V&V of a Spacecraft's Autonomous Planner through Extended Automation

Martin S. Feather
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109, USA
+1 818 354 1194
Martin.S.Feather@Jpl.Nasa.Gov

Ben Smith
Jet Propulsion Laboratory,
California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109, USA
+1 818 353 5371
Ben.D.Smith@Jpl.Nasa.Gov

SYNOPSIS:

We have introduced and used significant automation during the verification and validation (V&V) of a spacecraft's autonomous planner. This abstract describes the problem we faced, the solution we employed, and the applicability of our approach in a general V&V setting.

PROBLEM:

Cost, performance and functionality concerns are driving a trend towards use of self-sufficient autonomous systems in place of human-controlled mechanisms. Our focus has been the verification and validation (V&V) of a spacecraft's autonomous planner. This planner generates the sequences of high-level commands that control the spacecraft. The planner is part of a self-sufficient autonomous system that will operate a spacecraft over an extended period, without human intervention or oversight. Hence, V&V of the planner is crucial.

The planner can exhibit a much wider range of behaviors that the command sequence mechanisms of more traditional spacecraft designs. Furthermore, it must respond correctly to a wide range of circumstances. Together, these raise some new challenges for V&V.

As for any complex piece of software, a major focus of V&V revolves around thorough testing. The new V&V challenges manifest themselves during testing as the following combination of characteristics:

- The planner's output (plans) are detailed and voluminous, ranging from 1,000 to 5,000 lines long.
- Each plan must satisfy all of the flight rules that characterize correct operation of the spacecraft. There are over 200 such flight rules.
- The information pertinent to deciding whether or not a plan passes a flight rule is dispersed throughout the plan.
- The thorough testing of the planner yields thousands of such plans, spanning the wide range of circumstances in which the planner is expected to operate.

As a consequence, manual inspection of more than a small fragment of plans generated in the course of testing is impractical.

SOLUTION:

Our approach has been to automate the checking of plans. The automated system checks each plan for adherence to all of the flight rules input to the planner. This verifies that the planner is not generating hazardous command sequences. The automated system also performs some validation checks. These arise from a gap between the "natural" form of a flight rule, and the way in which it must be re-encoded so as to be expressed to the planner. The automated system checks a direct encoding of the "natural" statement of the flight rule, thus helping validate that the planner and its inputs are accomplishing the desired

behavior.

We use a database as the underlying reasoning engine of our system to automatically check plans. To perform a series of checks of a plan, we load the plan as data into the database, having previously created a database schema for the kinds of information held in plans. We express the flight rules as database queries. The database query evaluator is used to automatically evaluate those queries against the data. Query results are organized into those that correspond to passing a test, which we report as confirmations, and those that correspond to failing a test, which we report as anomalies.

