		Byte	2.00	O	
	$\mathrm{Msg}_{\mathrm{C\text{-}2}}$	32	9	O	O		Byte	2.56	O	
	$\mathrm{Msg}_{\mathrm{C\text{-}3}}$	44	16	O	O		Byte	2.31	O	
	$\mathrm{Msg}_{\mathrm{C\text{-}4}}$	50	4	O	O		Byte	2.78		O
	$\mathrm{Msg}_{\mathrm{C\text{-}5}}$	54	18	O	O		Byte	2.56	O	O
	$\mathrm{Msg}_{\mathrm{C\text{-}6}}$	6	3	O	O		Byte	3.00		O
	$\mathrm{Msg}_{\mathrm{C\text{-}7}}$	4	2	O	O		Byte	3.30	O	
	$\mathrm{Msg}_{\mathrm{C\text{-}8}}$	32	9	O	O		Byte	4.63	O	O
	$\mathrm{Msg}_{\mathrm{C\text{-}9}}$	44	10	O	O		Byte	2.78	O	O
	$\mathrm{Msg}_{\mathrm{C\text{-}10}}$	54	16	O	O		Byte	2.75	O	O
	$\mathrm{Msg}_{\mathrm{C\text{-}11}}$	62	15	O	O		Byte	3.32	O	
	$\mathrm{Msg}_{\mathrm{C\text{-}12}}$	52	17	O	O		Byte	2.50	O	O
	$\mathrm{Msg}_{\mathrm{C\text{-}13}}$	6	3	O	O		Byte	2.50		O
	$\mathrm{Msg}_{\mathrm{C\text{-}14}}$	24	7	O	O		Byte	3.33	O	
	$\mathrm{Msg}_{\mathrm{C\text{-}15}}$	6	4	O	O		Byte	3.75		O
$\mathrm{Sensor}_{\mathrm{D}}$	$\mathrm{Msg}_{\mathrm{D\text{-}1}}$	17	7	O	O		Byte	5.29	O	O
$\mathrm{Sensor}_{\mathrm{E}}$	$\mathrm{Msg}_{\mathrm{E\text{-}1}}$	Variable length	4	O	O		Byte	3.00	O	O
	$\mathrm{Msg}_{\mathrm{E\text{-}2}}$	9	5	O	O		Byte	3.00	O	O
$\mathrm{Sensor}_{\mathrm{F}}$	$\mathrm{Msg}_{\mathrm{F\text{-}1}}$	236	67	O	O		Byte	3.75	O	O
$\mathrm{Control}_{\mathrm{G}}$	$\mathrm{Msg}_{\mathrm{G\text{-}1}}$	7	4	O	O		Byte	5.12		O
	$\mathrm{Msg}_{\mathrm{G\text{-}2}}$	284	85	O	O		Byte	3.33	O	O
$\mathrm{Sensor}_{\mathrm{H}}$	$\mathrm{Msg}_{\mathrm{H\text{-}1}}$	26	16	O	O		Byte	7.25	O	O
	$\mathrm{Msg}_{\mathrm{H\text{-}2}}$	75	17	O	O		Byte	3.25	O	O
$\mathrm{Sensor}_{\mathrm{I}}$	$\mathrm{Msg}_{\mathrm{I\text{-}1}}$	3	7	O	O		Bit	2.25	O	
	$\mathrm{Msg}_{\mathrm{I\text{-}2}}$	3	7	O	O		Bit	2.25	O	
	$\mathrm{Msg}_{\mathrm{I\text{-}3}}$	3	7	O	O		Bit	2.25	O	
	$\mathrm{Msg}_{\mathrm{I\text{-}4}}$	3	7	O	O		Bit	2.25	O	
$\mathrm{Control}_{\mathrm{J}}$	$\mathrm{Msg}_{\mathrm{J\text{-}1}}$	18	90	O	O		Bit	3.75	O	O

Interface names consist of subsystems connected to local proxies and their identifiers: subsystem identifier. "O" in columns for the delimiter and interpreted value means to be applicable as a positive answer.

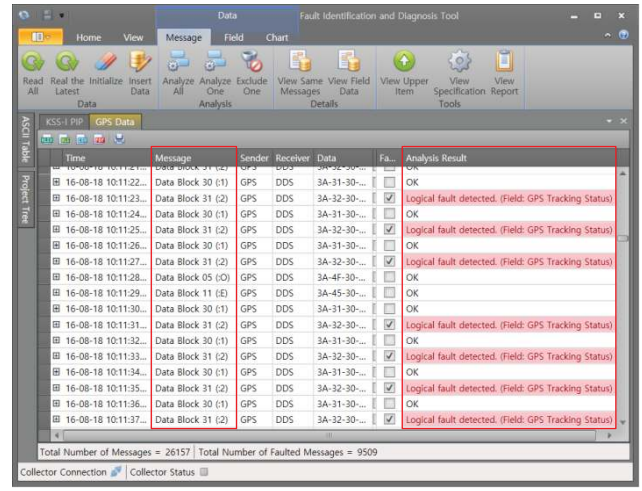
Table 6 summarizes key results of overall structure and logical modeling. Thirty-two messages in the interfaces were modeled whose lengths are 3 to 284 bytes. The number of fields in each message, that is, $N(X \in SM)$, increases if the message length is longer or if the fields are separated by bit units. For example, $\mathrm{Msg}_{\mathrm{G\text{-}2}}$, the longest message, has 85 fields in 284 bytes. Whereas, $\mathrm{Msg}_{\mathrm{J\text{-}1}}$ has 90 fields in only 18 bytes because it is divided by bit units as a typical example of customized communication protocols.

