Component Airbag: a Novel Approach to Develop Dependable Component-Based Applications

Roberto Pietrantuono
Dipartimento di Informatica e Sistemistica - Università degli Studi di Napoli Federico II
Via Claudio 21, 80125
Naples, Italy
roberto.pietrantuono@unina.it

ABSTRACT

The increasing use of "commercial off-the-shelf"(COTS) components in safety critical scenarios, arises new issues related to the "dependable" use of third-party software in such contexts. The characteristics of these components, designed for a generic use, are such to make unpredictable the effects of their use whenever they are integrated in the entire system. The author's Ph.D project aim at proposing an approach to improve dependability of COTS based application, which consists of the following phases: i) each component is stimulated by proper workloads in order to learn the failure behavior; ii) from failure behaviors, the component failure model is defined; and iii) once the failure model is known for each component, the "component airbag" is thus created, i.e. a container able of exploiting the failure model in order to monitor and prevent the component from failing. An existent literature analysis, regarding the more used dependability assessment and improvement strategies, is also presented.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Reliability*

General Terms

Reliability, Verification

Keywords

Dependability, COTS

1. INTRODUCTION

Nowadays, the trend of using COTS components to develop software products in several contexts, is increasing in a substantial way, mainly due to their ability to reduce development costs and time. Generally, when we talk about COTS, we mean a reusable software module, developed and delivered by third-party organizations; it is usually provided without source code and, sometimes, with incomplete specifications. Third-party organizations are usually not aware of environment in which their product will operate; so it is generally hard to integrate their components, in a dependable way, in critical applications. Moreover, existing strategies and techniques about dependability enhancement are either focused on only one aspect, as assessment ones, or they are insufficient and still far from providing acceptable results. Therefore, making dependable such components is a challenging issue. Author's research project focuses on defining and implementing a novel approach that, given a COTS component, aims to build a sort of "airbag", i.e. a special container that knows its failing behavior in the new environment and prevent it from failing. The proposed approach consists of three phases: i) a workload-driven stress analysis is conducted to understand the component failing behavior, including the interactions between the considered component and its environment; ii) starting from the failure behavior, a failure model is defined for each component being evaluated, which classifies the manifested failures and their root causes; and iii) finally, acquired knowledge is exploited to monitor component at run time, to prevent the component from going in a critical and failure prone state. The rest of this paper is organized as follows. Background and related works are analyzed in section 2. The first paragraph of such section illustrates those issues related to dependability assessment; the second one is about dependability improvement techniques. The objectives and approaches of author's Ph.D project are outlined in section 3.

ESEC/FSE'07, September 3–7, 2007, Cavtat near Dubrovnik, Croatia.
ACM 978-1-59593-812-1/07/0009.

2. BACKGROUND AND RELATED RESEARCH

2.1 Dependability Assessment

2.1.1 Main Approaches

Most common approaches on the dependability assessment are Field Failure Data Analysis and Dependability Benchmarking. Such approaches outline procedures to follow for characterizing the system under study with quantitative and qualitative measurements. FFDA approach allows assessing dependability by means of failure data [7]. It acts at operational phase, through a field analysis of system behavior under real workloads. Since FFDA acts independently by knowledge of internal structure of the system, it can be used for COTS systems. On the other hand, for components that fail rarely, this approach has to be substituted

or, however, integrated by other techniques. As for dependability benchmarking, its goal is to specify a procedure for measurements evaluation related to system behavior in the presence of faults [9]. The trickiest issue that this technique has to address is the definition of faultloads, from which the accuracy of entire measurement process depends on.

2.1.2 Main Techniques

Previous approaches consider a combined use of several techniques, such as simulation and modeling. As for the latter, almost every attempt to model system dependability uses probabilistic models. The most used model is the Bayesan model, such as the one proposed in [16], followed by Markov chain model [5]. The main difference, among these works, is the choice of parameters and data to feed the model. By classifying them according to this criterion, we obtain: i) models that use "historical" data [12]; ii) models that use data derived from testing [13] or iii) models that use information derived from design [8]. As for simulation techniques, the most used one is fault injection [6]. Also testing techniques are used for dependability assessment, due to their ability of providing useful data for estimations of parameters related to dependability (like fault tolerant mechanism coverage), as well as for refinement of models obtained through modeling techniques.

2.2 Dependability Improvement

Works oriented to the dependability improvement, generally operate on several development cycle stages, from system design phase (mainly, proposes of models) to testing phase, until the operational phase. Making a coarse classification, we may distinguish approaches that use monitoring to react to faults (as fault tolerance [3, 17]) or to act proactively to them (as adaptation strategies [20, 4, 18]). Several approaches aim to build the expected behavior model in order to compare it to the actual behavior. Finally, there are works aiming at component testability improvement and other works that use testing and analysis techniques.

