Architecture Level Qualitative Reasoning About Safety

Tacksoo Im, Soujanya Vullam and John D. McGregor
Clemson University
School of Computing
Clemson, USA
Email: {tim,svullam,johnmc}@cs.clemson.edu

Abstract

Architects use a variety of techniques to evaluate designs for the desired levels of specific quality attributes. Reasoning frameworks are used to guide architecture definition by determining the extent to which a software architecture satisfies its quality requirements. There is much work on reasoning about quality attributes such as performance and modifiability but there has been little work on defining a reasoning framework for safety.

We present a safety reasoning framework that is based on the observation that safety hazards lead to accidents when certain other quality requirements of the system are not satisfied. This naturally leads to the use of reasoning frameworks for these other attributes as a means to indirectly reason about safety. We present our technique for using standard safety engineering activities to provide the data needed for the safety reasoning framework. A risk-based qualitative reasoning technique is used to combine the results from the set of reasoning frameworks and make a judgment on satisfaction of safety requirements by the architecture.

1. Introduction

Safety is an essential element in the design of many dependable systems. Safety is defined as the freedom from unacceptable risk of an event that results in loss of life, injury, or property damage [20]. What is unacceptable varies from one domain to another. Software cannot cause direct harm but software embedded in automobiles, aircraft, robots, and even elevators has the potential for causing accidents, which result in injury or death. These accidents are the result of hazards, events that occur outside the control of the system [20].

Architects design an architecture to meet safety requirements by introducing structures and relationships among those structures that reduce the likelihood of a hazardous event occurring and that mitigate the impact of those accidents that do occur [1]. The problem is there is no inherent safety issue in an architectural structure. The safety issues arise from the context in which the software is used.

The accidents that safety design is intended to prevent are often the result of failures of the system to meet specific quality requirements. For example, a failure to meet a performance goal may cause a synchronization problem, which results in loss of control at a critical moment. Even qualities concerning the static structure of the architecture, such as maintainability, can affect the safety of the system by reducing the rate at which defects are removed or increasing the rate at which defects are injected during maintenance. Our work on safety design is based on the idea that safety requirements are closely related to other quality requirements.

At the earliest stages of safety analysis, system requirements and the context in which the system will be deployed are analyzed to identify hazards [9] [22]. Certainly some accidents result from hardware failures over which the software has no control, but many accidents occur because of software failures. A safety analysis of the software examines the functional and non-functional software requirements. These requirements are analyzed to associate individual requirements with specific hazards.

The architect needs a method by which a proposed modification of the architecture can be compared to the original architecture to determine which will result in safer systems. This can be done by examining the affect of the modification on the safety hazards. The modified architecture cannot have increased the risk of certain hazards leading to accidents nor can it have increased the impact of those accidents that do occur.

In this paper we present a technique for reasoning about safety during architecture definition that specifies the required analyses. This technique is presented in the context of our on-going work on the design of dependable systems [13]. Dependability is defined differently in different domains. For purposes of this paper we will consider domains in which dependability is considered to be a combination of reliability, availability, confidentiality, integrity, maintainability, and safety [3].

The main contribution of this paper is a qualitative reasoning framework for safety. In particular we present a novel analytic theory for estimating the safety of systems built from the architecture. This theory relates the safety of a system to the system's ability to satisfy its other quality

requirements. The theory uses traditional safety analysis techniques as a starting point and traces back to the causes of potential safety hazards.

In the next section we will provide some background on safety analysis and reasoning frameworks including our efforts with respect to reliability and confidentiality. We then provide a description of a safety reasoning framework. We apply the framework to an on-going case study. We conclude by describing the integration of safety analysis into the larger dependability reasoning framework and comparing our work to that of others.

2. Background

In this section we present a brief introduction to two topics used later in the paper.

2.1. Hazards

As previously stated, a hazard is an event that might happen and if it does injury or death may result. This fits the definition of a risk and gives a good analogy for thinking about safety hazards. According to the Software Engineering Institute (SEI), risk is the possibility of suffering a loss. That loss can result from varied causes. In the study of safety, risk plays a pivotal role as safety is directly related to risk. The SEI's risk definition explains risk precisely, it includes [2]:

- A description of the current conditions that may lead to the loss.
- A description of the loss.

Managing risks can be broadly categorized into two types of activities: *risk evaluation* and *risk mitigation*. Risk evaluation of safety-related events - hazards - is evaluating the severity of the injury that would result. The types of losses are considered while enlisting the safety requirements [7]. The risk evaluation phase starts with finding the possible losses and probability of occurrence of the losses individually which is similar to the identification of hazards and hazard probabilities. Thus the risk evaluation phase in the context of safety is instrumental in determining the safety requirements of the system.