Now, let us examine $\mathrm{Msg}_{\mathrm{B\text{-}2}}$ and $\mathrm{Msg}_{\mathrm{E\text{-}1}}$ to explain specific types of message delimiters. First, since $\mathrm{Msg}_{\mathrm{B\text{-}2}}$ has a fixed-length without any header and footer, it should be classified with a message length (i.e., d_{length}). On the other hand, the length of $\mathrm{Msg}_{\mathrm{E\text{-}1}}$ is variable because its depth field has a floating-point number with 3 to 6 bytes. The variable length was not actually recognized before test_5, which will be explained in the following subsection. After test_5, $\mathrm{Msg}_{\mathrm{E\text{-}1}}$ was accepted that it cannot be classified with the message length; instead, it should be classified into the header and the footer. Except for these cases, all the messages are generally modeled with d_{hdr} and d_{ftr}. To sum up, the messages in this study used two structural types: (1) d_{hdr} and d_{ftr} and (2) d_{length}.

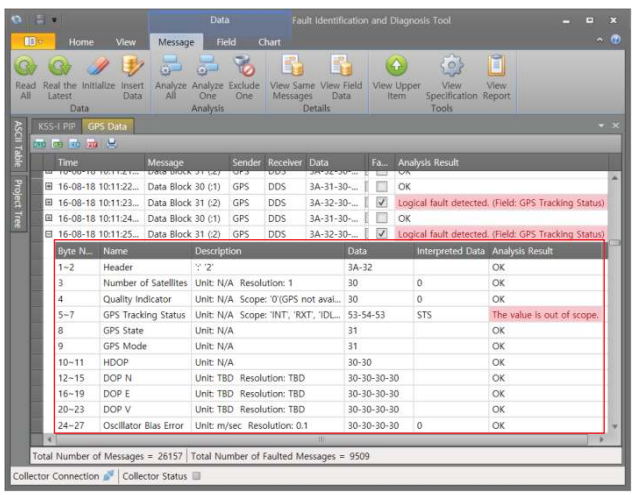
For logic modeling, the ninth column of Table 6 shows the average number of transition functions (i.e., δ_{ext} and δ_{int}) to be used for modeling fields in each message. For example, 5.29 in $\mathrm{Msg}_{\mathrm{D\text{-}1}}$ means that more than five transition



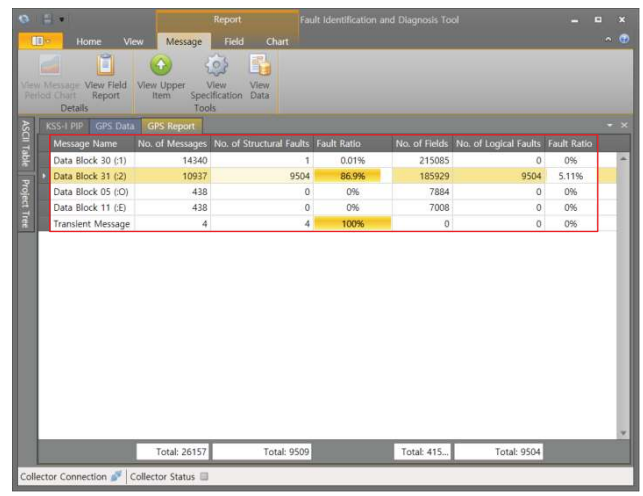
(a) Read of interface signals: GPS messages



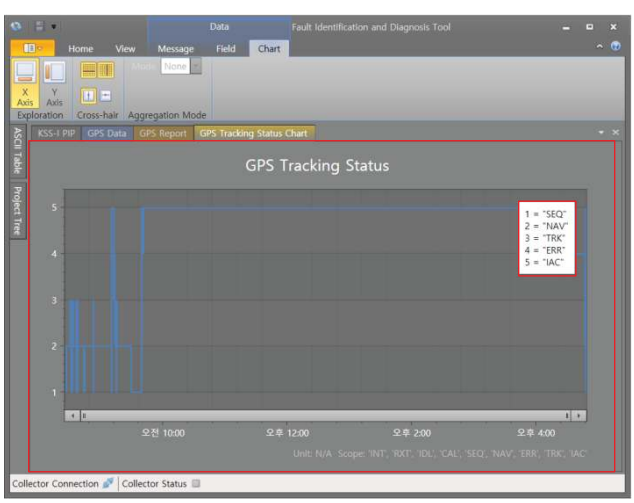
(b) Signal identification and diagnosis at message level



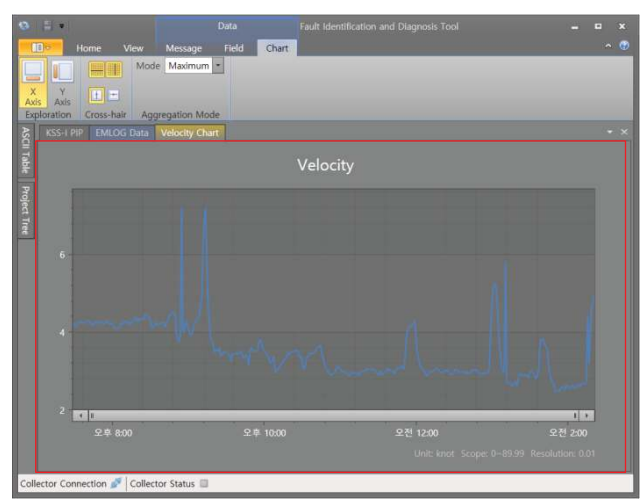
(c) Signal identification and diagnosis at field level



(d) Report of faults identification in message and field levels



(e) Analysis of time series data: Tracking status field



(f) Analysis of time series data: Velocity field

Figure 9: Results of fault detection and diagnosis using developed software.

functions were used to model each field. In this column, most messages have numbers larger than 2, which means that a group of transition functions was used to interpret the field and diagnose faults. Finally, the decoded values could be either numbers such as velocity, yaw, pitch, and depth or characters (e.g., textual message or behavioral mode).

