An Empirical Analysis of Providing Assurance for Self-Adaptive Systems at Different Levels of Abstraction in the Face of Uncertainty

Erik M. Fredericks and Betty H. C. Cheng
Department of Computer Science and Engineering
Michigan State University, East Lansing, Michigan, 48824, USA
Email: {freder99, chengb}@cse.msu.edu

***Abstract*—Self-adaptive systems (SAS) must frequently continue to deliver acceptable behavior at run time even in the face of uncertainty. Particularly, SAS applications can self-reconfigure in response to changing or unexpected environmental conditions and must therefore ensure that the system performs as expected. Assurance can be addressed at both design time and run time, where environmental uncertainty poses research challenges for both settings. This paper presents empirical results from a case study in which search-based software engineering techniques have been systematically applied at different levels of abstraction, including requirements analysis, code implementation, and run-time validation, to a remote data mirroring application that must efficiently diffuse data while experiencing adverse operating conditions. Experimental results suggest that our techniques perform better in terms of providing assurance than alternative software engineering techniques at each level of abstraction.**

I. Introduction

A self-adaptive system (SAS) can reconfigure at run time in response to changing system and environmental conditions [1], [2], [3], [4] and may face unexpected conditions for which it was not explicitly designed [5], [6], [7]. Examples of SAS reconfigurations include selecting a new configuration of system parameters, adapting run-time behavior, or selecting different modes of operation. Moreover, providing assurance for an SAS at different phases of the software life cycle attempts to ensure that the system will behave according to its requirements. However, uncertainty in both the system configuration and environmental parameters may cause design-time decisions, such as elicitation of requirements or derivation of test cases, to no longer reflect the operating context in which the SAS operates [8], [9]. In particular, uncertainty in the system configuration can be a result of partially informed decisions or assumptions regarding SAS requirements or adaptations to the SAS as a result of a reconfiguration. Uncertainty in the environmental parameters can be a result of a misunderstanding of the execution environment. Previously, we have introduced a suite of search-based techniques to address assurance at different levels of SAS abstraction. More specifically, we presented AutoRELAX [10], [11] to address requirements-based assurance, Fenrir [12] to target code-based assurance, and Proteus [13] and Veritas [14] to address assurance via run-time adaptive testing. This paper describes an empirical study to illustrate how we systematically applied each of our techniques to the same application at design time and run time to provide assurance at different levels of abstraction.

In general, it is difficult for an engineer to fully anticipate all conditions that an SAS may experience throughout its lifetime [1], [3], [4], [5]. To this end, different techniques, some of which leverage search-based heuristics, have been developed for verifying and validating an SAS at both design time [15], [16], [17], [18], [19], [20] and run time [18], [21], [22], [23], [24], [25], however these techniques generally evaluate requirements with respect to design-time decisions and may not consider unexpected changes to the SAS operating context. As such, techniques for enabling SAS run-time assurance are required to ensure that requirements are robust, that the code has been implemented correctly, and that the SAS behaves according to its requirements at run time.

This paper presents results from an end-to-end empirical study in which assurance is addressed at the requirements, implementation, and run-time testing levels for an SAS using search-based techniques. To this end, we overview and present results from AutoRELAX, Fenrir, and Proteus, respectively. AutoRELAX is a requirements-based assurance technique for introducing flexibility into a system goal model using a genetic algorithm [10], [11]. Fenrir is a code-based assurance technique for exploring how an SAS may exhibit different behaviors in uncertain environments using novelty search [12]. Proteus is a framework for performing run-time test adaptation [13], [14]. Combined, these techniques address assurance for an SAS at three different levels of abstraction.

To this end, we applied each technique to a remote data mirroring (RDM) network application provided by industrial collaborators [26], [27], where RDM is a data protection technique for maintaining data availability and accessibility, to enhance assurance in response to uncertainty. In particular, the RDM network comprises a set of physically remote data mirrors, each of which are connected by network links, tasked with diffusing data to each data mirror connected to the network. As such, the RDM is subject to network uncertainty in the form of dropped or delayed messages, random network link failures, and unexpected damage to RDM sensors. The RDM application has been modeled as an SAS, and as such, can self-reconfigure to mitigate these uncertainties. Possible reconfigurations include changes to the network topology or

updates to data propagation techniques at individual data mirrors (i.e., asynchronous vs. synchronous propagation).

Experimental results suggest that applying each of our techniques at different stages of SAS development can provide specific improvements over existing or manual approaches to ensure that the system performs according to its requirements. The remainder of this paper is structured as follows. Section II overviews relevant background information, including the RDM application, goal-oriented requirements engineering, and the RELAX specification language. Following, Section III describes each of our assurance techniques. Section IV then presents our experimental results at each level of SAS abstraction. Lastly, Section V presents our conclusions and discusses future work.

