Tool-Supported Software Contingency Analysis

Robyn Lutz[1], Ann Patterson-Hine[2], Scott Poll[2], Charles Domagala[3], and Sudipto Ghoshal[3]
[1]Jet Propulsion Lab, Caltech/ISU, [2]NASA Ames Research Center, [3]Qualtech Systems, Inc.
rlutz@cs.iastate.edu, {apatterson-hine,spoll}@mail.arc.nasa.gov, {chuck,sudipto}@teamqsi.com

Abstract

Model-based development of aerospace software systems often defers fault modeling until after the functional behavior of the system has been specified. This has the advantage of keeping the model less cluttered and easier to review. However, deferring consideration of potential contingencies (anomalous situations) until after the modeling of nominal behavior has been completed has several disadvantages. Early consideration of contingencies is especially important in autonomous systems so that adequate monitorability and controllability will be built in from the beginning. We report our experience using tool-supported integrated modeling of a system and its faults within a widely-used, software-engineering framework to analyze monitorability and controllability. We describe results from application to three aerospace subsystems (an unpiloted aerial vehicle, a spacecraft, and a power testbed). We propose that use of this approach on autonomous systems can help keep the model easy to review and update while providing early integration of contingency analysis into the functional model.

1. Introduction

Some aerospace projects focus development efforts on the normal behavior required of the system to such an extent that consideration of failure scenarios is deferred until after design implementation. In this paper we report experience showing that early analysis of contingency scenarios that the software must handle provides many advantages in building autonomous aerospace systems.

A contingency is an anomaly that must be anticipated and handled during operations. Contingencies include, but go beyond, traditional fault protection. They are obstacles to the fulfillment of the system's high-level requirements that can arise during real-time operations. Contingency handling involves requirements for detecting, identifying, and responding to such anomalous scenarios.

The requirements for contingency handling in many safety-critical, autonomous systems encompass not only traditional fault protection and responses to undesired events but also unexpected environmental or operational scenarios that could contribute to hazards. This is especially true for unmanned and robotic space systems, since the software must monitor for the occurrence of the undesirable behaviors and command an appropriate response without human assistance.

Building the system model with faults integrated into it, rather than later composing a system model with a separately developed fault model, simplifies designing for contingencies. We describe below how our approach—bi-directional contingency analysis to identify missing monitorability and controllability requirements, followed by model-based static analysis to identify the diagnostic adequacy of the design—improved the robustness of the target applications.

With tool-supported fault modeling and static analysis of anomalous scenarios, we can both view the faults associated with each component and connector and statically identify which contingencies cannot be diagnosed with available data and current placement of monitors. Similarly, we can explore which recovery strategies are supported by the current design and formulate recommendations to fill gaps.

The advantages of using a single model for system modeling and safety analysis is well documented [7]. However, there is significant variation in how this is achieved. In some companies the fault model is developed separately and later integrated into the functional model, while elsewhere the model first is developed and then the lowest levels of the model are populated with faults. To keep the model of any large system readable and maintainable as development progresses, tool-supported specification and analysis is necessary.

We report in this paper on our experience in three recent aerospace projects using contingency analysis with tool-supported integrated modeling of the system and its faults to analyze monitorability and controllability. The three systems are the imaging subsystem on an unpiloted aerial vehicle (UAV), the critical-pointing software on a spacecraft, and an electrical power subsystem testbed for evaluation of aerospace health management technologies.

The work described here involves only static analysis of the modeled system during development, not model-based diagnostics for autonomous operations or run-time monitoring for adaptive reconfiguration. Although those approaches can offer increased autonomy, they were not used by any of the three projects described here.

The contribution of this paper is to show how use of this approach helped identify missing monitorability and controllability requirements and design elements. What this means for other aerospace projects is that applying tool-assisted contingency analysis techniques seemed to be a practical way to identify what needed to be modeled to detect, diagnose and respond to anomalous scenarios.

The rest of the paper is organized as follows. Section 2 presents the technique we used to identify contingencies. Section 3 describes the subsequent integrated modeling of the systems and the failures. Section 4 reports the results from application of this software-engineering approach to three aerospace systems and summarizes the lessons learned. Section 5 describes related work, and Section 6 discusses research goals related to practical use of this technique.

