Error Propagation in the Reliability Analysis of Component based Systems

Petar Popic, Dejan Desovski, Walid Abdelmoez, Bojan Cukic
Lane Department of Computer Science and Electrical Engineering
West Virginia University
{petarp, desovski, rabie, cukic}@csee.wvu.edu

Abstract

Component based development is gaining popularity in the software engineering community. The reliability of components affects the reliability of the system. Different models and theories have been developed to estimate system reliability given the information about system architecture and the quality of the components. Almost always in these models a key attribute of component-based systems, the error propagation between the components, is overlooked and not taken into account in the reliability prediction.

We extend our previous work on Bayesian reliability prediction of component based systems by introducing the error propagation probability into the model. We demonstrate the impact of the error propagation in a case study of an automated Personnel Access Control System. We conclude that error propagation may have a significant impact on the system reliability prediction and, therefore, future architecture-based models should not ignore it.

1. Introduction

COTS-based (commercial of-the-shelf) software products and COTS-based software development are becoming increasingly popular in the software engineering community. The objective is to design components that can be simply installed and cooperate well with existing system components. Current operating systems, Internet browsers and office product groups are just few among the numerous examples of the products that fit into the COTS software category. Many software packages bought by typical computer users belong to this category.

The frequent use of COTS components makes them suitable for numerous applications. Besides their cost advantage, if these components are reliable, the reliability of the system in which they are assembled is expected to be high too.

COTS-based development raises new issues in software architectures research: ensuring the reliability of the interaction between components, correlations between system failures and failures of particular components, etc. Component–based software reliability engineering can help us gather and analyze information that is relevant for solving quality related problems. Component failure rates have a major role in reliability estimation process of the system as a whole. The problem we face is how to combine these pieces of the reliability puzzle together.

Several methods are available for estimating and analyzing the reliability of component-based software systems. These methods can be broadly classified in two groups: the system level approaches and component-based approaches. System level approaches threat system as a whole and they do not include valuable component information in the analysis [7]. In component-based approaches, information about the quality of components is crucial for the reliability analysis.

According to Popstojanova et. al. [8], we can classify component-based models as state-based models, path-based models and additive models. The state based models [9, 10] utilize control graphs created using Discrete Time Markov Chains, Continuous Time Markov Chains or Semi-Markov Processes to characterize the application architecture. These models presume component and interface failure rates which can also vary over the time [9]. The path-based models in their analysis have similar steps as the state-based models observing the various executions paths and their frequencies that particular program can exercise [8]. These models may require an executable program and complete source code for testing in order to create necessary path information. The path information is further combined with the failure behavior to predict reliability. Additive models do not explicitly analyze application architecture. These models pay more attention on the failure data of individual components and then estimate reliability by combining this information. The methodology behind

Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE 2005)
1071-9458/05 $20.00 © 2005 **IEEE**

additive models assumes that component reliability can be modeled by a non-homogenous Poisson process, thus permitting that system reliability be represented as a weighted sum of component reliabilities [8].

Singh et. al. [1] proposed a Bayesian approach to reliability prediction and assessment of component-based systems. In this paper, we extend their work by adding a very important architectural attribute that is often overlooked in order to simplify reliability modeling: the error propagation probability in software architectures. This attribute has to be considered in order to achieve accurate reliability estimation of any system. Error propagation probability represents the probability that an erroneous state generated in one component propagates to other components, instead of being always successfully detected and masked at its source.

The outline of the paper is as follows. In Section 2 we present related work on how error propagation can be computed for a given program. Section 3 talks about the Bayesian based reliability estimation and assessment of component-based systems. We extend this model by incorporating the error propagation probability in Section 4. Section 5 presents the case study where we demonstrate the effect of the component failure rates and error propagation on the overall system reliability. The conclusions are presented in Section 6.

2. Error Propagation in Software

It is known that a software fault and the resulting error in one component can be propagated to the other interacting components causing their failures. Component failures are, therefore, seldom independent. However, architectural software reliability models assume this independence. Error propagation is important from developer's perspective, since the estimates can be used to find the most affected components, and take actions in improving the reliability of the system by applying error detection or error recovery mechanisms, such as wrappers [14]. There have been several studies on estimating the error propagation between software components or modules [4, 14, 15]. Error propagation models have been validated through fault injection experiments, followed by the comparison of the actual error propagation results with the ones predicted by the model.

In this paper, we adopt the error propagation model of Nassar et. al. [3]. This model emerged in a project that investigates wide range of architectural attributes such as *change propagation probability,* and *requirements propagation probability* [11]. One of the reasons for selecting Nassar's model is its reliance on UML design artifacts, the notation we use for the reliability modeling too. However, we need to stress that other methodologies for calculating the error propagation probability can be utilized in our reliability model as well, as described in Section 4.

2.1. Computing Error Propagation

In [4], error propagation is calculated from the information available early on in the lifecycle - in the system design stage. The information about the structure and semantics of the code is not available at this stage, but the information about the flow of control and data within components and between components exists in appropriate UML artifacts.

