Towards Traceability Link Recovery for Self-Adaptive Systems

Reihaneh H. Hariri and Erik M. Fredericks
Oakland University
Rochester, MI 48309
{rhosseinzadehha, fredericks}@oakland.edu

Abstract

Self-adaptive systems (SAS) automatically mitigate environmental changes and unexpected system issues at run time by adapting towards optimal configurations that enable continual requirements satisfaction. The increasing proliferation of SASs presents engineering challenges that reflect issues experienced by non-adaptive systems, more specifically, ensuring that continuing assurance for software artifacts is provided. In particular, ensuring that requirements traceability links are appropriately managed at run time in SASs can be an error-prone procedure and may require significant effort from a requirements engineer. Natural language processing (NLP) techniques have been used to recover broken or missing traceability links efficiently between requirements and other artifacts, however, performing traceability link recovery can introduce significant overhead for SASs. Specifically, the state-space explosion of possible combinations of environmental states, system parameters, and expressed behaviors can lead to states in which no traceability link exists, thereby necessitating recovery. This paper proposes Adaptive Requirements Traceability (ART), a conceptual framework for handling traceability recovery in terms of SASs. We motivate this framework with an illustrative example in the networking domain.

Introduction

Self-adaptive systems (SAS) are often exposed to combinations of system and environmental conditions (i.e., operating contexts) that may prevent satisfaction of its requirements. To manage unexpected changes, an SAS self-adapts to new configurations at run time through continuous monitoring of itself and its environment (Kephart and Chess 2003), including adaptations to run-time software artifacts (Fredericks, DeVries, and Cheng 2014; Fredericks and Cheng 2015; Ghannem et al. 2016; Dömges and Pohl 1998). Given the relative complexity of an SAS's implementation, software maintenance becomes a highly-demanding task, resulting in the necessity of promoting typically design-time techniques to be first-class entities in the SAS framework. As such, this paper focuses on enhancing assurance, supported by natural language processing (NLP) techniques, by considering requirements traceability to be a run-time entity that self-adapts alongside an SAS (Cheng et al. 2009; Andersson et al. 2009).

Gotel and Finkelstein described requirements traceability as "the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases)" (Gotel and Finkelstein 1994). Traceability is used to describe relationships between related artifacts, such as requirement-to-test case links, code-to-code links, and code-to-document links, and is critical during design and run-time for both software development and maintenance. Insufficient traceability in software has been shown to be less maintainable, leading to failures and defects resulting from inconsistencies from artifacts to implementation (Parizi, Lee, and Dabbagh 2014; Dömges and Pohl 1998). Moreover, extracting appropriate information from artifacts to create a relevant traceability link is a highly challenging task (Arunthavanathan et al. 2016), however such a task is often necessary to enhance assurance.

This paper describes our conceptual approach to reconstructing traceability links, at run time, in the SAS domain. Specifically, we consider the situations in which traceability links are broken at run time as a result of a system reconfiguration. For instance, test cases and/or requirements may be disabled at run time as a result of an adaptation, resulting in a broken traceability link that must be resolved for the dangling artifact (i.e., either the test case or requirement left enabled). As such, our approach aims to leverage NLP techniques for automatically discovering and resolving possible traceability links at run time. To support this endeavor, we incorporate techniques from the NLP domain for use in recovering traceability links between different types of software artifacts (Borg, Runeson, and Ardö 2014; Arunthavanathan et al. 2016).

Therefore, we introduce Adaptive Requirement Traceability (ART), a conceptual framework for SASs to support automated maintenance of traceability links between artifacts as the system self-adapts, in particular, focusing on both extraction and discovery of links between requirements and test cases at run time. In this view, traceability link recovery can be considered a search problem that, based on the identified operating context, may require a different search technique to realize an appropriate recovery strategy.

The remainder of this paper is organized as follows. First, we motivate the need for automatic link recovery at run-time. Following, we describe the proposed ART framework. Lastly, we summarize this work and future directions.

Motivation

This section motivates the need for adaptive requirement traceability at run-time. In particular, we describe an illustrative example from the networking domain, the remote data mirroring (RDM) application, that is used to clarify challenges in maintaining traceability for SASs.

Remote data mirroring application. RDM is data protection technique for ensuring uninterrupted access to data and providing fault tolerance via data replication on physically-remote servers (Ji, Veitch, and Wilkes 2003; Keeton et al. 2004). The RDM application is represented as a large network of interconnected servers that has been modeled as an SAS. Specifically, the network can reconfigure itself in terms of its network topology (e.g., in case of a network link failure) and its data transmission protocols (e.g., from synchronous to asynchronous, based on monitored network traffic conditions).

