Integrating Planning, Diagnosis, and Execution for Vehicle Systems Management

Gordon Aaseng[1], **Minh Do, Jeremy Frank, Chuck Fry, Adam Sweet**

[1]Intelligent Systems Division, NASA Ames Research Center
Authors listed in alphabetical order.

Abstract

We describe a prototype Vehicle System Manager (VSM) for NASA's Gateway, a human-capable spacecraft that will also be capable of autonomous operations. The VSM design focuses on replanning in the presence of faults. The VSM consists of an execution system, planner, and fault management system, integrated via an over-arching mission management component. We describe the VSM architecture and each of its components. We describe a series of use cases, centered on a spacecraft propulsive operation that can fail at different times, for different reasons, and how the VSM detects and responds to these failures. We show the VSM is capable of detecting faults and loss of capability, and subsequently replanning, in the presence of each failure scenario.

1 Introduction

NASA plans to construct a habitable spacecraft, currently referred to as Gateway (Crusan et al. 2018), in the vicinity of the Moon. Gateway consists of a Habitat, Airlock, Power and Propulsion Element (PPE), and Logistics module. Gateway will support up to 4 crew for 30 days. The PPE will provide orbital maintenance, attitude control, communications with Earth, space-to-space communications, and radio frequency relay capability in support of extravehicular activity (EVA) communications. The Habitat provides habitable volume and short-duration life support functions for crew in cislunar space, docking ports, attach points for external robotics, external science and technology payloads or rendezvous sensors, and accommodations for crew exercise, science/utilization and stowage. The Airlock provides capability to enable astronaut EVAs as well as the potential to accommodate docking of additional elements, observation ports, or a science utilization airlock. A Logistics module will deliver cargo to the Gateway.

The need for autonomy for Gateway is reflected in the Gateway concept of operations (Crusan et al. 2018) and its requirements, as well as requirements for the first of its components, the PPE (NASA 2018). NASA has developed and tested multiple technologies to enable the autonomous operation of a dormant space habitat. These technologies include a Vehicle System Manager (VSM), integrated with flight software, to control the habitat. The work in this paper builds on previous VSM work (Badger, Strawser, and Claunch 2019; Aaseng et al. 2018) to develop and demonstrate autonomy technology using contemporary flight software and automated reasoning technology.

While plan synthesis for complex operations such as those for Gateway is an interesting problem, we focus on the problem of managing faults. When there are no faults or unexpected events, there is no need to replan, and execution of the current plan proceeds without disruption. Managing faults requires detection of the faults, replanning and execution to respond to faults, including in the event that mission objectives can no longer be accomplished when the fault occurs.

For the purposes of the VSM, we are focused on the impact of faults on mission plans. It is important to distinguish *faults* from *impacts*. Fault response may depend not only on what fault occurred, but its impact. The loss of a single component may reduce mission capability (the loss of a spacecraft's only engine means it can't perform propulsive maneuvers) or cause a loss of redundancy (the loss of one of two engines still leaves a spacecraft able to maneuver) . Understanding the fault management technology and level of abstraction is important when discussing the design of the VSM as a whole. For example, when faults are detected at a low level of abstraction (say, part of a spacecraft power system), it is necessary to map faults to impacts at the level of abstraction at which plans are represented.

Faults can be transient or permanent; for the purposes of this paper, transient faults can be mitigated via fault recovery plans or behaviors, and permanent faults are detected by observing that the fault persists after mitigation efforts. This functionality can be implemented in different ways, i.e. at the level of an executive, or at a higher level of mission management. In general, fault impacts may lead to replanning. This requires the VSM to integrate information from the fault management system and other sources to both decide whether replanning is needed, and how to construct the planning problem to solve.

The main architectural components of the VSM include Fault Management, Execution and a Planner. The integration of all components of a VSM requires an over-arching component that integrates information from, and orchestrates the behavior of, each of the individual components described above. Hearkening back to early work in autonomous agents, we introduce a Mission Management and Metareasoning function (M3) that executes a continuous

control loop, watching for faults and impacts, and replanning when necessary. The key challenges to overcome arise due to the specific interfaces of planning, scheduling and fault management, and how M3's design is driven by these interfaces. We describe a series of scenarios based on a propulsive maneuver that can fail for different reasons, at different times. We demonstrate the VSM's ability to detect faults and replan in several scenarios derived from this use case.

In Section 2, we describe a replan/rescheduling problem that involves *temporal* and *causal* relations between tasks with different priorities. We describe each of the components of the VSM in more detail in Section 3. In Section 4 we walk through several scenarios to describe how the VSM detects and replans in the presence of faults. We describe previous work in Section Section 5. We conclude the paper with some of our future work in Section 6.

2 Motivating Scenario: Spacecraft Burn Replanning

Problem Specification: our scenario is centered on a major propulsive maneuver, or "burn", during which one or more engines fire to change Gateway's trajectory or orbit. It is common practice to ensure there are two possible (sets of) engine(s) prepared to conduct important maneuvers at a specified time; this can be thought of as a form of contingent plan. The optional engine choice is referred to as the "down-mode". For our scenario, the primary engine is a single powerful engine able to generate a large amount of thrust. The secondary set of engines that can be used in the "down-mode" are less powerful engines. The nominal burn task is referred to by `MAIN`, the task performed in the event of a down-mode is `BACKUP`. There are two important timepoints related to the burn period: *time of ignition (TIG)* and *engine shutdown (ES)* (fancy names for start time and end time of the burn).