Table 7: Overall test results: fault-detection results.

Test no.	Evaluation index	Interface									
		$Sensor_A$	$Sensor_B$	$Sensor_C$	$Sensor_D$	$Sensor_E$	$Sensor_F$	$Control_G$	$Sensor_H$	$Sensor_I$	$Control_J$
$Test_1$	NI_{total}	677,229	1,744,894	288,906	314,987	21,403	2555	39,529	618,028	4,105,489	N/A
	NI_{fault}	0	0	0	0	0	1327	14,366	0	0	N/A
$Test_2$	NI_{total}	381,452	741,652	66,487	134,920	102,586	14,013	16,862	162,438	1,537,041	N/A
	NI_{fault}	0	0	0	0	76,828	0	1168	0	0	N/A
$Test_3$	NI_{total}	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	126,533
	NI_{fault}	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0
$Test_4$	NI_{total}	1,294,454	1,524,532	188,250	277,205	2,691,498	52,296	34,650	554,447	2,704,102	N/A
	NI_{fault}	0	0	0	0	0	0	8213	0	0	N/A
$Test_5$	NI_{total}	1,021,158	1,238,562	26,157	225,154	2,132,133	12,911	28,144	450,184	1,843,669	N/A
	NI_{fault}	0	0	9509	0	0	0	0	0	00	N/A
$Test_6$	NI_{total}	578,971	2,070,717	138,547	376,578	128,331	3830	47,072	489,510	4,752,324	N/A
	NI_{fault}	0	0	30,193	0	0	0	0	0	0	N/A
$Test_7$	NI_{total}	197,753	629,961	56,431	112,904	85,332	68	14,003	214,499	953,452	N/A
	NI_{fault}	0	0	0	0	0	0	0	0	0	N/A
$Test_8$	NI_{total}	1,151,185	2,076,729	188,025	372,984	3,664,990	991	39,169	750,569	3,610,105	N/A
	NI_{fault}	0	0	0	0	0	0	0	0	0	N/A
$Test_9$	NI_{total}	N/A	11,004	N/A	N/A	10,582	N/A	N/A	3496	8877	N/A
	NI_{fault}	N/A	2331	N/A	N/A	0	N/A	N/A	3496	0	N/A

NI_{total} is the number of obtained messages from each interface. NI_{fault} is the number of faulted messages from each interface. Some interface is not applicable for a specific test, which is represented by N/A.

5.4. Test Results. Figure 9 shows some results of $test_5$ and $test_6$ using the developed software. All the figures except Figure 9(e) are relevant to the GPS messages.

Figure 9(a) shows the messages transmitted between the GPS and the DIS. During the 72 hours, 26,157 messages were monitored, which are chronologically arranged in the main table. Because the messages had not been evaluated yet, two columns—Message and Analysis Result—are empty, and the number of faulted messages at the bottom of the table is also zero. By pushing the Analyze All icon in the ribbon bar, the messages were analyzed. In the main table of Figure 9(b), during 15 seconds, two message types (Data Block 05 and Data Block 11) were identified just once, and two types (Data Block 30 and Data Block 31) were distinguished continuously. After the analysis, all the messages regarding Data Block 31 were diagnosed with logical faults, of whose rows in Analysis Result are shaded in red. Of the 26,157 messages, 9509 messages were faulted, which are indicated at the bottom of the table.

To examine where and why the faults occurred in each message, the message can be opened out so that every field is displayed. In Figure 9(c), the opened message has a logical fault at the fourth field due to the unexpected value. To be specific, the fourth field was modeled not to send STS. However, during $test_5$, the relevant system actually sent that value, which leads to a contradiction between the modeling and the real data. Finally, the overall results were shown by pushing the View Report icon. In Figure 9(d), more than 80% of messages in Data Block 32 were faulted during $test_5$.

The software also provides a time series chart for an interpreted value of each field. Figure 9(e) shows a numeric chart for the velocity value in the EM log message, and Figure 9(f) represents a chart for tracking status in the GPS message. These charts facilitate the trend of the values according to the progressed time at a glance. For example, $test_5$ was for acquiring navigation data at various velocities, and the chart exactly visualizes when the velocity is changed. The INS needs the GPS data to calibrate the navigation data, and the GPS status can be found on the chart regularly.

Table 7 summarizes (1) how many messages were acquired from interfaces of all the tests and (2) how many faults were detected among them. The numbers of the obtained messages from each interface (i.e., N_{total} in Table 7) are all different for the following two main reasons. First, each interface has different message types as well as the types have different transmission cycles. For example, $Control_G$ has two message types (i.e., $Msg_{G\text{-}1}$ and $Msg_{G\text{-}2}$) and they are transferred every eight seconds. On the other hand, $Sensor_I$ has four message types with 0.125-second cycles. In this case, NI_{total} of $Sensor_I$ is arithmetically 128 times more than that of $Control_G$ if they operate in the same amount of time $(128 = (4/2) \times (8/0.125))$. Next, because scenarios of the tests are all distinguished, the subsystems are operated situationally. The periscope system for $Sensor_A$ is normally operated during the vessel moves above the specific depth (i.e., periscope depth). This means that any messages in $Sensor_A$ will not be transferred when the vessel dives below the depth. In this context, we can assume that $test_2$ was carried out below the periscope depth over a longer period than $test_1$.