Nassar et. al. define the *error propagation probability (EP)* from component A to component B as follows:

$$EP(A,B) = P([B](x) \neq [B](x') \mid x \neq x').$$

[*B*] is the function of component *B* which captures all the outcomes of executing *B* (the state of *B* and the outputs of B), *x* is an instance of the connector *X* used for communication between components *A* and *B*. Finally, *EP*(*A, B*) represents the probability that a fault and associated error state in *A* will be propagated to *B*. In other words, the outcome of the execution of *B* will be changed as the result of the error that occurred in *A*.

Having the architecture of *N* components, *EP* is an $N \times N$ matrix, where the entry in row *A* and column *B* is the error propagation probability from component A to component *B*. *EP*(*A, A*) is equal to *1*, because if a component reached an error state then it is assumed to remain in the error state [4]. If a component implements some error correcting mechanisms, this assumption would need to be changed. *EP*(*A, B*) is a conditional probability because the probability that an error propagates from *A* to *B* is calculated under the condition that component *A* actually transmits a message to component *B*.

The authors of [3] claim that the next formula expresses the error propagation probability between *A* and *B* derived from the probabilities of messages being sent from *A* to *B* and their state information:

$$EP(A \to B) = \frac{1 - \sum_{x \in S_B} P_B(x) \sum_{y \in S_B} P_{A \to B}\left[F_x^{-1}(y)\right]^2}{1 - \sum_{v \in V_{A \to B}} P_{A \to B}[v]^2},$$

where:

- $P_B(x)$ is the probability of observing component *B* in state *x* (S_B is the set of states of the component *B*),
- $P_{A\to B}[F_x^{-1}(y)]$ is the probability of the transmission of a message (from *A* to *B*) that causes component *B* to transit from state *x* to state *y*,
- $P_{A\to B}[v]$ is the probability that message *v*, $v \in V_{A\to B}$, is sent over the connector from *A* to *B*.

Further, $F_x^{-1}(y) = \{v \in V_{A\to B} \mid F_x(v) = y\}$. P_B and $P_{A\to B}$ represent the assumed or estimated probability distributions on the set of states, and the set of messages, respectively. $P_B(x)$ is related to the operational profile of the system, that may be represented by annotated UML diagrams.

Assuming that states of *B* and the messages being passed from component *A* to component *B* through the connector all have an equal probability, then the error propagation formula above can be simplified into

$$EP(A \to B) = \frac{1 - \frac{1}{|S_B||V_{A\to B}|^2}\sum_{x\in S_B}\sum_{y\in S_B}\left|F_x^{-1}(y)\right|^2}{1 - \frac{1}{|V_{A\to B}|}}.$$

The authors of [3] further define *Unconditional Error Propagation*, which we find particularly useful for reliability modeling. Denoted by *E*(*A, B*), it is defined as the probability that an error propagates from *A* to *B*, without being conditioned by an actual occurrence of a message from *A* to *B*. *E*(*A, B*) is calculated from the *Transmission Probability Matrix, T*(*A, B*). Each entry in *T*(*A, B*) indicates the probability of the connector from *A* to *B* being activated during a canonical execution. The purpose of matrix *T* is *"to reflect the variance in frequency of activations of different connectors during a typical execution"* [4]. The *Unconditional Error Propagation* is obtained from

$$E(A,B) = EP(A \to B) \cdot T(A,B),$$

T(*A, B*) is calculated from the number of messages between components *A* and *B* in the given UML system model divided by the number of all observed messages in the system. In other words, *T*(*A, B*) represents an estimate of the probability that a message is sent from *A* to *B*.

3. Early Reliability Prediction for Component-Based Systems

We already mentioned that several techniques exist for reliability analysis of component based software systems. However, only a few of them can be applied in early stages of system development, i.e., before an executable version of the entire system is available. The model of Sing et. al. [1], which we extend here, has the following characteristics:

1. Applicability early in the software development lifecycle.
2. Seamless integration with UML diagrams.
3. A model supports reliability prediction in the system design phase and reliability assessment based on the observed failure behavior [5].

This reliability model was based on several assumptions:

- Existence of information about failure rates for components and connectors in an architecture.
- Independence of the failures among different components.
- Component failures follow the principle of regularity, i.e., a component is expected to exhibit the same failure rate whenever it is invoked.

3.1 Component based software modeling in UML

UML has become a de facto standard notation for expressing functional attributes of software systems. Component-based software modeling takes advantage of many UML features. Some of those features are the graphical representation of UML diagrams and the possibility of introducing notational extensions. The annotations introduce extra information supporting different tasks. In case of reliability modeling, *Use Case Diagrams, Sequence Diagrams* and *Deployment Diagrams* require annotations [2].

