

Identifying Contingency Requirements Using Obstacle Analysis on an Unpiloted Aerial Vehicle

Robyn R. Lutz, JPL/Caltech and Iowa State University, rlutz@cs.iastate.edu
Stacy Nelson, NelsonConsulting/QSS, NelsonConsult@aol.com
Ann Patterson-Hine, Ames Research Center, apatterson-hine@mail.arc.nasa.gov
Chad R. Frost, Ames Research Center, cfrost@mail.arc.nasa.gov
Doron Tal, Ames Research Center, dtal@email.arc.nasa.gov

Abstract

This paper describes experience using Obstacle Analysis to identify contingency requirements on an unpiloted aerial vehicle. A contingency is an operational anomaly, and may or may not involve component failure. The challenges to this effort were: (1) rapid evolution of the system while operational, (2) incremental autonomy as capabilities were transferred from ground control to software control and (3) the eventual safety-criticality of such systems as they begin to fly over populated areas. The results reported here are preliminary but show that Obstacle Analysis helped (1) identify new contingencies that appeared as autonomy increased; (2) identify new alternatives for handling both previously known and new contingencies; and (3) investigate the continued validity of existing software requirements for contingency handling. Since many mobile, intelligent systems are built using a development process that poses the same challenges, the results appear to have applicability to other similar systems.

1. Introduction

This paper describes experience using Obstacle Analysis [15] to identify contingency requirements on an unpiloted aerial vehicle (UAV). A contingency is an operational anomaly that may or may not involve component failure. The problem was how to identify, reason about, and specify the software requirements for handling contingencies on an experimental, autonomous helicopter.

Unpiloted Aerial Vehicles (also called unmanned or uninhabited aerial vehicles) are attractive candidates for future surveillance of high-risk environments such as volcanoes or forest fires; for automated surveys, such as of highway traffic or electrical lines; and for assisted search and rescue

operations, such as for downed pilots or lost hikers [22].

Existing approaches to UAV autonomy tend to use ad hoc approaches to anomalies. The focus has been primarily on developing new system capabilities rather than on identifying contingencies. However, to achieve the levels of robustness necessary to eventually fly in populated airspaces, a more structured approach is needed. A recent study, for example, found the most common cause of UAV failure to be emergency procedures [13].

The requirements for anomaly handling in these autonomous systems have to encompass not only traditional fault protection (e.g., the failure of a sensor or an overpressure in a fuel tank) but also unexpected environmental or operational scenarios that could contribute to hazards. An example of a contingency that does not involve component failure is a strong crosswind that buffets the UAV and interferes with the camera's ability to acquire the images needed for mission success. These broader classes of anomalies that must be anticipated and handled are referred to as "contingencies" [8]. The software requirements for detecting, identifying, and responding to such anomalies we call "contingency handling."

The challenges to the effort reported here were threefold: (1) rapid evolution of the system while operational, (2) incremental autonomy as capabilities were transferred from ground control to software control and (3) the eventual safety-criticality of such systems when they begin to fly in national airspace.

Rapid Evolution. The Autonomous Rotorcraft Project is a NASA UAV project for autonomous helicopter research for low-altitude flight [24]. Two small Yamaha RMAX helicopters, originally developed for remote-piloted crop-dusting in farming regions, serve as the platforms to demonstrate the new autonomous software. The autonomous RMAX

Identifying Contingency Requirements Using Obstacle Analysis on an Unpiloted Aerial Vehicle

Robyn R. Lutz, JPL/Caltech and Iowa State University, rlutz@cs.iastate.edu
Stacy Nelson, NelsonConsulting/QSS, NelsonConsult@aol.com
Ann Patterson-Hine, Ames Research Center, apatterson-hine@mail.arc.nasa.gov
Chad R. Frost, Ames Research Center, cfrost@mail.arc.nasa.gov
Doron Tal, Ames Research Center, dtal@email.arc.nasa.gov

Abstract

This paper describes experience using Obstacle Analysis to identify contingency requirements on an unpiloted aerial vehicle. A contingency is an operational anomaly, and may or may not involve component failure. The challenges to this effort were: (1) rapid evolution of the system while operational, (2) incremental autonomy as capabilities were transferred from ground control to software control and (3) the eventual safety-criticality of such systems as they begin to fly over populated areas. The results reported here are preliminary but show that Obstacle Analysis helped (1) identify new contingencies that appeared as autonomy increased; (2) identify new alternatives for handling both previously known and new contingencies; and (3) investigate the continued validity of existing software requirements for contingency handling. Since many mobile, intelligent systems are built using a development process that poses the same challenges, the results appear to have applicability to other similar systems.

1. Introduction

This paper describes experience using Obstacle Analysis [15] to identify contingency requirements on an unpiloted aerial vehicle (UAV). A contingency is an operational anomaly that may or may not involve component failure. The problem was how to identify, reason about, and specify the software requirements for handling contingencies on an experimental, autonomous helicopter.