Hazards are identified using one of several techniques, such as Functional Hazard Analysis[25], [8]. This technique evaluates a representation of a system and its operational procedures to determine if humans or environment will be exposed to hazards and take apt measures to prevent loss [24]. Hazards can be categorized based on the criticality level, which reflects the consequences of the hazard and the intensity of loss. Based on the criticality level, hazards that are of high criticality level (catastrophic) can be analyzed first to generate scenarios for reasoning frameworks, when evaluating quality attributes.

Risk mitigation has two approaches, decreasing the probability of hazard occurrence and decreasing the intensity of loss caused by hazards. Modifying the architecture to increase the level of a quality reduces the probability of occurence of certain hazards. Reducing the impact when a hazard becomes an actual incident is beyond our technique.

2.2. Reasoning frameworks

A reasoning framework is used by an architect to evaluate an architecture in terms of a specific quality. "A reasoning framework provides the capability to reason about quality attribute behavior based on analytic theory [5]." A reasoning framework defines six elements: problem description, analytic theory, analytic constraints, model representation, interpretation, and evaluation procedure. Problem description gives the actual set of quality requirements for the particular quality attribute which is being evaluated.

Analytic theory is the basic means of computing a measure of the quality. *Analytic constraints* are limitations on the analytic theory. *Model representation* is the description of techniques used to represent the model of system. The techniques are derived from the analytic theory represented in a form compatible with the evaluation procedure. *Interpretation* is the method by which this representation is generated from architecture description. *Evaluation procedure* is the formula by which quality measurements are calculated from the model representation [5]. Therefore a reasoning framework takes quality attribute requirements as input and gives estimates of quality values for a system as output.

"Scenarios are brief narratives of expected or anticipated use of a system from both development and end-user viewpoints [17]." To use a reasoning framework in architecture design, quality scenarios describe how a use of the system affects a specific quality attribute. The scenarios drive the identification of responsibilities which are the elements in the architecture representation.[5].

Architecture Assistant (ArchE) is a tool developed by the SEI to support the evaluation of architectures against reasoning frameworks [4]. In ArchE the architecture is represented by a set of responsibilities that are related to one another by relationships. The responsibilities are derived from the scenarios using responsibility-driven design. The responsibilities are indicative of the system and remain fixed for all qualities. The relationships are specific to a quality so there is a set of relationships for each quality. Our technique will take advantage of these modeling elements.

3. Safety Reasoning Framework

Accidents are caused by failures (hazards that have been realized) and there is an inherent nature to each failure [26]. A failure of omission can indicate an availability problem, a failure to produce a correct value can indicate a reliability problem, and timing issues are indicative of performance problems [26]. These failures become a safety problem if

Function	Failure Condition	Phase	Effect	Class	Verification
system function	what causes the function to fail	when	what effect on the system	severity of the failure	documentation of the failure

Table 1. Results of Function Hazard Analysis

any such failure results in a loss of human life, injury or property damage. Even failures that result from usability problems can lead to safety problems [15] [19].

Any quality attribute that can increase the probability of a hazard resulting in an accident should be part of the safety analysis. For example, the safety of a software-based fire alarm system is largely dependent on how reliably the fire is detected and processed by the system. This is an example where a failure to satisfy a reliability requirement becomes a safety problem.

Failure to satisfy an availability requirement can also cause a safety problem in many cases. A software based monitoring and alarming system can cause a safety problem when it becomes unavailable. Maintainability was shown to be a problem in the Boeing 777 accident near Perth. In that incident, a failure to properly maintain a failed accelerometer resulted in the output of contradictory readings to the pilot [14]. Usability issues where the software causes user distraction are also known to cause safety problems.

We hypothesize that safety can be analyzed using a set of reasoning frameworks depending on the nature of the hazards and then the aggregate result from the reasoning frameworks can be used to characterize the level of safety in the architecture. This is because the failures that cause many safety problems can each be classified into a failure to satisfy a specific quality requirement. In this section, we present a safety reasoning framework which leverages this observation while codifying some of the existing techniques and approaches. We first outline the actions in a traditional safety analysis and then we will use the canonical outline, containing six sections, for describing reasoning frameworks to describe our framework.

3.1. Initial Safety Analyses

Our reasoning framework builds on the information collected in a traditional system safety analysis.

Functional Hazard Analysis

Functional Hazard Analysis (FHA) identifies hazards by examining the potential effects of functional failures[25]. Those that can result in injury or death are safety hazards and are the input to the Fault Tree Analysis. The potential effect of a functional failure will also help determine the root cause of the failure.

The information from the FHA is captured in a form such as shown in Table1.