2. Contingency Analysis

Fault models tend to describe low-level faults and how they propagate. On the other hand, contingency analysis begins with high-level anomalous scenarios that must be avoided or dealt with in order to meet system requirements, and then considers how and whether those anomalous scenarios could feasibly occur. Put another way, fault models tend to be FMEA-like (Failure Modes and Effects Analysis, described below), proceeding via forward analysis from faults to consequences, whereas contingency analysis tends to be FTA-like (Fault Tree Analysis, also described below), beginning with high-level risks (e.g., failure to power a critical load) and working backward to investigate potential contributing causes (e.g., an erroneous switch command).

For effective analysis, we need to explore both directions, i.e., to perform a bi-directional analysis. In previous work we have described a process that combines a forward analysis from failure modes to effects with a backward analysis from high-level contingencies to causes early in project development [9]. In this paper we extend that earlier work by showing how early contingency analysis can ease tool-supported fault modeling and describing recent results from applications to aerospace systems.

To identify contingencies in these applications we first performed a structured investigation into possible failures using Bi-Directional Safety Analysis (BDSA) [10]. We have previously found BDSA to be useful in analyzing spacecraft fault-protection software. BDSA combines a Software Failure Modes, Effects and Criticality Analysis (SFMECA) to identify failures that would prevent achievement of the functional requirements of the system with a Software Fault Tree Analysis (SFTA) to find possible contributing causes.

The SFMECA structured the investigation of possible software failures. For each input it considered the effect of its absence, corruption, or untimely arrival. It also investigated what would happen if the software hung or failed in each state, or if a transition took it to a wrong state. Fig. 1 shows an excerpt from the power subsystem SFMECA.

To investigate contingencies that did not involve the failure of single components, we also performed a backward analysis. SFTA decomposes a root node into its logical, component preconditions. SFTA considered combinations of circumstances that could together cause a problem.

We used the results from the BDSA as input to a goal-oriented requirements engineering technique called obstacle analysis [15]. It considers alternative ways to handle obstacles (i.e., impediments) to the system's requirements. The process of obstacle analysis identified the contingencies of concern.

Once contingencies have been identified, they need to be resolved. There are some standard strategies to resolve them, such as adding a new software requirement so that the contingency is avoided or changing the domain so that the situation can no longer occur. Resolution involves evaluating and selecting from among the available alternatives. We found that this often involved generating new software requirements for additional monitorability.

3. Integrated Fault Modeling

The bi-directional analysis of contingencies provided initial answers to the questions involved in the subsequent modeling of failures:

(1) What agents or components are involved in

Data	Failure Mode	Description	Effects	Mitigation
Voltage Sensor Input (EI142, 161, 165, 167, 181)	Absent	Data unavailable, missing	Cannot detect if Battery 1 voltage or Battery 2 voltage is low	Look at relay state or current to determine state; calibrate before key maneuvers or phases; track noise
Voltage Sensor Input (EI142, 161, 165, 167, 181)	Incorrect	Out of calibration; Excessive noise; Sensor ID wrong	False diagnosis possible; may use bad battery (V <= 20 volts) or switch needlessly (V > 20 volts)	Look at relay state or current to determine state

Figure 1 SFMEA Excerpt

detecting it? How can the occurrence of a failure be known?

(2) What agents are involved in diagnosing it? How can the failed component be known?

(3) What agents are involved in recovering from it? How can the needed recovery actions be commanded?

We selected the TEAMS toolset [12] for diagnostic modeling for two main reasons. First, it was a system modeling and static-analysis tool familiar to the engineers with whom we worked. Second, having won a NASA Space Act Award, it was of interest to a broader NASA audience to see whether software failures could be modeled in TEAMS.

TEAMS provides hierarchical, graphical modeling capabilities. A model consists of components and connectors, together with component-level failure modes, and checks (called tests) placed at appropriate points to monitor for the occurrence of the failures.

One of the most useful features is that TEAMS automatically produces a diagnostic tree. This tree was useful for static analysis of whether each modeled failure could be detected and diagnosed with the available sensor data and existing checks. Fig. 2 shows an excerpt from the diagnostic tree for the power subsystem testbed.

In those cases in which the existing software checks could not uniquely identify the fault with the available checks, TEAMS listed the set of possible, indistinguishable failures together in a leaf node. This set is called an "ambiguity group" in TEAMS. The fault-modeling approach is bi-directional in that it performs a forward analysis from low-level faults to the situations of concern that they can cause, and a backward analysis from the faults to the feasibility of their occurrence. This maintained traceability between the system and the fault model.