2.2.1 Behavioral Model

There is a great deal of research studies that proposes new approaches aiming at construction of behavioral models of system under study. Some of them, suggest of modeling system behavior at design stage [14], but in most cases reverse engineering techniques are proposed, aiming at constructing component behavior model through system execution observation, as [10]. These techniques are useful when software is poorly documented (as with COTS). Most used approach consists of observing external component interactions and of using a finite state machine to build a model [19]. However, this has some limitations: i) it does not give any information about component internal state; ii) it is a heuristic technique, so that it only detects a behavior which deviates from the output of the model, but this does not mean that detected deviants are error manifestations; and iii) most of them do not take account of concurrency issues.

2.2.2 Testability improvement

Approaches which improve component testability, focuses on development phase. They aim at building self-testable component, adding characteristics which can be either exploited at testing phase, or at run time, for component monitoring, such as [1] and [11]. These approaches differ substantially from others, because they refer to component development phase and because they are strongly oriented to unit testing. This difference is also their main weakness, because most of problems in using COTS are due to their integration in the new environment, rather than reliability of the component itself.

2.2.3 Testing and Analysis

If we consider COTS component dependability improvement, all testing and analysis techniques that assume source code knowledge are not of interest. A number of works, assuming knowledge of specifications, are oriented to their formalization, in order to automatize test cases generation or to ease the analysis process. Among these, most of them use models like finite state automatons or variation of it [2]. A separate trend are stress or robustness testing techniques. Robustness testing aim at stressing interface of the system, providing it unforeseen inputs and observing the reaction [15]. Looking at COTS-based system, it has the advantage of considering the system as black box, requiring only an interface knowledge. On the other hand, it suffers the limitation of not considering system state.

3. OUR PROPOSAL

As previously described, none of proposed strategies own by itself necessary characteristics to set, in any aspect, the problem of reliable use of COTS components. Dependability assessment approaches are useful to provide precious information about system behavior and failure modes, but they are limited to evaluation of dependability attributes. On the other hand, dependability improvement strategies often impose very stringent constraints to provide acceptable solutions, like source code knowledge, state-less or small dimension components: respect to their goals, they are still far from providing acceptable results. But by combining them, it is possible to enhance significantly the dependability of COTS-based applications. Following this idea, the author's project focuses on a particular component paradigm, namely the CORBA Component Model (CCM)[1], in order to abstract and generalize its fundamental steps versus other component environments. The proposed approach aims to the creation of an "airbag" tailored for each third-part component which have to be integrated in the system. This airbag will act as a "improved" container which is able to monitor interactions between component and environment, in order to prevent the transition of the component from a working state to a critical one. The proposed approach consists of three steps, depicted in figure 1. The first two ones take place at component integration phase; the third one operates at run time. The goal of the first step is to learn the component failure behavior, through a workload-driven stress testing. This phase focuses on the construction of a component failure model. In this stage, we operate according to the following steps: i) selection of suitable workloads to stress the component; ii) component stressing test both via its interactions toward other components, and between component and container, in order to simulate interactions with environment; iii) failure model definition; iv) correlating each derived class of failures to behavioral pattern observed before the failures. The derived failure modes for that

[1]Object Management Group; "The Corba Component Model", www.omg.org

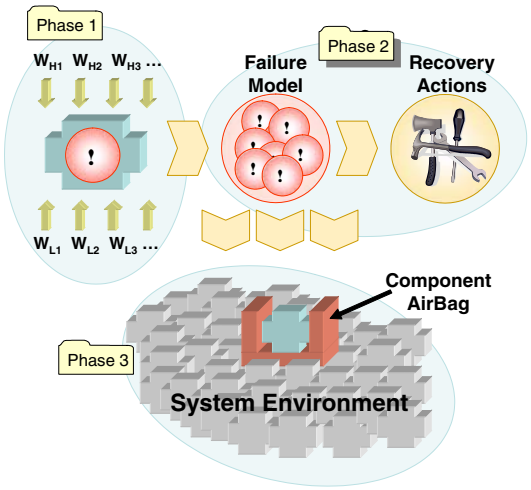

Figure 1: Overview of the proposed approach.

component with their correlation, along with generic parameters that are effective for any component, will be "criticism" parameters for the state of that component. These parameters will allow the construction of a tailored airbag, fully aware of behavior of its component under stress conditions. Moreover, for each derived failure mode, we will define specific recovery actions. Once the model has been defined, the second phase, at run time, will be proactive respect to classified failures; i.e., the built airbag will have to recognize, during execution, behavioral patterns that, with a certain probability, might led the component to a certain failure. After this, airbag will apply on time the appropriate predefined recovery actions. If such actions are not defined, the system is simply led to a safe state. Acting in this way, the created airbag will operate as an error detection and treatment module, opening the way to new on-line diagnosis strategies.

