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

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 of 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
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.
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.

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

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

$$SM_{Message} = \langle X, Y, S, \delta_{ext}, \omega \rangle$$

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

$$X = \{ (f_i, l_i) | 1 \leq i \leq n \},$$

$f_i$ 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

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

$d_j$ 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;

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

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

Components of $SM_{Message}$ are contained within $\langle \rangle$. Every notation in $SM_{Message}$ 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 $\rightarrow$ 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, $\delta_{ext}$, receives all the fields comprising a message and appropriate delimiters.
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:

\[
LM_{\text{Field}} = \langle X, Y, S, \delta_{\text{ext}}, \delta_{\text{int}}, \omega \rangle
\]

\(X\) is an input set of fields, where

\(X = \{v_{\text{tgt}}, v_{\text{ref}}\}\),

\(v_{\text{tgt}}\) is the initial value of the target field,

\(v_{\text{ref}}\) is the initial value of the reference field;

\(Y\) is an output set of logical faults, where

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

\(v\) 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

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

\(f_{\text{int}}\) is the initial value of the target field,

\(f_{\text{dec}}\) is the decoded value of \(f_{\text{int}}\) after the previous transition function,

\(P = \{p_{j} | 1 \leq j \leq n\}\) is a set of parameters for logical rules,

\(R = \{(r_{t}, r_{f}) | r_{t}, r_{f} \in \{\text{false}, \text{true}, \text{idle}\}\}\) 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 \(LM_{\text{Field}}\) is the field transition functions, (i.e., \(\delta_{\text{ext}}\) and \(\delta_{\text{int}}\)); thus, we have summarized their practical

<table>
<thead>
<tr>
<th>Structure type</th>
<th>(x \in 2^X)</th>
<th>(d \in D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header and footer</td>
<td>((f_1, l_1))</td>
<td>(d_{\text{hdr}})</td>
</tr>
<tr>
<td></td>
<td>((f_2, l_2))</td>
<td>(d_{\text{ftr}})</td>
</tr>
<tr>
<td>Header and length</td>
<td>((f_1, l_1))</td>
<td>(d_{\text{hdr}})</td>
</tr>
<tr>
<td></td>
<td>((f_2, l_2))</td>
<td>(d_{\text{ftr}})</td>
</tr>
<tr>
<td>Header</td>
<td>((f_1, l_1))</td>
<td>(d_{\text{hdr}})</td>
</tr>
<tr>
<td>Footer</td>
<td>((f_2, l_2))</td>
<td>(d_{\text{ftr}})</td>
</tr>
<tr>
<td>Length</td>
<td>((f_1, l_1))</td>
<td>(d_{\text{length}})</td>
</tr>
</tbody>
</table>

\(f_1\) and \(f_2\) mean the first and last fields, respectively. \((f_1, l_1)\) 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, \(\delta_{\text{ext}}\) 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, \(\omega\), 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 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, \(\omega\) 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, \((f_1, l_1)\) 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:
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 5 special functions: user-defined equation, regular expression [55], and ASCII expression ($\delta_{12}$ to $\delta_{14}$ 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 LMField.

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 ($\delta_{ext}$, $\delta_{ext}$).

(2) If the bearing field is not available, the system confirms that its hexadecimal values are all 0x20 ($\delta_{14}$).

(3) If the bearing field is available, the system next identifies if the field is newly updated by comparing its values with predefined patterns ($\delta_{13}$).

(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 ($\delta_{12}$).

(5) The system finally converses its current data type, that is, ASCII characters, into unsigned integers ($\delta_{int}$).

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 $\delta$ 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 $v_{brg}$;

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

$S = \{(f_{int}, f_{dec})\} \times \{p_1, p_2, p_3, p_4\} \times \{(r_1, r_f)\}$, where

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

$\delta$ is for either $\delta_{ext}$ or $\delta_{int}$. 

