

FDIR STRATEGY VALIDATION WITH THE B METHOD

Denis Sabatier⁽¹⁾, Brice Dellandrea⁽²⁾, David Chemouil⁽³⁾

(1) ClearSy System Engineering, 320, av Archimède, les Pléiades III Bât A, 13857 AIX-EN-PROVENCE, FRANCE. Email: Denis.sabatier@clearsy.com

(2) Thales Alenia Space, 100 bd du Midi BP 99 06156 CANNES la Bocca Cedex, FRANCE. Email: brice.dellandrea@external.thalesaleniaspace.com

(3) CNES, 18 avenue Edouard Belin, 31401 TOULOUSE Cedex 9, FRANCE. Email: david.chemouil@cnes.fr

ABSTRACT

In a formation flying satellite system, the FDIR strategy (Failure Detection, Isolation and Recovery) is paramount. When a failure occurs, satellites should be able to take appropriate reconfiguration actions to obtain the best possible results given the failure, ranging from avoiding satellite-to-satellite collision to continuing the mission without disturbance if possible. To achieve this goal, each satellite in the formation has an implemented FDIR strategy that governs how it detects failures (from tests or by deduction) and how it reacts (reconfiguration using redundant equipments, avoidance manoeuvres, etc.). The goal is to protect the satellites first and the mission as much as possible.

In a project initiated by the CNES, ClearSy experiments the B Method to validate the FDIR strategies developed by Thales Alenia Space, of the inter satellite positioning and communication devices that will be used for the SIMBOL-X (2 satellite configuration) and the PEGASE (3 satellite configuration) missions and potentially for other missions afterward. These radio frequency metrology sensor devices provide satellite positioning and inter satellite communication in formation flying. This article presents the results of this experience.

1. CONTEXT

In formation flying mode, the position of each satellite and the inter-satellite communications are obtained thanks to devices called RFS (radio frequency sensors). These devices can transmit radio messages and measure relative positions by comparing different reception points. The formation is organised as slaves communicating to a single master, each satellite having a spare RFS as shown in Fig.1. There are no slave-to-slave communications.

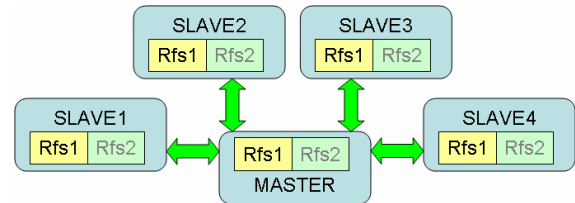


Figure 1. An example with 4 slaves.

Of course, each satellite has a predefined time slot for emission, thus avoiding 2 satellites emitting at the same time. All slaves synchronize their clocks using the master messages, and emit in their time slot. The time slot scheme is fixed, preconfigured and known by each satellite.

The FDIR mechanisms are embedded in the on-board computer of each satellite. Their role is to turn a satellite to RFS2 if needed and to quit the formation-flying mode when necessary, in particular before any satellite-to-satellite collision. To avoid random behaviour, a FDIR program can suspend the applicative functions while taking actions. FDIR programs can exchange data on inter-satellites links, as long as these links are functional enough.

When a FDIR undertakes such actions, it creates periods in which the satellites will certainly not be able to communicate or to measure positions. In particular, turning to RFS2 takes time because the new RFS needs to boot, then to negotiate different kind of information with its counterpart before being fully functional. Of course, turning the master to RFS2 has the largest impact.

2. DETERMINING GENERAL GOALS

Once the context is understood, it might be a good idea to guess the general goals that the chosen FDIR should achieve. By common sense, a perfect FDIR should turn on RFS2 immediately on a given satellite if and only if RFS1 has a potentially dangerous cause of failure, whatever the cause. This supposes that every satellite could immediately detect any dangerous cause on its RFS. This is obviously unfeasible.

In this “God mode”, the FDIR would be straightforward: if my RFS1 has a dangerous defect, turn to RFS2, if already done quit the formation. The distributed nature of the satellite formation would be of no importance then. So the detection tests available to satellites for determining the defects are paramount. Realistic goals for the FDIR mechanisms cannot be chosen without realistic hypotheses about the detection tests. Let’s examine detection tests first.