Use Case Diagrams (UCD) provide not only functional description of a system, but also graphic description of how external entities interact with the system. There are two parameters annotated in a UCD. The first parameter is the probability that an actor *i* interacts with the system, denoted by q_i ($\sum_i q_i = 1$). The second parameter is the probability of an actor *i* using the specific system use case *x*, denoted by P_{ix}

($\sum_i \sum_x P_{ix} = 1$). These annotations are then combined to create the probability of a system behavior occurring during system execution. The probability of occurrence of a system behavior *x* is given by

$$P(x) = \sum_{i=1}^{m} q_i \cdot P_{ix} \ , \qquad (1)$$

where *m* is the number of actors that use the given system behavior.

A S*equence Diagram* (SD) describes how groups of software components interact to accomplish a particular task. There exists at least one SD for each Use Case. However, more than one SD may exist for a single UCD. In case of multiple sequence diagrams within the same use case we divide the cumulative probability *P(x)* by the number of sequence diagrams in the use case. Annotations of an SD depict the number of periods each component is in a busy state. When an interaction enters the component or the component is invoked, it enters a busy period. Busy periods are counted. Variable bp_{ij} represents the number of busy periods that the component C_i exhibits in the sequence diagram *j*. The failure probability of component C_i in scenario *j* is represented by Θ_{ij} and calculated from the following equation:

$$\Theta_{ij} = Prob(failure\ of\ C_{ij}) = 1 - (1 - \Theta_i)^{bpij} \qquad (2)$$

A *Deployment Diagram* (DD) describes the application environment in terms of different systems in which computations take place and connectors (networks) between them. The annotations of DD include connection failure probabilities, denoted by Ψ_i. In order to represent failure rate of the communication channel between components *l* and *m*, first the number of interactions between these two components in SD *j* has to be counted. This number is denoted by $|Interact(l,m,j)|$. Using the failure probability of the connector Ψ_i, the reliability Ψ_{lmj} can be calculated from [2]:

$$\Psi_{lmj} = (1 - \Psi_i)^{|Interact(l,m,j)|} \qquad (3)$$

3.2 System Reliability Prediction

The collected information form UCD, SD, and DD, is sufficient to create model for the reliability of the whole system. By combining equations 1, 2, and 3, Cortellessa et. al. [2] derived the following expression for the system level failure rate calculation:

$$\Theta_s = 1 - \sum_{j=1}^{K} P_j \left(\prod_{i=1}^{N} (1 - \Theta_i)^{bpij} \cdot \prod_{(l,i)} (1 - \Psi_{l,i,j})^{|Interact(l,i,j)|} \right) \qquad (4)$$

This equation is used in the ECRA tool to predict the reliability of the component-based application [6]. The reliability prediction algorithm generates beta distributions from component failure rates and it requires the user to enter the confidence intervals for failure rates of each component [5].

4. Extending the model

One of the assumptions of reliability models discussed so far is the independence of failures among different components. Inter-component failure independence implies that error states of one component will not propagate to the other components. This fact simplifies the modeling, which is the primary reason why all existing reliability models for component-based systems assume it [8]. However, in practice this assumption usually does not hold.

4.1 Including error propagation probability

The goal of our research is to extend the reliability model from [1, 2] by including failure propagation. We also extended ECRA system reliability tool [6] with failure propagation feature.

From equation (2) the estimated probability of success of a single component C_i in given sequence diagram SD_j, under the assumption of failure independence, is:

$$Prob(success\ of\ C_{ij} \mid failure\_independence) = 1 - \Theta_{ij} = (1 - \Theta_i)^{bpij} \qquad (5)$$

If we remove the independence assumption, then the probability of success of C_i in SD_j must take into account possible failures that may be propagated to it from other components. The component will execute successfully if and only if it does not excercise any of its own faults and no faults are propagated to it from the other components:

$$Prob(success\ of\ C_{ij}) = Prob(success\_of\ C_{ij}/failure\_independence) \times Prob(no\_error\_propagated) \qquad (6)$$

If, for the given system, we have the *Unconditional Error Propagation Matrix E(A, B)*, the probability that a component C_k will propagate a single fault to the component C_i can be estimated by the following formula:

$$Prob(error\_propagated\ k \rightarrow i) = E(k,i)\Theta_{kj}, \quad (7)$$

i.e., it is proportional to the failure rate of the component C_k in scenario *j*, and the error propagation probability from component C_k to C_i. Thus, the probability of the complementary event – no error propagated from component C_k to component C_i is given by:

$$Prob(no\_error\_propagated\ k \rightarrow i) = 1 - E(k,i)\Theta_{kj}, \quad (8)$$

Taking into account all other components in the system, we obtain the probability of success of the component C_i as follows:

$$1 - \Theta_{ij} = \left(1 - \Theta_i\right)^{bpij} \prod_{k=1}^{N} \left(1 - E(k,i)\Theta_{kj}\right), \quad (9)$$

where *N* is the number of components in the system. Please note that *E(i, i) = 0,* so the component doesn't affect its own failure rate through the propagation.

