A Requirements Modelling Language to Facilitate Avionics Software Verification and Certification

Andrés Paz* and Ghizlane El Boussaidi†
École de Technologie Supérieure, Université du Québec
Montreal, Canada
Email: *andres.paz-loboguerrero.1@ens.etsmtl.ca, †ghizlane.elboussaidi@etsmtl.ca

***Abstract*—Engineering avionics software is a complex task. Even more so due to their safety-critical nature. Aviation authorities require avionics software suppliers to provide appropriate evidence of achieving DO-178C objectives for the verification of outputs of the requirements and design processes, and requirements-based testing. This concern is leading suppliers to consider and incorporate more effective engineering methods that can support them in their verification and certification endeavours. This paper presents SpecML, a modelling language providing a requirements specification infrastructure for avionics software. The goal of SpecML is threefold: 1) enforce certification information mandated by DO-178C, 2) capture requirements in natural language to encourage adoption in industry. and 3) capture requirements in a structured, semantically-rich formalism to enable requirements-based analyses and testing. The modelling language has been developed as a UML profile extending SysML Requirements. A reference implementation has been developed and an empirical validation was performed in the context of an industrial avionics case study.**

***Keywords*—Requirements modelling language, requirements-based testing, avionics software, DO-178C, certification.**

I. Introduction

Safety is a major concern for the aviation industry. Avionics systems must be developed appropriately to avoid, or at least mitigate, posing undue harm to anyone or anything in the aircraft's operational environment. The advent of software as the controller of behaviour for avionics systems has made software a prime contributor to these systems' potential failure conditions [1, 2]. Software may erroneously control their behaviour or mislead the systems' users (*i.e.* the pilots or other avionics systems) into carrying out inappropriate actions [3]. Thus, aviation authorities impose stringent regulation, like DO-178C [4] and its European equivalent ED-12C, on the development of software for airborne systems. The aim of the DO-178C guideline is to produce software that is validated and verified for its airworthiness, *i.e.* reliable and safe-to-use in flight.

For DO-178C, just as with any software development project, requirements engineering is a critical phase. The majority of errors found in safety-critical software usually have their origin in this phase. Nevertheless, over 50% of them are only detected and corrected in the final phases of development [5]. DO-178C prescribes objectives that need to be satisfied in this regard: 1) requirements are developed in a hierarchical way with explicit bi-directional traces between them, 2) requirements exist for both normal-range and robustness (abnormal-range) conditions, and 3) specific test cases are developed from the requirements to test software responses to normal-range and robustness inputs and conditions. Current industry practices have requirements specified using constrained natural language and managed with the help of textual requirements databases, like IBM DOORS [6, 7]. Although the use of a constrained natural language brings some discipline and rigour to requirement specifications, this practice is still focused on writing requirements in human-readable prose. This inevitably raises problems for satisfying the DO-178C objectives. Natural language indeed facilitates communication between stakeholders but it is not a suitable form of specification for supporting interrelationships, decomposition, requirements-based analyses and testing [8]. In such context, it becomes essential to provide a requirements specification language.

Several requirements specification languages exist (*e.g.*, [7, 9–16]). Some of them only support the specification of natural language-based requirements (*e.g.*, [9, 11, 12]). Others allow the expression of semantically richer requirement statements than natural language statements (*e.g.*, [13–16]). However, all of the latter force requirements to be captured in an already structured form, which restrains their adoption even when they can enable requirements-based analyses and testing. Moreover, none of these languages provides sufficient support for the interrelationships and decomposition as defined by DO-178C.

This paper proposes SpecML. SpecML is a modelling language that provides a requirements specification infrastructure for avionics software in the context of DO-178C. SpecML is designed as a UML profile augmenting SysML Requirements by integrating constructs from several existing approaches. The goal of SpecML is threefold. First, enforce certification information mandated by DO-178C like the interrelationships and decomposition of requirements. Second, capture requirements in natural language to smooth the way for its adoption in industry. Third, provide facilities to capture requirements in a structured, semantically-rich formalism to enable requirements-based analyses and testing. The language is open and may be tailored to other industries and regulatory guidelines and standards. SpecML is validated by the development of a reference implementation and the evaluation of its effectiveness with an avionics industrial case study.

The remainder of this paper is organized as follows. Section II provides an overview of the necessary background

knowledge and examines related work. Section III describes SpecML. Section IV presents SpecML's reference implementation and validates the language through experimentation with an avionics software case study. Finally, Section V draws conclusions and outlines future work.

II. Background and Related Work

The work has been developed in an industrial setting of avionics companies, which are required to deliver safety-critical software compliant with DO-178C. Subsection II-A describes the DO-178C guideline, giving special attention to some of the objectives for the requirements specification, design and verification processes. Subsection II-B discusses related work. This information will serve as the necessary background knowledge for a reader to understand the motivations behind SpecML.

