Applying Design Diversity to Aspects of System Architectures and Deployment Configurations to Enhance System Dependability

Matthew J. Hawthorne and Dewayne E. Perry
Department of Electrical and Computer Engineering
The University of Texas at Austin
mhawthorne@ece.utexas.edu, perry@ece.utexas.edu

Abstract

Design diversity has been proposed as a strategy for reducing the number of co-occurring faults in multiple redundant versions of a given application. While much research has focused on methods for enhancing the multiversion diversity of applications or estimating system dependability, relatively less work has been done to incorporate supporting system and infrastructure elements into design diversity frameworks, even though these aspects of system design can negatively impact dependability. We propose general diversity-enhancing properties that can be used to extend design diversity "downward" through layered software subsystems, operating systems, and hardware platforms, and "outward" through networks, power supplies, and other external infrastructures. We identify three key properties of diverse systems: modal diversity, geographical diversity, and ecological diversity, and develop a conceptual framework for applying these properties to all aspects of a software-based system to increase the effective independence of diverse application versions by using integrated hardware-software "channels" to reduce system-induced common failure modes.

1. Introduction

Software-automated systems, including safety-critical control systems [4, 8, 24], have become commonplace, making dependability a paramount architectural [18] concern. Much research on software dependability has focused on enhancing system dependability by using N-version programming and similar methods that use redundancy to enhance fault tolerance [3, 5, 16], as well as to improve software quality [15].

This paper extends design diversity to the entire deployable system, including hardware (processing units, storage units, and related physical components), communication networks, supporting infrastructure (power supply, etc.), and the operating system and layered software subsystems, all of which are potential sources of errors and failures that can reduce the overall dependability of software-based systems [14]. This paper examines some of the ways software redundancy and design diversity are being used to increase the dependability of software systems, and explores ways to apply design diversity principles to supporting elements to enhance overall system dependability.

As defined by [1], "dependability" describes related system properties including availability, reliability, safety, confidentiality, integrity, and maintainability. This paper is primarily concerned with system reliability and availability, along with certain aspects of integrity, and possibly, safety.

The remainder of the paper is organized as follows. Section 2 gives a brief introduction to software redundancy and design diversity research. Section 3 discusses limitations of applying design diversity only to the application being developed. Section 4 proposes some general cross-cutting properties that can be used to extend design diversity beyond the application being developed. Section 5 presents a conceptual model for modeling redundant systems as sets of diverse hardware-software processing "channels". Section 6 discusses influences and limitations of the approaches presented in this paper. Section 7 summarizes the conclusions of the paper, and discusses our ongoing and planned research.

2. Redundancy and Design Diversity

The use of redundancy to enhance system dependability is based on the assumption that if one version of the application fails, the remaining versions are likely to give the correct response, making the system resistant to faults. To be effective, redundancy requires that different versions fail independently (i.e., have non-overlapping failure patterns [14]).

N-version programming [16], in which two or more versions of a system are developed independently by different teams of developers, as well as related approaches using multiple versions of off-the-shelf (OTS) and open source software (OSS) components [7, 11, 19] have been used to build diverse redundant systems. Connector-based architectures [22] have been developed to support multiple versions of OTS components in diverse systems. While achieving failure pattern diversity [6, 12, 15], and making valid estimates of reliability [20, 21] remain areas of active research, design diversity has increased the overall dependability of software systems where it has been employed [2, 4, 9, 10, 15, 17, 25].

3. Limitations of Application Design Diversity

When applied only to software development processes, design diversity-based approaches are largely limited in scope to influencing the diversity of the application being developed, although design diversity may induce a kind of data diversity in surrounding non-diverse subsystems by introducing variation in application-level interactions with those subsystems [14]. However, if the supporting elements of a system, such as hardware, operating systems, and layered subsystems, contain errors or vulnerabilities, system dependability may be degraded no matter how dependable the application software itself is. Since complex software systems nearly always contain residual errors [23], some supporting system errors are a virtual certainty. Viruses, Trojan horses, worms, and other deliberate attacks may also impact dependability, as may surrounding infrastructure such as the network and power supply. Many modern software systems depend on local network or Internet access, making them vulnerable to connectivity disruptions; and since all computer systems need power to operate, the dependability of the local and regional power grids can also impact effective system dependability. Many additional examples could be listed, but an exhaustive list is not practical, nor is such a list necessary to demonstrate the need to include external system and infrastructure elements when designing and deploying dependable software systems.

