Journal of *Marine Science and Engineering*

MDPI

Article

Continuous Contract Based Verification of Updates in Maritime Shipboard Equipment

Georg Hake [1,*], Carl Philipp Hohl [2] and Axel Hahn [1]

[1] Department of Computing Science, University of Oldenburg, 26129 Oldenburg, Germany; axel.hahn@uol.de
[2] FZI Research Center for Information Technology, 76131 Karlsruhe, Germany; hohl@fzi.de
* Correspondence: georg.hake@uol.de

Abstract: Modern control systems in the maritime domain are increasingly controlled by software systems and become subject to updates and configuration changes during operation. Moreover, with the shift to autonomous vessels and cars, these software-based systems are taking on more and more safety-critical tasks, so the risks associated with system failures are increasing. Unlike before, it becomes necessary to verify the continuously adapting modules of a vehicle not only before deployment, but to establish continuous verification capabilities during all phases of the product lifecycle, from the design to the system in operation. Hence, in case of an update, deviations from the expected behavior can be automatically detected and relevant measures can be initiated. In this work, a contract-based verification framework is presented that includes automatable and formally analyzable behavioral descriptors in form of assumption-guarantee contracts for all phases of the software lifecycle to provide static and dynamic verification capabilities alongside a dynamically changing system composition. By utilizing contractually defined behavior descriptions, classic test procedures, such as simulations, are supplemented by a formally testable level that is applied to all phases of the update process. A conceptual-deductive methodology was chosen, building on the identified requirements to develop an overarching update framework that adds contractual descriptions to the traditional development case. Based on the presented framework, the verifiable modification of a safety-critical software system is demonstrated. The approach is evaluated using a maritime collision-avoidance system and the verification steps are evaluated along the update process. The framework offers a novel approach to complement existing test procedures by enabling formal impact analysis and incremental verification of updates.

Keywords: updates; assumption-guarantee contracts; life cycle; adaptive; corrective; perfective; verification; impact analysis; safety

Citation: Hake, G.; Hohl, C.P.; Hahn, A. Continuous Contract Based Verification of Updates in Maritime Shipboard Equipment. *J. Mar. Sci. Eng.* **2021**, *9*, 688. https://doi.org/10.3390/jmse9070688

Academic Editor: Spyros Hirdaris

Received: 25 May 2021
Accepted: 17 June 2021
Published: 23 June 2021

1. Introduction

The control and monitoring units used to regulate a ship's distributed electrical and mechanical machinery, navigation, balance and steering systems are increasingly being replaced by software-based modules that converge at a central location. This trend continues with the introduction of Maritime Autonomous Surface Ships (MASSs) where control is built entirely by software-based systems, replacing the responsibilities of the human operator so that the potential for misconfiguration can have dire consequences. Hence, it becomes necessary to verify the integration of these safety-critical systems and their interaction with each other within a heterogeneous system complex. As, the individual modules derive from different developers and suppliers, it needs to be verified that the integration of the individual parts into a complete system of sub-systems results in a valid configuration [1].

In order to avoid possible errors when assembling the modules into an overall system, so far the composition for a ship was approved before deployment for a fixed system configuration with regard to functional, time and safety characteristics, yet seldom changed

after set into operation [2]. However, the possibility for remote Over-The-Air (OTA) updates allows for modifications even after deployment and during operations [3]. Moreover, unforeseen environmental situations or the unpredictable entry into a system alliance with previously unknown vehicles can lead to unanticipated configurations.

There are plenty of examples of misbehavior due to software updates and unforeseen interdependencies. For example, BIMCO reports of "incidents, where ships for example, suffer complete blackouts, and malfunctions in radar and other related systems, as a result of unforeseen difficulties with a software update" [4]. Other examples can be found in the Viking Cruise ship incident in March 2019, in which a faulty sensor reading reported an oil level that was too low, triggering a chain reaction of dependent systems that led to a complete shutdown of systems and a near grounding of the ship [5]. Finally, the known grounding of the Royal Majesty is also due to the fact that the Global Positioning System (GPS) module returned incorrect navigation data of which the crew was not aware of [6].

The examples demonstrate, the necessity for proving that a modified system configuration still meets the previously approved safety criteria after an update and during operation. Since the approval process has so far been designed to certify a fixed system composition at a set time before deployment, it is therefore necessary to redesign the verification process for dynamically changing systems and keep the re-verification effort after a change as low as possible. For this reason there have already been numerous efforts, both on the part of the producers of system modules and on the part of the certification bodies, to establish processes that enable the controlled import of updates. For example, a working group of CIRM/BIMCO [4], as well as the IACS [7] and the International Organization for Standardization (ISO) [8] made suggestions on how an update process must be structured.

For an update to one module of a system this includes checking the temporal, functional and safety related interdependencies between the system's constituents on the basis of a scenario space that is as broadly covered as possible, optimally limited to the impacted parts of the overall system. Traditionally, this took the form of simulative testing procedures that used targeted scenario selection to demonstrate that an overall system met its promised goals and that hazards could be sufficiently excluded. A recent approach has been conducted by the Open Simulation Platform (OSP), which uses digital twins and co-simulation to provide test management systems for heterogeneous modules from different vendors [9]. However, such an approach lacks completeness, formal correctness, and deterministic behavior [10].

This is where this work comes in, by providing a formal complement to existing virtual verification efforts, such as the OSP. With the application of assumption-guarantee contracts to the model of the system each of its modules get annotated with a formal specification about its behavior under the expected environmental conditions. This extends the model with formal specifications that enable Virtual Integration Testing (VIT) of the modules to identify inconsistencies early on that would not be fully covered by a simulative test procedure.

To make this possible two requirements need to be met. Firstly, the system development must be undertaken in a modular way instead of the traditional monolithic development approach. The overall system then consists of individual modules, which in turn form a system network and declare clearly defined interface descriptions and services so that the interaction between the modules is separated and traceable. This facilitates the identification of dependencies between individual modules during an update and narrows down the detection of an update's impact.

Secondly, the provided services on the interfaces must be formally defined. For this, assumption-guarantee contracts provide this capability by founding on a theoretical basis that provides methods to describe a module in the form of premises about the module's environment and assurances about the output that enable to examine and justify the combination of heterogeneous modules. The supplier of a module update can provide (i) solely a contract description of his module, (ii) a model and the assigned contracts of the module and its setting within the system configuration and eventually (iii) the

implemented binary of the module with the contracts as metadata for monitoring and test purposes.

For this reason, the update management of a dynamically changing system is subdivided into its different update types (perfective, adaptive, corrective) and the contracts are applied to the modules to identify deviations from expected behavior in all phases of the system's lifecycle. Along the whole process a feedback loop is applied to monitor an approved system, which is already in operation, based on the assigned contracts and to initiate appropriate steps as soon as a breach of contract occurs in one of the components of the composite system.

What sets the presented approach apart from others is the division of the entire development, delivery, integration and verification process into overlapping phases, which merge into each other and at the same time contain refinement stages and back-references, so that a development cycle is created. In addition, classical test and simulation concepts are extended by a formalization stage extended by contracts, so that a system model can be developed from a very abstract informally described model to a detailed, verified and executable system, while at the same time all stages are connected and can be linked back together.

The methodology used in this work corresponds to a conceptual-deductive approach. The requirements of the research gap, which were identified in the course of a literature review, are conceptually transferred into an overall process framework in the course of an argumentative procedure and demonstrated by means of a case study [11].

For this, the presented work is structured as follows. First, an overview of the update process of software components for safety-critical software systems on domain-independent vessels is presented. Building on the higher-level process view, the theoretical concepts behind contract-based modeling and compositional verification capabilities are introduced. Complementing this, a new interaction model between the relevant stakeholders is proposed. After delineating the framework, it is shown how each of the phases of the overall process can be complemented with contracts and how this enables verification within each process step. Thereupon, the intercommunication steps between the phases are shown in order to demonstrate the higher-level verification. To demonstrate the approach, the process steps are extended along a generic minimal example and the different system changes in the form of three different update types. The domain-agnostic approach is finally evaluated using a domain-specific, safety-critical collision avoidance system from the maritime domain.

