Improving the Reliability of Mobile Software Systems through Continuous Analysis and Proactive Reconfiguration

Sam Malek [1] Roshanak Roshandel [2] David Kilgore [1] Ibrahim Elhag [1]

[1] *Dept. of Computer Science*
George Mason University
{smalek, ckilgor1, ielhag}@gmu.edu

[2] *Dept. of Comp. Sci. & Software Engr.*
Seattle University
roshanak@seattleu.edu

Abstract

Most of the current software reliability analysis approaches are geared to traditional desktop software systems, which are relatively stable and static throughout their execution. In this paper, we present a framework targeted at mobile computing domain that addresses the uncertainties associated with the reliability analysis in this setting. Moreover, the framework's architecture-centric reliability estimates are leveraged to improve the runtime reliability of the system through dynamic architectural reconfiguration.

1. Introduction

Mobile, distributed, and pervasive software systems are characterized by their highly dynamic configuration, unknown operational profile, and fluctuating execution conditions. Most existing software reliability analysis approaches (e.g., [3][4][5]) are geared to traditional desktop software systems, which are relatively stable and static throughout their execution. Therefore, assessing the reliability of mobile software systems requires new principles, models, and tools that incorporate the underlying uncertainties associated with such systems.

At the same time, since often an accurate estimation of the mobile software system's execution context is not available at design-time, it is infeasible to determine an optimally reliable architectural configuration for the software system prior to its deployment. Therefore, a run-time reconfiguration of the software system may be necessary to improve its reliability.

In this paper, we provide an overview of a framework that aims to alleviate the challenges associated with the analysis and improvement of software reliability in mobile and dynamic settings. The framework adopts an architecture-centric approach that furnishes reliability predictions at the level of system's architectural constructs. Unlike previous approaches that rely primarily on the system's past history (i.e., execution log) to predict its future reliability, our analysis considers information from a variety of sources, such as the software architectural models, the system's execution profile, domain experts' knowledge, and other relevant contextual information. The reliability predictions are leveraged by the framework to find an optimal software configuration with respect to reliability, which is then effected at run-time.

The remainder of the paper is organized as follows: Section 2 outlines the challenges of assessing and improving the reliability of mobile software systems; Section 3 provides an overview of our framework; and finally Section 4 describes our approach in realizing the framework. The paper concludes with an outline of our future work.

2. Challenges

Assessing and improving the reliability of mobile systems is a challenging multi-faceted problem. In particular, we elaborate on 5 key challenges.

Challenge 1: *Impact of Context on Reliability***.** In a mobile, distributed, and pervasive setting, failures may occur due to either internal software design and implementation defects, or external hardware and network problems (limitations). While the focus of our work is on the former, we need to account for the fact that the manifestation of a coding defect in the form of software failure also heavily depends on the run-time characteristics external to the software system, which is known as *context* [1]. For instance, a software defect

may result in failure only when the network throughput is reduced to a certain threshold, potentially due to the overflow of the communication buffers. There is a wide spectrum of contextual properties that may influence the reliability of a software system: changes in the hardware platform (e.g., depleted battery, memory usage), fluctuations in the network (e.g., network drop outs, bandwidth variations), unanticipated usage, timing of operations, and so on.

Challenge 2*: Impact of Dynamism on Reliability.* To deal with contextual changes, mobile software systems are often developed to be adaptive. The adaptations to a system's architectural configuration alters its behavior, which in turn impacts its reliability. For instance, consider a hypothetical software system shown in Figure 1a, which consists of three software components running in the same Operating System (OS) process (e.g., JVM process). This configuration may be efficient, since the components are able to share resources (e.g., global variables) within the same process. On the other hand, if any of the software components fails, the whole system is vulnerable to failure, as failure of one component that crashes the process can easily propagate to other components. Figure 1b shows an alternative configuration of the system that by the same reasoning is less efficient but more resilient to failure. Allocation of software components to OS processes is an example of crucial configuration decisions that directly impact the reliability of a software system. Therefore, in the mobile computing domain, where architectural reconfiguration occurs frequently, the ability to assess its impact on reliability and determine the most reliable (i.e., *optimal*) configuration is critical.

