Assessing Software Reliability at the Architectural Level

Apostolos Zarras and Valerie Issarny
INRIA
Domaine de Voluceau - Rocquencourt - 78153 Le Chesnay - France
email: {Apostolos.Zarras, Valerie.Issarny}@.inria.fr

Abstract

Modeling software architectures has been proved to be beneficial for facilitating the interaction among different stake-holders involved in the software development process, separating design concerns, promoting software reuse and evolution. However, it is still not clear whether software architecture description is compulsory for analyzing qualities like performance and reliability. Trying to clear out the previous question, this paper justifies the necessity of software architectural modeling towards assessing software reliability.

1 Introduction

Research results in the field of software architecture showed that modeling software architectures allows to clearly separate different design concerns, it promotes software reuse and evolution [4]. Finally, modeling software architectures eases the interaction among different stake-holders involved in the software development process. However, it has been stated [4] that software reliability and performance can not be assessed at the architectural level. In this position paper we argue that software reliability and performance are mainly assessed at the architectural level. Regarding performance, a preliminary modeling method was proposed in [7]. This position paper focuses on the reliability aspect. More specifically, Section 2 details the steps of a method, proposed for modeling software reliability. As it comes out, those steps are mainly performed at the architectural level and formal software architecture description is essential towards accomplishing them. Section 3 summarizes the outcome of this position paper.

2 Modeling Reliability

Formally, software reliability is the probability that a software system is performing successfully its required functions for the duration of a specific mission profile [6]. Calculating reliability is, however, not a new challenge. Several techniques were proposed in the past years for addressing this issue [6]. Although different from each other, those techniques can be put under the common basis of a general reliability modeling method based on software architecture description. The steps of the method are detailed in the remainder of this section, first for the simple case of a software system that can not be repaired, and then for the more complex case of a repairable one.

2.1 Modeling non-repairable software systems

Based on the definition of reliability given at the beginning of this section, the first step that must be performed towards modeling the reliability of a non-repairable software system is to model the system itself. More specifically, the structure and the behavior of the system must be precisely described in a way that is well understandable for all the different actors that participate in the overall development and evaluation process. To achieve the former, any architecture description language would do. However, the software architecture community recently identified UML as a candidate for describing software architectures in a standard way. UML *class diagrams* can be used to describe components and connectors, *object diagrams* can be used to describe one or more runtime configurations of the particular system. Moreover, *state-chart diagrams*, *collaboration diagrams* can be used for delineating the behavior of the overall system, and the behaviors of the individual architectural elements that constitute it. Specific examples of such UML descriptions can be found in [2, 3].

To exemplify the method presented in this position paper, we use the example of a software system that consists of a CORBA server providing some services. The example system further comprises a legacy system (e.g. IBM CICS) used by the CORBA server towards serving service requests coming from external CORBA clients. Since the legacy system does not belong to the CORBA *"world"*, communication between the CORBA server and the legacy cannot be achieved directly. To deal with the previous problem, two replicas of a CORBA *facade* are used. The facade exports a CORBA interface that matches the specification of the legacy system. Based on this structure the CORBA server diffuses a request to the facade components which in turn call, at-most-once, the corresponding functionality provided by the legacy. Figure 1, gives an overview of the runtime configuration of the system.

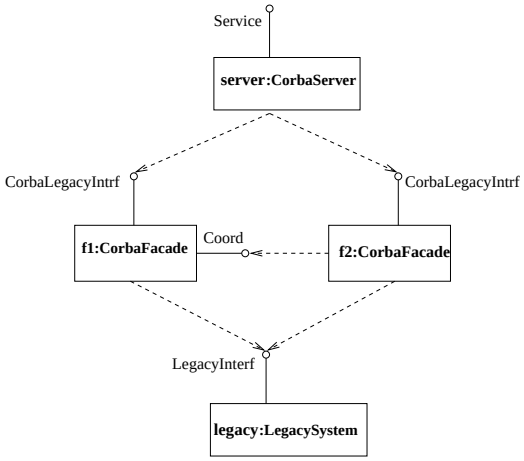

Figure 1: Example: Software system configuration in UML

After the system's architecture is well-defined, the next step towards modeling reliability is to give a mission statement [5]. A mission statement identifies success criteria, which in other words is an abstract description of the behavior expected by the system. In UML, the previous can be achieved though the definition of *use case diagrams* describing the system behavior as manifested to external users of the system. Following our example scenario, the system is supposed to provide services to external users through the use of a legacy system. Figure 2, gives the corresponding use case diagram.