4. System Design Diversity

Since the dependability of a given software system depends on a diverse set of external factors that is impossible to fully characterize *a priori* without knowing the details of the system, we next describe three general properties that may be used to enhance the diversity of virtually any aspect of a given system: *modal diversity*, *geographical diversity*, and *ecological diversity*.

4.1. Modal diversity

"Modality" is the extent to which any system function is limited with regard to available modes of operation. Any function that can be done only one way exhibits high modality, and could potentially become a single point of failure. *Modal diversity* is the extent to which any system function is designed to utilize multiple modes of operation. For example, a system that can utilize a direct broadband connection, a telephone modem connection, or a satellite link to communicate with other systems exhibits modal diversity in networking. A system that can contact a plant operator by telephone or digital pager (e.g., if the operator fails to respond to a screen warning) demonstrates modal diversity in its human interface., as does a system with multiple power sources (e.g., main power grid and backup generator).

4.2. Geographical diversity

"Locality" refers to the "localness" of a software system. A system that exhibits a high degree of locality (e.g., by being deployed on a single processor or only a few processors in close physical proximity) is intrinsically less dependable in certain respects than a system that is distributed over a larger area, because a high-locality system is more likely to be subject to local effects, such as network or power outages, random incidents (e.g., system accidentally unplugged), etc. *Geographical diversity* is the extent to which the physical and software elements of a system are widely distributed geographically, making them less subject to common local effects.

4.3. Ecological diversity

"Ecology" refers to the system environment in which an application runs, including hardware, operating systems, and other software. Homogeneous system ecologies do not contribute to system-level diversity, nor can they enhance application-level diversity [14]. In addition, homogeneous systems are more vulnerable to common-mode failures caused by viruses and worms, which often affect only closely related operating systems or components. *Ecological diversity* is the extent to which an application operates in a heterogeneous system environment. For example, a system deployed on a single operating system is not ecologically diverse with respect to its operating system.

4.4. Other system diversity properties

Other system properties enhance or characterize aspects of system diversity. Among the more interesting are *temporal diversity*, *control diversity*, and *combinational diversity*.

Temporal diversity is the ability of a system to adapt to temporal variability (variable timing of events). Examples include the ability to handle widely varying network delays, or variable data-induced delays between the start and the end of transactions.

Control diversity refers to the diversity of the automatic and human interface control of a system. For example, a system that utilizes a distributed control algorithm exhibits greater control diversity than one that depends on a single server system to control all remote nodes.

Combinational diversity is diversity induced by different nodes in a system running on unique sets of hardware, system software, and application software, the *combination* of which varies across the system.

5. A Conceptual Model for Diverse Systems

Design diversity can be used to model systems as sets of diverse hardware-software "channels", or parallel hardware-software execution paths [14]. These channels enhance the independence of redundant application versions by allocating each version to a distinct set of supporting hardware and software. Hardware, software and infrastructure functions are generically modeled as different "aspects" of the system, and for each modeled aspect, a corresponding set of available implementation options and dependencies is built. These options are used to design a distinct integrated channel for each version of the application software. A conceptual model of such a system is depicted in **Figure 5.1**.

