Tracing Architectural Concerns in High Assurance Systems (NIER Track)

Mehdi Mirakhorli and Jane Cleland-Huang
School of Computing, DePaul University
243 S. Wabash Ave, Chicago, IL 60604
630-788-3810
{mehdi, jhuang}@cs.depaul.edu

ABSTRACT

Software architecture is shaped by a diverse set of interacting and competing quality concerns, each of which may have broad-reaching impacts across multiple architectural views. Without traceability support, it is easy for developers to inadvertently change critical architectural elements during ongoing system maintenance and evolution, leading to architectural erosion. Unfortunately, existing traceability practices, tend to result in the proliferation of traceability links, which can be difficult to create, maintain, and understand. We therefore present a decision-centric approach that focuses traceability links around the architectural decisions that have shaped the delivered system. Our approach, which is informed through an extensive investigation of architectural decisions made in real-world safety-critical and performance-critical applications, provides enhanced support for advanced software engineering tasks.

Categories and Subject Descriptors
D.2.11 [**Software Architectures**] Patterns

General Terms
Algorithms, Management, Measurement, Performance, Design

Keywords
Software Traceability, Architecturally Significant Requirements, Design Rationale, Architectural Preservation.

1. INTRODUCTION

Although a carefully designed and robust architecture is necessary for delivering a software system that satisfies its stakeholders' goals and quality concerns, it is not sufficient for guaranteeing that those quality concerns are satisfied over the lifetime of the software system. Unfortunately, ongoing maintenance activities resulting from new or modified requirements, general code re-factorings, performance tweaks, and bug fixes can erode the benefits derived from initial architectural decisions. Such *Architectural Erosion*, occurs when large and small architectural decisions are intentionally or accidentally violated [1]. In a best case scenario, these violations decrease the maintainability of the system, making it difficult to understand and modify; while in more extreme cases architectural erosion may lead to catastrophic and systemic failure.

In practice, the architecture of a system and its qualities can be maintained using traceability methods that help developers fully understand the impact of design or implementation changes on architecturally significant requirements. Kruchten [2], Burge [3], and others have proposed a proactive approach to preventing design degradation through using design rationales to document architectural decisions. They argue that explicitly recording design decisions, justifications, alternatives, and conflicting perspectives, is necessary in order to preserve architectural qualities. Such approaches utilize traceability links to establish relationships between design decisions, rationales, and architectural views, but do not provide specific guidelines for how the traceability links should be established. Without careful consideration, the resulting traceability links can easily proliferate, creating a situation in which traceability is difficult and costly to maintain, and in which links degenerate into an inaccurate and non-useful state [4].

This situation is illustrated through a series of traceability meta-models that were produced from a study of high-end traceability practices in industry [5]. The Traceability Information Model (TIM) depicted in Figure 1, is derived from these meta-models and shows significant redundancy of traceability paths for establishing relationships between issues, conflicts, alternative options, arguments, rationales, assumptions, requirements, and design decisions. For example decisions can be traced directly to requirements, or can be traced indirectly through either rationales or through issues and conflicts. There is no guidance to inform architects as to the best way to establish traceability, and as a result, practitioners are ill-informed as to how best to accomplish this task. Furthermore, various studies have highlighted the need to simplify the creation and use of traceability as a major process improvement need [6].

The traceability problem is exacerbated when tracing quality constraints related to *architecturally significant requirements* (ASR), which describe concerns such as reliability, security, and maintainability. Such ASRs often have a broad reaching impact across the system and are realized through components and behaviors that are visible in a variety of architectural and implementation views at very different abstraction levels.

Unfortunately, standard traceability meta-models do not begin to address this degree of complexity.

Prior to proposing more specific traceability guidelines for tracing ASRs, we conducted an extensive study of tactical architectural decisions in highly dependable and complex avionic systems. This study provided the foundations and motivation for a new decision-centric traceability (DCT) approach that includes a meta-model describing the required traceability links, and a strategic process for applying the meta-model. We demonstrate that our approach establishes semantically rich traceability links that can be used effectively to preserve architectural qualities.