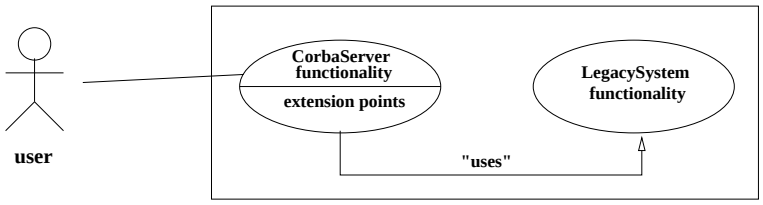

Figure 2: Example: Abstract system behavior in UML

The third step of the reliability modeling method comprises mapping the expected, abstractly defined, system behavior into a concrete one supported by a subset of the architectural elements that constitute the software system. In UML, the previous step corresponds to the construction of a *collaboration diagram* (or possibly a *sequence diagram*), describing a specific interaction between objects that make up the system configuration, which satisfies the required behavior. In terms used by the software architecture community, we could say that the behavior described in the collaboration diagram refines the one described in the use case diagram. Furthermore, it should be noted that the collaboration diagram originates from the object diagram that describes the overall system configuration. Figure 3 depicts a concrete behavior provided by the software system used in our example. This behavior satisfies the abstract one given in Figure 2.

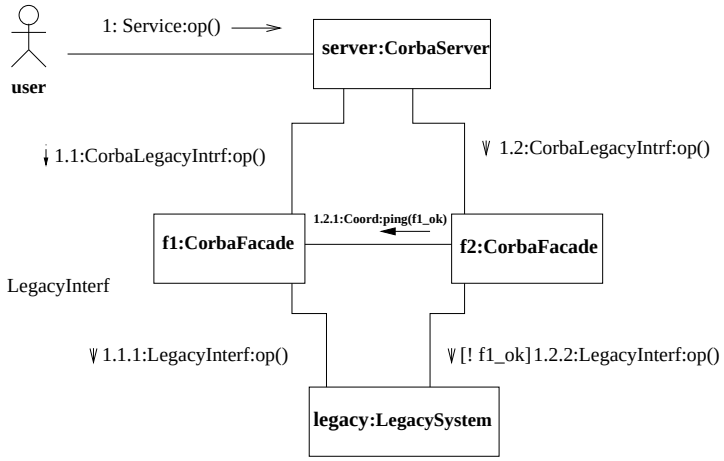

Figure 3: Example: Concrete system behavior in UML

Hence, based on the collaboration diagram it is now possible to identify the architectural elements that are necessary towards accomplishing the specified mission. Following, the next step towards modeling reliability is to approximate the failure rate, Mean Time To Failure (MTTF), and reliability for each one of those elements. The typical technique for accomplishing the previous task comprises testing for a rather long period the various architectural elements needed to accomplish a mission and monitoring potential failures. The traces resulted from monitoring are then analyzed so as to determine a function that properly describes the overall distribution of failures. Finally, based on the failure distribution, deriving the failure rate, MTTF, and reliability is straight-forward.

At this point one may argue that a significant part of the work needed for assessing reliability is performed during this last step which involves testing and does not have to do with modeling the architecture of the system. However, when looking a bit closer to this process, it is clear that the failure distribution approximation is basically the approximation of the failure behavior of the elements that make up a system. Consequently this relates to the semantics of those elements and

it is still part of modeling the architecture of the given system. Hence, the failure behavior of an architectural element comprises its failure rate, MTTF and reliability parameter.

Assuming that in the general case a failure is an event generated by an architectural element, describing it in UML can be done with the use of a *"signal classifier"*. More specifically, a *"signal" class* describing failures generated by a particular architectural element would include attributes that correspond to the failure rate and MTTF. Going back to our example scenario, Figure 4, gives the definition of the *signal classes* that describe the failure behaviors of the individual components that constitute the overall system architecture.