2.1. Causes, replacements and tests

Obviously each failure has a cause that once removed will remove the problem. To remove a cause, the only available action on each satellite is to replace the running RFS if a spare one is available, i.e. to turn to RFS2 if not already done. This will solve all the problems due to causes on this RFS. So causes are linked to possible replacements in a predefined scheme. Some causes are not linked to any replacements: they are either external perturbations or failures of other devices than RFS, with no spare and thus unfixable.

We have access to the causes (and thus to the replacements) by measuring their effect, by way of detection tests. When a test gives a positive detection, it is linked to a list of possible causes with probability. In general, a negative detection gives a more versatile information. As before, the link between tests and causes can be considered as predefined and known.

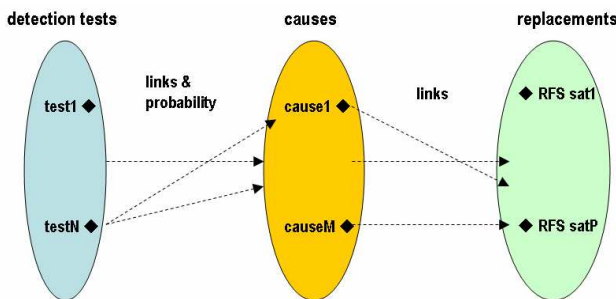


Figure 2. Causes, tests and replacements.

In fact, detection tests could be false alarms: then we have a test related to no cause. We will assume that this never happens, because then it would be impossible to prove any property concerning good justifications for replacing a RFS or breaking the satellite formation.

2.2. Detection tests: functions tests

A first idea to detect failures is to measure errors or missing messages in a certain time length. Let’s D be the time length and Emax the maximum number of errors, we can use the condition “between now minus D and now, there has been more than Emax errors” to define the presence and absence of the considered failure. We will call this a “function test”.

When the test indicates no failure, it characterizes the state of the system at now minus D: a failure can have occurred since then. When the test indicates a failure, it can give as an extra information the date of the last error of the Emax sequence. This date is between now minus D and now, and the state of the system was incorrect at this date.

This kind of test has the advantage of verifying the true functionality. In particular, it is one of the rare ways to be sure that a cause has really disappeared, or at least that its effects have become sparse enough to be tolerable. Thus, it is a good way for a FDIR program to decide when to remove the application suspension.

As seen before, these tests give either a negative answer (no defect) related to now minus D, or a positive answer (defect) related to now minus something lower than D. In periods when a positive answer is normal (for instance, for communication tests after a RFS reboot), the test can be performed as well: we just have to remove all positive answers related to dates in these periods. By definition, negative answers prove the defect disappearance. The time sequence (Fig. 3) below illustrates this.

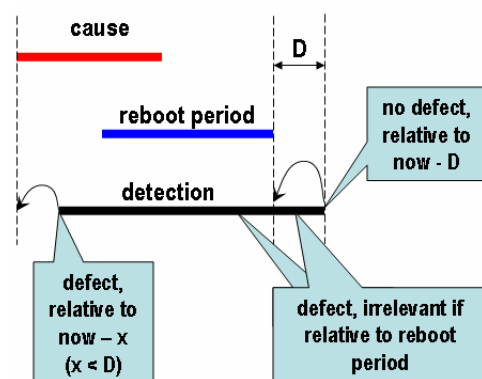


Figure 3. Detection by a function test

So, function tests can be seen as simple Boolean conditions with an associated date if positive.

2.3. Detection tests: explicit detections

Another interesting category of tests are the tests that detect specific abnormal events, either thanks to hardware means or to occasions for soliciting less frequently used functions. In general, a negative detection by these tests is not sufficient to prove the absence of causes that could disturb a given function. Reversely, a positive answer is generally an explicit proof that a failure is present. We will call this kind of tests “explicit detections”.

Explicit detections can be seen as Boolean values turned to TRUE on specific events, at dates when the detection occurs. At least one cause is then certainly present, and

this remains true as long as no replacement is done. After a replacement chosen through the predefined list of possible causes, this certainty of a fault disappears (reset of the associated Boolean value). If the real cause is indeed still there, we admit that nothing has to be done until its next explicit detection.

Note that an explicit detection could be related to external causes that might disappear suddenly. Then we admit that the possible replacement made in response to the detection was unavoidable. In other words, if confirmation delays are needed to get rid of external transient causes, they must be included in the test itself. More generally, using this kind of tests if sensible to external conditions might not be a good idea.