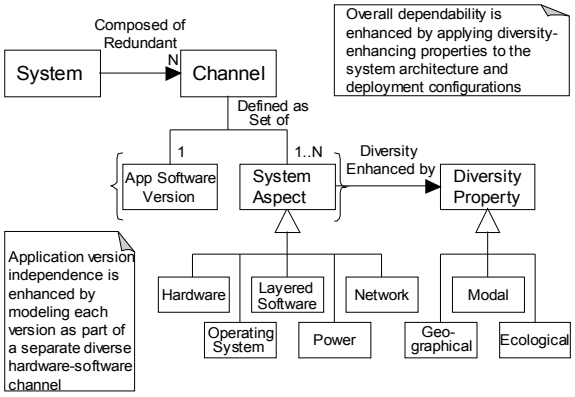

Figure 5.1. Conceptual system model

Designing the core channel model for this kind of diverse multi-channel system involves four basic steps. The first step is to build an *aspect model* defining which aspects of the system will be modeled. The aspect model may include all aspects of the system and its supporting infrastructure, or the system designer may choose to focus on the aspects of the system for which diversity is feasible (i.e., those aspects for which multiple diverse options are available). Next, an *implementation model* is built that defines one or more implementation options for each modeled aspect of the system. The final steps are to build a *channel model* defining a set of two or more execution channels for the system, and to assign implementation options for each modeled aspect to each channel.

For example, let *A* = {*application version, application framework, operating system, and hardware platform*} represent the set of system aspects being modeled for a new system. The system architect has available two independently developed versions of the application software (*V1* and *V2*), which were built using two different application frameworks (*.NET* and *EJB*). Configurations have been tested using two hardware platforms (*x86* and *Macintosh*), running a total of three operating systems (*Windows XP*, *Linux*, and *Mac OSX*). A channel model is built defining the four-channel system represented in **Figure 5.2**.

Note that the system has two channels, *C1* and *C3*, that are mutually diverse with respect to *A* (i.e., diverse with respect to the set of system aspects being modeled), while the remaining channels, *C2* and *C4*, are not mutually diverse with respect to the other channels. *C2* and *C4* are examples of combinatorially diverse channels, where not all of the modeled system aspects are unique relative to the other channels, but the *combination* of aspect implementations is unique. This may still provide some measure of inter-channel independence.

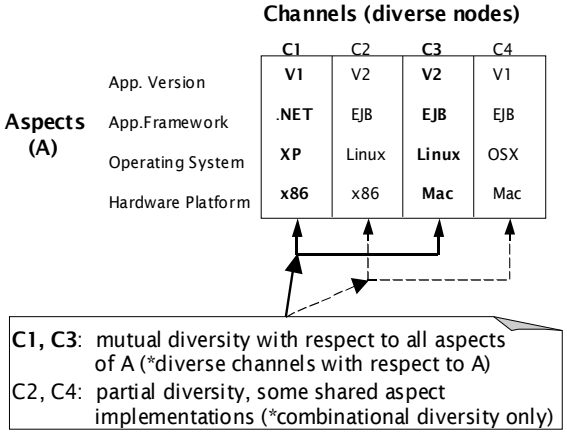

Figure 5.2. Example of aspect-diverse channels

The hybrid deployment/component architecture model of a channel-based system presented in this section is included to illustrate how design diversity can be applied to enhance the inter-version independence of diverse multiversion software systems. As such, the model is missing many details that are necessary to fully specify an actual system architecture. These include implementation component interdependencies; inter-layer and inter-component connection frameworks, if employed [22]; system startup, control and synchronization protocols and algorithms; and an adjudication mechanism to evaluate the output from the parallel execution channels, among many other details omitted for clarity and brevity.

6. Discussion

The extended design diversity approach presented in this paper, including the notion of multiple diverse software computation "channels", builds on previous research, most notably [13] and [14], as well as earlier hardware channel research. A kind of partial diversity somewhat similar to the combinational diversity in this paper is suggested in [13], although the scope ("lower levels" in the system architecture), and purpose (protection against failures of particularly important functions) are quite different, as is the implied granularity. Timing diversity similar to the temporal diversity in this paper is also described in [14].

The rationale for believing that extending design diversity to additional aspects of system architecture and infrastructure design will result in increased system dependability is based, in a general intuitive sense, on results from the extensive body of research in the area of software design diversity and multiversion programming, as well as the equally extensive body of research on hardware fault tolerance. Redundancy is the most common approach used to make all kinds of digital systems more fault-tolerant.