Table 8: Overall test results: fault-diagnosis results.

Test number	Fault case	Interface type	Message type	Interface data	Fault diagnosis	Fault category
$Test_1$	$Fault_1$	$Sensor_F$	Msg_{F-1}	02-30-31-30-30-39-33-37-…-35-31-*20-20-20-…-20-20-20-30*-30-33-35-…-03	The modeling should be revised that all the fields regarding a specific section are full of "0x20" if no targets are detected in the section.	Logical fault
	$Fault_2$	$Control_G$	Msg_{G-2}	0D-0A-48-32-35-31-31-…-20-*2B*-30-38-35-33-…-53-2B-…	The modeling should be revised that the interpreted value of the sign field can be "+" although the precondition field is unavailable.	Logical fault
	$Fault_3$	$Control_G$	Msg_{G-2}	0D-0A-48-32-35-31-31-…-20-2B-*30-38-35-33*-…-53-2B-…	The modeling should be revised that the speed field has meaningful information although the precondition field is unavailable.	Logical fault
	$Fault_4$	$Control_G$	Msg_{G-2}	0D-0A-48-32-35-31-31-…-20-2B-30-38-35-33-…-53-2B-*4F*-53-2B-…	The modeling should be revised that the interpreted value of the headline sonar field is "O" instead of "0" if sonar systems are available.	Logical fault
	$Fault_5$	$Control_G$	Msg_{G-2}	0D-0A-48-32-35-31-31-…-*00-00-00-00*-20-00-00-00-…	The modeling should be revised that every byte of the range field can be 0x00 as well as [0x30, 0x39].	Logical fault
$Test_2$	$Fault_6$	$Sensor_E$	Msg_{E-1}	*2A*-30-30-30-31-32-30-31-36-0D-0A	The delimiters should be changed from header and length to header and footer for variable lengths.	Structural fault
	$Fault_7$	$Sensor_E$	Msg_{E-1}	2A-30-30-30-31-*2D-36-34-34-35-33-39*-0D-0A	The pressure field should be revised to have 4 to 7 bytes including sign characters optionally.	Logical fault
	$Fault_8$	$Control_G$	Msg_{G-2}	0D-0A-48-32-35-31-31-…-*00-00-00-*…-30-30-35-40	The modeling should be revised that the target field is full of "0x00" if the relevant target is not identified.	Logical fault
$Test_4$	$Fault_9$	$Control_G$	Msg_{G-1}	*05-0A*-41-43-4B-20-40	The modeling should be revised that multiple headers, that is, "0D-0A" and "05-0A" are allowed.	Structural fault
$Test_5$	$Fault_{10}$	$Sensor_C$	Msg_{C-12}	3A-32-30-30-*53-43-53*-31-31-30-…30-30-53-0D-0A	The modeling should be revised that the interpreted value of the status field contains "STS."	Logical fault
$Test_6$	$Fault_{11}$	$Sensor_C$	Msg_{C-12}	3A-32-30-30-*49-41-43*-31-31-30-…30-30-53-0D-0A	The modeling should be revised that the interpreted value of the status field contains "IAC."	Logical fault
	$Fault_{12}$	$Sensor_C$	Msg_{C-12}	3A-32-30-30-*54-52-4B*-31-31-30-…30-30-53-0D-0A	The modeling should be revised that the interpreted value of the status field contains "TRK."	Logical fault
$Test_9$	$Fault_{13}$	$Sensor_B$	Msg_{B-1}	03-01-01-00-00-00-00-…-1B-C3-79-58-84-8A-00-00-*00*-3F	$Sensor_B$ needs to be refined to send "0x00" or "0x54" for the test field although the corresponding system is initialized.	Logical fault
	$Fault_{14}$	$Sensor_H$	Msg_{H-1}	*0D-0A*-20-20-20-…-38-43-35-03	$Sensor_H$ should be refined to send the message with an accurate header.	Structural fault

Data in italics in interface data mean parts for fault diagnosis. Square brackets in fault diagnosis are used for regression expression. These results are extended from our previous study [12].

In Table 7, the interfaces where local faults were detected are marked italics. The number of the local faults (i.e., NI_{fault}) was counted if $r_j \in SM.Y$ or $r \in LM.Y$ has *False* once in an individual message. For example, in $Sensor_C$ of $test_5$, 9509 messages were detected to be structurally or logically faulty among 26,157 messages (this is the case of Figure 9(d)). Synthetically, six interfaces except $Sensor_A$, $Sensor_D$, $Sensor_I$, and $Control_J$ were faulted. Note that $Sensor_C$ and $Control_G$ had local faults in more than two tests. This implies that the causes of the faults are distinct according to the test, which will be explained in Table 8.

To evaluate the relative magnitude of the detected faults in each interface, Figure 10 illustrates fault ratios. In $Sensor_E$, $Sensor_F$, and $Control_J$, more than half of the messages were faulted. In $Control_J$ of $test_9$, the 100-percent ratio means all the messages in this interface failed to be interpretable. Although $Sensor_C$ in $test_6$ has more faulted messages than the case of $Sensor_F$ in $test_1$, the fault ratio of $Sensor_F$ is twice higher than that of $Sensor_C$.

