Validation of Reasoning and Decision-Making of a Satellite Autonomous On-board Software

Fabrício de Novaes Kucinskis
INPE - Instituto Nacional de Pesquisas Espaciais
Avenida dos Astronautas, 1758
São José dos Campos / SP – 12227-010 – Brasil
fabricio@dea.inpe.br

Abstract - **We have been working at INPE on the concept of goal-based operations. This means commanding our satellites through goals (i.e. "take pictures of 'x' place, store them and send me later"), instead of sequences of commands. When receiving a goal, the satellite reasons to convert it to the appropriate commands.**

Goal-based operations involve the increase of the satellite's autonomy. To accomplish this, we are developing an on-board service that manages a knowledge base, and a replanning application that reasons over this knowledge to decide the best way to achieve a goal. However, autonomous applications present a challenge for validation: how can one validate a system that is made to take actions that would be expected from a human operator?

This paper describes a work in progress on the increase of the operational autonomy of INPE's satellites, and presents our concerns and approach regarding the validation of the autonomous on-board software.

1. INTRODUCTION

The number of projects for autonomous software on-board spacecrafts has been increasing in recent years, showing a clear trend for the adoption of goal-based and autonomous operations in space missions in a near future.

Following this trend, we at the Brazilian National Institute for Space Research (INPE, in the Portuguese acronym) have developed a prototype for a model-based, on-board autonomous replanning software [1].

This first experience gave birth to a wider approach, an 'autonomy kernel' in the form of an on-board service. This service, called the Internal State Inference Service (ISIS), manages a knowledge base and provides states inference, resources profiling, constraint propagation and simple temporal networks features for on-board autonomous applications that perform tasks such as prognosis and goal-based mission replanning.

One of the aspects of this work that we are concerned with is the verification and validation (V&V). Due to the capability of autonomous software to reason and make decisions based on the knowledge base, without the interference of human operators, the need to verify and validate them is even greater than for non-autonomous software. However, little is found in the literature regarding V&V for autonomous software.

This paper presents our concerns and approach regarding V&V for ISIS and its first consumer, an on-board replanning application fondly named LetMeDo.

2. THE INTERNAL STATE INFERENCE SERVICE AND THE ON-BOARD REPLANNING ARCHITECTURE

ISIS [2] is an on-board service that, roughly speaking, provides knowledge about the domain for on-board applications. Such applications can consume the services to reason over this knowledge and make decisions, increasing the autonomy of future INPE's satellites. Examples of consumers are prognosis[1] applications and on-board replanners.

ISIS comprises a structural and behavioral model of a given domain/discipline in the space segment, such as payload operation or attitude control. The domain depends on the purpose of the applications that will consume the service.

The service gives access to the model both from ground as well as from another on-board application. Through ISIS it is possible to start an inference session, to query future satellite states, to submit changes on the actions that will take place (to perform 'what-if' scenario analysis), and more. It is also possible to update the model, in order to correct it or reflect a new behavior of the satellite (faulty equipment, for example).

The first consumer we are developing for the service is a general-purpose on-board replanner. This replanner, called

[1] A prognosis on-board application is one that tries to forecast future error conditions, by inferring the future states from the current state and the predicted/scheduled actions. Once a possible future error is predicted, the application can warn the authority (the operations personnel or a reconfiguration mechanism) in order to perform the preventive actions.

LetMeDo, applies Constraint Satisfaction Problems (CSP) techniques over the model to achieve a set of given goals. The goals are defined by a 'Problem Composer' as violated constraints to be solved.

After receiving the problem to solve, LetMeDo starts an ISIS inference session and performs queries and scenario analysis over the model. If a solution is found, the replanning is finished with success. The resulting software architecture, as well as the data flow between the components, are shown in Figure 1.

It is important to notice that both ISIS and LetMeDo are generic components that contain domain-specific elements – the on-board model and heuristics for the class of problem to solve. The Problem Composer is also domain-specific.

