Model-Based On-Board Decision Making for Autonomous Aircraft

Johann Schumann[1], Nagabhushan Mahadevan[2], Michael Lowry[3], and Gabor Karsai[4]

[1] *SGT, Inc., NASA Ames Research Center johann.m.schumann@nasa.gov,*

[2,4] *ISIS, Vanderbilt University Nag.Mahadevan@vanderbilt.edu, Gabor.Karsai@vanderbilt.edu,*

[3] *NASA Ames Research Center michael.r.lowry@nasa.gov,*

ABSTRACT

Powerful, small and lightweight sensors in combination with advanced failure detection, diagnosis, and prognostics techniques provide up-to-date data on the health status of an Unmanned Aerial System (UAS) or autonomously piloted vehicle. This information must be used for automatic planning and execution of contingency actions to keep the UAS safe in adverse conditions. We present DM (Decision Maker), a software component which uses model-based reasoning and backtracking search to iteratively construct contingency plans that are safe for the UAS to execute and pose minimal interruption to the mission goals. The DM is a discrete decision making system has been developed within the NASA Autonomous Operating System (AOS) project and fills the gap between Prognostics and Health Management and autonomous flight operations. In this paper, we describe DM and its reasoning algorithm and present the supporting modeling framework for the construction of system and fault models. Flights with a DJI S1000+ octocopter with fault injection will be used as our case study.

1. INTRODUCTION

Modern system health management and diagnostics components of the flight software can provide detailed up-to-date information about the operational status and failure modes of the aircraft during the flight. Advanced, model-based prognostics systems can deliver—during the flight—accurate estimates on the remaining capacity (e.g., battery), estimated time before a component (e.g., a bearing) fails (remaining useful life), or the probability, with which the aircraft can successfully and safely abort a take-off given its current acceleration (Schumann, Zollitsch, Mumm, & Holzapfel, 2018).

For a manned aircraft this information can be suitably presented to the pilot, who then makes a decision on the appropriate action to take. During flight, knowledge, training, and a large amount of experience enables the pilot to quickly make decisions to alleviate the problem and find means to continue and complete the mission or safely end the flight. But what happens if there is no pilot on board or the aircraft cannot be controlled remotely? Safe autonomous operation of an Unmanned Aerial System (UAS) or an aircraft that is autonomously piloted requires that the UAS can swiftly react to unexpected failures without human intervention and execute a suitable contingency plan to safely conclude the mission. This capability has to go way beyond simple, often hard-coded procedures like “land immediately” or “return to home base”.

In this paper, we present DM (Decision Maker), a discrete model-based on-board contingency planning system for UASs that has been implemented on top of NASA’s Autonomous Operating System (AOS). During the flight, DM is provided with system health and failure information about the on-board systems, relevant prognostics estimates, as well as environmental awareness information, e.g., about other aircraft or air traffic control information. This information is provided in real-time by the on-board prognostics, diagnostic, and monitoring systems. DM then performs the following two tasks: (1) select and execute a suitable active diagnosis procedure if it is necessary to disambiguate failure modes, and (2) construct a contingency plan, which retains safety and impacts the mission as little as possible. This contingency plan is constructed as a sequence of discrete, customizable actions and must reflect all operational restrictions and limitations that are imposed by the detected failures. For example, a weak battery would prohibit strong climbs, a stuck aileron would limit the safe roll rate, or a broken LIDAR altimeter might make it impossible to safely fly at a low altitude over rough terrain. A typical contingency plan might entail a modification of the flight plan giving priority to safety. For

example, instead of climbing over a mountain to reach the intended destination airport, the flight plan could be altered to fly around a high mountain and reach an alternate destination safely.

We have developed a customized domain-specific modeling language using Web-GME (Maroti et al., 2014) to capture failure modes, fault propagation, failure impact, active diagnosis procedures, and operational failure impacts. DM's contingency planning algorithm uses a combination of backtracking search, constraint solving, and simplified prognostics algorithms in order to come up with suitable and safe candidates for contingency plans.

The main contributions of this paper are:

- realization of a model-based on-board autonomous contingency planner,
- model-based activation and triggering of active diagnostic procedures,
- use of advanced, schema-based discrete search to come up with suitable contingency plan, taking into account the constraints imposed on state-predictions by the current flight plan, air traffic control information, system and failure state, capacity reductions and prognostic estimates, and
- implementation as part of the AOS system and test-flight at NASA on DJI S1000+ octocopter.

The rest of the paper is structured as follows: in Section 2 we briefly describe the underlying AOS system and the NASA cFS/cFE architecture as well the on-board monitoring, diagnosis, and prognostics systems. Section 3 provides a high-level architecture of our model-based contingency planner and describes principles of operations. In Section 4 we focus on our modeling language and the various models that are used by the contingency planner. Section 5 describes details of the algorithms underlying the Decision Maker (DM). A case study with a test flight and injected failures is presented in Section 6. In Section 7, we highlight related work before we discuss future directions and conclude (Section 8).

2. BACKGROUND

Our model-based contingency system has been implemented as part of the NASA Autonomy Operating System (AOS). In this section, we provide a brief background of AOS, its underlying software platform cFS, and about the on-board diagnosis and monitoring systems.

2.1. NASA Core Flight System

The NASA core Flight System[1] (cFS) is an Open-Source platform and framework that has been developed at the NASA Goddard Space Flight Center. Its component-based design, layered software and dynamic runtime environment has been used in a number of successful space missions and has been certified for man-rated space applications. The cFS architecture simplifies the flight software development process by providing the underlying infrastructure and hosting a runtime environment for mission-specific applications.

2.2. The Autonomy Operating System (AOS)

The Autonomy Operating system (AOS) is a software system that enables core capabilities for the autonomous operations for an unmanned aircraft (Lowry et al., 2018). It is based on the NASA cFS system and provides a higher-level layer of infrastructure and applications for the execution of flight plans, natural-language communication with Air Traffic Control (Lowry, Pressburger, Dahl, & Dalal, 2019), Diagnostics, Prognostics, and contingency planning (Schumann et al., 2019).

The AOS architecture shown in Figure 1 illustrates the structure of the system. The underlying cFS system provides a "software bus," a publish-subscribe architecture, which is used by numerous applications to communicate with each other. Apps can be activated on regular schedule or can be event driven. Numerous apps are provided by the cFS software (shown as circles and "lollipops" in Figure 1) and help to facilitate the design of new flight software.

AOS is communicating with a low level flight control software to obtain sensor and aircraft status information and to issue low level commands. In our case, we use a slightly modified version of the Open-source ArduCopter software,[2] running on a PixHawk hardware,[3] which directly interfaces with sensors and controls the motors of the aircraft.

The AOS applications and engines (shown as yellow boxes) provide specific capabilities and the "knowledge bus" infrastructure that enable autonomous UAS operations: automated reasoning, based upon an underlying Prolog or Z3 reasoner, is used for navigation, contingency planning, and interaction with Air Traffic Control (ATC). Spoken ATC commands are processed by the Natural Language Processing (NLP) unit (Lowry et al., 2019). PLEXIL is an event-driven planner (Verma, Jonsson, Pasareanu, & Iatauro, 2006) that has been customized to execute flight plans for nominal/off-nominal operations, and procedures, which require Air Traffic Control (ATC) interaction.

The applications for diagnosis, prognostics, and runtime assurance are based on the Diagnostic Reasoner (DR) and the R2U2 (Realizable, Responsive, Unobtrusive Unit) components, which will be described in some more detail in the following subsections.