Table 8 summarizes diagnostic results of the faults in the overall tests. In total, fourteen fault cases were diagnosed within seven message types: three cases are for structural faults and 11 are relevant to logical faults. The structural faults, which $r_j \in SM.Y$ is *False*, came from incorrect headers and length. The logical faults have three diagnoses: (1) wrong field interpretations, (2) missed status information, and (3) incorrect relations between neighboring fields. Specifically, Msg_{E-1} had structural and logical faults simultaneously.

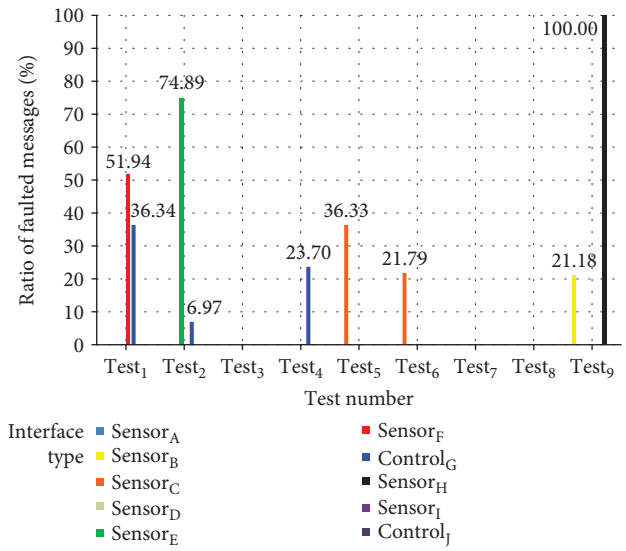


Figure 10: Fault ratio at interface level.

Before $test_2$, $Msg_{E\text{-}1}$ was modeled with a header and length for delimiters. However, real messages could not be classified with the current delimiters due to their variable lengths; thus, the modeling was revised to use a header and footer. Then, we looked over logic modeling, focusing on which field influenced the variability. It was proved that the pressure field could be represented with 4 to 7 bytes, including sign characters.

Figure 11 shows how the fault cases are influenced on the messages with the same type. Because NM_{total} in Figure 11 is regarding the same message type, it is a subset of NI_{total} in Table 7. From Figure 11, we summarized the following findings. First, eight fault cases cause more than half of the faulted messages. For example, $fault_5$ brought about more than 14,000 faulted messages during 72 hours. Next, $fault_2$ and $fault_3$ cause the same number of faulted messages, which means that they were complementary and occurred at the same time. Additionally, $fault_7$ and $fault_{12}$ were relatively difficult to detect and diagnose because they were scenario-dependent faults. If the scenarios are different, the results will be different. This means that test scenarios to cover all the cases are also important. Finally, the number of faulted messages increased as time passed since one fault case occurred.

5.5. Discussion. For synthesized analysis, Figure 12 summarizes how many fault cases were diagnosed and resolved in each test. Note that the numbers in this graph are not total numbers of faulted messages.

Invalid interface protocols led to unforeseen incompatibilities between subsystems that could not be revealed until they were integrated. The first eight tests were carried out to validate the interface protocols at the system design phase. During the tests, the structure and the logic models had been gradually revised by fixing the current faults for the next test. For example, $Msg_{G\text{-}2}$ in Table 7, whose field has "0" as an interpretation value before $test_5$, modified the interpreted value after the test. Consequently, the number of fault cases decreased as the tests progressed. Because the seventh and eighth tests had no interface faults, the interface protocols were almost fully assured to be validated.

Let us examine $test_3$ as a special case. As explained in Section 5.2, it was not considered initially in our tests. Nevertheless, the interface data for the weapon control system could be evaluated because we analyzed various types of interface data in Table 1, generalized their properties, and formalized them with a mathematical form. Fortunately, no faults were detected in this interface. Indeed, formal representations of interface data and flexible modeling using the software are particularly beneficial in arbitrary system developments.

Finally, the faults in $test_9$ made the corresponding subsystems resolve their unexpected behaviors at the system integration phase. For example, $Sensor_B$ needed transient time for initialization, and during this period, it should have been revised to send an appropriate value in the test field. As the system development progressed, integration problems became harder and more expensive to solve, so it was paramount to figure out potential faults as early as possible [21]. In this application, only two faults were found in the ninth test, which means that the previous eight tests significantly reduced the integration problems.

To sum up, the faults were identified until $test_8$ induced a revision of the structural and logical modeling. In other words, the DSME as a shipbuilding integrator continuously revised and validated the communication protocols based on the results of the eight tests. After that, the DSME verified the developed systems via resolving the faults in $test_9$. These fault-resolving activities had been conducted during the system design and integration phases, which is a clear difference between the previous fault-resolving studies. The proposed work played a vital role in the overall submarine renovation project.

6. Conclusion

In this study, we are mainly concerned with intersystem faults whose results are observable outside the systems. Our goal was to find patterns in the interface data that do not conform to expected behaviors.

The main contribution of this study is theoretical and practical. From the theoretical viewpoint, we categorized the interface faults into structural and logical levels, and they were evaluated based on mathematical modeling formalism. The core concept in the formalism is to support explicit functions for transitions and fault decisions. Thus, the proposed formalism could be applicable to customized protocols as well as standardized ones, which is suitable for arbitrary system development. From the practical perspective, the developed software facilitates graphical modeling via creation, arrangement, and revision of the modeling elements. The system integrator could constantly evaluate and supplement the interface protocols at the design phase and the interacted subsystems at the integration phase. It has been successfully utilized for a submarine renovation project.