Unpiloted Aerial Vehicles (also called unmanned or uninhabited aerial vehicles) are attractive candidates for future surveillance of high-risk environments such as volcanoes or forest fires; for automated surveys, such as of highway traffic or electrical lines; and for assisted search and rescue

operations, such as for downed pilots or lost hikers [22].

Existing approaches to UAV autonomy tend to use ad hoc approaches to anomalies. The focus has been primarily on developing new system capabilities rather than on identifying contingencies. However, to achieve the levels of robustness necessary to eventually fly in populated airspaces, a more structured approach is needed. A recent study, for example, found the most common cause of UAV failure to be emergency procedures [13].

The requirements for anomaly handling in these autonomous systems have to encompass not only traditional fault protection (e.g., the failure of a sensor or an overpressure in a fuel tank) but also unexpected environmental or operational scenarios that could contribute to hazards. An example of a contingency that does not involve component failure is a strong crosswind that buffets the UAV and interferes with the camera's ability to acquire the images needed for mission success. These broader classes of anomalies that must be anticipated and handled are referred to as "contingencies" [8]. The software requirements for detecting, identifying, and responding to such anomalies we call "contingency handling."

The challenges to the effort reported here were threefold: (1) rapid evolution of the system while operational, (2) incremental autonomy as capabilities were transferred from ground control to software control and (3) the eventual safety-criticality of such systems when they begin to fly in national airspace.

Rapid Evolution. The Autonomous Rotorcraft Project is a NASA UAV project for autonomous helicopter research for low-altitude flight [24]. Two small Yamaha RMAX helicopters, originally developed for remote-piloted crop-dusting in farming regions, serve as the platforms to demonstrate the new autonomous software. The autonomous RMAX

system has an onboard attitude sensor (accelerometers and gyros), a GPS sensor, and a communication modem [22]. The rotorcraft has four cameras—a stereo pair of cameras to provide input to a passive range estimation algorithm, a camcorder to provide video recording, and a color camera to provide situational awareness. Ground support is housed in an instrumentation trailer with a GPS ground station, radio modems for communication with the aircraft, and video displays of the camera images.

Two key components of the software architecture for the system are Apex and CLAW. The Apex reactive planner selects actions for execution onboard the rotorcraft based partly on a library of stored partial plans. Apex has the capability for path planning around obstacles. The inner and outer-loop Control Laws (CLAW) provide attitude stabilization and waypoint guidance control. CLAW has a hardware model that is used for limited hardware-in-the-loop testing on the ground. In addition, an operator on the ground can adjust variable values and alter the state of the control law during execution to provide ground control. A simulation environment provides both testing of the reactive planner and a high-fidelity simulated view of the rotorcraft and its operating environment synthesized from telemetered vehicle state and position data.

The rapid development process for the UAV is similar to incremental prototyping except that the vehicle is already operational (flying) during the process. Requirements evolve rapidly as autonomous features are added regularly.

Incremental Autonomy. Autonomy is the capability of a software system to make control decisions on its own. The advantage of autonomy in an unpiloted vehicle is that it can react faster than ground-based controllers to failures, anomalous situations, and changes in environment. On the UAV incremental autonomy means either (1) something previously done manually (by operator control) is now done automatically or (2) something that could not be done manually is now done automatically. An example of the first category of incremental autonomy is the shift from a remote-piloted landing (human-in-the-loop) to an autonomous landing. An example of the second category of incremental autonomy is the move from downlinking to the ground all images taken by the onboard cameras to downlinking only those images in which onboard software finds a feature of interest.

Safety-critical focus. The system is an experimental rotorcraft for exploring the feasibility of unpiloted vehicles in the national airspace. Issues of software safety are thus of interest. An early

understanding of possible contingencies and of requirements on the software responsible for detecting, identifying, and handling contingencies contributes to building in safety features.

This paper reports experience with a novel application of goal-oriented requirements engineering to study contingencies in a rapidly evolving, autonomous system. Our approach to contingency analysis was similar to an informal application of Letier and van Lamsweerde's goal-oriented techniques for obstacle analysis [15]. They define an obstacle as a set of undesirable behaviors. The results reported here are preliminary but show that Obstacle Analysis helped: (1) identify new contingencies that appeared as autonomy increased; (2) identify new alternatives for handling both previously known and new contingencies; and (3) investigate the continued validity of existing software requirements for contingency handling. The paper describes the consequences of these results in terms of the requirements engineering of autonomous systems. Since many mobile, intelligent systems are built using a development process that poses the same challenges, the results reported here may have applicability to other similar systems such as mobile robots and rovers.

2. Related Work

The work on contingency analysis described here draws on or has implications for work in three areas: requirements evolution, fault handling in autonomous systems, and goal-oriented requirements analysis. We describe related work in the first two areas in this section. Related work in the last area appears in Section 3 with the description of the approach.

Most work in requirements evolution addresses the pre-implementation phases of a system. Anton and Potts, for example, use goals and obstacle analysis to refine evolving requirements in a developing system [1]. There have also been several studies describing agile approaches for handling rapidly evolving systems (e.g., the "evolutionary prototyping" in [5]). The authors recommend these approaches for market-driven domains such as e-commerce rather than for critical systems.

