Workshop on Reliability in Embedded Systems (in conjunction with SRDS), October 2001.

Analyzing Dependability of Embedded Systems from the User Perspective

Beth Latronico
ECE Department
Carnegie Mellon University
Pittsburgh, PA, USA
beth@cmu.edu

Christopher Martin
ECE Department
Carnegie Mellon University
Pittsburgh, PA
cmartin@andrew.cmu.edu

Philip Koopman
ECE Department
Carnegie Mellon University
Pittsburgh, PA
koopman@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. We suggest evaluating the opportunity for success of a user's mission in terms of flexibility in selecting a series of tasks to accomplish a specified goal. With this user perspective, we create graphs to represent user states and tasks, and explain how some aspects of system dependability can be assessed through standard graph analysis techniques.

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, where [Laprie92] defines dependability as "the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers." (We actually go further and include user/system interactions within the umbrella of computer system dependability as well.)

Many embedded systems have been designed to control 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. 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 behavior can provide insight into the dependability of the system.

In particular, as a step in expanding the discussion of dependability assessment, 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 rather an issue of the richness of functionality available to a user to perform workarounds or alternately accomplish goals in the presence of exceptional events or failures. 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.

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. Dependability in the face of partial system failures can thus be improved by adding additional paths to desired states. (Note that in general this does not necessarily require adding additional hardware, which would of course otherwise tend to increase the number of components subject to failure.) Furthermore, mission failures can be avoided by providing multiple paths away from undesirable states, which in many cases involves intentionally lengthening the path a user must traverse to reach a mission failure state. This graph approach is based on the structural characteristics of a graph of user paths rather than component reliability estimates, and seems broad enough to encompass multiple design disciplines beyond hardware.

Building on the idea of user missions, we have discovered a number of reasons why it is important to consider the user's entire path through the system rather than just individual interactions. 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, there may be alternative (technical or non-technical) workarounds that enhance global dependability. For example, if an elevator is broken or overloaded, the user may take the stairs if only traveling one floor, reducing system load as perceived by other users. As a result, it is important to consider the entire set of possible user interactions (and indeed in many systems the set of possible interactions by concurrent users) when evaluating the dependability of many systems in realistic operating scenarios.

2. APPROACH

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* is a path that achieves a desired goal. A *mission failure* 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.

Using the mission-based approach, dependability can be maximized by maximizing the probability of mission success. However, in the absence of accurate dependability estimates for components, the use of simple heuristics can help in understanding relative dependability and in identifying possible dependability "bottlenecks" or likely mission failure modes:

- Maximize the number of independent paths from any node to a mission success graph node. (In particular, maximize the number of possible arc cuts/node removals that can be tolerated while still leaving a mission success path in place.)
- Minimize the number of paths to a mission failure graph node. (Note that arcs must be provided to account for system component failures that lead to mission failures.)
- Maximize the number of nodes between "typical" mission success paths and graph failure nodes. (This amounts to requiring more steps to be executed to reach a failure than to reach a success, giving more opportunities for the user to recover from a problem.)
- At any particular graph node, provide an arc toward mission success that is more likely to be taken by a user than any arcs toward mission failure, ideally providing more arcs toward success than toward failure.

The general principles presented above can be alternately represented by three questions that we ask when applying our approach:

Given start and end states, how many complete, distinct paths exist between them? All user *missions* for the system must be determined in this step, whether they be of the *success* or *failure* variety. It is in this manner that the user's complete involvement with the system can be determined. In other words, is the likelihood of mission success high (suggesting high dependability), or are there many ways to achieve a mission failure (low dependability)? In a more elaborate form detailed probabilities could be considered when evaluating paths, but this is in general cumbersome and requires information that may not be available at design time.

Given a user state, what transitions exist to subsequent states? For each node in the user 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 most likely does not 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. If most of the arcs lead to mission failure, then that node may present a high risk of mission failure in practice.

Given two mission paths, which portions are identical? This portion of the analysis allows the embedded designer to focus his or her attention on the potential sources of dependability problems within a system. Since mission successes and failures are likely to share common sub-paths though the system, the challenge lies in making undesirable nodes (mission failures) more difficult to reach without compromising the correct operation of the rest of the system (mission successes).

The general idea of this approach is to make it easy for the user to achieve mission success. This can be done by giving multiple chances to divert from failure toward success (long path length to failure) and providing a rich set of possible ways to succeed (multiple paths toward success from any node). Of course, this approach tacitly assumes that the user will not actively strive for failure. Nonetheless, it forms a starting point for assessment and comparisons and provides a way to identify potential problem spots using the familiar formalization of paths through a directed graph.