We can rewrite the old equations expressing component reliability using expression (9). The equations now look as follows:

$$\begin{cases} \Theta_{1j} = 1 - \left(1 - \Theta_1\right)^{bp1j} \prod_{k=1}^{N} \left(1 - E(k,1)\Theta_{kj}\right) \\ \vdots \\ \Theta_{Nj} = 1 - \left(1 - \Theta_N\right)^{bpNj} \prod_{k=1}^{N} \left(1 - E(k,N)\Theta_{kj}\right) \end{cases} \quad (10)$$

If we assume that the system has *K* different sequence diagrams, we obtain the failure rates for each of the components in each sequence diagram. Then, using the formula similar to (4) we can calculate the failure rate of the entire system:

$$\Theta_s = 1 - \sum_{j=1}^{K} P_j \left(\prod_{i=1}^{N} \Theta_{ij} \cdot \prod_{(l,i)} \left(1 - \Psi_{l,i,j}\right)^{|Interact(l,i,j)|} \right) \quad (11)$$

We must note that the solutions to the model (10) contain all possible error propagation paths between the components, including the possible circular dependencies. If such circular dependencies exist, the solutions will overestimate the failure rates of the components (i.e. the obtained failure rates will be higher because of the recursive propagation). However, in practice we observed that the overestimation is reasonably small and the results obtained are representative for the studied systems. We consider this to represent the worst-case reliability analysis, meaning that the failure rates will be lower than those derived by our model.

4.2 Special cases with closed form solutions

In the case when the entire error propagation matrix is zero the resulting model (equations 10 and 11) is the same as the reliability estimation model of Singh et al. [1]. When error propagation probabilities are not equal to *0* (i.e., the error propagation probabilities have been calculated by the methodology described in Section 2.1, or some other method) our new reliability model takes this information into account.

In general, the system of equations (10) that models failure rates of system components and includes error propagation is non-linear. A closed form solution cannot be obtained. We use numerical methods to obtain the solutions for the component failure rates and calculate the failure rate of the system.

For some architectural patterns observed in the UML sequence diagrams the system of equations (10) is linear, and closed form solutions can be obtained. The simplest of these patterns is when two components interact with each other in isolation. For these components, say *A* and *B*, the failure rate equations are:

$$\begin{cases} \Theta_{Aj} = 1 - \left(1 - \Theta_A\right)^{bpAj} \left(1 - E(B,A) \cdot \Theta_{Bj}\right) \\ \Theta_{Bj} = 1 - \left(1 - \Theta_B\right)^{bpBj} \left(1 - E(A,B) \cdot \Theta_{Aj}\right) \end{cases}$$

Solving the system equation, we obtain the following:

$$\Theta_{Aj} = \frac{1 - \left(1 - \Theta_A\right)^{bpAj} + \left(1 - \Theta_A\right)^{bpAj} \cdot \left(1 - \left(1 - \Theta_B\right)^{bpBj}\right) \cdot E(B,A)}{1 - \left(1 - \Theta_A\right)^{bpAj} \cdot \left(1 - \Theta_B\right)^{bpBj} \cdot E(A,B) \cdot E(B,A)}$$

$$\Theta_{Bj} = \frac{1 - \left(1 - \Theta_B\right)^{bpBj} + \left(1 - \Theta_B\right)^{bpBj} \cdot \left(1 - \left(1 - \Theta_A\right)^{bpAj}\right) \cdot E(A,B)}{1 - \left(1 - \Theta_A\right)^{bpAj} \cdot \left(1 - \Theta_B\right)^{bpBj} \cdot E(A,B) \cdot E(B,A)}$$

This special case can be generalized to a system with *N* components, as presented in the scenario diagram in Figure 1. In this case there exists a method invocation cycle: C1 calls a method from C2, C2 from C3, etc. At the end, C_N returns the results of the processing to C1.

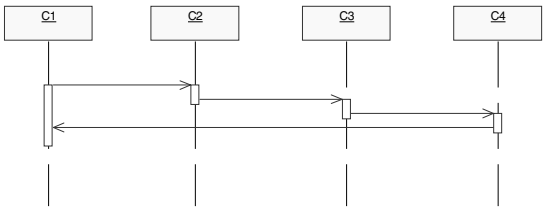

Figure 1: Scenario Diagram with calling cycle

While the expressions representing component failure rates of the diagram from Figure 1 are more complicated than those above, they can still be obtained analytically.

5. Case Study

In our experiments, we utilize ECRA (Early Component-based Reliability Assesment) tool [6]. ECRA implements reliability analysis methodology presented so far. We analyzed an actual Personnel Access Control System (PACS) system to demonstrate the results of our methodology. We present two sets of early reliability prediction results for the PACS model; In the first set of simulations we used methodology from [1, 2], which calculates system failure rate based on the failure independence assumption. We also used our new methodology, which takes in account error propagation among components. Calculated results demonstrate that these two approaches result in different values of predicted reliability. In the second set of experiments we changed the failure rate of the component which is known to have a critical impact on the system reliability. Performing two simulations, with and without error propagation, we demonstrate that the resulting reliability estimates differ quite dramatically.