Fault type	Massage type	NM_{total}	NM_{fault}	Fault ratio at message level (%) ($NM_{fault}/NM_{total} \times 100$)
$Fault_1$	$Msg_{F\text{-}1}$	2555	1327	51.94%
$Fault_2$	$Msg_{G\text{-}2}$	19,764	14,366	72.69%
$Fault_3$	$Msg_{G\text{-}2}$	19,764	14,366	72.69%
$Fault_4$	$Msg_{G\text{-}2}$	19,764	5662	28.65%
$Fault_5$	$Msg_{G\text{-}2}$	19,764	14,366	72.69%
$Fault_6$	$Msg_{E\text{-}1}$	102,586	76,528	74.60%
$Fault_7$	$Msg_{E\text{-}1}$	102,586	1203	1.17%
$Fault_8$	$Msg_{G\text{-}2}$	8431	1168	13.85%
$Fault_9$	$Msg_{G\text{-}1}$	17,330	8213	47.39%
$Fault_{10}$	$Msg_{C\text{-}12}$	10,937	9504	86.90%
$Fault_{11}$	$Msg_{C\text{-}12}$	35,402	30,060	84.91%
$Fault_{12}$	$Msg_{C\text{-}12}$	35,402	133	0.38%
$Fault_{13}$	$Msg_{B\text{-}1}$	10,812	2331	21.56%
$Fault_{14}$	$Msg_{H\text{-}1}$	3496	3496	100.00%

NM_{total} is the number of obtained messages with the same message type from each interface.
NM_{fault} is the number of faulted messages with the same message type from each interface.

Figure 11: Number of faulted messages and fault ratios at message level.

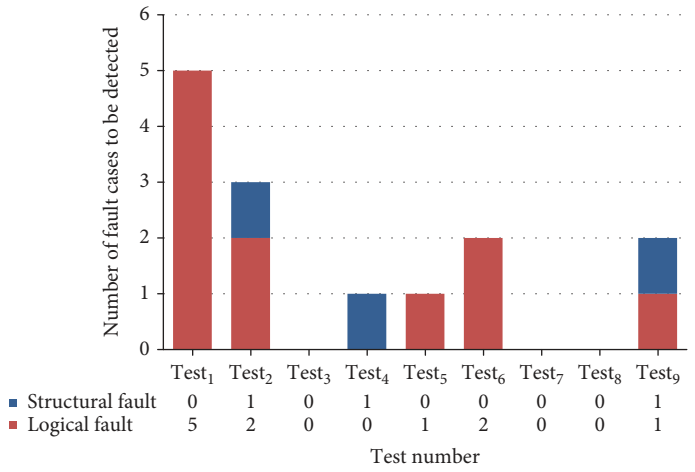


Figure 12: Number of fault cases in tests.

All the works in this study were based on real data acquired from submarine systems. The interface faults regarding incorrect design and abnormal implementation can be resolved during designing and integrating complex systems. The proposed work have facilitated to reduce system development time and avoid dangerous situations during a shipbuilding project. The faults interested in this study are relevant to individual interface data; thus, detection and diagnosis of a sequence of multiple interface data will remain for future work.

Abbreviations

ASCII: American Standard Code for Information Interchange
CTD: Conductivity, temperature, and depth
DEVS: Discrete event system specification
DIS: Data integration system
DSME: Daewoo Shipbuilding and Marine Engineering
EM log: Electromagnetic log
ESM: Electronic Support Measure
FSM: Finite system machine
GPS: Global Positioning System
GUI: Graphical user interface
INS: Inertial navigation system
I/O: Input/output
MVVM: Model-view-viewmodel
OOBN: Object-oriented Bayesian networks
PIP: Product improvement program
SLDC: System development life cycle
SoS: System of systems
WPF: Windows Presentation Foundation.

Data Availability

The interface data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] M. L. Butterfield, J. S. Pearlman, and S. C. Vickroy, "A system-of-systems engineering GEOSS: architectural approach," *IEEE Systems Journal*, vol. 2, no. 3, pp. 321–332, 2008.

[2] L. B. Rainey and A. Tolk, Eds., *Modeling and Simulation Support for System of Systems Engineering Applications*, Wiley, Hoboken, NJ, USA, 2015.

[3] M. Jamshidi, *Systems of Systems Engineering: Principles and Application*, CRC Press, Boca Raton, FL, USA, 2008.

[4] P. D. Groves, *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*, Artech house, London, UK, 2013.

[5] H. Panetto and A. Molina, "Enterprise integration and interoperability in manufacturing systems: trends and issues," *Computers in Industry*, vol. 59, no. 7, pp. 641–646, 2008.

[6] P. Graignic, T. Vosgien, M. Jankovic, V. Tuloup, J. Berquet, and N. Troussier, "Complex system simulation: proposition of a MBSE framework for design-analysis integration," *Procedia Computer Science*, vol. 16, pp. 59–68, 2013.

[7] A. M. Madni and M. Sievers, "Systems integration: key perspectives, experiences, and challenges," *Systems Engineering*, vol. 17, no. 1, pp. 37–51, 2014.

[8] L. Sassaman, M. L. Patterson, S. Bratus, and M. E. Locasto, "Security applications of formal language theory," *IEEE Systems Journal*, vol. 7, no. 3, pp. 489–500, 2013.

[9] D. C. Sturman and G. A. Agha, "A protocol description language for customizing failure semantics," in *Proceedings of IEEE 13th Symposium on Reliable Distributed Systems*, pp. 148–157, Dana Point, CA, USA, October 1994.