Run-time Traceability Recovery Link Challenges. Broadly, traceability can refer to linking any type of software artifacts. In this paper, however, we focus on traceability between *requirements* and *test cases*. Traditional techniques establish traceability links between requirements and artifacts during software development. An SAS, however, can change frequently over time and must continuously monitor changes and react accordingly. Existing traceability recovery techniques may need to be extended to handle run-time concerns. Specifically, we consider the cases where new requirements and/or test cases may be added to and/or removed from the SAS online, resulting in a traceability framework that must automatically discover a link at run time. Moreover, requirements and test cases are considered to be first-class entities in the SAS framework to enable run-time reasoning and validation, and may be adaptive to support additional flexibility.

Requirements And Test Cases in RDM. For the RDM application, consider a *traceability matrix* that has been developed to track links between requirements and test cases. Such a matrix is often used to graphically demonstrate links. In a non-adaptive system, a traceability matrix is generally fixed in size. However, in the SAS domain it can be more common for requirements and/or test cases to adapt at run time as a result of the self-reconfiguration process. Such traceability adaptations can include adding, deleting, enabling, and disabling artifacts at run time. These changes pose severe challenges for traceability. In terms of a traceability matrix, an SAS may require a three-dimensional representation to demonstrate changes in traceability over time by representing requirements, test cases, and *system configurations* over time. Moreover, such a system would require a fully-automated process for link discovery and recovery.

For example, consider the RDM application. In response to unexpectedly-dense network traffic, several data mirrors have adapted to use asynchronous message processing. As such, any requirements and/or test cases that focus specifically on synchronous messages processing will no longer be relevant and may be disabled entirely *while the current operating context is in effect*. As such, any traceability links that exist where one or more artifacts are now disabled/removed must be updated online.

When and Where to Perform Traceability Recovery. We anticipate that traceability recovery would mimic the SAS MAPE-K loop (Kephart and Chess 2003). Specifically, the *monitoring* phase would determine that an operating context change has occurred, necessitating an update to the state of run-time artifacts. Following, the *analyzing* phase would determine that a traceability link is broken or otherwise needs to be recovered. The *planning* phase would consider tradeoffs in selecting an appropriate recovery strategy, and *executing* would then execute the recovery plan. In this regard, the SAS adaptation engine could be augmented to mitigate not only contextual uncertainty, but uncertainty within assurance.

How to Recover Traceability Links. Several automated and semi-automated techniques and tools exist for requirement traceability recovery at software development lifecycle (Falessi, Cantone, and Canfora 2013). We next present techniques that we consider relevant to the SAS domain, in particular, those that leverage NLP techniques to aid in parsing prose requirements and test cases.

Requirement Traceability Recovery Approaches using NLP and IR. Information Retrieval (IR) and NLP have been used for traceability recovery for many years. Trustrace has been proposed as a trust-based traceability recovery technique which combines data mining with IR to improve recovery accuracy (Ali, Guéhéneuc, and Antoniol 2013). Moreover, a comprehensive study of traceability link recovery indicates that algebraic IR models, such as latent semantic indexing (LSI) and the vector space model (VSM), are used more often than probabilistic models (Borg, Runeson, and Ardö 2014). Generally, NLP techniques will compute similarity between textual artifacts and use a similarity rank for candidate link consideration. VSM, by comparison, automatically calculates the degree of relevance between source and targets to identify links.

According to Falessi et al., NLP techniques for traceability recovery can be categorized into four main dimensions: algebraic model, term condition, weighting schema, and similarity metrics (Falessi, Cantone, and Canfora 2013; Falessi et al. 2017). Based on their presented results, a combination of techniques comprising VSM, raw frequency, Stanford, and cosine provide ideal results in terms of NLP-based link recovery. As a result, we propose to use a similar approach for recovery in the SAS domain.

Recovery Approaches using Search-based Techniques. Search-based techniques are often used for problems where there is no clear solution as a result of an enormous solution space, with heuristics including genetic algorithms (GA) (Holland 1992; Goldberg 1989), multi-objective optimization (e.g., NSGA-II) (Deb et al. 2002; Ghannem et al. 2016), hill climbing, and simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983). Recently, several studies have adopted search-based techniques in combination with recovery techniques (e.g. IR, ML, and NLP) to achieve better performance for recovering traceability links. For example, LDA-GA is an approach that applies GA to latent dirichlet allocation (LDA) to show a near-optimal configuration for traceability link recovery (Panichella et al. 2013). Another approach defines requirement traceability recovery as a multi-objective search problem and used NSGA-II for link recovery (Ghannem et al. 2017). To futher motivate this need, consider that many NLP techniques result in a ranked list of term pairs that require human intervention to select correct links. Given the complexity introduced by SASs, coupled with the need to move to the run-time domain, a search heuristic can be ideal to sift through ranked lists to automatically identify correct links.