Our model-based contingency planning framework is cen-

[1] https://cfs.gsfc.nasa.gov/

[2] http://ardupilot.org/copter/

[3] pixhawk.org

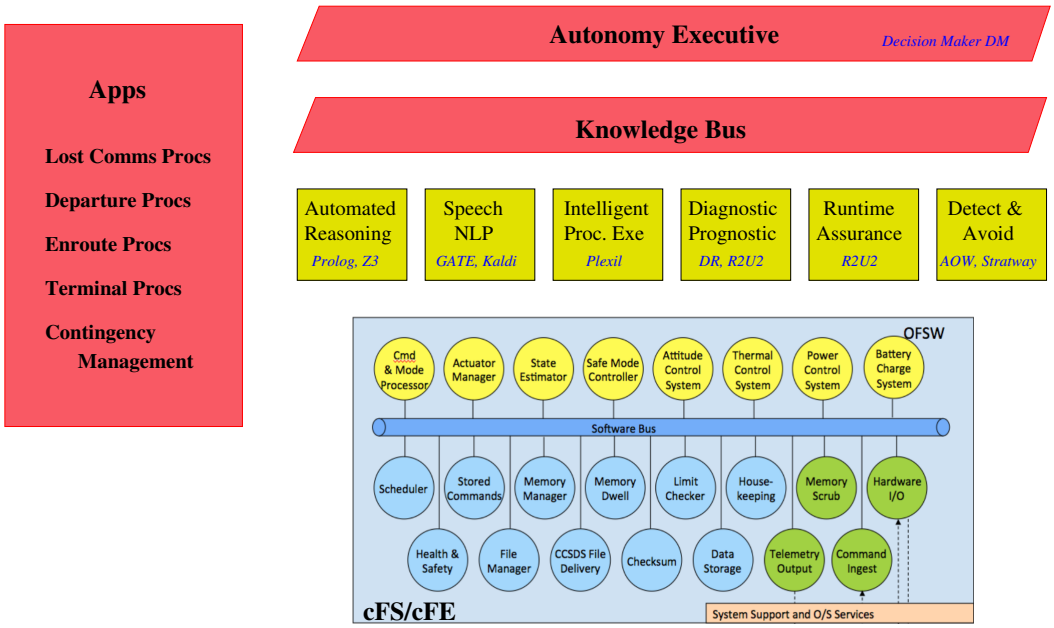


Figure 1. High-level architecture of AOS based on cFS/cFE. cFS architecture diagram from (Gundy-Burlet, 2013)

tered around the the model-based Decision Maker "DM", which exchanges information using the Knowledge Bus with the other AOS components.

2.3. Diagnostic Reasoner (DR)

The Diagnostic Reasoner (DR) is a cFS component, which monitors and diagnoses the vehicle on which AOS is running (Schumann et al., 2019). Based on sensor information from the vehicle, DR performs fault detection, and if an anomaly is found, it will perform fault isolation to identify the current failure mode(s). If it is not possible to isolate the fault to a single failure mode, the diagnosis result contains an ambiguity group of several potential failure modes. Analog sensor data received from AOS are preprocessed and checked against given value ranges and thresholds by the Limit Checker (LC), a built-in component of the cFS system. For diagnosis, DR uses a dependency matrix (D-matrix) approach (Luo, Tu, Pattipati, Qiao, & Chigusa, 2005) to determine the state of a system component as "GOOD", "BAD", "SUSPECT", or "UNKNOWN": based upon the results of tests, e.g., $U_{batt} > 16V$, the algorithm consults the D-matrix to determine, which components are might be affected by these test results. Tests results are defines as discrete values "PASS", "FAIL", or "UNKNOWN", if no data are available. Even for large D-matrices relating hundreds of tests to thousands of failure modes, the algorithm (described in (Schumann et al., 2019)) is efficient enough to enable real-time diagnosis. The binary D-matrix, which relates the diagnostic tests for a component to the failure modes of that component is generated automatically from our system and failure models as described in Section 4.

2.4. Online Monitoring System (R2U2)

R2U2 (Realizable, Responsive, Unobtrusive Unit) is a framework and tool for the continuous monitoring of safety-critical and embedded cyber-physical systems (Reinbacher, Rozier, & Schumann, 2014; Rozier & Schumann, 2017). R2U2 combines past-time and future-time Metric Temporal Logic, probabilistic reasoning with Bayesian networks, and model-based prognostics. Like the other components of AOS, R2U2 is implemented as a cFS application and activated at a regular rate of 1Hz. The preprocessing unit of R2U2 "PRE" reads sensor and status values from the cFS message bus and, using a set of customizable filters, operators, and discretizers, produces Boolean values, which are then processed by the R2U2 temporal engine (Geist, Rozier, & Schumann, 2014; Schumann, Roychoudhury, & Kulkarni, 2015). In a similar way, prognostics results "PRGN" are calculated and fed into the R2U2 engine (Kulkarni, Roychoudhury, & Schumann, 2018; Schumann et al., 2015).

The R2U2 engine is a software implementation of a processor, which provides efficient monitors for past-time logic as well as synchronous and asynchronous observers for future-time Metric Temporal Logic. The underlying algorithms are described in (Reinbacher et al., 2014; Rozier & Schumann, 2017). Safety and liveness properties are specified as temporal logic formulas and their results are used by DM for planning and decision.

For example, after a change of the target heading of the UAS, we expect that the UAS heading becomes aligned within 5 seconds. Short glitches should be ignored. This property only

needs to hold while the UAS is in the "auto mode" and thus following a flight-plan:

$$mode\_auto \rightarrow \Diamond_{[2s]} hdg\_achieved \vee \Diamond_{[3s]} hdg\_changed$$

The implication ensures that the property is only checked when the vehicle is in auto mode. Then, a mis-alignment of the heading is ignored if it lasts less than 2 seconds, unless we encountered a change in direction $hdg\_changed$ within the last 3 seconds, requiring that the new heading has been achieved no later than 5 seconds after the change.

The properties used in this paper have been specified in past-time logic. More properties and examples are discussed in (Rozier & Schumann, 2017). R2U2 can also perform real-time Bayesian reasoning (Geist et al., 2014), using the results of temporal formals as inputs to support probabilistic root cause analysis by, for example, estimating the likelihood of a specific sensor failure. That information is also be passed to the DM.

3. MODEL-BASED ARCHITECTURE FOR ACTIVE DIAGNOSIS CONTINGENCY PLANNING

AOS is separating the capability of health and contingency management into two modules: diagnosis/prognostics and decision-making. In the case of an adverse event or failure, diagnosis is performed first, to determine what failure has occurred. Decision-making is done next, to improve the resolution of the fault diagnosis and to determine the impact of the failure followed by what actions might need to be done to recover from the failure and fly the aircraft to safety. The selected contingency plan is then executed by the on-board PLEXIL engine.

Figure 2 shows the flow of information. Sensor and status data S from the UAS are transmitted by the ArduCopter flight software into the AOS system. These data are used to perform diagnosis and to monitor the UAS system (subsystems grouped by a dashed line). The model-based diagnosis apps DR and the LC (Section 2.3) perform fault detection and isolation; R2U2 dynamically monitors sensors and software and performs prognostics (Section 2.4). The current health status of the UAS, which is updated at 0.5Hz is then handed over to the DM (Decision Maker, Section 5 below) component. Based upon the system health status and the current flight plan, the DM performs logic-based search to find (a) active diagnostic procedures to improve the diagnostic resolution (if necessary) and (b) flight plans, which can be safely executed under the current circumstances. If necessary, such contingency plans might contain emergency actions like, for example, cutting short the flight, diversion to a nearby airport for emergency landing, or an immediate ditch by activating an on-board parachute. The DM can additionally obtain information provided by the on-board data base (DB) and messages sent from the ground station. The generated active diagnosis or contingency plan is sent to the plan execution module PLEXIL, which is in charge of commanding or navigating the UAS and interacting with Air Traffic Control. This component emits the sequence of commands U and waypoints that the UAS will follow. All components of the health management and contingency planning system are model-based.

