Towards Domain-Specific Modeling Methodology for Avionic Safety-Critical Systems

Emanuel S. Grant, *Member, IAENG*

***Abstract*— The use of domain-specific modeling languages and associated methodologies, provide support in application domain where the safe and reliable operations of the systems are of paramount importance to the users and organizations, and wherein the domains are well understood and documented. One such area of domain-specific modeling application is in the field of avionic systems. For software systems to be used onboard aircrafts they must be certified, and as such certification protocols have been established for developing these safety-critical systems. These established protocols are usually represented as textual documents and inherently are difficult to apply directly in software development environments. The work presented herein proposes a graphical modeling representation for an avionic software system certification specification and an accompanying model-driven methodology for implementing the certification specification. This work is based on the RTCA Software Consideration in Airborne Systems and Equipment specification and the Unified Modeling Language. The presented model-driven methodology, incorporates the use of formal specification techniques to satisfy many of the verification requirements of the RTCA specification. The benefit of this work is in the transformation of textual description to graphical models in support of precise software system development, and a rigorous model-driven software development methodology for avionic soft-ware system development.**

***Index Terms*— Domain-Specific Modeling Language, Formal Specification Techniques, Model Transformation, Safety-Critical Systems, Software Engineering.**

I. Introduction

DOMAIN-SPECIFIC modeling languages (DSMLs) [1] are a subset of software visual modeling languages that are characterized by having a vocabulary of terms and concepts that are fundamental to the problem and solution domains under consideration. Thus, DSMLs are identical, in definition, to that of software modeling languages, but with the added feature that the terms and concepts of the language have little applicability outside the specific domain. DSML are most suitable in large application domains that are well-understood, with respect to the requirements, and are critical to the viability of work in the specified domain. Consequently, safety-critical and mission-critical problem domains are candidate areas for the use of DSMLs.

An integral aspect to the development of safety- and mission-critical systems is the verification of system correctness. This system verification may be accomplished in various format, with the most frequently used is that of formal specification techniques (FST) [2]. FSTs are methodologies wherein, mathematical representations of the system are developed for the purpose of carrying out rigorous analysis to identify deficiencies in the design of the system.

The reliable operation of safety-critical and mission–critical systems [3] are fundamental to the environment in which they are used. Some of these systems, apart from the issue of reliability, also must be developed following established standards and guidelines. One such application domain is that of the field of onboard avionic systems. Software system that are intended to be used in manned and soon to be established unmanned, aerial vehicles have to certified before they can be placed in their operational environment. This certification process sets out development processes, procedures, and artifacts that must be implemented before the software systems are approved.

Many of these software development standards and guidelines exist in the form of textual documents and must be understood by the developers in order to achieve the stated objectives. Fundamentally, as it is with software system requirements that exist in textual format, there may be ambiguities in the textual representations of these avionic software system development guidelines. Consequently, the developers are faced with this added complexity in carrying out their tasks.

The current standard for avionic software development that is used in the United States of America (USA) is the RTCA DO-178C “Software Consideration in Air-borne Systems and Equipment Certification” [4]. A corresponding EUROCAE (Europe-an Organization for Civil Aviation Equipment) guideline for software development, the ED 12, was development in conjunction with DO-178C for use in Europe.

The work presented in this report is composed of two phases. The first phase addresses the transformation of the DO-178C specification from a textual representation to a graphical representation, in the form of the UML (Unified Modeling Language [5]. The second phase of the work involves the definition of a model-driven object-oriented software development methodology that is in compliance with the DO-178C and incorporates FSTs for verification and validation of the system under development. The suite

Manuscript received March 20, 2018; revised April 5, 2018.
E. S. Grant is with the Department of Computer Science, University of North Dakota, Grand Forks, ND 58202-9015, USA (phone 701.777.4133, fax: 701.777.330, email: emanueel.grant@engr.und.edu).

of models is the DSML for avionic system development.
The remaining sections of this paper is described as follows. Section II provides a description of the research areas of the work done. Section III describes the work on transforming the DO-178C specification and the methodology defined. Section 4 describes three projects on which the methodology was applied and the outcomes. The next section gives a concluding statement and a look at future work in this area.

