Developing a Framework for Trustworthy Autonomous Maritime Systems

Dana Dghaym[1,*], Stephen Turnock[2], Michael Butler[1], Jon Downes[2], Thai Son Hoang[1] and Ben Pritchard[3]

[1] {d.dghaym t.s.hoang, mjb}@ecs.soton.ac.uk (Electronics and Computer Science, University of Southampton, UK)

[2] {s.r.turnock, j.j.downes}@soton.ac.uk (Maritime Robotics Lab, University of Southampton, UK)

[3] ben.pritchard@uk.thalesgroup.com (Thales, UK)

ABSTRACT

A key risk with autonomous systems (AS) is the trustworthiness of the decision-making and control mechanisms that replace human control. To be trustworthy, systems need to remain safe while being resilient to unpredictable changes, functional/operational failures and cybersecurity threats. Rigorous validation (does the solution satisfy the stakeholders' requirements and system's needs?) and verification (is the system free from errors?) are essential to ensure trustworthiness of AS. Current engineering practice relies heavily on Verification and Validation (V&V) test-and-fix of system characteristics which is very time-consuming and expensive, limiting the possibilities for exploration of alternatives in system design.

We present an approach to identifying and analysing mission requirements for squads of autonomous missions. Clear definition of requirements is an important pre-requisite for mission planning and for V&V of mission management. We use a structured approach to requirements identification and use formal modelling to help remove ambiguities in the requirements and to specify formal properties that should be satisfied by the missions. Our approach is being evaluated through consideration of a combined mission of the commercial C-Cat3 Unmanned Surface Vehicle (USV) (ASV Global, 2019) with deployment /recovery of small Unmanned Underwater Vehicles (UUV) within a shipping channel whereby the USV has to safely maintain station for a long period and then proceed to recover the UUV, while maintaining a communication link to an Unmanned Aerial Vehicle (UAV).

Keywords: Formal Methods; Event-B; Requirements; Maritime Autonomous Systems.

1. INTRODUCTION

Autonomous systems offer the potential of reducing the cost and ensuring the safety of humans. However, managing a squad of heterogeneous autonomous systems can be costly requiring a large number of people to complete a mission. This paper will focus on the early phases of designing an integrated mission management system for heterogenous autonomous assets, which we call the Integrated Mission Management System (IMMS). The aim of this system is to reduce the cost of missions requiring multiple platforms.

Mission management involves the following activities: planning of a mission after identifying the mission goals, mission execution and reviewing the mission. While we have trustworthy autonomous vehicles working as separate entities, our aim is to build a trustworthy management system to ensure the trustworthiness of the overall system.

* d.dghaym@ecs.soton.ac.uk

Studies have shown that the cost of fixing errors during testing is 10 times more than during the construction phase and can increase to more than 25 times post release (Leffingwell, 1997), and many problems discovered in software systems are related to shortcomings in requirements elicitation and specification processes (MacDonell et al. , 2014). In this paper, we will show how we can apply formal modelling to develop a requirements analysis framework for identification of anticipated range of operational environments for autonomous missions, including human operator interactions, together with precise specification of safety and security envelopes for enactment of autonomous missions.

This paper is organised as follows: Section 2 presents some background information about Event-B formal method. Section 3 gives an overview of the approach followed in identifying and analysing the mission requirements. Section 4 describes how we apply formal modelling to identify the system requirements. Finally we present our conclusions in Section 5.

2. BACKGROUND

In this section, we present Event-B, a formal method for system analysis and modelling. We have chosen Event-B because Event-B supports modelling at a system level rather than only at a software level. Event-B also has a good extensible tool support and a user can apply both theorem proving and model checking, supported by ProB (Leuschel & Butler, 2008), to the same model. A survey of formal verification tools have found that Event-B supported by the toolset Rodin comes closest to supporting the goals of a *correct-by-construction* designs (Armstrong et al. , 2014). In this paper, we use Event-B to address the ambiguity and inaccuracy of requirements specifications.