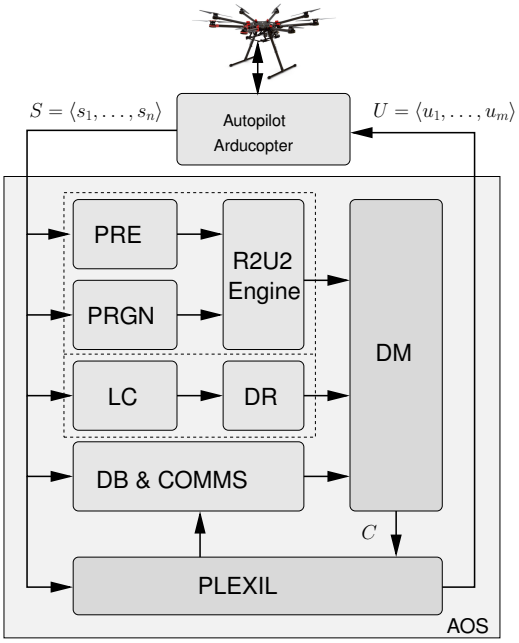


Figure 2. High-level architecture of our model-based health management and contingency planning system. Diagnostics (DR) and monitoring (R2U2) components with their data preprocessing (PRE and LC) and prognostics (PRGN) are marked by a dashed line.

4. SYSTEM MODELING AND D-MATRIX GENERATION

The models corresponding to the AOS components for the UAS under study are created in a domain specific modeling language (DSML) – the Fault Modeling Language (FML).

The DSML and the associated model instances were created in Web-GME (Maroti et al., 2014), an Open-source meta-programmable platform. Web-GME provides a graphical platform to define the rules of the DSML in a meta-model. Thereafter, it allows users to create model instances that correspond to the defined meta-model. It has extensive API support for writing custom plugins to generate artifacts from the models. It provides a web-based collaborative platform where multiple users can view and edit the models simultaneously. It has a built in version tracking system that allows the users to create tags, branch, fork and merge during the model development process.

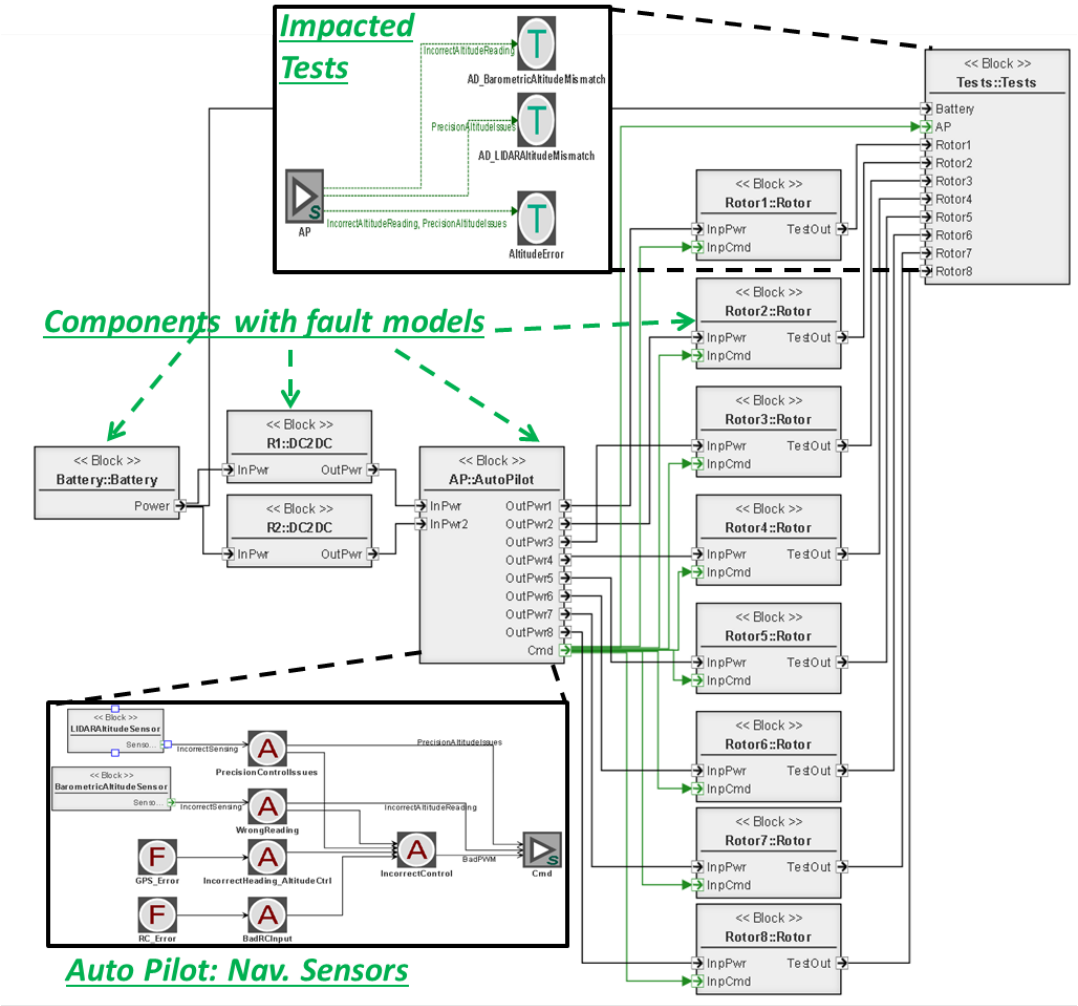


Figure 3. S1000: System model with fault propagation

Our DSML, the Fault Modeling Language allows the users to describe the

(a) system architectures,

(b) functional decomposition,

(c) fault propagation model,

(d) fault impact,

(e) diagnosis refinement procedures, and

(f) fault masking and/ or recovery plans.

4.1. System Architecture and Fault Propagation Model

FML uses the SysML (Friedenthal, Moore, & Steiner, 2008) style Block Diagram models to capture the components (blocks), their interfaces (ports), and their interaction (wiring between the ports). The SysML internal block diagram models are extended to capture the fault model within the component and its propagation across the system. The fault model captures the fault sources (Faults), deviations from nominal or expected behavior (Anomalies), observable anomalies (Tests), and functionality degradation (Effects). The edges represent cause-effect relationship and the fault propagation. Labeled edges from and to the ports capture fault propagation across component boundaries. The fault model uses the concepts defined in Timed Failure Propagation Graphs (TFPG) (Abdelwahed, Karsai, & Biswas, 2005; Abdelwahed, Karsai, Mahadevan, & Ofsthun, 2009).

Figure 3 shows the system model for the DJI S1000+ octocopter. The blocks represent components and/or subsystems of the S1000+. The top level shows the wiring diagram for signal and power flow between the components. Each component contains the faults associated with that component. The fault propagation model for each component accounts for the anomalies and the functional degradation that result from the faults originating in the component as well as failure effects that propagate from other components. The failure ef-

fect propagation from other components is represented by labeled edges (failures) connected to the input and output ports. Figure 3 shows a portion of the fault model in the auto pilot subsystem associated with the navigational sensors—in particular, the barometric altitude sensor and the LIDAR-based precision altitude measurement sensor. The Test block in this figure is not associated with a real component in the system, but is a logical model that groups all the alarms/tests associated with the system. The fault propagation model captures the triggering criteria for these alarms/tests in terms of the failure effects that flow into these tests. Figure 3 shows the possible tests that could trigger if there is a problem associated with one of the altitude measurement sensors.

4.2. Functional Decomposition Model

Systems are designed to satisfy specific functional requirements. FML allows the user to capture these functions and order them hierarchically to create a functional decomposition model. The models include high-level functions which are related to lower level functions through AND and OR nodes. The AND nodes imply that all the lower level functions are required to provide the higher level function. The OR node corresponds to the case where the higher level function can be satisfied by any one of the lower level functions. The lowest level functions in this hierarchy are referred to as primitive functions and these are related to one or more components or subsystems in the system architecture model that implement the primitive function.