II. BACKGROUND

A. *Domain Description*

In this work the definition of safety-critical systems and the accompanying mission-critical systems are areas of software development are viewed with the same level of criticality but define separately. Both areas of software development garner greater resources and effort than other software application domains. Safety-critical systems are characterized as those systems that may result in harm or death to persons who are using they systems or on who the systems are being used, should the system fail. Mission-critical systems are characterized as those systems that an organization may experience significant financial, reputation, or resource loses, should such systems fail. The focus of this work is on the development of systems in a manner to minimize the possibilities of system failure by maximizing the software development activities of rigor and tractability at the requirements and design stages of development.

B. *Avionic Software Development Specification*

The RTCA DO-178C is the de facto standard guideline for avionic software development in the USA. A corresponding EUROCAE (European Organization for Civil Aviation Equipment) guideline for software development, the ED 12, was development in conjunction with DO-178C for use in Europe. The DO-178C describes the requirements for certification of software system in airborne operation but does not specify how these requirements are to be met. It is left up to the software developers to incorporate notations, processes workflows, and methodologies that comply with the objectives of DO-178C for certification. This standard supports the use of formal specification methods, objected oriented-methodologies, mode-based development and verification, and CASE tool usage. DO-178C had a predecessor, DO-178B, which has been retired with the release of DO-178C in 2012. DO-178B was last updated in 1992 and did not address many of the current software development methodologies, specifically objected-oriented, model-based development, and formal specification. The DO-178C was codified to address those and other deficiencies in DO-178B. The DO-178A addresses the certification of electronic hard-ware for use in airborne operation.

C. *UML*

Unified modeling language is a standard modeling language used to visualize, specify, construct, and document the artifacts of software intensive systems [5]. Diagrams are categorized in UML as structure and behavior diagrams. Structure diagrams represent static compositions of a system e.g. Class, Component, Object, Deployment, and Package diagrams. Behavior diagrams describe dynamic features of a system e.g. Use-case, Activity, and State diagrams. The expressive nature, informality, user-friendliness, and comprehensive diagrams are widely used to design various critical systems.

Several UML diagrams are used in this research in order to show the usefulness of them to depict a system from high-level to low-level descriptions. Package diagrams are used to portray the high-level view of a system while activity diagrams portray the activities in detail at granular level, and class diagrams are a set of classifiers. Class package and activity diagram are used to represent the DO-178C specification.

D. *Formal Specification Techniques*

Formal specification techniques (FSTs) use mathematical concepts to describe software systems with precision through rigorous analysis [7]. The use of FSTs is not a substitute for graphical software models; they are complementary. While formal models reveal inconsistencies and omissions, the informal model is an explicable version of the formal models. The specification language chosen in this work is Z notation.

The excessive cost during the implementation and early test phases are most times caused by errors in specification and design phases [6]. A specification written in a FS notation models the proposed system by naming the components of the system and expressing constraints between those components. Its formal basis enables mathematical reasoning, and hence proves that desired properties are consequences of the specification [6]. From these proofs, it can be determined whether the system will behave in a desirable manner; assuming the specification is accurate and complete.

III. METHODOLOGY

A. *Phase I*

In conducting the first phase of the research work, i.e. transforming the DO-178C document from its textual representation to a graphical representation, a series of UML diagrams are developed which comply with the DO-178C specification. The specifications are carefully converted from high-level to low-level models in a hierarchical manner.

The DO-178C specification for avionic software development is made up of twelve sections, two annexes, and two appendices. To develop a graphical representation of DO-178C specification a series of UML Package, Class, and Activity diagrams were developed. The Package diagrams were used to capture the high-level descriptions of the specification in the first phase of the work. Figure 1 illustrates the upper-most package diagram which presents the relationships between the twelve sections of the document.

The numerical references in the packages and activities of the graphical representations of the DO-178C specification, contained in the figures: are the section numbers from the DO-178C specification and have been added for traceability. Each package of Fig. 1 is decomposed into a lower-level package diagram, as illustrated in Fig. 2, wherein the DO-

178C Software Planning Process 4.0 UML Package Mode is presented. This is done for each package of each model, until all textual paragraph of the DO-178C specification is represented as a component of a UML class diagram or an activity diagram.