More specific support is based on probabilistic models for the reliability of diverse systems [6, 12]. The Littlewood and Miller (LM) model described in [12] and [13] computes the probability that two diverse versions of a program, developed using two different methodologies, A and B, will fail under the same conditions (input), as the product of the probabilities that each program will fail given the same input, plus the covariance of the probability functions of A and B over the range of possible inputs:

$$E_x(\Theta_A(X)\Theta_B(X)) = E_x[\Theta_A(X)]E_x[\Theta_B(X)] + Cov(\Theta_A(X)\Theta_B(X)), \quad (6.1)$$

where $Cov(A,B)$ denotes the covariance of A and B.

Combining the LM model assumption that forced diversity results in different error probability distributions over the population of programs, with the observation that diverse hardware platforms, operating systems, and OTS and OSS components developed on those systems typically share many characteristics of forced diversity methodologies (e.g., mutual isolation; different design objectives, directives, and constraints; different development languages and implementation environments; etc.), the LM model can be extended in a straightforward manner to include system components used to implement aspects of a system design:

$$E_x(\Theta_{A_C}(X)\Theta_{A_D}(X)) = E_x[\Theta_{A_C}(X)]E_x[\Theta_{A_D}(X)] + Cov(\Theta_{A_C}(X)\Theta_{A_D}(X)), \quad (6.2)$$

where A_C and A_D represent a pair of system components, C and D, used to implement a given system aspect A.

Practical limitations to the extended design diversity approach proposed in this paper are mainly the result of increased system complexity and its attendant costs and risks, as well as the limited number of implementations available for many system and infrastructure components. Potential reliability gains must be measured against real and potential costs, including increased system administration overhead, the risk of errors or inadvertent security breaches caused by unfamiliarity with diverse systems or components, and increased development, deployment and maintenance costs, among others.

7. Conclusions and Further Research

In this paper, we build on the observation that system and infrastructure components that lie outside the scope of the application being developed can fail, introducing common failure modes into the system that can reduce the independence of diverse application versions and degrade the dependability of the system as a whole. We propose general diversity-enhancing properties that can be applied to the full range of application, system and infrastructure elements. We also present a conceptual model for increasing the effective independence of diverse application versions by designing diverse, vertically integrated processing channels in which versions can execute.

Current research includes exploring the effects of system diversity on dependability, as well as developing comprehensive architectures to model and implement diverse systems including system and infrastructure elements. Ongoing work includes

developing tools that utilize aspect-oriented programming (AOP) techniques to generate diverse system architectures and deployment configurations statically based on compile-time configuration directives, dynamically based on current application and system state, or both. Future plans include extending the configuration system to support self-healing runtime behavior, such as automatically switching to a different implementation if a component fails or becomes compromised.

7. References

[1] A. Avizienis, J.-C. Laprie, and B. Randell, "Fundamental Concepts of Dependability", *Technical Report 739*, Department of Computer Science, University of Newcastle upon Tyne, 2001.

[2] G. Bishop, "Software Fault Tolerance by Design Diversity", in M. Lyu (Ed.), "Software Fault Tolerance", John Wiley & Sons, 1995, pp. 211-229.

[3] S. Brilliant, J. Knight, and N. Leveson, "Analysis of Faults in an N-Version Software Experiment", *IEEE Trans. on Software Engineering*, vol. 16, no. 2, 1990, pp. 238-247.

[4] D. Briere and P. Traverse, "Airbus A320/A330/A340 Electrical Flight Controls – A Family of Fault-Tolerant Systems", in *Proc. 23rd Intl. Symposium on Fault-Tolerant Computing (FTCS-23)*, Toulouse, France, 22-24, 1993, pp. 616-623.

[5] L. Chen and A. Avizienis, "N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation", *Digest of 8th Annual Intl. Symposium on Fault-Tolerant Computing*, Toulouse, France, June 1978, pp. 3-9.

[6] D. Eckhardt and L. Lee, "A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors", *IEEE Trans. on Software Engineering*, SE-11, 1985, pp. 1511-1517.