4. CONCLUSIONS

Author's Ph.D project focuses on the definition and implementation of a novel integrated approach, which considers several dependability enhancement strategies in order to exploit advantages of each one. It, considering CCM as case study, aims at creating an airbag tailored for each component to be integrated, able of acting as a wrapper aware of component failure modes, and able of applying, at run time, reactive and proactive actions to make the component a self healing one.

5. REFERENCES

[1] S. Beydeda. Self-testability in unit testing. In *Proc. of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, 2005.

[2] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *Proc. of the 21st international conference on Software engineering (ICSE '99)*, 1999.

[3] P. de C. Guerra, C. Rubira, A. Romanovsky, and R. de Lemos. A fault-tolerant software architecture for cots-based software systems. *SIGSOFT Software Engineering Notes*, 2003.

[4] J. Georgas. Knowledge-based architectural adaptation management for self-adaptive systems. In *Proc. of the 27th international conference on Software engineering (ICSE '05)*, 2005.

[5] H. L. Guen, R. Marie, and T. Thelin. Reliability estimation for statistical usage testing using markov chains. In *Proc. of the 15th International Symposium on Software Reliability Engineering (ISSRE'04)*, 2004.

[6] M. Hsueh, T. Tsai, and R. Iyer. Fault injection techniques and tools. *IEEE Computer*, 1997.

[7] R. K. Iyer, Z. Kalbarczyk, and M. Kalyanakrishnam. Measurement-Based Analysis of Networked System Availability. *Performance Evaluation Origins and Directions Journal, LNCS*, 1769, 2000.

[8] P. Johannessen, C. Grante, A. Alminger, U. Eklund, and J. Torin. Hazard analysis in object oriented design of dependable systems. In *Proc. of the International Conference on Dependable Systems and Networks (DSN '01)*, 2001.

[9] H. Madeira, K. Kanoun, J. Arlat, D. Costa, Y. Crouzet, M. D. Cin, P. Gil, and N. Suri. Towards a framework for dependability benchmarking. *Proc. of the 4th European Dependable Computing Conference (EDCC'02)*, 2002.

[10] L. Mariani and M. Pezzè. Behavior capture and test: Automated analysis of component integration. In *Proc. of International Conference on Engineering of Complex Computer Systems*, 2005.

[11] E. Martins, C. Toyota, and R. Yanagawa. Constructing self-testable software components. In *Proc. of the International Conference on Dependable Systems and Networks (DSN '01)*, 2001.

[12] N. Nagappan, T. Ball, and B. Murphy. Using historical in-process and product metrics for early estimation of software failures. In *Proc. of the 17th International Symposium on Software Reliability Engineering (ISSRE '06)*, 2006.

[13] H. Okamura, H. Furumura, and T. Dohi. On the effect of fault removal in software testing - bayesian reliability estimation approach. In *Proc. of the 17th International Symposium on Software Reliability Engineering (ISSRE '06)*, 2006.

[14] R. Pettit and H. Gomaa. Modeling behavioral design patterns of concurrent objects. In *Proc. of the 28th international conference on Software engineering (ICSE '06)*, 2006.

[15] C. Shelton, P. Koopman, and K. Devale. Robustness testing of the microsoft win32 api. In *Proc. of the International Conference on Dependable Systems and Networks (DSN'00)*, 2000.

[16] H. Singh, V. Cortellessa, B. Cukic, E. Gunel, and V. Bharadwaj. A bayesian approach to reliability prediction and assessment of component based systems. In *Proc of the 12th International Symposium on Software Reliability Engineering (ISSRE'01)*, 2001.

[17] J. De Alemida, S.R. Rota. Run-time monitoring for dependable systems: An approach and a case study. 2004.

[18] G. Valetto and G. Kaiser. Using process technology to control and coordinate software adaptation. In *Proc. of the 25th International Conference on Software Engineering (ICSE '03)*, 2003.

[19] L. Wendehals and A. Orso. Recognizing behavioral patterns at runtime using finite automata. In *Proc. of the International workshop on Dynamic systems analysis (WODA '06)*, pages 33–40, 2006.

[20] J. Zhang and B. Cheng. Model-based development of dynamically adaptive software. In *Proc. of the 28th international conference on Software engineering (ICSE '06)*, 2006.