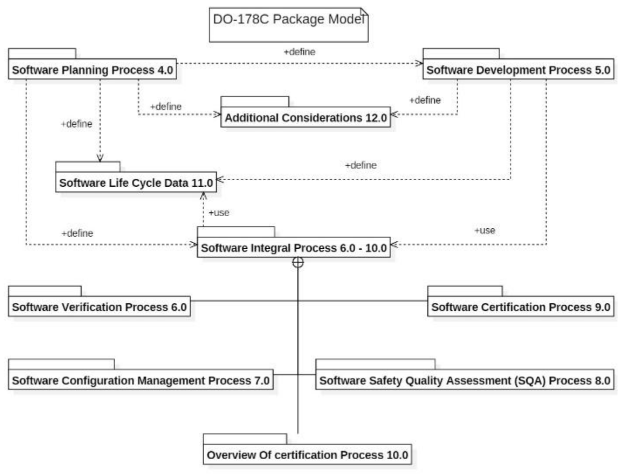

DO-178C Package Model
Software Planning Process 4.0
+define
Software Development Process 5.0
+define
+define
+define
Additional Considerations 12.0
+define
Software Life Cycle Data 11.0
+use
+define
+use
Software Integral Process 6.0 - 10.0
Software Verification Process 6.0
Software Certification Process 9.0
Software Configuration Management Process 7.0
Software Safety Quality Assessment (SQA) Process 8.0
Overview Of certification Process 10.0

Fig. 1. DO-178C High-Level Package Diagram

The stereotypes <<data item>> are decomposed into UML class diagrams, and the <<process>> stereotypes are decomposed into activity diagrams, which are the focus of the first phase of the work.

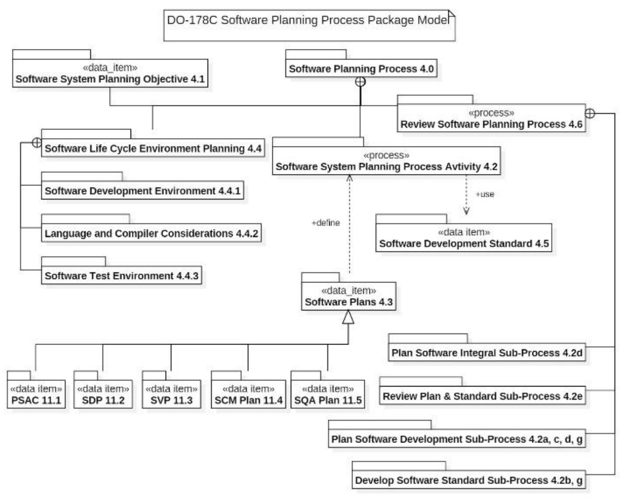
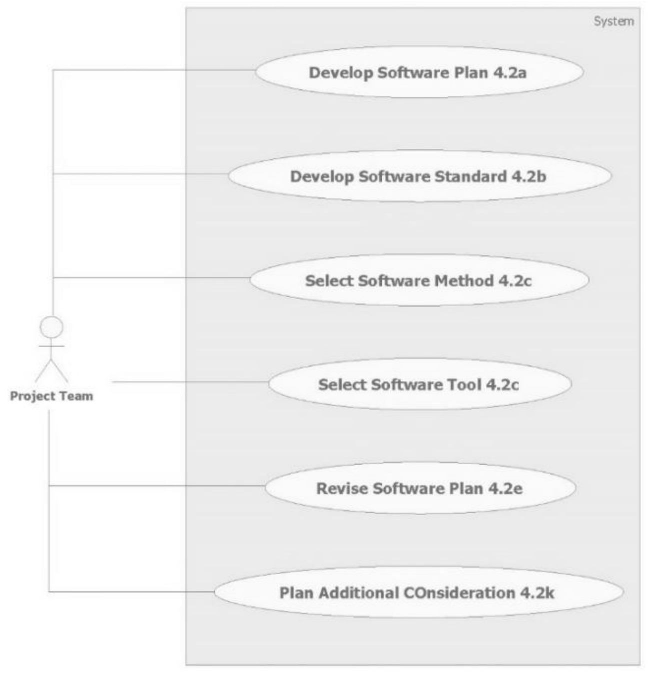