2. Related Work

Stakeholders in the maritime industry actively pursue activities to improve the maintenance process for shipboard equipment. A joint working group from several industry members [4] as well as the IACS [7] led to recommendations and best practices on how an update should be performed and what needs to be logged. Another approach to virtual verification of software modules in a simulation context is offered by the Open Simulation Platform, which enables co-simulation of models from different vendors based on Functional Mock-up Units (FMUs) [12]. The approach can be extended by contracts in that it can represent deterministic behavior and compensate for the semantic gap [10].

Further criticism of the current process of update management on maritime systems and efforts to change it comes from ISO, which wants to improve the logging of changes on ship equipment by introducing a standardized software log that will record data traffic between modules in a uniform manner [8]. The problem was taken further by [13] in assessing the general standardization of ship component interoperability for analysis and proposing changes.

A similar approach to linking safety evidence, the assertion and the argument that the evidence supports the assertion is presented by the Dependability Engineering Innovation for Cyber Physical System (CPS) (DEIS) project [14] in the form of Digital Dependability Identitiess (DDIs). DDIs are based on the Open Dependability Exchange (ODE) meta-model

and include the complete dependability information of the CPS component its describing. DDIs are based at their core on the Structured Assurance Case Metamodels (SACMs) 2.0 of the Object Management Group (OMG) [15] which is related to the Goal Structuring Notation (GSN). For the integration of individual DDIs into a System of System (SoS) in a semi- to full-automated way, the interfaces of the DDIs for different IP Protection Levels levels (White-, Grey, Black-Box) have to be formalized for this purpose, where in the fully automated Black-Box and Runtime-Case Conditional Safety Certificatess (Consertss) are used. These have a large overlap with the contract-based approach presented here, as they are based on a service-based system architecture model and assumption-guarantee approach [16].

Another approach of model-driven engineering and dependency analysis of complex safety-critical systems is shown by [17] within the CHESS framework. Chess provides domain-independent modeling and analysis tool support starting from requirements analysis, through the design phase, to deployment on the target platform. By applying V&V tools such as OCRA, nuXmv and xSAP, contract-based testing procedures are also enabled. These include model checking, contract-based composition analysis of state machines, and contract-based safety analysis [17].

Another branch of research is Continuous Development, in which contract-based approaches are also used. Warg et al. [18] present a development process for dependent systems and show how continuous assurance processes can be realized with the help of contracts for product lines. As with this work, contracts are used and demonstrated along the development process of dynamically changing systems, but not along all phases of the product's lifecycle and not specifically for all forms of update processes.

Further work in the application of contracts led to approaches for multi-viewpoint consistency for concurrent design [19] and a platform-based design methodology that provides vertical and horizontal integration [20]. Furthermore, [21] show how an update process can gain self-protective capabilities by introducing contracts in distributed embedded systems.

While all the approaches presented here highlight a subsection of the overall process from a concrete perspective, in this paper we highlight the interconnection of the individual modules within the overall process and thus elaborate an overall approach which enables end-to-end verification in the maritime domain.

3. Overarching Update Verification Process

In the following, an birds-eye view on the overall process is introduced. As depicted in Figure 1, six different phases for the update process are distinguished. It is assumed that the system under consideration is already in operation and has gone through a complete development and acceptance process. The steps presented here thus represent the process of an update of that existing system. Nevertheless, in order to develop safe updates, each update cycle must be considered as a new development process.

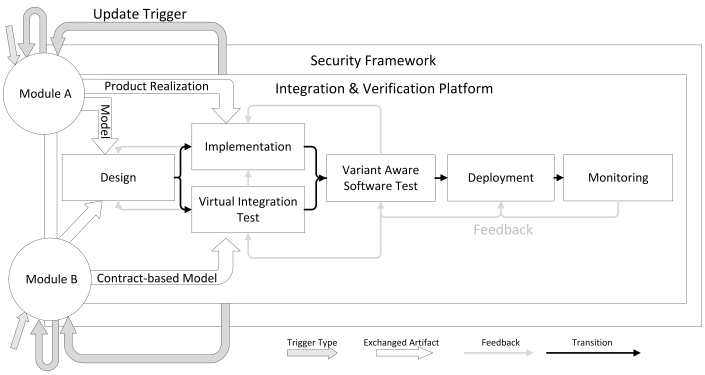

Figure 1. Overarching development, delivery, integration and verification process with overlapping phases.

The process starts with the trigger for an update. The main reasons for triggering an update are:

- **System Behavior Issues**: The system displays unwanted behavior during operation (e.g., a bug). This may occur if the development and testing of the previous product was incomplete, faulty or a change to a system peripheral changed the behavior of inputs or outputs of the system.
- **Regulatory or Requirement Change**: This type of trigger occurs if a new set of requirements have emerged due to changes in the regulatory environment of the system or because an incomplete set of requirements was specified for earlier development.
- **Feature Request**: If new functionality is required from the system, the maintainer can decide to develop an update that implements the new functionality.
- **Security Vulnerability**: Should a security vulnerability become apparent during operation, it must be closed promptly to prevent intrusion by third parties.

An update trigger can occur externally, from the module vendor, or during the integration and verification of the sub-modules. The integration process in Figure 1 takes place within the Integration and Verification Platform, where an integrator merges and tests the individual modules. The entire process is surrounded by a security framework that ensures the security of the transmission path and protects against cyber attacks.

Depending on the type of trigger, different changes to the system have to be made. The first step of the process is to analyze the system, be it using engineering knowledge or a system model or a combination of the two. During the analysis, it needs to be established if there have been requirement changes to the system and whether partitional changes to the system are required (i.e., is there functionality that is either combined, split up or distributed differently) and if it results in a change of safety considerations. The result of this analysis is then used to determine the steps that need to be taken during development of the update in order to ensure its safety.

As further depicted in the Figure 1, the system integrator can step in at different phases within the update process depending on the provided artifacts from the supplier. Unlike other development pipelines, the update process presented in Figures 1 and 2 has the intermediate stage of a VIT. A VIT can already be carried out before the final implementation of the update solely on the basis of the module contracts. The system integrator can determine on the basis the contracts and their relations defined by the model whether the module components are consistent and compatible to each other. This means that feedback can be given to the developers and planners of the module to be updated at an early stage if the planned changes do not harmonize with the existing system.

The following phase is the variant aware software test. This is where the higher-level test and simulation process takes place with numerous scenarios to rule out hazards and put the safety properties through their paces. It can be both performed solely virtual or with the actual sensors and actuators in a full system context.

The last two phases represent the deployment of the completed and released binaries in combination with the interface description based on the contracts. The final stage represents the system in operation, in which the contracts are used to continuously monitor the promised behavior and to provide direct feedback in the event of misbehavior. A continuous feedback stream (as depicted in light grey in Figure 1) is available for this purpose, which makes it possible to jump back to the necessary phase within the update verification process.

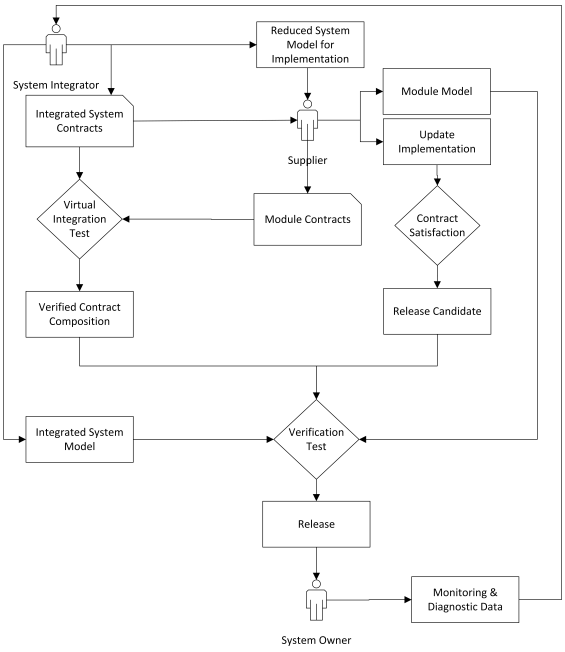

Figure 2. Communication Diagram between the different stakeholders in the process.

4. Subdivision of the Exchange Process between the Involved Stakeholders

After an update trigger has occurred or at the start of development work, the development process is mutually carried out between different groups of interest. The approach can be reduced on the assumption that 3 different stakeholder groups are driving the development and maintenance of the system under consideration: the system integrator, the suppliers and the system operator. The relationship between the stakeholders is shown in Figure 2.