Requirements evolution post-deployment has been studied primarily from the viewpoint of how it can be managed. The focus is on establishing processes to scope or evaluate proposed changes, e.g., in terms of traceability [7] or change-impact studies. Similarly, maintenance methodologies tend to focus on classifying and managing requirements changes rather than on analyzing changes [2].

The domain of concern in most requirements evolution work is the business environment rather than safety-critical systems such as UAVs. There has been little work that has looked at evolving requirements in operational, critical systems. DeLemos is an exception, modeling an operational system in which requirements evolution (specifically, automating the self-destruct feature of a rocket) can be structured so that the architectural components remain unchanged while their interactions adapt to the changed requirements [9]. Lutz and Mikulski showed that requirements evolved during operations on seven spacecraft to compensate for anomalies caused by hardware degradation or the occurrence of rare events [17,18].

Several recent papers describe autonomy requirements for existing or planned space missions, including fault handling [3,6]. Although the re-planning software onboard each of these systems can respond to some mission anomalies, the focus to date has been on verification of normal behavior rather than on identification of anomaly-related requirements.

Other researchers, motivated by problems with fault identification on rovers, have presented improved algorithms for fault detection or responses. Verma, Langford and Simmons present an algorithm for estimating the dynamic state of a system (e.g., fault identification) from noisy measurements of continuous variables [23]. Dearden et al. generate contingency plans for rovers in the presence of uncertainty regarding timing and resources [8]. In both cases, the algorithms assume that requirements for fault and contingency monitoring and handling have been defined separately (the problem addressed in this paper).

A few researchers specifically address safety constraints in autonomous systems. For example, Fox and Das describe the deployment of software agents for intelligent decision-making in safety-critical medical applications [12]. A European Space Agency ESTEC project has investigated how to ensure safety and dependability of autonomous space software, based on lessons learned from non-space autonomous domains. Among their recommendations is a "safety bag" or safety supervisor that checks constraints at execution time [22]. FDIR (Failure Detection, Identification and Recovery) is handled by a separate module.

The work described here also builds on previous work in vehicle health management. Patterson-Hine et al. modeled the engine and transmission subsystems on a UH-60 helicopter [20]. They investigated potential failures or malfunctions of hardware components, together with the downstream

effect on the system and the fault visibility on the ground via various instrumentation suites. Whalley et al. subsequently described the challenges involved in using vehicle health modeling of a UAV to assist in automated transition from remote control of the vehicle to computer control [24]. Such a transition will require improved contingency analyses and motivates the work reported here.

3. Obstacle Analysis Approach

Our approach was to use a simplified version of the framework for goal and obstacle analysis provided in [14,15,16] to guide our investigation of software requirements for contingency identification and handling. This application of the obstacle analysis was informal and manual. The use of Obstacle Analysis met the criteria for lightweight applications of formal methods to requirements modeling described by Easterbrook et al. [11]: applied in response to an existing development problem, applied selectively to only some critical portions of the requirements, offers only a partial solution without guarantees of completeness or correctness, and fed back into the development process to improve the product.

In this section we give a brief description of Letier and van Lamsweerde's obstacle-analysis framework. The reader is referred to [14,15,16] for a detailed account of the goal-oriented approach. The subsequent section then describes our use and adaptation of the obstacle analysis framework.

The underlying rationale for this work is that contingencies are either obstacles to achieving goals or are indications that the goals are unrealizable with the available agents. Contingency handling involves the introduction of new goals or new agents to resolve an obstacle. Incremental autonomy, which involves replacing human agents with software agents, produces new sub-goals to achieve the autonomy, as well as new obstacles, new sub-goals to resolve those obstacles, and new alternatives to resolving previous obstacles.

For example, initial collision avoidance (e.g., of the rotorcraft with simulated buildings) was done via remote control by a pilot steering the airborne UAV from the ground. Later, collision avoidance was done by calculating a path through the obstacles before flight began and then autonomously executing the planned path during flight. Still later, the path calculation handled new obstacles that appeared during flight. Future software will autonomously plan a path during flight based on real-time processing of images taken by the UAV's cameras and on software that autonomously detects changes in

the imaged target. Any solution for contingency analysis thus needed to be readily expandable and maintainable.

An advantage of autonomy is that it adds flexibility to a mission, allowing it to take advantage of unanticipated science opportunities that would be missed if observations had to wait for human interaction. The work described here is not concerned with this sort of a daptive re-planning for science gain, but solely with negative contingencies that can place the system in a hazardous state. Some of the results appear to be applicable for intelligent systems that re-plan for opportunistic science gain, but that investigation is beyond the scope of the work described here.

Goals. A goal defines a set of desired behaviors. Goal-oriented requirements engineering organizes goals in an AND/OR structure (i.e., a directed acyclic graph) in which the AND nodes refine the goals into sub-goals both of which must be satisfied and the OR nodes provide alternative ways to meet the goals. Refinement continues until a sub-goal (i.e., a terminal goal) can be achieved by an agent. Software requirements are terminal goals assigned to software agents.