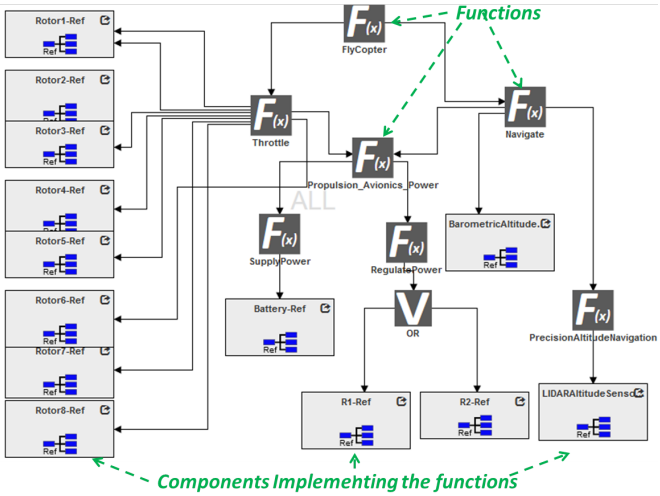


Figure 4. S1000: Functional Decomposition Model

Figure 4 captures the functional decomposition model for our UAS. The components are associated with functions they implement. When the AND/OR nodes are missing, the default relationship corresponds to an AND node. In this case, there is only one OR node (marked with a "V") to indicate that the power regulation function can be handled by either one of the regulators.

4.3. Fault Impact Models

The impact of the fault on the system functions are captured in multiple FML models. The fault propagation model in FML allows the user to capture the degradation or loss of a function in terms of an Effect node (Figure 5A). The Effect node, represented by an "E" in the fault model, cross-references a specific function in the functional decomposition model. The idea is to indicate that specific set of fault conditions can lead to the function loss or degradation effect. In this case, the model needs to be compiled to understand the different fault scenarios that can produce this detrimental effect.

A more explicit model is the Fault Impact model that relates Fault Ambiguity Groups (sets of one or more faults) to one or more functions that are impacted. The model also capture any changes to the safe/permissible operation in terms of modifications to system variable ranges and system mode. The Fault Impact Model in Figure 5A captures the impact of Error in Precision Altitude sensing on the navigation function and the associated restriction on the minimum altitude (buffer) required to be safe in the presence of such a fault. While this example captures a single fault, it is quite normal for multiple faults in the ambiguity set to have similar impacts on the functions and the operational variables (Schumann et al., 2019).

4.4. Diagnostic Refinement Model

The diagnostic refinement model captures the fault ambiguity set and the corresponding active diagnosis procedures (and the impacted tests) that could help in isolating the fault source. The fault ambiguity set references the faults in the system model that could not be disambiguated based on the operational tests. Each active diagnosis procedure cross-references the tests in the system model that would be activated by the procedure. The tests marked with an "AD" prefix in the Test sub-block in Figure 3 are examples of tests whose status could be assessed when a specific active diagnosis procedure is executed. When one of the associated "AD" tests fails, it helps to isolate the fault candidate or at least prune the fault ambiguity set. Additionally, if a test passes, it indicates that the associated faults are not present and hence should be dropped from the ambiguity set.

Figure 5B lays out prescriptive active diagnosis procedures for mismatch in altitude measurement among the associated sensors. The "Altitude Check" active diagnosis procedure indicates those tests whose status could be evaluated to prune the fault set. While the model does not include the procedural code for executing the active diagnosis procedure, it captures the operational constraints under which the procedures

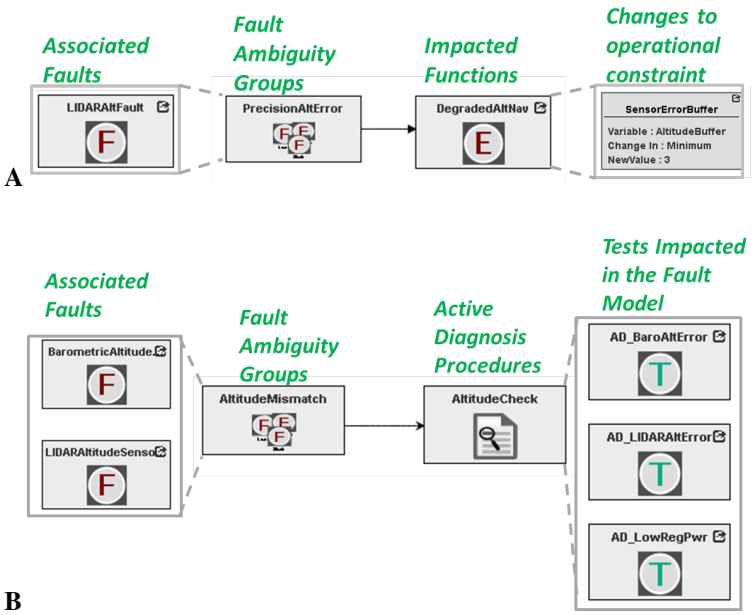


Figure 5. **A**: S1000: Fault Impact Model. **B**: S1000: Diagnostic Refinement Model.

may be executed such as system mode, minimum altitude, ground speed, environmental conditions, etc. (Schumann et al., 2019).

4.5. Recovery Models

Recovery models map the diagnosed faults (triggering faults) to the recovery plans that may be executed. Furthermore, they list the requirement to execute the recovery plan in terms of mode and system operational variables. They also capture the resulting system mode once the recovery procedure has been completed and any resulting operational constraints due to the recovery procedure. For example, a recovery plan might call out to switch to the backup battery. Since, however, that battery has a lower capacity, strong climbs should be avoided after the switch. In contrast to contingency plans, recovery plans concern system reconfiguration (e.g., use a different sensor) and operational changes to subsystems of the UAS.

4.6. Cross links between models

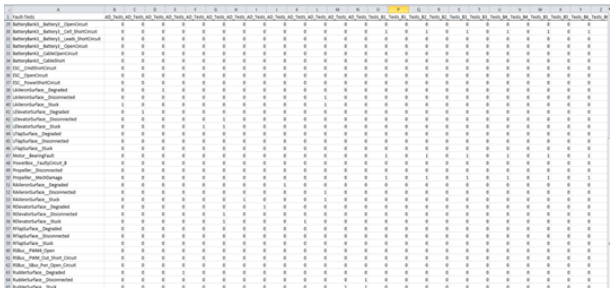
FML allows cross-links or cross-references across the different model types to provide context to each model. For instance, the lowest level functions in the functional decomposition model are related to blocks/components in the system model, which implement the associated function(s). Likewise, the fault propagation model references functions in the "Effect" nodes to capture functional degradation due to the presence and propagation of one or more faults. The fault ambiguity groups in the fault impact model and refinement model include one or more faults cross-referenced from the fault propagation model. The active diagnosis procedures in the refinement model cross reference any Tests in the system model that can be assessed when the active diagnosis procedure is executed. System variables are cross-referenced in the fault impact, refinement and recovery models to capture changes to operational range or operating constraints to execute the procedures.

4.7. Compiling FML models

FML provides plugins to compile the models into artifacts that can be used in the deployed AOS system. FML generates a D-matrix based on the fault propagation model for each operational mode of the system, which is a matrix that relates tests to failure modes. The D-matrix generated from this system is shown below (Figure 6A).

Apart from the D-matrix, the FML generates Prolog code to set up the interface between DR and DM (Figure 6B). It generates Prolog rules that aid DM to understand the DR output (the diagnostic hypothesis for the current set of faults). Furthermore, FML generates Prolog code to capture model information that allows DM to set up the functions and variables impacted, the possible active diagnosis procedures that could be executed to refine the diagnosis output, possible local recovery actions to arrest or mitigate a fault and resulting impact on the system performance. DM uses these rules at every stage to plan the next course of action, an active diagnosis procedure, a recovery procedure, or updates to the mission.