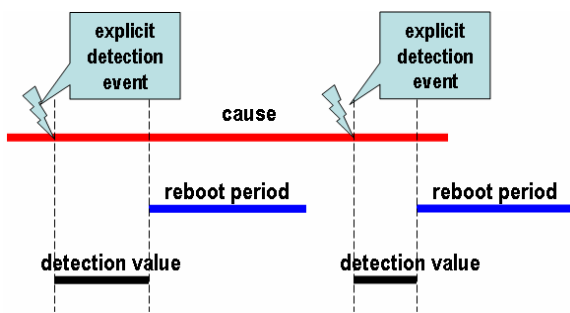


Figure 4. Explicit detection test

2.4. FDIR global goals, revisited

All the detection tests that we considered, either function tests or explicit detections, can be seen as Boolean values. Each test is either TRUE or FALSE, and the idea is to define our goals in terms of actions regarding these detections.

In this distributed scheme, it is not always easy to use our detection tests optimally. For instance, if a test on one satellite can detect a failure with another satellite RFS as only possible replacement, communication problems could render impossible the transmission of this knowledge to the concerned satellite. So we have a non ambiguous diagnostic and no way to take the appropriate action. Another example concerns the irrelevance of a function test during reboots. If it's the reboot of another satellite, it might be difficult to guess the time length during which positive detections of this function test should be ignored. This pitfall could at worse result in unnecessary replacements.

For all these reasons, at this stage we define only goals "in principle", without precise enough criteria to decide or prove if they are fulfilled.

Preliminary goal 1: remove irrelevant detections. If a positive detection is related to a date when it might be irrelevant (date during an action for a function test or

date before an already started replacement for an explicit detection), avoid as much as possible to start any new action for this detection.

Preliminary goal 2: minimize unnecessary actions and blockages. If several positive detections occur, the next replacement should if possible be chosen using the intersection of the possible causes. Unnecessary blockages (application suspension for instance) should be minimal.

Global goal: react as fast as possible to any positive detection, by justified replacement or formation breaking. We assume, of course, that the detection tests are chosen to avoid false alerts.

Not surprisingly, our revisited main goal matches what we said at the beginning of paragraph 2, only now we define it regarding a set of detection tests. We cannot be more precise with these goals without knowing more about the tests, in particular about the link between some tests and the availability of communications, which permits the FDIR centralization.

3. FDIR STRATEGY FROM SIMBOL-X

For the missions SIMBOL-X and PEGASE, Thales Alenia Space has developed FDIR strategies. We can now examine one of these strategies using the above notions. For confidentiality reasons, we won't go into too specific details.

3.1. Principle of operation

The chosen FDIR strategy is based on a set of tests that we categorized as follows:

- A set of tests that can be seen as a single communication function test, let's call it PbComm;
- A set of tests that concerns the position metrology, and can be seen as an explicit detection test, let's call it PbMetro;
- Other communication tests that can be seen as explicit detection tests for the communication;
- Hardware built-in tests that can be seen as explicit detections with all possible causes located in the local RFS.

All these tests are available on each satellite, master or slave. For the master, each test (excepted built-in tests) are specific to each master-slave link. All tests (excepted built-in tests) have possible causes located in the local RFS, or in the RFS at the other side of the link, or external (non-RFS) causes. Every positive test is considered as a valid justification for at least one replacement, and the application suspension should remain until PbComm has disappeared.

The chosen FDIR is described as a combination of state machines. The normal state of each satellite is called RFS_OP, while the formation-breaking one is RFS_NOP.

3.2. Low level B models

We first described all the possible transitions as B events, without any causality links, in a draft model. Thus, we have events concerning the evolution of each satellite in a given state, and events modelling the environment. The following figure illustrates this:

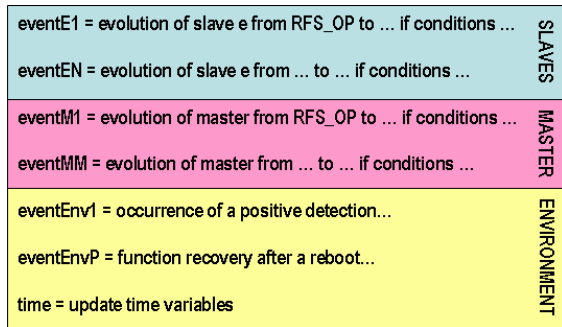


Figure 5. Unlinked draft model

As the FDIR algorithm is based on timeouts, we use time variables updated by a so called “time event”. It’s in fact a way to introduce orders between events: if some action was started when entering a certain state, and if we know that this action should terminate before the timeout of this state, then the event corresponding to leaving this state is guarded through time variables to the condition that the action is finished. Time variables at this level allows us to express time condition as they are coded in the program, leaving their abstraction into order of events laws for later.