2. HIGH ASSURANCE SYSTEMS

Our study involved reviewing the specifications of several high-assurance software systems including the Airbus A320/330/340 family, Boeing 777, Boeing 7J7 [7][8], NASA robots [9], and also performance centric systems such as Google Chromium OS. For each of the systems studied we identified critical quality goals, architecturally significant requirements, architectural decisions, design solutions, and views and models in which each of these techniques were visible. These findings were used to drive the traceability scheme described in this paper.

Reliability, availability, and fault tolerance were identified as primary concerns for the flight control systems of both the Airbus and Boeing. Due to space constraints, the examples in this paper focus around the reliability requirement, defined as *the likelihood of loss of aircraft function or critical failure is required to be less than 10^{-9} per flight hour* [10] .Similarly, our investigation of the CHROME browser identified security, portability, reliability, and availability as specific concerns. As a result of this study we observed seven issues that significantly influenced the proposed traceability approach. Each of these is discussed below:

- **Decisions are Hierarchical.** Strategic architectural decisions are hierarchical in nature. Although both high and low level decisions are often traceable back to individual user or system level requirements, it is often only the lower-level decisions that are traceable forward to the architectural views. This was illustrated by the different ways in which redundancy was achieved in the various avionics systems. Decisions included the degree of redundancy, the nature of redundant components, and the method used to consolidate results. For example, the airbus architecture used logical redundancy through use of multi-processes on a single processor, and then adopted an approach called “2-Self Checking Programming” to consolidate results. In this approach, a command unit’s results are compared to those of a monitoring unit and in the case that the results do not match, airplane control is switched to another computer. This example demonstrates that high-level architectural decisions are often associated with a fairly extensive set of subsequent lower-level decisions which impose constraints on the behavior, structure, and deployment of the system, and which work synergistically to support and shape the higher level decision. Traceability solutions must therefore explicitly link high-level decisions to their related low-level decisions, and then provide the means of tracing lower level decisions into the architectural design.

- **Visibility of Architectural Decisions.** Different architectural decisions are visible in different views, therefore traceability links must be established across a wide variety of architectural views. From observing the various architectural decisions in both the avionics systems and in Chrome, it is evident that different decisions have very different scopes of impact. While some decisions, such as Single Sign in (SSN), may deal with the structure of the system and lead to creation of sub-systems, layers or specific components, other decisions, such as fault recovery or performance decisions, affect how elements interact, perform their responsibilities or appear at runtime, and are more likely to be visible in behavioral models and runtime views. Furthermore, a layered view may show tactical decisions concerning the system’s portability, while a deployment view may depict decisions concerning the system’s performance and reliability. Some decisions are visible in multiple views, for example a decision might be traced to both architectural models (e.g., a class, an interface, a process or thread, a package or subsystem etc.) as well as implementation artifacts. An effective traceability scheme therefore needs the ability to trace ASRs to a wide variety of architectural and implementation views.

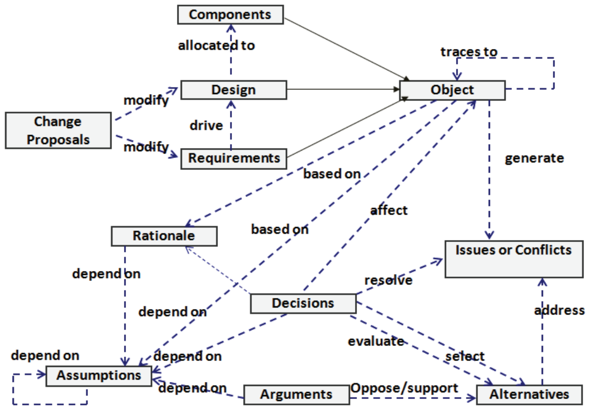

Figure 1. Components from Ramesh’s Metamodel [5]

- **Granularity of Architectural Impact.** Many architectural decisions are characterized by a set of roles and constraints. For example, an availability tactic such as heart-beat is traced at a coarse-grained level to a component that emits pings and another that listens for those pings and reacts accordingly. However, it may also be traced to one or more variables controlling the heart-beat rate. To preserve system reliability we need to trace the heart-beat decision to both the component level and to the variable level. Traceability links must therefore be created and maintained at various levels of granularity.