A. DO-178C

The DO-178C guideline [4] is the de facto standard for developing airworthy software for airborne systems [17]. Its regulatory scope encompasses the complete software development life cycle, from requirements specification to verification of the resulting software. DO-178C prescribes, among others, the following processes: requirements, design, and verification and validation (V&V). For each of these processes DO-178C defines airworthiness assurance needs in the form of process objectives. The requirements process develops (*i.e.* refines and decomposes) system requirements allocated to software (SRATS) into high-level software requirements (HLRs) suitable for directing the software design activities. The design process covers the development of low-level software requirements (LLRs) from the HLRs, and the development of the software architecture. SRATS, HLRs and LLRs must be traced between them to their originating requirement. In such cases when a requirement cannot be directly traced to an originating requirement it must be identified as a derived requirement. Requirements must exist for both normal-range and robustness (abnormal-range) conditions.

V&V is a transverse process responsible for detecting and reporting errors that may have been introduced during any process. For this, V&V must perform a combination of reviews, analyses and testing over the processes' outputs. Reviews and analyses of the outputs of the requirements and design processes must ensure that HLRs are traceable to SRATS, LLRs are traceable to HLRs, and HLRs and LLRs exist for both normal-range and robustness (abnormal-range) conditions. Testing involves the creation of specific test cases from the HLRs to test software responses to normal-range and robustness inputs and conditions.

B. Related work

Several requirements modelling languages exist that are based on UML. SysML [9] adds specific constructs to UML for capturing requirements and their interrelationships. Nonetheless, requirements are expressed with simple natural language statements and the semantics of relationships are ambiguous and open to interpretation. MARTE [10] is intended for describing real-time and embedded systems. However, MARTE is focused on non-functional aspects of requirements. Thus, lacks the necessary constructs to create a complete requirements specification on its own.

SafeML [11] extends UML for capturing safety-related requirements allocated to software and enabling the monitoring of the design and implementation of the software with regard to the provided safety requirements. The approach in [12] targets the specification of test models that can serve both for software testing and as supporting evidence in DO-178 certification processes. The approach consists of a UML profile extending UML activity diagrams with safety-related information and traces to requirements. These two languages, however, capture requirements in natural language, meaning the safety-related information must be manually extracted to support their verification.

A number of approaches exist allowing the expression of semantically richer requirements than natural language statements (*e.g.*, [7, 13–16]). RSML (Requirements State Machine Language) [13] captures requirements with a state-like formalism as functions describing the mapping between inputs and outputs of the system in face of disturbances. RSML features hierarchical abstraction to hide low-level information and make the specification more readable. Two levels of hierarchical abstraction are suggested to meet DO-178C requirements hierarchy. RSML has been used in the avionics domain [13]. Despite that, a complete formalization of the language was not created and the language is not openly available [18]. Furthermore, the use of hierarchical abstraction obscures the separation between HLRs and LLRs. The language even allows users to include design-level information in the requirements specification, a practice that is greatly discouraged by DO-178C.

RDAL (Requirements Definition and Analysis Language) [7] is a standardized modelling language for capturing, validating, analyzing and verifying system requirements. Requirements can be expressed using both natural language statements and Use Case Maps (UCM), a sublanguage of the User Requirements Notation (URN). A major drawback is that neither RDAL nor UCM were designed with DO-178C objectives in mind. Moreover, UCM, like RSML, is prone to convey design information.

The approach in [15] proposes the use of Horizontal Condition Tables (HCTs) for requirements specification of safety-critical software. An HCT defines a requirement as a single function describing a single behaviour that computes a single output. The studies in [14, 19] propose the property-based requirements (PBR) theory as a method for solving the problems of ambiguity, inconsistency and incompleteness in natural language-based requirements specifications. A PBR is defined as a constraint for the system enforcing a property whenever a condition is met. SpeAR (Specification and Analysis of Requirements) [16] is an open-source tool and formal language for capturing and analyzing requirements. The

language is designed to read like natural language although it has the formal semantics of Past Linear Temporal Logic. Thus, it supports proofs of critical properties about requirements, like logical entailment and logical consistency, using model checking. SpeAR can also capture natural language-based requirement statements but only for those requirements that cannot or are not intended to be formalized. Although SpeAR has been developed with DO-178C compliance in mind, it has a vocabulary that is not specific to DO-178C and does not enforce certain mandatory information (*e.g.*, rationale). HCTs, PBRs and SpeAR force requirements to be captured in an already structured form, which restrains their adoption in industry. Moreover, they lack constructs for expressing timing constraints and clocks, interrelationships between requirements, and the identification of derived requirements.

III. SpecML: requirement specification modelling language for avionics software

SpecML is proposed as a language that: 1) enforces certification information mandated by DO-178C like the interrelationships and decomposition of requirements, 2) captures requirements in natural language to smooth the way for its adoption in industry, and 3) provides facilities to capture requirements in a structured, semantically-rich formalism to enable requirements-based analyses and testing. In order to provide such a comprehensive solution, SpecML was designed as a hybrid language extending and combining features, on one hand, from SysML and MARTE, and, on the other hand, from PBR theory.