By its nature, a model-based system is meant to deal with complex problems, difficult to predict in detail or represent with simple sets of rules. Ultimately, a good model-based software should be able to solve problems that aren't even expected, nor completely known in advance. How to verify and validate software with such characteristics?

3. The Lack of Adequate V&V Techniques for Autonomous Space Systems

Since the beginning of ISIS design, we have been concerned with the V&V process for autonomous software. So, we made a survey on V&V techniques for autonomous space systems.

Little information was found, mostly pointing that the current V&V techniques aren't enough for the given problem. As Brat and Jónsson [3] stated, "our current validation techniques struggle with existing mission systems and now we are faced with validating autonomous systems that can exhibit a much larger set of behaviors".

In 2001, the Research Institute for Advanced Computer Science (RIACS) and the Carnegie Mellon University organized a workshop to discuss and identify the main challenges of V&V for autonomous systems in future NASA missions [4]. Participants from the V&V and autonomous and adaptive systems communities were invited, as well as NASA engineers.

The attendees pointed the limitations of V&V techniques and ranked a set of good practices and promising techniques, but made clear that the possible solutions were still far from be able to deal with autonomous systems.

One of the topics over which the workshop attendees identified more work to be done was the V&V for model-based systems (do not confuse it with 'model-based V&V'). According to them, there is the need to define and gain experience in the Software Enginnering process for model-based systems, which represent a significant part of the autonomous software being developed in the space field.