Agents. Agents (e.g., human operators, hardware devices, or software components) are active objects to whom the implementation of the sub-goals are assigned. (Note that this definition of an agent differs from the notion of autonomous software agents. In the goal-oriented approach an agent may, or may not be software, and may or may not be autonomous.) Software components are the most common agents for UAV autonomy. In the UAV project, the rapid evolution of the system meant that new agents (autonomous features, as well as sensors, cameras, range-finders, etc.) were regularly being integrated into the operational flight system.

Obstacles. An obstacle describes a set of undesirable behaviors. For example, an obstacle to the goal "Store Camera Images in Memory" is "Images Exceed Available Memory." This type of obstacle is called a "non-satisfaction" obstacle since it obstructs the satisfaction of a goal. [15]. Obstacles cover a broad space of possible barriers to achieving required functionality. Contingencies are those obstacles that can arise during real-time operations, such as failures or other anomalous events or scenarios of concern. (Note that, to avoid confusion with the common UAV term "obstacle avoidance"—meaning "collision avoidance"—we use the term "obstacles" in this paper only to mean obstructions to desired behaviors.)

Like goals, obstacles are organized and refined in AND/OR structures, and are associated with the

goals they impede. The example in [15] uses a table to specify the goals, assigned agents, and obstacles for the system being analyzed (the safety-critical London Ambulance System). Obstacles usually are associated with terminal goals, i.e., goals assigned to individual agents. Obstacle refinement patterns, described both formally and as heuristics of the form "if the specification has such or such characteristics then consider such or such type of obstacle to it," are described in [15]. We report experience with these refinement patterns below.

Obstacle resolution. Once obstacles have been identified, they need to be resolved. There are some standard strategies to resolve the obstacles [15]. For example, to eliminate the obstacle, we can consider getting rid of the goal that it obstructs, assigning a different agent so that the obstacle does not occur, adding a new goal to require that the obstacle be avoided, changing (or "de-idealizing" the goal), or changing the domain such that the obstacle can no longer occur. Another strategy is to just tolerate the obstacle, perhaps by adding a new goal (e.g., a new requirement for contingency handling) to mitigate the consequences of the obstacle, or perhaps by deciding to accept the occasional occurrence of the obstacle.

Obstacle resolution involves evaluating and selecting from among the available alternatives. Often this involves the generation of new sub-goals to eliminate, reduce, or tolerate the identified contingency. These resolutions yield new software requirements when assigned to software agents.

4. Results: Contingency Requirements from Obstacle Analysis

We first performed a baseline contingency analysis for two selected subsystems (communications and perception) recommended by the project. Our involvement began about the time that the rotorcraft began regular autonomous flights within a constrained airspace with autonomous collision-avoidance path-planning around stationary objects such as simulated buildings. The project involved a tight-knit team of experts working together with strong technical leadership. Most of our knowledge of the system came from participation in weekly team meetings and from discussions with the experts on the team rather than from limited project documentation.

The project considered contingency analysis of the communications subsystem to be important because many UAVs currently rely upon flight termination (including commanded hard landings) to stop the vehicle in case of communication failure. Improved software contingency handling for communications

failure would make it possible for the UAV to fly to a safe rally point (a pre-designated location) and land normally. The project also considered contingency analysis of the perception system (i.e., the cameras and range finders) to be important. This was both because the cameras are assigned mission-critical responsibilities during some operations and because the cameras can provide backup position or ranging information when other, primary components fail.

4.1 Obstacle Identification

To help identify obstacles for communication and perception we used Bi-Directional Safety Analysis [19]. We selected BDSA because it combines a forward analysis (from potential failure modes to their system effects) with a backward analysis (from the failures to their contributing causes). The forward analysis is similar to a Software Failure Modes, Effects, and Criticality analysis (SFMECA). The backward analysis is similar to a Software Fault Tree Analysis (SFTA). The combination of the forward and backward analyses has proven to be a powerful way to identify and understand the causes of software-associated failures in systems. Previous applications include its use to validate fault protection software on two spacecraft and to analyze thruster failure modes on another.

As input to the obstacle analysis, we produced only those portions of the goal-graph about which there was some confusion or controversy. This was done in order to facilitate review of accuracy by experts. For example, Fig. 1 shows a portion of the goal graph for 3-dimensional collision avoidance. For other

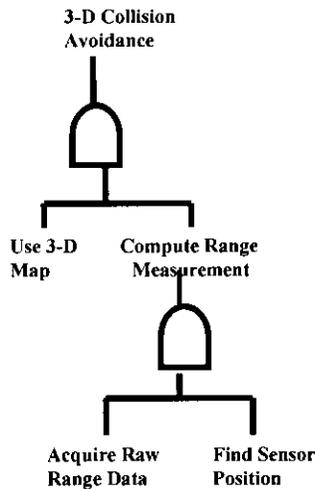


Figure 1. Goal graph

Goals we used architectural diagrams, state diagrams, and functional dependency graphs that we had earlier produced as input to the obstacle analysis rather than producing a separate goal-graph. The decision not to produce a complete goal-graph meant that the obstacle analysis was less rigorous, with no formal proofs of completeness possible. However, obstacle analysis explicitly supports the possibility of such lightweight applications as a way to guide inquiry into potential obstacles. In our case, iterative project review of our specifications gave some assurance that we had adequately captured the high and mid-level goals.