The avionics industry considers that UML, with profiles such as SysML and MARTE, provide better long term sustainability and interoperability over completely custom-built domain-specific languages [20]. A list of benefits justifying the use of UML in the avionics industry can be found in [11]. PBR theory is aligned with model-driven engineering, which the avionics industry is leaning towards to reduce software and development complexities and to support them in their certification endeavors [2, 8, 11]. PBR theory has been acknowledged in the SysML specification as an improvement to SysML Requirements to address a formal expression of requirements in SysML and expanding its ability to support requirements-based analyses and testing.

The methodology in [21] has been followed to build SpecML. This methodology was proposed with the purpose of guiding the development of UML profiles that are technically valid (*i.e.* do not contravene the UML standard) and of good quality. The approach requires the development of two artifacts that are distinct but closely related: a conceptual model of the domain (or domain metamodel) and the UML profile itself. The general flow of the approach is fairly simple: develop the domain metamodel and, then, map its concepts to elements in the UML metamodel. Depending on the domain metamodel's complexity there may need to be a few iterations of the process to ensure conformance with UML. In the following subsections, we discuss SpecML features. A usage example of the profile is presented in Section IV.

A. The SpecML profile

SpecML is designed as a UML profile that augments SysML Requirements with new constructs. SysML 1.5 was selected since this version is more open to extensions regarding requirement specification compared to previous versions. SpecML has the common structure of a UML profile: data types, stereotypes and OCL constraints. No custom data types are defined, hence, UML primitive types are used to define the attributes owned by the stereotypes. Specialized stereotypes are defined to capture key concepts of requirements specification in accordance with DO-178C. OCL constraints are defined to enforce additional necessary checks to achieve DO-178C objectives. The stereotypes and OCL constraints in SpecML can be categorized into three groups: 1) requirement hierarchy, 2) requirement interrelationship, and 3) requirement formalization.

B. Requirement hierarchy with SpecML

Figure 1 shows the stereotypes to represent DO-178C's requirement hierarchy. The abstract stereotype `Requirement` extends the SysML `AbstractRequirement` stereotype. This inheritance provides the facilities to capture natural language requirement statements (the `text` attribute) and specify an identifier (the `id` attribute). On top of that, the SpecML `Requirement` stereotype adds attributes to further characterize a requirement, like `type` (structural, behavioural, mixed), `source` (*e.g.*, acquirer, operator, certification authority, certification standard) and `status` (pending review, reviewed and accepted, reviewed and incorrect). The `isDerived` attribute is used to indicate that the requirement is not directly traceable to higher level requirements because it specifies behaviour beyond what has been specified in them.

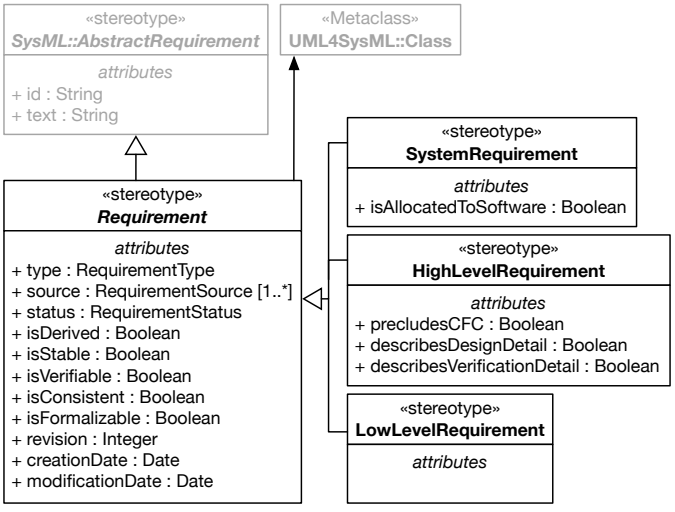

Fig. 1. Requirement hierarchy stereotypes.

The `SystemRequirement`, `HighLevelRequirement` and `LowLevelRequirement` stereotypes specialize the `Requirement` stereotype to define the requirement hierarchy found in DO-178C. The `SystemRequirement` stereotype represents

the highest level requirements in the hierarchy. SystemRequirements are system requirements allocated to software (SRATS) when the isAllocatedToSoftware attribute is set to *true*. Recall that SRATS are the ones developed and refined into software requirements. System requirements are specified during the system life cycle processes, beyond the regulatory scope of DO-178C. Only SRATS are considered for software development. However, the SystemRequirement stereotype provides the possibility to formalize any system requirement to enable its analysis and testing as well.

The HighLevelRequirement stereotype (HLR for short) represents the software requirements that are produced directly from the refinement of SRATS. Attributes of HighLevelRequirement (*i.e.* precludesCFC, describesDesignDetail and describesVerificationDetail) support analyses for DO-178C compliance. For instance, OCL constraints defined over the last two attributes enforce the specification of a rationale when these attributes are set to true. A rationale is mandatory in these situations for DO-178C certification. The precludesCFC attribute indicates if the requirement intends to prevent one or more of the identified software contributions to the system's failure conditions.