The example in Section 4 illustrates the use of this approach. In an example embedded system, we show how the principles and questions of this approach can help the system designer evaluate the dependability of a system from the user's perspective, both at the mission and state transition level. While the example is based on statecharts, these principles can be measured using variants of standard graph properties such as reachability and deadlock. As a consequence, this approach should be extensible to nearly all formal graph formats.

3. RELATED WORK

Our work attempts to improve dependability through analyses of user missions with two constraints - dependability of a path is assessed relatively, not absolutely, and hardware and software requirements as well as human interface issues must all be represented. Three research issues emerge here. First, why is dependability so difficult to measure absolutely? Second, what attempts have been made to assess and improve relative dependability? Finally, what other concepts are similar to the 'user mission graph'?

Measuring system-wide dependability presents many challenges. For hardware, techniques like the 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

Table 1 : Transitions for User Mission Graph

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

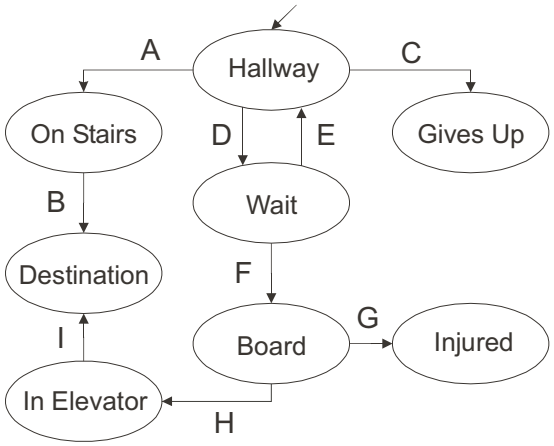

Figure 1: Example User Mission Graph

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. 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. All things considered, absolute dependability prediction will not be a realistic goal for everyday systems for quite a while.

Much work has been done in the area of safety (one component of dependability) involving relative techniques. The most widely known is Fault Tree Analysis (FTA), which takes a list of hazards to be avoided and postulates root causes [Villemeur92]. The FTA method has been traditionally applied to hardware (for example, [Krasich00]), but is generic enough to be used for software as well. The goal of the FTA method is generally to ensure 'no single point of failure', supplying some sort of alternative for every task the system must accomplish. Processes have evolved for reducing errors in software requirements [Leveson00] and human interfaces [Degani98]. Our work is similar to the FTA approach in that the system should provide users with alternative paths through the system, and considers exceptional cases by attempting to maximize the difficulty of reaching undesirable states.

There are a host of graph formats available, although most typically do not center on the user. The most prevalent in software design is the statechart notion, developed by Harel [Harel87] and used by the Unified Modeling Language (UML) community. However, statecharts are typically defined per object in the system, and most object-oriented methodologies do not advocate defining an object for the user. The most closely related approach is part-whole statecharts [Pazzi00], which attempts to model the entire system behavior, in addition to the constituent components. We aim to leverage the analysis powers of graphical formats without requiring numerical dependability estimates.

4. EMBEDDED SYSTEM EXAMPLE

The example elevator graph in Figure 1 is a high 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 have been omitted to simplify the example). For this elevator, the door control mechanisms and various lanterns are analyzed. Once the user successfully boards the elevator, it is assumed he or she reaches the desired destination. The door control 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 literature, elevators are decidedly non-trivial systems. We have simplified this example to illustrate relevant points – our example is not meant to be considered representative of a complete elevator.)

This particular example is based on the format for statecharts originally proposed by Harel [Harel87] and adopted into the UML standard, although the technique should be adaptable to most graph formats. Note that transitions here all involve user volition. For example, the user must push a button to trigger the Hallway Wait transition, and the user must decide to board once the doors are open to activate the Wait Board transition. The user can be in one of eight possible states: Hallway, On Stairs, Wait, Board, In Elevator, Destination, Gives Up or Injured. Hallway is the user's start state, and Destination, Injured and Gives Up are possible end states, as no transitions exit these states. The user's primary mission is to reach the Destination state. A mission success for this example is any complete path leading from Hallway to Destination; a mission failure is any complete path leading from Hallway to Injured.

The following sections illustrate how to glean dependability information from this graph. Section 4.1 illustrates paths that achieve mission success or mission failure. Section 4.2 shows how hardware redundancy can increase dependability. Section 4.3 provides a human interface example. Both of these examples demonstrate the use of multiple paths to increase dependability. Finally, Section 4.4 tackles the problem of undesirable states. Guidelines are provided on how to improve dependability by extending the path length and difficulty to reach undesirable states. This section also discusses dependability trade-offs by examining an example of a safety versus performance conflict, a topic that typically arises during software requirements definition. These examples show how hardware and software requirements combined with human interface components influence the quality and multiplicity of paths through the system.