We first performed a forward analysis of the communications and perception subsystems. The SFMECA structured the investigation of possible failures, since it considered the absence, corruption, or untimely arrival of each input. Similarly, the SFMECA considered what would happen if the software hung or failed in each state, or if a transition took it to a wrong state. Details of the method appear in [19]. In this way, several contingencies that involved component failure were identified, such as attaching incorrect metadata to an image (vehicle pose, attitude, GPS position, etc.) with the effect that the metadata does not reflect the image content, and image compression failure that stresses available memory.

Table 1 shows an excerpt from the SFMECA for the data input “Request for Image Processing” received by the onboard camera software. The context is that a raw image is grabbed from the UAV’s video camera stream and is processed (e.g., compressed) onboard before being sent to the ground mode on the subsystem (camera) and the system.

To investigate contingencies that did not involve the failure of single components, we also performed a backward analysis. The SFTA is a graphical decomposition of a root node into its logical, component preconditions [19]. The SFTAs took as root nodes the negations of communication and perception-related goals. In some cases the root nodes were hazards (e.g., collision) that negated high-level goals (e.g., collision avoidance). The SFTA considered combinations of circumstances that could together cause a problem. The SFTAs helped identify alternate ways to get to an undesirable state and, indirectly, to guide analysis of fault detectability and fault propagation.

As an example, we consider a portion of the SFTA for the root node “Failure of Stereo Imaging.” (The SFTA is not shown here due to space constraints.) The root-node failure occurs when either the Left_Camera_Fails OR the Right_Camera_Fails. However, the left camera is redundant in that there is

Table 1. SFMECA Excerpt for the Camera Image Processing Request

<i>Item</i>	<i>Generic Failure Mode</i>	<i>Failure Mode Description</i>	<i>Effects</i>	<i>Criticality</i>	<i>Mitigation</i>
Request for Image Processing	Absent	No processing command received	Raw image not compressed; buffer limit could be exceeded; downlink stressed	Minor	Use default processing; set default to "compressed" to limit memory usage and bandwidth
	Incorrect	Processing settings may be inappropriate for conditions	Poor quality; surveillance mission or autonomous landing may require usable image	Minor to Major	Restrict processing choices based on available info about mission, resource constraints
	Timing	Requested processing applied to earlier/later image	Delay in getting usable image	Minor	Time-tag images to detect discrepancy

both a grayscale and a color camera on the left side. The left camera failure thus occurs only when both the `Grayscale_Camera_Fails` AND the `Color_Camera_Fails`. The SFTAs anchored the investigation of faults that might prevent the functional requirements from being satisfied.

The SFTA provided information about: (1) the goal being studied (the negation of the root node); (2) potential obstacles to the goal (the leaf nodes of the fault tree); (3) necessary detection mechanisms (how to determine that the obstacle has occurred.) That is, any sub-goal to detect the occurrence of a node must be assigned to an agent that has sufficient monitoring capability as discussed below; and (4) possible obstacle resolutions. For example, for an AND node in the fault tree, we can sometimes use the other branch as backup, (such as using the left color camera when the left grayscale camera fails), while for an OR node each child node often requires a distinct resolution strategy.

Some obstacles are environmental. The recent DARPA contest among autonomous vehicles in the Mojave Desert illustrated the importance of contingencies to detect and respond to environmental anomalies. For example, one vehicle's forward progress ended when it became entangled in a barbed-wire fence that it could not "see." Another vehicle went off course when it confused the sun, which was low in the sky, with a huge object to be avoided [4].

In the UAV domain, experience has shown that one obstacle to wireless communication for UAV can be interference from other wireless devices in the vicinity. Several alternative resolutions exist to this obstacle: a return to remote (human) piloting,, ignoring the risk (what [15] calls "live with it", creating an operational policy to not use the wireless

channel in crowded environments, or switching communications to an alternate medium. This last alternative was the best, but required additional hardware and software. The resolution itself thus evolved with the initial resolution being a return to piloted control and a policy change (i.e., a shift in domain assumptions), and the subsequent, longer-term resolution being to switch key transmissions between the rotorcraft and the ground to an alternate medium.

4.2. Obstacle Resolution

Identification of Alternative Resolutions. The construction of the SFMECA provided some insights into resolution options. The right-hand column of the table is a "Mitigation" column and describes ways to eliminate or mitigate the failure mode in each row. The SFTA also helped identify options for resolving obstacles since finding a way to negate leaf nodes removed the occurrence of some obstacles. The obstacle resolution patterns in [15] also provided some guidance. For example, one pattern involves the situation where a condition persists for an interval prior to the obstacle's occurrence. Hence, the obstacle can be anticipated and, perhaps, prevented prior to its occurrence. The "Agent Substitution" obstacle resolution pattern was the one most directly applicable to autonomy. It captures how timeliness obstacles (such as when ground control cannot react quickly enough) can be resolved by transferring responsibility to onboard software.