The LowLevelRequirement stereotype (LLR for short) represents the software requirements from which source code can be directly implemented without further information, *i.e.* the detailed design. This stereotype can be used as a stand-alone element to capture natural language or formal requirement statements, or be applied onto UML/SysML model elements that represent the software's design.

C. Requirement interrelationship with SpecML

Figure 2 presents the stereotypes for defining the types of relationships that can occur between requirements. The RefineReqt stereotype is an alias to the SysML DeriveReqt stereotype to align it with the DO-178C vocabulary. The stereotype represents a bi-directional trace in which a requirement can be refined into a lower-level requirement. This relationship goes from the refining requirement (*e.g.*, the HLR) to the refined requirement (*e.g.*, the SRATS).

The Copy relationship is as defined in SysML but with a more constrained usage. HLRs must be very detailed so as to guide the software's design. However, it could occur that SRATS are, in fact, very detailed so as to guide the software's design without any further refinement into HLRs. In this case HLRs must be defined and related to their corresponding SRATS using the Copy relationship, which will indicate that they are a read-only copy of the supplier requirement (*i.e.* the SRATS).

The Derive stereotype is included to make it possible to trace derived requirements indirectly to a higher level requirement through the mediation of the requirement from which they were derived. Requirements at the same level of the requirements hierarchy may experience some interdependence. The Coupled stereotype makes it possible to represent such relationship between two requirements.

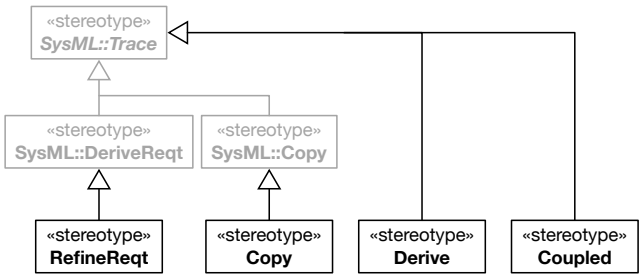

Fig. 2. Requirement interrelationship stereotypes.

OCL constraints (not shown due to lack of space) defined over all these relationship stereotypes enforce the necessary checks to achieve DO-178C objectives.

D. Structured, semantically-rich requirements with SpecML

One of the notable features of SpecML is its ability to capture requirements in a structured, semantically-rich formalism in order to enable requirements-based analyses and testing. Figure 3 shows the stereotypes to capture requirement formalizations. The formalism is founded on the property-based requirement (PBR) theory as defined in [14, 19], and the MARTE profile, since, as mentioned in Section II-B, PBR theory lacks constructs for expressing timing constraints and clocks. In avionics systems, the values of their properties commonly need to be evaluated repetitively at a given frequency. This information is part of the system requirements but cannot be captured using the PBR as it was defined. The MARTE profile (particularly the *CoreElements*, *Time* and *NFPs* sub-profiles) contains constructs for specifying such time-dependent behaviour and constraints. Thus, SpecML borrows these constructs from the MARTE profile to annotate PBRs that are time-sensitive.

The following expression formalizes a PBR:

Req: $[\,\text{when } C\,] \rightarrow val(O.P) \in D \subset im(P)$.

The term Req is a mandatory, unique requirement identifier. The rest of the expression is intended to read as follows: *"when condition C is met, the value(s) of property P of object O shall be in the subset D of the set of possible values for P"*. The presence of a condition C is optional as indicated by the presence of square brackets. The theory states that the conjunction of a finite set of PBRs $\{\text{Req}_n\}$ denotes the system.

The PropertyBasedStatement stereotype establishes a formalized statement of a requirement following the PBR theory. The previous PBR expression is broken down to simplify its representation with the profile. The id attribute in PropertyBasedStatement captures the Req term of the expression. An additional text attribute can hold a textual description of the formalization if necessary. The optional expression $[\,\text{when } C\,]$ is a *condition* of actualization in the context of the requirement. This expression can be captured with a SysML ConstraintBlock and linked to the PropertyBasedStatement with a dependency stereotyped by Condition. The mandatory expression $val(O.P) \in D \subset im(P)$ is a *predicate* representing the constraint over the value of a system

property. This expression can be captured as well with a SysML ConstraintBlock and linked to the PropertyBasedStatement with a dependency stereotyped by Predicate.

One property-based statement may not be sufficient to capture the entire requirement described in the natural language statement. Thus, additional PropertyBasedStatements may be introduced as part of the requirement's formalization. A dependency stereotyped by Formalization links each PropertyBasedStatement to the requirement. The requirement is, therefore, interpreted as the conjunction of the specified PropertyBasedStatements.

