M.S. Project Report

Using Mission Graphs to Identify Mission Dependability Bottlenecks in Complex Systems

Christopher L. Martin

Department of Electrical and Computer Engineering

Carnegie Mellon University

Prof. Phil Koopman, advisor

December 2001

Using Mission Graphs to Identify Mission Dependability Bottlenecks in Complex Systems

Christopher L. Martin

Department of Electrical and Computer Engineering

Carnegie Mellon University

cmartin@ece.cmu.edu

Abstract

Embedded systems of today pose difficult dependability challenges. Hardware and software requirements as well as human interface components all contribute to or detract from the overall dependability of a system. Assigning a 'dependability number' to a system is becoming increasingly subjective due to the confluence of these three areas. In particular it is important to go beyond composing individual component reliability predictions, and additionally consider factors such as ease of user workaround in the face of a partial system failure. In this paper we shall present an approach that attempts to detect these dependability bottlenecks within embedded systems and investigate its ability to represent users' ability to interact with partially failed systems. We propose a graph-based approach that is partially based on composing and extending Unified Modeling Language (UML) standards. This 'mission graph' concept is meant to take advantage of the user's perspective to help system designers understand what is really going on in complex systems. We apply this approach to an example embedded system, and examine the experimental results to determine the feasibility of the proposed approach. Finally, the mission graph approach is used to further investigate the workaround concept and how it applies to users attempting to accomplish their goals even in the face of component failures.

1. Introduction

The notion of assessing dependability of embedded systems must go far beyond traditional reliability measurement techniques to be useful for everyday products. While traditional fault tolerant computing techniques provide tools for combining component reliability estimates to predict system reliability, real systems contain components (such as software) for which reliability estimation is difficult or impossible, include design defects, operate in unanticipated environments, and in general are expected to degrade more or less gracefully in the presence of component failures. Thus, the notion of overall dependability is much more relevant than any single metric such as reliability for embedded systems. ([Laprie92] defines dependability as "the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers.")

We would actually like to actually go further and begin to include user/system interactions within the umbrella of computer system dependability as well. This is due to the fact that many embedded systems have been designed to control or guide the user for various purposes, such as ensuring safety and directing traffic flow. The user has a set of possible states to inhabit, and system actions provide the user with opportunities to change states. When considering control flow types of embedded systems, it is oftentimes the case that people can use these systems differently, and can thus be considered "components" of the system. Hence, it may be useful to represent people as system components. Even though the user plays a significant role in the operation of these types of systems, historically, system behavior graphs have only included the system's components and not the user's behavior. Our work aims to show how analysis of the user's expected behavior can provide insight into the dependability of the system.

In particular, we would like to assess the ability of a user to interact with a system and achieve a mission success even in the face of partial system failures, whether they be due to component failures, extreme environments, design errors, or other causes. This is not exclusively a human/computer interface issue (although certainly that is a factor), but instead an issue of the richness of functionality available to a user to perform workarounds or alternately accomplish goals in the presence of partial system failures. It is in fact the case with many real systems that several ways can exist for a user to accomplish their goal, and hence minor user flexibility can help improve system-level robustness. Thus we are considering things that help a user/system combination to succeed at a mission.

While field data from operational systems is probably the most accurate place to assess dependability, such information is costly to collect and generally comes too late to help system designers make tradeoffs in creating novel systems. So instead, we are searching for simple metrics that can be applied at design time to produce a system that is likely to be robust in terms of survival of minor failures. The goal is to permit rapid comparison of alternate designs and identification of likely dependability "bottlenecks" early in the design cycle. This area of exploration is not yet mature enough for us to aspire to create absolute dependability predictions. So instead, we seek simple, easy to apply metrics that seem likely to be relevant to dependability for typical embedded systems.

In this paper we shall present an approach that attempts to detect these dependability bottlenecks within embedded systems and investigate its ability to represent users'interactions with partially failed systems. We propose a graph-based approach that is partially based on composing and extending Unified Modeling Language (UML) standards. This 'mission graph' concept is meant to take advantage of the user's perspective to help system designers understand what is really going on in complex systems. We apply this approach to an example embedded system, and examine the experimental results to determine the feasibility of the proposed approach. Finally, the mission graph approach is used to further investigate the workaround concept and how it applies to users attempting to accomplish their goals even in the face of component failures.

Section 2 explores existing work in related fields. Section 3 articulates the motivation behind our research on this topic, describes the mission graph approach and proposes its usefulness in detecting mission-level dependability bottlenecks. Section 4 applies the stated approach to a realistic embedded system example and analyzes the results of the exercise. Section 5 uses the mission graph approach in an effort to better understand the effects that workarounds can have on relieving dependability bottlenecks.