4. Applications and Lessons Learned

In this section we describe our use of this tool-assisted technique on three projects: the imaging subsystem on an experimental UAV, critical-pointing software to be reused on a Mars spacecraft, and an electrical power-system testbed for helping design robust, aerospace health-management technologies. All three applications are or will be safety-critical. All three also involve planned evolution toward increased autonomous monitoring and control for anomalous situations. In all three applications we used the modeling and diagnostic capabilities of the TEAMS toolset to explore contingency scenarios.

4.1 Autonomous Rotorcraft Project

Our first application of tool-supported contingency analysis was to the Autonomous Rotorcraft Project (ARP), a small UAV. Requirements for autonomous handling of a range of operational contingencies were gradually added. The rotorcraft was planned as a safety-critical system both because it eventually will need to perform autonomous take-off and landing in populated areas and because some of the intended missions are safety-critical, such as locating downed pilots or determining the spread of forest fires.

Our main contribution was to identify contingencies of concern on two of the UAV's subsystems, imaging and communication. The goal was to help make the vehicle more robust in detecting and responding to failures and other anomalous situations (e.g., strong crosswinds that interfered with imaging, etc.). Our biggest challenge was that the system was evolving on an on-going basis with new instruments (e.g., a range finder) and new software (e.g., improved vision algorithms) being added regularly. A bi-directional

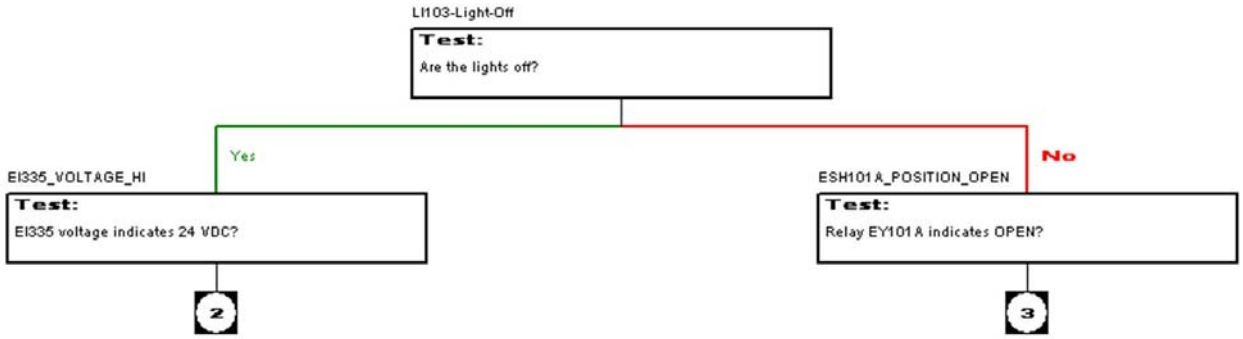

Figure 2 Excerpt from Diagnostic Tree

contingency analysis was performed first and identified several contingencies. For example, one failure mode that could contribute to several hazardous scenarios was attaching incorrect identification data (vehicle pose, attitude, GPS position, etc.) to an image. The effect was that the identification data did not reflect the image content, so could mislead the software into an erroneous control decision. The SFMECA tables had a "Mitigation" column that described ways to eliminate or mitigate the failure mode in each row. Similarly, the SFTA identified ways in which undesirable states could be reached, thereby providing insights into how some contingencies could be prevented.

In order to support the changes involved in the rotorcraft's growth in autonomy, we built a small, architectural-based, graphical TEAMS model of the imaging subsystem using project documentation and weekly group discussions with the project experts. We then used the diagnostic tree automatically produced by TEAMS to evaluate whether the modeled failures were detectable and diagnosable.

If a failure lacked monitorability (i.e., could not be uniquely identified with the available data), it appeared in an ambiguity group. For example, at one point the as-modeled system lacked the monitoring capability to distinguish inadequate stereo imaging due to a camera's dirty lens from inadequate stereo imaging due to a range-calculation failure. The diagnostic tree confirmed that these two failures were not currently distinguishable in the modeled system.

When TEAMS found an ambiguity group of indistinguishable failures, we knew that we needed to consider adding checks in the model (and the software) to gain more insight into what had gone wrong. Sometimes we found that we could not add a check because the model (and the software) currently lacked access to the signal data needed to perform it.

The project applied our results, e.g., by modifying the camera controller to enable autonomous switching between the color and video cameras. The project also built a translator between the TEAMS diagnostic tree (expressible from within the tool in XML) and the language used by the rotorcraft planner. We did a small demonstration of the feasibility of hardware-in-the-loop testing to verify the accuracy of the model on the grounded vehicle. For example, loss of stereo imaging due to an all-black image from the left camera was triggered by manually covering the lens on the left camera and observing that the correct message (switch to right camera) was sent to the planner.