There are a series of tasks in the plan constrained relative to the *TIG* and *ES*. The temporal constraints on different tasks and time-windows are described below (with time units measured in minutes):

- `MAIN` and `BACKUP` burn durations are respectively 4 minutes and 1 hour.
- The burn is required to be within a `Thermal Excursion` period. `Thermal Excursions` can last at most two hours. Consecutive `Thermal Excursions` must be separated by at least 10 hours.
- The backup engines need to be heated by executing a `BACKUP Preheat`; the constraint for this action is `[TIG - 90, ES]`. This contingent action is needed because if the `BACKUP` down-mode burn is needed, it may not be possible to pre-heat those engines immediately. It is unconditionally executed (even if down-mode is detected early enough.)
- If the main engine is used for the burn, we turn off the main engine heaters, and execute `MAIN Htr Deact` during the burn; the relevant duration constraint is `[TIG - 20, ES + 60]`. Note the `MAIN Htr Deact` action can be removed if the burn down-mode is detected and occurs early enough, i.e. before `TIG - 20`. Otherwise, if this action was started and the burn down-mode occurs late (i.e. in the interval `[TIG - 20,TIG]`), it will end when the longer `BACKUP` burn ends.
- Other burn related tasks and their respective durations are: (1) `Doppler & Ranging`: `[TIG - 390, TIG - 30]`; (2) `Burn Doppler` : `[ES + 30, ES + 60]`; (3) `Tank Pressurization`: `[TIG - 120, TIG - 60]`; (4) `Data Recording & Downlinking`: `[TIG - 15, ES + 5]`.

In addition to the burn related tasks, there are 3 payload tasks (`DFTO-1`,`DFTO-2`,`DFTO-3`) that occur after the burn. The temporal constraints on those planned payload tasks are:

- Payload task durations are respectively `dur(DFTO-1) = 30`, `dur(DFTO-2) = 30`, `dur(DFTO-3) = 60`.
- Payload tasks can not overlap.
- `DFTO-1` and `DFTO-3` require a `Thermal Excursion`.
- `DFTO-3` must be performed within the time-window of `[TIG + 660, TIG + 780]`
- a 6-hour `Ranging` activity starts after `DFTO-3` ends.

In addition to the constraints above, there are *preferences* the users would like to see satisfied in the final solution:

- `DFTO-1` ideally should occur within the 2-hour time window `[TIG + 570, TIG + 690]` where the *70m DSN* antenna is visible (DSN = Deep Space Network). There is a `DFTO-1-Downlink` activity contingent on this constraint being respected.
- `DFTO-1` ideally precedes `DFTO-2`.

These preferences can be weighted, but for our purposes, it is only important to know a total order on the two preferences, in the event both can't be satisfied.

Failure Scenarios: As described in the introduction, our VSM is primarily designed to detect and respond to faults. This scenario focuses on faults that may cause the primary engine to be unavailable at *TIG*, causing the engine down-mode.

Fault vs Impact: We differentiate between what has failed (e.g. a controller card or switch) and the impact (loss of engine that requires the burn down-mode). The VSM must be able to determine both what has failed, and the impact of this failure on the plan. Looking ahead to Figure 2 (bottom) we see that there are two independent paths of control to the main engine. This makes the propulsion system 2-fault tolerant; that is, two independent faults are required to lose the ability to fire the main engine.

Permanent vs. Transient Fault: the faults in our scenario can be either *permanent* or *transient*. Transient faults can only be detected by trying recovery plans and observing their success or failure. Recovery plans can be attempted only a limited number of times, in this case, 3. The VSM must have

the ability to determine whether a fault, and subsequent loss of capability or redundancy, is a transient or permanent fault.

Impact of down-mode on the plan: Since the BACKUP burn duration is 15 times the MAIN burn duration (60 min vs 4 min), all burn-related activities that are tied to the Engine Stop (ES) time will need to be rescheduled to different times if there is a burn down-mode. Moreover, if the down-mode happens before the MAIN Htr Deact task starts, then we can remove it from the new plan, but otherwise we must leave it in the plan, and extend its duration. Finally, the required separation of thermal excursions will also cause replanning of the DFTO activities at the end of the plan and possible omission of DFTO-1-Downlink. We do not consider faults impacting the backup engine.

The table below illustrates the main features of the burn down-mode scenarios we have described above. There are a total of 4 faults that could occur; of the 15 possible sets of faults that occur, we can divide the scenarios into those in which the two permanent faults occur, and those in which one or no permanent faults occur. Only in the event of two permanent faults do we need to consider when the second permanent fault has occurred, either before or after PCA-SM-Heater-Deactivation has started. Finally, there is the preferences on the DFTO constraints. The result is 4 main fault scenario variations (transient only, one permanent fault, fault timing, and preferences). Table 1 shows the number of options for each of the three faults and planner preferences, and notes on each variant.

Class	Property	#	Notes
Fault	RPC-open	2	Transient
Fault	PDE-Failed	2	Permanent
Fault	Timing	2	Before/After MAIN Htr Deactivation start
Planner	Preferences	2	DFTO order vs DSN coverage

Table 1: Scenario Variations.