[7] C. Gacek, T. Lawrie and B. Arief, "Interdisciplinary Insights on Open Source", In *Proc. of the Open Source Software Development Workshop*, University of Newcastle, Newcastle upon Tyne, UK, Feb. 25-26, 2002, pp. 68-82.

[8] G. Hagelin, "ERICSSON Safety System for Railway Control", in *Software Diversity in Computerized Control Systems*, U. Voges (ed.), Springer-Verlag, New York, 1987, pp. 11-21.

[9] L. Hatton, "N-Version Design Versus One Good Version", *IEEE Software*, vol. 14, 1997, pp. 71-76.

[10] H. Kantz and C. Koza, "The ELEKTRA Railway Signalling-System: Field Experience with an Actively Replicated System with Diversity", in *Proc. 25th IEEE Annual Intl. Symposium on Fault-Tolerant Computing (FTCS-25)*, Pasadena, CA, 1995, pp. 453-458.

[11] T. Lawrie and C. Gacek, "Issues of dependability in open source software development", *ACM SIGSOFT Software Engineering Notes*, vol. 27, no. 3, ACM Press, New York, 2002, pp. 34-37.

[12] B. Littlewood and D. Miller, "Conceptual Modeling of Coincident Failures in Multi-Version Software", *IEEE Trans. on Software Engineering*, SE-15, 1989, pp. 1596-1614.

[13] B. Littlewood, P. Popov and L. Strigini, "Modelling software design diversity – a review", *ACM Computing Surveys*, vol. 33, no. 2, 2000, pp. 177-208.

[14] B. Littlewood, Lorenzo Strigini, "A discussion of practices for enhancing diversity in software designs", *DISPRO Project Draft Technical Report LS_DI_TR-04*, version 1.1d, 23 Nov., 2000.

[15] J. Knight and N. Leveson, "An Experimental Evaluation of the Assumption of Independence in Multiversion Programming", *IEEE Trans. on Software Engineering*, vol. SE-11, Jan., 1986, pp. 1511-1517.

[16] J. Knight, N. Leveson, and L. St. Jean, "A Large Scale Experiment in N-Version Programming", In *Dig. FTCS-15: 15th Annual Intl. Symposium on Fault-Tolerant Computing*, Ann Arbor, MI, June, 1985, pp. 135-139.

[17] M. Lyu (Ed.), *Software Fault Tolerance*, John Wiley & Sons, 1995.

[18] D. Perry and A. Wolf, "Foundations for the Study of Software Architecture", *ACM Software Engineering Notes*, 17(4), Oct. 1992, pp. 40-52.

[19] P. Popov, "Reliability Assessment of Legacy Safety-Critical Upgraded with Off-the-Shelf Components", in *Proc. SAFECOMP 2002*, Catalina, Italy, Springer, 2002, pp. 10-13.

[20] P. Popov, A. Romanovsky and L. Strigini, "Choosing effective methods for design diversity – how to progress from intuition to science", in *Proc. SAFECOMP '99, 18th Intl. Conf. on Computer Safety, Reliability and Security*, Toulouse, France, Springer, 1999, pp. 272-285.

[21] P. Popov, L. Strigini, J. May and S. Kuball, "Estimating Bounds on the Reliability of Diverse Systems", *IEEE Trans. on Software Engineering*, vol. 29, no. 4, 2003, pp. 345-359.

[22] M. Rakic and N. Medvidovic, "Increasing the Confidence in Off-the-Shelf Components: A Software Connector-Based Approach", in *Proc. of the 2001 Symposium on Software Reusability*, 2001.

[23] B. Randell, "Turing Memorial Lecture: Facing up to Faults", *The Computer Journal*, vol. 43, no. 2, 2000, pp. 95-106.

[24] J. Roquet and P. Traverse, "Safe and Reliable Computing on Board the Airbus and ATR Aircraft", *SAFECOMP 1986*, Sarlat, France, Oct. 1986, pp. 93-97.

[25] U. Voges (Ed.), "Software Diversity in Computerized Control Systems", *Springer-Verlag*, Wien, 1988.