A



B

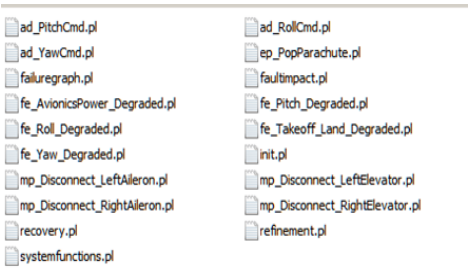


Figure 6. FML output artifacts: generated D-matrix (A) and Prolog files (B)

5. LOGIC-BASED DECISION-MAKING FOR CONTINGENCY MANAGEMENT

During flight, the contingency management system DM has to continuously solve the following problem: can the UAS continue to execute its mission, given the current state of the aircraft and current failure modes? If a safe execution of the mission should not be possible, the DM has to find a suitable contingency plan, which can comprise an alternative ending of the mission, like landing at an emergency airport, pulling the parachute, or another emergency action. During this activity, the DM also needs to trigger active diagnosis procedures if necessary to pinpoint the exact cause of the failure. The DM will always attempt to device a contingency plan that (a) obeys all safety requirements, (b) can be executed safely, given the current and predicted state and capabilities of the UAS, and (c) is least intrusive with respect to the original mission goals.

The main input of the DM is comprised of the current flight plan (i.e., mission) as list of way-points $G = \langle w_1, \ldots, w_n \rangle$, which the UAS is supposed to visit one after the other. Each waypoint is defined as a quadruple $w_i = [lat_i, lon_i, alt_i, spd_i]$ with latitude lat_i, longitude lon_i, altitude alt_i, and target airspeed spd_i. Alternatively, waypoints can be addressed by name as published in the flight charts. The on-board data base can automatically perform the conversion.

In addition, the DM receives input about the current failure state of the aircraft F. This information is provided by DR and R2U2 and contain zero or more individual failure modes and potentially one or more ambiguity groups. Static information about the aircraft performance, waypoints, airports and emergency landing spots, as well as volumes reserved by UAS Traffic Management (UTM) is provided by the on-board data base and via connection to the ground station. With these inputs, the DM (Decision Maker) performs two tasks: failure disambiguation and contingency planning. The DM is implemented in SWI Prolog,[4] which provides powerful automated reasoning, backtracking search and constraint-logic programming. It scheduled by the cFS to be executed every 2 seconds.

5.1. Failure Disambiguation

Whenever the failure modes F contain a failure mode ambiguity group A, there is not enough initial diagnostic information to isolate the failure to a single component or subsystem. For example, an "inconsistent altitude reading" corresponds to an ambiguity group with three elements, because the barometric altimeter, the GPS, or the on-board LIDAR system might be the cause for the inconsistency. In many cases, a proper contingency management requires that the actual root cause is determined, as it can pose severe restrictions on the UAS capabilities. Therefore, DM attempts to perform failure mode disambiguation by triggering the execution of an active diagnosis procedure. When the active diagnosis procedure is run, additional sensor readings allow the diagnosis system to uniquely identify the failure. In our example, an active diagnosis procedure would command the UAS to climb for a short amount of time. During this commanded climb, all measurements concerning the on-board altimeters (Baro, GPS, LIDAR) are carefully monitored and discrepancies are used to determine the root cause. If, for example, the LIDAR reading remains constant whereas barometric altitude and GPS altitude increase during the climb, it can be reasoned that the LIDAR altimeter is faulty.

DM considers the active diagnosis procedures in the same way as contingency plans. However, they have a different goal (disambiguate the failure rather than safely end the mission) and different conditions under which they can be carried out.

5.2. Planning for Contingencies

For the task of constructing a contingency plan, the DM employs a recursive schema-based planning algorithm, which uses backtracking search, constraint logic, and logic-based programming.

For each possible contingency action $\mathcal{A}$ (e.g., divert to emergency airport, pull parachute) we define a *schema*, which is a recursive piece of code. A schema represents a piece of a contingency plan with "holes", which need to be filled by calculations, constraint reasoning, or recursive invocation of other schemas. (Figure 1). The goal of the UAS mission is to fully execute the full remaining flight plan $G = \langle w_i, \ldots w_\perp \rangle$,

[4] swiprolog.org

Algorithm 1 Listing of schema for contingency action $\mathcal{A}$

```
[G', F', C', P' ] = schema_A(G, X, F, C, P)   ▷ G = current flight plan, X = state of UAS, F = failure modes
                                               ▷ C = capabilities and constraints, P = plan so far
C' = get_ac_capabilities(X, F, C)              ▷ what are the current capabilities of the aircraft?
if is_applicable(A, X, C') then
    [Y, G_1] = outcome(A, X, G)                ▷ what is the outcome of A if it would be executed?
                                               ▷ new AC location would be Y with new flight plan G_1
    [X', F'] = predict_forward(X, F, Y)        ▷ predict state of UAS flying from X to point Y
    if is_safe(X', C') then                    ▷ A is safe to be executed
        [G', P'] = schemas(G_1, X', F', C', P.append(A))   ▷ now check the rest of the new flight plan G_1
    end if
end if
```

where $w_\perp$ is the final destination of the mission. Each leg of the flight plan $w_i \rightarrow w_{i+1}$ will be flown in sequence given the current state of the aircraft X and failures F. The DM is then recursively attempting to execute the current flight leg in simulation. If this is possible safely, then the next leg will be processed. If not, other schemas will be tried. The other schemas might require specific actions to be performed (e.g., active diagnosis, reconfiguration, or launching the parachute) or might change the future flight plan, in case, an emergency airport needs to be reached. The search procedure stops if the original goal has been reached, an alternative safe landing place can be reached, or if all possibilities are exhausted. In that case, the DM triggers the ultimate plan, parachute-assisted crash landing.

Listing 1 shows an abstracted, high-level description of a schema for contingency action $\mathcal{A}$. The schema is called given the current flight plan G, aircraft state X, failure modes F, external constraints and capabilities C, and an empty partial contingency plan $P = []$. In a first step, the current aircraft capabilities are calculated based upon the current state, failure modes, and given constraints. After consultation of the aircraft failure model and failure impact model, a new set of constraints is calculated, reflecting the effects of failure on the aircraft behavior. For example, a faulty LIDAR could result in a constraint $alt_{AGL} > 30$ft requiring that the UAS needs to stay at least 30ft above the ground, because barometric and GPS altimeter have larger error margins. Another typical constraint concerns strong climbs, which need to be restricted when engine or elevator failures occur or the battery is weak.

If the considered contingency action $\mathcal{A}$ is applicable in the current state under the current constraints, it is provisionally selected. Then, the implications of $\mathcal{A}$ on the aircraft state Y and the future flight-plan G_1 are calculated by the function `outcome`. Depending on the contingency action that might be $G_1 = G.rest()$ if we can fly to the next way-point, or G_1 might be set to the route to a suitable emergency airport. The code for `outcome` can include queries to aircraft and failure models, queries to the operational database (e.g., to obtain a route to an emergency landing spot), as well as calls to other schemas.

If the contingency action $\mathcal{A}$ is to be carried out successfully, the aircraft must be able to safely transition from the current state X to Y, e.g., be able fly to the first waypoint of the emergency route. This is checked by a simple forward prediction of the system state of the aircraft. This task involves both checking of constraints as well a state updates. Typical constraints concern limitations of altitude or speed, violations of reserved areas of the airspace, or limitations on maneuvers. State updates usually concern consumables. For example, the battery state of charge (SoC) is updated using a simple, deterministic model to reflect the drain of the battery while executing the flight from X to Y. Forward prediction can even change the failure modes if, for example, $\mathcal{A}$ is a failure mitigation action.

If the outcome of the forward prediction is deemed to be safe, action $\mathcal{A}$ is incorporated into the plan. The DM then needs to recursively needs to plan for the remaining parts of G' under the updated aircraft state X', failure modes F', and constraints C'.

In case any of the checks fail, the contingency action $\mathcal{A}$ cannot be considered at this point. Our logic-based search algorithm therefore performs a backtracking step and tries the next available schema. The backtracking search guarantees that all possible alternatives can be tried. The schemas are ordered in such a way that the least mission-intrusive contingency actions are tried first. Table 1 lists a selection of relevant schemas.