3 VSM Design

The VSM architecture consists of multiple software components, integrated via the cFS flight software message bus architecture. The *Planner* is responsible for revising the burn plan in the event of fault or loss of capability resulting in burn down-mode. The *Executive* executes plan tasks and returns results. Fault management is divided into three components: *Fault Detection (FD), Diagnostic Executive (DE)* and *Fault Impacts Reasoner (FIR)*. Finally, all components are overseen by the *Mission Manager and Metareasoner (M3)*. We implemented our VSM using a flight-software framework adopted by the Gateway program. Core Flight System (cFS) is a platform and project independent software framework and set of reusable software applications (McComas, Wilmot, and Cudmore 2016). There are three key aspects to the cFS architecture: a dynamic run-time environment, layered software, and a component based design. It is the combination of these key aspects that makes it suitable for reuse on any number of NASA flight projects and/or embedded software systems. Figure 1 shows our key autonomy software components operating within the cFS software framework. Our software components are built as Core Flight System (cFS) applications and operate over the cFS bus, exchanging messages with other components and utilize cFS services. Subsequent sections are dedicated to explain their roles and inter-operation in more details.

3.1 Planner

While the plan as described above includes a large number of features such as causal links, temporal constraints, preferences, and contingent actions, for our purposes the plan has been generated offline, and so we don't require the planner to perform complex de-novo plan synthesis. Instead, the problem to solve is plan repair in the presence of the loss of the main engine and subsequent burn down-mode. The replanning problem is defined as an MILP; we use lp_solve to implement the Planner.

MILP Encoding: Apart from the burn, for each task X in our scenario, there are the following variables:

- t^s_x: start time of X (continuous floating point value).
- t^e_x: end time of X (continuous floating point value).
- Binary variable b_x with $b_x = 1$ means X exist in the final plan and $b_x = 0$ means it is not in the final plan.

Besides variables associated with tasks, there are the following various time-points:

- t_{TIG}: Time of Ignition.
- t_{ES}: Engine Shutdown time.
  - For the nominal burn, when MAIN is executed, then $t_{ES} = t_{TIG} + 4$.
  - If faults lead to a burn down-mode and BACKUP is needed instead, then $t_{ES} = t_{TIG} + 60$.
- t_{bd}: Burn down-mode occurrence time.
- 70m DSN time window $[t^s_{70m}, t^e_{70m}]$ of 2 hours that starts at 9.5hrs after t_{TIG}: $t^s_{70m} = t_{TIG} + 570$ and $t^e_{70m} = t^s_{70m} + 120$.
- Thermal Excursion Windows TE1 and TE2 that are 10 hours apart and lasting up to 2 hours each: $[t^s_{TE1}, t^e_{TE1}]$, $[t^s_{TE2}, t^e_{TE2}]$
  - $t^e_{TE1} \leq t^s_{TE1} + 120$: up to 2hrs long.
  - $t^e_{TE2} \leq t^s_{TE2} + 120$: up to 2hrs long.
  - $t^s_{TE2} \geq t^e_{TE1} + 600$: separate by 10hrs.
- Three fixed-duration payload (DFTO1-3): $[t^s_{DFTO1}, t^e_{DFTO1}]$, $[t^s_{DFTO2}, t^e_{DFTO2}]$, and $[t^s_{DFTO3}, t^e_{DFTO3}]$
  - $t^e_{DFTO1} = t^s_{DFTO1} + 25$
  - $t^e_{DFT02} = t^s_{DFTO2} + 25$
  - $t^e_{DFTO3} = t^s_{DFTO3} + 60$
- DFT03 time-window $[t^s_{DFT03_W}, t^e_{DFT03_W}]$ where the DFT03 task can occur within. This time-window starts 11 hours after t_{TIG} and last 2 hours: $t^s_{DFT03_W} = t_{TIG} + 660$ and $t^e_{DFT03_W} = t^s_{DFT03_W} + 120$.

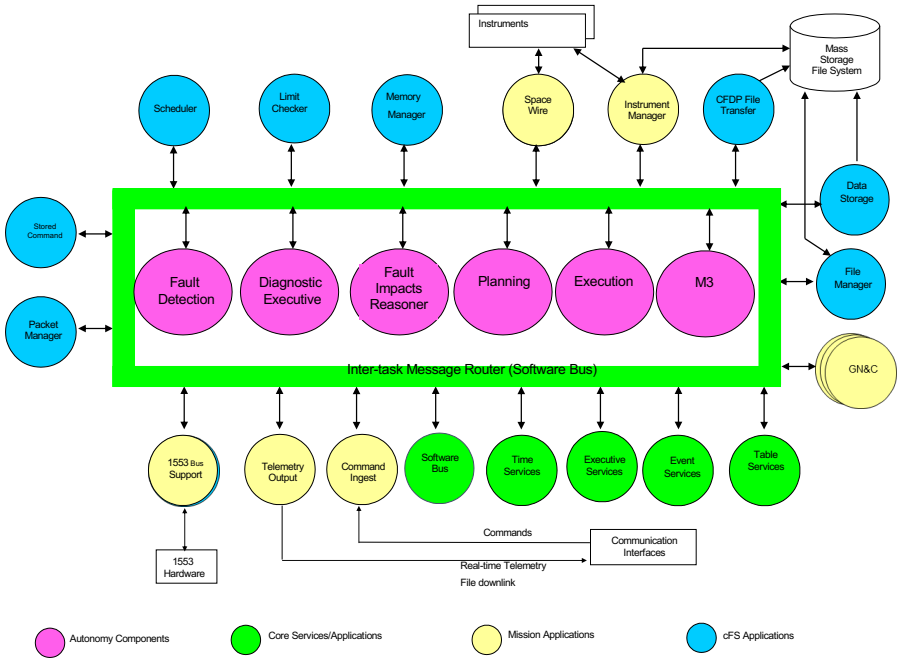

Figure 1: The VSM Architecture

Encoding size optimization: If task X is mandatory (i.e., X definitely appears in the final solution), then we can exclude b_x. Any task that has a fixed duration, then having both t_x^s and t_x^e are redundant and thus we only need one of them. Moreover, if there is a fixed-time constraint between the start/end time of a given task and *TIG* or *ES* then we also do not really need to have t_x^s and/or t_x^e modeled explicitly.

