Science of Computer Programming 97 (2015) 426–457

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Reliability prediction for component-based software systems: Dealing with concurrent and propagating errors

Thanh-Trung Pham [a,*], Xavier Défago [a], Quyet-Thang Huynh [b]

[a] School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), Nomi, Ishikawa, Japan
[b] School of Information and Communication Technology, Hanoi University of Science and Technology, Hanoi, Viet Nam

ARTICLE INFO

Article history:
Received 3 June 2013
Received in revised form 7 March 2014
Accepted 26 March 2014
Available online 15 May 2014

Keywords:
Reliability modeling and prediction
Component-based software systems
Error propagation
Software fault tolerance mechanisms
Concurrently present errors

ABSTRACT

One of the most important quality attributes of a software system beyond its functional attributes is its reliability. Techniques for predicting reliability of a software system based on the design models can help software architects in evaluating the impact of their design decisions on the system reliability. This can help to make the system more reliable and avoid costs for fixing the implementation. However, existing reliability prediction approaches for component-based software systems are limited in their applicability because they either neglect or do not support modeling explicitly several factors which influence the system reliability: (i) error propagation, (ii) software fault tolerance mechanisms, and (iii) concurrently present errors. In this paper, we present a reliability modeling and prediction approach for component-based software systems that considers explicitly these reliability-relevant factors. Our approach offers a reliability modeling schema whose models are automatically transformed by our reliability prediction tool into Markov models for reliability predictions and sensitivity analyses. We evaluate our approach in two case studies with reliability predictions and sensitivity analyses. Via these two case studies, we demonstrate its applicability in supporting design decisions.

1. Introduction

To meet the increasing requirements for software support from many different areas, software systems become increasingly complex. In this situation, to assure the system reliability, i.e. its ability to deliver its intended service to users, many classes of techniques in software reliability engineering have been deployed throughout the development process. One of such classes of techniques is the class of reliability prediction techniques based on the design models. These techniques can help to make the system more reliable by assisting software architects in evaluating the impact of their design decisions on the system reliability. This can help to save costs, time, and efforts significantly by avoiding implementing software architectures that do not meet the reliability requirements.

However, existing reliability prediction approaches for component-based software systems suffer from the following drawbacks and therefore are limited in their applicability and accuracy. In essence, these drawbacks are consequences of the assumption that components fail independently and each component failure leads to a system failure, which is common to most existing reliability models for component-based software systems [1].

* Corresponding author.
E-mail addresses: thanhtrung.pham@jaist.ac.jp (T.-T. Pham), defago@jaist.ac.jp (X. Défago), thanghq@soict.hut.edu.vn (Q.-T. Huynh).

http://dx.doi.org/10.1016/j.scico.2014.03.016
0167-6423/© 2014 Elsevier B.V. All rights reserved.

1.1. Ignoring error propagation

According to Avizienis et al. [2], an error is defined as the part of a system's total state that may lead to a failure. The cause of the error is called a fault. A failure occurs when the error causes the delivered service to deviate from correct service. The deviation can be manifested in different ways, corresponding to the system's different failure types. For example, two failure types that could be defined are content failures (the content of a system service's output deviates from the correct one) and timing failures (the delivery time of a system service deviates from the correct one).

Errors can arise because of internal faults. For example, a bug in the code implementing a component is an internal fault. This fault causes an error in the internal state of the component if the code is executed. Errors can arise because of external faults. For example, an erroneous input appears as an external fault to a component and propagates the error into the component via its interface. Errors can also arise because of both internal faults and external faults, e.g. an erroneous input (an external fault) is also the application of an input (the activation pattern) to a component that causes the code with a bug (an internal fault) of the component to be executed.

However, not all errors in a component lead to component failures. A component failure occurs only when an error in a component propagates within the component up to its interface. Similarly, not all component failures lead to system failures. A component failure in a component-based system is an error in the internal state of the system. This error leads to a system failure only when it propagates through components in the system up to the system interface.

During this propagation path, an error can be detected,[1] and therefore stops from propagating, e.g. an erroneous input is detected by error detection of components. An error can be masked, e.g. an erroneous value is overwritten by the computations of component services before being delivered to the interface. An error can be transformed, e.g. a timing failure received from another component service may cause the current component service to perform computations with outdated data, leading to the occurrence of a content failure. An error can also be concurrently present with another error, e.g. a content failure received from another component service is also the activation pattern that causes the current component to perform unnecessary computations with corrupted data, leading to the concurrent presence of a content failure and a timing failure.

It is possible to see that the reliability of a component-based software system, defined as the probability that no system failure occurs, is strongly dependent on the error propagation path. The challenge of analyzing the reliability of a component-based software system becomes even more significant when the system embodies parallel and fault tolerance execution models. A parallel execution model has multiple components running in parallel, resulting in many concurrent error propagation paths. A fault tolerance execution model has a primary component and backup components, and the order of their executions is highly dependent on their error detection and error handling. This results in many different error propagation paths.

As an example, in a parallel execution model, an error in the input for the components running in parallel may be masked by the computations of a certain number of components while the computations of the other components may transform the error into multiple errors of different failure types, leading to a set of multiple errors of different failure types in the output of the execution model. As another example, in a fault tolerance execution model, an error of a certain failure type in the input for the primary component and backup components may be transformed into an error of other failure type by the primary component without being detected, leading to an error in the output of the execution model without activating the backup components.

Although error propagation is an important element in the chain that leads to a system failure, many approaches (e.g. [3–8]) do not consider it. They assume that any error arising in a component immediately manifests itself as a system failure, or equivalently that it always propagates (i.e. with probability 1.0 and with the same failure type) up to the system interface [9]. On the other hand, approaches that do consider error propagation (e.g. [9,10]) typically only consider it for a single sequential execution model. Since modern software systems often embody not just a single sequential execution model, but also parallel and fault tolerance execution models to achieve multiple quality attributes (e.g. availability, performance, reliability), ignoring the consideration of error propagation for these two latter execution models makes these approaches no more suitable for modeling complex software systems with different execution models.

1.2. Ignoring software fault tolerance mechanisms

Software Fault Tolerance Mechanisms (FTMs) are often included in a software system and constitute an important means to improve the system reliability. FTMs mask faults in systems, prevent them from leading to failures, and can be applied on different abstraction levels (e.g. source code level with exception handling, architecture level with replication) [11]. Their reliability impact is highly dependent on the whole system architecture and usage profile. For example, if a FTM is never executed under a certain usage profile, its reliability impact is considered as nothing.

Analyzing the reliability impact of FTMs becomes apparently a challenge when they are applied at architecture level, in a component-based software system because: (1) FTMs can be employed in different parts of a system architecture,

[1] Software fault tolerance mechanisms, if any, can then provide error handling for the detected error.