Fault Tree Analysis

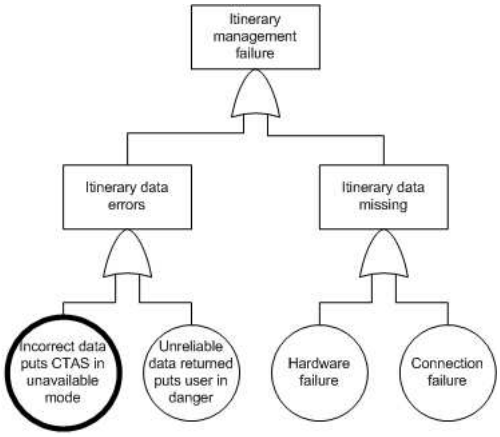

Figure 1. A Fault Tree

Fault Tree Analysis (FTA) identifies the root causes of undesired events including those events that result in injury or death. For our purposes only those hazards that are identified as safety critical during the Functional Hazard Analysis are subjected to FTA. FTA uses a graphical symbology that identifies different types of events and linkages.

In Figure1 node (1) identifies the fault, which is a failure in this case, being analyzed. The “inhibit” node (2) represents an event that must also occur in addition to the ones being further analyzed. In our case this would include hardware events and in some cases certain software events that are excluded from the analysis. Node (3) is an OR node and supports modeling cases where one of two possibilities exist. Node (4) is an AND node that indicates that both basic event (5) and basic event (6) must occur for the failure to occur.

The Fault Trees for the safety critical hazards are a basic input into our reasoning framework. In our reasoning framework we will be looking for basic events, those circles at the bottom of the diagram, that signify quality requirements that are not met.

3.2. Problem Description

Safety is defined as the freedom from accidents which result in a loss of human life, injury, property damage, etc [21]. These accidents occur as a result of failures during system execution, but may be caused by either dynamic or static faults in the architecture. Whether a hazard results in an accident is outside the control of the system, but failure to satisfy certain quality requirements increases the likelihood of an accident. Table 2 illustrates some of the failures that can affect the probability of a hazard resulting in an accident.

The input to the safety reasoning framework is the output from the initial safety analysis described in section 3.1. This input is a set of safety hazards. A safety scenario

Quality Attribute	Hazard-affecting events
Performance	A hard deadline is missed
Reliability	Incorrect output is generated
Availability	A system element that was supposed to be in service is not ready for use when needed
Confidentiality	Information of highly sensitive nature is visible to unauthorized persons
Integrity	Highly sensitive data is modified by an unauthorized person
Usability	A user is distracted by a non-intuitive user interface
Maintainability	A critical fault is repaired incorrectly

Table 2. Quality Attributes Used to Analyze Safety

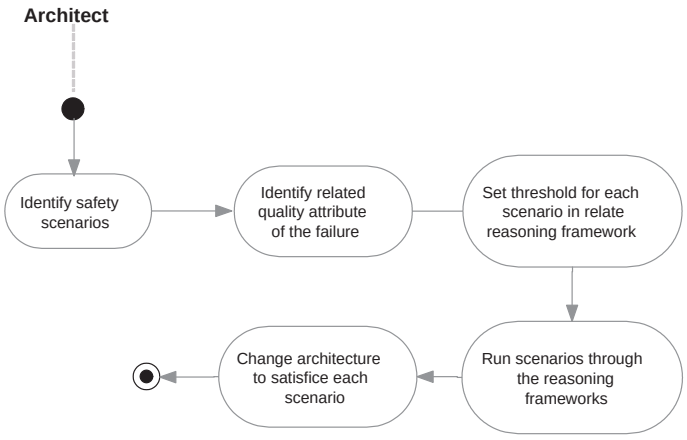

Figure 2. Safety Analysis

is constructed for each of the identified safety hazards. A system safety requirement sets an expected tolerance for each hazard.

The general scenario for the safety reasoning framework, which defines the form of a safety scenario, contains the information described in Table 3.

Stimulus:	Any stimulus that addresses a quality requirement that can increase the likelihood that a hazard will cause an accident
Source of the stimulus:	Any actor of the system
Environment:	The stimulus is introduced during the execution of the system.
Artifact:	System and potentially some real-world accident that results in injury or death
Response:	The system fails and the user is harmed in some way.
Response measure:	What is the impact of the failure?

Table 3. General Scenario for Safety

3.3. Analytic Theory

The analytic theory for the safety reasoning framework involves a set of transformations between a set of analytic theories. As stated previously, a safety scenario is classified as being related to availability, reliability, or some other quality. The safety scenario is transformed into a scenario in the appropriate reasoning framework(s), as shown in Figure 3.

Additional information may be required to complete the transformation to a specific reasoning framework. For example, the performance reasoning framework needs a maximum latency value. The target reasoning framework is applied and the scenario is evaluated. As is usually the case, the reasoning framework computes an estimate for the quality using the current architecture and determines whether that value satisfies the scenario requirements or not. That judgment is returned as input to the safety reasoning framework.

The safety reasoning framework combines the results for all qualities for all scenarios using a typical risk weighting scheme. The scenarios are prioritized and the architect checks that high priority scenarios have been satisficed.[1] The prioritization is similar to the technique used for any risk-based analysis. The priority is based on the cost if an accident occurs and the probability of the hazard causing an accident.