4.1 Missions

The first aspect of the graph to consider is **how many complete, distinct paths exist from the start state to the end states?** For this approach, two paths A and B will be distinct if the set of states visited in A are different from the set of states visited in B or vice-versa. (If one set is a subset of the other, the paths are still considered distinct.)

This graph has four distinct paths. Two are mission successes - MS1: (Hallway, Wait, Board, In Elevator, Destination) and MS2: (Hallway, On Stairs, Destination). Two are mission failure paths - MF1: (Hallway, Wait, Hallway, Gives Up) and MF2: (Hallway, Wait, Board, Injured).

The embedded system must be considered in its usage context and not as an isolated computer-centric system. In real buildings people may well walk one flight of stairs if the elevators are overloaded, especially if they are going down instead of up. This provides a sort of safety valve if system operation is degraded. The fact that there are two paths to mission success reveals an important property of the real system, even though users may be unlikely to use the stairs.

In this case there are two mission failures, MF1 and MF2, with MF1 being the slightly more preferable scenario (at least no one is injured!). Though this ranking is also fairly subjective, there exists a large body of research on hazard analysis (e.g. [Gowen92]) that could provide for more formal guidelines. Additionally, mission failure path MF1 points out 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 undependable.

As simple as this example seems, it can still provide useful information for comparisons. 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). In buildings with multiple elevators the graph would have multiple parallel paths to mission success based on elevator replication. Because the Wait node involves pressing the Hall Call button, having multiple Hall Call buttons for a group of elevators would provide multiple paths out of Wait to Board, eliminating a potential dependability bottleneck (if there are eight elevators but only one button to call them in the hallway, that button is a dependability bottleneck).

4.2 State Transition Analysis

In this section of the example, we examine two states in the graph and ask **how many transitions exist to subsequent states?** Hardware redundancy considerations are examined in 4.2.1, while human interface effects on dependability are looked at in 4.2.2.

4.2.1 The Hallway → Wait transition

Beyond brute force hardware redundancy, heterogeneous redundancy can be used to provide alternate paths between the Hallway and Wait states (Figure 2). Even inexpensive elevators typically 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!)

An interesting property of this example is that it exploits redundancy installed for providing higher quality service to provided degraded mode service as well. Normally the direction buttons give the elevator a hint about the user's intended destination, and reduce crowding in an elevator by letting the elevator bypass floors if the current direction does not match the desired direction of travel. But if one button is broken the other can be used to ensure that the elevator eventually stops to perform passenger pickup, even if it is an inefficient time to do so. This provides a way to gracefully degrade in the face of a broken component without having to resort to brute force redundancy (although that could be used as well if desired and affordable).

4.2.2 The Waiting → Boarding transition

The human interface is key to providing alternative quality paths from Wait to Board (Figure 3). 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. Adding an up/down lantern to the elevator enhances 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.

In the case of Figure 3 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.

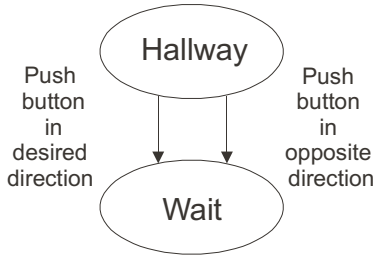

Figure 2: Hardware Redundancy Example

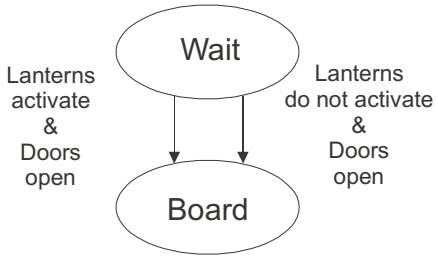

Figure 3: Human Interface Example

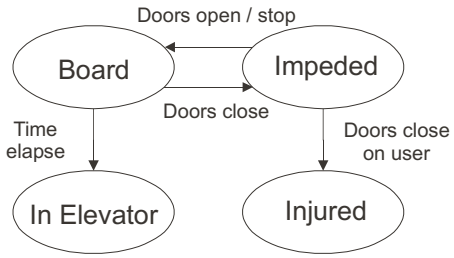

Figure 4: Impeded as an Added State

4.3 Identification of Common Sub-sequences

Here, we examine **what portions of MS1 and MF2 are identical?** Graph sub-sequences that are shared between success and failure paths are inherently risky and bear special consideration in terms of evaluating the number and nature of diversion points. In MS1, the user successfully boards the elevator and is delivered to his or her destination, while in MF2 the user is injured or upset by being crushed or even bumped by the doors while boarding the elevator.