- **Tacit Architectural Knowledge.** Some architectural decisions represent tacit knowledge which is rarely articulated and, by definition, never documented. To prevent architectural erosion, tacit knowledge must be articulated as a design decision, even if it is not traceable into any specific architectural view. For example, an architect might decide that concurrent threads should not have direct write access to shared data. If traceability is to be used to prevent architectural erosion, tacit decisions must be explicitly articulated, documented, and traced. Sometimes the impact of decisions are so broad as to make it infeasible to trace them to specific components. Although a full discussion of this issue is outside the scope of this paper, possible solutions include aspect weaving, or use of coarse grained traceability links which in effect relate the decision to large sections of the solution domain**.**

- **Architectural Trade-offs.** Design decisions exhibit tradeoffs and interdependencies, which need to be captured so that the impact of a change to one decision is understood across the broader context of other architecturally significant decisions. Our study identified numerous examples of potential design trade-offs. For example, redundancy requirements in avionics systems, trade-off against weight requirements and performance requirements, and similarly security requirements in Chrome trade-off against performance. Relationships between decisions, including both positive contributions and negative trade-offs need to be explicitly modeled as traceability links [11].

• **Rich Semantics.** Balasubramaniam's prior study on traceability highlighted the need for traceability links to be semantically typed [5]. This is especially important in tracing ASRs because of the varied roles played by different components in realizing architectural decisions. For example, a given component might provide diversity for an N-Version design strategy, while another component might be responsible for coordinating voting tactics. Semantically typing each traceability link provides enhanced support for making sense of the traceability links and using them to support important tasks such as architectural preservation.

• **Minimalistic Strategy.** Complex high-assurance and high-performance systems are rich with design decisions [2][3]. Given the known problems of creating, maintaining, and using traceability links, it is important to develop a minimalistic traceability strategy that removes redundancy, while retaining only those traceability links needed to support critical software engineering tasks such as impact analysis and architectural preservation.

3. DECISION-CENTRIC TRACEABILITY

Based on the observations from the case study, we propose a decision-centric traceability (DCT) meta-model for capturing architectural decisions and establishing long-term relationships between architectural components and the non-functional requirements and concerns they are designed to satisfy. The model is depicted in Figure 2, and shows that traces are established from ASRs, through tactical architectural decisions to architectural elements visible across various architectural views. Trace redundancy is removed, while retaining the expressivity of the more traceability model. Figure 3 illustrates how the model might be instantiated for a redundancy decision that involves voting and N-Version programming.

Establishing all traces through architectural decisions, and disallowing traces from ASRs directly to the architectural design, addresses all of the issued identified through our case studies. As depicted in Figures 2 and 3, decisions can be structured hierarchically to capture the way in which sub-decisions help to achieve higher level decisions. Traces can then be established at any level of the decision hierarchy. Architectural trade-offs between decisions can be modeled by establishing negative contribution links. Documenting traces around design decisions supports intuitive visualizations and provides enhanced support for helping trace users to understand the impact of a decision across multiple architectural views. As decisions are frequently realized through the use of standard tactics, design patterns, or constraints, many of which include recognized roles; traceability links can be semantically typed to depict specific roles played by the architectural component in realizing a specific decision. For example, in Figure 3, traceability links to architectural components contributing to the redundancy tactic are semantically typed as "Coordinates", "Assigned to", "Provides Diversity",