Lessons learned. The most useful results from the static analysis were that the bi-directional contingency analysis found missing software requirements for monitorability, in which the data signals needed to run the checks were unavailable. We also found that the rapid evolution toward autonomy required tool support to maintain traceability between the high-level contingency descriptions and the changing low-level instrumentation and associated potential failures.

4.2 MER Critical-Pointing Software

The second application of tool-supported contingency analysis was to the Critical-Pointing Software developed for the Mars Exploration Rover (MER) spacecraft launched in 2003, and planned for reuse on the Mars Science Lab spacecraft to be launched in 2009. The Critical-Pointing software autonomously points the spacecraft appropriately in case commandability of the spacecraft from Earth is lost (e.g., due to loss of sun-point) and before trajectory correction maneuvers. Mars Science Lab will have expanded autonomous capabilities and advanced fault-tolerance software.

Unlike on the rotorcraft, in this application the contingencies were known. Our job was instead to see whether our contingency-analysis technique could help verify the adequacy of the design on which the Mars Science Lab critical pointing was to be built. The challenge in this application was to ensure that the

output from the available data signals and software checks adequately distinguished among the possible contingency scenarios. The question that we wanted to answer was exactly what is known on the ground when we receive a "Critical-Pointing Failed" message from the spacecraft in telemetry. This message is sent when the critical-pointing software terminates its effort to successfully point the spacecraft.

We thus constructed a TEAMS model containing the components and signals involved in the critical-pointing software as well as their failures. We then used the diagnostic tree automatically produced by TEAMS to investigate the different ways in which the critical-pointing software could decide that it had failed to achieve its goal.

Lessons learned. The most useful result from our work was that we found a gap in the MER software documentation prior to reuse on the next project. Specifically, when we modeled the software based on the available documentation and inspected the automatically generated diagnostic tree, we found an ambiguity group. That is, the available data signals and software checks on the spacecraft could not distinguish two contingencies that we wanted to distinguish. Although the flown code was entirely correct, our model, which was based on the available design documentation, failed to distinguish between these two anomalous situations.

4.3 Advanced Diagnostics & Prognostics Testbed

Our third application of the tool-supported contingency analysis has been to the Advanced Diagnostics and Prognostics Testbed (ADAPT) currently under development at NASA Ames [1]. It is a power-system testbed loosely modeled on the exploration vehicle's Electrical Power System and built using COTS components. ADAPT provides system functions of power generation, storage, distribution, monitoring, and control. It is being built incrementally to support human-rated missons such as Constellation. It serves as a platform for evaluating various modeling and diagnostic analysis tools including TEAMS. The testbed is thus heavily instrumented with many signals.

ADAPT, like the ARP, experiences rapid evolution in terms of added autonomy. For example, many of the reconfigurations of loads that must currently be done manually later will be automated. In power systems on future spacecraft, many of the checks and recovery actions currently done in hardware (or not at all) on current spacecraft will be controlled by software. Thus, the adequacy of the system's software monitorability for diagnosis and controllability for recovery will become key challenges. We need to know which contingencies might be identifiable by software in the future and which responses might be commandable by software in the future.

To anticipate this concern and provide some design direction, we performed a SFMECA on the data signals needed to diagnose and respond to some high-level, critical failure scenarios identified by the project engineers. For example, one such failure scenario is loss of power to all loads on a single load bank from the current power source. Each failure scenario identified a hardware fault that can cause the power loss (here, relay 4L1 on the wiring diagram fails open), indications to help diagnose the cause, and a set of contingent reconfiguration steps to recover from the loss. The SFMEA mitigation column provided insights into what we need to model for future software monitoring (e.g., battery health or load states), testpoints where loads off/on can be confirmed, and which data provide critical or redundant information. For example, if the relay state is reported incorrectly, the voltage or current sensor data can be used to confirm the relay state.