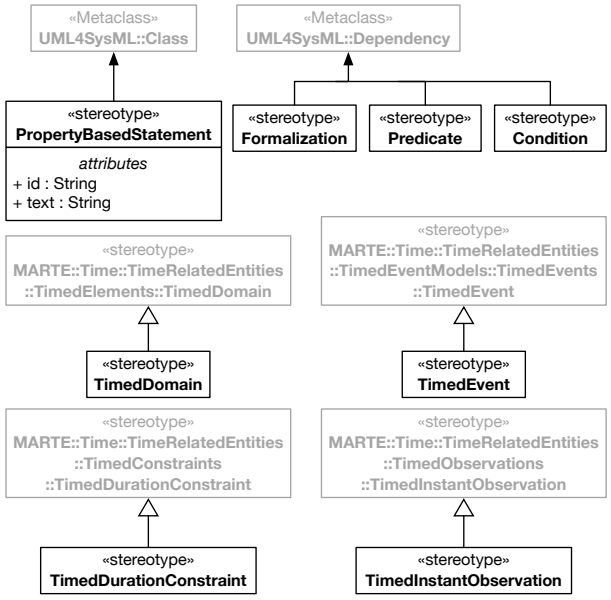

Fig. 3. Requirement formalization stereotypes.

The TimedDomain stereotype indicates a container of clocks. The stereotype must be applied onto a UML Package. Requirements in a TimedDomain package may use the clocks contained in it to express behaviour that is time-dependent. The TimedEvent stereotype is as defined by the MARTE specification but with a more constrained usage. The stereotype establishes a non-functional annotation on a requirement indicating that the specified behaviour needs to be performed with a predetermined frequency (*i.e.* it is explicitly bound to a clock).

The TimedDurationConstraint stereotype imposes a constraint on the temporal distance between two events. This stereotype is included to allow the expression of timing constraints between specified behaviour. The TimedInstantObservation stereotype is as defined by the MARTE specification but with a more constrained usage. The stereotype denotes an instant in time that is associated with an event occurrence and observed on a given clock. This stereotype is included to allow the observation of event occurrences and allowing their use in the expression of timing constraints on the specified behaviour. The stereotype must only be applied to a PropertyBasedStatement.

IV. Validation

This section reports on the validation of SpecML. Subsection IV-A describes the reference implementation of SpecML used for the validation. Subsection IV-B presents the avionics software case study carried out to evaluate the language's effectiveness. Subsection IV-C discusses the results of the case study. Subsection IV-D discusses threats to validity and limitations.

A. Reference implementation

SpecML is tool-independent, any UML modelling tool supporting UML profiles could be used to implement it. A reference implementation was implemented with the Eclipse Papyrus modelling environment [22]. The reference implementation comprises three components: 1) the *profile*, 2) the *validation rules*, and 3) the *modelling tooling*. The *profile* component defines the stereotypes for the language. The *validation rules* component defines the OCL constraints for the language and utilizes Papyrus' model validation framework for their execution by the user while creating a model. The *modelling tooling* component provides the user with facilities to create a specification model, *i.e.* editor with palette and context menus, and properties view.

Figure 4 displays a screenshot of the SpecML reference implementation. The middle of the screen shows the model editor with a model being created. On the right of the screen is the palette with the available language constructs. On the left center of the screen is the model explorer presenting all the elements currently in the model. At the moment, the model contains one SRATS (in red) and one HLR (in blue). The bottom of the screen displays a model validation error message indicating a violation of an OCL constraint by one of the model elements. The element causing the violation is marked in both the model editor and model explorer. The error message suggests options to the user for fixing the violation.

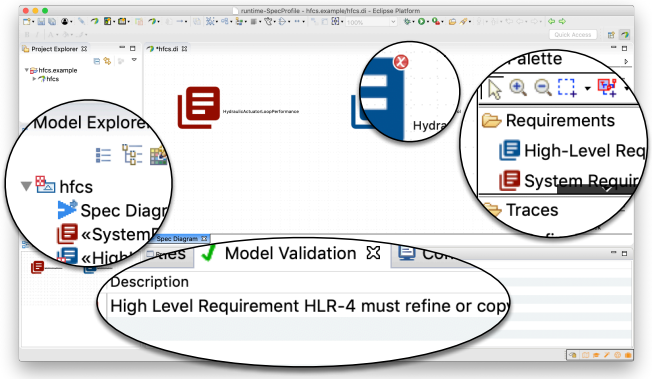

Fig. 4. Screenshot of the SpecML reference implementation.

B. Case study

In order to validate SpecML, it has been used in an avionics industrial case study of an aircraft's flight control software

(FCS). An open-source system description and software requirements written in natural language of an FCS is presented in [23]. The FCS is responsible for providing attitude[1] and attitude rate control based on pilot input commands to keep them within the flight envelope of the aircraft. The FCS controls three hydraulic actuators that allow the aircraft to pitch up or down, roll right or left, and yaw right or left. There are eleven system requirements allocated to software (identified by the prefix SR_ and a unique number), which have been refined into thirteen high-level software requirements (identified by the prefix HLR_ and a unique number). For lack of space only one SRATS and its refining HLR are discussed.

SR_4 Hydraulic Actuator Control Loop Performance. The FCS shall control the hydraulic actuator position with a minimum bandwidth of 10 Hz and a minimum damping of 0.4.