DO-178C Software Planning Process Package Model
«data_item»
Software System Planning Objective 4.1
Software Planning Process 4.0
«process»
Review Software Planning Process 4.6
Software Life Cycle Environment Planning 4.4
«process»
Software System Planning Process Avtivity 4.2
Software Development Environment 4.4.1
+use
+define
Language and Compiler Considerations 4.4.2
«data item»
Software Development Standard 4.5
Software Test Environment 4.4.3
«data_item»
Software Plans 4.3
Plan Software Integral Sub-Process 4.2d
«data item»
PSAC 11.1
«data item»
SDP 11.2
«data item»
SVP 11.3
«data item»
SCM Plan 11.4
«data item»
SQA Plan 11.5
Review Plan & Standard Sub-Process 4.2e
Plan Software Development Sub-Process 4.2a, c, d, g
Develop Software Standard Sub-Process 4.2b, g

Fig. 2. DO-178C Software Planning Process 4.0 UML Package Model

Fig. 3 represents the DO-178C Software Planning Process (Section 4 of the DO-178C specification) as an UML Use Case Diagram, wherein the user is the project development team. Fig. 3 is decomposed into a set of UML use case diagrams, class diagrams, and activity diagrams. Fig. 3 is one of the models contained in the Software Planning Process 4.0 of the Fig. 1 package-level model

B. Phase II

The methodology of the research is presented as a UML activity diagram, in Fig 4. The format of the activity diagram includes the related DO-178C specification section number for traceability reference. The "Conduct Software Requirement 5.0" and "Conduct Software Requirement Process 5.1" are the sub-activity events that are fundamental to the successful software development at the early stage of the methodology. The models that are output from the "Conduct Software Requirement Process 5.1", namely, UML Requirement Class Diagram, Use Case Diagram, and Use Case, are then input to the "Conduct High-Level Design 5.2.2" process. These models will be refined and transformed during this and later stages of the methodology. It is to be noted that the models of Fig 4 are an instantiation of the methodology; for other application domains other


System
Develop Software Plan 4.2a
Develop Software Standard 4.2b
Select Software Method 4.2c
Project Team
Select Software Tool 4.2c
Revise Software Plan 4.2e
Plan Additional COnsideration 4.2k

models and formal specification notation may be used.

Fig. 3: DO-178C Software Planning Process 4.0 UML Use Case Diagram Representation

Fig. 5 elaborates the requirement sub-activities of Fig. 4. The input to the sub-activity (Acquire Requirement 5.1.2) are the "Software Development Plan 11.2" and "Software Requirement Standard 11.6" outputs from the prior sub-activity of Fig 4, namely, "Conduct Software Planning 4.0".

As discussed earlier, various UML diagrams are created to implement Model Driven Development approach for different modules and sub-modules of DO-178C. In an earlier study [7], modules 4.0, 5.0 and 7.0 of DO-178C specification were selected for transformation.

C. Formal Specification Techniques

Formal specification techniques (FSTs) employ mathematical concepts to represent software systems with precision to conduct rigorous analysis [2]. The use of FSTs is not a substitute for modeling software systems ass graphical models; they are complementary. While formal models can reveal errors I the requirement specification and system design, the informal model is a visually understandable representation of the software system.

The formal specification notation used in this work is the Z notation [8]. A specification written in the Z notation, models the system design by representing the components of the system and expressing constraints on and between those components. Its formal basis enables mathematical reasoning, and hence proves that desired properties are consequences of the specification [6]. From these proofs, the system's behavior is assessed to be in a desirable or undesirable state.

System behavior should always be deterministic in the domain of safety critical systems. These software systems encompass highly complex processing and have a high demand for reliability and accuracy. Due to the continuous use of UML in software development, there is a need to resolve the informal semantics of the models it produces. Transforming UML models into Z equivalences also provide formal analysis to accomplish verification and validation of software systems.