2.1 Event-B

Event-B (Abrial, 2010) is a formal method for system development. One of the main features of Event-B is the use of ***refinement*** to introduce system details gradually into the formal model. An Event-B model consists of two parts: ***contexts*** and ***machines***. Contexts are the static parts of the model. A Context contains ***carrier sets***, ***constants***, and ***axioms*** that constrain the carrier sets and constants. Machines are the dynamic parts of the model. A machine contains ***variables*** v, ***invariants*** I(v) that constrain the variables, and ***events***. An event comprises a guard denoting its enabling-condition and an action describing how the variables are modified when the event is executed. In general, an event e has the following form, where t are the event parameters, G(t, v) is the guard of the event, and v := E(t, v) is the action of the event.

e == **any** t **where** G(t,v) **then** v := E(t,v) **end**

A machine in Event-B corresponds to a transition system where ***variables*** represent the states and ***events*** specify the transitions. Contexts can be ***extended*** by adding new carrier sets, constants, axioms, and theorems. Machine M can be ***refined*** by machine N (we call M the abstract machine and N the concrete machine). The state of M and N are related by a gluing invariant J(v, w) where v, w are variables of M and N, respectively. Intuitively, any “behaviour” exhibited by N can be simulated by M, with respect to the gluing invariant J. Refinement in Event-B is reasoned event-wise. Consider an abstract event e and the corresponding concrete event f. Somewhat simplifying, we say that e is refined by f if f's guard is stronger than that of e and f's action can be simulated by e's action, taking into account the gluing invariant J. More information about Event-B can be found in (Hoang, 2013). Event-B is supported by Rodin (Abrial et al., 2010), an extensible toolkit which includes facilities for modelling, verifying the consistency of models using theorem proving and model checking techniques, and validating models with simulation-based approaches.

3. AN APPROACH FOR REQUIREMENTS ELICITATION

Defining the requirements of the IMMS is an iterative process, where in the initial version we focus on ***what we know***. We start by gathering information about the different available autonomous

platforms. In this case, we have three different physical assets from each of the three domains: surface, underwater and aerial. After defining the system goals, assumptions and constraints, which include communication and planning constraints in addition to identifying the failure and adverse conditions, we structure the requirements as follows:

1. Operator Safety Requirements
2. Platform Functional Requirements
    a. Unmanned Surface Vehicle (USV)
    b. Unmanned Underwater Vehicle (UUV)
    c. Unmanned Aerial Vehicle (UAV)
3. Possible Exceptions and Recovery Actions
4. Security Requirements

In the initial version, our focus is on the available assets and their interfaces to the IMMS. After analysing the existing requirements, we identify what is missing, in this case it is clearly the IMMS requirements or in other words ***what we want***. From what we know and what we want, we identify the functional and non-functional requirements of the IMMS, the IMMS interface requirements and information communication. The functional requirements include mission planning, mission execution and mission monitoring and review. Later, we can identify a common functionality among the different assets and generalise the platform-specific requirements.

Capturing the requirements in a well-defined document is not enough. The document can be still prone to different interpretations from the different team members coming from different backgrounds. Therefore, it is important to have a precise specification to eliminate any ambiguities and remove any defects. For this we use Event-B, introduced in Section 2.1, to capture the system requirements precisely. In Section 4 we present an early attempt at modelling the high level requirements of the system using Event-B.

Figure 1 presents our proposed approach for eliciting requirements for autonomous missions. This approach is based on our experience in using formal modelling for system verification, it is a generic approach which is applicable for the integration of multi-platforms. Our approach is iterative where we augment these requirements as a result of continuous analysis. This approach requires continuous interaction between two main stakeholders the domain experts, which in our case includes two parties: the different platform owners and the client (Thales), and the formal methods experts. The platform owners will identify what are the feasible requirements and the client identifies what is the purpose of the system. The formal methods experts will work on analysing the available resources to identify what is missing and what is ambiguous which will need clarification from the domain experts, who should also approve or reject any identified requirements. In the next sections we apply this approach to defining the requirements of the IMMS.