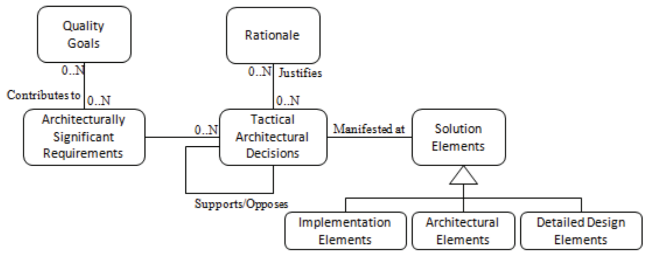

Figure 2. Decision-Centric Traceability Meta-Model

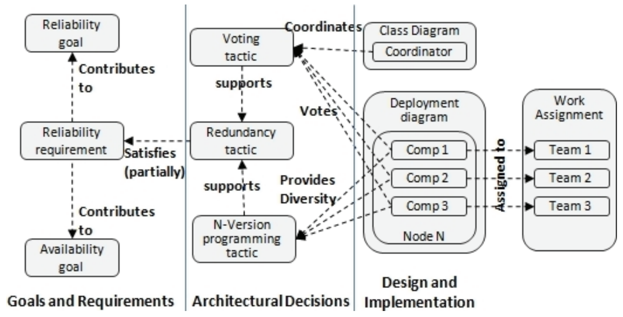

Figure 3. DCT Trace graph for redundancy decisions

Table 1: Trace Links in traditional versus DCT meta-models

Tactic	Traditional	DCT
Voting	19	15
Load Balancing	15	9
Heart Beat	7	5
Computational Efficiency	9	6
Role Based Access Control	7	5
Recovering From Attack	13	8

which immediately conveys the role of the component in realizing the decision.

Last, but not least, the DCT solution significantly reduces the number of traceability links needed. For example the redundancy tactic depicted in Figure 2 includes only 13 traceability links, as opposed to 20 traceability links that would have been required by the traditional traceability model. Table 1 depicts the links needed to model several design tactics using both traditional and DCT approaches. The traditional traces were generated using the metamodel depicted in Figure 1. As trace maintenance is costly and error-prone, reducing the number of links can translate into significant cost savings.

4. DCT IN PRACTICE

To evaluate the DCT model in practice we designed a simple experiment that evaluated its ability to support architectural preservation. The experiments were based on the requirements, decisions, and architectural views previously constructed for the Lunar Robot [12]. Two tools were developed. The first provided support for constructing traceability matrices to connect architectural decisions to components in various architectural views. This prototype monitors design changes in the StarUML modeling tool and utilizes the traceability links to provide real-time notification of potential architectural erosion as changes are made to various architectural views. The second prototype is implemented as a Plug-in for VisualStudio.NET and performs a similar function at the code level. In both cases, changes in the design or code are monitored and warning messages are displayed

Table 2. Results from Traceability Experiment

Trace Scenario		Trigger	Desired results	Impacts ASR?	Warning
1	A programmer copies code from component C' to component C.	Program P changes	Warning message generated that program P is part of N-Version Programming tactic, and that Component C may not share code with other voting components C1, C2,…,Cn.	Yes	True positive.
2	New methods are added to component C, to invoke a service provided by C2.	Design Refactoring	Neither C, C2, nor C3 have documented impacts on any ASRs. No warning messages are issued.	No	True negative.
3	C and C' are directly communicating components. They are refactored to communicate via a shared data structure.	Design Refactoring	Warning message that Components C is performance critical; its implementation should be based on an efficient algorithm	No. The change does not impact performance .	False positive.

to inform developers when their current efforts might cause design erosion and impact critical quality concerns such as reliability, dependability, or safety.

Eighteen different traceability scenarios were generated, and for each scenario we identified a trigger event, and articulated the potential negative impacts of the change as well as the potential architectural impacts that DCT should identify. In order to have scenarios representative of classes of known changes, we generated scenarios based on a published *Software Architecture Change Characterization Scheme* [14]. The purpose of the experiment was to determine whether DCT's meta-model was capable of correctly performing the impact analysis and issuing appropriate warning messages. Experiments were triggered as described, and the messages generated by the tool were compared to the desired outcomes shown in Table 2. Three of the eighteen scenarios and their ideal outcomes are shown in Table 2.

Nine of the scenarios affected architectural decisions, and the prototype tools correctly recognized these potential impacts and issued appropriate warnings. The remaining nine scenarios represented either micro-changes or functional changes that did not affect architectural decisions. Of these, our tool correctly filtered out five scenarios, but issued false positive warnings for the remaining four. The impact of false positives can be mitigated through providing interactive tools that control the timing of warning messages. False positives can also be reduced by creating more finely-grained traceability links, however these will increase the cost and effort required to create and maintain links, and therefore this approach is not suggested. False positives can be further managed by careful design of the GUI which makes warning messages visible yet non-obnoxious.