Selection of a Resolution among Alternatives. Consideration of tradeoffs weighed heavily in the selection of obstacle resolutions in this application. For example, there are two different sensors that can both be used for range finding. However, they vary

significantly in power consumption, precision, number of data points, range, position on the UAV, etc. While each can provide some backup capability if the other is inoperable, any swap involves some intelligent tradeoff analysis either on the part of a ground operator or on the part of the recovery software. Other selection tradeoffs that we encountered involved the choice of color or black-and-white images (with color using more memory), the levels of image compression to be used vs. the CPU usage, the image quality vs. the downlink bandwidth, and the number of stored image frames vs. the size of images.

In some cases a real-time decision has to be made by the onboard software as to how to resolve an obstacle. Alternatives to resolving obstacles in such situations were evaluated on four criteria: (1) Vehicle health status indicators. An example is degraded telemetry bandwidth, which indicates that the transmission of images to the ground may need to be reduced. (2) Availability indicators. This differs from the first criterion in that the first criterion refers to real-time data whereas the second criterion refers to whether the component has been installed on the rotorcraft at this time. It was found useful to separate the two criteria to address those situations in which the preferred alternative was not yet operational, but soon would be. In this paper we draw examples both from past software contingencies and from future, anticipated contingencies. However, in the rapidly evolving UAV system it was essential to maintain rigor in checking consistency between components' current availability and use of the software that invoked it. (3) Image quality indicators. Examples are that the image is not "empty" (i.e., all-black) and that the attributes (such as size) of the image match the attributes of the requested image. (4) Mission profile. At this time a primary consideration is the planned usage of resources—memory to store images, bandwidth to downlink images as well as status data and power usage.

4.3. Deriving Software Requirements

In the example given above, where the obstacle to storing images is "Images Exceed Available Memory," alternative resolutions included "Reduce Number of Images To Be Stored" and "Reduce Size of Images To Be Stored" (i.e., compression). Each of these resolutions itself involved refinement into several sub-goals. For example, reduction of the number of images can be based on how old the image is or on some other assigned or calculated priority measure. The calculated priority can be based on the UAV location, orientation, and camera-pointing

angle or on the actual content of the image. The first option yields an approximation to the likelihood that the target of interest was captured in the image; the second option discards images that did not include the target.

The first option (prioritizing the images based on location, orientation, and pointing angle) was only feasible if the camera software could access the GPS data, which it currently could not. This is an example of what Letier and van Lamsweerde call the *unrealizability* problem, meaning that not all stated goals are realizable by agents in the system. They give some pragmatic conditions for unrealizability in [14]. For our application, the two most important conditions were Lack of Monitorability and Lack of Controllability.

Missing monitorability requirements. One of the most useful results of the Obstacle Analysis approach on the UAV was in identifying gaps between the capability of the assigned software agents to monitor for certain states or events and the need to have them do so. Most of the new software requirements found during the contingency analysis involved the dependency of certain detection, isolation, or recovery actions on specific capabilities that the software currently lacked. In terms of goal orientation, these goals were currently unrealizable. Most often these gaps involved data that the software needed to perform its functions, such as messages to which it needed to be subscribed or resource usage states that it had to track. Identification of missing monitor data added ground visibility into the system state and helped us design for verifiability.

Missing controllability requirements. Similarly, the obstacle analysis identified several instances in which the resolution involved the software being asked to set the values of variables that it did not control. For example, the software responsible for dynamically throttling the writing of images into memory based on their priority level must be able to control (i.e., change) the value of the priority threshold for grabbing images. Another example is that adjusting the jpeg quality of images in response to a low-light contingency can only be realized by software that controls the jpeg parameters. Both monitorability and controllability are important for establishing the software requirements needed for consistent increments in software autonomy.

In some cases, one obstacle can require multiple new software requirements to achieve detection and resolution. For example, mitigating the obstacle where images exceed available memory involves both image compression and a "cancel" command to reverse acquisition of requested images that are no longer needed. In other cases one resolution may

remove several obstacles. This was the case with the addition of accelerometer data on a previous helicopter [20].

4.4 Evolution and Autonomy

The investigation described above identified three main ways that incremental autonomy affects the analysis of contingencies.

4.4.1. Nothing lost. Contingency analysis of the evolving system first verified that previously existing software requirements to detect, isolate, and respond to contingencies were still valid. This involved checking that, for every previously identified obstacle to a goal that could still occur, that the previously identified resolution (usually a derived software requirement) could still handle the obstacle. For example, when the capability was added for autonomous pivoting of the UAV around a target, the existing handling of the obstacle “communication lost” remained valid.

An unexpected finding was that the project sometimes chose to “dial down” the selected level of autonomy, essentially disabling some existing features. Our assumption had been that the level of autonomous contingency handling would only increase. However, at times (e.g., a demonstration or a flight test of a new component) some degree of autonomous control was returned to the remote (human) pilot. These instances usually did not involve a simple rollback to a previous version but a pruned version of the current software. Contingency analysis in such situations was done in an ad hoc manner with consistency maintained primarily by inspection.