HLR_4 Hydraulic Actuator Loop Control. Each hydraulic actuator loop shall be implemented as a PID (proportional/integral/derivative) control loop operating at a 1 ms frame rate. The proportional gain shall be 0.3. The integral gain shall be 0.12. The derivative gain shall be 0.02.

The specification of SR_4 and its refinement by HLR_4 using the SpecML reference implementation is shown in Figure 5. Figure 6 presents the specification using the SysML notation. HLR_4 is formalized with one `PropertyBasedStatement` and one `ConstraintBlock` as the predicate. Note that no condition is stated. The predicate is defined in terms of more basic `ConstraintBlocks` that map to the different mathematical expressions in a PID loop. These `ConstraintBlocks` are nested into the `ConstraintBlock` representing the predicate through the SysML containment relationship. The MARTE `TimedEvent` stereotype is used to capture the 1 ms frame rate operation of the PID loop. The stereotype is applied onto the `PropertyBasedStatement` and displayed as a comment.

Producing the requirement specification model of the FCS with SpecML is only an intermediate goal in the software development process. The requirements are then allocated to entities of the design models intended to satisfy them using the SysML `Satisfy` dependency. SysML parametric diagrams can also be included to describe usages of the constraint blocks formalizing the requirements in the constraining of properties of the entities in the design models. This modelling establishes the method of evaluating design compliance with the specified requirements. Moreover, the property-based statements can be used to generate test cases.

C. Discussion

The process of using SpecML to specify the requirements of the FCS was iterative. Three domain experts and one software development expert reviewed the SpecML requirements specification model. Four benefits of using SpecML can be highlighted with the case study.

The first benefit regards requirement specification in accordance with DO-178C. All the FCS requirements (SRATS and HLRs) were modelled in a hierarchical way along with their interrelationships satisfying DO-178C objectives. The second benefit is that SpecML can relieve requirements engineers from the error-prone and labor-intensive work of manually verifying every requirement for compliance with DO-178C objectives. While building the specification model, several errors related to deviations from DO-178C objectives were detected with the reference implementation and corrected accordingly. The third benefit pertains to facilitating communication between stakeholders by capturing requirements in natural language while still allowing requirement formalization to enable analyses and testing. The thirteen HLRs were formalized using the specialized stereotypes for such purpose. As with any model, SpecML does not actually perform requirement analyses or testing, it is intended to provide facilities that enable requirements-based analyses and testing.

It is to be acknowledged that requirements specification standards vary from one company to another. This results in different ways of writing the same requirement statement. The fourth benefit is that SpecML can accommodate review efforts already put in place to check compliance with requirements specification standards. Requirement statements in SpecML are first represented in natural language and must be clearly defined. The quality of the requirements is dependent on such descriptions since they will be translated into PBR statements.

D. Validity and limitations

A systematic approach was followed for the analysis of DO-178C, as well as for the development of SpecML. In-depth discussions to validate the proposed stereotypes in SpecML took place with industrial practitioners that have ample expertise in both avionics development and DO-178C certification. There is a threat of having missed some concepts of the DO-178C guideline when analyzing it. This threat was mitigated by having a thorough understanding and iterative analysis of the guideline. Feedback from the industrial practitioners was also considered to mitigate the threat.

We evaluated SpecML through the FCS case study, which limits the conclusions about its usefulness and usability. However, the FCS was considered by the involved practitioners to be complex and representative of their industrial needs. Furthermore, MathWorks published the FCS case study to showcase the usability and highlight features of their MDE tool suite, which helps to deem it as representative. The use of an open-source system description allows comparisons and replication of the evaluation. The goal of this evaluation is not to prove SpecML applies to all avionics software requirements but rather show its application is feasible and can lead to DO-178C-compliant requirement specifications as well as to effective requirement analyses and testing. The results of the evaluation suggest SpecML is likely to provide engineers with potential gains in these regards.

SpecML's requirement formalization approach is primarily based on the property-based requirement (PBR) theory in [14, 19]. This is a fairly recent theory of requirement specification and more studies are needed to widely identify its limitations

[1]The aircraft's orientation about its center of mass.