2. Prior / Related Work

There currently exist large bodies of work that deal with reliability analysis within a particular domain, whether it be focused on hardware, software, or human-error considerations. For hardware, techniques such as Failure Mode Effects Analysis (FMEA) take advantage of predictable numbers for hardware failures [Villemeur92]. Unfortunately hardware rarely exists in isolation, and software measures have proved more elusive. Comprehensive studies have been done of software reliability measures [Smidts00]. However, these measures were ranked by expert opinion since direct correlation to software failure is difficult to prove. Software FMEA also exists, but only identifies outcomes due to postulated faults – it does not ensure that a system will not enter a hazardous state during operation [Goddard00]. Human users add additional complications. Human error rates and reaction times have been extensively studied [Dreyfuss93] and can depend on a wide variety of factors, including level of training. However, because everyday embedded systems may be deployed in multiple environments and have heterogeneous user populations, it is difficult to assign a single number to the dependability on an interface. Since a modern, complex embedded system contains facets from all three of these domains, we need to move beyond composing individual reliability estimates, which may not be representative of the true performance of the system.

There have been efforts that have attempted to evaluate the dependability of entire systems. In [Brehm96], system dependability is evaluated based on models constructed from the specified characteristics of each of its components. However, the underlying inputs to the model (particularly software and human error representations) can be overly simplistic, and do not take into account the complex interactions these components can have with the rest of the system. Similar to the goals of our approach, [Zemany91] aims to provide a mission-level reliability evaluator. While their objective is to compute several reliability figures of merit that can form the basis of trade-off studies, it is unclear how to use the results of their tool in practical applications. Finally, work in [Kalbarczyk99], and earlier in [Goswami97], describes an approach to evaluate system level dependability using hierarchical simulation. However, the strength of the hierarchical model also provides its weakness - detailed reliability models of the underlying simulated systems are needed for the approach to be truly effective. We believe our approach has promise for providing useful results in practical situations, but like other previous work it is not a complete solution.

Finally, we believe it is important to review related work in the human-computer interaction (HCI) field in order to highlight the differences between their goals and what we are attempting to accomplish with our approach. Perhaps most significant is the GOMS framework [John96], and its related architecture Soar [Rosenbloom93]. Work in this area concerns itself mostly with the goal of evaluating the performance of the human interface component of a system, rather than that of the underlying system as a whole. Also, [Maxion00] investigates human error due to poor user interface design that does not allow the user to adequately cope with exceptional cases (e.g. partial failures). While these topics of study are related to our approach, we seek not just to evaluate the human interface component of the embedded system, but to analyze the entirety of the complex interaction that the user has with the system.

The current state-of-the-art in mission or system level dependability analysis is still in its relative infancy. However, as the complexity of systems continue to increase beyond what traditional component-wise reliability estimates can handle, new approaches are needed that can better integrate the dependability characteristics of all parts of a system into a more representative whole. This specific problem was stated by [Raghavan01], which illustrated the spectrum of current dependability analysis capabilities. He states that as one moves upward in the complexity spectrum from hardware to component to system to mission dependability analysis, the effectiveness of currently available techniques diminishes greatly. He comes to the conclusion that much work needs to be done if reliability estimation at the highest levels is to become plausible.

3. Approach

We would like to further refine the idea of mission-level reliability while incorporating the observation that real systems have built in redundancy that make them more dependable. In systems that involve users (which is indeed often the case with embedded systems deployed in the field), we can observe that people can compensate for partial system failures via the term generically known as a "workarounds". The value of user-available workarounds in desktop, end-user software and in office systems has been explored in [Gasser86], while workarounds in a more traditional embedded system (in this case electronic medical equipment suffering from a partial failure due to a Y2K bug) were examined in [Manning99].

Real, complex systems often fail because of some kind of dependability bottleneck, which is exactly where workarounds would prove their most effective in helping real systems complete their mission. We would like to benchmark, whether it be by a figure of merit or relative comparison, the robustness implications of operational redundancy (i.e. through workarounds). Also, we would like to be able to identify ways to relieve dependability bottlenecks that prevent workarounds from actually working. Also, these comparisons should be available at design time so as to enable dependability comparisons of proposed systems.

Finally, we believe people can be a natural part of redundancy, and we would like to investigate describing how people interact with partially working systems in a formal manner. We believe integrating the user into the dependability analysis of the system for two reasons. First, the user carries implicit state information that affects how the system is used but that may not be entirely known by the system at all times, such as a desired destination for an elevator system. Second, system reliability is measured from the perspective of the end-user interacting with the system. Evaluating the system as it performs as a whole, just as if it were deployed in the field and in use by the customer, is essential in making a good reliability estimate [Musa96]. As a result, it is important to consider the user as part of the dependability equation for an embedded system when dealing with realistic operating scenarios.

3.1 Mission Graphs

If examined from the user perspective, dependability can be seen as a user successfully completing a mission, which consists of a series of tasks. For example, a mission may be riding an elevator, which consists of a series of tasks such as calling the elevator, boarding it, and so on. One way to represent dependability, then, is a graph depicting the possible user paths through the system.

This idea attempts to leverage descriptions of a system which are already available in the Unified Modeling Language (UML). The UML supplies an approach to express requirements and design decisions at various stages in the product development life cycle [UML99]. Ideally, the mission graph of the system from the user's perspective can be created by compiling all of the use cases into a single graph depicting all possible paths though the system.