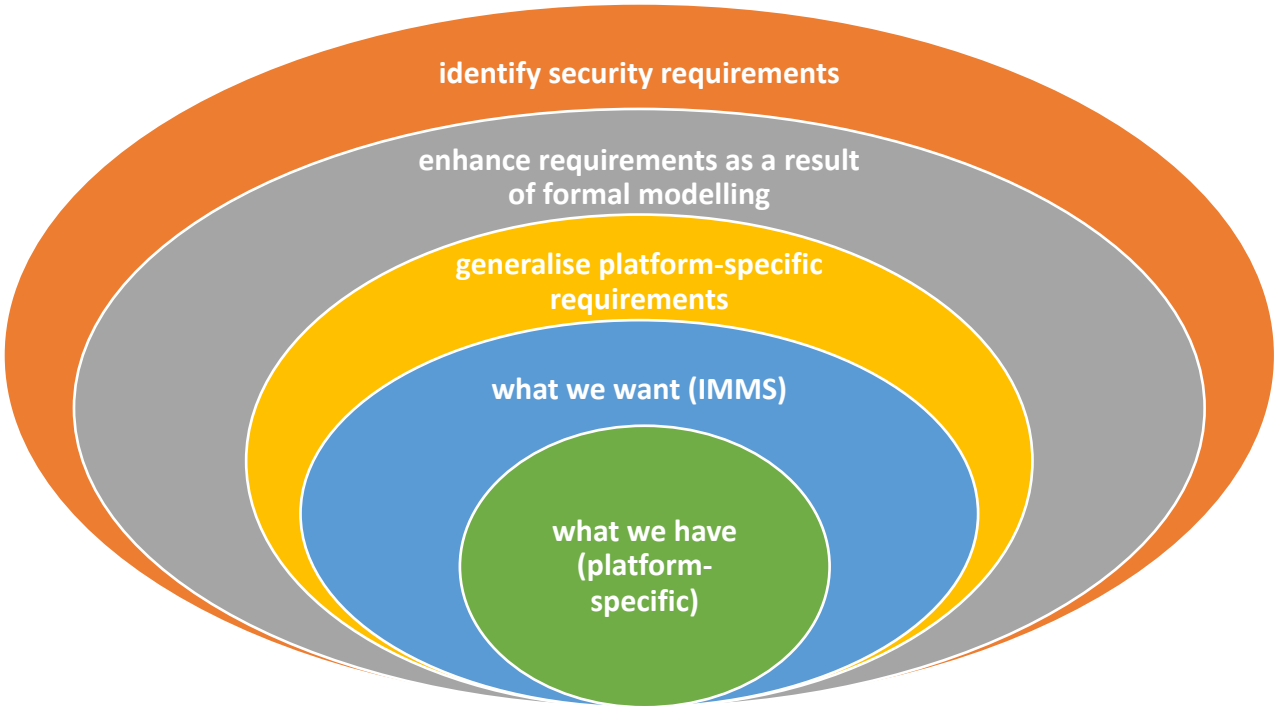

Figure 1: An approach for eliciting requirements

3.1 Analysis of an IMMS Safety Requirement

In this section, we will focus on one of the safety requirements of the IMMS, which we use as a running example to illustrate our approach rather than presenting the full set of requirements. In the first stage we looked at identifying the operator Safety Requirements (SF) of the available assets, one of these requirements is:

SF1. Collision Avoidance (CA): The UxVs (unspecified unmanned vehicles) do not have collision avoidance mechanisms. Collision avoidance with other vehicles and other possible obstacles is maintained by thorough planning, which can be updated during the mission should conditions change. Additionally, maintaining *visual line of sight*, receiving video feeds from platforms and defining mitigation scenarios assist with collision avoidance.

CA Requirement Analysis: The available vehicles do not have collision avoidance mechanism. Therefore, in order to avoid collisions:

A. Initial planning should take into considerations the different assets positions and any known obstacles in the environment.
B. During the mission when situational awareness is available and the assets are communicating, the plans can be updated and sent to the assets to avoid collisions.
C. A *timeout* should be predetermined for the assets with a predetermined plan to follow in case of communication loss.

Identifying IMMS functional requirements: By analysing the **SF1,** we can identify some of the IMMS Functional Requirements (FR) which **should** include:

FR1. The IMMS must have the ability to specify/assign the required vehicles to perform a mission.
FR2. The IMMS must assign tasks to the specified vehicles.
FR3. The IMMS must provide the vehicles with initial plans prior to starting a mission.
FR4. The IMMS must have the ability to modify plans of assigned vehicles during the mission executions.