This issue of adjustable autonomy has been described by Schreckenghost et al. in the context of space life support systems where a human may need to override autonomous when an anomaly occurs [21]. We plan to investigate whether constraint-checking techniques developed to solve a similar problem in product families (“de-scoping” of features for a baseline product) apply to the UAV domain.

4.4.2. Something gained. Enhancements to the rotorcraft provided new ways to detect, identify, or respond to existing contingencies. Often these enhancements resulted in new sub-goals (e.g., added autonomous functionality) and in new “OR” branches for the obstacle resolution model (i.e., new alternatives for how to handle contingencies).

In general, new sensor agents (e.g., articulation of the stereo cameras so that they can swivel up and down) provided alternative monitoring sub-goals for

contingency detection or alternative control sub-goals for contingency recovery. New software agents (e.g., the capability to dynamically add collision-avoidance targets to the path-planning calculation) offered improved autonomous capabilities for responding to contingencies and resulted in more recovery options.

4.4.3. No free lunch. As more autonomy is required, new obstacles and dependencies were also introduced. That is, with the addition of new agent capabilities came the possibility of new contingencies. For example, the addition of a new laser brings with it the failure modes of the laser to be considered, as well as whether these failure modes are detectable on the ground and by onboard software. Some new agents resulted in new requirements for calibration before use. The addition of a new feature also brings with it potential resource contention (e.g., for power) as well as the need to identify new feature dependencies (e.g., between the fidelity of color images and passive-range algorithms).

While the focus of incremental autonomy tends to be on what is gained in terms of more rapid and flexible handling of contingencies, contingency analysis also looks at the potential threats introduced by the new software complexity. Contingency analysis investigated the failure modes for new agents, new opportunities for feature interactions, and new possibilities for conflicts among goals (most commonly in terms of resource contention). Additional issues such as mixed-initiative control (where the rotorcraft receives inconsistent commands from the onboard software, the ground software, and the remote, human pilot), sensor fusion problems (where the replacement of a single sensor by a suite of perhaps heterogeneous sensors requires the software to compose the data and handle data inconsistencies), and coordination problems (where a fleet of rotorcraft must coordinate their movement and resource usage), were not obstacles in the current system but are potential, future obstacles.

The three categories above (nothing lost, something gained, no free lunch) are clearly interrelated. The evolution of the rotorcraft creates new contingencies but also new options for handling those contingencies. Incremental autonomy results in both new goals and new obstacles, and prompts changes to both obstacle refinement and obstacle resolution. Contingency analysis helped prioritize the objectives of the contingency responses.

Two areas in which existing obstacle analysis techniques provided only limited guidance on the UAV were in analyzing fault isolation and in identifying feature interactions. It is often easier in Failure Detection, Isolation and Recovery (FDIR) to

determine that a failure has occurred (detection) than it is to figure out precisely what happened and how to prevent it from propagating (isolation). For example, if downlink communication stops, it can be quite difficult to isolate the problem. We are working to extend the obstacle refinement patterns to more explicitly address failure isolation issues in this domain.

With regard to detecting interactions as new features are added, Doerr provides guidelines for detection of feature interaction in product lines that were readily transferable to the UAV application [10]. For example, one such guideline that was useful on the UAV is that, if feature B uses feature A, then all features that also use feature A must be identified. Doerr gives an example from the mobile phone domain, where both sending a short message and placing a call use the network component. To avoid conflicts, it is important to identify that both features use the same component.

5. Lessons Learned

The advantages of goal-oriented obstacle analysis for the on-going identification of contingency requirements in the experience reported here were:

- Obstacle analysis helped identify new software contingency requirements in the UAV application. For autonomous systems that are experimental or involve highly innovative features, incremental autonomy seems to be a common mode of development. Incremental autonomy creates new software requirements both for increased functionality and to handle new contingencies. With the added complexity come more ways for the system to display undesirable behaviors as well as new ways of achieving goals.
- Obstacle analysis gave a structured way to reason incrementally about new alternatives for handling contingencies that might need to be addressed. The capacity for incremental reasoning was important in this setting because the system was both currently operational and also evolving rapidly to add autonomous features.
- Obstacle analysis supported evaluation of the continued validity of existing software contingency requirements as the system and the requirements evolved. The cyclical nature of goal-oriented requirements engineering, in which new goals are introduced to resolve an obstacle, but must then also be included in an iterative analysis,

matched the nature of the UAV project quite well. This approach provided as output a specification that explicitly linked requirements and contingencies. Assignment of a goal to a hardware or human agent at one point in time could be replaced by assignment to an autonomous software agent as the software evolved. The contingency analysis could then be updated by checking whether existing requirements were still valid and whether any new obstacles should be generated as a result of the change.

By supporting the identification of software contingency requirements, obstacle analysis contributed to the building of a more robust system.

