An architecture-based software reliability model

Conference Paper · February 1999

DOI: 10.1109/PRDC.1999.816223 · Source: IEEE Xplore

Wen-Li Wang
Pennsylvania State University
28 PUBLICATIONS **507** CITATIONS
SEE PROFILE

Mei-Hwa Chen
University at Albany, State University of New York
54 PUBLICATIONS **1,742** CITATIONS
SEE PROFILE

An Architecture-Based Software Reliability Model

Wen-Li Wang Ye Wu Mei-Hwa Chen
Computer Science Department
SUNY Albany
(wlwang,wuye,mhc)@cs.albany.edu

Abstract

In this paper we present an analytical model for estimating architecture-based software reliability, according to the reliability of each component, the operational profile, and the architecture of software. Our approach is based on Markov chain properties and architecture view to state view transformations to perform reliability analysis on heterogeneous software architectures. We demonstrate how this analytical model can be utilized to estimate the reliability of a heterogeneous architecture consisting of batch-sequential/pipeline, call-and-return, parallel/pipe-filters, and fault tolerance styles. In addition, we conduct an experiment on a system embedded with three architectural styles to validate this heterogeneous software reliability model.

Keywords: Software Architecture, Architectural Style, Markov Model, Reliability Estimation

1 Introduction

Early quality prediction at the architecture design stage is highly desired by software managers and developers, as it provides a means for making design decisions and thereby facilitates effective development processes [7]. If a design flaw, especially in large-scale software system, is detected in the implementation or testing phase, it is more difficult and more expensive to make changes than at the architecture design stage. Software architecture analysis aims at investigating how an architecture meets its quality requirements [1] based on the structure and the correlation among the components of the software. It not only facilitates component-based software development, but also provides a means for early quality prediction. Therefore, the quality of component-based software can be predicted by using software architecture analysis methodologies.

Software architecture, which describes the structure of software at an abstract level [11, 21], consists of a set of *components*, *connectors* and *configurations*. Furthermore, a repeatable pattern that characterizes the configurations of components and connectors of software architectures is considered as an architectural style [8, 9]. Many architectural styles have been identified [23, 28] with new styles continuously emerging [18, 19, 27]. Thus, a practitioner is faced with the challenge of selecting suitable styles, or modeling configurations of selected styles, for designing the architecture to a given software specification. In such a situation, a method or model to predict or evaluate the reliability of a heterogeneous style software system can certainly provide a means through which designers can configure the architecture that best fits their quality demands.

Our previous studies [5, 29] suggest that at the architecture design stage it is possible to select an architectural style that can provide better performance and/or availability than others, if the characteristics of the application environment can be realized. In this paper, we present a model that analyzes the reliability of the system that combines heterogeneous architectural styles including batch-sequential/pipeline, parallel/pipe-filters, call-and-return, and fault tolerance styles. System reliability is analyzed on the basis of the reliability of components and connectors in these architectural styles. In addition, the operational profile is taken into account by utilizing the transition probabilities from one component to others. Assuming that the reliabilities of components and connectors are independent of the transition probabilities, we present how Markov model can be utilized to predict the reliability of a heterogeneous software architecture, following the transformation from architecture view to state view. Moreover, a system embedded with three architectural styles is utilized to validate this heterogeneous software reliability model.

In Section 2, we describe and revise a current component-based reliability model. And Section 3 presents a reliability model based on individual architectural style. The heterogeneous software reliability model is given in Section 4. And the experiment we conducted to validate the heterogeneous reliability model is shown in Section 5. Our conclusions and future research directions are in Section 6.

2 Component-Based Reliability Modeling

In this section, a component-based reliability model developed by Cheung [6] is described that takes the reliability of each component and the operational profile into account. In this model, a state diagram which depicts the system behavior is used. A state represents the execution of a single component and the transition probability from one state to another is obtained from the operational profile of a system.

The reliability of a software system depends on the execution sequence of states and the reliability of each individual state. Based on Markov chain properties, the transition between states is assumed as a Markov process. It means that components to be executed in next state will depend only on the components of current state and the components of the next state will not have any dependency to the past history of current state.

The state diagram is a directed graph in which each node S_i represents a state and the transition from state S_i to S_j is represented by a directed edge (S_i, S_j). Let R_i denote the reliability of S_i and P_{ij} be the reaching probability from S_i to S_j. Based on the state diagram, we then define a transition matrix, *M,* and the value of the entry *M(i,j),* which can be

$$M = \begin{array}{c} \\ S_1 \\ S_2 \\ \vdots \\ S_i \\ \vdots \\ S_{n-1} \\ S_n \end{array} \begin{array}{c} \begin{array}{ccccccc} S_1 & S_2 & .. & S_i & .. & S_{n-1} & S_n \end{array} \\ \left[\begin{array}{ccccccc} 0 & R_1P_{12} & .. & R_1P_{1i} & .. & R_1P_{1(n-1)} & R_1P_{1n} \\ R_2P_{21} & 0 & .. & R_2P_{2i} & .. & R_2P_{2(n-1)} & R_2P_{2n} \\ \vdots & \vdots & .. & \vdots & .. & \vdots & \vdots \\ R_iP_{i1} & R_iP_{i1} & .. & 0 & .. & R_iP_{i(n-1)} & R_iP_{in} \\ \vdots & \vdots & .. & \vdots & .. & \vdots & \vdots \\ R_{n-1}P_{(n-1)1} & R_{n-1}P_{(n-1)2} & .. & R_{n-1}P_{(n-1)i} & .. & 0 & R_{n-1}P_{(n-1)n} \\ R_nP_{n1} & R_nP_{n2} & .. & R_nP_{ni} & .. & R_nP_{n(n-1)} & 0 \end{array}\right] \end{array} \quad ....(1)$$

computed as $R_i \times P_{ij}$, indicates the successful arrival at state S_j from S_i, i.e., the correct execution of the S_i and the occurrence of the transition from S_i to S_j.