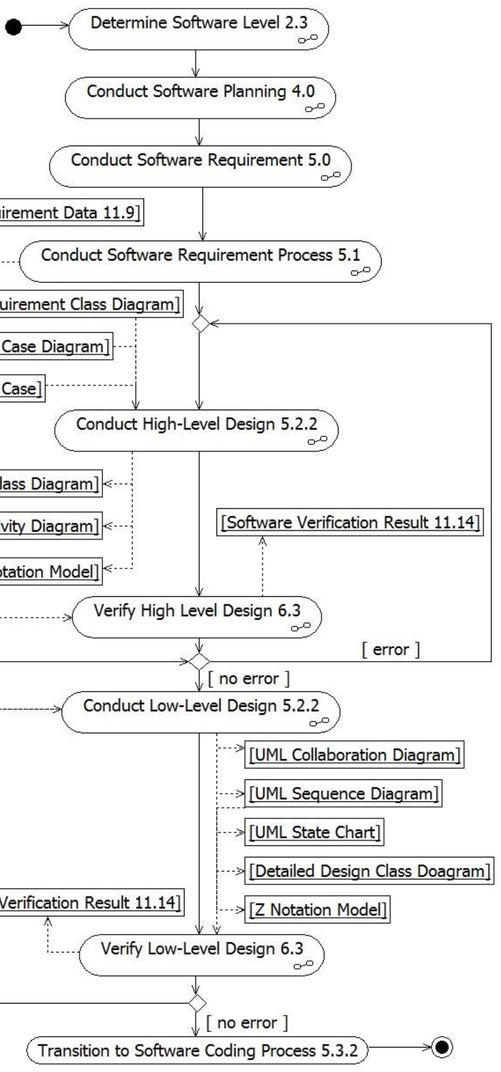

Determine Software Level 2.3
Conduct Software Planning 4.0
Conduct Software Requirement 5.0
[Software Requirement Data 11.9]
Conduct Software Requirement Process 5.1
[UML Requirement Class Diagram]
[UML Use Case Diagram]
[UML Use Case]
Conduct High-Level Design 5.2.2
[Design Level Class Diagram]
[UML Activity Diagram]
[Z Notation Model]
[Software Verification Result 11.14]
Verify High Level Design 6.3
[error]
[no error]
Conduct Low-Level Design 5.2.2
[UML Collaboration Diagram]
[UML Sequence Diagram]
[UML State Chart]
[Detailed Design Class Doagram]
[Z Notation Model]
[error]
[Software Verification Result 11.14]
Verify Low-Level Design 6.3
[no error]
Transition to Software Coding Process 5.3.2

Fig. 4. UML Activity Diagram of the Research Methodology

IV. RESULTS

A. Case Study I

The reported work has it genesis on a project at the University of North Dakota (UND) for the development of an air-worthiness system for the operation of un-manned aerial vehicles (UAV) [7]. With the increasing use of UAVs in military and civilian areas there is an increasing need to develop and advance the reliability, availability, and performance of these safety-critical systems. An obstacle to the use of UAS is the interaction between UAV and manned aerial vehicles (MAV). Towards achieving integration of UAV and MAV flights there is the need for systems that ensures the possibility of an incident between these vehicles be the same as or better than that which now exists for MAVs operation [4]. The UND – UAS Risk Mitigation Project was started to address this problem. This project supported UAV experimentation and training, and assistance to civilian authorities.