Our basic premise is that the failure to satisfice certain quality requirements can potentially degrade system safety. The individual reasoning framework for a quality attribute can contribute to a safety analysis since safety is an emergent quality attribute of the system. To analyze a safety scenario using these reasoning frameworks, the potential hazards in the system's environment are identified using FHA. A further analysis identifies the quality requirements that affect the system's ability to meet its safety requirements using FTA. For each safety scenario, the quality attributes that participate in that scenario are identified. An appropriate scenario is constructed for that quality and submitted to the corresponding reasoning framework. That framework determines whether the scenario is satisficed by the current architecture. A boolean value (satisficed/not satisficed) is returned to the safety reasoning framework. The safety reasoning framework considers all results from all the constituent reasoning frameworks for a particular safety scenario to determine whether the safety scenario is satisficed.

3.4. Analytic Constraints

The main constraint of the safety reasoning framework is the focus on system qualities. There are failures that are

1. We use the term "satisfice" since the outcome of the analysis does not tell us the optimal solution to satisfy the safety requirements but instead it tells us that the scenarios have reached their minimum allowed thresholds.

due to functional problems that are not the result of a failure to satisfy a quality requirement[21]. For each scenario, the reasoning framework returns a result that shows whether the architecture is satisficing it or not based on the network of responsibilities that are part of the system.

3.5. Model Representation

The model representation for the safety reasoning framework is a mapping that transforms the individual safety scenarios into scenario(s) in the appropriate reasoning framework(s). It is the set of arrows shown in Figure3 that originate from the safety scenario. Constructing the mapping will require the architect to provide the additional information that is needed for the target reasoning framework that is not used by the safety reasoning framework.

The resulting model is a network of reasoning frameworks that are joined together by transformations and return of results. This is made possible by considering each reasoning framework to be a view on a single architecture. In tools such as ArchE the architecture is a network of responsibilities. Different reasoning frameworks add their own linkages between responsibilities, but the complete set of responsibilities and links are a representation of the architecture.

The architect makes changes to the network as a result of the feedback from all of the reasoning frameworks. The individual reasoning frameworks re-evaluate the status of each scenario and report back an update on whether each scenario is satisficed by the new architecture.

3.6. Interpretation

The hazards obtained by FHA are subjected to FTA to identify their respective causes and the quality attributes behind them. The hazards are prioritized based on criticality as mentioned in section 2. These hazards are associated with their respective quality attributes (causes). The safety scenarios associated with each hazard are mapped to the scenarios of the quality attributes related to the corresponding hazards. Figure 3 gives a conceptual view of the mapping. In this example, the safety scenario is mapped to performance and availability scenarios.

3.7. Evaluation Procedure

The quality attribute scenarios that are derived from the safety scenarios are each evaluated in the appropriate reasoning framework. The result of that evaluation is locally judged to have satisficed its requirement. The data available to the safety reasoning framework are boolean, pass/fail, judgements for each scenario.

We exploit the risk-based nature of hazards to rank the hazards identified in the initial safety analysis. This approach

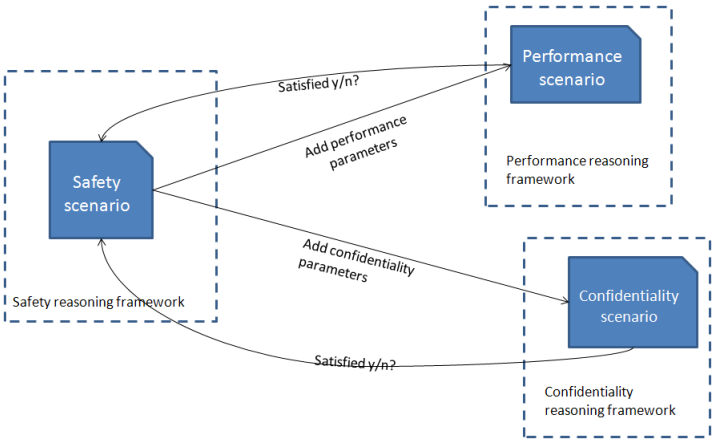

Figure 3. Transformations between theories

uses the probability that the hazard will occur, the probability that the hazard will lead to an accident if it does occur, and the cost if there is an accident to determine the ranking. The ranking is an ordered list of satisficed and not satisficed scenarios. There is no intention to derive a single score from the set of safety scenarios.

The ranking of the scenarios gives the architect an order in which to attack the scenarios that have not satisficed their requirements. Suggestions regarding how to improve the safety aspect of the architecture can be made to the architect using tools that draw on the reasoning frameworks for the derived scenarios.

The qualitative aspect of the safety reasoning framework shows up when we think of which transformations to apply. Certain tactics will improve some scenarios and diminish others (based on general trade-offs) [12] and by applying these tactics based on that generally observation (a qualitative method) we may be able to satisfice all the safety scenarios (or not).