The supplier When an update trigger is released, the supplier (e.g., the product vendor) can choose in which form to make his update available to the system integrator (e.g., the shipping company). In addition to the implementation of the actual update in the form of (i) a binary and the supplementary meta information, the module can be specified in form of a (ii) model and (iii) a formal behavior description. Utilizing these different forms of exchange results in time saved along the exchange process and enables a staged form of verification. The supplier can provide the system integrator with an adapted model and/or a formal specification in advance, before the actual implementation of the module has been completed.

As the system integrator requires information from the suppliers on how the module to be installed will behave, the ideal way would be that both the integrator and the supplier disclose all their information and offer complete transparency regarding his the functioning of the system.

However, often both parties do not disclose internal information in order to protect the product design. Therefore, in the interaction diagram as depicted in Figure 2, the vendors receive a reduced system model encompassing the modules requested from them. In addition to the reduced system model also the related contracts are provided to express what the supplier's modules have to achieve. Here, the concept of services and the principle of single concern, which places demands on the individual modules in terms of the required range of services, isolates the individual functions from one another and makes the modules independent of an integration platform. With the help of the reduced system model and the contracts, the supplier receives distinct instructions for his work and the tests to be performed.

Independent of the integrated system, static code analysis or software in the loop testing procedures can be used to demonstrate contract satisfaction. At the same time, the supplier also makes requirements for his individual module and can specify the

assumptions his module needs and guarantee the promised functionality. The supplier has a vested interest in the fact that his module is built into the final system. It is therefore in his own interest that his system is compatible with the overall system, so that certain efforts regarding the adaptation of the submodule to the overall system can be accepted by the supplier without having to reveal too much of his intellectual property. The supplier itself can control how much of its intellectual property to make available to the integrator. If no information about the internal implementation of the module is provided in form of a black-box model, only the behavior along the interfaces is described.

The system integrator Providing test artifacts in early phases of the update process allows the system integrator to check the integration conditions, to perform virtual integration testing (VIT) for compatibility and consistency, run a co-simulation with the models, and perform an impact analysis to determine which other modules are affected by the change. This enables incremental verification of the updated module before implementation, as depicted in Figure 2 where the VIT can be performed before the final verification test.

The role of the system integrator stands between the suppliers and the system operator and is responsible for the composition of the individual components provided by the suppliers. The system integrator can be involved in the development of sub-components as well or may be solely responsible for the integration of the individual parts into a system composition. For this, it is necessary to define the functional scope of the system in varying granularity levels and from different perspectives in a model based engineering approach.

Within the system model, the task is to combine the loose ends of the sub-modules to an integrated system. During this process inconsistencies, interferences, dead locks and dependencies between individual modules can become present. Hence, the faultlessness of the system composition to a possible approval authority needs to demonstrated in order to obtain approval for the operation of the system. For such an approval, the integrator is presented with minimum requirements and standards regarding safety and security from external sources, which his system composition must comply with. To represent these external requirements, the contracts can be utilized to reflect the desired system behavior in a comprehensible way [22].

The system integrator further provides a reduced system model with only the necessary information required for the supplier to define the scope in which the supplied module needs to function. To define the required services at its interfaces, the system integrator in the presented approach utilizes the contract's assumptions along the interfaces, while the supplier can verify whether its implemented services fulfill the defined assumptions. If, for example, the guarantees of the module contract provided by the supplier fulfill the assumptions of the system contract provided by the system integrator, the supplier's module can be integrated into the overall system. Moreover, the system integrator can verify without a complete module implementation if a working composition of its system configuration is valid, while the supplier would only be required to show that his module works sufficiently within the range of the contract.

The system operator Finally, the system operator is always interested in working with the most up-to-date, feature-rich and customized system, which also has water or road approval. Therefore, information about the current system state is provided in exchange for an adapted and updated system that is approved for the respective application. For this reason, he will generate, evaluate and report back monitoring and diagnostic data, as well as contract violations. The assumption-guarantee contracts can be used as observables and monitored, so that early co-development of contracts can also be used here for evaluation.

All in all, one can speak of a triangular relationship in which each stakeholder has an interest in the overall system retaining its approval whenever the internal composition, external circumstances or laws change but is only willing to a certain extent to disclose information for its own interests.

5. Enhancing the Traditional Development Process with Assumption-Guarantee Contracts

In order to realize the overall process, it is necessary that the stakeholders involved have a common way to define their requirements across all development phases and to be able to integrate them into a common verification process with all participants, without experiencing a disadvantage in the form of high additional effort or the disclosure of protected intellectual property.

As pointed out in the Section 3, two things are necessary to be able to describe a SoS formally and to be able to capture the extent of an update: a modular system architecture and a formal behavior description at the interfaces of the modules. Starting with the description methodology, the goal of the approach presented here is that the scope of an update can be clearly delimited. For this it is necessary to select a compositional system configuration approach, which makes it possible to arrange a total system from individual parts. In order to cope with the exponentially growing complexity during the composition process, structuring approaches such as layered design or component based design are often applied in practice. Component based design is widely used in the automotive domain (e.g., in the context of AUTOSAR [23]) as well as in the avionics domain (e.g., in the context of Aeronautical Radio, Incorporated (ARINC) [24]). In addition, model-based approaches such as the model-based development are applied independent of the domain for example with tools and languages such as SysML, AADL or Matlab Simulink. Finally, there are approaches of VIT (applied for example in Ptolemy [25] and Metropolis [26]), as well as platform-based design [27].

A minimal example of a generic compositional SoS configuration can be seen in Figure 3, with two sequential modules M_1 and M_2 connected along their interfaces with connection a, b and c as well as the associated contracts C_{M_1}, C_{M_2} and C_{SoS}.

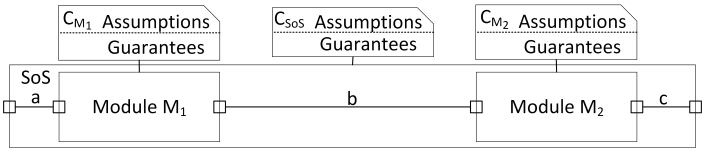

Figure 3. A reduced example of a compositional SoS with two modules connected in series and contracts attached to the parent module and the two sub-modules [22].

All of the approaches presented address the existing complexity of system compositions with abstraction and refinement in their basic concept, both vertically across different levels of abstraction and horizontally on the same level of abstraction. While layered design and model based design focus on the vertical process, component-based design on the horizontal process and platform-based design integrates both dimensions.

The use of contracts complements these existing approaches in that they make implicit assumptions between system levels and at module interfaces explicit and provide a way to formalize the behavior. Furthermore, it allows to extend the interface specification with module behavior by introducing multiple viewpoints for non-functional properties. Thus, if the products of different suppliers are to be combined in an overall system, the explicit representation of the module properties allows the clear distribution of responsibilities and verifiable integration by the system integrator [27].

An assumption-guarantee contract is described by a tuple $C = (A, G)$ in which A represents the assumption about the environment of the module M under which the guarantee G contractually assures how M will perform [27].

An exemplary contract can be seen in Figure 4. Here the contract of the GPS of an Integrated Bridge System (IBS) is enhanced and the assumptions and guarantees are exemplified. For example, an assumption in this example is the occurrence of a frame every 30 s. In combination with the other assumptions, the GPS can then guarantee the calculation of position, speed, direction or altitude within a specified timeframe.