The user's interaction with the system of interest is modeled as a directed graph in which the nodes are tasks and the arcs can be traversed to accomplish all relevant task sequences. A *mission* is a complete path from a start node to an end node through the system. A *mission success* (MS) is a path that achieves a desired goal. A *mission failure* (MF) is a path that does not achieve a desired goal, but instead reaches a failure point that has negative consequences, such as incorrect service, exceeding permissible service time, user injury, or equipment damage. Additionally a mission failure can occur when a user reaches a dead end within the graph, either due to a design oversight or due to a component failure that "breaks" an arc in the graph during usage. Potential cycles in graphs are resolved by the fact that users tend not to try the same approach to a mission forever.

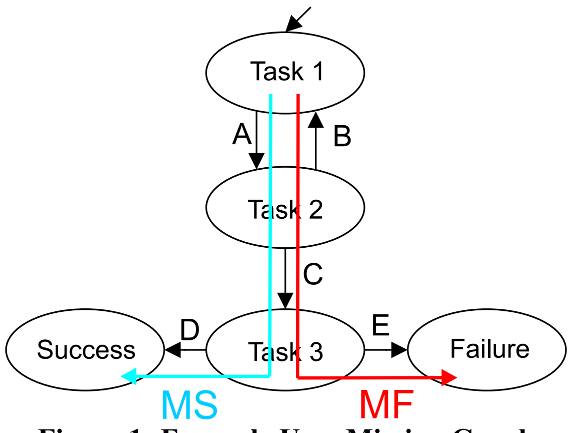

Figure 1: Example User Mission Graph

In this context, missions can be equated with UML scenarios. A scenario describes a way to use a system to accomplish some function [Hsia94]. In UML, scenarios can be represented in sequence diagrams. [Latronico01a] uses formal language constructs to annotate sequence diagrams using the minimum required amount of information. Their work proposes the use of a grammar-based solution to resolve the accuracy of sequence diagrams for complex systems. It is often the case that users must often traverse several sequence diagrams during the course of their interaction with a system. The application of formal language theory to the analysis of sequence diagrams ensures that to we will always know which sequence diagram to 'execute' next from whatever state in the system the user happens to be in when they take action. Their approach allows us to take advantage of the fact that scenarios now, due to their LL(1) grammar representation, have well defined unique pre- and post- conditions. In this approach, missions are 'stitched' together from those scenarios that have pre- and post- conditions that match up, beginning with a start state and finishing with an end state within the mission graph.

3.2 Dependability Bottleneck Detection

We believe that it is feasible to use a min-cut algorithm [West96] on a mission graph to identify mission dependability bottlenecks within a system. For the initial foray into dependability analysis using this technique, we separate the results of applying the min-cut to the graph into only two categories: those states that only have one transition to a subsequent state, and those that have many. It is important to acknowledge that a significant assumption is made here. When applying this algorithm we assume that each transition has equal weight - that is, no transition is either more or less likely to be taken than any other, nor is any transition more or less likely to be broken than any other.

The results of applying the min-cut algorithm to the graph are meant to focus the embedded system designer on the source of potential dependability bottlenecks. For each node in the mission graph, arcs out of that node are considered. This serves an important purpose: if there are many outward arcs that lead to mission success, then that node is less likely to represent a dependability bottleneck. However, if there is only a single arc out of the node to mission success, the user may become "trapped" if a component failure occurs that disables that arc.

So, in the absence of accurate dependability estimates for components, the use of this simple min-cut heuristic can help in understanding relative dependability and in identifying possible dependability "bottlenecks" or likely mission failure modes. It is meant to form a starting point for assessment and comparison and provides a way to identify potential problem spots using the familiar formalization of paths through a directed graph.

After applying the approach to a given user-mission graph, specific information regarding the system should have been discovered. This information can assist the designer in making enlightened choices about how to incorporate workarounds to increase the dependability of the examined system. After all arcs within the graph have been investigated, it may become clear that additional arcs are needed in order to eliminate dependability 'bottlenecks' within the system. If only one arc between nodes along a mission success path exists, the loss of that arc due to partial system failure would result in an inability for the user to complete his or her mission.

4. Application of Approach

4.1 Statement of Hypothesis

In the following sections, we present an example in which we apply the previously detailed approach to an embedded system. We examine an elevator, which is a typical flow-control example of an embedded system. This example will focus on testing our hypothesis, that is: **will using the min-cut algorithm on mission graphs identify mission dependability bottlenecks?**

4.2 Elevator Example

The example mission graph in Figure 2 is a mission-level depiction of a user attempting to reach another floor in a building. Transitions are listed in Table 1. The bulk of the example revolves around boarding an elevator (the details of exiting were included in the experiments, have been omitted in this explanation to help make the example graph more tractable). For this elevator, the elevator car summoning devices and various lanterns are analyzed. Once the user successfully boards the elevator, it is assumed he or she reaches the desired destination. The car summoning buttons and lanterns were sufficient to illustrate our target focus on hardware and software requirements in addition to human interface factors, so other elevator components, such as the drive control, are omitted here for clarity. (It is important to note that, contrary to what one might gather from the literature, elevators are decidedly non-trivial systems. We have simplified this example mission graph to illustrate relevant points.)