II. Background

This section describes the RDM application, goal-oriented requirements engineering, and the RELAX specification language.

A. RDM Application

This section presents the RDM application obtained from an industrial collaborator [26], [27]. RDM is a data protection technique that prevents data loss and maintains availability by storing copies of data on physically remote servers (i.e., data mirrors). An RDM can provide continuous access to data by replicating data to data mirrors and moreover can ensure that data is not lost or damaged. If an error or failure occurs to a data mirror, then recovery is enabled by requesting a new copy of the affected data or reconstructing the data as provided by another data mirror. The RDM must also minimize consumed bandwidth and make certain that distributed data is not lost or corrupted to enable assurance.

To mitigate uncertainty, the RDM can reconfigure at run time to tolerate dropped messages, delayed messages, and network link failures. Each network link within the RDM network incurs a cost in terms of budget (i.e, monetary cost to activate a network link) and performance (i.e., throughput, latency, and loss rate). These metrics, in turn, summarize the performance and reliability of the RDM network. In addition, the RDM can reconfigure in terms of network topology and data propagation protocols. Network topology can be reconfigured by activating and deactivating network links, and data propagation for each individual data mirror can be defined as asynchronous or synchronous. Asynchronous propagation collects updates at the transmitting data mirror and periodically transmits data to the receiving mirror. Synchronous propagation ensures that the receiving mirror receives and writes incoming data prior to completion at the transmitting data mirror. The RDM can be modeled as an SAS application based on its ability to reconfigure at run time [28].

B. Goal-Oriented Requirements Engineering

Goal-oriented requirements engineering (GORE) is a graphical approach to capturing high-level objectives and constraints that a system must satisfy and can be used to guide the elicitation and analysis of requirements. Specifically, a *functional* goal declares a service that must be provided, a *non-functional* goal imposes a quality constraint, a *safety* goal declares a critical objective that must always be satisfied to mitigate dangerous situations, and a *failsafe* goal specifies a safe fallback state in case of failure [29]. Furthermore, goals may be designated as invariant (i.e., "Maintain" or "Avoid" goals), requiring that they must always be satisfied, or non-invariant (i.e., "Achieve" goals), indicating that they may be temporarily unsatisfied. Moreover, safety and failsafe goals are always considered to be invariant to ensure the continuing safe operation of the SAS. A goal can be measured using utility functions that quantify run time satisfaction, yielding a normalized value that quantifies goal satisfaction [30].

GORE decomposes high level goals into finer-grained subgoals with a directed acyclic graph [31], where each subgoal must be satisfied for its parent to also be satisfied. KAOS [31] is an approach for systematically refining goals using AND and OR refinements, continuing until each goal has been assigned to an agent. Leaf-level goals are considered to be requirements. Figure 1 presents the left half of the RDM goal model. For our implementation of the RDM application, the goal model acts as both a requirements specification and as a model for quantification of RDM run-time behavior.

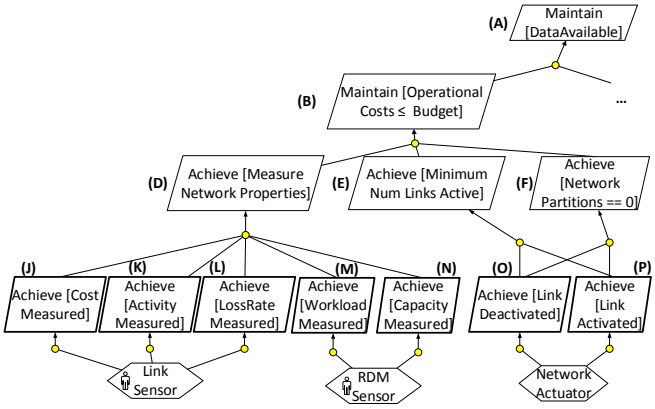

Fig. 1. Left half of RDM goal model (excerpted for brevity).

C. RELAX Specification Language

RELAX [5], [7] is a requirements specification language used to identify and assess sources of uncertainty. In particular, RELAX uses fuzzy logic functions to capture uncertainty in requirements, adding flexibility in terms of how and when a requirement can be satisfied. For instance, Goal (F) in Figure 1 will be violated if a network link fails, causing the RDM network to become partitioned. As such, the addition of a RELAX operator that tolerates a transient or temporary network partition provides flexibility for the RDM to continue to satisfy its key objectives while minimizing the need to reconfigure, thereby reducing the cost of adaptation to the SAS. RELAX operators can only be applied to non-invariant goals, as invariant goals are precluded from adaptation. Table I lists

the RELAX operators used in this paper. For example, Goal (F) can be RELAXed to become "Achieve [Network Partitions AS CLOSE AS POSSIBLE TO 0]," thereby tolerating a transient network partition.

