Model Checking: Cleared for Take Off

Darren Cofer

Rockwell Collins, Advanced Technology Center
400 Collins Rd. NE
Cedar Rapids, IA 52498
ddcofer@rockwellcollins.com

Abstract. The increasing popularity of model-based development tools and the growing power of model checkers are making it practical to use formal methods for verification of avionics software. This paper describes a translator framework that enables model checking tools to be easily integrated into a model-based development environment to increase assurance, reduce cost, and satisfy certification objectives. In particular, we describe how formal methods can be used to satisfy certification objectives of DO-178C/ED-12C, the soon-to-be-published guidance document for software aspects of certification for commercial aircraft.

Keywords: model checking, verification, certification, avionics

1 Introduction

Modern commercial aircraft contain millions of lines of complex software, much of it performing functions that are critical to safe flight. This software must be verified to function correctly with the highest levels of assurance, and aircraft manufacturers must demonstrate evidence of correctness through a rigorous certification process. Furthermore, the size and complexity of the on-board software are rising exponentially. Current test-based verification methods are becoming more expensive and account for a large fraction of the software development cost. New approaches to verification are needed to cope effectively with the software being developed for next-generation aircraft.

Formal analysis methods such as model checking permit software design models to be evaluated much more completely than is possible through simulation or test. This permits design defects to be identified and eliminated early in the development process, when they have much lower impact on cost and schedule. Advances in model checking technology, the adoption of model-based software development processes, and new certification guidance are enabling formal methods to be used by the aerospace industry for verification of software.

This paper provides an overview of our work applying model checking to the development of software for commercial and military aircraft. Model checking is being used to provide increased assurance of correctness, reduce development cost, and satisfy certification objectives. We also discuss the new certification guidance

supporting the use of formal methods that will be included in DO-178C, the industry standard governing software aspects of aircraft certification.

2 Model Checking and Model-Based Development

Model-based development (MBD) refers to the use of domain-specific modeling notations such as Simulink or SCADE to create detailed software designs that can be evaluated for desired behavior before a system is built. MBD environments allow the engineer to create a model of the system early in the lifecycle that can be executed on the desktop, analyzed for desired behaviors, and then used to automatically generate code and test cases. The emphasis in model-based development is to focus the engineering effort on the early lifecycle activities of modeling, simulation, and analysis, and to automate the later lifecycle activities of coding and testing.

Formal methods may be applied in a MBD process to eliminate requirements, design, and coding errors, and should be viewed as complementary to testing. While testing shows that functional requirements are satisfied for specific input sequences and detects some errors, formal methods can be used to increase confidence that a system will always comply with particular requirements when specific conditions hold. Informally we can say that testing shows that the software does work for certain test cases while formal methods show that it should work for all cases. It follows that some verification objectives may be better met by formal, analytical means and others might be better met by testing.

Although formal methods have significant technical advantages over testing for software verification, they are only just beginning to be used in the aerospace industry. The additional cost and effort of creating and reasoning about formal models in a traditional development process has been a significant barrier. Manually creating models solely for the purpose of formal analysis is labor intensive, requires significant knowledge of formal methods notations, and requires that models and code be kept tightly synchronized to justify the results of the analysis.

The value proposition for formal methods changes dramatically with the introduction of MBD and the use of automated analysis tools. Many of the notations in MBD have straightforward formal semantics. This means that it is possible to use models written in these languages as the basis for formal analysis, removing the incremental cost for constructing and updating separate verification models.

In collaboration with the University of Minnesota under NASA's Aviation Safety Program, Rockwell Collins has developed a translation framework that bridges the gap between some of the most popular industrial MBD languages and several model checkers (Fig. 1). These automated tools allow us to quickly and easily generate models for verification directly from the design models produced by the MBD process [1]. The counterexamples generated by model checking tools can be translated back to the MBD environment for simulation. This tool infrastructure provides the means for integration of formal methods directly and efficiently into the MBD process. Software engineers can continue to develop design models using the tools that they are already familiar with.

