# [illegible]

**Steve Easterbrook**
NASA IV&V Facility,
100 University Drive, Fairmont, West Virginia 26505,
steve@atlantis.ivv.nasa.gov

**Robyn Lutz, Rick Covington, John Kelly**
NASA Jet Propulsion Lab, Pasadena, California

**Yoko Ampo**
NEC Corp, Tokyo, Japan

and

**David Hamilton**
Hewlett Packard Corp, San Diego, California

## *Abstract*

This paper describes three cases studies in the lightweight application of formal methods to requirements modelling for spacecraft fault protection systems. The case studies differ from previously reported applications of formal methods in that formal methods were applied very early in the requirements engineering process, to validate the evolving requirements. The results were fed back into the projects, to improve the informal specifications. For each case study, we describe what methods were applied, how they were applied, how much effort was involved, and what the findings were. In all three cases, the formal modeling provided a cost effective enhancement of the existing verification and validation processes. We conclude that the benefits gained from early modeling of unstable requirements more than outweigh the effort needed to maintain multiple representations.

**Steve Easterbrook**

# , B o n

## *1 Fault Protection*

For NASA spacecraft, the term fault protection is used to describe system elements that avoid, detect and respond to perceived spacecraft faults. There are generally two over-riding requirements when a fault occurs: the system needs to

Requirements engineering processes within NASA appear to have reached a “quality ceiling” in which the currently employed development and assurance techniques have been optimized so much that no further improvements can be expected. This effect is shown in the data from formal inspections, in which the number of defects found in the requirements phase is seven times higher than in the code phase [10]. There is a significant lack of effective methods and tool support for the requirements phase in comparison to those available for detailed design and coding.

The lack of rigorous requirements engineering techniques is well illustrated in the fault protection area. Fault Protection requirements are derived from failure models of the target system, along with various safety analyses. From these sources, individual requirements are written to identify different fault conditions, initiate the appropriate response, and monitor the outcome. The results are expressed in a combination of tables, diagrams and prose, with an emphasis on prose for baseline requirements. The result is a large complex set of requirements documents, in which interactions between requirements can be hard to identify, let alone validate. Further problems arise from the fact that fault protection requirements are more volatile that most other requirements, as they are sensitive to any change during the development of the target system.

The complexity of fault protection means that it is hard to demonstrate that the system and software requirements for fault protection adequately describe everything that is needed to achieve the goal of providing robust spacecraft. Formal methods can help provide this validation in a number of ways. The process of formalising a specification provides a simple validation check, in that it forces a level of precision and explicitness far beyond that needed for informal representations. Once a formal specification is available, it can be formal challenged [3], by defining properties that should hold, and proving that they do indeed hold. Formal challenges may be achieved both through the use of mathematical proofs, and through state exploration or ‘model checking’.

Rushby [3] points out that there is considerable scope for selective application of formal methods. For example, formal methods can be applied just to selected components of a system, and can be used just to check selected properties of that system. Most importantly, a great deal of benefit can be derived from formal methods without committing a project to the use of formal notations for baseline specifications. In the studies described in this paper, we used formal modeling to find errors in critical parts of existing informal specifications, but did not replace the informal specifications with their formal counterparts. This approach is consistent with the advocacy of multiple representations as a way of overcoming analysis bias.

## *3 Formal Methods and NASA*

A multi-center team within NASA has been exploring the potential of formal methods [12, 13]. The team combines personnel with experience in formal methods, in the domains where formal methods are being applied, in software assurance and V&V, and in technology transfer. A series of studies by this team have explored formal methods on a number of NASA programs, including Space Shuttle [5], Space Station [14, 15], and Cassini [16]. Throughout these studies, the emphasis has been on pragmatic application of formal methods in areas where there appears to be the greatest need. Results of these studies are described in two NASA guidebooks [17, 18].

Although the three studies described here used different tools and notations, the basic approach is the same:

1) restate the requirements in a clear, precise and unambiguous format;
2) identify & correct internal inconsistencies
3) test the requirements by proving statements about expected behavior.
4) feed the results back to the requirements authors.

In two out of the three studies, step 1 involved an intermediate, informal notation, as a prelude to translating the requirements into the formal specification language. The intermediate notation helped to clarify ambiguities, and gain a better understanding of the structure of the requirements. This in turn helped to determine how the formal notation would be used.

The purpose of this study was to assist with the independent assessment of the fault detection, isolation and recovery (FDIR) requirements for the space station. Verification of the space station FDIR systems is particularly problematic, as FDIR functionality is distributed across many of the flight computers. The development and construction schedule for the space station does not permit full integration testing of the entire architecture prior to on-orbit assembly. Hence, FDIR functionality must be verified through a combination of inspection, testing and analysis.