5.1 The ECRA tool

ECRA (see user interface in Figure 2) is a homegrown tool built at the High Assurance Systems Research Center of West Virginia University [6]. It automates reliability assessment of component-based systems using UML diagrams and the methodology described in [1, 2]. We modified ECRA tool to accommodate model extensions. The tool accepts three types of UML diagrams:

- The use case diagram - ECRA requires detailed description of the actors, use cases, and modules within each use case.
- The sequence diagram – ECRA's input are sequence diagrams. From them, it calculates the number of busy periods for each component in each diagram.
- The deployment diagram – ECRA requires the information of the name of each processor and processes, along with the description of connections between them.

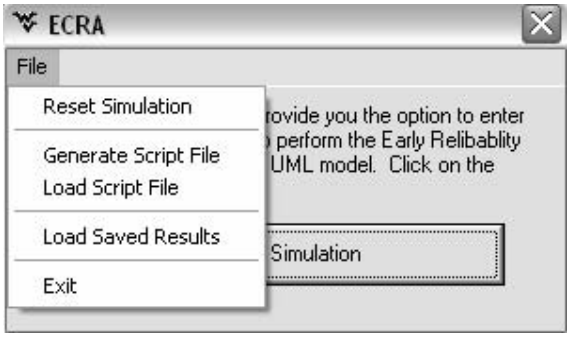

Figure 2: The User Interface of ECRA tool

The mentioned diagrams constitute an input into the ECRA tool. The diagrams need to be annotated with component and connector failure rates and the error propagation matrix *E*.

If the error propagation matrix is not known (or not available), ECRA tool will assume that failures cannot be propagated between components.

The outputs of the ECRA tool are the plots representing:

- Prior beta distributions of component and connector failure rates.
- A histogram plot of the predicted system reliability.
- In addition to system failure probability ECRA also provides the 95% confidence interval around the system reliability estimate.

5.2 Personnel Access Control System (PACS)

To demonstrate the importance of failure propagation in component-based software reliability modeling, we analyzed the reliability of Personnel Access Control System (PACS) [13]. PACS system controls physical access to a restricted area. Authorized users receive a personal ID card and a personal identification number (PIN). In order to get access, the user swipes the ID card which contains user name and a unique number (SSN) through a card reader. After searching its database of user names and SSNs to validate privileges, PACS system instructs the user to enter a four-digit PIN number. If the entered PIN matches a stored PIN, the system allows the user to enter the area through a gate. Two types of actors use

Table 1: Information record of PACS components

Component	Name	Failure Probability	Confidence Interval	Busy Periods		
				bpi1	bpi2	bpi3
C1	Communication Driver	0.002	(0.001,0.003)	24	4	5
C2	PACS	0.005	(0.003,0.007)	4	1	1
C3	Validator	0.002	(0.001,0.003)	1	0	0
C4	User LCD	0.006	(0.003,0.009)	3	1	1
C5	Officer LCD	0.006	(0.003,0.009)	0	1	1

PACS system: the card holder, and the security officer. PACS guides the card holder with messages written on a single-line display screen. Security officer monitors and controls the PACS using a console with another single-line display screen, an alarm, a reset button, and a gate override button. In its current form, requirements specification for PACS originated from a US government agency. The requirements are relatively simple, but realistic. Systems built according to this specification have been deployed. An annotated use case diagram of PACS system is shown in Figure 3.

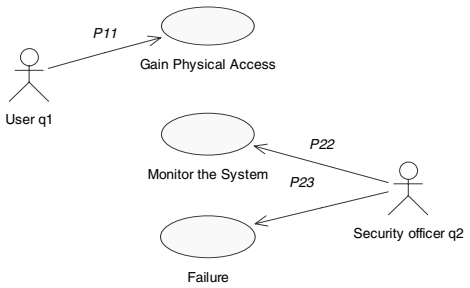

Figure 3: Annotated UCD for the PACS system

In annotated UCD, the usage probability for user type *q1*, the card holder, is *0.96*, and for the security officer, *q2, is 0.04*. The user to use-case probability for scenario *P11* is *1*, for *P22* is *0.75*, and for *P23* is *0.25*. These numbers are based on the estimated operational profile from the requirements document [13]. It is expected that *96%* of the time PACS system operates by granting access to legitimate users of the system. 4% of the time some additional monitoring from the security officer is required, out of which ¼ or 1% is due to system failure, and the rest, 3%, is due to incorrectly typed PIN numbers, or defective ID cards.
For each use case scenario there is exactly one sequence diagram. Figure 4 illustrates the so-called failure scenario, which is one of the three different sequence diagrams in PACS. This scenario has been annotated with the number of busy periods for each component.