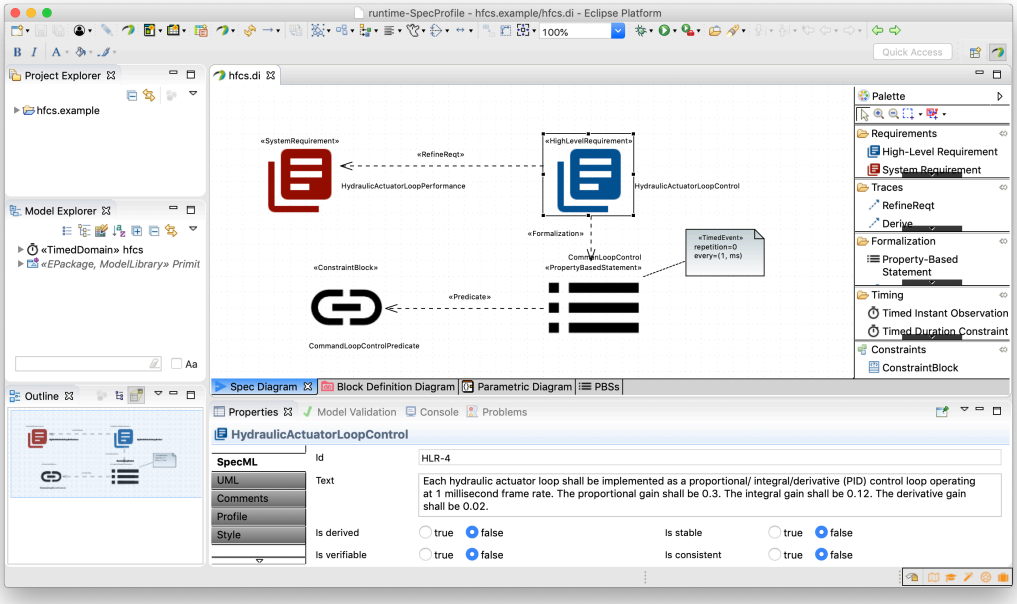

Fig. 5. Screenshot of the specification and formalization of HLR_4 with the SpecML reference implementation.

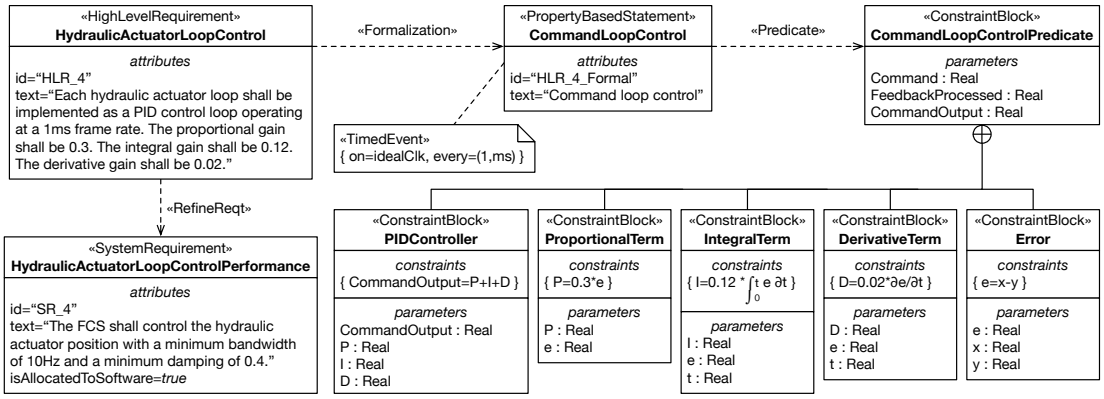

Fig. 6. Specification and formalization of HLR_4 with SpecML.

about the kinds of requirements that can be effectively specified as PBRs. The FCS case study certainly does not cover all kinds of requirements. For instance, the case study does not include requirements involving the sequencing of property evaluations which is supported by SpecML.

SpecML's reference implementation is dependent on the Eclipse Papyrus modelling environment. Some of the features of SpecML could not be properly implemented because of limitations in Papyrus' UML, SysML and MARTE implementations. Papyrus does not implement SysML 1.5, only SysML 1.4, which limits extensions that can be done to the Requirement stereotype. Papyrus' implementation of the MARTE profile was archived and could not be used successfully to integrate it with Papyrus' current implementation of UML. However, this by no means limit SpecML as its reference implementation can be developed differently.

V. Conclusion

Current avionics software industry practices have requirements specified using natural language, which facilitates stakeholder communication but makes them difficult to use in requirements-based analyses and testing. Thus, an approach that combines both natural language and formal requirements is important to allow stakeholder communication and requirements-based analyses and testing. This paper proposes SpecML, a modelling language that provides a requirements specification infrastructure for avionics software in the context of DO-178C. SpecML captures requirements in natural language and the structured formalism of PBRs, and provides facilities to satisfy the DO-178C objective regarding requirements specification and their verification.

A reference implementation was developed on top of the Eclipse Papyrus modelling environment. The validation of

SpecML was carried out through the realization of an empirical evaluation with the FCS case study. This showed the effectiveness of SpecML for requirement specifications. All the requirements of the FCS were modelled and, in addition, checked for compliance with DO-178C objectives. Then, it was showed that requirements-based analyses and testing can be performed from the information captured in the resulting SpecML model.

Future work involves an empirical evaluation carried out on larger scale industrial systems. As part of this, the language should be refined and improved. SpecML can be extended in several ways, including: 1) the integration of constructs to represent data dictionaries (*i.e.* data requirements), and 2) the development of a methodology to translate natural language requirement statements into PBR statements. The reference implementation is considered a prototype. Improvements can be made to the modelling tooling to provide a more interactive inspection and editing of requirements. Additional functions can be developed as well, like the (semi-)automatic generation of reports from the data that is captured.