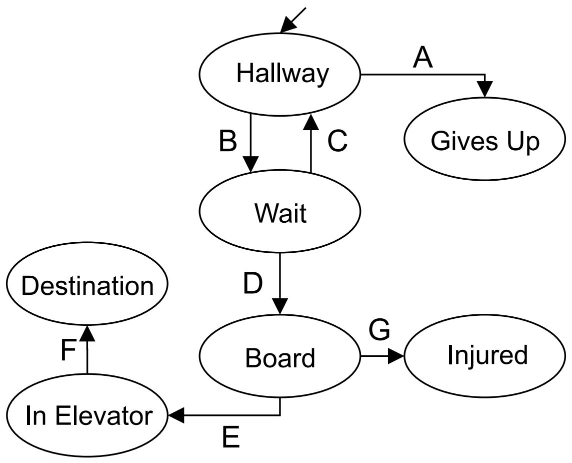

Figure 2: Elevator Example

Arc	Description
A	User times out (frustrated)
B	User presses call button
C	User times out (excessive wait)
D	Doors open / lanterns activate
E	Doors close on user
F	User boarding time elapses
G	Doors close, elevator travels to destination

Table 1: Transitions for elevator example

4.3 Experiment Set-up and Description

In order to accurately model the effects of the experiment, we have captured the behavior of an example elevator and its users in simulation. The example mission graph presented in Figure 2 represents a subset of modeled passenger behaviors within the simulation.

The simulation is in fact a very complex, representative model of the system we are attempting to investigate. The elevator in the simulation is both well specified (via a 26 page requirements documents written by a former elevator architect) and thoroughly implemented (over 10,000 lines of source code comprise the model). Due to the important role that the users of this system play in the dependability assessment, the simulation's model of the passengers is thorough. The elevator user specification alone is several pages that consists of 20 behaviors of each consisting of a single verb-condition pair, i.e. each behavior specifies a single action to be taken by a simulated passenger under a certain condition. (Each requirement statement is of approximately equal complexity) For example: “A Passenger *p* at Floor *f* where a CarLantern is On in their desired direction shall attempt to enter the Car if the Door is sufficiently far open.” The complete passenger specification resulted in over 1,000 lines of code for the passenger model alone.

The simulation used models the conditions a single hoistway elevator in an office building may be subject to over the course of a standard 24 hour business day. The following test scenario descriptions highlight the different passenger traffic patterns that occur:

- LIGHT_TRAFFIC: (extremely early in morning/late at night) When elevator is occupied, it only contains one or two passengers. There is little contention for its use, and elevator remains more than 30% idle.
- MEDIUM_TRAFFIC: (normal daytime use) Moderate contention for the elevator, idle 5% to 10% of the time.
- HEAVY_TRAFFIC: (morning/evening rush hour) Constant contention for elevator, idle 0% of the time and there are always outstanding calls.

4.4 Testing the Hypothesis

We shall now apply the min-cut algorithm to this example and examine the results. In the following sections, we will examine the three sets of results that emerge from this investigation.

4.4.1 Min-Cut Equals One, Bottleneck Exists

In this first test, we will examine an example where applying the min-cut algorithm to the graph correctly identified a dependability bottleneck. In the provided original mission graph of the elevator, we can see that the arc labeled D is the only transition from the Wait to Board state. The following experiment will examine the effects of removing this transition.

Simulation results of users attempting to use the elevator in the face of broken lanterns serve to highlight the issue (cf. Table 2). In this case, users are rigidly specified to only board the elevator if the lantern corresponding to their desired direction of travel is lit.

In all test case scenarios involving these users, NO passengers were successfully delivered because no one boarded the elevator when their lantern was broken.

Performance Decrease Due to Broken Car Lanterns			
	LIGHT_TRAFFIC	MEDIUM_TRAFFIC	HEAVY_TRAFFIC
Users	N/A[1]	N/A[1]	N/A[1]

[1] Passengers were never successfully delivered in this test

Table 2: Results with failed car lanterns

In these tests, the min-cut algorithm was able to successfully uncover a significant dependability bottleneck. When the one transition detected by the algorithm was removed, it proved to be fatal to the operation of the system.

4.4.2 Min-Cut Equals One, Bottleneck Does Not Exist

In this second test, we will examine a situation where the min-cut algorithm detects only one arc connecting a state to its successors, which should result in a dependability bottleneck should this transition become unavailable. However, in this particular example, this turns out not to be the case, and we would like to understand why.

In the provided example elevator mission graph, the min-cut algorithm reports that transition B is the only arc provided that allows the user to transition out of the initial Hallway state. So, it should follow that breaking this arc should prevent the user from achieving their mission successfully. Simulation, however, proved this to not always be the case.