They asked what kind of requirements would a customer

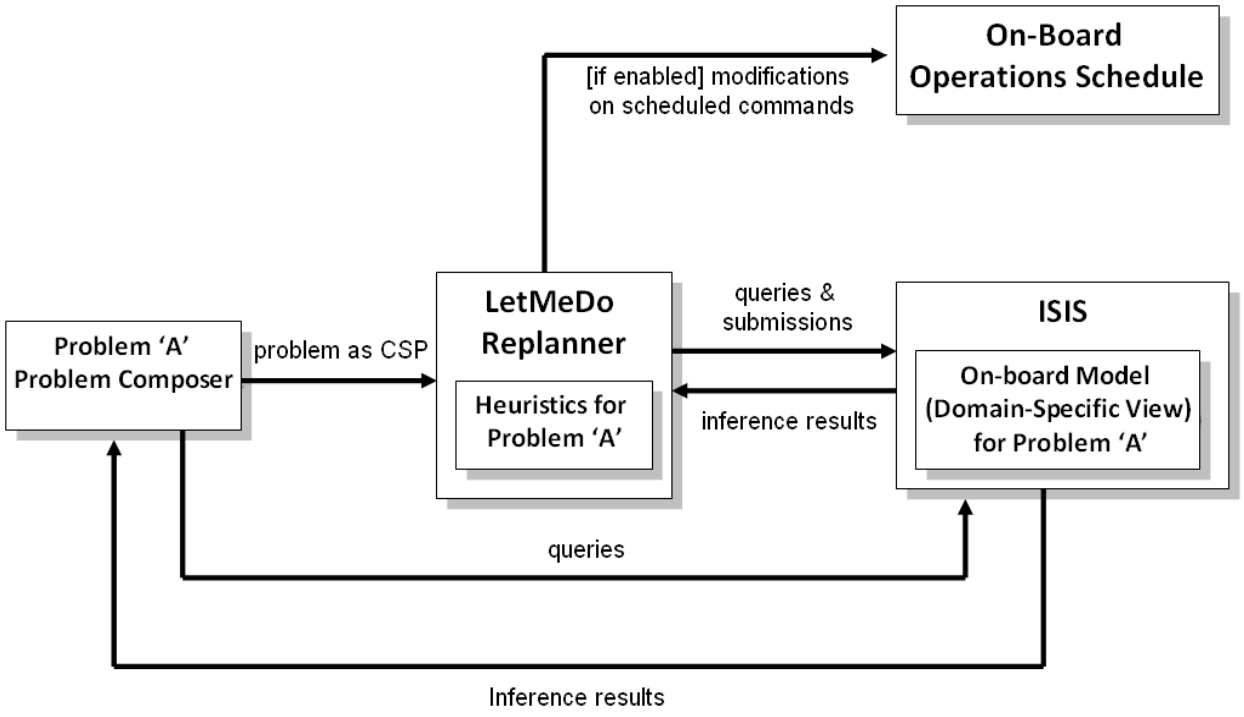

Figure 1 - The replanning process and on-board components

expect? How could these requirements be expressed and verified conveniently? Is it possible to develop or specialize a theory and practice for this kind of system?

According to the workshop conclusions, a natural approach would be to decompose the V&V problem across the three core components of a typical model-based system for spacecrafts: 1) the 'plant' (spacecraft or flight software), 2) the engine and 3) the model. This approach seemed interesting for our work.

Furthermore, two of the raised questions are in line with our concerns: how does one verify/validate a model? And, given a 'valid' model, how to verify/validate the decisions of the autonomous system? The attendees left those questions open.

4. Our Approach for the Validation of the Autonomous Behavior

Based on the RIACS 'divide-and-conquer' approach for autonomous systems, we've decided to split our V&V problem into smaller parts. The first thing to separate was the 'V' from the 'V'.

Verification

There are many concepts for verification, but all of them agree on one point: it is always performed against the products of the software development process, mainly against the requirements. However, this is not that simple for autonomous systems: Pecheur et al. [4] reported that "stating formal requirements for autonomous and adaptive systems is hard and, as such, not something often done during system development at NASA".

So, we started an effort to improve the quality and level of detail of our requirements, making them as formal as possible. This culminated into dozens of definitions and almost two hundred technical requirements. An analysis of these requirements has shown that the current Software Engineering techniques are adequate to verify them.

It seems that the fact that an on-board software will show autonomous behaviour does not impact its verification. Validation, however, is a very different problem.

Validation

To validate is to determine if a product (in our case, an autonomous software) fulfills the customers' expectations. Generally speaking,, the customer expects that an autonomous software makes the right decisions, the ones that he would make in its place. It's clear that 'the right decision' is an abstract concept, difficult to validate, especially when this decision can happen under unpredicted situations.

Returning to the 'divide-and-conquer' idea, we've divided our software architecture in components to deal with separately, in terms of validation: 1) the service, 2) the model and 3) the replanner and Problem Composer.

We noticed that there is nothing special, in terms of autonomy, related to the service. It's an ordinary monitoring and control software that can be validated through the current Software Engineering techniques.

The other components of our architecture are responsible for the autonomous behavior and, as such, we didn't know in advance how to perform their validation. How can one validate software that is made to take actions that would be expected from a human operator?

Our tentative answer was to submit the on-board knowledge, reasoning and decisions to the operators in a gradual process: we first validate the knowledge before allowing it to be used, and them we start validating the reasoning – that will not perform any autonomous action until it is enabled to do so.

Model Validation

We validate the model by creating a Model Validator application. Every time a new satellite's commands schedule (the operations plan) is received from ground or changed by the on-board replanner (after it is allowed to run), the Model Validator runs inference sessions based on the on-board model.

The results of the inference sessions are a set of timelines with the predicted states of the satellite, from 'now', to the last scheduled command. Then, the Model Validator will start acquiring, at regular intervals, the observed states to compare with the predicted ones.

If deviations between the expected (modeled) and the observed behaviors are detected, the model is flagged as 'invalid', and its use in the replanning process will not be allowed anymore. The results of this comparison are also sent to ground as reports to the operations personnel, which will analyze them and determine which corrections shall be made on the model. This will be repeated until no more relevant deviations are detected.