Table 4: Examples of $\delta$ classified into 5 operations.
If the comparison is $f \times 0$, as a regular expression,

- $v_{brg} = false$ → false
- $v_{brg} = true$ → true

$p_1 = 0 \times 20$ as a hexadecimal number,
$p_2 = [p - s][0 - 5][0 - 9]$ as a regular expression,
$p_3 = [0 - 3][0 - 5][0 - 9]$ as a regular expression,
$p_4 = unsigned integer as data type;

$\delta_{int}^{10} : (v_{tgt}) \times (p_1) \times (idle, idle)$
$\rightarrow (f_{int}, f_{dec}) \times (true, false) \text{ if } v_{brg} \text{ is not available},$
$\rightarrow (f_{int}, f_{dec}) \times (false, false) \text{ if } v_{brg} \text{ is available};$

$\delta_{int}^{15} :$
$(v_{tgt}) \times \emptyset \times (true, false)$
$\rightarrow (f_{int}, f_{dec}) \times (true, false),$
$(v_{tgt}) \times \emptyset \times (false, false)$
$\rightarrow (f_{int}, f_{dec}) \times (false, false);$

$\delta_{int}^{14} :$
$(f_{int}, f_{dec}) \times (p_1) \times (true, false)$
$\rightarrow (f_{int}, f_{dec}) \times (true, true) \text{ if } v_{tgt} \text{ is correct},$
$(f_{int}, f_{dec}) \times (false, false) \text{ if } v_{tgt} \text{ is not correct};$

$\delta_{int}^{13.1} :$
$(f_{int}, f_{dec}) \times (p_2) \times (false, false)$
$\rightarrow (f_{int}, f_{dec}) \times (true, true) \text{ if the comparison is correct},$
$(f_{int}, f_{dec}) \times (false, false) \text{ if the comparison is not correct};$

$\delta_{int}^{13.2} :$
$(f_{int}, f_{dec}) \times (p_3) \times (false, false)$
$\rightarrow (f_{int}, f_{dec}) \times (true, false) \text{ if the comparison is correct},$
$(f_{int}, f_{dec}) \times (false, true) \text{ if the comparison is not correct};$

$\delta_{int}^{9} : (f_{int}, f_{dec}) \times (p_4) \times (true, false) \rightarrow (f_{int}, f_{dec}) \times (true, true);$

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

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, Node1 corresponds to $X$ of LM_field. Node6 and Node7 are relevant to Y, and the others are represented by two transition functions: $\delta_{ext}$ or $\delta_{int}$. 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, LM_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...
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 LMbearing. 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.

<table>
<thead>
<tr>
<th>SDLC</th>
<th>Test no.</th>
<th>When (yy.mm)</th>
<th>Test site</th>
<th>Subsystems of local interfaces</th>
<th>Objective</th>
<th>Test period</th>
</tr>
</thead>
</table>
| 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].
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 test3 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., $\delta_{int}^{1}$, $\delta_{int}^{4}$, $\delta_{int}^{12}$, and $\delta_{int}^{13}$) were used to model the logic of the velocity field.
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, Msg G-2, the longest message, has 85 fields in 284 bytes. Whereas, Msg J-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 Msg B-2 and Msg E-1 to explain specific types of message delimiters. First, since Msg B-2 has a fixed-length without any header and footer, it should be classified with a message length (i.e., \( d_{\text{length}} \)). On the other hand, the length of Msg E-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, Msg E-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 \( d_{\text{hdr}} \) and \( d_{\text{ftr}} \). To sum up, the messages in this study used two structural types: (1) \( d_{\text{hdr}} \) and \( d_{\text{ftr}} \) and (2) \( d_{\text{length}} \).

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

Table 6: Main results of interface data modeling.