Challenge 3: *Difficulty of Predicting Reliability.* The ability to determine an optimally reliable configuration for a software system hinges on the ability to accurately estimate the reliability of the software system in its future operation. The majority of existing reliability analysis approaches (recent surveys [3][4][5]) primarily rely on software monitoring and assume that a software system's past reliability to be indicative of its future reliability. While this assumption may be reasonable for the traditional desktop systems that are relatively stable, it does not apply to the domain of mobile systems. For instance, the fact that a mobile software system has not crashed over the past few hours does not necessarily imply that it won't crash in the next few hours. This is particularly due to ever-changing context and architecture of such systems.

Challenge 4*: Fine-Grained Reliability Prediction.* To be able to estimate the reliability of an architectural configuration, we need to be able to assess the reliability of its constituents. For example, in the scenarios depicted in Figure 1, it is possible to determine the optimal system configuration only if we can first assess the reliability of its software components: If the components are extremely reliable, Figure 1a is the preferable configuration, which as mentioned earlier is more efficient; If the components are highly unreliable, Figure 1b is preferred as it is more resilient to failure.

Challenge 5*: Scalable Online Analysis.* The majority of reliability estimation approaches that rely on traditional stochastic-based approaches (e.g., Markov Chains, Bayesian networks) are known to be computationally expensive [3][4][5]. Fine-grained analysis at the level of the system's components, instead of the system as a whole, further exacerbates the problem. The scalability of analysis is extremely important in the mobile computing domain, where such analysis would need to be performed online at runtime.

3. Overview of the Framework

Figure 2 depicts an overview of the framework. Software architectural models form the core of the framework. A running software system that realizes those architectural models resides on top of a context-aware execution platform (*context-aware middleware* [1][7]), which is intended for use in mobile computing domain. The middleware provides facilities for monitoring the running software, as well as its external execution context. Moreover, the middleware provides facilities for enacting changes to the running software. As shown in Figure 2, the reliability-driven reconfiguration framework consists of three conceptual software components, which are deployed as meta-level software components on top of the middleware. These components collaborate to monitor, assess, and improve the system's reliability as follows.

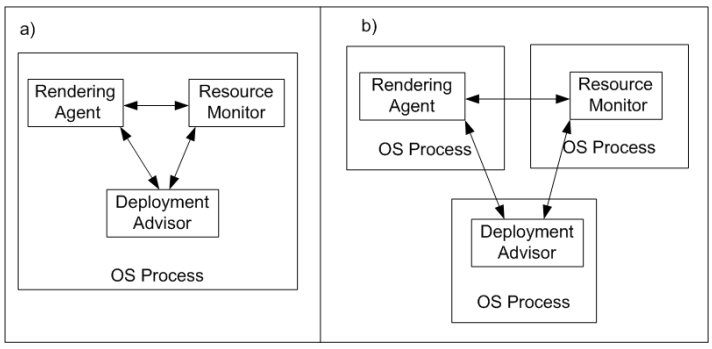

Figure 1. Two configurations for a hypothetical system: (a) a more efficient but less reliable configuration, and (b) a less efficient but more reliable configuration.

At design time and before the system's implementation is complete, an initial set of *Architecture-Based Reliability Model*s are developed (depicted in Figure 2*)*. These models are used during the design and development to assess a variety of design choices and to serve as predictors for the future reliability of the system. Unlike traditional architectural models, they embody contextual properties necessary for reliability analysis of mobile systems. As will be described below, these models are expected to be updated and refined at runtime.

The *Architecture-Based Reliability Model*s are then used by the *Reliability Analyzer* to estimate the reliability of the software system in terms of its constituent components and connectors. These fine-grained reliability estimates are utilized by the *Configuration Selector* to determine the most suitable architectural configuration. Once the *Configuration Selector* has selected a suitable initial configuration, the middleware is used to deploy the software system onto a set of mobile hosts. An underlying assumption is that the middleware provides support for deployment, execution, monitoring, and adaptation of a software system in terms of its architectural constructs (e.g., components, connectors, and configuration). We have developed such a middleware in our previous work [7].