Performance Decrease Due to Broken Elevator Summoning Buttons			
	LIGHT_TRAFFIC	MEDIUM_TRAFFIC	HEAVY_TRAFFIC
Users	N/A[1]	178%	27%

[1] Passengers were not successfully delivered in this test after one hour of wait time

Table 3: Results with failed car summoning buttons

Compared to the baseline, nominally functioning elevator, an elevator with even a few of its hall call buttons malfunctioning suffered significant performance penalties, though in two out of the three test loads users were still able to complete their mission (cf. Table 3). In these tests, users would not realize that the button they were using to call the elevator was malfunctioning and would simply continue to press the button to summon the elevator to their respective floor. In the first test load LIGHT_TRAFFIC, some users remain undelivered to their goal because the elevator remains unaware of their presence and so will never arrive to pick them up. The second test load MEDIUM_TRAFFIC shows a significant performance penalty caused by the broken buttons, since users at floors with broken buttons are only picked up by chance when other users wish to be delivered to their floor. The final case HEAVY_TRAFFIC illustrates the natural resilience the system has under heavy loads. The relatively moderate (as compared to the previous load) performance penalty is due to the fact that the constant traffic of people to all floors results in frequent stops to unload at every floor. The opportunity for passengers to also board helps ameliorate the fact that an elevator summoning button on a particular floor may be broken.

The obvious question to ask is: How did this happen? Based on our mission graph, the results of applying the min-cut algorithm stated that the system should have failed should that one transition become broken. The problem here lies with the fact that there were actually additional scenarios available to the user that were not explicitly represented in the mission graph. Our example mission graph fails to represent that passengers are not necessarily required to press the button to call the elevator to their floor. However, the natural redundancy built into the system allowed it to cope for this design oversight.

We suspect this capability comes from the fact that state machines that are used to design components of a system are in practice richer than the scenarios that generated them. For a complex embedded system that involves a human in the loop, it can be virtually impossible to generate EVERY usage scenario for a system using the Unified Modeling Language because of potentially unanticipated behaviors that users can exhibit. Indeed, in this case, there was a scenario 'built-in' to the system that was not included in the UML design. Helping designers understanding this complex interplay that an embedded system can have with its users is in fact one of the key goals of the mission graph and simulations.

So, in this case, the min-cut algorithm incorrectly identified a transition as a dependability bottleneck within the mission graph. However, this did not necessarily have any ill effects, and in fact helped the designer focus their attention on where to look to better understand a complex interaction between the system and its user.

4.4.3 Min-Cut Equals Many, Bottleneck Does Not Exist

In this final test, we will examine a situation where the min-cut algorithm revealed that many arcs existed between states. In this case, we would like to test that removing one of the arcs should still result in correct operation of the system.

In the original example, arc B is the only transition available from the Hallway to Wait states. However, by specifying "smarter" passengers, we create another transition available to users, which removes the bottleneck of only one arc. Heterogeneous redundancy can be used here to provide an alternate path between the Hallway and Wait states (cf. Figure 3). Most elevators have two buttons per floor in the hallway, one for going up and the second for down. The user's preferable action is to push the button in the desired direction of travel. However, the user can summon the elevator by pushing the button in the opposite direction as well. (Indeed, some impatient users push both!)

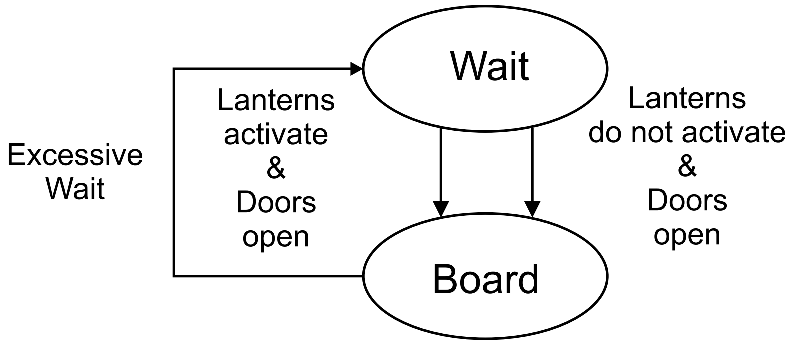
Wait
Excessive Wait
Lanterns activate & Doors open
Lanterns do not activate & Doors open
Board

Figure 3: Additional transition provided by "smart" users who push button in opposite direction

Simulating broken elevator summoning buttons proved correct the hypothesis that the system would still operate correctly (cf. Table 4). Users trained to recognize broken buttons and call the elevator with the other button demonstrated resiliency in the face of the partially broken system. Indeed, this strategy proves to be successful, as an insignificant performance penalty is incurred across all test scenarios. This is most likely due to the elevator's delivery schedule becoming only momentarily confused by riders traveling in unexpected directions.

Performance Decrease Due to Broken Elevator Summoning Buttons			
	LIGHT_TRAFFIC	MEDIUM_TRAFFIC	HEAVY_TRAFFIC
Users	<1%	1%	<1%

Table 4: Results with failed car summoning buttons

This experiment has shown that where the min-cut algorithm reports many arcs are available to transition from one state to the next, the removal of of these arcs still allows the system to function correctly.

5. Workarounds

As discussed in Section 4, discovering the 'single point of failure' scenario is one of the main objectives of the approach. As suggested in the previous section, application of a simple min-cut algorithm to the mission graph enables the designer to see which portions or a system are most likely to be hardest hit by partial system failures. Uncovering these dependability bottlenecks in the system can help increase overall system dependability by indicating to the designer where to best add new features (with the appropriate arcs to the mission graph) in order to give the user a chance to work around partial system failures. We would like to use the refined mission graph concept to analyze the effects that workarounds can have on alleviating dependability bottlenecks that were uncovered using the approach detailed in Section 3. Here, we investigate the benefits that two types of workarounds (Global, or mission, level and state-transition level) that were originally proposed in [Latronico01b] can have on overall system dependability.