With this search-based contingency planning algorithm, the DM will always come up with a contingency plan that

- tries to perform the original mission as long as it is possible in a safe manner,
- tries to execute mitigation actions that change the configuration of the UAS, e.g., to switch to a backup battery or to turn of unnecessary subsystems to save battery power,
- tries to select contingency actions that are as little disruptive to the mission as possible. This includes
  - a shortened flight plan that sacrifices not so important goals, for example, a package delivery,

Table 1. Schemas for on-board contingency planning. $G = \langle w_i, ...w_\perp \rangle$

	Action $\mathcal{A}$	Severity	Contingency plan	Description
[]	empty	0	[]	mission concluded, no action to be taken
FP	flight-plan	0	$\langle w_i, ...w_\perp \rangle$	follow flight plan to next waypoint w_i
AD	active_diagnosis	0–2	$\langle w_k, D_a, ... \rangle$	perform active diagnosis D_a at waypoint w_k
R	reconfig	1	$\langle w_i, w_{i+1}, ... \rangle$	reconfigure aircraft (e.g., alternate battery or sensors)
S	shortcut	1	$\langle w_{i-1}, w_{i+1}, ... \rangle$	skip waypoint w_i. Fly directly to w_{i+1}
DA	deviate-airport	2	$\langle w'_1, w'_2, \ldots, w'_\perp \rangle$	deviate to emergency airport at waypoint $w'_\perp$ using emergency flight plan $\langle w'_1, w'_2, \ldots, w'_\perp \rangle$. Original mission G is terminated
LI	land-immediate	3	$\langle w' \rangle$	land at nearest safe waypoint w'. G is terminated
P	parachute	4	[]	pull parachute at current location. G is terminated

- diverting the UAS to a suitable and safe emergency airport, and
- pulling the parachute as a last resort to minimize crash impact.

Figure 7 visualizes the search for a contingency plan. The aircraft is currently at $\langle w_1, w_2 \rangle$ and DM determined that it can fly safely to w_3, where it will perform an active diagnosis procedure. In the worst-case outcome, the original flight-plan cannot be followed anymore, neither can the aircraft be reconfigured or a shortcut taken (unsuccessful alternatives marked in red). The currently active schema diverts the aircraft to an emergency airport, which can be reached via waypoints $\langle w'_1, w'_\perp \rangle$. That new flight-plan is being checked by DM for feasibility and safety. Reaching a safe landing spot $w_\perp$ concludes the search and results in the contingency plan $\langle w_1, w_2, ad, w'_1, w'_\perp, \perp \rangle$. Options that still could be explored by DM are shown in blue. Note that the Prolog system does not keep the entire search tree in memory, but just the active path (shown in black). Therefore, DM has a small memory footprint.

Numerous additional constraints must be obeyed during the search for a suitable contingency plan. For example, certain contingency actions must not be repeated (e.g., switching of batteries), others might be re-tried in certain intervals. Our reasoning-based framework furthermore allows us to use constraint satisfaction techniques to calculate important parameters of the contingency procedures. For example, a safe all-engines-out glide into an emergency airport requires a minimal altitude at the preceding waypoints. This minimal altitude is automatically propagated backwards to the current aircraft position and will cause a backtracking step if the aircraft is not high enough for the glide.

6. CASE STUDY

The DM has been test-flown successfully as the contingency manager component of AOS. In this paper, we describe the setup and execution of a test-flight, which was carried out with a NASA DJI S1000+ octocopter at the NASA Ames research center (Figure 8). Although this case study does not exercise most of the advanced features and capabilities of AOS and DM, our successful test flight shows the basic principles of on-board diagnosis and decision-making with DM.

This UAS is controlled by a PixHawk flight computer, running a customized version of ArduCopter. The on-board AOS system is running under Linux on a small NUC computer, which also establishes a WiFi connection to the demonstration ground station. The goal of this test-flight was to demonstrate our on-board diagnosis and prognosis capabilities working together with the DM. The overall flight scenario, which we do not describe in detail here involved three UASs, which controlled and coordinated by the NASA UTM system.[5]

The task of our UAS was to make a survey and perform measurements while flying a meandering trajectory. Figure 8A shows the designated trajectory as a green dashed line. UTM has assigned the blue volume of the airspace, a space in which the UAS can move freely. The other two UASs in this scenario have been assigned the yellow and orange regions. Obviously, a UAS is not allowed to leave its own volume without an emergency and trespass into volumes reserved by other UASs. There are three helipads, marked "H" on which a UAS can land.

For carrying out the survey, precise altitude readings are required. To this end, the UAS is equipped with a simulated LIDAR. When this test-flight starts, the UAS is provided with the sequence of waypoints (green dashed lines), the overall geometry, as well as a current snapshot of the UTM volumes. After take-off[6] the UAS flies along the trajectory at an altitude of about 4m AGL. Figure 9A,B show the actual flown

[5] https://utm.arc.nasa.gov/index.shtml

[6] Due to safety reasons, AOS does not control takeoff or landing. Rather takeoff and landing is performed by the safety pilot, who then hands over control to the AOS, once the UAS is stable in the air and its flight control has been checked out.

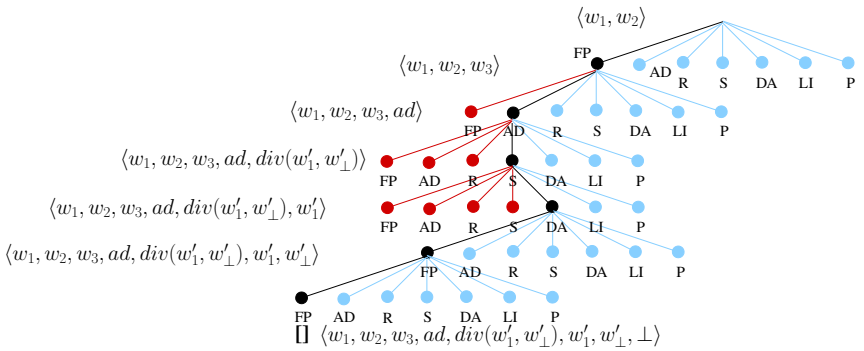


Figure 7. Search tree of DM for current situation $\langle w_1, w_2 \rangle$. Current path (black), unsuccessfully explored (red), potentially explorable (blue).

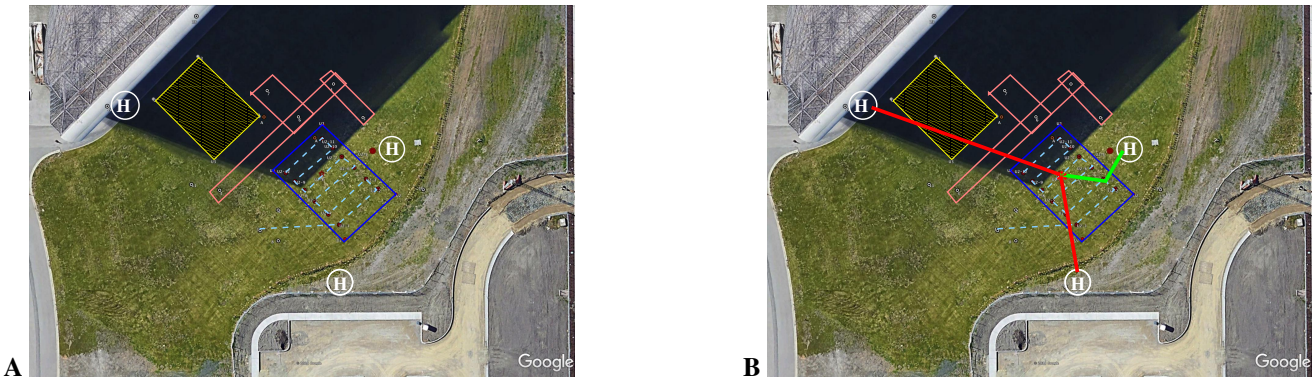


Figure 8. **A:** Image of the test field with UTM reserved volumes; the intended flight plan G for our UAS is shown as green dashed lines. **B:** Situation after the active diagnostic procedure (climb) has been performed and and revealed a LIDAR failure. Since mission and safety constraints forbid the continuation of the mission, a flight to an emergency heliport (H) is considered. Solid lines mark potential flight routes that had been considered by the DM. Red lines correspond to rejected contingency plans; the flight plan shown as green solid lines is selected by DM and sent to AOS and PLEXIL for execution. Satellite image ©Google.

vertical and horizontal profile as measured by the on-board sensors.