Independent assessment is an oversight activity, covering all aspects of the system, including hardware, software and operational procedures. The aim is to assure an appropriate level of safety in the development of the space station. At the time of this study, the independent assessment panel was seeking some assurance that the high level FDIR concept was clearly defined and validated, before it flowed down to end item requirements. Subsequent changes to the FDIR concept would have significant impacts throughout the requirements and design of the entire system. For these reasons, the independent assessment panel commissioned a formal analysis of the high level FDIR function. The study was jointly funded by NASA headquarters, as part of the pilot program in formal methods.

The need that arose from the independent assessment dovetailed with the aims of the inter-center formal methods team. We had completed some preliminary studies of Space Shuttle re-engineering requirements [5], which had demonstrated the potential for formal methods as a requirements assurance technique. However, this work concentrated on analyzing change requests for an existing system. The space station work was a chance to get in early in the high level (system) requirements phase for an entirely new system. We needed to investigate whether there were any significantly different problems associated with applying formal methods to the early modeling activities in a requirements phase for a new system.

## *1 Approach*

Three views of the FDIR had been documented: the functional concept diagram (FCD) which is a flowchart like representation of the generic FDIR algorithm; baseline FDIR requirements; and capabilities, in which the requirements are grouped into related functional areas. This study concentrated on the first two of these views, developing a formal model of each, and testing traceability between them.

The four step approach described above was used. In this study, restating the FCD involved a process of abstracting out common features before it could be translated into PVS [19]. The baseline requirements were translated directly into PVS. PVS was chosen for this study, because it provided an automated theorem proving support, and because the specification language appeared to be readily understandable to engineers and programmers. Internal consistency of the models was tested using PVS typechecking, while the expected behavior was analyzed by defining theorems expressing required properties, and showing that they followed from the model using the PVS proof assistant.

The first step was to analyze the FCD. The original FCD contained 53 processing steps, making it rather complex. As a first step in the analysis, this diagram was partitioned, in order to create a more abstract view. For example, the first 12 steps involved checking parameters for out of tolerance conditions, the next 7 dealt with safing, the next 8 dealt with checking for functional failure, and so on. In addition, each step was labeled as one of three procedural categories: performing automated procedures, checking for anomalous conditions, and recording/reporting results. Finally, the conditions under which control is passed to higher level FDIR domains were identified. Six categories of condition under which this occurs were identified. The result of this initial analysis was a more structured (informal)

model of the FDIR processes. This model was informally checked for reasonableness, and for traceability to the original FCD. A number of anomalies were discovered at this stage, which were reported to the requirements authors.

The next step was to formalise the model in PVS. A consistent terminology was developed, and all objects and attributes referenced in the FCD were expressed in PVS. Figure 1 shows two fragments of PVS generated at this stage. The resulting definitions were typechecked using the PVS tool. Typechecking helped to eliminate several types of errors in the specification, including typos, syntax errors and type consistency errors.

```
message: type =
{  parameter_OK,
   parameter_verified,
   safing_not_allowed,
   safing_executed,
   ...
}

% parameter is ok when its tolerance
% check has just ran and the parameter
% is OK (i.e. within tolerance)
rr_parameter_ok: axiom
   forall (t: tolerance_check):
   (  on(just_ran(t, time) and
      OK?(t(time)))
   iff
      record_check(time)(parameter_OK, t)
   )
```

Figure 1 Sample PVS Definitions

Finally, the PVS specification was validated by using the PVS proof assistant to prove claims based on the specification. An example of such a claim is “at any domain level, if a failure occurs then it will always be recovered at some domain level”. Although this claim was not very profound, several missing assumptions were detected in the process of proving it. For example, several sequencing constraints needed to be defined explicitly, even though the FDIR documentation had stated that no such constraints should be inferred from the requirements. A total of 14 claims were defined and proved. Most of these were type correctness conditions (TCCs), which mainly serve to ensure internal consistency of the model.

The second part of the study was to analyze the baseline system requirements for FDIR. A distinction can be drawn between the primary space station system, and the FDIR system that monitors the primary system. The formal modeling concentrated only on the latter. The prose requirements were translated into PVS, using the definitions and types generated in the first part of the study. Translation of these requirements into PVS proved to be relatively straightforward. Figure 2 gives an example.

```
Requirement: automatic hazard and hazardous condition detection: ISSA
shall automatically detect any out-of-tolerance condition or functional
performance parameter that exhibits a time to catastrophic or critical
effect of less than 24 hours.

automatic_hazard_condition_detection: axiom
   forall (p:parameter)
      param_out_of_tol?(p) AND time_to_effect(p)<24 =>
      exists(d:fdir_domain): detection(p,d) = automatic
```

Figure 2 An Example of a Requirement Translated into PVS

The process of translating these requirements revealed a number of relatively minor ambiguities and incompletenesses. For example, the distinction between the primary system and the FDIR system was not clear in

the original requirements. Other ambiguities surrounded the use of terms such as “anomaly”, “out-of-tolerance” and “functional failure”.