TABLE I
RELAX OPERATORS

Temporal Operators
AS EARLY AS POSSIBLE
AS CLOSE AS POSSIBLE TO [frequency]
AS LATE AS POSSIBLE
Ordinal Operators
AS FEW AS POSSIBLE
AS CLOSE AS POSSIBLE TO [quantity]
AS MANY AS POSSIBLE

III. ASSURANCE TECHNIQUES

This section overviews AutoRELAX, Fenrir, Proteus, and Veritas.

A. AutoRELAX

AutoRELAX [10], [11] is a design-time technique for automatically providing requirements-based assurance for an SAS. Specifically, AutoRELAX uses a genetic algorithm to explore the solution space of possible combinations of RELAX operators as applied to a system goal model. AutoRELAX searches for a RELAXed goal model based on three criteria. First, the RELAXed goal model must *minimize SAS adaptations* to reduce cost, either in terms of budget or system impact (c.f., Equation 1). Second, AutoRELAX must *minimize the number of RELAX operators* to reduce needless flexibility in system requirements (c.f., Equation 2). For example, Figure 1, Goal (E) could be RELAXed to state "Achieve [Minimum Number of Network Links Active AS LATE AS POSSIBLE]" to introduce flexibility in when Goal (E) is satisfied, as a transient network partition may require that additional network links be activated. However, too many applied RELAXations may cause the SAS to be unnecessarily flexible. Third, the *invariant requirements must always be satisfied*, otherwise the goal model is considered a failure (c.f., Equation 3). Collectively, these three objectives, aggregated into a single fitness value (c.f., Equation 3), guide the search process towards a RELAXed goal model that is optimized to mitigate uncertainty in both the system configuration and environmental parameters at the requirements level.

$$FF_{nrg} \;=\; 1.0 - \Big(\frac{|relaxed|}{|Goals_{non-invariant}|}\Big) \tag{1}$$

$$FF_{na} \;=\; 1.0 - \Big(\frac{|adaptations|}{|faults|}\Big) \tag{2}$$

$$fitness = \begin{cases} \alpha_{nrg} * FF_{nrg} + \alpha_{na} * FF_{na} & \text{invariants true,} \\ 0.0 & \text{else.} \end{cases} \tag{3}$$

B. Fenrir

Fenrir [12] is a design-time technique for automatically exploring how an SAS reacts to uncertainty at the implementation level. In particular, Fenrir uses novelty search [32] to generate a suite of operating contexts (i.e., combinations of system and environmental parameters) that cause an SAS to exhibit different behaviors at run time. Fenrir generates an execution trace that represents SAS execution, including functional behaviors and paths of reconfiguration, in response to each individual operating context. Fenrir then measures the differences in execution between each generated execution trace to determine which operating context yielded the most diverse behaviors. An SAS engineer can then examine the resulting set of execution traces to determine if the SAS reacted appropriately to its operating context, possibly uncovering latent errors or feature interactions in the SAS codebase. Moreover, as each operating context is paired to an execution trace, the output generated by Fenrir can provide a representative overview of SAS behavior in response to uncertainties posed within each operating context.

C. Proteus and Veritas

Proteus [13] and Veritas [14] are techniques that address assurance at run time by performing adaptive, run-time testing for test suites and test cases, respectively. Testing is adapted at run time to reflect changing conditions, as test suites and test cases derived at design time may no longer be applicable as a result of unexpected environmental changes or SAS reconfigurations. To this end, Proteus is a managing framework for performing run-time testing, including text execution and *coarse-grained* test suite adaptation. Proteus invokes Veritas to perform *fine-grained* test case parameter value adaptation at run time. Veritas is a search-based technique that optimizes test cases at run time using an online evolutionary algorithm [33].

Both Proteus and Veritas perform run-time adaptation to ensure that test suites and test cases, respectively, remain *relevant* to changing operating contexts experienced by the SAS. As test cases typically exhibit a pass or fail behavior, we define a *test case relevance* value to quantify how applicable test cases are to their operating context, where the relevance value shows the distance between the observed and expected values of the test case. A sample relevance calculation is shown in Equation 4:

$$relevance_{TC_i} \;=\; 1.0 \;-\; \frac{|value_{observed} - value_{expected}|}{|value_{expected} + value_{variance}|} \tag{4}$$

where $value_{observed}$ is the observed value of the test case, $value_{expected}$ is the test case expected value, and $value_{variance}$ is the maximum value that the test case may take, as defined by the test engineer, to ensure that its relevance value remains normalized on $[0.0, 1.0]$. Moreover, the test engineer must correlate each test case to at least one goal in the goal model for run-time validation to ensure that test adaptation does not result in an invalid test case. The relevance

value shown in Equation 4,[1] coupled with run-time monitoring of the satisfaction of the correlated goal(s),[2] determines if an executed test case is a:

- **True positive:** Test has passed successfully (i.e., $relevance_{TC_i} \geq 0.75$, goal satisfied). No test or SAS adaptation is required.
- **True negative:** Test has failed successfully (i.e., $relevance_{TC_i} < 0.75$, goal violated). No test adaptation is necessary, however the SAS requires reconfiguration.
- **False positive:** Test has passed incorrectly as the SAS is experiencing an error (i.e., $relevance_{TC_i} \geq 0.75$, goal violated). Test adaptation is required to realign the test with the operating context. Moreover, the SAS requires reconfiguration to resolve the error.
- **False negative:** Test has failed incorrectly as the SAS is not experiencing an error (i.e., $relevance_{TC_i} < 0.75$, goal satisfied). Test adaptation is required to realign the test with the operating context.

Proteus performs coarse-grained adaptation by adapting test suites at run time to reduce the impact that run-time testing can have on an SAS. Specifically, an SAS engineer defines a default test suite for each operating context and Proteus derives new test suites at run time based on the results of run time testing. Test cases that are *true positives* are deactivated from execution, as they do not require re-validation. *True negatives*, *false positives*, and *false negatives* remain active and are continuously executed throughout SAS execution.

Upon detection of a false positive or false negative, Proteus invokes Veritas to adapt the test case parameter values as the test case has become invalid with respect to its operating context. Veritas executes the (1+1)-ONLINE evolutionary algorithm [33] to search for an optimal test case expected value with respect to the current operating context.

The adaptation capabilities of Proteus and Veritas provide two key benefits. First, coarse-grained adaptation can reduce the overall impact of executing test cases at run time by minimizing the amount of test cases executed. Second, fine-grained adaptation maximizes the overall relevance of executed test cases. Together, Proteus and Veritas enable run-time adaptive testing for an SAS.

IV. Experimental Results

This section describes the experimental setup and presents results from applying AutoRELAX, Fenrir, and Proteus with Veritas to the RDM application.

A. Experimental Setup

For this paper, the RDM application was implemented as a completely connected graph, where each node in the graph represented a data mirror and each edge represented a network link. To establish statistical significance, we performed 50 experimental treatments. For each treatment, the RDM network comprised between 15 and 30 data mirrors and was required to disseminate between 100 and 200 messages. Lastly, the RDM simulation was executed for 300 timesteps.

The RDM network was subjected to uncertainties within its configured parameters and in the environment in which it executed. Possible uncertainties included randomly dropped or delayed messages, network link failures, and random failures to data mirrors. The RDM network could reconfigure in terms of updates to the network topology or data mirror propagation parameters to mitigate uncertainty.

The RDM goal model comprised 23 goals, 2 of which were designated invariant and therefore were precluded from RELAXation. The remaining 21 goals were designated non-invariant and therefore could be RELAXed. The RDM test specification comprised 36 test cases in total, where 7 test cases monitored invariant conditions and were precluded from adaptation, and 29 test cases monitored non-invariant conditions and could be adapted by Veritas. The RDM simulation was also augmented to report system and environmental conditions not typically available to RDM sensors. Examples include monitoring internal variables that participate in the RDM reconfiguration engine and data structures that maintain all objects within the simulation environment.

We compared and evaluated each technique against a Control experiment in which no adaptation, in terms of the goal model, test suites, or test cases, was enabled. For each experiment, we conducted 50 treatments to establish statistical significance.

B. Experimental Results

First, we applied AutoRELAX to the RDM application to address requirements-based assurance. We compared AutoRELAXed goal models to a goal model that had been manually-RELAXed,[3] and to an unRELAXed goal model (i.e., the Control). Figure 2 presents boxplots of the average fitness values calculated for the RDM application, where fitness was calculated based on the functions presented in Equations 1 – 3. As is demonstrated by these results, AutoRELAX attains a significantly higher fitness value than can be found with the manually-RELAXed or Control goal model.

Based on these results, we conclude that AutoRELAX can generate better configurations of RELAX operators than can be found by a requirements engineer for this particular experiment. For the remainder of this study, we use the best RELAXed goal model (hereafter termed "$RDM_{RELAXed}$") found by AutoRELAX to enable requirements-based assurance.

We then applied Fenrir to $RDM_{RELAXed}$ to address code-based assurance. As such, Fenrir generated a suite of different execution traces in response to novel environments. We then selected a subset of the resulting RDM traces that exhibited the largest number of run-time errors for manual analysis, as the RDM could maximally experience upwards of several hundred errors depending on the operating context. Specifically, the RDM was attempting to send messages via

[1] For this paper, a relevance threshold of 0.75 was selected to indicate a passing test case.