At about 120 seconds into the flight, the LIDAR altimeter fails through fault injection by the ground station. The diagnostic reasoner and R2U2 report discrepancies in altitude readings. Since the cause cannot be determined (the fault could lie with the LIDAR, the barometric altimeter, or the GPS), the DR reports an ambiguity group to the DM. The red line and dot in Figure 9 indicates when this happens. Since this fault signature might be safety-relevant, the DM consults the on-board fault model and is provided with the information that there is an active diagnosis procedure, which could disambiguate the faults. This active diagnosis procedure is defined as a climb by 4m, during which all values of all altimeters are being monitored. During that climb, all sensors should report an increasing altitude. Our "broken" LIDAR returns constant values, so the diagnoser is able to resolve the ambiguity toward "LIDAR failure". In our model, the active diagnosis procedure is specified that it can only be carried out at specific locations (waypoints). For example, performing this diagnostic climb while flying under a bridge would be prohibited. The purple lines in Figure 9 show when the active diagnosis takes place. The immediate climb is evident.

With this diagnosis, the DM now starts the search for a suitable contingency plan. Based upon the mission profile, a working LIDAR is required to perform the rest of the survey. So, a "proceed as planned" is not possible. There is no second LIDAR on board, so the reconfiguration schema fails. Similarly, the pre-defined profile does not allow to take any short-cuts. Next on the list of available schemas (Table 1) is the deviation to an emergency airport. A look-up in the on-board data base reveals that there are three possible landing spots. DM now tries to plan trajectories to the emergency airports, given the current state of the UAS. The landing spot to the south fails, because it is behind the fence; flying over the fence would require LIDAR precision navigation. The second airport to the North-west is the next candidate. While planning that trajectory, the DM detects that all available tra-

jectories would lead through UTM volumes, which have been reserved by other UASs. Therefore, that route is not safe. The third airport requires a more complicated approach route (green line in Figures 8B and 9B). However, no constraints fail, and this route is selected as the contingency plan. This plan is then executed by the PLEXIL planner, which brings the UAS safely to the landing spot (green dot). The vertical green line in Figure 9 shows, when the demonstration scenario ends successfully and the safety pilot takes over to manually land the vehicle.

Since a contingency plan was found, more severe actions like "land immediately" or popping the parachute did not need to be considered. Note, that the DM remains active even during execution of a contingency plan. That makes it possible to safely react to additional failures that might show up while flying to the emergency airport.

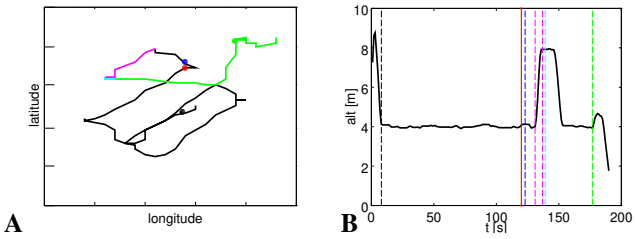


Figure 9. Horizontal (A) and vertical (B) flight profile for the test flight as recorded by the on-board sensors. Time points for fault injection (red), active diagnosis (magenta), and execution of contingency plan (green) are marked in the panels.

7. RELATED WORK

Due to restricted on-board computing capabilities, contingency management for autonomous spacecraft have been kept very simple: typically, when a failure occurs during a mission, the vehicle enters a defined "safe" mode and waits for commands from the ground station. Popular UAS autopilot systems, like ArduPilot[7] or PX4 (Meier, Honegger, & Pollefeys, 2015) provide a simple method for contingency management in the form of "fail-safe" actions: if a failure or a specific predefined condition occurs, then immediately a specific emergency action is triggered. Typical actions include: return to home plate, loiter at the current location, or land immediately.

Approaches to more complex contingency planning for UASs have been developed for the planning and pre-flight assessment (DiFelici & Wargo, 2016) or concern a "holistic" multi-level contingency management system that spans UAS, communications, weather, and battle teams (Franke, Hughes, & Jameson, 2006). The actual planning uses the Lockheed Martin tool TeamWorks (Franke et al., 2006) and only has limited control or monitoring capabilities on-board. On-board path planning by dynamic probabilistic reconfiguration is described in (Wzorek & Doherty, 2006), but does not incorporate diagnosis or failure-based contingency management.

Prognostics-based decision making in the Aerospace domain has been addressed, for example in (Balaban & Alonso, 2012). Here, techniques from optimization and game theory have been used for Dynamic Constraint Redesign (DCR), which enables decision making in a continuous space and deals with mission reconfiguration. The underlying numerical algorithms, based, for example on Particle Filters (Sweet et al., 2014) have been employed for different autonomous vehicles. This formulation can deal with complex and continuous mission reconfigurations but is mathematically more challenging and has a substantial computational footprint.

In the past, the Timed Failure Propagation Graph (TFPG) models and the diagnosis and anomaly detection algorithms in the associated tool set (Fault Adaptive Control Technology or FACT) have been used for real-time diagnosis, reconfiguration, and fault management of a generic aircraft fuel management system (Abdelwahed et al., 2009; Karsai, Biswas, Abdelwahed, Mahadevan, & Manders, 2006), actuator faults in manned and unmanned rotocraft (Karsai et al., 2006; Drozeski, 2005), and real-time software health management (Dubey, Karsai, & Mahadevan, 2011). In these cases the focus was on real-time diagnosis. A prescribed reconfiguration was followed in most cases. A model based diagnosis and deliberative reasoning scheme (Mahadevan, Dubey, Balasubramanian, & Karsai, 2013), wherein the reconfiguration decision was based on the solution of a Boolean satisfiability problem, was demonstrated for reliable software health management.

Our current work is different in that it uses and extends the TFPG models to capture the impact of faults on system performance and expected functionality. Furthermore, our approach models the performance impact of possible solutions for refining the diagnosis results and reconfiguring the system. The decision maker explores viable alternatives presented in the model taking into account current mission conditions and additional constraints imposed by the faults and the associated reconfiguration.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we we have described the Decision Maker DM, a model-based software component for autonomous contingency planning. System health and failure models are captured in our Fault Modeling Language FML and provide the necessary information to perform the logic-based reasoning.

DM closes the gap between the increasingly powerful on-board prognostics and health management systems and the control of the aircraft. On a manned aircraft, the pilot relies on knowledge and experience to detect when something

[7] http://ardupilot.org/ardupilot/

is wrong, performs the necessary active diagnosis procedures to pinpoint the failure, and comes up with a safe and suitable emergency and contingency plan. On an autonomous system, an automated component—our DM—has to perform this role. The model-based, schema-driven planning method uses discrete search to come up with a contingency plan that can be executed safely and that impacts the mission goals as little as possible.

As the DM closes the loop between perception/diagnosis/prognostics and control of the UAS, it must be considered as safety critical. Future work focuses on verification and validation (V&V) of such a system and its potential for certification. The foundation of DM on formal logic might make it possible to provide strong guarantees about the operations of DM.

REFERENCES

Abdelwahed, S., Karsai, G., & Biswas, G. (2005). A consistency-based robust diagnosis approach for temporal causal systems. In *16th International Workshop on Principles of Diagnosis* (pp. 73–79).