Having modeled both the FDIR concept diagram and the baseline requirements, the plan was to explore traceability between the two. An initial analysis indicated that there was little traceability. The requirements authors confirmed that the two documents expressed different kinds of requirements. The FCD describes the processing that is performed within an FDIR domain, while the baseline requirements describe a higher level view of the kinds of FDIR that must be provided.

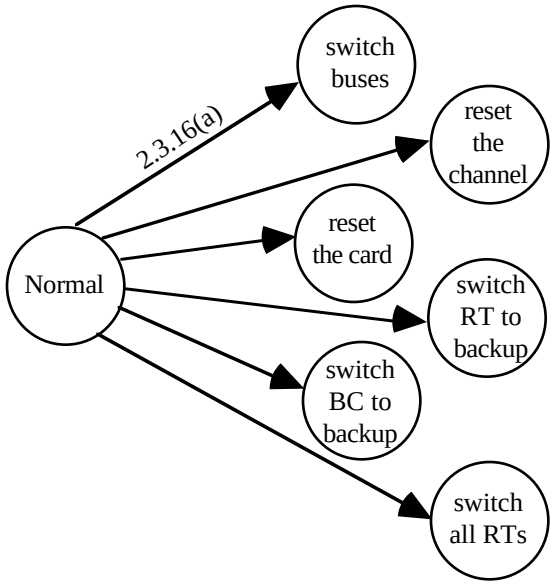
## 2 Findings

In general, the FDIR requirements were well thought out. However, although the FDIR requirements team understood them, there was some question over whether the documentation was sufficient so that system developers and other stakeholders would understand them. A total of fifteen issues were documented and discussed with the requirements authors. Most of these were minor ambiguities, inconsistent use of terms, and missing assumptions, discovered during the process of formalisation, which reduce the ability of developers to understand the requirements. Three of the issues were regarded as “high-major”:

1) There were inconsistencies in the FCD over reporting the status of safing, recovery and retry procedures. The intention was that the FDIR processes should report their status before, during and after execution of each procedure. However, some of the procedures were missing requirements for some of the reporting activities, so that most of them did not have requirements to report status at all three points. This was detected during the initial analysis of the FCD diagram.

2) The proper sequencing of FDIR processing is not clear from FCD. Although the FCD looks like a flowchart, the accompanying text makes it clear it should not be interpreted as a sequential process. However, some important

| | | | | | |
|---|---|---|---|---|---|
| | | | OR | | |
| C&C MDM acting as the bus controller | T | T | T | T | |




A on on o h on o C o h n o n on

## 2 Findings

In addition to a number of minor problems with inconsistent use of terminology, the following major problems were reported:

1) There were significant ambiguities in the prose requirements, as a result of the complex sentence structure. Some of these ambiguities could be resolved by studying the higher level FDIR requirements, and the specifications for the bus architecture. The ambiguities that arose from the combination of ‘ands’ and ‘ors’ in the same sentence could not be resolved in this way, and could lead to mistakes in the design. These ambiguities were detected in the initial reformulation of the requirements as truth tables.

2) There was one missing requirement to test the value of the Bus Switch Inhibit Flag before attempting to switch to the backup bus. This was detected during the test for disjointness in the SCR specification.

3) The prose requirements were missing a number of preconditions that enforce the ordering in which the inference rules should be applied. The accompanying flowchart for these requirements implied a sequence for these rules. An attempt had been made in the prose requirements to express this sequence as a set of preconditions for each rule, to ensure that all the earlier rules have been tested and have failed. The preconditions did not completely capture the precedences implied by flowchart. This corresponded with an informal observation made by the IV&V team that the ordering of the requirements should be made explicit. This problem was found during the test for disjointness in the SCR specification.

4) The timing constraints expressed in the requirements were incorrect. Several of the failure isolation tests referred to testing whether certain FDIR actions had already been tried “in the previous processing frame”. However, as each FDIR recovery action is followed by a time-out in order for the action to take effect, and as further FDIR intervention is only initiated on occurrence of errors in two consecutive processing frames, these tests can never be true. This was discovered during model checking of the PROMELA model.