5.1 Global Workarounds

In this section we will explore the effects on system dependability of incorporating a workaround at the highest level of a mission graph. Revisiting our original example, we can see the example mission graph (cf. Figure 2) has three distinct paths. One is a mission success - *MS1: (Hallway, Wait, Board, In Elevator, Destination)*. Two are mission failure paths - *MF1: (Hallway, Wait, Hallway, Gives Up)* and *MF2: (Hallway, Wait, Board, Injured)*.

Upon inspection, we observe the fact that there is only one mission success scenario is a dependability bottleneck. In this example, should any part of the elevator malfunction (i.e. any arc along the one mission success path become broken), the user will either be totally unable to achieve their mission successfully (cf. Section 4.4.1) or will experience greatly diminished quality of service (cf. Section 4.4.2). If possible, we would like to alleviate this mission-level dependability bottleneck by giving the user the choice of additional mission success scenarios.

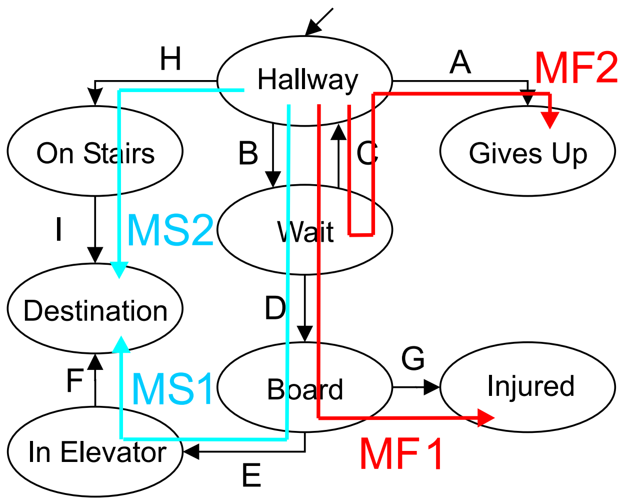

Figure 4: Enumerated missions for elevator including global workaround

Arc	Description
H	User times out (impatient)
I	User arrives at destination (walks)

Table 5: Additional transitions

In order to accomplish this, we can consider this embedded system in its usage context and not as an isolated computer-centric system. In real buildings people may well walk one or two flights of stairs if the elevators are overloaded, especially if they are going down instead of up. This provides a sort of safety valve if elevator system operation is degraded, overloaded or has malfunctioned completely. In order to model this behavior, we can introduce a second mission success scenario denoted *MS2: (Hallway, On Stairs, Destination)* (cf. Figure 4). This second mission success scenario illustrates the concept of a global workaround that may be available to users of embedded systems, and, in this case, helps to eliminate a potentially show-stopping dependability bottleneck at the mission level.

We can simulate the effects of adding another mission success path in our elevator. If we assume that the building that our elevator is in has stairwell access to all floors, we can give our passengers the following additional specific behaviors:

- If after five minutes of waiting for an elevator to arrive, a user has not successfully boarded an elevator, that user will take the stairs.
- It will take each user 30 seconds per floor to reach their destination, plus an initial 15 second penalty to exit the elevator line and travel to the stairs. (The precise times chosen are not critical to the experiment)
- Users will only take the stairs a maximum of two floors up and four floors down.

The results of simulating the above behaviors in our three test scenarios reveal mixed results. In the LIGHT_TRAFFIC scenario, the second mission success path makes no difference at all, since the elevator is operating at below capacity and every user is serviced in a timely fashion. The MEDIUM_TRAFFIC scenario exhibits the odd behavior that a very slight performance penalty is incurred when users are allowed to take the stairs. This result is actually somewhat reasonable, and reflects the fact that some users may attempt to "game" a control flow type of embedded system. In this case, the elevator was operating at its approximate capacity, but an "impatient" user take the stairs instead of waiting for the elevator to arrive. This outlier in the results indicates that taking the stairs for this particular user was the wrong choice, and would in fact have been better off waiting for the elevator to arrive.

Performance Increase Due to Inclusion of Stairs			
	LIGHT_TRAFFIC	MEDIUM_TRAFFIC	HEAVY_TRAFFIC
Average Transit	0%	- <1%	55%
Worst-case Transit	0%	- 4%	46%

Table 6: Results of incorporating global workaround (stairs)

The final test scenario, HEAVY_TRAFFIC, reveals the most significant insights into mission-level dependability bottlenecks. In this case, the elevator's passenger load has clearly exceeded its capacity, and the stairwells provide a pressure release mechanism for the excess capacity. Not only is the worst case delivery time of passengers reduced significantly, but also the average user's travel time is cut nearly in half by the introduction of this alternate success path. These experimental results drive home the point that systems that are performing according to designed specifications can still be "broken" from a practical perspective and therefore undependable. If an elevator is overloaded and the person desiring to use it has to wait longer than they expected, then the system is in some sense undependable.