Fixed-time Constraints:

- BACKUP Preheat (PH): $t^s_{PH} = t_{TIG} - 90$ (starting at $t_{TIG} - 1.5hrs$) and $t^e_{PH} = t_{ES}$ (end at t_{ES}).
- 6-hour of Doppler & Ranging (DR) (before burn) : $t^s_D = t_{TIG} - 390$ (starting at $t_{TIG} - 6.5hrs$) and $t^e_D = t^s_D + 360$ (last 6 hrs).
- 30 minutes of Burn Doppler(BD) (after burn): $t^s_{DR} = t_{ES} + 30$ (starting at $t_{ES} + 30$) and $t^e_{DR} = t^s_{DR} + 30$ (last 30 minutes)
- 1-hour of Tank Pressurization (TP): $t^s_{TP} = t_{TIG} - 120$ (starting at $t_{TIG} - 2hrs$) and $t^e_{TP} = t^s_{TP} + 60$ (last 1 hr).
- Data Recording and Downlink (DRD): $t^s_{DRD} = t_{TIG} - 15$ (starting at $t_{TIG} - 15$) and $t^e_{DRD} = t_{ES} + 5$ (end at $t_{ES} + 5$)
- MAIN Htr Deact (PHD): $t^s_{PHD} = t_{TIG} - 20$ (starting at $t_{TIG} - 20$) and $t^e_{PHD} = t_{ES} + 60$ (ending at $t_{ES} + 1hr$)[1].

[1]This constraint is only active when down-mode happens after it already starts (i.e., after $t_{TIG} - 20$). If down-mode happens before heater Deactivation already started at $t_{TIG} - 20$ then it will last until $t_{ES} + 60$.

Precedence Constraints:

- Burn require thermal excursion (within the first thermal excursion windows): $t_{TIG} > t^s_{TE1}$, and $t_{ES} < t^e_{TE1}$.
- DFTOs can't overlap: let x_{12}, x_{13}, and x_{23} being the binary variables represent the ordering constraints between the three payloads. Thus, $x_{12} = 1$ implies DFTO1 is before DFTO2 while $x_{12} = 0$ means DFTO1 is after DFTO2. With M being a large constant, the "non-overlapping" constraint is modeled as follow:
  - $t^s_{DFTO1} - t^e_{DFTO2} > M \times (x_{12} - 1)$
  - $t^s_{DFTO2} - t^e_{DFTO1} > -M \times x_{12}$
  - Similar constraints for x_{13} and x_{23}
- DFTO1 and DFTO3 requires second Thermal Excursions:
  - $t^s_{DFTO1} \geq t^s_{TE2}$ and $t^e_{DFTO1} \leq t^e_{TE2}$
  - $t^s_{DFTO3} \geq t^s_{TE2}$ and $t^e_{DFTO3} \leq t^e_{TE2}$
- DFTO1 requires 70m DSN coverage: $t^s_{DFTO1} \geq t^s_{70m}$ and $t^e_{DFTO1} \leq t^e_{70m}$
- Time window constraint for DFTO3: $t^s_{DFTO3} \geq t^s_{DFTO3_W}$ and $t^e_{DFTO3} \leq t^e_{DFTO3_W}$

Preferences:

- V_1 be the value assigned to satisfying the preference that DFTO2 is in the 70m DSN coverage window. Let y_1 be a binary variable representing if DFTO2 is within the 70m DSN coverage window (i.e., $y_1 = 1$ if DFTO2 is within the coverage window) and M is sufficiently large number.[2]
  - $t^s_{DFTO2} - t^s_{70m} > M \times (y_1 - 1)$
  - $t^e_{70m} - t^e_{DFTO2} > M \times (y_1 - 1)$
- V_2 be the value assigned to satisfying the preference that DFTO1 precedes DFTO2. Similar to the previous preference, we will use a binary variable y_2 to represent the precedence constraint between DFTO1 and DFTO2: $t^s_{DFTO2} - t^e_{DFTO1} > M \times (y_2 - 1)$
- $b_x = y_1$ for $x =$DFTO-1-Downlink, that is, if $y_1 = 1$ we ensure DFTO-1-Downlink is in the plan, otherwise DFTO-1-Downlink need not be in the plan.

Objective Function: $maximize\ V_1 \times y_1 + V_2 \times y_2$

3.2 Fault Management

We built our Fault Management applications using a Commercial Off The Shelf (COTS) diagnostic reasoner, called Testability Engineering and Maintenance System (Real Time) (TEAMS-RT) (Mathur, Deb, and Pattipati 1998). TEAMS-RT uses a declarative model of components, interconnectivity of components and associated information flow between them, fault modes that components can suffer from, logical tests indicating the presence or absence of faults, a and mapping from test results to fault modes.