Recovery Approaches using Machine Learning. Several studies have adopted ML classification to recover traceability links, some of which can automatically classify links (Mills 2017). One such approach, called LEarning and ANAlyzing Requirements Traceability (LeanArt), combines ML techniques with program analysis to recover traceability links. LeanArt used the multi-strategy learning approach (MLA) as it is crucial to find a learning algorithm that always carry an acceptable result (Grechanik, McKinley, and Perry 2007). ENRL is another approach that employs a supervised ML classifier (Bayesian model) combined with NLP techniques to estimate the number of remaining links in traceability recovery.

Adaptive Requirements Traceability

To the best of our knowledge, there are not many existing papers that deal with traceability link recovery for the SAS domain at run time (Ghannem et al. 2017), however there exists a strong need for this task to enhance run-time assurance, particularly in safety-critical systems where such assurance is necessary. Therefore, we propose ART as a conceptual approach for handling this problem. Specifically, we envision that ART will comprise a composition of different NLP techniques, including VSM, raw frequency, and cosine (the combination of which was previously discussed) to enable discovery of similarities between adaptive test cases and requirements at run time. These NLP techniques will be supported with search-based techniques, such as a GA, NSGA-II, or novelty search (Lehman and Stanley 2004) to provide the optimization aspect necessary for this work. Figure 1 presents a high-level overview of the ART framework, where rectangles are considered to be techniques and rounded rectangles to be actors/objects.

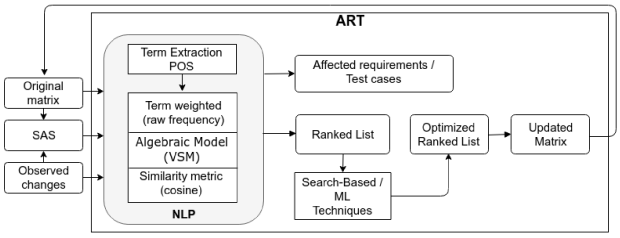

Figure 1: Conceptual ART framework.

Specifically, we will extend the standard traceability matrix to comprise the additional dimensions that an SAS generates (i.e., traceability links per generated configuration). Moreover inclusion of software test oracles and run-time requirements monitors internal to the SAS can be used to assist a search heuristic in also considering online satisfaction of key goals and objectives.

As shown in Figure 1, ART will leverage NLP (i.e., for term extraction, generating similarity metrics, etc.) to extract relevant terms from requirements and test cases, some of which may be newly added at run time, and compare these terms against an existing dictionary of terms. Following term extraction and identification, semantic similarity between artifacts will be evaluated using VSM to recover possible links. Cosine measuring will identify a similarity score between two artifacts, providing a ranking metric. Lastly, we envision that search heuristics can be used to optimize the generated ranking by examining the solution space presented by varying the NLP algorithm configurations or in providing additional flexibility in semantic search via similarity score calculation. This optimized list would result in an adaptive traceability matrix that would then be fed back into the SAS MAPE-K loop.

Run Time Considerations. Given the relative computational difficulty in performing ML, NLP, and search algorithms, promoting these tasks to run time is a non-trivial task. However, there exist approaches that have been proven to work at run time in these domains, and we anticipate that the inclusion of these techniques will be necessary.

In the search-based community, run-time performance is often gained by sacrificing searching power for speed. For example, the (1+1)-ONLINE EA might be used to provide evolutionary computation capabilities (Bredeche, Haasdijk, and Eiben 2010). In terms of ML, heavier tasks (such as training) may be run during design time, however there exist few papers on promoting a full ML suite to run time. When no run-time optimization is possible, we envision that an additional *traceability agent* could be introduced to perform heavy computations. This agent (either physical or logical) would be tasked with performing whichever computation is necessary to enable link recovery, and communicate those results back to the SAS's adaptation engine.

Discussion

This paper proposed a vision for traceability link recovery for adaptive software artifacts in the SAS domain. ART is a traceability recovery process that leverages a composition of NLP and search-based techniques to improve accuracy and performance of the recovery process. ART mimics the SAS MAPE-K loop to provide adaptive recovery alongside an SAS's adaptation engine. Moreover, considerations for elevating recovery to run time have been provided in terms of identified lightweight algorithms and possibilities for offloading tasks to additional agents.

