Design of a Certifiably Dependable Next-Generation Air Transportation System

Stephen A. Jacklin, Michelle M. Eshow, Michael R. Lowry, Willem Visser,
Ewen Denny, and Johann Schumann
NASA Ames Research Center

NASA is participating in a multi-agency effort to define and develop the Next-Generation Air Transportation System (NGATS) for the US. In order to provide needed gains in throughput without compromising safety, the NGATS will be more autonomous than anything implemented before in the national airspace system. Such a system will of course be software-intensive and safety critical.

Although the authority responsible for certifying US flight control software is the Federal Aviation Administration (FAA), the US air traffic management (ATM) software to date has not been certified by the FAA or another authority in the legal sense of the word. Nevertheless, the NGATS software must be created in such a way that it is certifiably dependable because it is safety-critical software. A likely requirement is that future ATM software be developed according to a detailed and well-documented software development process similar to the process the FAA has endorsed for airborne flight software using the well-known RTCA DO-178B standard, *Software Considerations in Airborne Systems and Equipment Certification*. The fundamental tenets of DO-178B are that aircraft software programs meet their intended function, do not negatively impact other systems or functions on the aircraft, and are safe for operation.

The main problem, however, is that the implementation of RTCA DO-178B, or another similar standard, will be exceedingly difficult because of the complexity of ATM systems. ATM software is complex due to many factors, including the distributed nature of the decision making, the variety and conflicting interests of users, the large number of states and constraints necessary to represent the entire system, the variability of the environment both geographically and temporally, the non-determinism of many system inputs such as weather, emergencies, and security events, and the number of different hardware/software systems and sub-systems that must interact. Moreover, ATM planning and trajectory-based automation software is exceedingly complex in itself in that is has a large number of interfaces among various program elements and with other computers. The creation of certifiably dependable software therefore requires a certain degree of automation in both the software design process, as well as in software verification and validation methods.

One increasingly popular method of automation is the automatic synthesis of code using autocoders. An autocoder works much like a compiler for a high-level programming language, except that the autocoder takes as input high-level specifications in the form of graphical diagrams, equations, or properties and generates source code as output. It has been said that auto-coding offers a number of advantages over conventional programming. First, it is a time saver. It is not only faster to go directly from equations or a block diagram to code the first time, but a great time saver when code must be modified. Second, by taking the human out of the coding process, less coding variability is introduced. This potentially serves to reduce risk. However, a criticism of some autocoders is that they obscure traceability and create unreadable code. Therefore, auto-coding cannot be used for safety-critical systems unless the verification and validation issues are addressed at the same time. Under the funding from NASA's NGATS projects, NASA Ames Research Center is investigating a number of means to offer advanced code generation and V&V capability.

AutoFilter is a tool being developed at NASA Ames to automatically generate certifiable code from high-level declarative requirement specifications. The AutoFilter tool not only generates code automatically from high level specifications, but also generates various human-readable documents containing both design and safety related information required by certification standards such as DO-178B. Program synthesis is accomplished through repeated application of schemas, or parameterized code fragment templates and a set of constraints formalizing the template's applicability to a given task. Schemas represent the different types of learning algorithms. AutoFilter applies rules of the logic backwards and computes, statement by statement, logical formulae or *safety obligations* which are then processed further by an automatic theorem prover. To perform this step automatically, however, auxiliary annotations are required throughout the code. AutoFilter thus simultaneously synthesizes the code and all required annotations. The annotations thereby allow automatic verification and produces machine-readable certificates showing that the generated code

does not violate the required safety properties. Within the next year, it is intended to show application of this method to develop part of Ames' TSAFE (Tactical Separation-Assured Flight Environment) code, which is a set of algorithms designed to act as a stand-alone (back-up) program to monitor high-level automation functions and to provide very near-term collision avoidance instructions to controllers and aircraft, as distinguished from tactical or strategic advisories. Automatic generation of parts of the TSAFE code from high-level specifications will provide an independent assessment of the manually implemented code, thereby verifying the correctness of the TSAFE algorithms or finding sources of possible error. The capability to provide certifiably correct software may then be expanded to other ATM applications.

Perhaps the most demanding requirement on creating certifiably reliable software is the need to provide total code testing coverage. For example, the DO-178B standard requires MC/DC (Modified Condition/Decision Coverage) testing. Coverage concerns the percentage to which 1) every decision in a program has been executed at least once, 2) every decision in the computer program has been taken with all possible outcomes at least once, 3) every condition in a decision in the program has taken all possible outcomes at least once, 4) every condition in a decision has been shown to independently affect that decision's outcome, and 5) all entry and exit points of the program have been tested. The problem for ATM software, however, is that the myriad of decision gates makes testing of every possible path through the software humanly impossible. However, the automated verification technique known as model checking has been designed specifically for this purpose.

Over the last decade, the formal method of model checking has become an important tool for the verification of finite state automata. Once a finite state model of the program has been developed using a logical abstraction method, the model checker can test all possible execution paths of the program and report any execution that leads to a violation of user-defined properties, including safety properties. Ames Research Center intends to initially apply the JavaPathfinder model checker to the Java part of the Profile Selector - Enroute (PFSE) code for route planning, conflict prediction, trial planning, and automated resolution of conflicts. PFSE is written in C++ and Java, and has two levels of multi-threading, one in the C++ management of all the functions, and another in the Java code, which generates the conflict resolution advisories. This study will not only examine the applicability of model checking, but will also investigate what design for verification strategies might be added to improve the verification and validation process.

One of these strategies will be the application of compositional verification methods to make model checking more effective for large systems across different programming languages and multiple computer interfaces. Although model checking can exhaustively test all paths through the software, it can do so only until the system resources (e.g., computer memory, user patience) are exhausted as well. In the case of ATM software, the number of combinatorial cases to be explored is too great. Compositional verification allows large, complex programs to be automatically decomposed into smaller program elements that can be model-checked individually. This method thereby allows model checking to be applied to very large systems, potentially providing full coverage of exceedingly complex ATM software. It also provides a thorough examination of all interconnections between program and subprogram elements, whether as part of the same computer, or on multiple computers. Compositional verification is facilitated through the use of the Unified Modeling Language (UML) to represent the overall system. UML consists of more than a dozen diagrammatic languages as well as text-based languages for the analysis and modeling of object-oriented software. A UML model and its corresponding Java or C++ code provide identical information about the component or system, and nearly all UML languages of interest have code generators. The artifacts produced during compositional verification may be used throughout the lifecycle: for example, they enable the detection and correction of system level properties while checking components of the system in isolation during unit testing. This approach may also be used to reduce the effort of integration testing since it can automatically infer that specific system level test cases are redundant if some unit level tests have been completed.

Whether or not the code NASA develops during its NGATS R&D will be used operationally, the validation of the automation algorithms and concepts, and a road map to certifying their implementation in an operational system, are necessary if the concepts are ever to see operational use.