Figure 2 (top) shows a simple TEAMS model fragment, consisting of three components, and three associated tests. Tests are 'logical tests' as opposed to physical sensors associated with the hardware. The direction of flow of capability is indicated by the arrows. In this example, we see two tests pass, and one test fails. Figure 2 (middle) shows the diagnostic matrix (D-Matrix) for this model. Each fault mode for each component occupies a row of the D-Matrix, and tests occupy columns. A 1 in a matrix cell indexed by (fault-mode, test) indicates that the test provides information about the fault mode. TEAMS uses a default reasoning assumption, which is that a *passed* test exonerates any fault mode informed by the results of the test, and that any fault mode that is not exonerated is a suspect. If only one suspect is left after all tests are evaluated, the fault mode is uniquely identified as responsible for the failed tests. Figure 2 (middle) also shows the use of the default assumption in reasoning about three hypothetical test results shown in the top figure.

Fault management is divided into three functions; Fault Detection, Diagnositc Executive, and Fault Impacts Reasoning, all of which use TEAMS models.

[2]The two constraints ensures that $y_1 = 1$ enforces that DFTO2 should be within $[t^s_{70m}, t^e_{70m}]$ while the reverse is not true: $y_1 = 0$ doesn't enforce NOT-containment. However, our objective function will ensure that it's beneficial to have $y_1 = 1$ so that there is no scenario where $y_1 = 0$ while DFTO2 is contained within the 70m DSN coverage.

Fault Detection (FD) processes system data to determine if any defined off-nominal conditions exist. Each off-nominal condition is associated with a test, which outputs pass, fail or unknown.

Diagnostic Executive (DE) receives test results from Fault Detection and packages them to send to TEAMS-RT. TEAMS-RT correlates test points with failure modes, as described previously; a failed test can implicate one or more failure modes, while a passing test exonerates failures. With sufficient data, a single failure mode can be identified that is responsible for all failed tests, and an unambiguous failure mode is identified. Otherwise, the reasoner determines the smallest set of possible failure modes that could be responsible for failed tests and presents an ambiguity group of possible failures. For the purposes of this paper, we will be working with a model that ensures faults can be uniquely isolated, and we assume no unknown test results. There are four faults DE can detect, as shown in Figure 2 (bottom).

Fault Impacts Reasoner (FIR). FIR receives failure information from DE and determines the resultant impacts of confirmed failures. Impacts include the loss of capability, such as the components that have lost electrical power due to a fault in the electrical system. The loss of redundancy due to a fault is also determined. Most critical functions in spacecraft depend on redundancy to assure the availability of the capability in spite of failures. Of particular concern is any capability that could be lost by a single additional failure, or has become zero-fault tolerant. FIR identifies these changes in redundancy to aid with identifying capabilities at increased risk, helping operators to determine next worst failures and take mitigation steps to reduce the risks of additional failures, if possible. FIR, coupled with DE, draws a crisp distinction between failed, that is, broken components, from components and capabilities lost or affected by the failure of a component. Figure 2 (bottom) shows part of the propulsion system TEAMS fault model, with two independent control flows ensuring the main engine functions. The loss of a single 'string' leads to loss of redundancy, indicated by a message LOR(MAIN), while the loss of both strings leads to loss of capability, LOC(MAIN). The details of how the TEAMS model and faults are evaluated to determine loss of capability or redundancy are described in (Aaseng et al. 2015); for our purposes, it is enough to know that M3 receives this information from FIR.

3.3 Executive

The plan is executed by the Plan Execution and Interchange Language (PLEXIL), developed as a collaborative effort by NASA and Carnegie Mellon University, and subsequently released as open-source software (Verma et al. 2006). PLEXIL is a language for representing flexible robust plans intended to be executed in an uncertain and changing environment. PLEXIL provides well defined execution semantics with contingency handling which can be formally validated and produces deterministic results given the same sequence of environmental inputs. PLEXIL's Execution Engine (executive) executes stored plans written in the PLEXIL