In this model, causality links are not expressed. For instance, the detection of a communication problem (PbComm) is described in one of the environment events, and the reaction of the satellite is described in another one guarded by PbComm = TRUE but nothing indicates that the delay between those 2 events is in fact very short. More, causality links between satellites are not expressed: for instance, the event corresponding to the start of reboot of a satellite and the event describing a communication problem detected by its counterpart are not linked. Nothing in this B model indicates that the second event will rapidly follow the first one.

For these reasons, the draft model is unsuitable for demonstrating interesting properties. The idea is to group the sequence of linked events that occur in response to a single solicitation into one global event, as shown in the next figure.

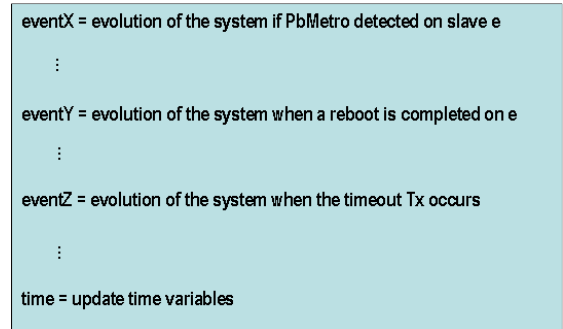


Figure 6. Linked low-level model

Only closely linked events should be grouped in that way, because new events occurring during the grouped sequence will no longer be considered in the linked B model.

For example, we could group the disappearing of PbComm on both satellites of a master-slave communication pair. In fact, this disappearing is obtained through mechanisms that involve some rapid data exchanges between the two satellites. If a new problem occurs during these exchanges, or if message buffering is possible somewhere, we can have strange situations with one satellite OK and the other in a different state, or even state instability. Once grouped, the B events assume that these cases never happen so we must be very careful that they are made impossible in practical. If not, specific B events should be added to model these cases.

Another difficulty when linking the events is that the number of possible sequences can be huge. For instance, if the unlinked draft model has N events that can occur in any order, in theory the linked model should contain N! events. Fortunately, the relevant sequences reflect real possible evolutions of the system in response to a single solicitation, so their number should be manageable if the system itself is manageable.

3.3. Goals redefined

Once the concrete algorithm is well understood, we can guess what properties are fulfilled and how these properties match the global goals defined before.

The 3 main properties found are:

- If there is any positive detection at t, the system cannot remain in RFS_OP (i.e.: at least one satellite will quit this state) in the time period from t to t plus a known constant.
- If the system quits RFS_OP and returns to this state afterward, at least one satellite has changed its RFS.
- If there is no positive detection for a given master-slave link during a period when the master left RFS_OP and returned to this state, then the concerned slave has not changed RFS.

These properties match the global goal: if any detection occurs, the system takes an appropriate action. The first law ensures that the system quits RFS_OP, the second one that if it returns to RFS_OP a replacement has been done and the last property ensures that this action is relevant. Note that it is still possible that the system finishes in RFS_NOP state without enough justification.

Some extra properties contribute to the preliminary goals 1 and 2 (remove false detections, minimize unnecessary action and blockages):

- A satellite cannot stay in states other than RFS_OP and RFS_NOP more than a known constant.
- If from a certain date t all RFS are and remain OK, if all satellites are either in RFS_OP or in after-reconfiguration states since less than the shortest timeout minus the longest D time of function tests, then the system will return to RFS_OP.

The first property forbids any loops in states between RFS_OP and RFS_NOP for a satellite. Note that loops with several satellites remain possible in theory. The second property is quite hard to write. For instance, it avoids cases where a system sane after reconfiguration would wait with some forgotten blockage until reaching RFS_NOP on timeout. Of course this does not work if one of the satellites is already in RFS_NOP, or in an after-reconfiguration state since too long (in this case, it will timeout to RFS_NOP before turning off all the detection tests).