[2] A goal is considered violated if its utility value equals 0.0.

[3] An SAS requirements engineer applied an optimal combination of RELAX operators to the RDM goal model.

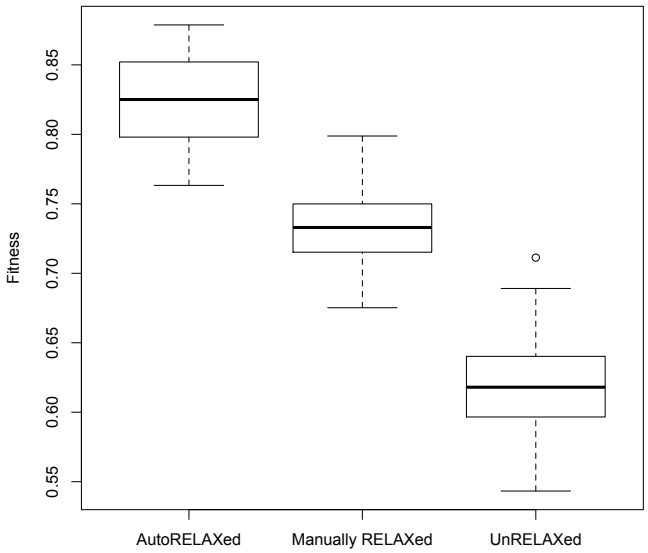

Fig. 2. Fitness values comparison between RELAXed and unRELAXed.

faulty data mirrors, thereby triggering an assertion error. We then rectified the problem uncovered by Fenrir by ensuring that a faulty data mirror is never selected for message distribution (hereafter termed "$RDM_{RELAXed-Fixed}$"). Upon re-executing $RDM_{RELAXed-Fixed}$, we found that, over 50 trials, the number of experienced run-time errors was reduced to 1 across all environments.

Finally, we addressed assurance at run time by performing adaptive online testing to $RDM_{RELAXed-Fixed}$ using Proteus and Veritas. For this experiment, we compared Proteus and Veritas test results to a Control in which no test suite or test case adaptation was performed. The objective of performing run-time test adaptation is to ensure that test cases remain relevant to the operating context, and moreover require that only relevant test cases are executed to reduce the burden on the testing framework. Test cases that monitor safety, failsafe, or invariant requirements are precluded from adaptation to ensure that invariant conditions are always checked for satisfaction.

Figure 3 presents boxplots of the averaged number of cumulative irrelevant test cases executed for each experiment, where an irrelevant test case is no longer applicable to the operating context (i.e., test case relevance $= 0.0$). These plots show that Proteus significantly reduces the amount of irrelevant test cases that were executed at run time (Wilcoxon-Mann-Whitney U-test, $p < 0.05$), indicating that Proteus can reduce the impact of run-time testing while maintaining the relevance of test suites.

Figure 4 presents the averaged number of cumulative false negative test cases executed for each experiment. Proteus significantly reduces the amount of experienced false negatives throughout SAS execution (Wilcoxon-Mann-Whitney U-test, $p < 0.05$), minimizing the need for test adaptation.

Figure 5 shows the averaged number of cumulative false positive test cases executed for each experiment. Proteus significantly reduces the amount of false positives that occurred throughout SAS execution (Wilcoxon-Mann-Whitney

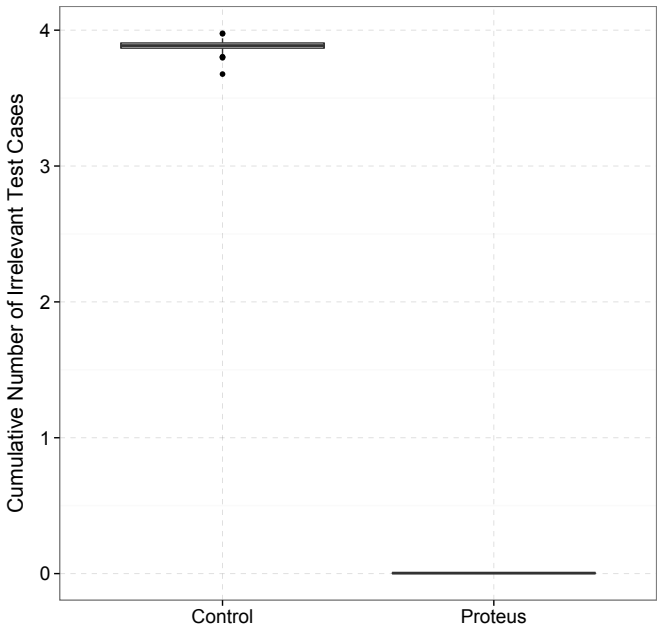

Fig. 3. Cumulative number of irrelevant test cases for each experiment.