The contingency analysis identified four categories of failure modes recommended for mitigation in future designs in order to provide a more robust system. The first category of failure mode could occur when the design incorrectly assumed that the observed value of a sensor equaled the actual state. This failure mode could be mitigated by the introduction of persistence counters and range checking, and the use of multiple sensors in making decisions. The second category of failure mode occurred when obsolete values were used in control decisions. This failure mode could be mitigated by the time-tags of data used in critical decisions. The third category occurred when a redundant command was not ignored as it should have been but instead triggered additional action. Non-occurrence of this failure mode was, in fact, enforced in hardware, so not a problem. The fourth category occurred when a load that was commanded off was assumed to have powered down but had not. This failure mode could be mitigated by confirming that the load had powered down before turning on a functionally equivalent load. The results of the analysis provided early insights into some issues to be addressed as diagnostics currently performed manually or in hardware are transferred to autonomous software monitoring and control.

Using the failure scenarios developed by the project engineer, modeling experts from QSI built a TEAMS model of ADAPT. As with the rotorcraft, early fault

modeling helped provide timely analysis so that the signals and instrumentation needed for future software autonomy can be more easily built into the design.

The modeling process built an integrated system and fault model. Forty faults and 252 tests are currently represented in the TEAMS model of the system [6]. Switches in the model control which tests are available in which operational modes. In addition, switches enforce mutual exclusion conditions that will be implemented in the control logic, such as that a battery charger can only charge one battery at a time. Automated, bi-directional analysis in TEAMS shows the forward propagation of one or multiple failure signals to the checks that provide detection points and backward tracing of which failure modes are detected by each check.

Lessons learned. Because the testbed is so highly instrumented, we found few missing monitorability requirements. There were some ambiguity groups (of size three and four) in which several faults were all detected by a single check. For example, there was no voltage sensor between a circuit breaker and a relay, so no automatable check can distinguish between failures of these two in the current design. Similarly, in one case a connector failure could not be distinguished from a sensor failure. In addition, we found some faults that could only be detected by human observation of symptoms. The diagnostic analysis of the model also showed that some tests were (properly) not used in certain modes of operation.

Most of the benefits of the application to date came from our demonstration of how TEAMS makes it easier to analyze and visualize the propagation of faults in a larger system with several functionally redundant units. For example, the diagnostic checks are divided into seven categories so that we can choose to run, e.g., only checks involving temperature sensors or only the checks relevant to a particular mode of operation (e.g., Battery 1 powering Load 1).

5. Related Work

De Kleer and Kurien have distinguished two traditions within fault diagnosis [4]. The first, fault-detection and isolation, has been primarily concerned with analyzing the diagnosability of the system and what instrumentation is needed to accomplish that diagnosis. For example, Azam et al. have previously used TEAMS, the tool we use here, to relate a hierarchical, graphical model of the data-driven system architecture to the failure modes and how they are detected [2]. We consider not only failures but also contingencies, a broader class of anomalous operational scenarios.

The second fault-diagnosis tradition is model-based diagnosis. Model-based diagnosis automates the comparison of observed behavior with a predictive model of the behavior to reason about diagnosis, usually with the goal of run-time repair [14,3,13]. Other promising approaches to model-based diagnosis combine off-line and on-line analyses [11]. Delgado, Gates and Roach provide a taxonomy and catalog of run-time, software fault-monitoring tools [5].

Our approach uses automated static analysis of a model to verify the robustness of the software design against envisioned contingencies. The primary limitation of our approach is that, because it does not reason about faults, it cannot reliably detect unforeseen contingencies and hence depends on the quality of the contingency identification.

6. Future Work

We here briefly describe some open issues.

Evolution and maintainability. A key research challenge to broader use of tool-supported contingency analysis remains how it handles evolution. In previous work we found that evolution of the rotorcraft brought more autonomy and new alternatives for improved monitoring [8,9]. Because the TEAMS model was hierarchical and the software architecture did not change, it was straightforward to add, replace, and delete components and connectors. It was also fairly straightforward to update the component failure modes in the model when new capabilities were added (e.g., adding a new range-finder also added failure modes associated with it). Being able to maintain the TEAMS model by adding new components and then producing an updated diagnostic tree reduced the risk that change introduced gaps in monitorability.

It is unlikely that systems in which the software and hardware architecture differ significantly (i.e., where the software architecture instead crosscuts the hardware components) would be as easy to model incrementally. In the aerospace applications studied here, the software architecture tended to follow the system architecture such that both hardware and software faults occurred within the architectural component. We believe that, for embedded spacecraft software, the hardware and software architecture also tend to be similar, but this issue needs to be investigated.

Failure of stereo imaging on the rotorcraft or loss of power to a critical load on a spacecraft can occur due to hardware or software faults. In a system that is evolving toward autonomous failure diagnosis and recovery, detection or control functions that are done

in hardware at one point in time are often done in software at a subsequent time. Modeling both the hardware and software faults that can impede achievement of a required behavior supports the evolution toward autonomy seen in these projects.