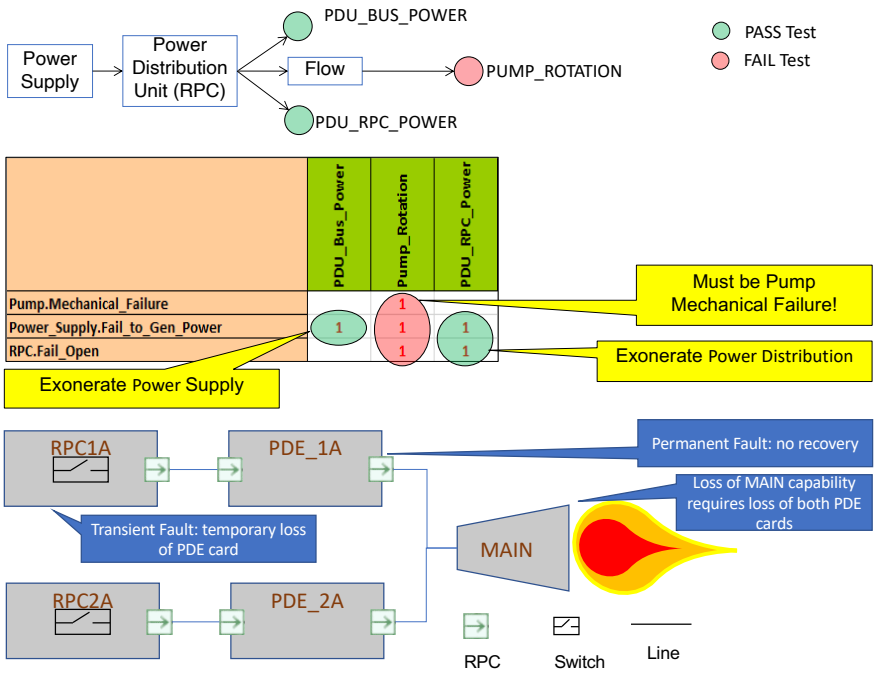

Figure 2: TEAMS models used for fault detection. Top: simple TEAMS model and tests. Middle: TEAMS Diagnostic Matrix and evaluation of a single set of test results. Bottom: simple TEAMS model of the propulsion system showing multiple paths of capability to the main engine.

language. PLEXIL is responsible for receiving tasks to execute from M3, and returning task execution status to M3.

For the purposes of the VSM, PLEXIL's job is to execute the fault recovery plan tasks. PLEXIL always 'succeeds' in these tasks; the DE messages received after recovery plans complete indicate whether the fault persists or not after recovery is attempted. Recall multiple recovery attempts are permitted; we describe how this is handled in the next section.

3.4 Mission Management and Metareasoning

The responsibility of the Mission Management and Metareasoning (M3) module is to integrate fault management, planning and plan execution. M3 is responsible for tracking the execution of each task in the plan, by sending the tasks to PLEXIL, and listening to the return status. M3 is also responsible for listening for DE and FIR messages, which indicate the presence of faults, and the need to replan. Finally, M3 is responsible for setting up the planning problem to solve.

M3's job is complicated by the combinations of faults that can occur, determining whether those faults are transient or permanent, and tracking the history of previously executed events that can influence replanning. Each fault-mode is mapped to a unique fault recovery task. However, M3 needs to keep track of whether the recovery plan resolves the fault or not by monitoring DE messages after execution completes. M3 also needs to keep track of how many recovery attempts have been tried; the rationale for this decision is provided at the end of this section.

Replanning after loss of capability requires solving the MILP after setting up initial state and constraints, which in turn depends on both parsing the FIR messages, and preserving execution history of previous events. An interesting issue that arises when transient faults occur: it is possible for a FIR message to indicate loss of redundancy if both control paths are impacted by faults, even if one of (or both!) are transient faults. Since neither DE nor FIR are able to detect transient faults, premature inspection of FIR messages could lead VSM to believe replanning is needed. M3 needs to flag faults as permanent and wait for fault recovery impacts to settle prior to handling FIR messages.

We now describe each of these responsibilities of M3 in more detail.

DE message handling and detection of transient fault: when any fault is detected, we first assume that it is *transient* and try to fix it. This is done by invoking the planner to ask for a fault recovery plan P_f to be sent to PLEXIL. As described previously, M3 can rely on DE messages to indicate the presence of a unique fault. Fault recovery planning is 'trivial' (e.g. look up RPC open-close plan in

response to DE message with RPC fault; this simple plan has no conflicts with any other tasks in plan, and can execute immediately). M3 then waits for a response from PLEXIL indicating this plan executed successfully. After PLEXIL executes P_f, M3 listens to DE's messages for certain period of time T to ensure that the fix's effect is fully realized. If the DE's messages indicate that the fault is not observed after the fixing action, then we know that it's not permanent and can stop the process. As noted above, M3 keeps track of the number of recovery attempts, and when a fault can't be resolved within the number of tries specified, the fault is considered permanent.

FIR message handling and permanent faults: in this stage, M3 goes through the following steps: First, M3 listens to FIR's messages for certain period of time S to ensure that the FIR message content stabilizes. This is necessary because the prior attempts to fix the fault may require time to settle out. Next, M3 accumulates system information from different components such as telemetry and FIR. In our scenarios, M3 must map consequences in FIR to constraints on a new planning problem. The simple case is that a FIR consequence does not lead to replanning at all, e.g. if there is only loss of redundancy to the main engine, there is no need to replan. In general, though, we would map every loss of capability or loss of redundancy option to a decision to replan or not. This step also includes determining whether or not `MAIN Htr Deact` has started by examining the history of executed tasks. Next, it sends the gathered system information to the planner in the form of a new initial state and constraints, to build the burn-replanning problem and wait for the planner to return a plan P.

Referring to Sections 3.1 and 3.2, M3 maps a FIR impact `LOC(MAIN)` to update the temporal constraint on t_{ES}, t_{TIG} from $t_{ES} = t_{TIG} + 4$ to $t_{ES} = t_{TIG} + 60$. M3 can also internally track variables $e_{x,s}$ and $e_{x,e}$ indicating that task x has started or finished, and conditionally set $b_x = 0$ if `MAIN Htr Deact` has not started.

To keep M3 design interfaces consistent, M3 invokes the planner and pass the plan to PLEXIL for execution. We could have designed PLEXIL recovery plans to listen to DE messages with a conditional execution loop to try 3 times, but to centralize all fault management message handling inside M3, we chose to keep PLEXIL plans, and thus its interface, simple.

The overall logic of M3 is shown in Figure 3.