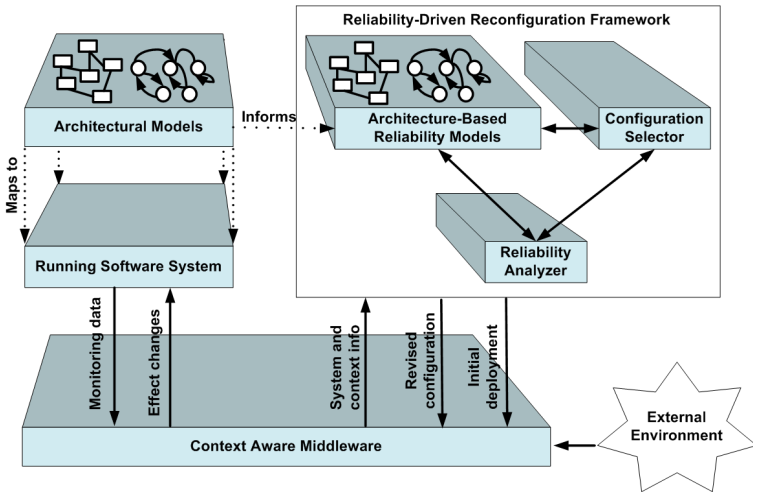

Figure 2. Reliability-Driven Reconfiguration Framework.

At run-time, the middleware monitors the software system for information that will be used to refine the initial design-time reliability predictions. This information is obtained from multiple sources, such as monitoring the internal software properties (e.g., frequency of failures, exceptions, and service requests), external properties (e.g., network fluctuations, battery charge), changes in the structure of the software (e.g., disconnection of components due to network drop outs, off-loading of components due to drained battery), and so on. Since the monitored data represents the most recent operational, structural, and contextual profile of the system's execution, they can be used to assess the system reliability more accurately. Note that unlike previous approaches we do not rely solely on monitored data. Instead, we incorporate monitored data with existing data obtained at design-time, which our preliminary experiments has shown to produce more accurate results; these two sources of information are highly complementary.

The enriched run-time models are then used again by the *Reliability Analyzer* to provide refined reliability estimates for the system's architectural constituents. In turn, the refined estimates are used by the *Configuration Selector* to compare and contrast alternative architectural configurations with respect to reliability. Finally, if a better configuration is found the *Configuration Selector* directs the middleware to effect the necessary changes.

4. Approach

Below we provide an outline of the approach that we have taken in the construction of the framework. We have structured the discussion in terms of the 5 key challenges identified in Section 2, which are presented slightly in a different order and in some cases together for the exposition purposes.

Challenges 1 and 3: We address the uncertainties associated with the reliability analysis in this setting by incorporating various design-time and run-time sources of information. In fact, our previous experience on early design-time reliability prediction has corroborated that reliability estimates are more meaningful when different information sources are analyzed and interpreted in parallel [2][9]. In preliminary experiments we observed a similar trend with run-time reliability estimation of mobile systems. We consider the following design-time sources of information: system engineers' knowledge, results obtained from the behavioral simulation of the architectural models, data available in the requirements documents, and the execution logs of functionally similar artifacts (e.g., previous versions of the system). Our design-time reliability estimates are continuously refined based on additional contextual parameters that become available at run-time: system's internal properties (e.g., frequency of failures, exceptions, and service requests), and system's external properties (e.g., network fluctuations, available battery charge, changes in location). The consideration of various sources of information distinguishes our work from the

previous techniques that rely solely on the system's execution log in estimating reliability [3][4][5].