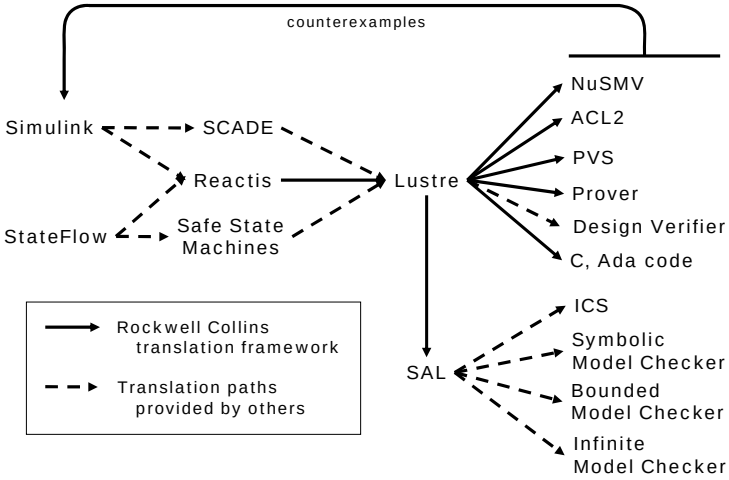

Fig. 1. Rockwell Collins translation framework.

The translators use the Lustre formal specification language, developed by the synchronous language research group at Verimag, as an intermediate representation for the models [2]. Models developed in Simulink, StateFlow, or SCADE are transformed into Lustre. Once in Lustre, the specification is loaded into an abstract syntax tree (AST) and a number of transformation passes are applied to it. Each transformation pass produces a new Lustre AST that is syntactically closer to the target specification language and preserves the semantics of the original Lustre specification (Fig. 2). This allows all Lustre type checking and analysis tools to be used after each transformation pass. When the AST is sufficiently close to the target language, a pretty printer is used to output the target specification. This customized translation approach allows us to select the model checker whose capabilities are best suited to the model being analyzed, and to generate an analysis model that has been optimized to maximize the performance of the selected model checker.

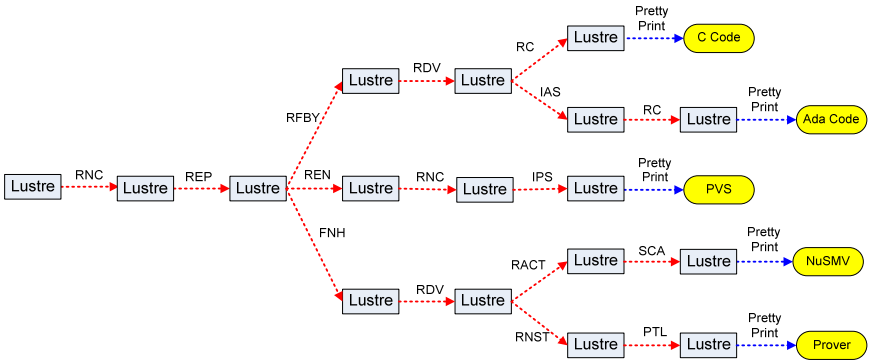

Fig. 2. Illustration of automated transformation steps used to translate Lustre models.

Since Lustre is the underlying language for SCADE models, the initial translation step is immediate. For Simulink and StateFlow models, we use the Reactis test case generator tool to support the initial translation step. We also use the Reactis simulator as the primary means for playback of the counterexample test cases. To ensure that each Simulink or Stateflow construct has a well-defined semantics, the translator restricts the models that it will accept to those that can be translated unambiguously into Lustre.

Our translation framework is currently able to target eight different formal analysis tools. Most of our work has focused on the NuSMV model checker and the Prover model checker. We can also use the same translation framework to generate C or Ada source code.

3 Benefits of Model Checking

The potential benefits of using formal methods, including model checking, are well-known. In this section we focus on three benefits and how they relate to the aerospace industry.

The traditional justification for the use of formal methods has been to provide increased assurance of correctness, especially for systems or components that implement safety-critical functions. Model checking excels in this area, providing comprehensive exploration of system behavior and exposure of design errors.

However, the strongest motivation for adoption of model checking in the industry seems much more likely to be cost reduction. The ability to detect and eliminate defects early in the development process has a clear impact on downstream costs. Errors are much easier and cheaper to correct in the requirements and design phases than during subsequent implementation and integration phases.