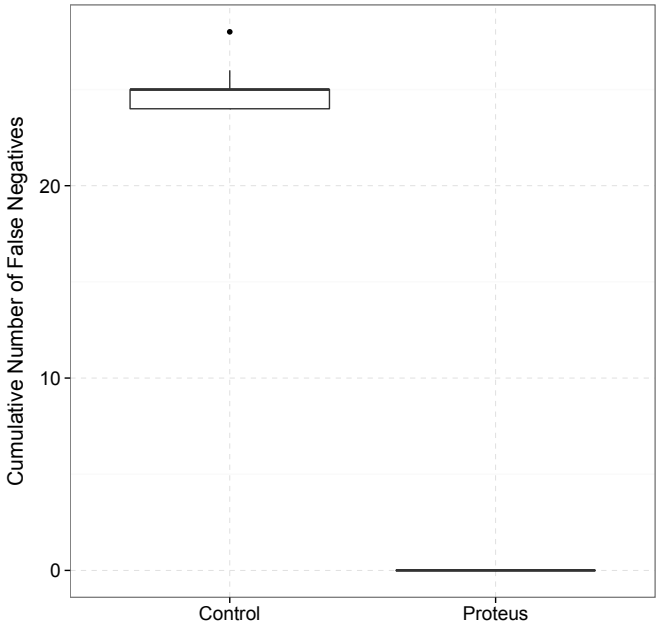

Fig. 4. Cumulative number of false negative test cases for each experiment.

U-test, $p < 0.05$), again minimizing the need for excess test adaptation.

Lastly, Figure 6 shows the average test case relevance calculations for Veritas-optimized test cases and Control test cases. Veritas significantly increases test case relevance (Wilcoxon-Mann-Whitney U-test, $p < 0.05$), indicating that performing online adaptation can realign test cases to changing environments.

The results presented in Figures 3 – 6 demonstrate that Proteus and Veritas can increase the overall relevance of test cases in response to uncertainty. Specifically, Proteus assists in

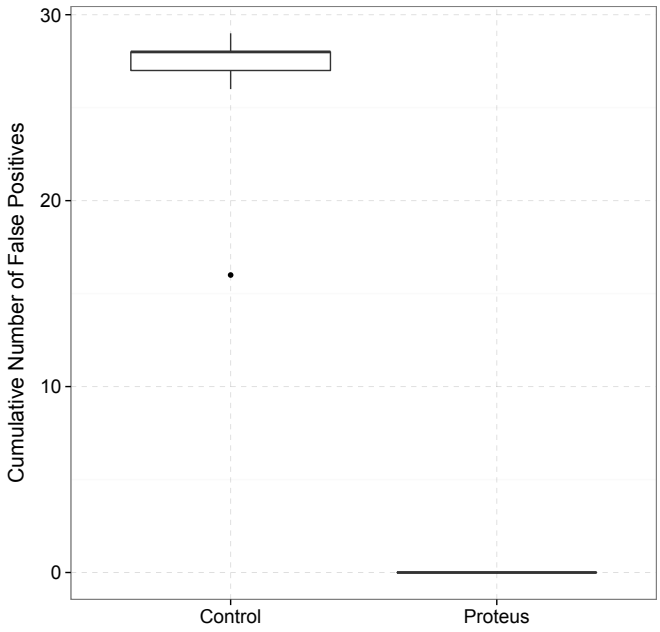

Fig. 5. Cumulative number of false positive test cases for each experiment.

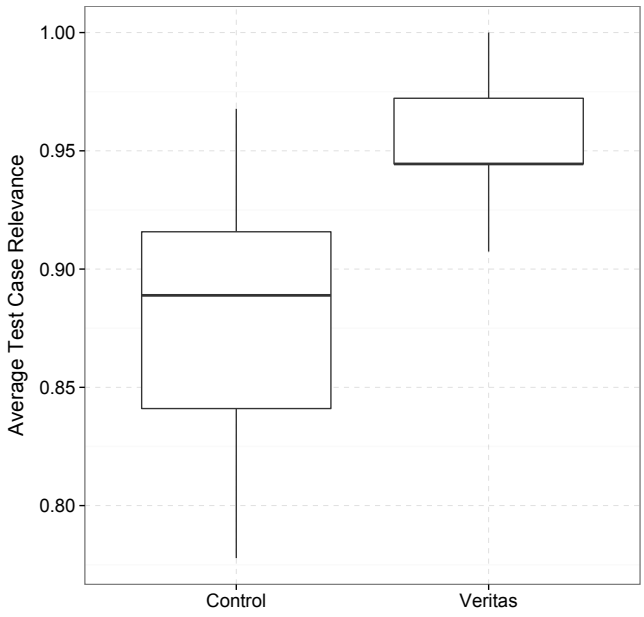

Fig. 6. Comparison of average relevance between Veritas and Control test cases.

reducing the number of unnecessary test cases executed, and Veritas ensures that test case parameter values adapt alongside the environment.

