Improving FDIR of Spacecraft Systems with Advanced Tools and Concepts

Eric Bensana, Xavier Pucel, Christel Seguin

Improving FDIR of Spacecraft Systems with Advanced Tools and Concepts

Eric Bensana, Xavier Pucel, Christel Seguin
ONERA Toulouse, 2 av. Edouard Belin
31077 Toulouse CEDEX 4

July 5, 2013

Faults in spacecraft systems are an important problem, mainly because of the cost of downtime, and because their remoteness makes maintenance more difficult. This is why automated handling of faults can greatly enhance the system overall performance. This automated fault management relies on dedicated functions for fault detection, identification, and recovery (FDIR), that are often interleaved with the system, which makes it difficult to guarantee tolerance with respect to a particular anomaly, and makes the system difficult to maintain as well. On the other hand, several advanced computational tools exist that are known to support the tasks of FDIR. In this paper, starting from the current state of affairs in spacecraft system development, we develop and test several options for enhancing the quality of FDIR functions. First, we use software validation and verification tools to prove that the FDIR functions meet some functional quality goals. A second option we explore is to re-implement FDIR functions by Model-Based Reasoning algorithms, that are guaranteed to produce exact results with respect to a model of the system's behaviour. In each option, we use and compare several software tools, we compare the effort required to adapt, integrate and use them, and estimate the overall benefits they provide.

1 Verification and Validation of FDIR

Verification and validation of FDIR functions aims at offering guarantees of functional quality, as in how the system reacts to faults. It also supports the prototyping and early development of FDIR functions, which can greatly enhance their integration and utility in the system. These approaches rely on a formal representation of the system's behaviour, in absence and in presence of faults, as well as a representation of the quality requirements in the same formalism. Because these approaches involve computationally expensive algorithms, the modelling phase is critical for the success of the validation tasks, and we show in two studies how this complexity can be handled, for instance by analysing subsystems separately, or by using abstract or situational models.

In a first study with *Astrium*, we have validated the FDIR functions for a satellite thermal control system by analyzing separately 3 aspects of the system in 3 different modelling tools: the system architecture with *Altarica* and

Cecilia OCAS, the control software with *Scade 6* and the physical components with *HyTech*. In each case, the tool offered both a language for modelling the subsystem's behaviour and FDIR requirements, as well an a model specific algorithm for checking which requirements were met.

In a separate study with *Thales Alenia Space*, we have validated an Attitude and Orbit Control System the *COMPASS* and *SLIM* tools. Separate analyses were performed for a model of the nominal behaviour (in absence of faults) and for a model that accounts for faults. In both cases, the model is qualitative and focuses on the system's architecture and ability to make use of the component redundancies.

Experience shows that although modelling a system's behaviour and requirements can be difficult, validating FDIR algorithms helps discovering design problems and correcting them. Validation tools have proven to be useful for critical systems development in general, and for FDIR modules in particular.

2 FDIR with Model-Based Reasoning

Model-Based Reasoning for FDIR is a different development approach, that rely on formal models of the system's behaviour, and analysis algorithms. It usually requires the encapsulation of the FDIR functions in a separate module, which makes validation and maintenance easier, at the potential cost of increased development time due to the architectural constraint and the complexity of the analysis algorithms. In this section we present two studies: the first study consists in the complete implementation of a FDIR module in a satellite flight control software demonstrator, the second study consists in the integration of academic diagnosis modules in a flight control software.

2.1 A BDD based FDIR module

The first implementation of a Model-Based FDIR module took place in a flight control software demonstrator from *CNES*. The system was already implemented in a hierarchical architecture. Some equipment redundancy mechanisms were already in place at the lowest level of the hierarchy, while at the topmost level several global reconfiguration modes could be activated, such as turning off the payload, or switch to safe flight mode. We used a custom made Binary Decision Diagram (BDD) library to build our FDIR module. We expressed the precondition for each reconfiguration action as a BDD, and implemented a state monitoring mechanism that encodes the system's current state as a BDD as well. Our FDIR module had a hierarchical architecture that maps that of the system, so that state monitoring and reconfiguration actions could be taken at any level of the architecture, thus restricting the affected part of the satellite and effectively containing the faults. BDD tend to lower memory consumption, and can easily be implemented without dynamic memory allocation, which makes them suited for satellite systems. The implementation was successful.

2.2 Complex tools and models

The second implementation of a FDIR module aimed at integrating and comparing several Model-Based Diagnosis tools. The challenge behind this integration

relies on the fact that each tool provides different interpretations for diagnosis : the *Lydia* tool from TU Deltf produces sets of faulty components, the *Diades* tool from LAAS a classification of faults, the *Livingstone* tool a trajectory (i.e. a sequence of events) that explain an abnormal behaviour. We have also included a custom made diagnosis engine based on *Prolog Eclipse* based on predicate logic. Despite the variety of tools, we have established a common interface, based on common concepts such as modes, commands and faults, common data types such as assignments, observations, candidate and diagnosis, and common functions, such as initialization, model loading, observation and analysis, and integrated the four diagnosis engines in a flight software simulator. While the integration of the diagnosers was successful, it was difficult to evaluate the optimal granularity of the models to produce a useful diagnosis while containing the algorithmic complexity of the diagnosis reasoning. Moreover, it exhibited that without being guided by a collection of possible reconfigurations, it is difficult to judge how precise a diagnosis, and how precise a model, is needed.

Our Model-Based implementations of FDIR modules have confirmed our expectations in terms of greater modularity and maintainability. We also claim that they offer greater potential for validation, since the modeling formalisms for Model-Based FDIR are similar to those used in validation. We have discovered that modelling the system can be difficult, especially when there is no clear specification of the FDIR objectives. Such specification can be a predefined set of possible reconfiguration actions that must be taken automatically, which when available makes the implementation possible and efficient.