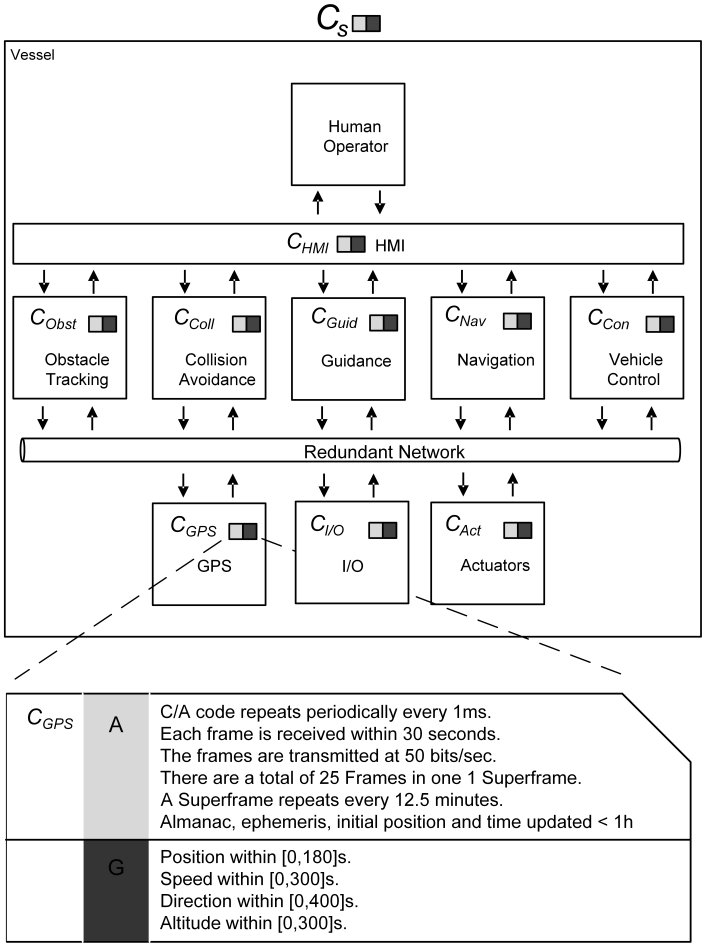

Figure 4. An exemplary modular perspective on an Integrated Bridge System enhanced with contracts and an exemplary contract for the GPS module.

To be able to represent the compositional properties of the system and the update, the symbols $\preccurlyeq$, $\wedge$ and $\otimes$ are used to represent the refinement, conjunction and compositional relationships of the modules.

For the contracts $C_{M_1} = (A_{M_1}, G_{M_1})$ and $C_{M_2} = (A_{M_2}, G_{M_2})$, depicted in Figure 3, the composition of contracts $C_{\text{SoS}} = C_{M_1} \otimes C_{M_2}$ is given by [22,27]:

$$\begin{aligned} A_{C_{M_1} \otimes C_{M_2}} &= \text{weakest} \left\{ \text{A} \;\middle|\; \begin{array}{c} A \wedge G_{M_2} \Rightarrow A_{M_1} \\ \text{and} \\ A \wedge G_{M_1} \Rightarrow A_{M_2} \end{array} \right\} \\ G_{C_{M_1} \otimes C_{M_2}} &= G_{M_1} \wedge G_{M_2} \end{aligned}$$

Thus the assumption $A_{C_{M_1} \otimes C_{M_2}}$ is defined by the weakest Assumption A for which $A \wedge G_{M_2} \Rightarrow A_{M_1}$ and $A \wedge G_{M_1} \Rightarrow A_{M_2}$ holds. Furthermore, the guarantee $G_{C_{M_1} \otimes C_{M_2}}$ is defined by the conjunction of $G_{M_1} \wedge G_{M_2}$.

By specifying the composition of a modularized system in this form, it becomes possible to check for properties, such as compatibility, refinement and abstraction. Moreover, it is possible to run test scenarios without implementing the update or the entire system, and thus check the system end-to-end solely on the contracts.

Possible Application Scenarios along the update process include thorough test cases under a broad search space (e.g., all possible situations the contracts specifications allows) in requirement-based testing, completeness verification, automatic test vector generation, automatic observer generation and applicability in Hardware in the Loop (HIL), Software

in the Loop (SIL), Model in the Loop (MIL) testing as well as in monitoring on the system in operation [27].

Furthermore, contracts do allow for an early detection of the impact of an update, to incrementally verify only the parts affected by the change [28]. Finally, the compositional characteristics can be linked to hierarchical assurance-cases, such that a (re-)certification can be facilitated [22].

Further reasons for the use of VIT during the entire lifecycle of a software module are possible cost savings in the diagnosis of functional components, the increasing modularization of functionality via containerization and virtualization and the associated service-oriented design of systems whose requirements can be converted directly into contracts. The following chapters will show how the presented properties of contracts are useful for the verification of updates and how they can be integrated into the development and integration process.

6. Phases of the Update Process

In Section 3, the general update process was presented in an abstract way from a bird's eye view. Building on this, the following section will now take an internal view of the higher-level phases. In doing so, it will be shown how the contracts are used and passed on in each of the phases in order to achieve an increasingly enriched verification.

6.1. Design

As depicted in Figure 5, the update process begins with a trigger that initiates the change and requirements management. Therefore, the existing design must be adapted to the update. Starting with the update type, the impact of the change on the existing system is analysed. A distinction is made between 3 different update types [28]: corrective, perfective and adaptive updates.

Corrective Updates: Corrective updates are understood to be the correction of errors in existing modules. In this way, it can be determined during the development time through tests, simulations and VIT, but also during operation by monitoring the observables obtained from the contracts, that misbehavior occurs which must be corrected. These can be critical errors that violate a security or a safety property, so that such a bug must be fixed as quickly as possible. Accelerated procedures such as continuous deployment or a mandatory push and install on the clients that are in operation can be used here. For corrective updates, bugs in an existing model of a module are fixed in the design phase. Furthermore, the module contracts already exist and must also be renewed. However, it is often the case that the contracts do not have to be adapted further, as the framework conditions defined by the contracts do not have to be affected by the correction of the error.

Perfective Update: A perfective update optimizes an existing functionality. In the case of an existing module, the aim of the update is therefore to improve performance, but also to further expand the reliability of the module. Here, too, permanent monitoring can be used to determine whether and under what circumstances modules are not delivering their full performance or are at their limits. Based on the analysis results, countermeasures can then be taken. For example, it is possible to make the performance of a module more efficient by replacing an underlying algorithm, thus increasing the computation speed. Equivalent to corrective updates, in most cases this only changes the internal implementation of the module, but leaves the assumptions and guarantees of the module contracts largely untouched, since the limit ranges in which the module functions have not changed.

Adaptive Update: In an adaptive update, a functionality is introduced into the existing system. The system is thus extended. This means that it must be determined how the newly introduced module or the new function of a module fits into the overall system. Thus, an impact analysis is carried out first to determine which dependencies exist and whether chain reactions or inconsistencies occur. In addition, it is determined which requirements the existing system has for the new function (e.g., which existing modules access the functionality) and which conditions the module has for the system (e.g., which

resources does the new function require). Once the impact analysis has been performed and the requirements identified, the models and contracts are derived, which in turn are passed on to the VIT step and the implementation phase.

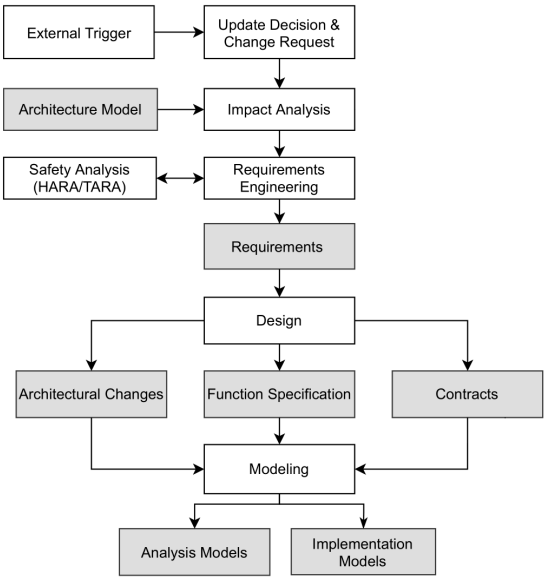

Figure 5. Design Phase of the Update Development Process.

The identified impact of each of the three update types then results in requirements for the update and the existing system. These can then be transferred into the model based design.

There the design is then divided again into the architectural changes to the previous system, the functional specification of the update and the contracts. All three areas flow into the model to be updated. The results are the analysis models for abstract verification without actual realization of the module, as well as implementation models that can then be transferred to implementation and configuration management.

The developed process does not specify a model-based design to which to adhere when implementing changes that have been designed into a system. For the development of a functional system model, a modeling framework such as MATLAB/Simulink or Ptolemy can be applied. Functionality should be modeled in close conjunction with the architecture. The systems environment can be modeled according to the specified requirements of the system. This ensures early model verification and reduces failure rates in later test stages. These functional models should be used for machine code generation of the update package.

6.2. Virtual Integration Test

Introducing a new module or changing an existing functionality can lead to chain reactions within the overall system. Individual modules are used by several functions, so that dependencies and inconsistencies can occur that have not been considered. Even in medium-sized models, dependencies between individual modules can no longer be identified without a great deal of search effort and with certainty of completeness. At this point the formal behavioral descriptions can be utilized to identify dependencies between the module units and to check the interaction for correctness (as depicted in Figure 6). This is where the second phase of the presented framework comes into play. The extended model with formal contracts makes the temporal and functional processes and guarantees evaluable in a virtual integration test.