An additional benefit which may become increasingly important is the ability to satisfy certification objectives through the use of formal methods, including model checking. As the first two benefits have been described in detail elsewhere, we will touch on these briefly and devote most of the remainder of the paper to the use of formal methods as part of the certification process.

3.1 Increased Assurance

Model checking performs a comprehensive evaluation of system behavior over all reachable states and allowable inputs. This provides much more effective error discovery capability compared with testing.

As an illustration, in the Certification Technologies for Flight Critical Systems (CerTA FCS) project funded by the U.S. Air Force, we analyzed several software components of an adaptive flight control system for unmanned aircraft [3]. In this project we analyzed the redundancy manager software which implements a triplex voting scheme for fault-tolerant sensor inputs. We performed a head-to-head comparison of verification technologies with two separate teams, one using testing

and one using model checking. Neither team communicated directly with the other, and both teams started with identical software models and requirements to be verified.

Both teams developed extensions to their base verification technologies. The model checking team extended their existing tools to add support for several new block types found in the software. Likewise, the testing team also made comparable investments in enhancing their testing environment. These one time, non-recurring costs were not included in the final comparison of the effectiveness of testing and model checking.

The model checking team developed a total of 62 properties for analysis from the original software requirements. Analysis of these properties with the model checker uncovered 12 errors in the redundancy management logic. In similar fashion, the testing team developed a series of tests cases from the same set of software requirements. However, testing failed to find any errors in the software.

The conclusion of both teams was that in this case study, model checking was more effective than testing in finding design errors. Some of the errors found by the model checking team would be difficult, if not impossible, to discover through testing. For example one such error involved a complex timing interaction between the inputs to the voter which resulted in a good sensor being declared faulty.

3.2 Reduced Cost

A key benefit of using model checking in an industrial context turns out to be cost savings. Savings can be achieved through early detection and elimination of errors as well as through automation.

Our first application of model checking to an actual product was the mode logic of the Rockwell Collins FCS 5000 Flight Control System used in business and regional jet aircraft [4]. The mode logic determines which lateral and vertical flight modes are armed and active at any time. Analysis of an early specification of the mode logic found 26 errors. Seventeen of these were found by the model checker. Of these 17 errors, 13 were classified by the FCS 5000 engineers as being possible to be missed by traditional verification techniques such as testing and inspections. One was classified as being unlikely to be found by traditional techniques. The ability to eliminate these errors during modeling, as opposed to during testing in the lab (or worse, during aircraft integration testing) results in significant savings.

In a more recent example, we used our translation and model checking tools to analyze the leader selection software for a multi-node redundant flight control system. The selection logic was implemented using Simulink/Stateflow and its basic functionality validated through simulation. The design was then analyzed using model checking and improved to eliminate the counterexamples identified. The verified design was then autocoded and tested on prototype hardware. The implementation achieved 100% successful test case passage on the first attempt. Eliminating the need for rework cycles to correct errors found during lab testing may have reduced development time by half.

In the CerTA FCS project discussed above, we discovered that not only was model checking more thorough, it was actually less costly than verification through testing.

The testing team required 50% more time to develop and execute the required test cases compared to the time needed to formalize properties and analyze them with the model checker. Certainly this may not always be the case, but this experiment demonstrates that the degree of automation possible in an MBD environment makes it possible to perform model checking very efficiently.

3.3 Certification Credit

A third benefit of using formal methods is the evidence that can be provided in satisfaction of certification objectives. Certification can be defined as legal recognition by a certification authority (usually governmental) that a product, service, organization, or person complies with specified requirements. In the context of commercial aircraft, certification consists primarily of convincing the relevant certification authority (the FAA in the U.S. or EASA in Europe) that all required steps have been taken to ensure the safety, reliability, and integrity of the aircraft. Software itself is not certified in isolation, but only as part of an aircraft. Certification differs from verification in that it focuses on evidence provided to a third party to demonstrate that the required activities were performed completely and correctly, rather on performance of the activities themselves.

For software in commercial aircraft, the relevant certification guidance is found in DO-178B, "Software Considerations in Airborne Systems and Equipment Certification" (known in Europe as ED-12B) [5]. Certification authorities in North American and Europe have agreed that an applicant (aircraft manufacturer) can use this guidance as a means of compliance with the regulations governing aircraft certification.