Both **FR1** and **FR2** can be inferred from **A**, since planning should know the positions of the mission assets, then it should have the ability to assign these assets to a mission and give them tasks to perform a mission. From both **A** and **C**, **FR3** is deduced which will result in providing the vehicles with plans in the case of normal and failure behaviours. **FR4** is clearly concluded from **B** where plans should be updated should problems arise.

In this section, we have shown how we can identify some of the system requirements by analysing the requirements of *what we know.*

4. MODELLING IN EVENT-B

A key strength of Event-B is refinement, which allows us to abstract away from details and focus on different problems at different levels of refinements. The goal of this early modelling is an attempt to understand the system under development, remove ambiguities and identify important missing properties of the system to enhance the requirements.

Our Event-B model starts with an abstract level defining a mission as a set of tasks. Then, we introduce two refined levels as: vehicles and mission planning.

4.1 Abstract Level: Mission

At this level we define a mission as a set of tasks and introduce the events: *define_mission*, *start_mission* and *complete_mission*. These events will execute in the following order: <*define_mission*; *start_mission*; *complete_mission* >.

This level has three simple events, however right at the start of modelling we have to take a modelling decision: *Can the IMMS manage multiple simultaneous missions*?

For this project we will manage one mission at a time and leave this question as a future research question. Other questions that we have identified at this level are as follows:

- *What are the conditions for starting a mission, or we can ask, do we need all the vehicles to be present to start a mission?*
- *What are the conditions for completing a mission?*

We can define a new requirement for the IMMS, related to starting a mission:

FR5. The IMMS should define a minimum criterion for starting a mission.

In the following section, we will show how we can model this FR5 requirement by introducing a new refinement level.

4.2 First Refinement: Vehicles

In the first refinement level, we introduce vehicles and their capabilities and the possibility of assigning vehicles to mission tasks. Figure 2 represents a class diagram of the static part of the model, Vehicles context, while Figure 3 represents a class diagram of the dynamic part of the vehicles model. This is a UML-like representation of the model, with a formal translation to Event-B called UML-B class diagram (Snook & Butler, 2008) (Snook & Butler, 2003). This class diagram shows the different relationships between the different classes of the model and some of the class methods which are translated to events in the Event-B machine. In the context, the classes are translated to either sets or constants, in our case sets, while in the machine they are translated to variables but the machine can still reference the context classes as shown in Figure 3. The associations are translated to constants in the context and to variables in an Event-B machine.

The main functionality at this level is to ensure that a mission can only start after assigning vehicles with the minimum required capabilities defined to start the mission tasks. In Event-B this is ensured by defining the following invariant which must be maintained by all the events:

@inv1: missionStart = TRUE ⇒ requiresMin[missionTasks] ⊆ capabilities[assign~[missionTasks]]