<table>
<thead>
<tr>
<th>Interface type</th>
<th>Message type</th>
<th>Structure modeling (SM)</th>
<th>Logic modeling (LM)</th>
<th>Decoded value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>( \sum_{i=1}^{n} l_i \in X )</td>
<td>( N(X) )</td>
<td>( d_{\text{hdr}} )</td>
</tr>
<tr>
<td>Sensor_A</td>
<td>MsgA-1</td>
<td>14</td>
<td>6</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_A</td>
<td>MsgA-2</td>
<td>9</td>
<td>4</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_B</td>
<td>MsgB-1</td>
<td>28</td>
<td>20</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_B</td>
<td>MsgB-2</td>
<td>30</td>
<td>27</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-1</td>
<td>4</td>
<td>2</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-2</td>
<td>32</td>
<td>9</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-3</td>
<td>44</td>
<td>16</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-4</td>
<td>50</td>
<td>4</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-5</td>
<td>54</td>
<td>18</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-6</td>
<td>6</td>
<td>3</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-7</td>
<td>4</td>
<td>2</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-8</td>
<td>32</td>
<td>9</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-9</td>
<td>44</td>
<td>10</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-10</td>
<td>54</td>
<td>16</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-11</td>
<td>62</td>
<td>15</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-12</td>
<td>52</td>
<td>17</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-13</td>
<td>6</td>
<td>3</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-14</td>
<td>24</td>
<td>7</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_C</td>
<td>MsgC-15</td>
<td>6</td>
<td>4</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_D</td>
<td>MsgD-1</td>
<td>17</td>
<td>7</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_E</td>
<td>MsgE-1</td>
<td>Variable length</td>
<td>4</td>
<td>O</td>
</tr>
<tr>
<td>Sensor_E</td>
<td>MsgE-2</td>
<td>9</td>
<td>5</td>
<td>O</td>
</tr>
<tr>
<td>Sensor F</td>
<td>MsgF-1</td>
<td>236</td>
<td>67</td>
<td>O</td>
</tr>
<tr>
<td>Control G</td>
<td>MsgG-1</td>
<td>7</td>
<td>4</td>
<td>O</td>
</tr>
<tr>
<td>Control G</td>
<td>MsgG-2</td>
<td>284</td>
<td>85</td>
<td>O</td>
</tr>
<tr>
<td>Sensor H</td>
<td>MsgH-1</td>
<td>26</td>
<td>16</td>
<td>O</td>
</tr>
<tr>
<td>Sensor H</td>
<td>MsgH-2</td>
<td>75</td>
<td>17</td>
<td>O</td>
</tr>
<tr>
<td>Sensor I</td>
<td>MsgI-1</td>
<td>3</td>
<td>7</td>
<td>O</td>
</tr>
<tr>
<td>Sensor I</td>
<td>MsgI-2</td>
<td>3</td>
<td>7</td>
<td>O</td>
</tr>
<tr>
<td>Sensor I</td>
<td>MsgI-3</td>
<td>3</td>
<td>7</td>
<td>O</td>
</tr>
<tr>
<td>Sensor I</td>
<td>MsgI-4</td>
<td>3</td>
<td>7</td>
<td>O</td>
</tr>
<tr>
<td>Control J</td>
<td>MsgJ-1</td>
<td>18</td>
<td>90</td>
<td>O</td>
</tr>
</tbody>
</table>

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.
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).
To be specific, the fourth field was modeled not to send STS. However, during test 3, the relevant system actually operated during the vessel moves above the specified 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 3 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 obtained 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., $N_{\text{total}}$ 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., $\text{Msg}_{G,1}$ and $\text{Msg}_{G,2}$) 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, $N_{\text{total}}$ of Sensor I is arithmetically 128 times more than that of Control G if they operate in the same amount of time (128 = $(4/2) \times (8/0.125)$).

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 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.