Acknowledgments. 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 Aeronautic and Space Administration. It was partially funded by NASA's Office of Safety and Mission Assurance Software Assurance Research Program. The first author's research is supported in part by National Science Foundation Grants 0204139 and 0205588. The authors thank Matt Whalley and the other members of the Autonomous Rotorcraft Project team for sharing their expertise and enthusiasm.

References

- [1] A. Anton and C. Potts, "The Use of Goals to Surface Requirements for Evolving Systems," *20th Int'l Conf Software Eng*, Computer Society, 1998, pp. 157-166.
- [2] K. Bennett and V. Rajlich, "Software Maintenance and Evolution: a Roadmap", in A. Finkelstein, ed. *The Future of Software Engineering*. ACM Press, New York, 2000, pp. 75-87.
- [3] G. Brat et al., "Experimental Evaluation of Verification and Validation Tools on Martian Rover Software," *Formal Methods in Systems Design Journal*, to appear.
- [4] Caltech, "Bob Gets His Learner's Permit," *Engineering and Science*, 2004, p. 5-7.
- [5] R. Carter, A. Anton, A. Dagnino, and L. Williams, "Evolving Beyond Requirements Creep: A Risk-Based Evolutionary Prototyping Model," *5th Int'l Symp Req Eng*, 2001, pp. 94-101.

- [6] S.Chien, et al., "Onboard Autonomy on the Three Corner Sat Mission," *Int'l Symp AI, Robotics, and Automation for Space*, Montreal, Canada, IEEE, 2001.
- [7] J. Cleland-Huang, C. Chang and M. Christensen, "Event-Based Traceability for Managing Evolutionary Change", *IEEE Trans on Software Eng*, Computer Society, Los Alamitos, Sept, 2003, pp. 796-810.
- [8] R. Dearden, et al., "Contingency Planning for Planetary Rovers," *3rd Int'l NASA Workshop Planning & Scheduling for Space*, Houston, Texas, October 2002.
- [9] R. deLemos, "Safety Analysis of an Evolving Software Architecture," *5th IEEE Int'l Symp High Assurance Systems*, Computer Society, 2000, pp. 159-167.
- [10] J. Doerr, "Requirements Engineering for Product Lines," Diploma thesis, U of Kaiserslautern, 2002.
- [11] S. Easterbrook, R. Lutz, R. Covington, J. Kelly, Y. Ampo, and D. Hamilton, "Experiences Using Lightweight Methods for Requirements Modeling," *IEEE Trans on Software Eng*, Computer Society, Los Alamitos, Jan., 1998, pp. 4-14.
- [12] Fox, J. and S. Das, *Safe and Sound, AI in Hazardous Applications*, AAAI Press, Menlo Park, CA, 2000.
- [13] T. Johnson, H. Sutherland, and S. Bush, "The TRAC Mission Manager Autonomous Control Executive", *IEEE Aerospace Conference*, Big Sky, Montana, 2001.
- [14] E. Letier and A. van Lamsweerde, "Agent-Based Tactics for Goal-Oriented Requirements Elaboration", *24th Int'l Conf Software Eng*, ACM Press, 2002, pp. 83-93.
- [15] E. Letier and A. van Lamsweerde, "Handling Obstacles in Goal-Oriented Requirements Engineering," *IEEE Trans on Software Eng*, Oct. 2000, pp. 978-1005.
- [16] E. Letier and A. van Lamsweerde, "High Assurance Requires Goal Orientation", *Int'l Workshop Requirements for High Assurance Systems*, Essen, September 2002.
- [17] R. Lutz and I. Mikulski, "Operational Anomalies as a Cause of Safety-Critical Requirements Evolution", *J Systems and Software*, 2003, pp. 155-161.
- [18] R. Lutz and I. Mikulski, "Empirical Analysis of Safety-Critical Anomalies during Operations", *IEEE Trans Software Eng*, Computer Society, March, 2004, pp. 172-180.
- [19] R. Lutz and R. Woodhouse, "Requirements Analysis Using Forward and Backward Search," *Annals of Software Eng*, 1997, pp. 459-475.
- [20] A. Patterson-Hine, et al., "A Model-based Health Monitoring and Diagnostic System for the UH-60 Helicopter," *Am'n Helicopter Society 57th Annual Forum*, AHS, Washington, 2001.
- [21] D. Schreckenghost, et al., "Adjustable Control Autonomy for Anomaly Response in Space-based Life Support Systems," *IJCAI-01 Workshop Autonomy, Delegation, and Control: Interacting with Autonomous Agents*, 2001.
- [22] *Software Product Assurance for Autonomy On-board Spacecraft*, European Space Agency ESTEC. <ftp://ftp.estec.esa.nl/pub/tos-qq/qqs/SPAAS/StudyOutputs>.
- [23] V. Verma, J. Langford, and R. Simmons, "Non-Parametric Fault Identification for Space Rovers", *Int'l Symp AI and Robotics in Space*, 2001.
- [24] M. Whalley, M. Freed, M. Takahashi, D. Christian, A. Patterson-Hine, G. Schulein, and R. Harris, "The NASA/Army Autonomous Rotorcraft Project", *Am'n Helicopter Society 59th Annual Forum*, 2003.