Abdelwahed, S., Karsai, G., Mahadevan, N., & Ofsthun, S. C. (2009). Practical considerations in systems diagnosis using timed failure propagation graph models. *Instrumentation and Measurement, IEEE Transactions on*, *58*(2), 240–247.

Balaban, E., & Alonso, J. (2012). An approach to prognostic decision making in the aerospace domain. In *Annual conference of the prognostics and health management society.*

DiFelici, J., & Wargo, C. (2016). UAS safety planning and contingency assessment and advisory reseach. In *2016 Integrated Communications Navigation and Surveillance (ICNS)* (p. 8E3-1-8E3-16).

Drozeski, G. R. (2005). *A fault-tolerant control architecture for unmanned aerial vehicles* (Unpublished doctoral dissertation). Georgia Tech.

Dubey, A., Karsai, G., & Mahadevan, N. (2011). Model-based software health management for real-time systems. In *2011 Aerospace Conference* (pp. 1–18).

Franke, J. L., Hughes, A., & Jameson, S. C. (2006). Holistic contingency management for autonomous unmanned systems..

Friedenthal, S., Moore, A., & Steiner, R. (2008). *A Practical Guide to SysML: Systems Modeling Language*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Geist, J., Rozier, K. Y., & Schumann, J. (2014). Runtime Observer Pairs and Bayesian Network Reasoners On-board FPGAs: Flight-Certifiable System Health Management for Embedded Systems. In *Proceedings Runtime Verification (RV14)* (Vol. 8734, pp. 215–230). Springer.

Gundy-Burlet, K. (2013). Validation and Verification of LADEE Models and Software. In *51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition.* American Institute of Aeronautics and Astronautics.

Karsai, G., Biswas, G., Abdelwahed, S., Mahadevan, N., & Manders, E. (2006). Model-based software tools for integrated vehicle health management. In *2nd IEEE International Conference on Space Mission Challenges for Information Technology (SMC-IT'06)* (pp. 438–442).

Kulkarni, C. S., Roychoudhury, I., & Schumann, J. (2018). On-board battery monitoring and prognostics for electric-propulsion aircraft. In *2018 AIAA/IEEE Electric Aircraft Technologies Symposium.*

Lowry, M., Bajwa, A. R., Pressburger, T., Sweet, A., Dalal, M., Fry, C., ... Mahadevan, N. (2018). Design considerations for a variable autonomy exeuctive for uas in the nas. In *2018 AIAA Information Systems-AIAA Infotech @ Aerospace.*

Lowry, M., Pressburger, T., Dahl, D., & Dalal, M. (2019). Towards autonomous piloting: Communicating with air traffic control. In *Scitech 2019.*

Luo, J., Tu, H., Pattipati, K., Qiao, L., & Chigusa, S. (2005). Graphical models for diagnosis knowledge representation and inference. In *Autotestcon, 2005. IEEE* (p. 483-489).

Mahadevan, N., Dubey, A., Balasubramanian, D., & Karsai, G. (2013). Deliberative, search-based mitigation strategies for model-based software health management. *Innov. Syst. Softw. Eng.*, *9*(4), 293–318.

Maroti, M., Kecskes, T., Kereskenyi, R., Broll, B., Volgyesi, P., Juracz, L., ... Ledeczi, A. (2014). Next generation (meta)modeling: Web- and cloud-based collaborative tool infrastructure. In *8th Multi-Paradigm Modeling Workshop.* Valencia, Spain.

Meier, L., Honegger, D., & Pollefeys, M. (2015). PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 6235–6240).

Reinbacher, T., Rozier, K. Y., & Schumann, J. (2014). Temporal-logic based runtime observer pairs for system health management of real-time systems. In *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)* (Vol. 8413, pp. 357–372). Springer.

Rozier, K. Y., & Schumann, J. (2017). R2U2: tool overview. In *Proceedings rv-cubes 2017* (pp. 138–156).

Schumann, J., Mahadevan, N., Sweet, A., Bajwa, A. R., Lowry, M., & Karsai, G. (2019). Model-based system health management and contingency planning for autonomous UAS. In *AIAA Scitech 2019 Forum.*

Schumann, J., Roychoudhury, I., & Kulkarni, C. (2015). Diagnostic reasoning using prognostic information for unmanned aerial systems. In *PHM15.*

Schumann, J., Zollitsch, A., Mumm, N., & Holzapfel, F. (2018). Safety monitoring and prognostics for automatic aircraft take-off. In *Proceedings of the Annual Conference of the PHM Society.*

Sweet, A., Gorospe, G., Daigle, M., Celaya, J. R., Balaban, E., Roychoudhury, I., & Narasimhan, S. (2014). Demonstration of prognostics-enabled decision making algorithms on a hardware mobile robot test platform. In *Annual conference of the prognostics and health management society.*

Verma, V., Jonsson, A., Pasareanu, C., & Iatauro, M. (2006). Universal Executive and PLEXIL: Engine and Language for Robust Spacecraft Control and Operations. In *Spacecraft control and operations, american institute of aeronautics and astronautics space 2006 conference.*

Wzorek, M., & Doherty, P. (2006). Reconfigurable path planning for an autonomous unmanned aerial vehicle. In *2006 International Conference on Hybrid Information Technology* (Vol. 2, pp. 242–249).

NOMENCLATURE

AGL	Above Ground Level
AOS	Autonomous Operating System
cFS/cFE	NASA Core Flight System/Executive
DM	Decision Maker
DR	Diagnostic Reasoner
DSML	Domain Specific Modeling Language
FML	Fault Modeling Language
GPS	Global Position System
LC	Limit checker
PLEXIL	Plan Execution Interchange Language
PRGN	Prognostics unit
R2U2	Realizable, Responsive, Unobtrusive Unit
TFPG	Timed Failure Propagation Graph
UAS	Unmanned Aerial System
UTM	UAS Traffic Management

BIOGRAPHIES

Dr. Johann Schumann received his PhD (1991) and German habilitation degree (2000) in Computer Science from the Technische Universität München (TUM) in Germany. He is engaged in research on system and software health management, autonomy for UAV, V&V of advanced air traffic control systems, and the automatic generation of reliable code. Dr. Schumann is author of a book on theorem proving in software engineering and has published numerous articles on automated deduction, automatic program generation, V&V of safety-critical systems, and neural network oriented topics.

Nagabhushan Mahadevan is a Senior Staff Engineer at the Institute for Software Integrated Systems, Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN, where his work is focused on using model-based techniques towards diagnosis, distributed diagnosis, software health management, and resilient cyber-physical systems. He received his M.S. degree in Computer Engineering and Chemical Engineering from the University of South Carolina, Columbia, and B.E.(Hons.) degree in Chemical Engineering from Birla Institute of Technology and Science, Pilani, India.

Dr. Michael Lowry is the NASA chief scientist for Reliable Software Engineering. After receiving his BS/MS from MIT and PhD from Stanford, all in computer science, he joined the Kestrel Institute as PI working on program synthesis. In 1993 he joined NASA Ames as group lead then area lead, and was promoted to chief scientist in 2008. Dr. Lowry is the editor of MIT Press "Automating Software Design" and serves on the editorial board of the journal Automated Software Engineering. He has published numerous papers principally on the topics of program synthesis and software V&V.

Dr. Gabor Karsai is a Professor of Electrical Engineering and Computer Science at Vanderbilt University, and Senior Research Scientist at the Institute for Software-Integrated Systems. He conducts research in the design and implementation of cyber-physical systems, in programming tools for model-driven development environments, in the theory and practice of model-integrated computing, and in real-time fault diagnostics. He received his B.Sc., M.Sc., and Dr. Techn degrees from the Technical University of Budapest, Hungary, in 1982, 1984 and 1988, respectively, and his PhD from Vanderbilt University in 1988. Dr. Karsai has worked several large DARPA projects in the recent past: advanced scheduling and resource management algorithms, fault-adaptive control technology that has been transitioned into aerospace programs, and model-based integration of embedded systems whose resulting tools are being used in embedded software development tool chains.