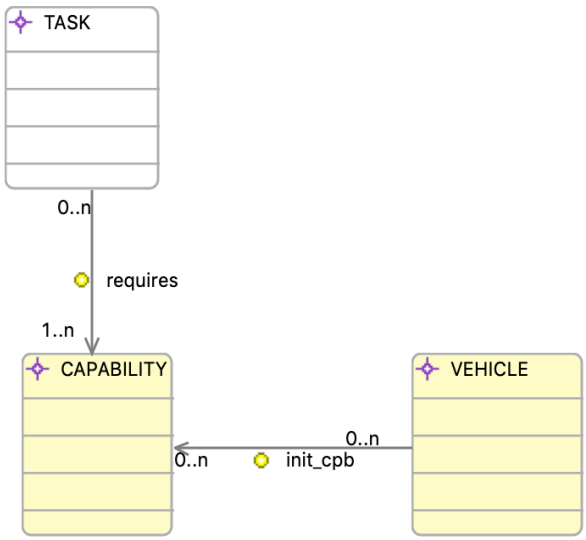

Figure 2: Vehicles Context

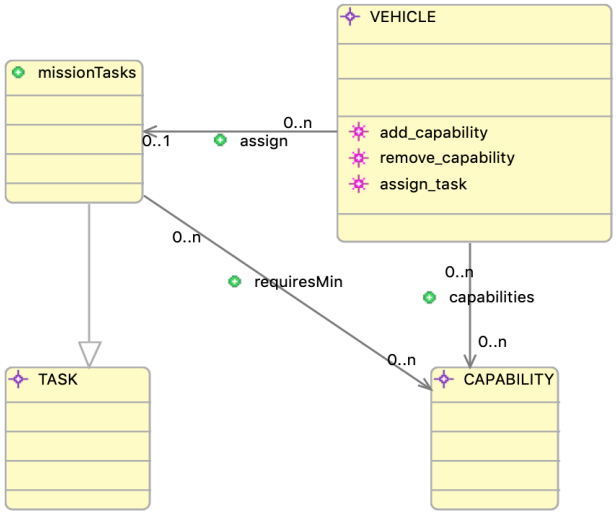

Figure 3: Vehicles Machine

In Event-B, proof obligations will be generated to ensure that all events will maintain the defined invariants. Invariant ***inv1*** will address some of the requirements, in this case requirement **FR5**. In addition to that Event-B can help to prove the consistency of the invariants, for example if we have additional invariants that conflict with each other, it will be impossible to prove the model and hence it will flag a problem to the modeller and requirements can be changed accordingly.

However, when starting this early we came up with additional questions that were not defined clearly in the requirements document, for example: *Do we allow vehicles to be deallocated from their tasks before mission completion?* If yes, then this invariant cannot be maintained by all the events, however we can address the minimum requirement of starting a mission as a guard in the event *start_mission*, which is a precondition to execute the event, but is not necessarily maintained by all the other events.

In this section, we have shown an example of how we can use Event-B to improve the requirements and identify some defects. We will also show how to trace the requirements in the model in Section 4.6. In this case, some invariants and guards are added to address some requirements for starting a mission that is why it is important to label the requirements to facilitate their traceability in the model.

4.3 Second Refinement: Mission Planning

At this level, we introduce mission plans abstractly as a series of locations, and in our model we ensure that a mission is not considered successfully complete until all vehicle plans are covered. We also introduce an event to set an initial plan before starting a mission and another event that enable modifying plans during execution, addressing requirements **FR.3** and **FR.4**.

This level also poses new questions about the conditions for modifying plans, is it always a response to some changes to the environment, do we immediately modify the plan or does the vehicle has to go through a safe state?

To answer these questions, we suggest defining generic states that apply to all the vehicles and define 'what are the activities that can occur during these states?'. These activities and the state transitions will be modelled as events in Event-B. A possible generic state-machine for the vehicles is shown in Figure 4.

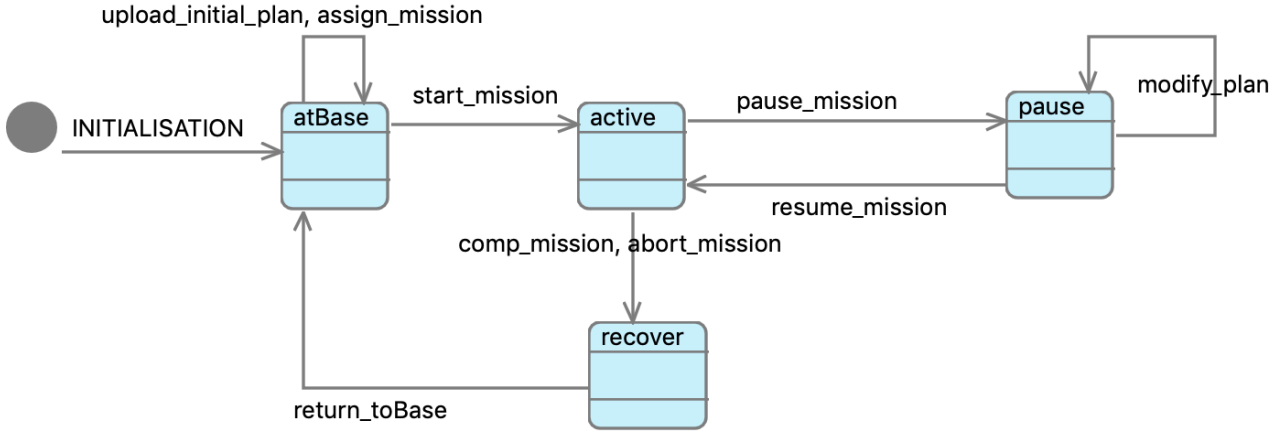

Figure 4: Generic Vehicle State machine

We can then refine the generic states by defining the tasks specific to each vehicle. For example, in an active state a USV will do the following events in order: move_to_survey_area, deploy_UUV, then recover_UUV. Similarly we can define another vehicle-specific events for a UUV such as: dive, survey and then surface.