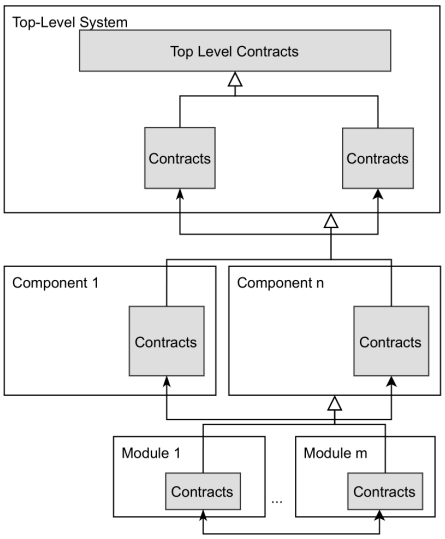

Figure 6. Compositional Verification based on a Virtual Integration Test.

In the generic example in Figure 3, the contractually described composition of the submodules is described as $C_{M_1} \otimes C_{M_2} \preccurlyeq C_{\text{SoS}}$. This generic case can be used to show which integration conditions must apply to an update.

The integration $C_{M_1} \otimes C_{M_2} \preccurlyeq C_{\text{SoS}}$ for the contracts $C_{\text{SoS}} = (A, G)$, $C_{M_1} = (A_{M_1}, G_{M_1})$ and $C_{M_2} = (A_{M_2}, G_{M_2})$ holds if and only if $A \wedge G_{M_1} \Rightarrow A_{M_2}$ and $A \wedge G_{M_2} \Rightarrow A_{M_1}$ and $G_{M_1} \wedge G_{M_2} \Rightarrow G$.

An update may result in a broken system composition if the independent implementability property is violated, which is defined as follows:

For all contracts C_1, C_2, $C_1^{'}$ and $C_2^{'}$, if C_1 is compatible with C_2 and $C_1^{'} \preccurlyeq C_1$ and $C_2^{'} \preccurlyeq C_2$ hold, then $C_1^{'}$ is compatible with $C_2^{'}$ and $C_1^{'} \otimes C_2^{'} \preccurlyeq C_1 \otimes C_2$.

A contract in this context is compatible if and only if $A \neq \emptyset$. Furthermore, two contracts are compatible if $C_1 \otimes C_2$ is fully described and compatible.

From this it can be deduced that the contractual compatibility of the modules is violated if (i) $C_1^{'}$ is not compatible with C_2, (ii) does not refine C_1 or (iii) the contract becomes inconsistent. This is the case when the guarantee of a contract remains unfulfilled at all times.

In the virtual integration test, the introduction of the update is thus checked to see whether it can be inserted into the overall network. This also makes it possible to check the completeness and finiteness of the interactions. From the virtual integration test, a supplementary release for the plans to update the module can be given in advance.

6.3. Implementation

The development phase builds on the implementation models and contracts. Since the contracts have already been checked for correctness in a previous phase, they now serve in the development phase as a benchmark for the contractually assured functional scope of the implementation and enable the implementation to be checked by means of SIL tests based on the assumptions and guarantees set.

The development phase otherwise follows standard best practices as depicted in Figure 7. An iterative approach can be chosen as well as agile methods (e.g., Continuous Deployment, DevOps).

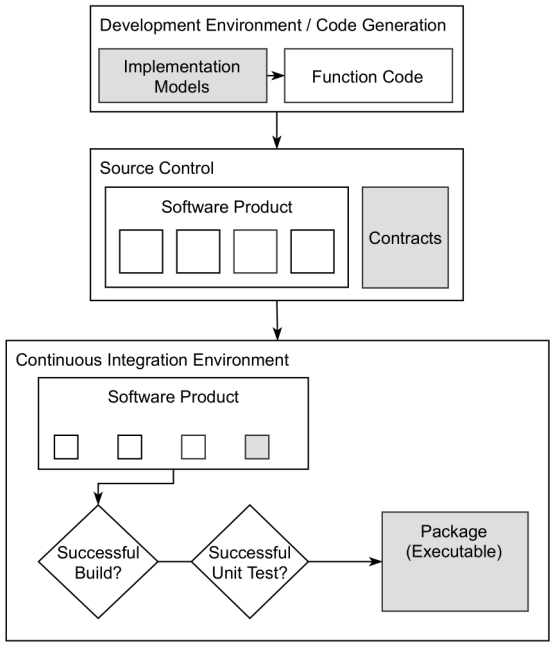

Figure 7. Implementation Pipeline for the Module Update.

Furthermore, the Development of code should follow strict standards and tools for code verification such as linting and static code analysis. The implementation models are then followed by functional code, which is transferred to a source control system. In addition to the software product consisting of the module units, this also holds the contracts, which are transferred as metadata.

Finally, a version management system is used in conjunction with an automated build and test environment. This ensures correct versioning, early builds and can support the link between requirements, contracts and code. Contracts can directly be used as satisfaction conditions during automated module tests. As a side effect, this produces traceability of changes throughout the development process. After successful build and unit tests, the update with the contract metadata is packaged for the test and deployment phases.

6.4. Variant Aware Software Test

Since CPS can display a high degree of variability, it is key to ensuring safety, that possible combinations of functionality are tested before deployment. Within the proposed process, verification is handled on different levels and with different goals:

- **Module Testing:** Single software modules are tested against their contracts during the implementation Phase.
- **VIT:** as described in Section 6.2.
- **Variant Aware Software Test:** Different variants of the system are tested dynamically as digital twins before deployment.
- **System in the Loop Tests:** Test the overall system for an applicable scenario space, both virtual and with the actual sensors and actuators.

For the test case generation, the test data, the contracts, requirements and variant configurations derived from the previous phases serve as an input. The tests include system tests, package tests and module tests. The variant aware software test phase, as depicted in Figure 8, assumes that the system including all its variants has been modeled in a model based design environment. This means, that all possible artifacts that can be installed in or on a system have two digital representations. On the one hand, as an architectural artifact which has assigned all requirements, contracts and interfaces and on the other hand as a functional model, either as machine code or as a functional component in a language such

as MATLAB/Simulink. With the use of a variant model, the artifacts of the architectural model can be selected and reduced to possible systems [29].

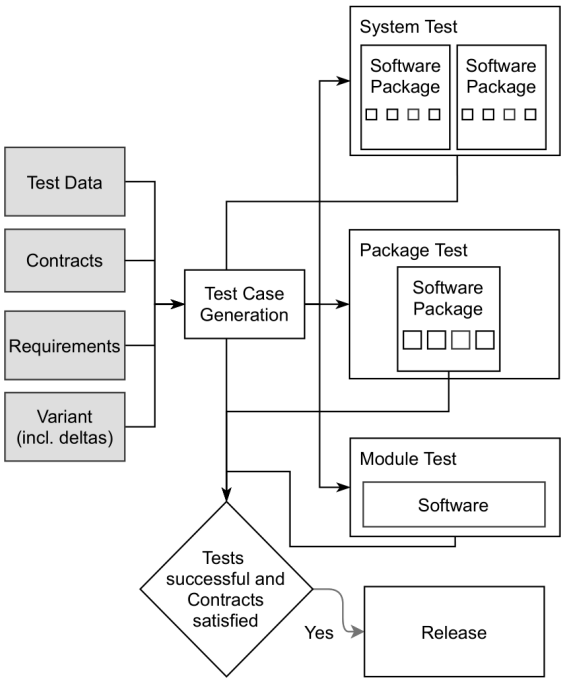

Figure 8. Variant Aware Software Test before Release of the Update.

For example, a system might have either an RGB-Camera or a B/W-Camera. Thus, there are two possible variants of the system, that need to be tested separately. Since the composition of modules may have hundreds to thousands configuration points, the combinatorics can increase the search space intensely. Therefore, variants are chosen, that are representative for a range of possible combinations. These variant representative models then can be used to generate a digital twin, incorporating the functional models. The digital twin is then tested in a simulation environment. Multiple representative variants are chosen for a system, ensuring the coverage of the complete possible configuration space. Specific variants can then be marked as eligible for a certain update, if the test for the configuration at hand has been passed and VIT was successful.