4. Example Application of Technique

We will illustrate our technique using the Clemson Travel Assistance System (CTAS) [24][23]. It is an itinerary planning system that plans routes and modes of transportation from one place to another. CTAS can be executed on various platforms including hand-held devices. The hand-held devices add GPS and use wireless connections to check schedules, make reservations, and check the status of various modes of transportation. Developers can develop plug-ins that integrate appropriate technologies to provide various services through CTAS.

The quality attribute requirements of CTAS include performance, security, functionality, reliability, usability, safety, efficiency, maintainability and portability. A *Quality Attribute Workshop* found that performance and modifiability

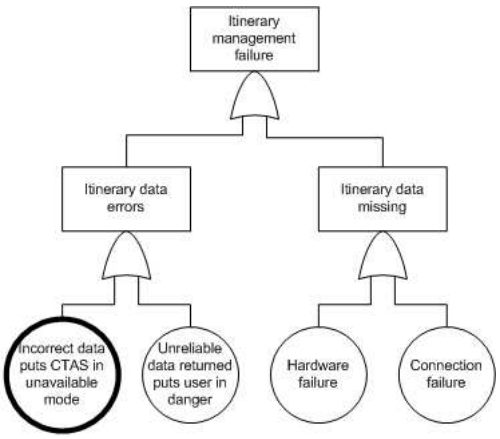

Figure 4. Fault Tree for Failure in Itinerary Management

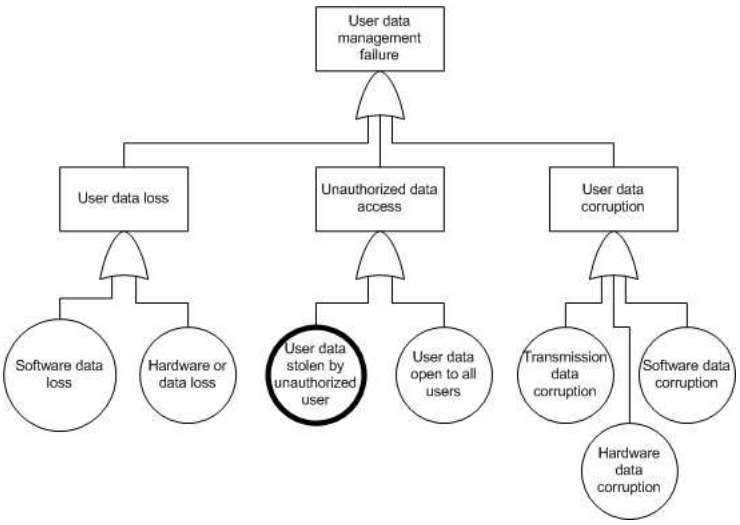

Figure 5. Fault Tree for Failure in User Data Management

(a sub-attribute of maintainability) are the two most important quality attributes of the system [24]. With these drivers in mind, the initial architecture representation of CTAS, showin in Figure 7 will be used to apply our technique of safety analysis and transformation. Due to space limitations we show only the most pertinent results.

4.1. Perform Initial Safety Analyses

The first step is to perform a functional hazard analysis on CTAS. Domain experts considered the context in which CTAS operates and identified potential failures. Table 4 shows a portion of the results of the FHA on CTAS.

The FHA reveals that we have hazards associated with the itinerary management component, the user data management component and the user interface component that can lead to safety problems in CTAS. The next step in our analysis is to use each hazard as the starting node of a fault tree and analyze the root cause those the hazard. Figures 4,5,6 illustrate the fault trees for the respective hazards.

In the FTA diagrams, the bold circles indicate the faults that are related to the quality attributes of CTAS. The next step is to turn these faults into safety scenarios that can be used as input to the safety reasoning framework.

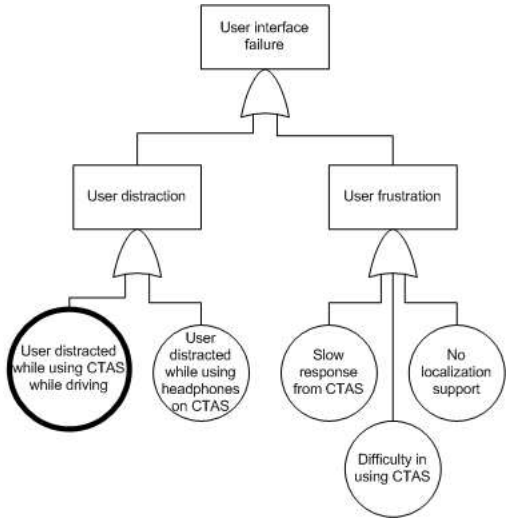

Figure 6. Fault Tree for Failure in User Interface

4.2. Translate into Safety Scenarios

The faults from the FTA that affect certain of the quality attributes are turned into safety scenarios. Using the general scenario for safety we construct the concrete scenarios. We will consider only two scenarios: the safety scenarios that are mapped to confidentiality and usability. In addition to our reasoning framework for confidentiality, we use a usability reasoning framework described in [6]. The two scenarios are:

- Confidentiality: Sensitive information such as the private information (address, telephone, social security number) of the user could be used to harm the user. Private information is entered in CTAS by the user and an unauthorized user tries to access the data.
- Usability: CTAS can be used while driving. In some situations the driver may be distracted from the road by hard to understand controls. The driver tries to enter a new destination address while driving and has difficulty dertiming which field the cursor is in.