The original version of the document, DO-178, was approved in 1982 and consisted largely of a description of "best practices" for software development. It was revised in 1985 as DO-178A, adding definitions of three levels of software criticality, with development and verification processes described in more detail. The current version, DO-178B, was approved in 1992. It defines five levels of software criticality (A – E) with specific objectives, activities, and evidence required for each level.

DO-178B allows for the use of formal methods to satisfy certification objectives, but it does so as an "Alternative Method." The processes and objectives in the document assume a traditional development process with test-based verification.

In 2005, RTCA and EUROCAE (the publishers of the DO-178/ED-12 standards) initiated work on a revision to be known as DO-178C/ED-12C. A committee (SC-205) was chartered to draft the new document, with the objectives of minimizing changes to the core document, yet updating it to accommodate approximately 15 years of progress in software engineering. Guidance specific to new software technologies was to be contained in "supplements" which could add, modify, or replace objectives in the core document. New supplements are being developed in the areas of tool qualification, object oriented design, model-based development, and formal methods. The current schedule calls for DO-178C to be approved by the end of 2010.

The inclusion of formal methods as a means of compliance with its own technology supplement (rather than an "alternative method") will open the door to aircraft manufacturers obtaining certification credit through the use of formal verification techniques including model checking. In the next section we describe the new certification guidance related to the use of formal methods.

4 DO-178C: New Certification Guidance

DO-178B does not prescribe a specific development process, but instead identifies important activities and design considerations throughout a development process and defines objectives for each of these activities. It assumes a traditional development process producing a collection of lifecycle data items that can be decomposed as follows:

- Software Requirements Process. Develops High Level Requirements (HLR) from the output of the system design process.
- Software Design Process. Develops Low Level Requirements (LLR) and Software Architecture from the HLR.
- Software Coding Process. Develops source code from the software architecture and the LLR.
- Software Integration Process. Combines executable object code modules with the target hardware for hardware/software integration.

The lifecycle data items and the processes that accomplish these transformations are shown in Fig. 3. The results of these processes are verified through the verification process. The verification process consists of review, analysis, and test activities that must provide evidence of the correctness of the development activities. The arcs in Fig. 3 correspond to verification activities and the labels identify the objectives for each activity. In addition, there are "verification of verification" objectives (not shown in the figure) to demonstrate the sufficiency of the verification activities themselves.

In general, verification has two complementary objectives. One objective is to demonstrate that the software satisfies its requirements. The second objective is to demonstrate with a high degree of confidence that errors which could lead to unacceptable failure conditions, as determined by the system safety assessment process, have been removed. As discussed in Section 3, formal methods can be used to meet these objectives – sometimes better than reviews or testing.

In drafting the Formal Methods Technology Supplement (FMTS) for DO-178C the committee had the following goals:

- Identify scope of applicability formal methods. Formal methods should no longer be treated as an "alternative method." Guidance should be provided regarding which objectives can be satisfied through formal methods and how that might be done. The focus of FMTS is the verification process and

associated activities and objectives. Partial use of formal methods is acceptable (applied to only some software, some requirements, or some objectives).

– Facilitate communication between applicants and certification authorities. FMTS should specify what evidence should be expected for satisfying objectives, what new process documentation is needed, and what additional or different activities are needed when using formal methods.
– Identify areas deserving of particular scrutiny when formal methods are used. FMTS should help to avoid common errors, and identify important questions that must be addressed during certification.
– Facilitate use of formal methods in the aerospace community. FMTS should not impose higher burdens than a traditional verification process, but it should also not do anything that would reduce the level of assurance provided by DO-178B.