The variant-based software test can be performed in isolation from the physical sensors and actuators by simulating the inputs and outputs of the SoS. Thus, the scenario space can be covered completely or randomized. Furthermore, a System in the Loop approach is possible, which integrates the physical system components.

When all test are successful and satisfy the requirements of the contracts the release can be approved.

6.5. Deployment of the Update

If an update has been verified, it can be marked as released and is bundled into a deployable (see Figure 9). A deployable contains one or multiple executables, their respective contracts and an identification for the variants it may be deployed on.

The deployment can be fulfilled in two different ways:

A priori: If an update for a system variant has been developed, tested and released, it is bundled and published to a server that on request by a client with the specified configuration sends the appropriate update to the client.

On the fly: If a certain system requests an update, the configuration of the system can be compared against the possible updates and their corresponding variants and a bundle is then packaged by the server, taking the released updates and their contracts and sending them to the system.

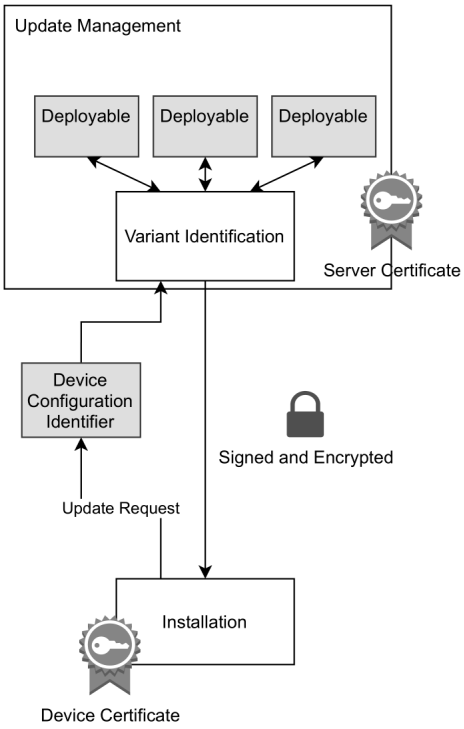

Figure 9. Signed and Encrypted Deployment of the Update.

The update package can now be transferred to the system in two different ways. Either a transfer to the target client is forced via push, so that it is ensured that the update is actually transferred, or it is requested by the client via pull request. In this case, the client transmits a device configuration identifier that tells the server which system configuration is involved, so that a suitable update package can be assigned.

The exchange is then secured by signing the client, server and the package, and the server and client identify themselves to each other by means of certificates. The signed and encrypted package is then transferred via a secure transmission to the target system, where the authenticity of the package can be verified and installed on the target system using an existing package management system.

The former is advisable in case of small fleets with high variability between each system, while the latter is a more generalized approach that scales better with big fleets in which the variability is high over the whole fleet but small between individual systems.

An existing implementation for providing the update packages between different participants in the maritime domain could is the maritime connectivity platform, as it provides a Maritime Identity Registry (MIR) for secure identification of the participants and a Maritime Service Registry (MSR) for service discovery [30].

6.6. Monitoring during Operation

In the last phase, the update is on the vehicle and is put into operation. This does not necessarily have to take place immediately after transmission and installation, as the update package can also be activated later. For example, if a function is to be introduced in time but already transferred to the target system, or if a function is already to be made available on the client and is activated by an activation mechanism, the activation of the update can be deferred.

In addition, a further stage is possible in that the module is activated but not yet live. That is, the calculations and data collection are already active, but the functionality is not yet made available to the other modules and the overall system. In this way, the module can be tested for real environmental data and its behavior under real scenarios before final activation of its functionality. In this way, any unforeseen behavior that was not considered in the previous phases can be uncovered or tested for real-world scenarios that were not

considered beforehand. This approach is particularly effective with a large active fleet. For example, the automotive manufacturer Tesla already uses an 'inert feature' in its fleet to test new features based on thousands of real-world routes before activating it [31].

The contracts supplied by means of metadata now serve as the basis for observables in this phase, defining the boundary conditions of module functionality by means of formal assumptions and guarantees. This sets a framework that can be monitored by means of an additional monitoring component (see Figure 10). It is important that the monitoring component can automatically start the monitoring of the update based on the contracts without the need to customize it itself or to have the module provide functionality for the monitor, as this cannot be expected from the suppliers. One way to realize this is to track the process via a central logging system. This way, the results of the respective functions can be time-stamped and continuously monitored. An example is provided by the ISO standard ISO/DIS 24060(en), which is currently under development and requires a Ship Software Logging System (SSLS) for software on board of a ship. In order to provide compliant information from the module to the SSLS the standards suggests "a data sentence message structure defined as the VER specified in IEC 61162-1 and sent by equipment implementing this standard. (encapsulated in 450 protocols with UDP datagrams)" [8].

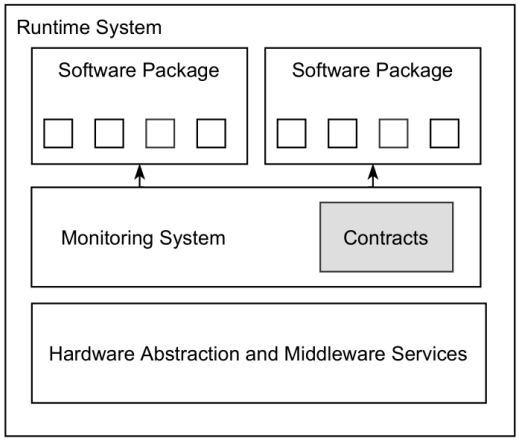

Figure 10. Monitoring of Updated Module During Runtime.

The monitoring function can then monitor the communication flow between the components and by means of the logging system and determine from the contracts if there is a breach. In the event of a malfunction, this can be fed back directly to the system integrator and the relevant supplier with the addition of the logbook and contracts, so that they can promptly initiate a new update process.

7. Evaluation of an Adaptive Update to a Maritime Collision Avoidance System

In the following, the presented process is demonstrated on a maritime collision avoidance system. The functionality of the system is reduced to the core-components depicted in Figure 11: A sensor (here the *radar module*) as input, the *collision avoidance module* providing the calculation and a *traffic display* (here an ECDIS) visualizing the output of the collision avoidance system. In this reduced system design, the ECDIS, in addition to the nautical chart display, object-related information and sensor data, provides the navigator with a visual alarm in case of critical approach, as well as a maneuver suggestion. The *collision avoidance module* is further subdivided into three sub-modules, with two already implemented (*traffic analysis, collision detection*) and one to be integrated via update (*conflict resolution*).

The continuously running *traffic analysis* module is providing the *collision detection module* a constant stream of the surrounding traffic. The *collision detection module* detects upcoming critical situations and calculates the CPA and DCPA. The *conflict resolution module* calculates a maneuver suggestion based on the situation-dependent COLREGs,

the detected traffic situation and assumed maneuvers of the other ships. The information is provided to the traffic display and the captain sees an alarm as well as a maneuver suggestion to avoid the upcoming collision.

The *traffic analysis* and *collision detection* alone can already provide an alarm for the captain of the ship, to manually decide for a maneuver. However, integrating a new functionality for automatic conflict resolution and maneuver suggestion further supports the seafarer in its decision and reduces the possibility for human errors in a critical situation.

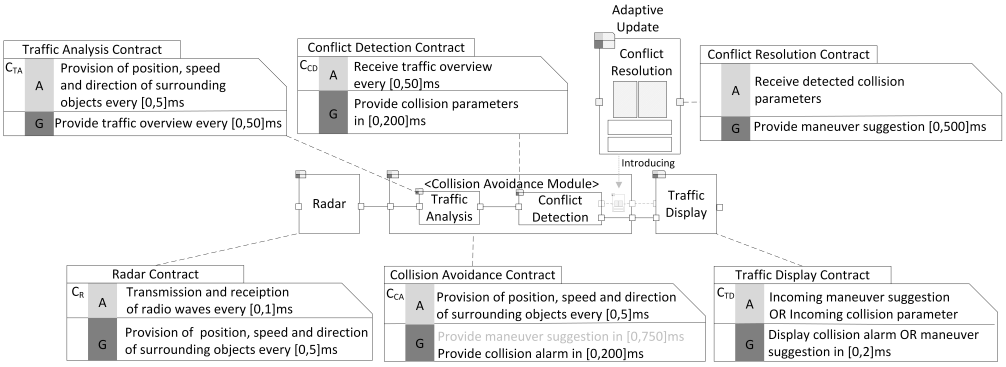

Figure 11. A collision avoidance system updated with a *conflict resolution module* as new functionality.