If there is no deviation between the predicted and the observed behaviors, the model can be considered valid – but not forever. The spacecraft ages, and hence the model has to reflect this. If on-board hardware fails, for example, the model will not be able to predict this new behavior. So, the Model Validator runs continuously, sending alerts to ground when it detects behavioral deviations that didn't exist before. This triggers a new validation process, to detect what has changed in the satellite and let the ground personnel to update the model.

Replanner Decisions Validation

After the validation of the knowledge (model) is performed, it is possible to validate the reasoning (replanner).

As we can't risk to put the mission in jeopardy by an incorrect autonomous decision made by the replanner, its validation (and, together with it, the Problem Composer) is performed 'off-line'. The validation is based on the gain of confidence, by the operations personnel, on the reasoning performed on-board. The main idea is to allow the replanner to decide what actions to take, but not – at first – to execute those actions.

With the model validated, the Problem Composer can define the goals to achieve and call the replanner. This will start the replanning process, querying the model (through ISIS) to perform 'what-if' scenario analysis, in order to solve the received problems – or else, to achieve the goals.

After the replanning is finished, being it successful or not, the replanner will not execute the resulting plan. Instead, it will send reports to the ground about the goals received, the resulting modifications on the plan to achieve them, and some key parameters that had driven the reasoning process.

The operations personnel will them determine the quality of the results the autonomous software provided. If they are not good enough, adjustments on the replanner, or even on the model, will be considered.

We'll keep a log of the operator's impressions and, when they consider that the replanner's responses are consistently adequate, the execution of the replanners actions can be enabled. Our expectation is that this process will show to the operations personnel what ISIS and the on-board replanner could do, if they let them to. That's why we called it LetMeDo.

It's still not clear to us yet if a change on the model should imply on a new validation process for the replanner. At first our answer would be 'no', as the search algorithms and heuristics will be the same. But a new model leads to new paths taken on the search process, which could be not taken before. So it is recommended to re-validate the replanner after any change on the model.

6. FINAL REMARKS

In this paper, we presented our concerns with the verification and validation of the on-board autonomous software that are being developed for future INPE's satellites.

In our project, we managed to narrow the V&V problem for autonomous software to the validation of the components that are responsible for the autonomous behavior: the on-board model and the replanner.

We came to the conclusion that the current Software Engineering techniques are adequate to the verification of such system.

For the validation of the autonomous behavior, however, we couldn't find any appropriate method. So, we decided to apply different approaches for each component of the autonomous architecture.

For the validation of the on-board knowledge, we compare the predicted behavior with what is observed. For the reasoning, we'll submit the decisions to the operations team. The software will be allowed to execute autonomously only after it gains enough confidence to do so.

The validation of autonomous systems is still an open field to explore, and we hope that our solutions could help on the maturing of concepts in direction of a more formal approach.

ACKNOWLEDGEMENTS

The author thanks the Inertial Systems for Space Applications (SIA) project funded by the Research and Projects Financing (FINEP) for the financial support.

REFERENCES

[1] F. N. Kucinskis and M. G. V. Ferreira, "Dynamic Allocation of Resources to Improve Scientific Return with Onboard Automated Replanning", in Space Operations: Mission Management, Technologies, and Current Applications, Progress in Astronautics and Aeronautics Series, v. 220, Chapter 20, pp. 345-359, AIAA, September 2007.

[2] F. N. Kucinskis and M. G. V. Ferreira, "Taking the ECSS Autonomy Concepts One Step Further, in Space Operations: Exploration", in Proceedings of the 11th International Conference on Space Operations, Hunstville, April 2010.

[3] G. Brat and A. Jónsson, "Challenges in Verification and Validation of Autonomous Systems for Space Exploration", in Proceedings of IJCNN'05: Performance of NeuroAdaptive and Learning Systems: Assessment, Monitoring, and Validation, 2005.

[4] C. Pecheur, W. Visser and R. Simmons, "RIACS Workshop on the Verification and Validation of Autonomous and Adaptive Systems", in AI Magazine, Volume 22, Number 3. AAAI, 2001.