[10] D. T. Nguyen, Y. Chae, and Y. Park, "Enhancement of data rate and packet size in image sensor communications by employing constant power 4-PAM," *IEEE Access*, vol. 6, pp. 8000–8010, 2018.

[11] W. Stallings, *Data and Computer Communications*, Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2004.

[12] K.-M. Seo, K.-P. Park, and B.-J. Lee, "Achieving data interoperability of communication interfaces for combat system engineering," *IEEE Access*, vol. 5, pp. 17938–17951, 2017.

[13] J. Wang, *Handbook of Finite State Based Models and Applications*, CRC Press, Boca Raton, FL, USA, 2013.

[14] K.-M. Seo, C. Choi, T. G. Kim, and J. H. Kim, "DEVS-based combat modeling for engagement-level simulation," *Simulation*, vol. 90, no. 7, pp. 759–781, 2014.

[15] M. Hofmann, J. Palii, and G. Mihelcic, "Epistemic and normative aspects of ontologies in modelling and simulation," *Journal of Simulation*, vol. 5, no. 3, pp. 135–146, 2017.

[16] S. Y. Diallo, J. J. Padilla, R. Gore, H. Herencia-Zapana, and A. Tolk, "Toward a formalism of modeling and simulation using model theory," *Complexity*, vol. 19, no. 3, 63 pages, 2014.

[17] B. Cai, H. Liu, and M. Xie, "A real-time fault diagnosis methodology of complex systems using object-oriented Bayesian networks," *Mechanical Systems and Signal Processing*, vol. 80, pp. 31–44, 2016.

[18] G. Lamperti and X. Zhao, "Diagnosis of active systems by semantic patterns," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 8, pp. 1028–1043, 2014.

[19] J. Poon, P. Jain, I. C. Konstantakopoulos, C. Spanos, S. K. Panda, and S. R. Sanders, "Model-based fault detection and identification for switching power converters," *IEEE Transactions on Power Electronics*, vol. 32, no. 2, pp. 1419–1430, 2017.

[20] A. F. de Loza, D. Henry, J. Cieslak, A. Zolghadri, and J. Dávila, "Sensor fault diagnosis using a non-homogeneous high-order sliding mode observer with application to a transport aircraft," *IET Control Theory & Applications*, vol. 9, no. 4, pp. 598–607, 2015.

[21] V. T. Do and U.-P. Chong, "Signal model-based fault detection and diagnosis for induction motors using features of vibration signal in two-dimension domain," *Strojniški vestnik – Journal of Mechanical Engineering*, vol. 57, no. 9, pp. 655–666, 2011.

[22] N. Pan, X. Wu, Y.-L. Chi, X. Liu, and C. Liu, "Combined failure acoustical diagnosis based on improved frequency domain blind deconvolution," *Journal of Physics: Conference Series*, vol. 364, article 012078, 2012.

[23] X. Li, D. Chang, H. Pen, X. Y. Liu, and Y. Yao, "Application of MVVM design pattern in MES," in *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pp. 1374–1378, Shenyang, China, June 2015.

[24] The Korea Economic Daily, "DSME wins 179 Bil. Won project to renovate 3 submarines," 2014, http://english.hankyung.com/business/2014/07/22/1314111/spanclasskeyworddsmespan-wins-179-bil-won-project-to-spanclasskeywordrenovatespan-3-submaries.

[25] D. V. Schrick, "Remarks on terminology in the field of supervision, fault detection and diagnosis," in *Proceedings of the IFAC Symposium on Fault Detection, Supervision Safety for Technical Processes*, pp. 959–964, Kingston upon Hull, UK, August 1997.

[26] M. R. Khaefi, J.-Y. Im, and D.-S. Kim, "An efficient DDS node discovery scheme for naval combat system," in *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 1–8, Luxembourg, September 2015.

[27] K.-M. Seo, W. Hong, and T. G. Kim, "Enhancing model composability and reusability for entity-level combat simulation: a conceptual modeling approach," *Simulation*, vol. 93, no. 10, pp. 825–840, 2017.

[28] NAVSEA, "Combat system engineering and integration," 2017, http://www.navsea.navy.mil/Portals/103/Documents/NSWC_Dahlgren/LeadingEdge/CSEI/CombSys.pdf.

[29] P.-Y. Chen, S. Yang, and J. A. McCann, "Distributed real-time anomaly detection in networked industrial sensing systems," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3832–3842, 2015.

[30] C. H. Porter, C. Villalobos, D. Holzworth et al., "Harmonization and translation of crop modeling data to ensure interoperability," *Environmental Modelling & Software*, vol. 62, no. 2014, pp. 495–508, 2014.

[31] B. S. Ahmed, K. Z. Zamli, W. Afzal, and M. Bures, "Constrained interaction testing: a systematic literature study," *IEEE Access*, vol. 5, pp. 25706–25730, 2017.

[32] S. Lin, Y. Wang, and L. Jia, "System reliability assessment based on failure propagation processes," *Complexity*, vol. 2018, Article ID 9502953, 19 pages, 2018.

[33] S. D. Eppinger, N. R. Joglekar, A. Olechowski, and T. Teo, "Improving the systems engineering process with multilevel analysis of interactions," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 28, no. 4, pp. 323–337, 2014.

[34] E. Santos and Y. Zhao, "Automatic emergence detection in complex systems," *Complexity*, vol. 2017, Article ID 3460919, 24 pages, 2017.

[35] B. Sklar, *Digital Communications Fundamentals and Applications*, Prentice-Hall, Upper Saddle River, NJ, USA, 2001.

[36] B. A. Forouzan, *Data Communications and Networking*, McGraw-Hill, New York, NY, USA, 2006.