This relatively basic example helps highlight the usefulness of the mission graph approach to provide a useful basis for comparison of global workarounds. Indeed, the workaround presented here is just one example of the mission graph helping the system designer understand what are potentially complex interactions between the user, the embedded system, and the environment. For example, in a building where stairwells are only for emergency use, it becomes clear that the elevators are then the only path to mission success (perhaps escalators might be installed at heavy traffic floors to improve dependability). Also, in buildings with multiple elevators the graph would have multiple parallel paths to mission success based on elevator replication. It is clear that while the benefit of global workarounds is still somewhat difficult to quantify, they can have a role to play in the alleviating mission-level dependability bottlenecks uncovered by inspection of the mission graph.

5.2 State Transition Level Workarounds

Next, we shall investigate the effectiveness of adding a workaround to an uncovered dependability bottleneck at the state transition level of a mission graph. In our original example mission graph (cf. Figure 2), the human interface is key to providing alternative quality paths from Wait to Board. Generally, users will board an elevator once the doors are open. However, efficiency can be increased by alerting the user when elevator arrival is imminent, and indicating the direction of travel. Up/down lanterns in elevators serve to enhance the system performance component of dependability by preparing users to board and by screening out users who wish to go in the direction opposite from the current elevator travel direction. This optimization is helpful for concurrent users desiring opposite travel directions and for the case where a user is discharged on a floor but is going in the wrong direction to pick up a different user waiting at that floor.

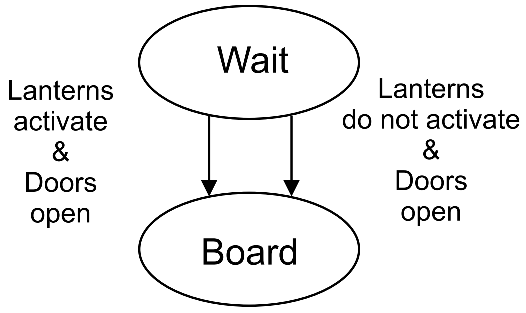
Wait
Lanterns activate & Doors open
Lanterns do not activate & Doors open
Board

Figure 5: Elevator human interface example

In the case of Figure 5 the primary transition of choice is that the user looks for an appropriate hallway lantern and enters when the doors are open. An alternate usage is that the user ignores the lantern (or the lantern does not illuminate) and enters solely because on the doors are open. While this may seem to be a minor difference, it points out that malfunctioning hall lanterns do not put the system out of service, and provides paths that correspond to user flexibility in boarding discussed in the preceding section.

Simulation results of trained and un-trained users attempting to use the elevator in the face of broken lanterns reinforce these points (cf. Table 7). Untrained users are rigidly specified to only board the elevator if the lantern corresponding to their desired direction of travel is lit. In all test case scenarios involving these users, NO passengers were successfully delivered because no one boarded the elevator when their lantern was broken. However, training that allows passengers to board the elevator even if their correct lanterns is not lit helps alleviate this dependability bottleneck. In fact, trained passengers help make this failure completely transparent.

Performance Decrease Due to Broken Car Lanterns			
	LIGHT_TRAFFIC	MEDIUM_TRAFFIC	HEAVY_TRAFFIC
Untrained Users	N/A[1]	N/A[1]	N/A[1]
Trained Users	0%	0%	0%

[1] Passengers were never successfully delivered in this test

Table 7: Results with failed car lanterns

This experiment reveals that even simple training can help users work around potential MAJOR dependability bottlenecks. The second scenario illustrated a case of an all-or-nothing bottleneck that existed when users were unaware of the cause of problem, but was eliminated when (simple) training was applied. In this case, the state transition level workaround proved effective in alleviating the dependability bottleneck that was uncovered in the analysis of the mission graph.

6. Future Work

Obviously, there is much work to be done in the field of system / mission level dependability. The work presented here is meant to be a first step in expanding the dependability assessment of embedded systems to more meaningfully include the potential behaviors of users. The mission graph concept, while useful in the context explored in this paper, must be further refined in order to handle more complex analyses. While the application of the min-cut algorithm to a mission graph has been shown to be useful when considering the existence of one-versus-many transitions, quantifying the benefit of other arc relationships is not yet possible. That is, how can we better understand the benefit of 'many' arcs to subsequent states? Are greater numbers of arcs always better than fewer, and can some notion of 'weighting' be incorporated? Is there a point at which diminishing returns takes effect? Also, it may be possible that annotating transitions within the mission graphs could be useful. Namely, is there a way to integrate the wealth of traditional reliability information into the mission graph in order to glean additional information from the construct? The mission graph concept is one that has been shown to be useful, yet there are research questions that exist that, if investigated, could make it an even more effective tool of understanding the dependability of complex systems.

Also, the workaround concept furthered in this work still still needs refinement before its effectiveness can truly be understood. Our belief remains that users are a part of improving dependability, but quantifying their contribution can become difficult. As demonstrated in this paper, mission graphs can help show where workarounds can be most effective, but exactly how to implement them and analyze their contribution to overall system dependability remains a complex prospect.