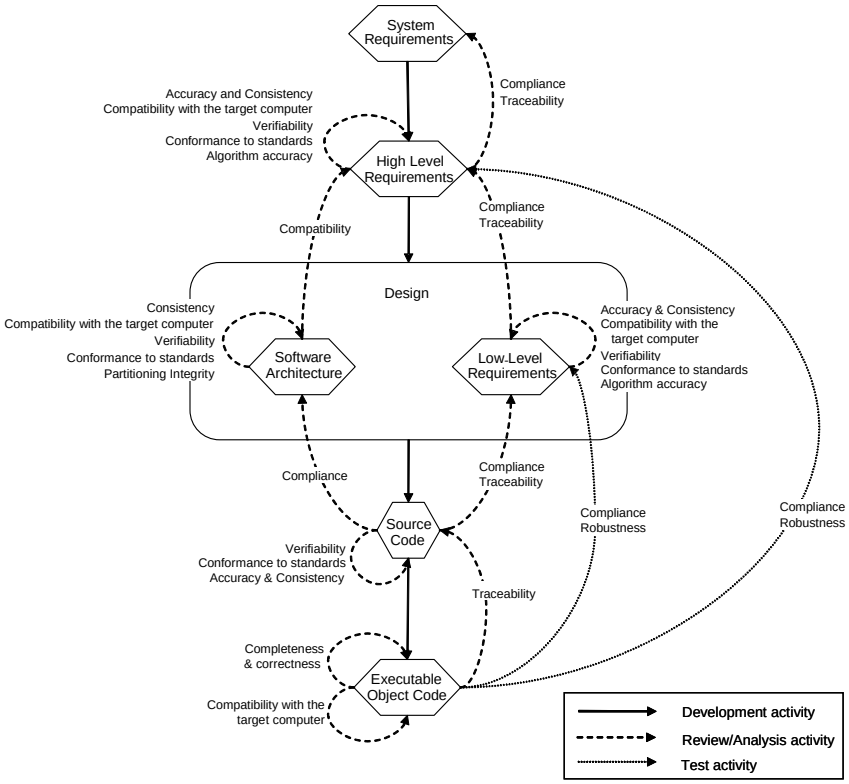

Fig. 3. DO-178 software development and verification activities.

General guidance is provided in FMTS that is applicable to the overall verification process when formal methods are used. This includes the following requirements:

- All formal notations used must have unambiguous, mathematically defined syntax and semantics.
- The soundness of each formal analysis method should be documented. A sound method never asserts that a property is true when it may not be true.
- All assumptions related to the formal analysis should be described and justified (e.g. assumptions about execution semantics on the target computer, or assumptions about data range limits).

Specific guidance is provided to describe how formal methods can be used to satisfy each of the common objectives for HLR and LLR shown in Fig. 3. These include compliance with requirements, accuracy and consistency of requirements, compatibility with the target computer, verifiability of requirements, conformance to standards, traceability between lifecycle data items, and algorithmic correctness.

A new objective to demonstrate requirements formalization correctness is defined. If a requirement has been translated to a formal notation as the basis for using a formal analysis, then review or analysis should be performed to demonstrate that the formal statement is a conservative representation of the informal requirement.

In addition, there is provision for some software testing to be replaced by formal analysis. DO-178B requires that test cases corresponding to the software requirements be produced and executed, and that adequacy (completeness) of these test cases be determined via structural coverage metrics. When using formal methods, verification is exhaustive so a requirement that has been verified formally has been completely covered. However, there is no guarantee that some requirement has not been omitted from the design. Four new objectives have been defined in the DO-178C FMTS to provide an equivalent level of assurance with regard to the adequacy of the formal verification activity.

5 Example: Model Checking for Certification Credit

In this section we show by means of an example how model checking could be used to satisfy some of the certification objectives in DO-178C with the FMTS.

In a modern aircraft, the primary way that aircraft status information is displayed to pilots is through computerized display panels. These display panels are designed to replace the dozens of mechanical switches and dials found in earlier aircraft and to present a unified and straightforward interface to critical flight information. The display panels are configurable to allow pilots to select different information for display, including navigational maps, aircraft system status, and flight checklists. However, some information is considered critically important and must always be displayed.

The Window Manager (WM) determines which applications should be displayed on which display area as well as the location of the cursor on the displays. It has several responsibilities related to routing information to the displays. First, the WM must update which applications are being displayed in response to user selections of display applications. Second, the WM must handle hardware or application failures. If a display fails, the WM decides which information is most critical and moves this

information to the remaining display. Another responsibility has to do with cursor management: some display applications support the cursor while others do not. It is the responsibility of the WM to ensure that the cursor does not appear on a display that contains an application that does not support the cursor. In the event of a failure, the WM must ensure that the cursor is not tasked to a dead display. A top-level model of a simplified Window Manager is shown in Fig. 4.