The concrete scenarios in the form of the general scenario are illustrated in Tables 5 and 6.

Function	Failure Condition	Phase	Effect	Class	Verification
Itinerary Management	Incorrect or obsolete geographic information for the user	run-time	Puts CTAS in a mode that is unavailable	High Criticality	User gets into a dangerous situation.
User Data Management	Disclosure of private data to unauthorized user	run-time	Disclosure of data is undetectable to the system	Medium Criticality	User is harmed by the abuse of the disclosed private data
User Interface	User interface causes user distraction problem	run-time	Usability problems are undetectable to the system	Medium Criticality	User gets into an accident because of the user distraction problem

Table 4. Results of Function Hazard Analysis on CTAS

Stimulus:	Access of confidential data
Source of the Stimulus:	User who is not authorized
Environment:	Normal mode
Artifact:	Personal Data of user
Response:	End user's personal data is accessed
Response Measure:	The amount of sensitive data the unauthorized user is able to access.

Table 5. Safety Scenario resulting from Confidentiality

Stimulus:	End user attempts to enter a new destination
Source of the Stimulus:	End user
Environment:	In a moving vehicle, CTAS is running on a hand-held device
Artifact:	System
Response:	End user looks at CTAS device and loses control
Response Measure:	The time the user must spend looking at CTAS to correctly enter data.

Table 6. Safety Scenario resulting from Usability

4.3. Apply the Constituent Reasoning Frameworks

Each of the safety scenarios is mapped to the specific attribute reasoning frameworks, as illustrated in Figure 3. For the scenarios in Tables 5 and 6 they are mapped to the confidentiality and usability frameworks respectively. We show the process only for the confidentiality scenario. The safety scenario in Table 5 is mapped to the confidentiality scenario in Table 7. The architect provides the Response Measure goal for the new scenario.

Table 7 shows the concrete confidentiality scenario that results from mapping the safety scenario to the confidentiality reasoning framework. The actual Response Measure when evaluated by the Confidentiality reasoning framework is false for the current CTAS architecture. That value is returned to the Safety reasoning framework as the result of the Confidentiality reasoning framework.

Stimulus:	Attempt to read CTAS user's social security number
Source of the Stimulus:	Unauthorized person
Environment:	End user's hand-held CTAS device
Artifact:	CTAS database
Response:	Unauthorized person spoofs the system and accesses the database
Response Measure:	false - we do not want teh data to be accessed

Table 7. Confidentiality Scenario

4.4. Apply the Safety Reasoning Framework

In this step, the architect determines to what extent the safety scenarios have been satisficed by the current architecture. When all of the safety scenarios have been evaluated in the constituent reasoning frameworks the architect has a set of true/false results. In our analysis, the confidentiality and the usability scenario were not satisfied by the current architecture. The architecture needs to be changed to satisfy the two scenarios under consideration.

The architect uses the ranking among the quality attributes, established early in the architecture definition process, to identify the order in which transformations are going to be applied to the architecture. In our example, the architect has previously determined that confidentiality takes precedence over usability.

Fortunately, there were no conflicts between the transformations that are to be applied in order to improve the safety of CTAS. The orginal architecture, as a set of responsibilities, is shown in Figure 7 and the transformed architecture is shown in Figure 8.

In order to improve the confidentiality scenario, an *authentication* responsibility was added. These transformations in the architecture are not conflicting; however, the realization of the authentication responsibility did reduce the usability of teh system. A *voice output* responsibility was added to improve the usability scenario. The revised architecture now satisfices the two safety requirements we