3.4. High level models and proof

The form of a first level model is clearly suggested by the 2 first properties: one single state for the whole system, being either RFS_OP, RFS_NOP or something else, a single detection variable denoting the disjunction of all detection tests, and a replacement counter. At this level we have a very small number of transition events, as shown in the next figure:

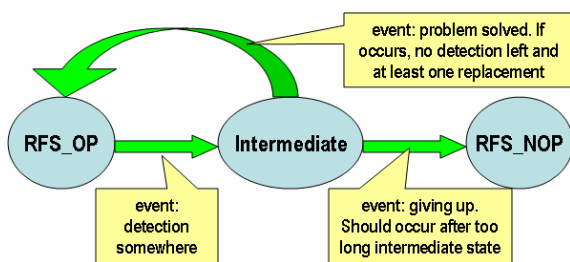


Figure 7. Simple high level B model

The transitive link from this high level model to the “linked low level model” presented before is quite clear concerning variables like global state or global detection. We say “transitive link” because this link crosses refinement intermediate levels. For events like the “problem solved” one, the link is connected to all

system evolution events in the low level model that *terminate* a sequence corresponding to the desired transition. This is illustrated in the next figure:

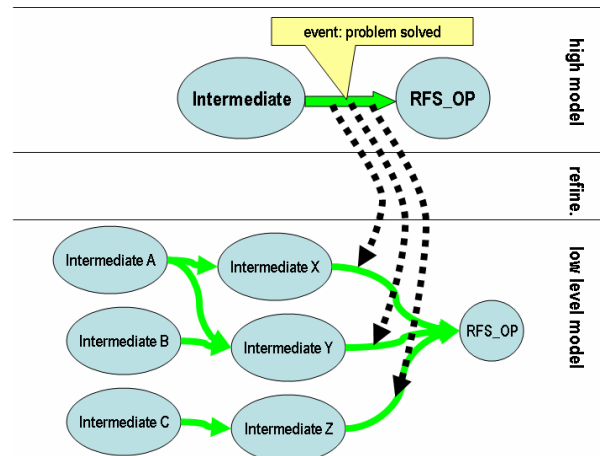


Figure 7. linking events

The “problem solved” event specifies that the replacement counter be increased of at least 1. With such a link, each low level linked evolution will have to prove that either it increases the counter, or that this counter was already increased in the previous state (intermediate X, Y or Z). This gives an idea of the mechanism of the proof.

3.5. Hypotheses necessary for proof

When conducting the proof reasoning, for each of the chosen properties some interesting hypotheses were found. We can examine these hypotheses on a sample of properties:

- *If there is any positive detection at t , the system cannot remain in RFS_OP (i.e.: at least one satellite will quit this state) within a period from t to t plus a known constant.*

In most cases the satellite detecting immediately quits the RFS_OP state, so it seems straightforward. Nevertheless, for metrology detection the strategy requires a successful dialog between master and slave before quitting RFS_OP. The property remains, because once a satellite has decided to start this dialog, it will never give up trying without quitting RFS_OP, and if the communication remains functional, we assume that the dialog will take less time than a known constant. If the communication fails, the satellite will quit RFS_OP without dialog.

- *If the system quits RFS_OP and returns to this state afterward, at least one satellite has changed its RFS.*

As we have seen, this concerns all the low level transitions returning to RFS_OP. These transitions

should not occur before at least one replacement. Some of these transitions start from a state where we can easily prove that a local replacement has been done, or a remote replacement (states not reached before a remote replacement). The problem is when the detection disappears before the first replacement. In the FDIR strategy studied, there is a mechanism using satellite-to-satellite exchanges to keep track of a disappearing detection. Therefore, if the detection disappears, this mechanism should prevent the formation from returning to RFS_OP without replacement. Thus, the proof of this property relies on this mechanism. We had to study this mechanism into deep details and even to add some hypotheses to make the proof possible.

One important point concerns the assumptions we made when grouping the events in the draft “separated parts” model into whole system linked events. In particular, we assumed that no message buffering was possible. This becomes paramount regarding the mechanism evocated above, using satellite-to-satellite exchanges to keep track of a disappearing detection. If message buffering was possible in this mechanism, it could lead to both satellites oscillating between RFS_OP and non-RFS_OP states. Fortunately, this can be avoided with some simple communication marker systems, and we assumed that it was indeed avoided.