V. Conclusion

This paper has presented an empirical study in which AutoRELAX, Fenrir, Proteus, and Veritas were applied to an RDM application to mitigate uncertainty at different levels of abstraction. AutoRELAX leverages a genetic algorithm to search for an optimal combination of RELAX operators to introduce flexibility into a system goal model. Fenrir explores how different combinations of system and environmental parameters affect an SAS in its implementation using novelty search. Proteus and Veritas provide adaptive, run-time testing capabilities to provide online assurance, where Veritas uses an online evolutionary algorithm to optimize test case parameter values. We demonstrated how each of these techniques can enable assurance for an RDM application that must efficiently disseminate data throughout the network to maintain data availability. The RDM experienced uncertainty in the form of unexpected network link failures, dropped or delayed data messages, and random data mirror failures. Experimental results suggest that our techniques enhance overall assurance for the RDM. In particular, AutoRELAX produced an optimal combination of RELAX operators to apply to the system goal model to tolerate requirements-based uncertainty. Fenrir enabled the identification of a flaw in the RDM codebase. Proteus ensured that only relevant and necessary test cases were executed at run time, and Veritas ensured that test cases remained applicable to changing operating contexts.

Future directions for this work include investigating how other search-based techniques, such as simulated annealing or multiobjective optimization, can augment our existing heuristics. Moreover, we intend to introduce further automation into our techniques, such as automatically inserting logging statements into the SAS codebase for Fenrir, or automatically generating default test plans for Proteus based on an SAS requirements specification. Lastly, we intend to study the scalability of our approaches as applied to more complicated goal models specifying real-world applications.

Acknowledgment

This work has been supported in part by NSF grants CCF-0820220, DBI-0939454, CNS-0854931, CNS-1305358, Ford Motor Company, and General Motors. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, Ford, General Motors, or other research sponsors.

REFERENCES

[1] B. H. C. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, and et al., "Software engineering for self-adaptive systems: A research roadmap," in *Software engineering for self-adaptive systems*. Berlin, Heidelberg: Springer-Verlag, 2009, ch. Software Engineering for Self-Adaptive Systems: A Research Roadmap, pp. 1–26.

[2] J. Kephart and D. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41 – 50, jan 2003.

[3] P. McKinley, S. Sadjadi, E. Kasten, and B. H. C. Cheng, "Composing adaptive software," *Computer*, vol. 37, no. 7, pp. 56 – 64, july 2004.

[4] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for re for self-adaptive systems," in *Requirements Engineering Conference (RE), 2010 18th IEEE International*, 2010, pp. 95 –103.

[5] B. H. C. Cheng, P. Sawyer, N. Bencomo, and J. Whittle, "A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty," in *Proc. of the 12th International Conference on Model Driven Engineering Languages and Systems*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 468–483.

[6] N. Esfahani, E. Kouroshfar, and S. Malek, "Taming uncertainty in self-adaptive software," in *Proc. of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. ACM, 2011, pp. 234–244. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025147

[7] J. Whittle, P. Sawyer, N. Bencomo, B. H. C. Cheng, and J. Bruel, "Relax: Incorporating uncertainty into the specification of self-adaptive systems," in *Requirements Engineering Conference, 2009. RE '09. 17th IEEE International*, 31 2009-sept. 4 2009, pp. 79–88.

[8] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 4, no. 2, pp. 14:1–14:42, May 2009.

[9] K. Welsh and P. Sawyer, "Understanding the scope of uncertainty in dynamically adaptive systems," in *Proc. of the Sixteenth International Working Conference on Requirements Engineering: Foundation for Software Quality*, vol. 6182. Springer, 2010, pp. 2–16.

[10] A. J. Ramirez, E. M. Fredericks, A. C. Jensen, and B. H. C. Cheng, "Automatically relaxing a goal model to cope with uncertainty," in *Search Based Software Engineering*, G. Fraser and J. Teixeira de Souza, Eds. Springer Berlin Heidelberg, 2012, vol. 7515, pp. 198–212.

[11] E. M. Fredericks, B. DeVries, and B. H. C. Cheng, "Autorelax: Automatically relaxing a goal model to address uncertainty," *Empirical Software Engineering*, vol. 19, no. 5, pp. 1466–1501, 2014.

[12] E. M. Fredericks, A. J. Ramirez, and B. H. C. Cheng, "Validating code-level behavior of dynamic adaptive systems in the face of uncertainty," in *Search Based Software Engineering*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 8084, pp. 81–95.