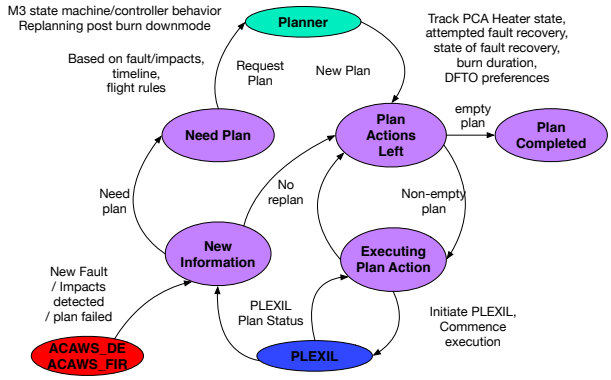

Figure 3: M3 Logic: Burn Replanning

4 How It Works

We evaluated the VSM on all of the scenarios described in Section 1. The VSM correctly detected transient vs permanent faults, and generated the correct plans in all cases.

Figure 4 shows an example of how the VSM replans when the burn down-mode occurs early, i.e. before `TIG-20`. At the top of this figure, we show the original burn plan, with the `TIG` time fixed, and the short 4 minute burn using the `MAIN` engine. The top of the figure also shows the fixed 70m DSN window, and the fixed window during which `DFTO-3` can take place. When the burn down-mode is detected early, which requires both permanent propulsion system faults to have occurred and been detected, the planner replans the burn, dropping the `MAIN Htr Deact` task, and also shifting the first thermal excursion window. This forces a shift of the later thermal excursion window, which forces a choice of which preference to satisfy. We also assume for this figure that preserving the order `DFTO-1, DFTO-2` is more important, than `DFTO-1` DSN coverage. Thus, the final plan preserves this ordering of the DFTOs, moving `DFTO-1` out of the DSN window; finally, we see the `DFTO-1-Downlink` task dropped from the final plan. Space precludes showing the other plans, but the VSM successfully replanned in all 7 scenarios.

5 Previous Work

Previous work in the area of intelligent systems often includes combinations of planning, execution and fault management, but usually not all three. The Autonomous Sciencecraft Experiment (ASE) from JPL (Tran et al. 2004), (Sherwood et al. 1998) demonstrated automated planning for a low-earth orbiting remote sensing satellite. This planner focused on optimizing science, and did not incorporate fault management. The T-Rex autonomous agent (McGann et al. 2008), used in undersea robotics, incorporated numerous special-purpose planners for path planning, task planning, tightly integrated with an execution system; this fielded agent also did not incorporate fault management.

One exception to this rule is the seminal paper on the Remote Agent (Jónsson et al. 1999) describes the first in-space autonomous agent employing AI techniques. The Remote Agent included a planner, executive, fault management, and Mission Manager. A second partial exception is MEXEC (Troesch et al. 2020), a planner/execution system evaluated for future deep space missions on the ASTERIA Cubesat mission, was integrated with fault management, but only to the extent that execution would terminate and clear the current plan in the event of a fault.

Our previous work in testing a VSM for human spaceflight (Aaseng et al. 2018) also demonstrated the integration of planning, execution and fault management, integrated via CFS. In one scenario, the planner performed load-shed in the presence of a power systems fault that limited energy availability in an eclipse, with some modest constraints coupling

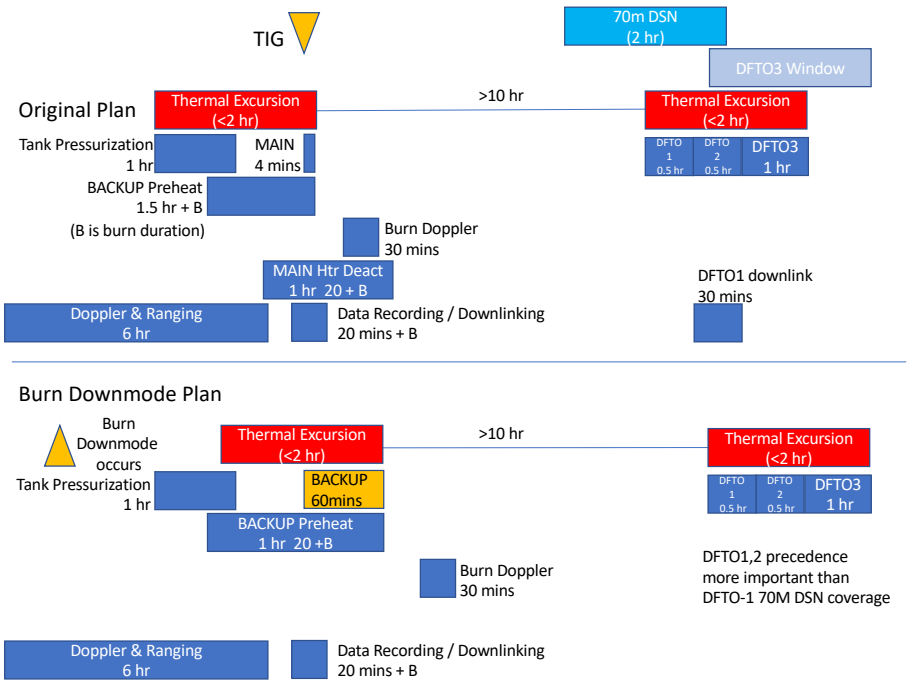

Figure 4: VSM Replanning when down-mode detected early

pairs of loads. In a second scenario, the planner reasoned about faults in one part of the life support system that led to shutting down other parts of the life support system. In our previous work, PLEXIL was responsible for integrating the planner and DE, but did not use information from FIR.