## 3 Observations

The study analyzed 15 pages of level 3 requirements, and was conducted over a period of four months, by one person working part time. The total effort was approximately 1.5 person months. The main effort was in formalising the requirements. Translation from the SCR model to PROMELA was relatively straightforward, and took two days

problems were more serious. It is unlikely that they would have been discovered in this phase without the use of formal methods.

A major problem during this study was the volatility of the requirements. New drafts of the requirements document were being released approximately every two months. This meant that in at least one case (finding 3 above), the problem had already been fixed by the time it was discovered in this study. This issue had already been observed informally and reported by the IV&V team, and had been addressed by reducing the complexity of this section of the requirements. We mitigated the problem of fluctuating requirements by only doing the minimum amount of modeling necessary to test the properties that were of interest. For example, the SCR model is not a complete state model, as it models only a subset of the state transitions expressed in the requirements. The transitions for returning to the normal state have not been modeled. This partial model was sufficient to perform the coverage and disjointness analysis.

It should also be noted that in order to perform the analysis in this study, the SCR notation was slightly misused. The modes shown in figure 4 do not represent true modes in the SCR sense – a more correct representation would express these as output events from the FDIR system. However, defining them as modes permitted the use of coverage and disjointness tests on the transitions. This represents a pragmatic approach in which the formal method is applied in whatever way gives the most benefit, without necessarily following the original intent of the method.

## f on on C n

The third study concerns the system level fault protection software for Cassini. Cassini is a deep space probe, to be launched in 1997, which will explore Saturn and its moons. System reliability is a major concern for Cassini, due to the duration of the mission. Fault protection is a major factor in providing the required levels of reliability. Fault protection software is therefore mission-critical, in addition to being a complex embedded system. The study examined the requirements for two main components of the fault protection system: the software executive that manages fault protection, and requirements for putting the spacecraft into a safe state.

The aim of this study was to explore the effectiveness of formal methods in supplementing traditional engineering approaches to requirements analysis. The Cassini project was interested in the potential of formal methods to provide an assurance that the fault protection requirements were correct, while the formal methods team was interested in the opportunity to apply formal methods early in the requirements process, where early modeling of unstable requirements might pose a challenge.

### *1 Approach*

For this study, the initial step of re-stating the requirements included the use of OMT diagrams. These were then used to guide the development of a PVS model of the requirements. Once the PVS model was checked for internal consistency, a number of properties were defined, to check that the software would function correctly and be hazard free.

The first step was the production of OMT diagrams representing the documented requirements (see figure 5). The original requirements were expressed in natural language. The production of object diagrams, state diagrams and dataflow diagrams, according to the OMT method, helped to define the boundaries and interfaces of the fault protection requirements, and helped to crystallize some of the issues that arose in the initial close reading of the requirements. A number of issues having to do with imprecise terminology, inconsistency between text and tables, and unstated assumptions were discovered during the OMT modeling.

Cassini Requirement: If Spacecraft Safing is requested via a CDS (Command and Data Subsystem) internal request while the spacecraft is in a critical attitude, then no change is commanded to the AACS (Attitude and Articulation Control Subsystem) attitude. Otherwise, the AACS is commanded to the homebase attitude.

```
saf: THEORY
% Example is excerpted from saf theory.
% Spacecraft safing commands the AACS to homebase mode, thereby
% stopping delta-v's and desat's.
BEGIN

aacs_mode:  TYPE = {homebase, detumble}
attitude:   TYPE

cds_internal_request:   VAR bool
critical_attitude:      VAR bool
prev_aacs_mode:         VAR aacs_mode

aacs_stop_fnc (critical_attitude, cds_internal_request, prev_aacs_mode):
   aacs_mode =
   IF critical_attitude
      THEN IF cds_internal_request
         THEN prev_aacs_mode
         ELSE homebase
         ENDIF
      ELSE homebase
```

symptoms of the original fault, whereas the lower level requirements correctly cancel a lower priority fault response to handle a higher-priority one.

Imprecise terminology: 6. These were largely documentation problems, including synonyms and related terms, and were revealed during the process of defining the PVS model.

suggest a fourth model, in which formal modeling is used to increase quality during the requirements and high level design phases, without necessarily producing a baseline formal specification, or verifying low level design and code.

Our studies also demonstrate that questions of tool support need not be a barrier to the adoption of formal methods. We conducted sophisticated validation of our models, via theorem proving and model checking, using tools that are essentially still research prototypes. In the 12 case studies surveyed by Gerhart *et al* [24], tool support was generally only used for syntax checking of specifications, and Gerhart suggests tool impoverishment is a barrier to wider use of formal methods. This may be true for the more complete process models used in case studies of the kinds described by Kemmerer [25], Hall [1] and Gerhart [24], but is not true of the 'lightweight' application of the kind we adopted.