3.6. Notes about methodology

In this experience, the most difficult part was to understand the context and the chosen FDIR mechanism before being able to define global goals. We often have to go bottom-up, understanding details just to guess good general laws. The risk when going deep into details is that the general laws do not emerge from details without a higher-level thinking, and reversely thinking about higher-level notions goes nowhere without knowing the real functions hidden within the details. Here this concerns the notions that finally emerged about detection tests (functional tests and explicit detections) and how they can be reset, and when it is allowed to resume normal operation.

If one refuses to go into details, or if one refuses to perform a more global reasoning, the process fails. Therefore, we obtain a “w” process, as illustrated below:

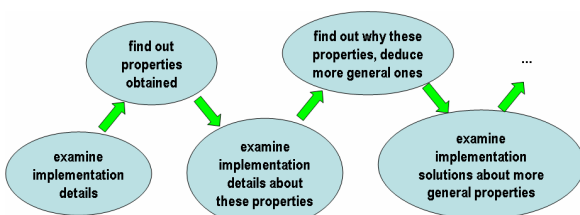


Figure 8. Process

In addition, when properties are found the chosen design is not always straightforward regarding these properties. For example, we can have a property stating that a certain state should not last more than a given limit, and this can be obtained through a combination of hypotheses and timeouts where a single global timeout would ensure the property directly.

4. TENTATIVE DESIGN FROM PROPERTIES

Now that we have a precise enough idea of the global goals and a first case study about how and to what extent they can be implemented, it is tempting to try to design a FDIR strategy directly derived from these properties. Nevertheless, such redesign “in theory” is not to be considered directly as an alternative, because it is not checked with real devices by real domain experts. We know that a design from the field often take into account hidden but paramount concerns.

4.1. Completing the context

As seen before, all the difficulty arises when the communication is broken, because then a satellite has to guess what its counterpart is doing or detecting to take the appropriate action. If the communication is functional enough, all the detection data can be centralized on the master and the master can decide all actions. To know if it is the case, we will assume that one specific function test (let’s call it PbComm again) denotes the ability to communicate over a given master-slave link.

In addition, we will assume that PbComm is positive or negative at the same time in the master and in the concerned slave. This is justified if the communication is done via numbered messages with a numbering system that links both ways. Then even if a satellite has for instance a defective emitter but a sane receptor, it will detect the communication loss at the same time as its counterpart because the received messages will become incorrect immediately. As every function is communication dependant, we assume that when PbComm is positive (communication broken) no other detection tests are necessary: all other functions are lost. Nevertheless, an explicit detection is still possible, for example, a hardware test that indicates which of the two concerned RFS is faulty.

For other detection tests, we assume as before function tests or explicit detections, all supposed to be a justification for a replacement.

4.2. Centralized FDIR

For all master-slave links without PbComm, the master can perform a centralized strategy. We can request that for every detection on these links, the master starts a replacement chosen using the intersection of all possible

causes associated to the present detections, with no delay (or more precisely, using the replacements indicated by the intersections of every groups having non-empty intersections)

Of course, any replacement will cause PbComm on the concerned link. We will assume a hypothesis on the decentralized FDIR strategy that we will describe later: *if the reconfiguration of a RFS starts right away when PbComm appears on a master-slave link and if the other RFS cannot cause PbComm, then the link comes back to no PbComm with only one RFS changed.* With this hypothesis, we can assume that the master can order RFS reconfiguration on any slave, or start reconfiguration of its own RFS, without any side effect from the decentralized FDIR.

All explicit detection tests shall be cleared if a RFS was replaced when PbComm disappears and if this RFS was a possible cause. All function tests shall consider PbComm periods as irrelevant detection periods. This ensures directly our preliminary goal 1 (remove irrelevant detections). We can also decide that the master shall order application suspension if and only if one link is defective with PbComm. This could match our preliminary goal 2 (minimize unnecessary replacements and blockages) if we consider that a blockage is always wise while one link is defective. The replacements done by the centralized strategy are never unnecessary because they belong to the intersection of all possible causes associated to the present detections. Note that for a cause randomly detected by explicit detections if the first RFS replacement does not solve the problem, at the next detection the master will change the remaining one (see Fig 4 in paragraph 2.3). Thus, we have the best possible actions regarding how the cause can be detected. The master shall break the satellite formation only when all the possible replacements associated to the detections are exhausted.