Challenges 4 and 5: As mentioned in the previous section, we have adopted an architecture-centric approach that inherently allows for compositional and hierarchical analysis of a software system's reliability. The overall reliability of a system is determined in terms of the reliability of its sub-systems, which are in turn determined in terms of the underlying components and connectors. Compositionality enables representing the system's characteristics as an aggregation of the detailed characteristics of its subsystems, while hierarchy enables abstracting away details in favor of focusing on the larger picture. These two properties in tandem make our analysis fine-grained, scalable, and efficient for execution at runtime. Moreover, we include the notion of architectural styles in our reliability models. Architectural styles pose constraints on the valid configuration and interaction of components, and can be used to (1) determine the component dependencies, and (2) reduce the space of valid configurations. In turn, styles help to reduce the complexity of reliability analysis, as well as the process of finding a good architecture.

Challenge 2: In the mobile computing domain, changes in context force the software system to adapt a variety of functional and non-functional properties for optimality. We have leveraged dynamic learning algorithms, such as Hidden Markov Models (HMMs) and Dynamic Bayesian Networks (DBNs), to build a highly adaptive reliability model that can be dynamically updated at runtime to account for the changes in the architecture. Both HMMs and DBNs are instances of Graphical Models [6], a hybrid modeling approach that combines probability theory and graph theory. Inference techniques applicable to graphical models enable leveraging observations about a system to update (or infer) and answer probabilistic queries about the system. In this case, the *probabilistic queries* are aimed at obtaining the probability of failures (inversely related to system reliability), while *observations* correspond to a large set of information obtained from various source of information in the manner described earlier. The core reliability model built based on architectural knowledge leverages these observations to respond to stochastic queries about the system. In turn, this is continuously used at run-time to optimize the architecture of the running mobile software system with respect to reliability.

5. Conclusion

Existing software reliability assessment techniques are not applicable to mobile systems that are innately dynamic and unpredictable [3][4][5]. We have presented a framework that addresses the challenges of analyzing the reliability of mobile software systems by adopting an architecture-centric approach, which takes into consideration various sources of information, including the system's contextual characteristics. When the execution conditions of a mobile software system change, the framework can be used to first assess and then improve its reliability through dynamic reconfiguration of its architecture. In our future work, we intend to evaluate the approach in the context of a real-world mobile software system that has been developed with an external industrial collaborator [8]. The work described here is part of an ongoing activity, as part of which we are developing tool-support for realizing the framework on top of a mobile middleware platform (Prism-MW [7]).

6. Acknowledgments

This work is partially supported by grant CCF-0820060 from the National Science Foundation.

7. References

[1] L. Capra, et al. CARISMA: Context-Aware Reflective Middleware System for Mobile Applications. *IEEE Transactions on Software Engineering*, Vol 29, 2003.

[2] L. Cheung, R. Roshandel, et al. Early Prediction of Software Component Reliability. *Int'l Conf. on Software Engineering*, Leipzig, Germany, May 2008.

[3] S. Gokhale. Architecture-Based Software Reliability Analysis: Overview and Limitations. *IEEE Trans. on Dependable and Secure Computing*, Vol 4, Jan 007.

[4] K. Goseva-Popstojanova, et al. Architecture-Based Approaches to Software Reliability Prediction, Int'l Journal of Computer & Mathematics with Applications, 46(7), October 2003.

[5] A. Immonen, et al. Survey of reliability and availability prediction methods from the viewpoint of software architecture. *Software and Systems Modeling*, Jan 2007.

[6] M. I. Jordan. Learning in Graphical Models, *MIT Press*, Nov. 1998.

[7] S. Malek, M. Mikic-Rakic, and N. Medvidovic. A Style-Aware Architectural Middleware for Resource Constrained, Distributed Systems. *IEEE Transactions on Software Engineering*, March 2005.

[8] S. Malek, et al. Reconceptualizing a Family of Heterogeneous Embedded Systems via Explicit Architectural Support. *Int'l Conf. on Software Engineering*, Minneapolis, Minnesota, May 2007.

[9] R. Roshandel, et al. Estimating Software Component Reliability by Leveraging Architectural Models. *Int'l Conf. on Software Engineering*, Shanghai, China, May 2006.