Acknowledgments

This research has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Consortium for Research and Innovation in Aerospace in Quebec (CRIAQ) under project AVIO-604. The authors thank project members for their comments on the proposed approach.

References

[1] M. Huhn and H. Hungar, "UML for software safety and certification: Model-based development of safety-critical software-intensive systems," in *Proc. of the Int. Dagstuhl Conf. on Model-based Engineering of Embedded Real-time Systems*, 2007, pp. 201–237.

[2] R. G. Pettit, N. Mezcciani, and J. Fant, "On the needs and challenges of model-based engineering for spaceflight software systems," in *IEEE 17th Int. Symp. on Object/Component/Service-Oriented Real-Time Distributed Computing*, Jun. 2014, pp. 25–31.

[3] M. P. E. Heimdahl, "Safety and software intensive systems: Challenges old and new," in *Future of Software Engineering, 2007. FOSE '07*, May 2007, pp. 137–152.

[4] "Software considerations in airborne systems and equipment certification," RTCA, Inc., Std DO-178C, 2011.

[5] P. H. Feiler, "Model-based validation of safety-critical embedded systems," in *IEEE Aerospace Conf.*, Mar. 2010, pp. 1–10. DOI: 10.1109/AERO.2010.5446809.

[6] B. Potter, "Complying with DO-178C and DO-331 using Model-Based Design," MathWorks, Natick, Massachusetts, USA, Tech. Rep., 2012.

[7] D. Blouin, "Modeling languages for requirements engineering and quantitative analysis of embedded systems," PhD thesis, Université de Bretagne-Sud, Dec. 2013.

[8] Y. Moy, E. Ledinot, H. Delseny, V. Wiels, and B. Monate, "Testing or Formal Verification: DO-178C Alternatives and Industrial Experience," *IEEE Software*, vol. 30, no. 3, pp. 50–57, May 2013.

[9] "Systems Modeling Language," OMG, Std, 2017. [Online]. Available: http://www.omg.org/spec/SysML.

[10] "UML profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems," OMG, Std, 2011. [Online]. Available: http://www.omg.org/spec/MARTE.

[11] G. Zoughbi, L. Briand, and Y. Labiche, "Modeling Safety and Airworthiness (RTCA DO-178B) Information: Conceptual Model and UML Profile," *Softw. and Syst. Modeling*, vol. 10, no. 3, pp. 337–367, Jul. 2011.

[12] H. Stallbaum and M. Rzepka, "Toward DO-178B-compliant test models," in *Proc. of the MODEVVA Workshop*, 2010, pp. 25–30.

[13] N. G. Leveson, M. P. E. Heimdahl, H. Hildreth, and J. D. Reese, "Requirements specification for process-control systems," *IEEE Transactions on Software Engineering*, vol. 20, no. 9, pp. 684–707, Sep. 1994.

[14] P. Micouin, "Toward a property based requirements theory: System requirements structured as a semilattice," *Syst. Eng.*, vol. 11, no. 3, pp. 235–245, Aug. 2008.

[15] M. Bialy, M. Lawford, V. Pantelic, and A. Wassyng, "A methodology for the simplification of tabular designs in model-based development," in *IEEE/ACM 3rd Workshop on Formal Methods in Softw. Eng.*, May 2015.

[16] A. W. Fifarek, L. G. Wagner, J. A. Hoffman, B. D. Rodes, M. A. Aiello, and J. A. Davis, "SpeAR v2.0: Formalized past LTL specification and analysis of requirements," in *Proc. of NASA Formal Methods: 9th Int. Symp., NFM.*, May 16-18, 2017, pp. 420–426.

[17] K. J. Hayhurst and C. M. Holloway, "Challenges in software aspects of aerospace systems," in *Proc. of the 26th Annual NASA Goddard Softw. Eng. Workshop*, Nov. 2001, pp. 7–13.

[18] M. W. Whalen, "A formal semantics for the Requirements State Machine Language Without Events (RSML-e)," Master's thesis, 2000.

[19] P. Micouin, *Model Based Systems Engineering: Fundamentals and Methods*, ser. FOCUS Series. Wiley, 2014.

[20] T. Le Sergent, F.-X. Dormoy, and A. Le Guennec, "Benefits of Model Based System Engineering for Avionics Systems," in *Proc. of the 8th ERTS Europ. Conf*, Jan. 2016.

[21] B. Selic, "A Systematic Approach to Domain-Specific Language Design Using UML," in *10th IEEE Int. Symp. on Object and Component-Oriented Real-Time Distributed Computing (ISORC'07)*, May 2007, pp. 2–9.

[22] The Eclipse Foundation. (2017). Papyrus Modeling Environment, [Online]. Available: https://www.eclipse.org/papyrus/ (visited on 12/29/2018).

[23] B. Potter. (2016). DO-178 Case Study, [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/56056-do178_case_study (visited on 11/13/2018).