6. Conclusion

In this work we have attempted to expand the discussion of dependability assessment for complex systems. It is clear that users of embedded systems are a part of improving dependability; systems can help users work around component failures, just as users can help systems work around partial failures. In order to take into account the user's complex interactions with an embedded system, we proposed the mission graph concept as a technique for uncovering mission dependability bottlenecks. Using this approach on a model of a realistic embedded system proved that it was feasible to uncover dependability bottlenecks within a complex system as well as represent the concept of a "workaround" in a precise manner. Adding workarounds was further explored in the context of alleviating dependability bottlenecks uncovered in a mission graph.

7. Acknowledgments

This work supported by the General Motors Satellite Research Lab at Carnegie Mellon University. The author would also like to thank John DeVale for his invaluable assistance during the early phases of this project, as well as Beth Latronico for her ceaseless support throughout the course of this work.

8. References

[Brehm96] Brehm, E., "System Dependability Assessment Tool." *Proceedings of the Second IEEE International Conference on Engineering of Complex Computer Systems*, 1996, p. 116-119.

[Dreyfuss93] Dreyfuss, H. "The Measure of Man and Woman: Human Factors in Design", *Watson-Guptill*, 1993.

[Gasser86] Gasser, L., "The Integration of Computing and Routine Work." *ACM Transactions on Office Information Systems*, Vol. 4, No. 3, July 1986, p. 205-225.

[Goddard00] Goddard, P., "Software FMEA Techniques", *Proceedings of the 2000 Annual Reliability and Maintainability Symposium*, Jan. 2000, p. 118-123.

[Goswami97] Goswami, K. K., Iyer, R. K., Young, L. "DEPEND: A Simulation-Based Environment for System Level Dependability Analysis." *IEEE Transactions on Computers*, Vol. 46, No. 1, January 1997, p. 60-74

[Hsia94] Hsia, P., et al. Formal Approach to Scenario Analysis. *IEEE Softwar*e, Vol.11, No.2, 1994, p. 33-41.

[John96] John, B. E., Kieras, D. E. "Using GOMS for User Interface Design and Evaluation: Which Technique?"*ACM Transactions on Computer-Human Interaction*, Vol. 3, Issue 4, December 1996, p. 287-319.

[Kalbarczyk99] Karbarczyk, Z., Iyer, R. K., Ries, G. L., Jaqdish, U. P., Lee, M. S., Xiao, Y. "Hierarchical Simulation Approach to Accurate Fault Modeling for System Dependability Evaluation." *IEEE Transactions on Software Engineering*, Vol. 25, No. 5, September/October 1999, p. 619-632.

[Latronico01a] Latronico, E., Koopman, P. "Representing Embedded System Sequence Diagrams as a Formal Language." *Proceedings of the 4th International Conference on UML 2001*, October 2001, p. 302-316.

[Latronico01b] Latronico, E., Martin, C., Koopman, P., “Analyzing Dependability of Embedded Systems from the User Perspective”, *20th IEEE Symposium on Reliable Distributed Systems - Workshop on Reliability in Embedded Systems*, October 2001, p. 32-36.

[Laprie92] Laprie, J.-C. (Ed.), “Dependability: Basic Concepts and Terminology in English, French, German, Italian and Japanese”, *Springer-Verlag*, 1992.

[Manning99] Manning, B. R. M. “Technical Guidelines on Embedded Systems.” *IEE Seminar on Year 2000: A Practical Approach to Medical Devices and Hospital Systems*, June 1999.

[Maxion00] Maxion, R.A., and R.T. Olszewski, “Eliminating Exception Handling Errors with Dependability Cases : A Comparative, Empirical Study”*, IEEE Transactions on Software Engineering*, vol.26, no.9, Sept. 2000, p. 888-906

[Musa96] Musa, J.D., Fuoco, G., Irving, N., Juhlin, B., Kropfl, D., “The Operational Profile,” *Handbook of Software Reliability Engineering*, Michael R. Lyu (ed), McGraw-Hill, New York, 1996. pp 167-216.

[Raghavan01] Raghavan, V. Workshop an Dependable Embedded Systems, SRDS 2001, Invited Talk, slides 5-7.

[Rosenbloom93] Rosenbloom, P. S., Laird, J. E., and Newell, A., “The Soar Papers: Research on Integrated Intelligence”,*MIT Press*, Cambridge, MA, 1993.

[Smidts00] Smidts, C., and M. Li, “Software Engineering Measures for Predicting Software Reliability in Safety Critical Digital Systems”, *U.S. Nuclear Regulatory Commission UMD-RE-200-23*, 2000.

[UML99] Unified Modeling Language Specification, Version 1.4, 2001. Available from the Object Management Group. http://www.omg.com. Accessed December 8, 2001.

[Villemeur92] Villemeur, A., “Reliability, Availability, Maintainability, and Safety Assessment”*,* vol. 1, *John Wiley and Sons,* 1993.

[West96] West, D. B., “Introduction to Graph Theory,” *Prentice Hall*, Upper Saddle River, NJ, 1996, p. 147-150.

[Zemany91] Zemany, P. “Applications for Faster to Reliability and Readiness Analysis of Complex Reconfigurable Fault Tolerant Systems.” *Proceedings of the 10th IEEE/AIAA Digital Avionics Systems Conference*, October 1991, p. 191-186.