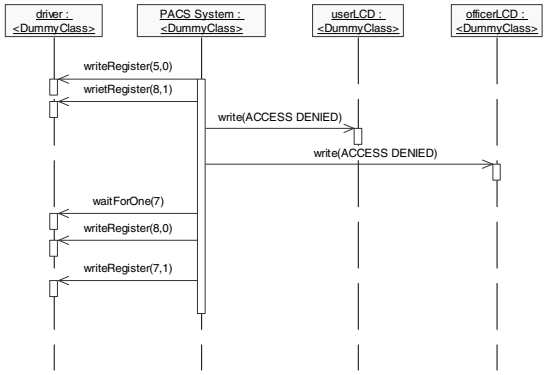

Figure 4: Annotated SD for the Failure case

The PACS architecture consists of five components (processes), which run on the same processor. Table 1 presents the information record for each component in the initial set of simulations. The record includes component's name, component's failure probability, a 95% confidence interval of the failure probability, and the number of busy periods the component exhibits in each sequence diagram. Table 2 shows the number of messages exchanged between components along all the three scenarios. The scenario numbering is consistent in Tables 1 and 2.

Table 2: Number of interactions per pair of components through scenarios

Pair(l, i)	Scenario 1	Scenario 2	Scenario 3
(C2, C1)	5	24	4
(C2, C3)	0	1	0
(C2, C4)	1	3	1
(C2, C5)	1	0	1

5.3 System reliability under failure independence assumption

The methodology by Singh et. al. [1] assumes failure independence. Equation (4) and the values from Table 1 are used in the calculation of system failure probability Θ_s as follows:

$$\Theta_s = 1 - [0.96\ (\left(1-\Theta_1\right)^{24}\left(1-\Theta_2\right)^{4}\left(1-\Theta_3\right)^{1}\left(1-\Theta_4\right)^{3}) + 0.03\ (\left(1-\Theta_1\right)^{4}\left(1-\Theta_2\right)^{1}\left(1-\Theta_4\right)^{1}\left(1-\Theta_5\right)^{1}) + 0.01\ (\left(1-\Theta_1\right)^{5}\left(1-\Theta_2\right)^{1}\left(1-\Theta_4\right)^{1}\left(1-\Theta_5\right)^{1})].$$

Figure 5 depicts the histogram of expected system level failure rates, Θ_S, under the assumption of failure independence. Θ_S is presented in the form of a histogram because it is obtained by Monte Carlo simulations of the expression above [1].

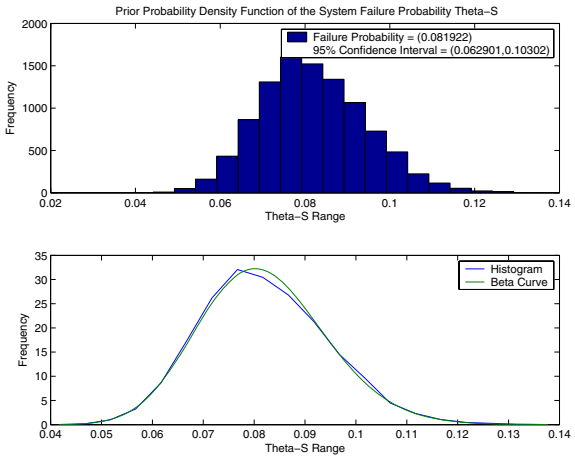

Figure 5: Predicted PDF of Θ_s assuming failure independence

The first observation is that this reliability prediction does not meet the system requirement [13], which limits system failure probability per demand to *0.01*. More reliable components than those assumed in Table 1 are needed if the system is to achieve the desired reliability.

More important for the subject of this paper, in a related project [16] we implemented PACS and empirically evaluated its reliability through extensive testing. To our dismay the observed system reliability was much lower than predicted by this model, even though component reliabilities were those indicated in Table 1. We concluded that the reliability model which excludes failure propagation probability is too optimistic.

5.4 System reliability with error propagation

Next, beside the component reliabilities used in the first experiment, we added error propagation matrix shown in Table 3. The matrix was calculated by the methodology described in Section 2.1. From UML state diagrams, sequence diagrams and the expected operational profile, we calculated the probability distribution of the set of states for each component, and probability distribution of the set of messages between each pair of components. First, we calculated the conditional error propagation probability matrix, and then the transmission probability matrix. The values in Table 3 represent the unconditional error propagation probability.

Table 3: The error propagation values among the PACS's components

	C1	C2	C3	C4	C5
C1	0	0.425169	0	0	0
C2	0.804878	0	0.02439	0.083388	0.073165
C3	0	0.687124	0	0	0
C4	0	0	0	0	0
C5	0	0	0	0	0

Equation (11) was used to set up the system failure probability expression that includes error propagation:

$$\Theta_s = 1 - [0.96\ (\Theta_{11}^{24} \cdot \Theta_{21}^{4} \cdot \Theta_{31}^{1} \cdot \Theta_{41}^{3}) + 0.03 \cdot (\Theta_{12}^{4} \cdot \Theta_{22}^{1} \cdot \Theta_{42}^{1} \cdot \Theta_{52}^{1}) + 0.01\ (\Theta_{13}^{5} \cdot \Theta_{23}^{1} \cdot \Theta_{43}^{1} \cdot \Theta_{53}^{1})]$$

Figure 6 depicts the histogram of expected system level failure rates, Θ_S, in case where the failure independence assumption is eliminated.