[37] Z. L. Wang, X. Yin, and C. M. Jing, "A formal method to real-time protocol interoperability testing," *Science in China Series F: Information Sciences*, vol. 51, no. 11, pp. 1723–1744, 2008.

[38] S.-C. Shin, J.-G. Shin, and D.-K. Oh, "Development of data analysis tool for combat system integration," *International Journal of Naval Architecture and Ocean Engineering*, vol. 5, no. 1, pp. 147–160, 2013.

[39] R. Isermann, "Model-based fault-detection and diagnosis – status and applications," *Annual Reviews in Control*, vol. 29, no. 1, pp. 71–85, 2005.

[40] X. Liu, Z. Gao, and M. Z. Q. Chen, "Takagi–Sugeno fuzzy model based fault estimation and signal compensation with application to wind turbines," *IEEE Transactions on Industrial Electronics*, vol. 64, no. 7, pp. 5678–5689, 2017.

[41] Z. Gao, X. Liu, and M. Chen, "Unknown input observer-based robust fault estimation for systems corrupted by partially decoupled disturbances," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 4, pp. 2537–2547, 2015.

[42] Z. Gao, C. Cecati, and S. X. Ding, "A survey of fault diagnosis and fault-tolerant techniques – part I: fault diagnosis with model-based and signal-based approaches," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 6, pp. 3757–3767, 2015.

[43] V. H. Ferreira, R. Zanghi, M. Z. Fortes et al., "A survey on intelligent system application to fault diagnosis in electric power system transmission lines," *Electric Power Systems Research*, vol. 136, pp. 135–153, 2016.

[44] I. V. de Bessa, R. M. Palhares, M. F. S. V. D'Angelo, and J. E. Chaves Filho, "Data-driven fault detection and isolation scheme for a wind turbine benchmark," *Renewable Energy*, vol. 87, pp. 634–645, 2016.

[45] X. Shuiqing, Z. Ke, C. Yi, H. Yigang, and F. Li, "Gear fault diagnosis in variable speed condition based on multiscale chirplet path pursuit and linear canonical transform," *Complexity*, vol. 2018, Article ID 3904598, 8 pages, 2018.

[46] B. S. Kim, B. G. Kang, S. H. Choi, and T. G. Kim, "Data modeling versus simulation modeling in the big data era: case study of a greenhouse control system," *Simulation*, vol. 93, no. 7, pp. 579–594, 2017.

[47] S. J. E. Taylor, A. Khan, K. L. Morse et al., "Grand challenges for modeling and simulation: simulation everywhere-from cyberinfrastructure to clouds to citizens," *Simulation*, vol. 91, no. 7, pp. 648–665, 2015.

[48] W. A. Khan, A. M. Khattak, M. Hussain et al., "An adaptive semantic based mediation system for data interoperability among health information systems," *Journal of Medical Systems*, vol. 38, no. 8, p. 28, 2014.

[49] A. Vijayaraghavan, W. Sobel, A. Fox, D. Dornfeld, and P. Warndorf, "Improving machine tool interoperability using standardized interface protocols: MTConnect™," in *Proceedings of 2008 International Symposium on Flexible Automation*, pp. 1–6, Atlanta, GA, USA, June 2008.

[50] E. Sorensen and M. I. Mikailesc, "Model-view-ViewModel (MVVM) design pattern using Windows Presentation Foundation (WPF) technology," *MegaByte Journal*, vol. 9, no. 4, pp. 1–19, 2010.

[51] R. Garofalo, *Building Enterprise Applications with Windows Presentation Foundation and the Model View View Model Pattern*, Microsoft Press, 2011.

[52] B. P. Ziegler, T. G. Kim, and H. Praehofer, *Theory of Modeling and Simulation Integrating Discrete Event and Continuous Complex Dynamic Systems*, Academic Press, 2000.

[53] B.-G. Kang, K.-M. Seo, and T. G. Kim, "Communication analysis of network-centric warfare via transformation of system of systems model into integrated system model using neural network," *Complexity*, vol. 2018, Article ID 6201356, 16 pages, 2018.

[54] S. Choi, K.-M. Seo, and T. Kim, "Accelerated simulation of discrete event dynamic systems via a multi-fidelity modeling framework," *Applied Sciences*, vol. 7, no. 10, p. 1056, 2017.

[55] A. Backurs and P. Indyk, "Which regular expression patterns are hard to match?," in *Proceedings of the 57th Annual Symposium on Foundations of Computer Science*, pp. 457–466, New Bruswick, NJ, USA, October 2016.

[56] Defense-Aerospace, "France awards contract for the mid-life upgrade of its Mirage 2000D fighters," 2016, http://www.defense-aerospace.com/article-view/release/175655/france-awards-mlu-contract-for-mirage-2000d-fleet.html.

[57] Navy Recognition, "Navantia & Indra to modernize Indonesian Navy corvette KRI Malahayati combat system," 2016, August 2017, http://www.navyrecognition.com/index.php/news/defence-news/2016/november-2016-navy-navalforces-defense-industry-technology-maritime-security-global-news/4559-navantia-indra-to-modernize-indonesian-navy-corvette-kri-malahayaticombat-system.html.

[58] Black & Veatch, "Plant improvement engineering services," 2018, https://www.bv.com/docs/energy-brochures/plant-improvement-engineering-services.pdf.

[59] Austin Energy, "Pumping system improvement project saves energy and improves performance at a power plant project summary," 2018, https://www.nrel.gov/docs/fy05osti/37537.pdf.