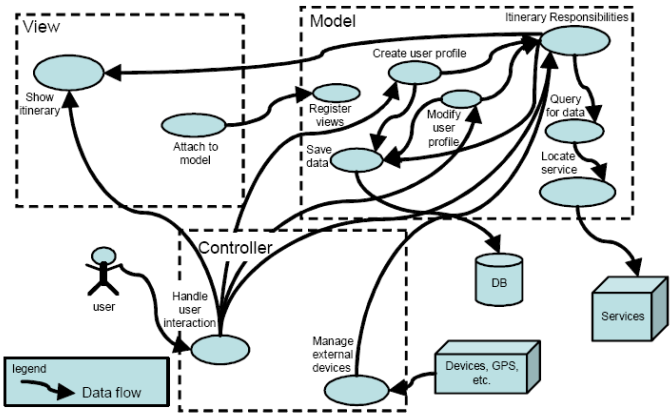

Figure 7. CTAS Architecture before Transformations

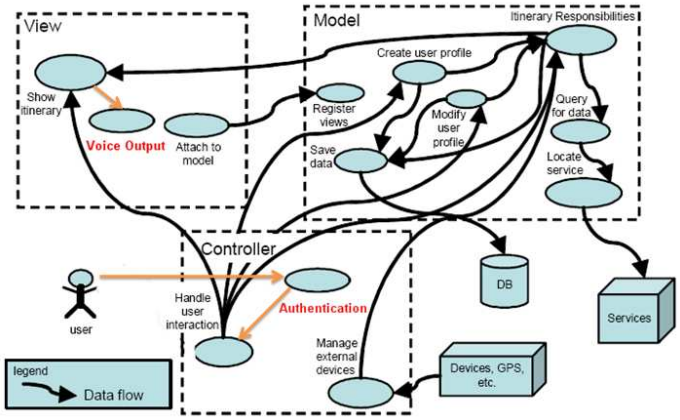

Figure 8. CTAS Architecture after Transformations

are investigating.

Table 8 shows the safety analysis as part of a larger quality analysis of the system. By applying our safety analysis technique we have determined how the current architecture of CTAS has satisficed our safety requirements as part of the overall quality requirements of CTAS.

Table 8. Current system quality profile

Attribute	Decision
Functionality	meets goal
Reliability	meets goal
Safety	meets goal
Efficiency	exceeds goal
Maintainability	meets goal
Portability	meets goal
system	meets requirements

5. Discussion of Technique

The output from our technique is an indication of whether the safety quality requirement as defined in a specific quality scenario is satisfied or not. The reasoning framework will be applied to the complete set of safety requirements and the result is a set of requirements, some of which are satisficed and some are not. Although risk-based approaches could be used to reduce this set of pass/fail values, the typical approach is for the architect to use the set of values to guide modifications to the architecture.

The architect exploits the mappings from safety to other quality attributes to modify the architecture. By examining each mapping, seeing the value of the derived attribute in that context and then looking at the advice given by that framework to improve the level of that attribute, the architect can make a change in the architecture that addresses the safety requirement. After the architect makes a set of changes the safety framework is reapplied to determine which safety requirements are now satisficed.

Currently the safety framework is not automated. Some of the reasoning frameworks for derived attributes are automated and use rules to give advice based on the patterns of relationships for that attribute. We are using the latest version of ArchE, which supports plug and play of reasoning frameworks, to implement these frameworks.

We do not claim that every hazard leads to a quality attribute failure. Hardware faults and failures due to other environmental factors are not addressed by this technique. The safety reasoning framework is useful where the architect can identify a derived attribute within the deifnition of teh safety requirement.

6. Related Work

At the architectural level, software safety has been analyzed using various methods. Techniques such as Failure Propagation and Transformation Notation (FPTN) [9] and Component Fault Trees (CFTs) [16] are used to evaluate whether a software architecture can meet the safety requirements. (The safety requirements are described in terms of hazards and the allowed probabilities of its occurance.) These approaches are based on modular failure propagation models that sees the systems in terms of components that create, transform and consume failures [11] and these techniques allow the architect to determine the likelihood that a hazard would occur based on the software architecture.

A shortcoming of these approaches is the assumption that all known hazards can be identified and codified as safety requirements before the architecture is built. Hazards can occur because of factors such as software, hardware and human behavior [27] and it is unrealistic to capture all the hazards that are caused by those factors or a combination of those factors. Therefore it is important to determine

the limitations of the safety analysis that is performed on the software architecture. In section 1, we describe how our technique mitigates this problem by taking a broader perspective of safety that includes quality attributes such as usability and maintainability.

Another limitation of the current approaches is the lack of support for changing an architecture to a "safer" architecture. There are a variety of safety tactics to improve the safety of an architecture [10] but it is hard to determine to what extent the application of these transformations improves the safety of the system. This is complicated by the fact that it is difficult to determine failure rates for components that are not even implemented or created yet. A technique that helps the architect in determining what is a "safer" is needed when ambiguities exists and definite values (such as the failure rates for certain components) are not present. In section 3, we explain how our technique can be used to derive a safer architecture.

Also, current techniques in safety evaluations do not consider the stakeholders or the organization that influence the evolution of architectures. Safety arguments [18] have been used to communicate between organization and stakeholders on how safety is achieved in a given system but current architecture evaluation techniques do not address this need for communication. Safety can mean different thing for different organization and stakeholders and there needs to be a way to address this difference. We show in section 3 that our technique can help the architect communicate how the architecture was derived based on drivers and the risks involved.

7. Conclusion and Future Work

We have demonstrated the viability of implementing a quality attribute reasoning framework by using other reasoning frameworks. The mappings used in this work transform a safety scenario with a required level of safety into a scenario in the target framework with the level of that attribute necessary to satisfy the original safety requirement. The results in the target reasoning frameworks provide the data for the safety reasoning framework to use to guide the architect toward beneficial modifications of the architecture.