4.5 Future Refinements & Security Requirements

In the previous sections, we have defined a high level abstraction of the IMMS, and we have shown how we used Event-B to identify new requirements, discover some ambiguities that require decisions from domain experts. After enhancing the requirements based on our formal modelling, we will introduce timing constraints, and model any remaining requirements by introducing new refinements. We will also extend the contexts to provide some instantiations to better fit with the mission types and provide a basis for mission validations.

In this paper, we have not described the security requirements, but in previous work (Omitola et al, 2018) and (Omitola, Rezazadeh, & Butler, 2019) we have used System Theoretic Process Analysis (STPA) (Leveson & Thomas, 2018) and STPA-Sec (Young & Leveson, 2013) to analyse the security requirements of maritime cyber-physical systems. In (Snook, Hoang, & Butler, 2017), we propose a general approach based on abstraction and refinement to analyse and construct security protocols using Event-B together with UML-B class diagrams and state-machines for diagrammatic visualisations. Regarding the IMMS security requirements, we intend to follow a similar approach using STPA to analyse, identify the unsecure scenarios and define the mitigation scenarios and constraints, then use Event-B and UML-B for verifying the system constraints.

4.6 Requirements Traceability in Event-B

In this section we show in detail how we can capture a requirement in Event-B. Table 1 uses requirement **FR.5** presented in Section 4.1 as an example. We show how existing events are extended with new guards and actions to capture the requirement. This requirement is enforced by

an invariant. The importance of the invariant is to ensure that the defined constrained is maintained by all the events.

Table 1: Requirement Traceability

Req. ID	Model	Representation in the model	Event-B Syntax
FR. 5	1st Refinement: Vehicles (Machine)	*define_mission*: Action to set minimum required capability	requiresMin ≔ t × cpb
		start_mission: Guard to check that the minimum required capabilities is already assigned to the mission tasks.	requiresMin[missionTasks] ⊆ capabilities[assign∼[missionTasks]]
		Invariant to ensure that a mission can only start if the required minimum capabilities are assigned (inv1).	missionStart = TRUE ⇒ requiresMin[missionTasks] ⊆ capabilities[assign∼[missionTasks]]

For example, we have an event *remove_capability* that allows us to remove a capability from a vehicle, if this event is not constrained, Event-B will not be able to discharge the proof obligation related to maintaining invariant (***inv1***) in this event, hence the modeller will discover that something need to be changed, in this case we need to add a guard that prevents removing a capability from a vehicle with assigned task.

5. CONCLUSIONS

Autonomous systems are safety-critical systems, hence the need for assurance techniques is a necessity for trusting such systems and be certified for use. Having a trustworthy part of the system is not enough to trust the trustworthiness of the overall system and having a heterogeneous system that involve platforms from different domains adds another level of complexity to gain certification. Maritime operating environments are particularly challenging for V&V. The environmental conditions can cause vehicle failures or impede communications i.e., interconnecting autonomous systems, could potentially open up these systems to more security attacks. Unpredictability of the maritime environment can mean that plans need to be updated autonomously during mission execution. It may not be feasible to characterise, fully, the environmental conditions and required system responses of AS in advance of deployment. In future work, we aim to be able to characterise the ***safety and security envelopes*** within which system responses should reside. An emerging approach to ensuring safety in Artificial Intelligence (AI)-based systems is to augment them with ***policing functions*** (Hoang et al, 2018) that monitor AI decision-making for conformance to safety/security envelopes so that, when an unsafe/insecure decision is detected, some failsafe action is invoked, e.g., command a drone to loiter or vessel to surface. Ideally, safety/security envelopes can be characterised in precise ways, making sure policing functions are amenable to the characterisation required for assurance cases in advance of deployment.