The transition from Boarding to Injured is a prime example of dependability trade-offs involved in designing embedded systems. Ideally, the safest system would never allow the user to enter the Injured state. One way to assure this is to make sure the transition condition - Doors Close - is always false. However, the elevator requires the doors to be closed in order to proceed to the destination (see the transition between In Elevator and Destination). Therefore, an elevator with doors that always remain open would have zero dependability because it has zero utility (never moves), although it would presumably be quite safe. In this case, altering the system configuration can provide a useful intermediate situation between completely safe and completely useful. Most elevators have a door reversal sensor which serves to detect if a person occupies the door aperture when the doors attempt to close. This provides an additional state, Impeded, between Board and Injured (Figure 4).

The software requirements for the door control involve many safety vs. performance considerations. First, the mission success elevator scenario and the mission failure elevator scenario share a common path subset - (Hallway, Wait, Board). Therefore, the difficulty of reaching Wait and Board should not be increased, as this would degrade performance in the mission success scenario. Therefore, the path (Board, Impeded, Injured) should be made as unlikely to occur as possible, and the path (Impeded, Board) should be made easy to traverse. Several ways exist to do this, such as making sure the doors do not close too quickly (reducing the chance of actual bodily contact), adding an additional transition from Impeded to Hallway via a recorded voice announcing that it is too late to board (as is commonly implemented on automated light rail vehicles), and providing more sophisticated ways to detect a door obstruction without making physical contact (which would show up as additional arcs on a graph of the system rather than the user graph).

A number of design questions arise following these observations. How long should the doors remain open? Should the doors open fully upon a reversal, stop in place, or continue to close despite the impeded passenger? How many times should the doors be allowed to re-open on the same floor? These decisions are difficult to make using the user mission graph only. Rather, the nature of the users and environment will guide the choice selected.

5. CONCLUSIONS

We have shown how the user mission graph approach can provide a relative assessment of dependability of embedded systems in situations involving users who cooperate in workarounds in the face of component failures. This approach takes into account hardware, software requirements, and human interface contributions to dependability. Dependability can be enhanced by seeking to modify some formal properties of graphs, such as the number of paths between two states, the number of transitions to neighboring states, and the character of paths to undesirable states.

Of course these techniques cannot solve all problems of dependable system design. In fact, they are just a starting point to expand thinking beyond traditional computer-system-centric dependability analysis. However, we believe that even the exercise of creating such graphs will help designers think through and identify dependability issues and points that bear special scrutiny with their systems. Additionally, we hypothesize that metrics based on these heuristics will be helpful in comparing alternate system designs with respect to some aspects of likely dependability, or at least with respect to brittleness in the face of partial system failure even in scenarios with flexible users.

There are many possible avenues for future work. Among them are incorporating a more rigorous notion of ranking and/or probabilities in the graph arcs, attaining better understanding of how to insert and delete arcs in response to component failures, and providing more precise formulations of algorithms suitable for use in automated tools that start from a machine-readable execution of a system such as statecharts entered into a UML-based design tool. Nonetheless we think that the general ideas presented will be useful to system designers and provide a fresh perspective on possible dependability assessment techniques in the face of the reality that traditional component-wise reliability numbers are both difficult to attain and not totally representative of all important aspects of dependability.

6. ACKNOWLEDGEMENTS

This research is supported by the General Motors Satellite Research Lab at Carnegie Mellon University and the United States Department of Defense (NDSEG/ONR).

7. REFERENCES

[Degani98] Degani, A., A. Kirlik, "Describing the Design Contributors to Mode Error", *Proceedings of the Fourth Annual Symposium on Human Interaction with Complex Systems*, March 1998, p. 112-115.

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

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

[Gowen92] Gowen, L., J. Collofello, and F. Calliss, "Preliminary Hazard Analysis for Safety-Critical Software Systems", *Proceedings of the Eleventh Annual International Conference on Computers and Communications*, Apr. 1992, p. 501-508.

[Harel87] Harel, D., "Statecharts: A Visual Formalism for Complex Systems", *Science of Computer Programming*, vol.8, no.3, June 1987, p. 231-274

[Krasich00] Krasich, M., "Use of Fault Tree Analysis for Evaluation of System-Reliability Improvements in Design Phase", *Proceedings of the 2000 Annual Reliability and Maintainability Symposium*, Jan. 2000, p. 1-7.

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

[Leveson00] Leveson, N., "System Safety in Computer - Controlled Automotive Systems", *SAE Congress*, March, 2000.

[Pazzi00] Pazzi, L., "Part-Whole Statecharts for the Explicit Representation of Compound Behaviors", *Proceedings of UML 2000, 3rd International Conference on the Unified Modeling Language,* Oct. 2000, p. 541-555.

[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.

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