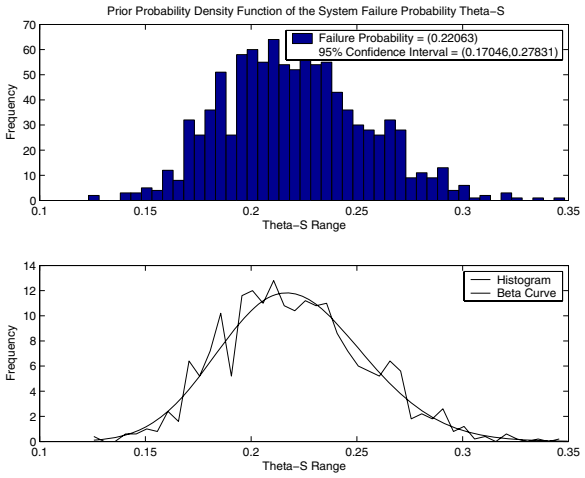

Figure 6: Predicted PDF of Θ_s using error propagation

In contrast with the first result, this system failure rate is much higher. The increase in failure rate is caused by the error propagation between the components. Predicted system reliability is much closer to the one observed in tests of PACS implementation. In fact, the measured reliability is within the 95% confidence interval calculated by the model. So, while additional effort is required to calculate the error propagation probability matrix, it appears that the precision of system level reliability prediction has increased. Unfortunately, an automated tool is still not available for calculating unconditional error propagation probability matrix from UML artifacts. In our experiments, the values in the matrix were computed by hand, requiring significant effort.

5.5 The sensitivity analysis assuming failure independence

Based on the error propagation matrix and prior experience with PACS, we observed that system reliability is the most sensitive to the changes in reliability of *Driver* component, *C1*. For the second set of experiments we used the same simulation parameters as before except for the failure rate of the *C1*. In the following two experiments, we assumed that the reliability of *C1* doubled to 0.001 failures per demand, with the 95% confidence interval of (0.0005, 0.0015). Figure 7 presents the results obtained by ECRA simulations when the error propagation information is not included in the model.

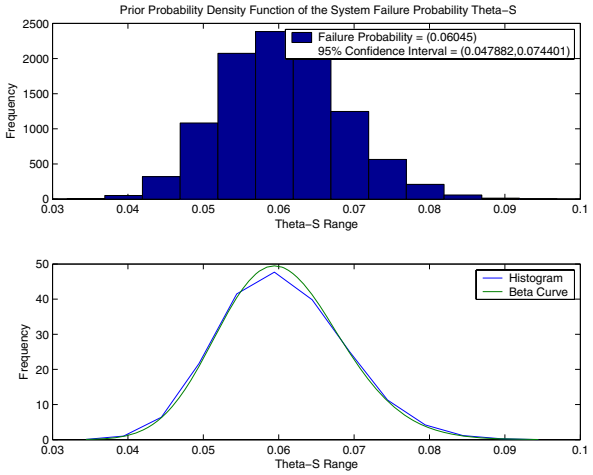

Figure 7. Predicted PDF of Θ_s with improved Driver component and no failure propagation

Comparing Figures 5 and 7 we observe that the improvement in the *Driver* component positively impacts system reliability. The expected system failure rate has decreased from 0.0819 to 0.0604, an improvement of 26.25%.

5.5 The impact of component reliability and error propagation to system reliability

Figure 8 presents the result of ECRA Monte Carlo simulations when error propagation information is included in the model.

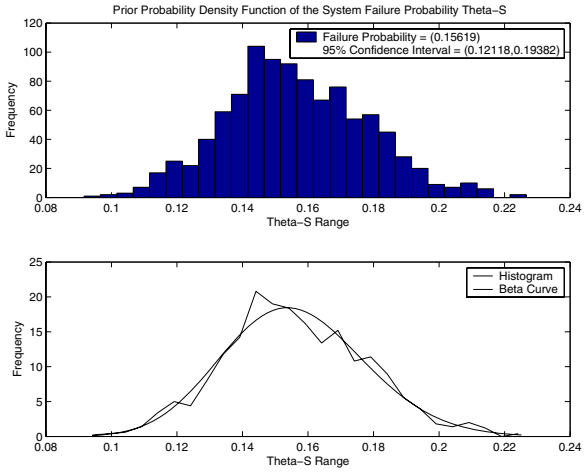

Figure 8: Predicted PDF of Θ_s using error propagation and improved driver component

Comparing Figures 6 and 8 we observe a substantial improvement in the system failure rate from 0.221 to 0.156, a 29.41% improvement.

This result matches closely the measured reliability of our implementation [16]. Predictive performance can be explained by the error propagation, since there is a substantial probability (*0.425* in Table 3) that an error in *Driver* component (*C1*) can affect the PACS component *C2*. Component *C2*, in turn, propagates this error to all the other components leading to a certain system failure (the row *C2* is has mostly non-zero elements). But even more importantly, the *Driver* component is heavily used, having 24 busy periods (see Table 1) in the usage scenario that is utilized about 96% of the time. Therefore, any faults in *C1* are more likely to lead to system error. This observation contrasts, for example, the error propagation behavior of *Validator* component *C3*. *C3* has a high error propagation probability towards the PACS component *C2* (0.687 in Table 3), but its usage frequency is much lower than that of *Driver* component (*C1*). Therefore, its overall contribution to system reliability is modest.