When detections leave several possibilities for actions, as long as the master chooses within the intersection of all possible causes associated we can assume that a specific choice function is used. This choice function can be optimized using the probability of occurrence of causes (see Fig 2 in paragraph 2.1) and any other clues available. There is no functional risk in this optimization, however complicated it gets.

4.3. De-centralized FDIR

When PbComm is raised on a master-slave pair, it is always possible that an additional explicit detection (for instance a hardware test) detects the faulty RFS on one satellite. This detection can occur after PbComm so the other satellite cannot have this information. To avoid unnecessary replacement, the other satellite should wait long enough for its counterpart to clear PbComm by

RFS replacement. We will assume that every satellite, when PbComm arises, waits at least the reconfiguration time of its counterpart, excepted if an explicit local detection indicates the local RFS as only possible cause. Lets call this rule 1.

With this hypothesis, we can request the following property: *if a detection test with only one possible replacement is positive since the beginning of a PbComm period, this replacement only shall be done* (if no additional causes arise after, of course). Note that if the detection test arises later in the PbComm period, the other satellite might reach its waiting limit and perform an unnecessary change of RFS. This is unavoidable: the 2 satellites cannot wait forever.

The above property is clearly obtained if both satellites obey rule 1. Also, the assumption we made on the decentralized strategy when defining the centralized one is correct: if the reconfiguration of a RFS starts right away when PbComm appears on a master-slave link and if the other RFS cannot cause PbComm, then the link comes back to no PbComm with only one RFS changed.

As we can see, it is no longer possible to guarantee that the replacements are chosen at best regarding the tests because this information is no longer centralized. We must define restricted properties like the previous one. But can we keep the property when giving up and breaking the formation, that says that all possible replacements shall be exhausted?

The answer is yes. One way to achieve this is to define a maximum time for local RFS reconfiguration when PbComm is on: when a slave has waited this time, plus a reconfiguration time, it knows that the master has changed RFS (and it has normally changed its local RFS). Then the satellite can quit formation flying. The same reasoning applies to the master. Therefore, we define rule 2: every satellite shall have a known maximum PbComm time after which it changes its RFS.

All decentralized strategies that comply to rule 1 and rule 2 and that quit formation flying only after both RFS are certainly changed fulfil our defined goals: no unnecessary replacement if immediate terminal detection, no unnecessary replacement interfering with the centralized FDIR, and quit formation flying only if possible causes are exhausted. To comply with preliminary goal 2 concerning blockages, we must assume that PbComm is evaluated continuously excepted during local reboots.

As long as a strategy complies to rule 1, rule 2 and the “quit formation flying” rule, it can use a specific choice function to decide when reconfiguring the local RFS. This choice function can be optimized using the

probability of occurrence of causes (see Fig 2 in paragraph 2.1) and any other clues locally available. There is no functional risk in this optimization, however complicated it gets.

4.4. Formalisation and Proof

This 2 level FDIR strategy with specific objectives for each level make things very simple. In fact, the formalisation of the chosen objectives is very close to their implementation at the abstraction level chosen for this study. So the associated B models merely describe the strategy, with simple proofs only on the decentralized part (for instance: possible causes exhausted when quitting the formation if we quit after the correct timeout).

5. DISCUSSION

As already discussed in paragraph 3.6, it took us some time and several “examine details” phases interlaced with several “guess global properties” phases to obtain a scheme suitable for formal proof and reasoning. When the scheme is ready and the key notions are clear, as usual it is tempting to redesign directly so that the wanted properties are satisfied by construction.

This suggests that the B method should be used during design phases rather than as an afterward validation method. In our experience, a team of both domain experts and formalization experts is the best way to find the design at the same time as when the way to prove its correctness is found, and to create a convergence of notions. This method of “design so we can prove” is probably more expensive at first than the “design and test all the cases” one, and is not suitable if the available amount is restricted to the minimum required to obtain a first usable product. A “**design constructed with its proof**” is an added precaution with a return over investment that can be great, but is always very difficult to quantify. These conclusions are quite general, we found nothing very specific to FDIRs here.

Another point concerns the knowledge of existing research work about failures and reconfigurations. The present study has been done using an existing FDIR as single input. With more time, it would have been possible to assimilate existing research results in this domain. One can guess that we would have obtained more well defined classical notions than the causes, tests divided into functional tests or explicit detections and replacement scheme that we used here.