The WM is essential to the safe flight of an airplane. If the WM contains logic errors, it is possible that critical flight information will be unavailable to the flight crew. Hence it is required to meet the Level A objectives of DO-178B.

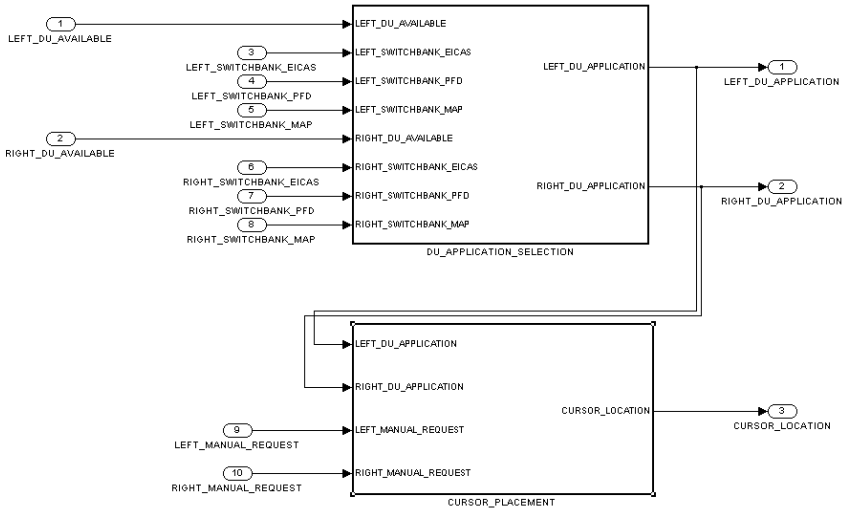

Fig. 4. Top-level Simulink model of a simplified Window Manager.

The WM verification effort [6] was conducted several years ago, before any consideration of incorporation of formal methods guidance in DO-178C. At that time, the primary objective was to identify and remove design errors early in the development process. The comprehensive analysis provided by model checking resulted in a higher assurance of correct behavior than test-based verification. Detection and correction of errors earlier in the development process (during design rather than test and integration) reduced the overall development cost.

The WM software was developed using a MBD process consisting of the following major activities:

- HLRs were initially expressed as English “shall” statements that were subsequently formalized as CTL for analysis.
- Software models were developed using model-based design tools (Simulink and Stateflow), and correspond to LLRs.
- The LLR models were analyzed using a model checker to verify whether or not they satisfy the HLRs.

– Source code was automatically generated from the LLRs and tested in conformance with a conventional test-based process.

Approximately 90% of the functional behavior of the WM application (in terms of the number of Simulink blocks) was verified using model checking. The remaining 10% of the model is in one subsystem that contains a significant number of floating point variables. This subsystem does not contain much mode-specific behavior and was verified using conventional methods.

The WM was divided into five subsystem models that were used for analyzing its behavior. Table 1 provides an overview of these subsystems and the analysis results.

Table 1. Window Manager analysis results.

Subsystem	Simulink Diagrams	Simulink Blocks	State Space	Properties	Errors found
GG	2,831	10,669	9.8×10^{9}	43	56
PS	144	398	4.6×10^{23}	152	10
CM	139	1,009	1.2×10^{17}	169	10
DUF	879	2941	1.5×10^{37}	115	8
MFD	302	1,100	6.8×10^{31}	84	14
Totals	**4295**	**16,117**	**n/a**	**563**	**98**

The above results show that formal analysis can be applied to large commercial software systems. The 98 errors found resulted in changes to the LLR models or changes to the HLRs. The corrected HLRs and LLRs were re-analyzed and found to be compliant.

The formal methods technology developed in the project was successfully transitioned to the product development organization. By the end of the project, all analysis work was being performed by Rockwell Collins software engineers, with minimal assistance from researchers.

With the new guidance provided in the Formal Methods Technology Supplement to DO-178C, many certification objectives could have been satisfied. Some examples follow.