Future work for this research includes a full evaluation of ART in terms of SAS applications in multiple domains. We also plan to examine how different search-based techniques can affect recall and precision of NLP-based automated recovery procedures.

Acknowledgements

This research has been supported in part by NSF grant CNS-1657061, the Michigan Space Grant Consortium, the Comcast Innovation Fund, and Oakland University. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Oakland University or other research sponsors.

References

Ali, N.; Guéhéneuc, Y.-G.; and Antoniol, G. 2013. Trustrace: Mining software repositories to improve the accuracy of requirement traceability links. *IEEE Trans. on Software Engineering* 39(5):725–741.

Andersson, J.; De Lemos, R.; Malek, S.; and Weyns, D. 2009. Modeling dimensions of self-adaptive software systems. *Software engineering for self-adaptive systems* 27–47.

Arunthavanathan, A.; Shanmugathasan, S.; Ratnavel, S.; Thiyagarajah, V.; Perera, I.; Meedeniya, D.; and Balasubramaniam, D. 2016. Support for traceability management of software artefacts using natural language processing. In *Moratuwa Engineering Research Conference (MERCon)*, 18–23.

Borg, M.; Runeson, P.; and Ardö, A. 2014. Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. *Empirical Software Engineering* 19(6):1565–1616.

Bredeche, N.; Haasdijk, E.; and Eiben, A. 2010. On-line, on-board evolution of robot controllers. In Collet, P.; Monmarché, N.; Legrand, P.; Schoenauer, M.; and Lutton, E., eds., *Artificial Evolution*, volume 5975 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg. 110–121.

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

Deb, K.; Pratap, A.; Agarwal, S.; and Meyarivan, T. 2002. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on* 6(2):182 –197.

Dömges, R., and Pohl, K. 1998. Adapting traceability environments to project-specific needs. *Communications of the ACM* 41(12):54–62.

Falessi, D.; Di Penta, M.; Canfora, G.; and Cantone, G. 2017. Estimating the number of remaining links in traceability recovery. *Empirical Software Engineering* 1–32.

Falessi, D.; Cantone, G.; and Canfora, G. 2013. Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques. *IEEE Transactions on Software Engineering* 39(1):18–44.

Fredericks, E. M., and Cheng, B. H. C. 2015. Automated generation of adaptive test plans for self-adaptive systems. In *Proceedings of 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS '15.

Fredericks, E. M.; DeVries, B.; and Cheng, B. H. C. 2014. 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*, SEAMS '14.

Ghannem, A.; Hamdi, M. S.; Kessentini, M.; and Ammar, H. H. 2016. Search-based requirements traceability recovery. In *Proceedings of SAI Intelligent Systems Conference*, 156–171.

Ghannem, A.; Hamdi, M. S.; Kessentini, M.; and Ammar, H. H. 2017. Search-based requirements traceability recovery: A multi-objective approach. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, 1183–1190.

Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc.

Gotel, O. C., and Finkelstein, C. 1994. An analysis of the requirements traceability problem. In *Proceedings of the First International Conference on Requirements Engineering*, 94–101.

Grechanik, M.; McKinley, K. S.; and Perry, D. E. 2007. Recovering and using use-case-diagram-to-source-code traceability links. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 95–104.

Holland, J. H. 1992. *Adaptation in Natural and Artificial Systems*. Cambridge, MA, USA: MIT Press.

Ji, M.; Veitch, A.; and Wilkes, J. 2003. Seneca: Remote mirroring done write. In *USENIX 2003 Annual Technical Conference*, 253–268. Berkeley, CA, USA: USENIX Association.

Keeton, K.; Santos, C.; Beyer, D.; Chase, J.; and Wilkes, J. 2004. Designing for disasters. In *Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, 59–62. Berkeley, CA, USA: USENIX Association.

Kephart, J., and Chess, D. 2003. The vision of autonomic computing. *Computer* 36(1):41 – 50.

Kirkpatrick, S.; Gelatt, C. D.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science* 220(4598):671–680.

Lehman, J., and Stanley, K. O. 2004. 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.

Mills, C. 2017. Automating traceability link recovery through classification. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 1068–1070.

Panichella, A.; Dit, B.; Oliveto, R.; Di Penta, M.; Poshyvanyk, D.; and De Lucia, A. 2013. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *Proceedings of the 2013 International Conference on Software Engineering*, 522–531.

Parizi, R. M.; Lee, S. P.; and Dabbagh, M. 2014. Achievements and challenges in state-of-the-art software traceability between test and code artifacts. *IEEE Transactions on Reliability* 63(4):913–926.