Our work on reasoning frameworks has led us to identify relationships among frameworks. Initially we found that one way to address the varying definitions of an attribute is to view that attribute as a composite of other attributes. Foe example, an attribute such as dependability is really an aggregation of other attributes including reliability, availability, confidentiality, integrity, maintainability and safety. Now our work on safety has led to the realization that safety, as a quality, is a facade that has dependencies on other attributes for analytic theories that provide estimates of attribute values. Our future work will explore these relationships more deeply and possibly identify other relationships as well.

References

[1] Nasa software safety standard. http://satc.gsfc.nasa.gov/assure/nss8719_13.html.

[2] Risk management. http://www.sei.cmu.edu/risk/.

[3] A. Avižienis, J. Laprie, and B. Randell. Fundamental Concepts of Dependability. *Proceedings of 3rd Information Survivability Workshop ISW 2000 (Boston, USA)*, pages 7–12.

[4] F. Bachmann, L. Bass, and M. Klein. Preliminary design of arche: A software architecture design assistant.

[5] L. Bass, J. Ivers, M. Klein, and P. Merson. Reasoning frameworks (cmu/sei-2005-tr-007). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2005.

[6] L. Bass and B. John. Supporting usability through software architecture. *Computer*, 34(10):113–115, 2001.

[7] N. Dulac and N. Leveson. Incorporating Safety in Early System Architecture Trade Studies. In *International System Safety Conference Proceedings*, 2005.

[8] C. A. Ericson. *Hazard Analysis Techniques for System Safety*. John Wiley and Sons, Inc., 2005.

[9] P. Fenelon, J. McDermid, M. Nicolson, and D. Pumfrey. Towards integrated safety analysis and design. *ACM SIGAPP Applied Computing Review*, 2(1):21–32, 1994.

[10] L. Grunske. Transformational patterns for the improvement of safety properties in architectural specifications. *Proceedings of the VikingPLoP*, 3:3–5.

[11] L. Grunske. Early quality prediction of component-based systems–A generic framework. *The Journal of Systems & Software*, 80(5):678–686, 2007.

[12] T. Im and J. McGregor. Security in the Context of Dependability.

[13] T. Im and J. D. McGregor. Toward a reasoning framework for dependability. In *DSN 2008 Workshop on Architecting Dependable Systems*, 2008.

[14] C. Johnson and C. Holloway. The Dangers of Failure Masking in Fault-Tolerant Software: Aspects of a Recent In-Flight Upset Event. In *System Safety, 2007 2nd Institution of Engineering and Technology International Conference on*, pages 60–65, 2007.

[15] P. Jordan. *An introduction to usability*. Taylor & Francis, 1998.

[16] B. Kaiser, P. Liggesmeyer, and O. Mäckel. A new component concept for fault trees. In *Proceedings of the 8th Australian workshop on Safety critical systems and software-Volume 33*, pages 37–46. Australian Computer Society, Inc. Darlinghurst, Australia, Australia, 2003.

[17] R. Kazman, G. Abowd, L. Bass, and P. Clements. Scenario-based analysis of software architecture. *IEEE software*, 13(6):47–55, 1996.

[18] C. Kelly. *Arguing Safety-A Systematic Approach to Safety Case Management*. PhD thesis, PhD Thesis, Department of Computer Science Green Report YCST 99/05, University of York, England, 1999 CiteSeer. IST-Copyright Penn State and NEC.

[19] D. Lapierre and J. Moro. *Five past midnight in Bhopal*. Grand Central Publishing, 2002.

[20] N. Leveson. Safeware: System Safety and Computers. 1995.

[21] N. Leveson. Software safety: why, what, and how. *ACM Computing Surveys (CSUR)*, 18(2):125–163, 1986.

[22] R. Lutz. Targeting safety-related errors during software requirements analysis. *The Journal of Systems & Software*, 34(3):223–230, 1996.

[23] J. McGregor, F. Bachman, L. Bass, P. Bianco, and M. Klein. Using an Architecture Reasoning Tool to Teach Software Architecture. In *Proceedings 20th Conference on Software Engineering Education & Training (CSEE&T 2007)*, pages 275–282, 2007.

[24] J. McGregor, F. Bachmann, L. Bass, P. Bianco, and M. Klein. An Experience Using ArchE in the Classroom. *Software Engineering Institute (CMU/SEI-2007-TN-001)*, 2007.

[25] P. J. Wilkinson and T. P. Kelly. Functional hazard analysis for highly integrated aerospace systems.

[26] W. Wu and T. Kelly. Safety tactics for software architecture design. In *Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International*, pages 368–375, 2004.

[27] W. Wu and T. Kelly. Failure modelling in software architecture design for safety. In *Proceedings of the 2005 workshop on Architecting dependable systems*, pages 1–7. ACM New York, NY, USA, 2005.