FM6.2 Software Verification Process Activities

a. Formal notations: Properties to be verified were specified in CTL. Formal definition of CTL may be found in [7]. The models analyzed were specified in Simulink and Stateflow. These models were given formal definition through the translation process, which includes a formal syntax and translation rules for each model element.

b. Soundness: The BDD and SAT algorithms used in the model checker are known to be sound. Details of the BDD algorithm used for model checking and its soundness can be found in [8]. Application of satisfiability solving to the model checking problem and its soundness are described in [9].

c. Assumptions: Any assumptions on the subsystem inputs necessary for the analysis were documented and justified.

FM6.3 Software Reviews and Analysis

i. Requirement formalization correctness: In this project, all requirements were captured and managed using the DOORS tool. For each requirement, the corresponding formalization was captured in DOORS with one or more CTL statements. Multiple independent reviews were conducted to ensure that the CTL statements accurately described the original English-language requirement.

FM6.3.1 Reviews and Analyses of the High-Level Requirements

d. Verifiability of HLR: The ability to express the high-level requirements for the system in CTL is a sufficient demonstration of verifiability in this example.

e. Conformance to standards: Requirements that do not conform to the standard for CTL syntax will be identified and rejected by the analysis tools. This feature of the tool would need to be qualified. Alternatively, conformance to CTL syntax can be easily checked by a manual review.

FM6.3.2 Reviews and Analyses of the Low-Level Requirements

a. Compliance with HLR: Analysis by model checking demonstrated that low-level requirements (the system model) complied with high-level requirements. This feature of the model checking tool would need to be qualified.

6 Conclusion

Adoption of model-based development methods is facilitating the use of model checking for verification of software in commercial aircraft. Model checking can provide increased assurance of correctness, reduced development costs, and (in the near future) satisfaction of certification objectives. Further research is needed to expand the range of models where model checking can be effectively applied. New analysis methods are needed to handle larger data types, floating point numbers, and non-linear functions.

Acknowledgments

The model checking and translation work described in this paper was accomplished with Steven Miller and Lucas Wagner from the Rockwell Collins ATC Automated Analysis group, and Michael Whalen and Mats Heimdahl from the University of Minnesota. The work was supported in part by the NASA Langley Re-search Center under contract NCC-01001 of the Aviation Safety Program (AvSP) and by the Air Force Research Lab under contract FA8650-05-C-3564 of the CerTA FCS program. The Formal Methods Technology Supplement for DO-178C has been developed by the Formal Methods Subgroup (SG6) of RTCA committee SC-205. Thanks to all.

References

1. Miller, S., Whalen, M., Cofer, D.: Software Model Checking Takes Off. Communications of the ACM, 53(2):58-64 (2010)
2. Halbwachs, N., Caspi, P., Raymond, P., Pilaud, D.: The synchronous dataflow programming language LUSTRE. Proceedings of the IEEE, pp. 1305-1320 (1991).
3. Whalen, M., Cofer, D., Miller, S., Krogh, B., Storm, W.: Integration of Formal Analysis into a Model-Based Software Development Process. In: Leue, S., Merino, P. (eds.) FMICS 2007. LNCS vol. 4916. Springer, Heidelberg (2008)
4. Miller, S., Anderson, E., Wagner, L., Whalen, M., Heimdahl, M.: Formal Verification of Flight Critical Software. In: AIAA Guidance, Navigation and Control Conference and Exhibit, San Francisco (2005)
5. DO-178B/ED-12B: Software Considerations in Airborne Systems and Equipment Certification. RTCA/EUROCAE (1992)
6. Whalen, M., Innis, J., Miller, S., Wagner, L.: ADGS-2100 Adaptive Display & Guidance System Window Manager Analysis. NASA Contractor Report CR-2006-213952 (2006)
7. Huth, M., Ryan, M.: Logic in Computer Science: Modelling and Reasoning about Systems. Cambridge University Press, 2nd ed. (2004)
8. McMillan, K. L.: Symbolic Model Checking. Kluwer Academic Publishers (1993)
9. Clarke, E. M., Biere, A., Raimi, R., Zhu, Y.: Bounded model checking using satisfiability solving. Formal Methods in System Design, 19(1):7-34 (2001)