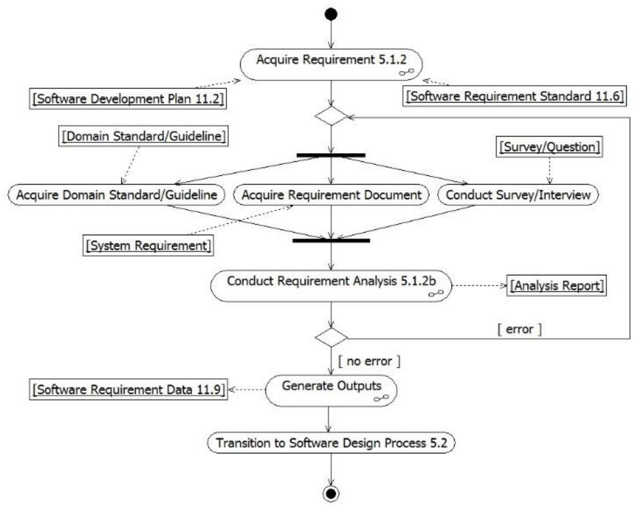
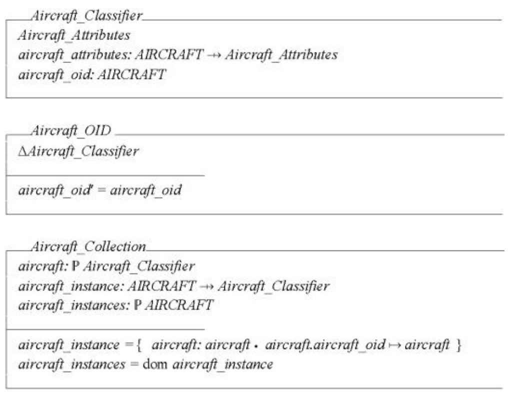

Acquire Requirement 5.1.2
[Software Development Plan 11.2]
[Software Requirement Standard 11.6]
[Domain Standard/Guideline]
[Survey/Question]
Acquire Domain Standard/Guideline
Acquire Requirement Document
Conduct Survey/Interview
[System Requirement]
Conduct Requirement Analysis 5.1.2b
[Analysis Report]
[error]
[no error]
[Software Requirement Data 11.9]
Generate Outputs
Transition to Software Design Process 5.2

Fig. 5. UML Activity Diagram of Research Requirement Processes

The UND – UAS Risk Mitigation Project system was made up of three core components; a, radar system, a data computation unit, and a display system. The display system software is the focus of the work for which the methodology of this work was defined.

This work resulted in the identification of multiple errors in the initial design models by means of the formal specification technique (FST) that was applied at the verification phase of the model-driven software development methodology employed on the project. In a safety-critical environment, these software design errors could result in catastrophic failure of the operating system. Examples of Z formal specification models developed on this project are presented in Fig. 6.

The display system was made up of UML class diagram of: 174 classes, 2,250 attributes, 383 associations, 580 operations (methods) and, 268 parameters. A subset of this class diagram that contained 9 classes with a combined total of 455 attributes, 16 associations and their multiplicities, and 56 operations were transformed to a Z notation representation. This derived 206 paragraphs in Z/EVES, which included the declaration of schemas, basic types, and axiomatic definitions.


```
Aircraft_Classifier
Aircraft_Attributes
aircraft_attributes: AIRCRAFT ⇸ Aircraft_Attributes
aircraft_oid: AIRCRAFT

Aircraft_OID
ΔAircraft_Classifier
aircraft_oid' = aircraft_oid

Aircraft_Collection
aircraft: ℙ Aircraft_Classifier
aircraft_instance: AIRCRAFT ⤔ Aircraft_Classifier
aircraft_instances: ℙ AIRCRAFT
aircraft_instance = {  aircraft: aircraft •  aircraft.aircraft_oid ↦ aircraft  }
aircraft_instances = dom aircraft_instance
```

Fig. 6. Example of Z notation model for UAS System [7]

B. *Case Study II*

A commercial airline company, as a part of its operation review, identified a problem in its information system structure. The company encountered what was identified as a “single-point of failure” in the process for dynamic assignment of aircrafts to airport terminal gates. That single point of failure in the process was the reliance on a single specific operator to conduct dynamic assignment of aircrafts to terminal gates. The process involves the listing of all aircrafts for assignment and the available gates. Aircrafts are classified based on certain attributes, such as size, capacity, manufacturer, arrival time, departure time, etc. Gates are classified based on certain attributes, such as, location to runway, fuel-port, accessibility, availability time, etc. Other constraints pertain to global considerations, such as available runway, taxiway path to runway, established departure timeframes, etc. [9].