Most of our observations of the benefits of formal methods are consistent with findings elsewhere. For example, we noted that a large number of minor problems are discovered during the process of formalising the requirements, and that the use of formal methods helps to focus attention on areas that are more susceptible to errors [26]. Formally challenging the models uncovered a smaller number of more subtle issues, of the kind that are hard to detect manually. Like Hall [1], we found that the use of intermediate, structured representations greatly facilitated the process of formalising the requirements.

Although we have not attempted any detailed quantitative analysis of the costs and benefits of the application of formal methods in these studies, it is clear that in each case the study added value to the project by clarifying the requirements and identifying important errors very early in the lifecycle. The costs, in terms of time and effort, were consistent with existing V&V tasks on these projects.

A number of observations arising from these studies are worth further discussion:

### *Who should apply the methods?*

In each of the studies, the formal analysis was conducted by experts in formal methods, who were external to the development project. There was a simple financial reason for this: it is cheaper and lower risk to have a small team of formal methods experts develop the specifications and perform the analysis than it is to train members of the development team. Our longer term goal is to have the developers produce formal specifications themselves, with a V&V team performing the analysis.

However, there are some interesting consequences of our use of external experts. Developing formal models of informal specifications involves a great deal of effort in understanding the domain, and figuring out how to interpret the documentation. As our external experts were unfamiliar with the projects prior to the studies, they did not share the assumptions that the requirements' authors had made. Our experts questioned everything, spurred on by the explicitness needed to build the formal models. They also needed to present parts of their models back to the developers, in order to check the accuracy of their interpretations. The result was a healthy dialogue between the developers and our formal methods experts. This dialogue exposed many minor problems, especially unstated assumptions and inconsistent use of terminology. This dialogue was clearly an important benefit.

Another aspect of this dialogue was that some of the issues that were raised were the result of misunderstandings by our experts, rather than genuine errors. The requirements authors therefore had to filter the issues, to pick out those for which the benefits of changing the requirements out-weighed the cost. This was especially true when the analysis revealed "interesting" off-nominal cases. A great deal of domain knowledge was needed to judge whether such cases were reasonable. The need for such filtering would be greatly reduced if the analysis was conducted by domain experts; however, the risk of analysis bias would then increase.

### *Is formal modeling of volatile requirements worthwhile?*

During early stages of the requirements process, there may be a great deal of volatility. In each case study, some effort was needed to keep the formal model up to date with evolving requirements. However, the studies indicate that there is no need to wait for the requirements to stabilize before applying formal methods. Early formalisation allowed us to crystallize some of the outstanding issues, and explore different options. Most importantly, it is during this early phase that the development team is most receptive to the issues raised from the formal modeling. This again emphasizes the importance of lightweight formal methods: the formal model itself can be discarded if the requirements change significantly, while the experience and lessons learned from it are retained.

### *Were intermediate representations useful?*

Intermediate representations were an important part of the formalisation process in each study. The type of intermediate representation varied across the studies: the first study used an annotated version of the original FCD flowchart, the second study made use of truth tables to clarify complex predicates, while the final study made extensive use of OMT diagrams. A large part of the effort in the formalisation process lies in understanding the existing requirements. These intermediate representations helped to refine this understanding, and therefore reduced the effort needed to generate and debug the formal models.

The intermediate representations also helped to create some initial structure for the formal models. They assisted with traceability between the formal and informal specifications, making it simpler to keep the formal model current. From our experience thus far, it seems that this benefit more than outweighs the extra cost of maintaining several representations, even during the early stages when requirements are most unstable.

## , Con on

The three studies described here were conducted as pilot studies to demonstrate the utility of formal methods and to help us understand how to promote their use across NASA. An important characteristic of these studies is that in each case the formal modeling was carried out by a small team of experts who were not part of the development team. Results from the formal modeling were fed back into the requirements analysis phase, but no attempt was made to introduce formal specification languages for baseline specifications.

We have shown that lightweight formal methods complemented existing development and assurance practices in these projects. If formal methods is seen as an additional tool in the V&V toolbox, then widespread application to existing large projects becomes feasible.

As a follow-up to the studies described here, we have begun to investigate the role of formal methods in the development of new spacecraft technology. As part of NASA's New Millennium program, new architectures are being developed using knowledge based systems to reduce the reliance of the spacecraft on ground support. Rather than produce a detailed statement of requirements, the project is using a rapid prototyping approach to explore the