[13] E. M. Fredericks and B. H. C. Cheng, "Automated generation of adaptive test plans for self-adaptive systems," in *Accepted to Appear in Proceedings of 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '15, 2015.

[14] E. M. Fredericks, B. DeVries, and B. H. C. Cheng, "Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty," in *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '14, 2014.

[15] J. Camara and R. de Lemos, "Evaluation of resilience in self-adaptive systems using probabilistic model-checking," in *Software Engineering for Adaptive and Self-Managing Systems.*, june 2012, pp. 53 –62.

[16] Y. Falcone, M. Jaber, T.-H. Nguyen, M. Bozga, and S. Bensalem, "Runtime verification of component-based systems," in *Proc. of the 9th international conference on Software engineering and formal methods*. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 204–220.

[17] A. Filieri, C. Ghezzi, and G. Tamburrelli, "A formal approach to adaptive software: continuous assurance of non-functional requirements," *Formal Aspects of Computing*, vol. 24, pp. 163–186, 2012.

[18] C. D. Nguyen, A. Perini, P. Tonella, and F. B. Kessler, "Automated continuous testing of multiagent systems," in *The Fifth European Workshop on Multi-Agent Systems (EUMAS)*, 2007.

[19] D. C. Nguyen, A. Perini, and P. Tonella, "A goal-oriented software testing methodology," in *Proc. of the 8th international conference on Agent-oriented software engineering VIII*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 58–72.

[20] A. J. Ramirez and B. H. C. Cheng, "Automatically deriving utility functions for monitoring software requirements," in *Proc. of the 2011 International Conference on Model Driven Engineering Languages and Systems Conference*, Wellington, New Zealand, 2011, pp. 501–516.

[21] C. Ghezzi, "Adaptive software needs continuous verification," in *Software Engineering and Formal Methods (SEFM), 2010 8th IEEE International Conference on*, Sept. 2010, pp. 3 –4.

[22] H. J. Goldsby, B. H. C. Cheng, and J. Zhang, "Models in software engineering," in *Models in Software Engineering*, H. Giese, Ed. Berlin, Heidelberg: Springer-Verlag, 2008, ch. AMOEBA-RT: Run-Time Verification of Adaptive Software, pp. 212–224.

[23] N. Qureshi, S. Liaskos, and A. Perini, "Reasoning about adaptive requirements for self-adaptive systems at runtime," in *Proc. of the 2011 International Workshop on Requirements at Run Time*, aug. 2011, pp. 16 –22.

[24] O. Wei, A. Gurfinkel, and M. Chechik, "On the consistency, expressiveness, and precision of partial modeling formalisms," *Information and Computation*, vol. 209, no. 1, pp. 20–47, 2011.

[25] G. Tamura, N. Villegas, H. Müller, J. Sousa, B. Becker, G. Karsai, S. Mankovskii, M. Pezzè, W. Schäfer, L. Tahvildari, and K. Wong, "Towards practical runtime verification and validation of self-adaptive software systems," in *Software Engineering for Self-Adaptive Systems II*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 7475, pp. 108–132.

[26] M. Ji, A. Veitch, and J. Wilkes, "Seneca: Remote mirroring done write," in *USENIX 2003 Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, June 2003, pp. 253–268.

[27] K. Keeton, C. Santos, D. Beyer, J. Chase, and J. Wilkes, "Designing for disasters," in *Proc. of the 3rd USENIX Conference on File and Storage Technologies*. Berkeley, CA, USA: USENIX Association, 2004, pp. 59–62.

[28] A. J. Ramirez, D. B. Knoester, B. H. Cheng, and P. K. McKinley, "Applying genetic algorithms to decision making in autonomic computing systems," in *Proceedings of the 6th international conference on Autonomic computing*, 2009, pp. 97–106.

[29] A. Van Lamsweerde and E. Letier, "Handling obstacles in goal-oriented requirements engineering," *Software Engineering, IEEE Transactions on*, vol. 26, no. 10, pp. 978–1005, 2000.

[30] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das, "Utility functions in autonomic systems," in *Proc. of the First IEEE International Conference on Autonomic Computing*. IEEE Computer Society, 2004, pp. 70–77.

[31] A. van Lamsweerde, *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[32] J. Lehman and K. O. Stanley, "Exploiting open-endedness to solve problems through the search for novelty," in *Proceedings of the Eleventh International Conference on Artificial Life (ALIFE XI)*. MIT Press, 2004.

[33] N. Bredeche, E. Haasdijk, and A. Eiben, "On-line, on-board evolution of robot controllers," in *Artifical Evolution*, ser. Lecture Notes in Computer Science, P. Collet, N. Monmarché, P. Legrand, M. Schoenauer, and E. Lutton, Eds. Springer Berlin Heidelberg, 2010, vol. 5975, pp. 110–121.