5. CONCLUSIONS

This paper has presented a meta-model for tracing architecturally significant requirements. It presents a rigorous, yet initial analysis of the issues facing architectural traceability. To our knowledge, these issues have not previously been addressed in such a systematic manner, and as a result existing traceability approaches for tracing ASRs and supporting design rationales tend to suffer from well documented maintenance, and usage problems [5][6] when applied to large and complex systems. Our DCT approach has been shown to support real-world traceability scenarios drawn from an extensive case study of high assurance and high performance applications, and to significantly reduce the complexity of the traceability infrastructure including the number of traceability links needed to capture important relationships between requirements, decisions, and architectural components. This new meta-model is expected to form the basis of future work in reverse engineering traceability links between requirements and design decisions, and for supporting long-term architectural maintenance activities such as impact analysis and architectural preservation.

6. ACKNOWLEDGMENTS

Funding for this work is partially provided by the National Science Foundation, grant number CCF: 0447594.

7. REFERENCES

[1] Perry, D. E. and Wolf, A. L. Foundations for the study of software architecture. *Softw. Eng. Notes* 17, 4(1992), 40-52.

[2] Kruchten, P.: An Ontology of Architectural Design Decisions in Software-Intensive Systems. *Workshop on Software Variability Mgmt*. Netherlands (2004).

[3] J. Burge, D.C. Brown, Software Eng. Using RATion-ale*, Journal of Sys. and Software*, 81(3): 395-413.

[4] Pohl K., Dömges R., Jarke M.: Towards Method-Driven Trace Capture. *CAiSE* 1997: 103-116

[5] Balasubramaniam R. and Matthias J., "*Toward Refer-ence Models for Requirements Traceability*", IEEE Trans. On Software Eng., Vol. 27, No. 1, Jan. 2001.

[6] Arnold, S. and Ricossa, S., Requirements Eng. @ CISCO, *Workshop on Reqs. Eng. ISSRE*, 2010, San Jose, CA, Nov. 2010.

[7] Daniel P. Siewiorek and Priya Narasimhan, *Fault-Tolerant Architectures For Space and Avionics Apps.*, NASA Ames Research http://ic.arc.nasa.gov/projects/ishem/Papers/Siewi.

[8] Aplin J.D. Primary flight computers for the Boeing 777 (1997) Microprocessors and Microsys., 20 (8), pp. 473-478.

[9] NASA Robot Web Links, http://prime.jsc.nasa.gov/ ROV/nlinks.html

[10] M. Sghairi, A. de Bonneval, Y. Crouzet, J.-J. Aubert and P. Brot , Challenges in Building Fault -Tolerant Flight Control System for a Civil Aircraft . *IAENG International Journal of Computer Science*, IJCS_35_4_07.

[11] Cleland-Huang, J., Marrero, W., and Berenbach, B. Goal-Centric Traceability: Using Virtual Plumblines to Maintain Critical Systemic Qualities. *IEEE Trans. Softw. Eng.* 34, 5. 2008, 685-699.

[12] Mirakhorli, M., Cleland-Huang, J., A Decision-Centric Approach for Tracing Reliability Concerns in Embedded Software Systems. *Workshop on Embedded Software Reliability (ESR)*, held at ISSRE'10, San Jose, Nov, 2010.

[13] Mirakhorli, M., Cleland-Huang, J., Transforming Trace Information in Architectural Documents into Re-usable and Effective Traceability Links, 6th Workshop on SHAring and Reusing architectural Knowledge (SHARK), *33rd Int. Conf. on Software Engineering (ICSE 2011)*, Waikiki, Honolulu, Hawaii, May 21-28, 2011.

[14] Byron J. Williams and Jeffrey C. Carver. 2010. Characterizing software architecture changes: A systematic review. *Inf. Softw. Technol.* 52, 1 (January 2010).