Integrating static and dynamic diagnostics. Since our focus has been on identifying missing monitorability and controllability requirements, to date we have performed only static analysis of the faults. However, run-time, model-based diagnostics are supported by another version of TEAMS. Future work will involve integrating the static and dynamic diagnostic capabilities of these tools so that model-based diagnostic analysis of real-time data from ADAPT can occur.

Prognostics. Future spacecraft power systems will need to provide enhanced prognostic (failure prediction) capabilities both for autonomous reconfigurations to avoid failures and to provide opportunity for human-in-the-loop crew decisions regarding potential reconfigurations. It is currently unclear how to systematically extend monitorability for diagnostics to monitorability for prognostics.

Tolerance of ambiguity. We need to have a better understanding of when we are willing to tolerate ambiguity in the identification of failures. For example, do we need to expend resources to uniquely identify a failure if the same resolution will be undertaken for all the failures in the ambiguity set? Criteria for making the design decisions regarding this issue tend to be ad-hoc and related to resource constraints rather than traceable to system-level requirements.

In summary, early bi-directional contingency analysis shows what needs to be modeled to identify failures in a system. Our experience is that this is especially useful for autonomous systems that are being built incrementally. Tool support for early integration of contingency analysis into the model, such as that provided by TEAMS, helps make the solution scalable to larger systems and to systems undergoing rapid evolution.

Acknowledgments

The authors thank Joseph Camisa for helpful explanations of power subsystems. The research described in this paper was carried out in part at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. It was funded by NASA's Office of Safety and Mission Assurance through the IV&V Facility. The first author's research is also supported by NSF grants 0204139, 0205588 and 0541163.

7. References

[1] *Advanced Diagnostics and Prognostics Testbed (ADAPT) System Description, Operations, and Safety Manual*, NASA Ames Research Center, Feb. 21, 2006.

[2] M. Azam, D. Pham, F. Tu, K. Pattipati, A. Patterson-Hine, and L.Wang, "Fault detection and isolation in the Non-Toxic Orbital Maneuvering System and the Reaction Control System", *Proc. Aerospace Conf, 5*, 2004, pp. 2892- 2902.

[3] Steven Brown, Charles Pecheur, "Model-Based Verification of Diagnostic Systems," *JANNAF* 2002.

[4] J. de Kleer and J. Kurien, "Fundamentals of model-based diagnosis," *Safe Process*, 2003.

[5] N. Delgado, A. Q. Gates, S. Roach, "A taxonomy and catalog of runtime software-fault monitoring tools", *IEEE Trans on Software Engineering*, 30, 2004, pp. 859-72.

[6] S. Ghosal, M. Azam and V. Malepati, *FDIR on the ADAPTS Test Bed, Phase III Progress Report 1*, Qualtech Systems, Inc., Oct. 2006.

[7] A. Joshi, S. P. Miller, M. Whalen and M. P. E. Heimdahl, "A Proposal for Model-Based Safety Analysis," *DASC 2005*.

[8] R. Lutz, A. Patterson-Hine, A. Bajwa "Tool-Supported Verification of Contingency Software Design in Evolving, Autonomous Systems", *ISSRE 2006*.

[9] R. Lutz, A. Patterson-Hine, S. Nelson, C. Frost, D. Tal, R. Harris, "Using Obstacle Analysis to Identify Contingency Requirements on an Unpiloted Aerial Vehicle", *Req Eng J*, 12(1), Jan, 2007, pp. 41-54.

[10] R. Lutz and R. Woodhouse, "Requirements Analysis Using Forward and Backward Search," *Annals of Software Eng*, 1997.

[11] T. Mikaelian, B. C. Williams and M. Sachenberger, "Model-based monitoring and diagnosis of systems with software extended behavior" *AAAI05*, 2005.

[12] Qualtech Systems, Inc., www.teamqsi.com.

[13] P. Robertson and B. Williams, "Automatic Recovery from Software Failure: a Model-Based Approach to Self-Adaptive Software," CACM, 2006, vol. 49, 3, pp. 41-47.

[14] L. Trave-Massuyes, I. Russell, J. Thomas, *Model-based Fault Detection and Diagnosis: Articles & Tutorials*, 2004.

[15] A. van Lamsweerde and E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering", *IEEE TSE*, Vol 26, 3, Oct., 2000.