6. Conclusions and Future work

In this paper, we extended the existing Bayesian approach for reliability estimation of component- based software systems [1, 2]. Like most other component-based reliability models, the old model assumed that system components will fail independently. To some extent, this assumption is realistic if components are accompanied by wrappers, which contain failure propagation close to its source. However, not all the component-based applications contain wrappers. Plus, the question is whether wrappers are able to prevent the propagation of component errors, i.e., corrupted states resulting from incorrect computations but not externally observable as failures. Error propagation, if it can be captured by the probability of occurrence, represents a meaningful extension to current reliability models.

We presented one possible approach to error propagation probability calculation and integrated it into the Bayesian methodology for reliability prediction of component-based systems. Our analytical and experimental observations indicate that error propagation can make a significant difference in system reliability prediction, especially if components "leaking" erroneous states are complex and frequently used.

While our early experiences are positive, they call for a more extensive set of experiments. New experiments should be conducted to investigate whether reliability predictions of models which take into account error propagation probability are statistically significantly more precise than those that do not.

References

[1] H. Singh, V. Cortellessa, B. Cukic, E Gunel, and V. Bharadway, "A Bayesian approach to reliability prediction and assessment of component based systems", in *Proc. 12th International Symposium on Software Reliability Engineering (ISREE'01),* Boca Raton, FL, October 2001.

[2] V. Cortellessa, H. Singh, and B. Cukic. "Early reliability assessment of UML based software models", *3rd International Workshop on Software Performance (WOSP '02)*, Rome, Italy, July 2002.

[3] D.M. Nassar, W.A. Rabie, M. Shereshevsky, N. Gradetski, H.H. Ammar, Bo Yu, S Bogazzi, and A. Milli, "Estimating error propagation probabilities in software architecture", *Technical report,* College of Computer Science*,* New Jersey Institute of Technology 2002.

[4] W. Abdelmoez, D.M. Nassar, M. Shereshevski, N. Gradetski, R. Gunnallan, H.H. Ammar, Bo Yu, and A. Milli, "Error Propagation in Software Architectures", in *Proc. 10th IEEE Int'l Software Metrics Symposium (METRICS 2004)*, Chicago, IL, Sept. 2004.

[5] B. Cukic, "The virtues of assessing software reliability early", *IEEE Software*, Vol 22, No. 3, May/June 2005.

[6] W. B. Smith, "Early component-based reliability assessment using UML Based software models". *MS Thesis,* Lane Department of Computer Science and Electrical Engineering, West Virginia University, 2002.

[7] J. Horgan, and A. Mathur, "Software testing and reliability", in *Handbook of software reliability engineering*, Michael R. Lye (editor), McGraw-Hill, New York, NY, 1996.

[8] K. Goseva-Popstojanova, and K.S. Trivedi, "Architecture based approaches to software reliability prediction", *International Journal Computers & Mathematics with Applications,* (to appear).

[9] S. Ghokale, M. Lyu, and K. Trivedi, "Reliability simulation of component based software systems", *Proc. 9th International Symposium on Software Reliability Engineering, (ISSRE'98)*, Paderborn, Germany, 1998.

[10] S. Ghokale, E. Wong, K. Trivedi, and J. R. Horgan, "An analytical approach to architecture based software reliability prediction", *Proc. Symposium on Application Specific Systems and Software Engineering Technology (ASSET'98)*, Dallas, TX, 1998.

[11] H. Ammar, S.M. Yacoub, A. Ibrahim, "A fault model for fault injection analysis of dynamic UML specifications", *International Symposium on Software Reliability Engineering (ISSRE '01)*, Boca Raton, FL, October 2001.

[12] V.R. Basili, and H.D. Rombach, The Tame project: Towards improvement oriented software environments, *IEEE Transactions on Software Engineering*, 1988.

[13] Requirements Specification for Personal Access Control System, National Security Agency, 2003.

[14] M. Hiller, A. Jhumka, N. Suri, "An Approach for Analysing the Propagation of Data Errors in Software", *Proc. Dependable Systems and Networks (DSN '01)*, Goteborg, Sweden, July 2001.

[15] M. Hiller, A. Jhumka, N. Suri, "PROPANE: An Environment for Examining the Propagation of Errors in Software", Proc. Int'l. Symp. Software Testing and Analysis (ISSTA '02), Rome, Italy, July 2002.

[16] M. Li, Y. Wei, D. Desovski, H. Nejad, S. Ghose, B. Cukic, C. Smidts, "Validation of a Methodology for Assessing Software Reliability", in *Proc. 15th International Symposium on Software Reliability Engineering (ISREE'04),* St. Malo, France, October 2001.