6 Conclusion & Future Work

We describe a prototype Vehicle System Manager (VSM) for NASA's Gateway, a human-capable spacecraft that will also be capable of autonomous operations. This VSM consists of an MILP-based replanner, TEAMS-based fault management, and PLEXIL as an executive, integrated via a single over-arching M3. We show the VSM is capable of detecting and replanning in the presence of different failure scenarios centered on a propulsive burn down-mode, which alters the timing of key activities. These scenarios feature transient and permanent failures, detection of loss of capability, and preferences and complex temporal constraints on plans.

We have demonstrated replanning of an existing plan featuring temporal constraints and preferences in the presence of faults. Obvious extensions to this include larger planning problems with more difficult features such as resources, goals and de-novo causal reasoning.

We restricted out scenario to cases where DE identifies unique root causes of faults; each fault maps to a recovery procedure. In general, fault ambiguity can be addressed by mapping sets of faults to recovery procedures, but the combinatorics become formidable. Alternatives include mapping loss of capability or redundancy to recovery procedures, which have the advantage of focusing of what was lost as opposed to what the possible causes were, but capturing the knowledge is still a challenge.

We note that FIR can identify loss of redundancy as well as loss of capability. In more sophisticated scenarios, M3 could implement a burn-down-mode when the `MAIN` has lost redundancy, and is zero fault tolerant, as opposed to lost.

We don't have a 'declarative' representation of M3 to represent the key design features we describe, such as the DE-plan fault recovery mapping, FIR-planner invocation, number of attempts prior to declaring a fault permanent, and so on. Codifying the configuration of M3 in a fully declarative way would formalize its function and enable more sophisticated behaviors to be specified in a more model-based way.

We would like to thank our Johnson Space Center friends in the Flight Operations Directorate, in particular David Lantz, for all their help in scoping the scenario in order to mature the VSM, and the Gateway program and the NASA Advanced Exploration System Program for all of their support for this work.

References

Aaseng, G.; Barszcz, E.; Valdez, H.; and Moses, H. 2015. Scaling Up Model-Based Diagnostic and Fault Effects Reasoning for Spacecraft. In *Proceedings of the AIAA Conference on Space Operations*.

Aaseng, G.; Frank, J.; Iatauro, M.; Knight, C.; Levinson,

R.; Ossenfort, J.; Scott, M.; Sweet, A.; Csank, J.; Soeder, J.; Loveless, A.; Carrejo, D.; Ngo, T.; and Greenwood., Z. 2018. Development and Testing of a Vehicle Management System for Autonomous Spacecraft Habitat Operations. In *Proceedings of the AIAA Conference on Space Operations*.

Badger, J.; Strawser, P.; and Claunch, C. 2019. A Distributed Hierarchical Framework for Autonomous Spacecraft Control. In *Proceedings of the IEEE Aerospace Conference*.

Crusan, J. C.; Smith, R. M.; Craig, D. A.; Caram, J. M.; Guidi, J.; Gates, M.; Krezel, J. M.; and Herrmann, N. 2018. Deep Space Gateway Concept: Extending Human Presence into Cislunar Space. In *Proceedings of the IEEE Aerospace Conference*.

Jónsson, A. K.; Morris, P. H.; Muscettola, N.; and Rajan, K. 1999. Next Generation Remote Agent Planner. In *Proceedings of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space*.

Mathur, A.; Deb, S.; and Pattipati, K. 1998. Modeling and Real-Time Diagnostics in TEAMS-RT. In *Proceedings of the American Control Conference*.

McComas, D.; Wilmot, J.; and Cudmore, A. 2016. The Core Flight System (cFS) Community: Providing Low Cost Solutions for Small Spacecraft. In *Proceedings of the* 30^{th} *AIAA /USU Conference on Small Satellites*.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwan, R. 2008. A Deliberative Architecture for AUV Control. In *Proceedings of the International Conference on Robotics and Automation*.

NASA. 2018. Spaceflight Demonstration of a Power and Propulsion Element (PPE). https://www.fbo.gov/spg/NASA/GRC/OPDC20220/80GRC018R0005/listing.html. See Amendment 6: Unique Requirements.

Sherwood, R.; Govindjee, A.; Yan, D.; Rabideau, G.; Chien, S.; and and, A. F. 1998. Using ASPEN to Automate EO-1 Activity Planning. In *Proceedings of the IEEE Aerospace Conference*.

Tran, D.; Chien, S.; Sherwood, R.; Castaño, R.; Cichy, B.; Davies, A.; and Rabbideau, G. 2004. The Autonomous Sciencecraft Experiment Onboard the EO-1 Spacecraft. In *Proceedings of the* 19^{th} *National Conference on Artificial Intelligence*, 1040 – 1045.

Troesch, M.; Mirza, F.; Hughes, K.; Rothstein-Dowden, A.; Bocchino, R.; Donner, A.; Feather, M.; Smith, B.; Fesq, L.; Barker, B.; and Campuzano, B. 2020. MEXEC: An Onboard Integrated Planning and Execution Approach for Spacecraft Commanding. In *Proceedings of the Workshop on Integrated Execution and Goal Reasoning (InTEX/GR)*.

Verma, V.; Jónsson, A.; Pasareanu, C.; and Iatauro, M. 2006. Universal Executive and PLEXIL: Engine and Language for Robust Spacecraft Control and Operations. In *Proceedings of the AIAA Space Conference*.