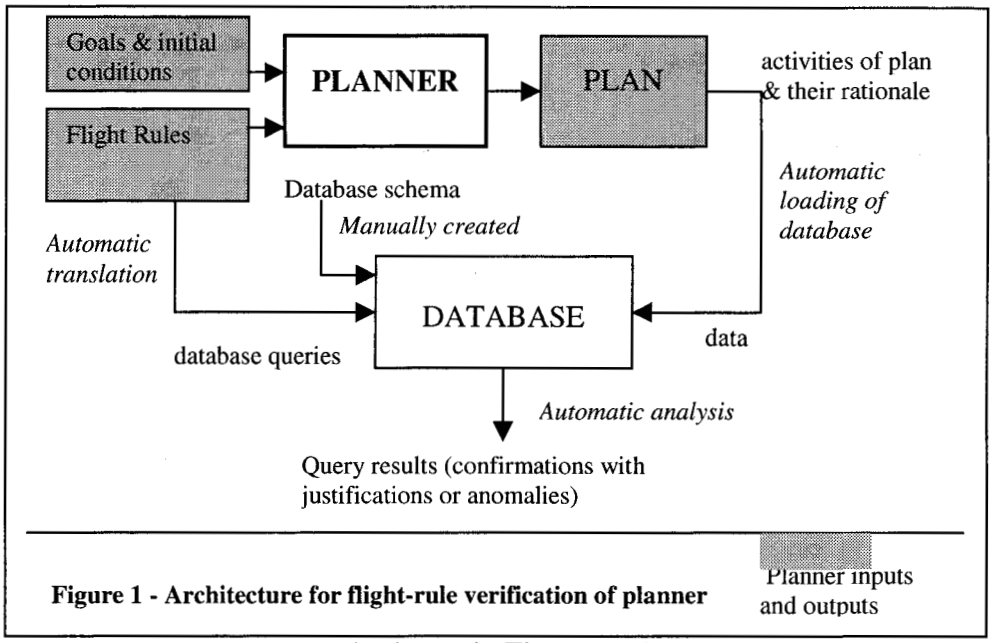

Figure 1 - Architecture for flight-rule verification of planner

The net result is that we can quickly and thoroughly check each plan. The automated checking code takes less than five minutes (on a Sun ULTRA Sparc) to perform each of several hundred checks of a large (5,000 line) plan and generate a report of the results. Plan generation is a search-intensive activity, and a planner is a complex piece of software precisely because of the need to perform this search in an effective and efficient manner. Conversely, once a plan has been generated, checking properties of that plan is relatively straightforward.

Because the flight rules themselves are numerous and detailed, and evolve over the course of software development, we have taken the automation one step further. We generate the verification part of the plan-checking code from the flight rules themselves, in the same form in which they are input to the planner. Using this capability, we are able to *automatically* regenerate the flight-rule checking code, whenever the set of flight rules input to the planner evolves. The architecture of this system is shown in Figure 1, above.

APPLICABILITY:

Our approach has been developed for, and applied to, V&V of a spacecraft's autonomous planner. However, we believe the approach has much wider applicability than this one project. The characteristics that identify when this approach is worthwhile and viable are as follows:

Worthwhile: The development of automated test checking code, rather than relying upon manually conducted checks, is warranted when:

- There are voluminous amounts of data to check, either because each test run yields lots of data, or there are numerous test runs, or both.
- The checking of a test run is complex, either because there are many checks to perform, or the checks themselves are hard to perform, or both.

These conditions render manual checking unsatisfactory.

Viable: The style of automated checking we developed requires the following conditions to hold:

- The data to check is self-contained. That is,

there is no need for human interaction to determine whether or not a check has been met. (In our planner task, each plan is a self-contained object from which it can be determined whether or not each flight rule holds.)

- The data to check is in a machine-manipulable form. That is, it is feasible to develop automated checking that will work directly off the form of data available, without human intervention. (In our planner task, plans have exactly this characteristic, since they are intended for consumption by the spacecraft's automatic executive.)
- Checking is easier than generation. That is, the code to check that a test run satisfies the desired conditions is simpler than the code that generates that test data.

  This has two positive consequences:

  1. The development of the automated test checking code will be a much lesser effort than the development of the system under test.
  2. The test checking code will run faster than the system under test (meaning it can easily keep up with the test data generated, and provide quick feedback to the test personnel).

  Both of these consequences were exhibited in our effort. The development of the planner took years, while the development of the plan checker months. For plans in the range of 1,000 to 5,000 lines long, the planner takes 3 to 10 minutes to yield the plan, while the plan checking code takes 30 seconds to 4 minutes to perform its checks of a plan.

Our automatic generation of flight-rule checking code reflects the same characteristics of an activity that is worthwhile and viable to automate:

- we have hundreds of flight rules to check
- individual rules can be quite complex
- the set of rules evolves over time
- flight rules are expressed in a machine-manipulable format (constraints input to the planner)
- the language of those rules (planner constraint language) is carefully proscribed so as to render plan generation feasible; the expression of those rules as checks can employ an extensible, general purpose language.

In our system, generation of the flight-rule checking code takes under 10 minutes and is completely automatic.

FURTHER OBSERVATIONS

Our problem and solution exhibit two further characteristics of general importance.

The value of redundancy and rationale: Each plan generated by the spacecraft's planner contains both a sequence of activities, and justifications for those activities. These justifications related each activity to the flight rules that were taken into account in planning that activity. Viewed solely as a command sequence, the presence of these justifications in the plan is redundant. However, these justifications serve two very useful roles for V&V purposes:

- they provide rationale for why the planner arrived at a plan. This rationale can be checked to ensure that the planner is not only arriving at the "right" solution (namely, a plan that adheres to all the flight rules), but is doing so for the "right" reasons. This gives the test team confidence to extrapolate the correct operation of the planner to a wide range of circumstances.
- they provide redundancy that contributes to our confidence in the checking code itself. Our test checking code independently performs the following three kinds of checks:

  1. that the activities of the plan adhere to all the flight rules,
  2. that there is a justification recorded with

each activity in the plan for every flight rule that the checker finds is applicable to that activity, and

3. that every justification recorded in the plan can be traced back to a flight rule.