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Test1</td>
<td>$N_{\text{total}}$</td>
<td>677,229</td>
<td>1,744,894</td>
<td>288,906</td>
<td>314,987</td>
<td>21,403</td>
<td>2555</td>
<td>39,529</td>
<td>618,028</td>
</tr>
<tr>
<td></td>
<td>$N_{\text{fault}}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1327</td>
<td>14,366</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test2</td>
<td>$N_{\text{total}}$</td>
<td>381,452</td>
<td>741,652</td>
<td>66,487</td>
<td>134,920</td>
<td>102,586</td>
<td>14,013</td>
<td>16,862</td>
<td>162,438</td>
</tr>
<tr>
<td></td>
<td>$N_{\text{fault}}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>76,828</td>
<td>0</td>
<td>1168</td>
<td>0</td>
</tr>
<tr>
<td>Test3</td>
<td>$N_{\text{total}}$</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>$N_{\text{fault}}$</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Test4</td>
<td>$N_{\text{total}}$</td>
<td>1,294,454</td>
<td>1,524,532</td>
<td>188,250</td>
<td>277,205</td>
<td>2,691,498</td>
<td>52,296</td>
<td>34,650</td>
<td>554,447</td>
</tr>
<tr>
<td></td>
<td>$N_{\text{fault}}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8213</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test5</td>
<td>$N_{\text{total}}$</td>
<td>1,021,158</td>
<td>1,238,562</td>
<td>26,157</td>
<td>225,154</td>
<td>2,132,133</td>
<td>12,911</td>
<td>28,144</td>
<td>450,184</td>
</tr>
<tr>
<td></td>
<td>$N_{\text{fault}}$</td>
<td>0</td>
<td>0</td>
<td>9509</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test6</td>
<td>$N_{\text{total}}$</td>
<td>578,971</td>
<td>2,070,717</td>
<td>138,547</td>
<td>376,578</td>
<td>128,331</td>
<td>3830</td>
<td>47,072</td>
<td>489,510</td>
</tr>
<tr>
<td></td>
<td>$N_{\text{fault}}$</td>
<td>0</td>
<td>0</td>
<td>30,193</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test7</td>
<td>$N_{\text{total}}$</td>
<td>197,753</td>
<td>629,961</td>
<td>56,431</td>
<td>112,904</td>
<td>85,332</td>
<td>68</td>
<td>14,003</td>
<td>214,499</td>
</tr>
<tr>
<td></td>
<td>$N_{\text{fault}}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test8</td>
<td>$N_{\text{total}}$</td>
<td>1,151,185</td>
<td>2,076,729</td>
<td>188,025</td>
<td>372,984</td>
<td>3,664,990</td>
<td>991</td>
<td>39,169</td>
<td>750,569</td>
</tr>
<tr>
<td></td>
<td>$N_{\text{fault}}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Test9</td>
<td>$N_{\text{total}}$</td>
<td>N/A</td>
<td>11,004</td>
<td>N/A</td>
<td>N/A</td>
<td>10,582</td>
<td>N/A</td>
<td>N/A</td>
<td>3496</td>
</tr>
<tr>
<td></td>
<td>$N_{\text{fault}}$</td>
<td>N/A</td>
<td>2331</td>
<td>N/A</td>
<td>N/A</td>
<td>0</td>
<td>N/A</td>
<td>N/A</td>
<td>3496</td>
</tr>
</tbody>
</table>

$N_{\text{total}}$ is the number of obtained messages from each interface. $N_{\text{fault}}$ is the number of faulted messages from each interface. Some interface is not applicable for a specific test, which is represented by N/A.
In Table 7, the interfaces where local faults were detected are marked in italics. The number of the local faults (i.e., $N_{\text{fault}}$) was counted if $r_j \in \text{SM}, Y$ or $r \in \text{LM}, Y$ has False once in an individual message. For example, in Sensor$_C$ of test$_6$, 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$_F$, Sensor$_D$, Sensor$_I$, and Control$_J$ were faulted. Note that Sensor$_C$ and Control$_C$ 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$_B$, 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.

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 $r_j \in \text{SM}, Y$ 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.