In this paper, we have proposed an approach for eliciting requirements for autonomous missions and formalising these as Event-B models. This is part of the functional process for an Integrated Mission Management for heterogenous autonomous systems. Figure 1 summarises the proposed approach and shows how we augment the requirements through continuous analysis. The proposed approach is iterative where we continuously need to do reviews which can influence the formal modelling on one hand and the formal modelling can influence the system requirements by identifying new requirements, removing ambiguities and defects. At the early stages we are using formal modelling for requirements and design analysis and to prove the consistency of the system properties. Later we could use formal modelling to verify and validate the high level plans.

ACKNOWLEDGEMENTS

This work is funded by Thales IMMS project.

REFERENCES

Abrial, J.-R. (2010). *Modeling in Event-B: System and Software Engineering*. Cambridge University Press.

Abrial, J.-R., Butler, M., Hallerstede, S., Hoang, T. S., Mehta, F., & Voisin, L. (2010). Rodin: an open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer*, *12*(6), 447–466. https://doi.org/10.1007/s10009-010-0145-y

Armstrong, R. C., Punnoose, R. J., Wong, M. H., & Mayo, J. R. (2014). *Survey of Existing Tools for Formal Verification*. https://doi.org/doi:10.2172/1166644

Hoang, T. S. (2013). *An Introduction to the Event-B Modelling*. https://doi.org/10.1007/978-3-642-33170-1

Hoang, T. S., Sato, N., Myojin, T., & Butler, M. (2018). Policing Functions for Machine Learning Systems. In *Workshop on Verification and Validation of Autonomous Systems: Satellite Workshop of Floc 2018* (pp. 1–10). Retrieved from https://eprints.soton.ac.uk/421233/%7D

Leffingwell, D. (1997). Calculating the Return Investment from more Effective Requirements Management. *American Programmer, 10*(4), 13–16.

Leuschel, M., & Butler, M. (2008). {ProB}: An Automated Analysis Toolset for the {B} Method. *Software Tools for Technology Transfer (STTT)*, *10*(2), 185–203.

Leveson, Nancy G. and Thomas, J. P. (2018). *STPA Handbook*. Cambridge, MA USA.

MacDonell, S. G., Min, K., & Connor, A. M. (2014). Autonomous requirements specification processing using natural language processing. *CoRR*, *abs/1407.6099*. Retrieved from http://arxiv.org/abs/1407.6099

Omitola, T., Downes, J., Wills, G., Zwolinski, M., & Butler, M. (2018). Securing Navigation of Unmanned Maritime Systems. In N. C. Schillai, Sophia M. and Townsend (Ed.), *Proceedings of the 11th International Robotic Sailing Conference: Southampton, United Kingdom, August 31st - September 1st, 2018.* (pp. 53–62). Southampton: CEUR-WS. Retrieved from http://ceur-ws.org/Vol-2331/paper5.pdf

Omitola, T., Rezazadeh, A., & Butler, M. (2019). Making (Implicit) Security Requirements Explicit for Cyber-Physical Systems : A Maritime Use Case Security Analysis. In G. A.-K. et Al (Ed.), *Proceedings of the 30th DEXA Conferences and Workshops*. Linz, Austria: Springer Nature Switzerland AG 2019. https://doi.org/https://doi.org/10.1007/978-3-030-27684-3_11

Snook, C., & Butler, M. (2003). UML-B : Formal modelling and design aided by UML, 1–32.

Snook, C., & Butler, M. (2008). UML-B and Event-B: an integration of languages and tools. In *The IASTED International Conference on Software Engineering - SE2008*. Retrieved from https://eprints.soton.ac.uk/264926/

Snook, C., Hoang, T. S., & Butler, M. (2017). Analysing Security Protocols Using Refinement in iUML-B. In C. Barrett, M. Davies, & T. Kahsai (Eds.), *NASA Formal Methods* (pp. 84–98). Cham: Springer International Publishing.

Young, W., & Leveson, N. (2013). Systems Thinking for Safety and Security. In *Proceedings of the*

29th Annual Computer Security Applications Conference (pp. 1–8). New York, NY, USA: ACM. https://doi.org/10.1145/2523649.2530277