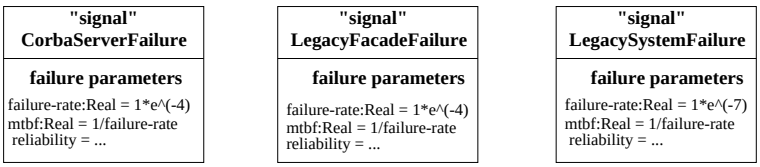

Figure 4: Example: Modeling component failures in UML

The very last step of the reliability modeling method is to calculate the overall reliability of the system. The collaboration diagram describing the concrete system behavior is significant in order to accomplish this step. Given the collaboration diagram it is straight-forward how to derive a reliability block diagram (RDB) and apply RBD analysis [5] so as to calculate the overall system reliability. In our example, from the collaboration diagram shown in Figure 3, it is obvious that the system works correctly as long as the CORBA server and the legacy work correctly, and if either of the facades works correctly. Hence, the reliability of the overall system is:

$$R_{System} = R_{CorbaServer} * (R_{Facade1} + R_{Facade2}) * R_{LegacySystem}$$

2.2 Modeling repairable software systems

Modeling reliability of a repairable software system is a slightly more complicated case. In principal, a software system can be repaired by the removal of faulty architectural elements or by the substitution of faulty architectural elements with spared ones. Hence, to model reliability in this case it should be possible to model that the configuration of a software system changes dynamically.

In the field of software architecture, the configuration of a system is usually modeled as a set of component instances that interact through connector instances. To capture the fact that some of those components are operational while others have failed we slightly modify this definition; a configuration comprises two different sets of components: the first set contains component instances that operate normally; the second set contains failed component instances. Based on the previous definition, modeling a component failure amounts to moving the component that fails from the first set into the second one. In the same spirit, modeling the substitution of a faulty component with a spare one amounts to adding the spare component into the set of operational components and removing the faulty component from the set of the failed components. Hence, repairing a system can be interpreted as changing the state of the system's configuration. Consequently, modeling a repairable system imposes that the underlying modeling method should enable specifying transitions that take the system configuration from one state to another one by removing or substituting faulty components. UML provides *state-chart* diagrams, allowing to describe a sequence of states that an object goes through during its lifetime in response to received stimuli. In our case, the object is the configuration of the system and the stimuli are failures modeled as signals.

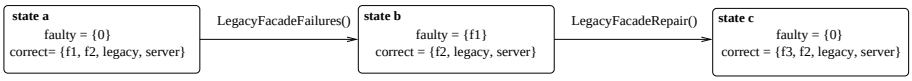

Figure 5: Example: Software system configuration state chart in UML

Returning to the example scenario, let us assume that the system configuration is initially in a state where all components work as expected. Figure 5 gives a subset of all possible state transitions that result from the initial state (state (a) in the figure). In particular, state (b) is the state where the legacy facade f1 has failed. The (a) to (b) transition is caused by a signal of type CorbaFacadeFailure (see Figure 4) which happens with a rate of $1 * e^{-4}$ failures per sec. State (c) is the state we get when the failed facade f1 is substituted by a spare f3. Summarizing, modeling all possible configuration state transitions the way previously proposed, enables deriving, in a straight forward manner, a Markov chain and applying Markov analysis to calculate the reliability of the overall system [1].

3 Summary

This position paper claims that assessing reliability can be achieved at the architectural level. To justify this claim this paper proposed a method for modeling reliability of a software system that can be either repairable or not. In both cases, it is clear that the subsequent steps of the proposed method are realized at the architectural level. Moreover, accomplishing each one of those steps strongly requires the formal description of the system's software architecture.

References

[1] R. Butler and W. Ricky. The SURE Approach to Reliability Analysis. *IEEE Transactions on Reliability*, 41(2), June 1992.

[2] C. Hofmeister, R.L. Nord, and D. Soni. Describing Software Archtiecture with UML. In *Proceedings of of the 1st Working Conference on Software Architecture*, pages 145–159. IFIP, 1999.

[3] N. Medvidovic and D.S. Rosenblum. Assessing the Suitability of a Standard Design Method. In *Proceedings of of the 1st Working Conference on Software Architecture*, pages 161–182. IFIP, 1999.

[4] N. Medvidovic and R. Taylor. Separating Fact from Fiction in Software Architecture. In *Proceedings of the 3rd International Software Architecture Workshop*, pages 105–108, November 1998.

[5] NASA. Reliability Block Diagrams and Reliability Modeling. Technical report, NASA Glenn Research Center, May 1995. http://www-osma.lerc.nasa.gov/rbd/rbdtut.html.

[6] M. Shooman. *Software Engineering - Design/Reliability/Management.* McGraw-Hill, 1983.

[7] B. Spitznagel and D. Garlan. Architecture-Based Performance Analysis. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, June 1998.