For each module of the SoS presented here, a contract was created that describes the behavior of the respective module at its interfaces. Of these, the contracts for the *radar* C_R, for the *collision avoidance module* C_{CA} and the *traffic display* C_{TD} describe the top-level behavior of the respective modules. One level lower, the *traffic analysis module* contract C_{TA}, the *conflict detection module* contract C_{CD}, and the *conflict resolution module* contract C_{CR} introduced by the update, refine the behavior of the *conflict avoidance parent module*. The *radar* and *traffic display modules* are not refined further.

For the purpose of demonstrating the procedure, we focus on the time constraints that the respective modules and sub-modules have to fulfill and on which we will show how the composition can be verified by an update along the overall process.

Specifically, the focus can be placed on the *collision avoidance module*. The previous *radar module* is not affected by the update. Thus, on the input side of the *collision avoidance module*, the assumption is satisfied by the output of the *radar* and does not change as a result of the update. On the output side, however, a new interface description for the *maneuver suggestion* must be introduced. For this purpose, a new guarantee is added to the existing contract of the *collision avoidance module*. Furthermore, the new timing specifications of the introduced functionality have to be declared in the guarantee of the *collision avoidance module* and the assumptions and guarantees of the *traffic display*. The procedure will be demonstrated in the following.

Design: The need to introduce a new function into the existing system triggers a new update process. After the impact analysis has been performed against the architecture model and the requirements and security analysis have been run, the system is redesigned against the existing system model. In the process, architectural, functional and contractual dimensions are redesigned. As shown in Figure 11, the first step in the design phase is to add the new module to the model. In addition, the contracts that the new module has to fulfill are generated.

There are two major changes in existing system model. First, the module is sequentially connected after the *conflict detection module*, since it depends on its results. Second, additional connections are introduced (shown in light gray dashed lines in Figure 11), so that in addition to the direct passing of the conflict parameters to the *traffic display*, a connection through the *conflict resolution module* to the *traffic display* can also be established.

The conflict resolution contract has the assumption *Receive detected collision parameters*. It is equal to its predecessor module's guarantee but without time restrictions. The reason

to omit the timing in the assumption lies in the randomness of the occurrence of a collision. Only when a collision scenario is present a conflict is detected and a manuever needs to be provided.

The result of the design phase is a revised component model, which has been extended by the new module and introduces a new contract. This new system composition is now verified in the following phases.

Virtual Integration Testing and Implementation: As shown in Section 6.2, the following composition must already apply to the existing *collision avoidance module* $C_{TA} \otimes C_{CD} \preccurlyeq C_{CA}$.

Due to the introduction of the new module C_{CR}, the following must now hold for the module after the update $C_{TA} \otimes C_{CD} \otimes C_{CR} \preccurlyeq C_{CA}$.

However, this refinement condition is not satisfied here, since there is no guarantee for the maneuver proposal so far, as this feature is only introduced by the update and it has to be provided with a time condition. Since the contract C_{CR} states that a maneuver proposal can be generated within $[0, 500]$ ms, but for this it relies on the previous calculation results from the *traffic analysis module* and *conflict detection module*, the composition must therefore include a guarantee for the sequential calculation of the final result. The guarantees of the C_{TA} and C_{CD} contracts guarantee their results in $[0, 50]$ ms and $[0, 200]$ ms, respectively. Thus, since the modules run sequentially, it takes 750 ms in the worst case to generate after a maneuver proposal in the event of a conflict. However, since the contract C_{CA} has no such guarantee so far, the guarantee: *'Provide a maneuver proposal in [0,750]'* ms is introduced.

Accordingly, the composition is fulfilled, since the sequence of guarantees along the interfaces is compatible and the composition refines the enclosing contract. Finally, the contract C_{TD} of the *traffic display* must be supplemented by the assumption that a maneuver proposal arrives and that a visualization can be prepared for the captain within a set time period based on the available data.

Variant Aware Software Test: The software test phase can now build on the results of the VIT, the requirements analysis and the model-based design. Here, the different granularity levels of the system presented in Section 6.4 can be tested. At the smallest level, the modules are tested both at *level 0*: *radar*, *collision avoidance* and the *traffic display*, as well as at *level 1*: *traffic analysis*, *collision detection*, *collision resolution* individually. In addition to traditional software tests, SIL and MIL tests at the module level, it can be verified that the actual realized modules meet the timing requirements of their contracts by staying within the specified time limits simulative or in real scenarios. This is then repeated in the composition at *level 0* for the *collision avoidance module*. Since it is already known from the VIT that the composition formula is correct, all that remains in this phase is to check that the actual implementation satisfies the promised guarantees under all circumstances. Once all tests have been passed, the module update can be released.

Deployment and Operation: The modeled, contractually secured, tested and packed module can now be released via a push or pull update. As already described in Section 6.5, the package is signed and transferred to the client via an encrypted connection.

On the client the package is checked and unpacked. The update binary of the *collision avoidance module* is then installed and activated or kept inactive. In addition to the installed module, a monitoring system exists on which the existing observables of the updated contracts C_{CA} and C_{TD} are updated. In addition, the contract C_{CD} is integrated as an observable. The monitoring system on the client can then continuously measure the behavior of the module interfaces of the *collision avoidance module* and detect breaks in the contractual behavior between input and output. For example, should a conflict be identified at *level 0* via the *collision avoidance module* within 250ms, a manuever suggestion must be provided at the output interface of the *collision avoidance module* not later than 500ms after the upcoming collision has been detected. If a maneuver suggestion is not given in time, a feedback can be made to the system owner, the operator and/or supplier and the process can be triggered again.

8. Conclusions and Outlook

The present work has mapped a continuous verification framework based on module contracts along the update process of maritime systems. For this purpose, a framework was presented that depicts all phases of the development process including feedbacks. Furthermore, the interactions between the involved parties were shown and the exchange between the stakeholders was demonstrated. For each phase, it was then shown how contracts can be used for verification and how the transitions between individual phases should be designed. Finally, the entire process was demonstrated on an adaptive update for a maritime collision avoidance system.

In conclusion, the process presented represents a first approach to enable measurability analysis along the update chain. The use of contracts enables formal assurance, testing, and monitoring capabilities that will be necessary as maritime systems evolve toward autonomous systems. It is also a useful complement to simulative approaches such as the OSP, which do not provide information about deterministic behavior.

The approach, however, remains dependent on the implementation of the contracts by the involved stakeholders. In practice, it is often the case that descriptions of the subsystems are only provided in the form of prose and are not formally transferred. However, since a worst-case approach can be used here, the additional effort is justifiable. A transfer of the interface description into contracts can also be assumed to be reasonable for the stakeholders in the process.

As an outlook for future work, therefore, hybrid approaches of the process are conceivable, in that assumptions must be made about modules that are not contractually described or are brought together with software modules that are not defined.

Furthermore, an update process enriched by contracts on the basis of modular development approaches can also be used to link a modular assurance case, so that an update also leads to the required security proofs being updated on the corresponding certification side. In addition, rollbacks and degraded modes can be described and preventively incorporated into the system, so that in the event of a downgrading of the system functionality, the system can be transferred to another state in a controlled manner.

In summary, the approach presented forms a basis to which existing methods can be incorporated and which simultaneously supplements existing approaches into an integrative verification approach for updates based on contracts.

Author Contributions: Conceptualization, methodology, validation, formal analysis, investigation, resources, data curation and writing: G.H.; C.P.H.; review, editing, supervision: A.H.; All authors have read and agreed to the published version of the manuscript.