The operator would compile the aircraft and gate lists and generate a standard assignment, based on the previous assignment cycle. The existing software system would then identify any assignment conflicts, which may arise from gate closures, incompatible aircraft-gate assignment aircraft late or none arrival, etc. The operator would then attempt to resolve the assignment conflicts by reassigning aircrafts based on prior experience of executing this process. Whenever that operator is unavailable, the new operator would conduct the same operation, but the resolution would be based on his experience.

The company recognized the failure that may arise if this system and process were not improved to be more efficient and effective. Consequently, a team of researchers from the University of North Dakota (UND) departments of Aviation and Computer Science were asked to look at the problem and develop a plan to mitigate the potentially problematic system and process. The UND team included researchers in genetic algorithm design and software engineering from the Department of Computer Science; it is the software engineering researchers’ work, which is specifically documented in this report.

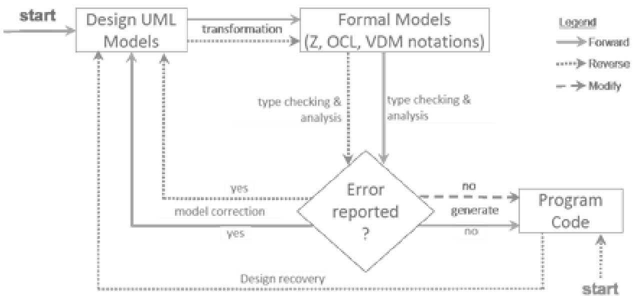


Fig. 7: Reverse-Engineering Modified Model-Driven Methodology

The software system documentation team opted to identify this system as a Level-A DO-178C system, in order to exercise as many of the model-driven methodology’s activities, as represented in Figure 4. The team’s effort was centered on that of reverse engineering a set of UML models of the genetic algorithm system for the purpose of verification, validation, and system documentation. A reverse engineering method was developed that overlaid the forward engineering methodology of Fig. 4, which is presented in Fig. 6.

The framework of Fig.7 was developed to incorporate a reverse-engineering strategy to complement the forward-engineering activities. Fig. 7 also illustrates the use of formal specification techniques for validating the reverse and forward engineering activities. The “Design UML Models” activity of Fig.7 is reflective of the Conduct High Level Design 5.2.2” and “Conduct Low-Level Design 5.2.2” of Fig. 4, and the “Formal Models” activity of Fig.7 is synonymous to the “Verify High Level Design 6.3” and “Verify Low-Level Design 6.3” activities of Fig. 4. The green (solid) arrowed lines represent the forward engineering path through the process model, while the red (broken) arrowed lines represent the reverse engineering path through the model. The forward engineering process commenced with the “Design UML Models” activities, while the reverse engineering process commenced at the “Program Code” activity.

The main UML model developed by the team was a set of activity diagrams that was implemented at the detailed-level of system modelling; an example is presented in Fig. 8. The limitation to producing just one type of UML model was borne out of the airline system administrators’ preference for just the necessary models to facilitate any immediate small-scale bug fixes, versus models to be used for system evolution. The nature of the contract between UND and the airline called for the software system’s on-going maintenance (evolution) to be further contracted out to a third party.