This makes it unlikely that the checking code has a "blind spot" that happens to overlook a fault in a plan.

The automated test checking code we automatically generate from planner flight rules checks this rationale.

Opportunities for validation: Verification was the original focus of our plan checker generation effort. By thorough checking of the planner's outputs (plans) against the flight rules given as input to the planner, we gained confidence that the internal operation of planner was correct. However, the effort also yielded significant opportunities for validation.

Validation opportunities arose from a gap between the most "natural" statement of a flight rule, and the form in which it must be re-encoded so as to be expressed to the planner. The planner constraint language is carefully proscribed so as to render plan generation feasible. On occasion, a flight rule cannot be expressed directly in this limited language. Instead, it must be (manually) subdivided into several separate rules that in conjunction will achieve the requisite condition, and that individually can be expressed in the constraint language. Our language for expressing checks is more general purpose than the planner constraint language. This means that it is possible to (manually) encode an automatic check corresponding directly to the original flight rule. By following this process, we are able to validate that the planner, and the encodings of flight rules given to it, do in fact achieve the original intent.

Note that there is a manual step to this validation - we must manually encode the original flight rules (expressed in natural language) as checking code. The checking code then runs automatically. However this manual step can take advantage of the framework established by the verification architecture and code.

In more general terms, we see that verification can be extended into the realm of validation when the verification language is more general than the language of the system being verified

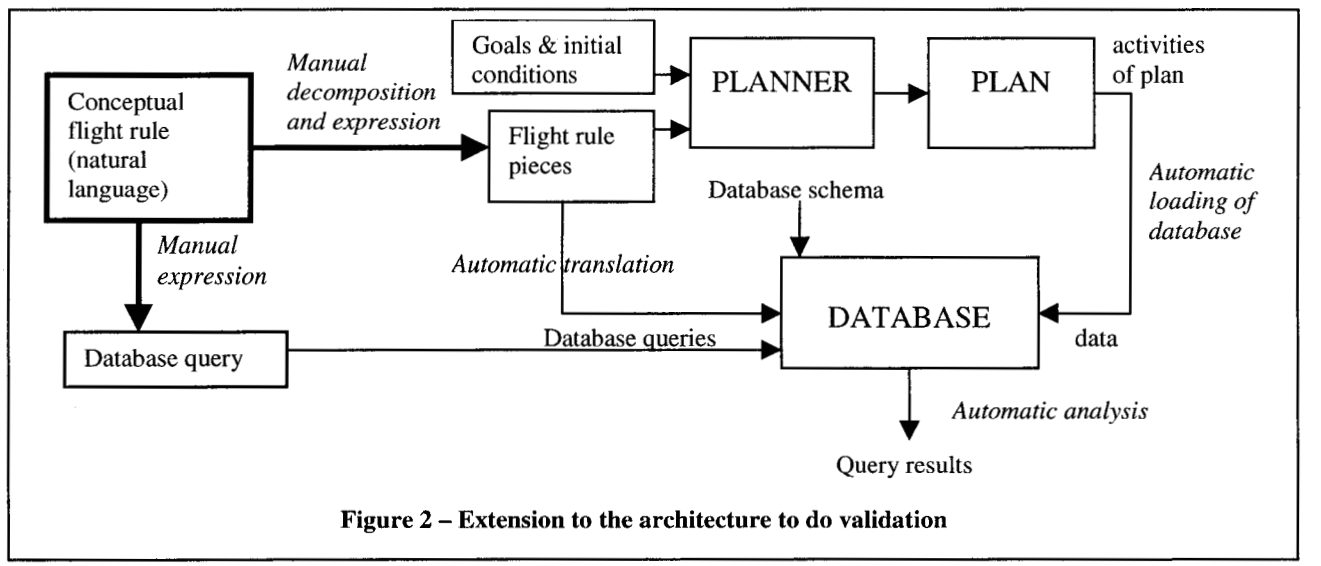

Figure 2 – Extension to the architecture to do validation