Funding: This work has been funded by the German Federal Ministry of Education and Research (BMBF) in the project Step-Up!CPS (Funding reference number: 01IS18080D). The responsibility for the content remains with the authors.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Request to the corresponding author of this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Barbier, M.; Bensana, E.; Pucel, X. A generic and modular architecture for maritime autonomous vehicles. In Proceedings of the 2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV), Porto, Portugal, 6–9 November 2018; pp. 1–6, ISSN: 2377-6536. [CrossRef]
2. Karlsen, A. D-Class Data smart Classification. In Proceedings of the Dynamic Positioning Conference, Houston, TX, USA, 9–10 October 2018.
3. Bauwens, J.; Ruckebusch, P.; Giannoulis, S.; Moerman, I.; Poorter, E.D. Over-the-Air Software Updates in the Internet of Things: An Overview of Key Principles. *IEEE Commun. Mag.* **2020**, *58*, 35–41. [CrossRef]

4. CIRM/BIMCO Joint Working Group. Industry Standard on Software Maintenance of Shipboard Equipment v1.0. 2017. Available online: https://www.cirm.org/publications/industry_standards/IndustryStandardonSoftwareMaintenanceofShipboardEquipmentv1-0.pdf (accessed on 23 June 2021).
5. Ibrion, M.; Paltrinieri, N.; Nejad, A.R. Learning from Failures in Cruise Ship Industry: The Blackout of Viking Sky in Hustadvika, Norway. *Eng. Fail. Anal.* **2021**, *125*, 105355. [CrossRef]
6. Degani, A. The Grounding of the Royal Majesty. In *Taming HAL: Designing Interfaces Beyond 2001*; Degani, A., Ed.; Palgrave Macmillan US: New York, NY, USA, 2003; pp. 100–120. [CrossRef]
7. IACS. *Rec. No.153—Recommended Procedures for Software Maintenance of Computer Based Systems on Board*; Technical Report Rec 53; IACS: London, UK, 2018.
8. ISO. *Ships and Marine Technology—Software Maintenance Requirements of Shipboard Equipment*; ISO DIS 24060(En); International Organization for Standardization: Geneva, Switzerland, 2021.
9. Pedersen, T.A.; Glomsrud, J.A.; Ruud, E.L.; Simonsen, A.; Sandrib, J.; Eriksen, B.O.H. Towards simulation-based verification of autonomous navigation systems. *Saf. Sci.* **2020**, *129*, 104799. [CrossRef]
10. Tripakis, S. Bridging the semantic gap between heterogeneous modeling formalisms and FMI. In Proceedings of the 2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), Samos, Greece, 19–23 July 2015; pp. 60–69. [CrossRef]
11. Wilde, T.; Hess, T. Forschungsmethoden der Wirtschaftsinformatik. *Wirtschaftsinformatik* **2007**, *49*, 280–287. [CrossRef]
12. Perabo, F.; Park, D.; Zadeh, M.K.; Smogeli, O.; Jamt, L. Digital Twin Modelling of Ship Power and Propulsion Systems: Application of the Open Simulation Platform (OSP). In Proceedings of the 2020 IEEE 29th International Symposium on Industrial Electronics (ISIE), Delft, The Netherlands, 17–29 June 2020; pp. 1265–1270, ISSN: 2163-5145. [CrossRef]
13. Thomas, D.; O'Malley, S. The Necessity of Standards for Maritime Informatics in Ship Operations. In *Maritime Informatics*; Lind, M., Michaelides, M., Ward, R., Watson, T.R., Eds.; Progress in IS; Springer International Publishing: Cham, Switzerland, 2021; pp. 33–45.
14. DEIS Project. Digital Dependability Identities and the Open Dependability Exchange Meta-Model. 2019. Available online: https://deis-project.eu/fileadmin/user_upload/DEIS_D3.1_Specification_of_the_ODE_metamodel_and_documentation_of_the_fundamental_concept_of_DDI_PU.pdf (accessed on 23 June 2021).
15. Structured Assurance Case Metamodel (SACM), v2.1. p. 90. Available online: https://www.omg.org/spec/SACM/2.1/About-SACM/ (accessed on 23 June 2021).
16. Reich, J.; Schneider, D.; Sorokos, I.; Papadopoulos, Y.; Kelly, T.; Wei, R.; Armengaud, E.; Kaypmaz, C. Engineering of Runtime Safety Monitors for Cyber-Physical Systems with Digital Dependability Identities. In *Computer Safety, Reliability, and Security*; Casimiro, A., Ortmeier, F., Bitsch, F., Ferreira, P., Eds.; Springer International Publishing: Cham, Switzerland, 2020; Volume 12234, pp. 3–17.
17. Ihirwe, F.; Mazzini, S.; Pierini, P.; Debiasi, A.; Tonetta, S. Model-Based Analysis Support for Dependable Complex Systems in CHESS. *arXiv* **2020**, arXiv:2009.06089.
18. Warg, F.; Blom, H.; Borg, J.; Johansson, R. Continuous Deployment for Dependable Systems with Continuous Assurance Cases. In Proceedings of the 2019 IEEE International Symposium on Software Reliability Engineering, WoSoCer Workshop, Berlin, Germany, 27–30 October 2019; IEEE Computer Society: Washington, DC, USA, 2019.
19. Vanherpen, I.K. A Contract-Based Approach for Multi-Viewpoint Consistency in the Concurrent Design of Cyber-Physical Systems. Ph.D. Thesis, University of Antwerp, Antwerpen, Belgium, 2018; p. 170.
20. Nuzzo, P.; Sangiovanni-Vincentelli, A.L.; Bresolin, D.; Geretti, L.; Villa, T. A Platform-Based Design Methodology With Contracts and Related Tools for the Design of Cyber-Physical Systems. *Proc. IEEE* **2015**, *103*, 2104–2132. [CrossRef]
21. Neukirchner, M.; Stein, S.; Schrom, H.; Ernst, R. A software update service with self-protection capabilities. In Proceedings of the 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010), Dresden, Germany, 8–12 March 2010; pp. 903–908. [CrossRef]
22. Hake, G.; Feuerstack, S.; Hahn, A. Towards Recertification of Modular Updates in Integrated Maritime Systems of Systems. In *Computer Safety, Reliability, and Security*; Casimiro, A., Ortmeier, F., Bitsch, F., Ferreira, P., Eds.; Springer International Publishing: Cham, Switzerland, 2020; Volume 12234, pp. 50–63. [CrossRef]
23. Fürst, S.; Bechter, M. AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform. In Proceedings of the 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W), Toulouse, France, 28 June–1 July 2016; pp. 215–217. [CrossRef]
24. Delange, J.; Pautet, L.; Plantec, A.; Kerboeuf, M.; Singhoff, F.; Kordon, F. Validate, simulate, and implement ARINC653 systems using the AADL. *ACM SIGAda Ada Lett.* **2009**, *29*, 31–44. [CrossRef]
25. Cardoso, J.; Siron, P. Ptolemy-HLA: A Cyber-Physical System Distributed Simulation Framework. In *Principles of Modeling*; Lohstroh, M., Derler, P., Sirjani, M., Eds.; Series Title: Lecture Notes in Computer Science; Springer International Publishing: Cham, Switzeland, 2018; Volume 10760, pp. 122–142. [CrossRef]
26. Balarin, F.; Watanabe, Y.; Hsieh, H.; Lavagno, L.; Passerone, C.; Sangiovanni-Vincentelli, A. Metropolis: An integrated electronic system design environment. *Computer* **2003**, *36*, 45–52. [CrossRef]
27. Benveniste, A.; Caillaud, B.; Nickovic, D.; Passerone, R.; Raclet, J.B.; Reinkemeier, P.; Sangiovanni-Vincentelli, A.; Damm, W.; Henzinger, T.A.; Larsen, K.G. Contracts for System Design. *Found. Trends Electron. Des. Autom.* **2018**, *12*, 124–400. [CrossRef]

28. Bebawy, Y.; Guissouma, H.; Vander Maelen, S.; Kröger, J.; Hake, G.; Stierand, I.; Fränzle, M.; Sax, E.; Hahn, A. Incremental Contract-based Verification of Software Updates for Safety-Critical Cyber-Physical Systems. In Proceedings of the 2020 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 16–18 December 2020.
29. Guissouma, H.; Lauber, A.; Mkadem, A.; Sax, E. Virtual Test Environment for Efficient Verification of Software Updates for Variant-Rich Automotive Systems. In Proceedings of the 2019 IEEE International Systems Conference (SysCon), Orlando, FL, USA, 8–11 April 2019; pp. 1–8.
30. Weinert, B.; Park, J.; Christensen, T.; Hahn, A. *A Common Maritime Infrastructure for Communication and Information Exchange*; IALA: St Germain en Laye, France, 2018.
31. Ross, P.E. Tesla Reveals Its Crowdsourced Autopilot Data—IEEE Spectrum. 2016. Available online: https://spectrum.ieee.org/cars-that-think/transportation/self-driving/tesla-reveals-its-crowdsourced-autopilot-data (accessed on 15 June 2021).