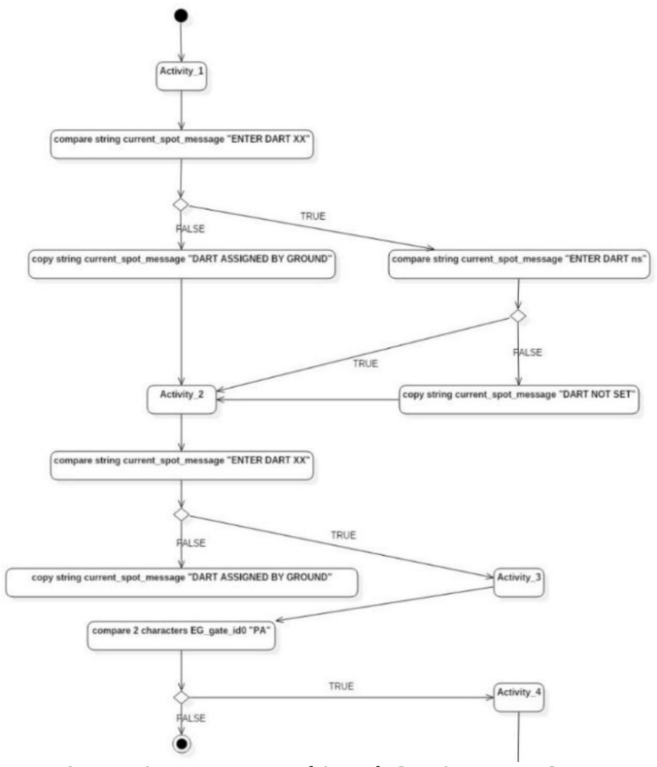

Activity_1
compare string current_spot_message "ENTER DART XX"
TRUE
FALSE
copy string current_spot_message "DART ASSIGNED BY GROUND"
compare string current_spot_message "ENTER DART ns"
TRUE
FALSE
Activity_2
copy string current_spot_message "DART NOT SET"
compare string current_spot_message "ENTER DART XX"
TRUE
FALSE
copy string current_spot_message "DART ASSIGNED BY GROUND"
Activity_3
compare 2 characters EG_gate_id0 "PA"
TRUE
Activity_4
FALSE

Fig. 8: UML Activity Diagram of Aircraft-Gate Assignment System

V. CONCLUSION

This report documents the work in transforming a specification, the RTCA DO-178C, for safety-critical system software development, from a textual representation to a graphical representation; in the form of the UML notation. Complementing this work, was the development of a model-driven software development methodology that is compliant with the DO-178C specification that is used in conjunction with the DO-178C graphical representation to design and develop software systems for onboard avionic operations. The output of this research effort is a DSML for avionic safety-critical software systems that is compliant with the RTCA DO-178C specification.

This work is primarily focused on the development of safety-critical systems that are now ubiquitous in daily life. Such systems appear in many fields, such as the medical, transportation, and home-care, products and services that are used in private. Professional, individual, and group spheres worldwide. The reliability of these systems is paramount to not only the users of these systems, but also to the future application of software systems in new domains. The increasing need for more reliable software systems will continue to grow and the need for more reliable software systems keeps pace.

This research effort is ongoing, with the research work being conducting to develop tools to automate aspects of using the DSML in application development. This future research effort is intriguing, as the tool (CASE) development effort will also have to abide by the DO-178C specification

REFERENCES

[1] R. E. Faith, L. S. Nyland, and J. F. Prins. “KHEPERA: A system for rapid implementation of domain specific languages. In Proceedings of the Conference on Domain-Specific Languages, 1997.

[2] R. B. France, A. Evans, K. Lano,, B. Rumpe “The UML as a Formal Modeling Notation” Computer Standards & Interfaces, vol 19, issue 7, 325—334, 1998.

[3] D. J. Smith,K. GL. Simpson, The Safety Critical Systems Handbook. Elsevier, MA: Cambridge, 2016.

[4] SC-205, Software Consideration in Air-borne Systems and Equipment Certification. Washington DC: RTCA, Inc., 2011.

[5] ISO/IEC 19501, Unified Modeling Language (UML) Information Technology - Open Distributed Processing, Version 1.4.2 (2005).

[6] R. B. France, J. Bruel, M. M. Larrondo-Petrie, “An Integrated Object-Oriented and Formal Modeling Environment”. In Proceedings of JOOP. 25--34. 1997.

[7] E. S. Grant, V. Jackson, S. Clachar, “Towards a Formal Approach to Validating and Verifying Functional Design for Complex Safety Critical Systems”, 2nd Annual International Conference on Software Engineering & Applications (SEA 2011), Singapore, GSTF, 2011.

[8] ISO/IEC 13568, “Information Technology: Z Formal Specification Notation - Syntax, Type System and Semantics” First ed. ISO/IEC 2002.

[9] E. S. Grant, P. Ajjimaporn, “Pedagogical Benefits from an Exercise in Reverse Engineering for an Aviation Software Systems” 10th International Conference on Computer Supported Education (CSEDU- 2018), Madeira, Portugal, 2018.