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

<table>
<thead>
<tr>
<th>Test number</th>
<th>Fault case</th>
<th>Interface type</th>
<th>Message type</th>
<th>Interface data</th>
<th>Fault diagnosis</th>
<th>Fault category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test$_1$</td>
<td>Fault$_1$</td>
<td>Sensor$_F$</td>
<td>Msg$_{g-1}$</td>
<td>02-30-31-30-39-33-37-...35-20-20-20-20-20-30-30-33-35-...-03</td>
<td>The modeling should be revised that all the fields regarding a specific section are full of &quot;0x20&quot; if no targets are detected in the section.</td>
<td>Logical fault</td>
</tr>
<tr>
<td>Test$_2$</td>
<td>Fault$_1$</td>
<td>Control$_G$</td>
<td>Msg$_{g-2}$</td>
<td>0D-0A-48-32-35-31-31-...20-2B-30-38-35-33-...53-2B-...</td>
<td>The modeling should be revised that the interpreted value of the sign field can be &quot;*&quot; although the precondition field is unavailable.</td>
<td>Logical fault</td>
</tr>
<tr>
<td>Test$_3$</td>
<td>Fault$_1$</td>
<td>Control$_G$</td>
<td>Msg$_{g-2}$</td>
<td>0D-0A-48-32-35-31-31-...20-2B-30-38-35-33-...53-2B-...</td>
<td>The modeling should be revised that the speed field has meaningful information although the precondition field is unavailable.</td>
<td>Logical fault</td>
</tr>
<tr>
<td>Test$_4$</td>
<td>Fault$_1$</td>
<td>Control$_G$</td>
<td>Msg$_{g-2}$</td>
<td>0D-0A-48-32-35-31-31-...20-2B-30-38-35-33-...53-2B-...</td>
<td>The modeling should be revised that the interpreted value of the sign field can be &quot;*&quot; although the precondition field is unavailable.</td>
<td>Logical fault</td>
</tr>
<tr>
<td>Test$_5$</td>
<td>Fault$_1$</td>
<td>Control$_G$</td>
<td>Msg$_{g-2}$</td>
<td>0D-0A-48-32-35-31-31-...20-2B-30-38-35-33-...53-2B-...</td>
<td>The modeling should be revised that the speed field has meaningful information although the precondition field is unavailable.</td>
<td>Logical fault</td>
</tr>
<tr>
<td>Test$_6$</td>
<td>Fault$_1$</td>
<td>Control$_G$</td>
<td>Msg$_{g-2}$</td>
<td>0D-0A-48-32-35-31-31-...20-2B-30-38-35-33-...53-2B-...</td>
<td>The modeling should be revised that the speed field has meaningful information although the precondition field is unavailable.</td>
<td>Logical fault</td>
</tr>
<tr>
<td>Test$_7$</td>
<td>Fault$_1$</td>
<td>Control$_G$</td>
<td>Msg$_{g-2}$</td>
<td>0D-0A-48-32-35-31-31-...20-2B-30-38-35-33-...53-2B-...</td>
<td>The modeling should be revised that the speed field has meaningful information although the precondition field is unavailable.</td>
<td>Logical fault</td>
</tr>
<tr>
<td>Test$_8$</td>
<td>Fault$_1$</td>
<td>Control$_G$</td>
<td>Msg$_{g-2}$</td>
<td>0D-0A-48-32-35-31-31-...20-2B-30-38-35-33-...53-2B-...</td>
<td>The modeling should be revised that the speed field has meaningful information although the precondition field is unavailable.</td>
<td>Logical fault</td>
</tr>
<tr>
<td>Test$_9$</td>
<td>Fault$_1$</td>
<td>Control$_G$</td>
<td>Msg$_{g-2}$</td>
<td>0D-0A-48-32-35-31-31-...20-2B-30-38-35-33-...53-2B-...</td>
<td>The modeling should be revised that the speed field has meaningful information although the precondition field is unavailable.</td>
<td>Logical fault</td>
</tr>
</tbody>
</table>

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].
Before test 3, MsgE-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 $N_{\text{Mtotal}}$ in Figure 11 is regarding the same message type, it is a subset of $N_{\text{Itotal}}$ 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, fault3 brought about more than 14,000 faulted messages during 72 hours. Next, fault2 and fault1 cause the same number of faulted messages, which means that they were complementary and occurred at the same time. Additionally, fault2 and fault12 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, MsgG-2 in Table 7, whose field has “0” as an interpretation value before test5, 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, 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 test8 made the corresponding subsystems resolve their unexpected behaviors at the system integration phase. For example, SensorB 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 test8 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 test9. 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.
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