Below we describe how to calculate the reliability from the transition matrix. Let $S = \{ S_1, S_2, ..., S_n \}$ be the set of states in the state diagram where S_1 is the initial state and S_n is the final state. $M^k(i, j)$ represents the probability of reaching state S_j from S_i through *k* transitions. Therefore, the reliability *R* beginning from S_i to S_j with total *k* transitions is represented as

$$R = M^k(i, j) \times R_j$$

From initial state S_1 to final state S_n, the number of transitions *k* may vary from 0 to infinity, where 0 means that the initial state is also the final state and infinity means that a cyclic loop may occur indefinitely among the states. Therefore, it is necessary to consider every possible outcome of state transitions.

Let *T* be a matrix such that

$$T = I + M + M^2 + M^3 + ... = \sum_{k=0}^{\infty} M^k$$

$$\approx \frac{I}{I - M} = (I - M)^{-1}$$

where *I* is the identity matrix of size $n \times n$. The over-all system reliability can be computed as follows:

$$R = T(1, n) \times R_n$$

with

$$T(1, n) = (-1)^{n+1} \frac{|E|}{|I - M|}$$

where $|I - M|$ is the determinant of matrix $(I - M)$ and $|E|$ is the determinant of the remaining matrix excluding the *n*th row and first column of the matrix $(I - M)$.

Cheung's model is based on one single initial state and one single final state, whereas a large system may have a set of initial states $I = \{ S_{i1}, S_{i2}, ..., S_{im} \}$ and a set of final states $F = \{ S_{f1}, S_{f2}, ..., S_{fn} \}$. In such a situation, we revised the problem of multiple initial and final states to one initial state and one final state problem by introducing a *super-initial* and a *super-final* state to the state diagram. Figure 1 shows how a state diagram with multiple initial states and final states in the dotted rectangular area can be converted to a state diagram with only a single initial state and a single final state. We add a *super-initial* state S^I and a directed edge (S^I, S_{ij}) with its transition probability P_{Ij}, observed from the operational profile, for each *j* = 1, 2, …., *m*. Similarly, we create a *super-final* state S^F and a directed edge (S_{fj}, S^F) with transition probability 1 for each *j* = 1, 2, …. , *n*. The reliabilities for both states S^I and S^F are assigned 1.

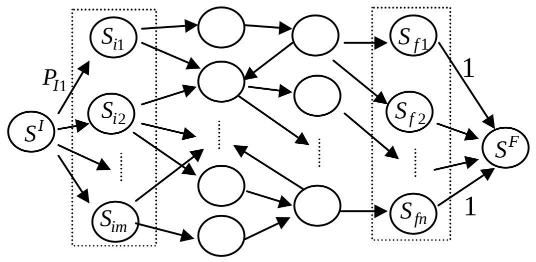

Fig. 1: One super-initial and one super-final state diagram

3 Style-based Reliability Modeling

We have described and revised Cheung's component-based reliability model; however, for a system with a complex architectural environment, this reliability model needs to be further refined to take the characteristics of different architectural styles into account. The key idea of this study is to follow the revised reliability model and to define architecture-based reliability model to compute the reliability of heterogeneous software systems which may be comprised of various architectural styles. In this section we describe how to model the reliability of an architecture style based on its characteristics. A model that computes the reliability of a software, composed of heterogeneous architectural styles, is presented in the next section.

The architectural styles we studied include batch-sequential/pipeline, parallel/pipe-filter, call-and-return, and fault tolerance styles. The reason for depicting these four basic types of architectural styles is because most of the additional architectural styles can be viewed as extensions of these four types. For example, client-server style is similar to call-and-return style but should strongly take connector reliabilities into account. Hierarchical and layered styles are similar to batch-sequential/pipeline style in their transition behaviors.

In order to utilize the Markov model, a transformation for each architectural style from an architecture view to a state view is introduced. Furthermore, based on the transformed state view, the transition matrix M (as shown in (1)) can be refined to obtain the style-based software reliability. The details of each style are described as the following, where in Figures 2 to 5, R_i represents the reliability of component C_i, and P_{ij} represents the probability of transition from component C_i to its successor component C_j. To take the connector reliability into account, P_{ij} could be adjusted as the original transition probability multiplied by the reliability of the corresponding connector.

3.1 Batch-sequential/pipeline style

Both batch-sequential and pipeline styles are running in a sequential order. They share the same architecture view and state view. Although in the batch-sequential style, outputs of a component are produced only after all its inputs are fully processed, whereas in pipeline style, output may be produced before inputs are fully consumed [22, 24].

These styles can be modeled as shown in Figure 2(a), where C_1, C_2, … , C_k are the components of the architecture and a component such as C_2 can only go to either one of its branching subsequent components. The transformation from architecture view to state view is one-to-one mapping, shown in Figure 2(b), where S_1, S_2, …. , S_k are the states of the Markov chain.

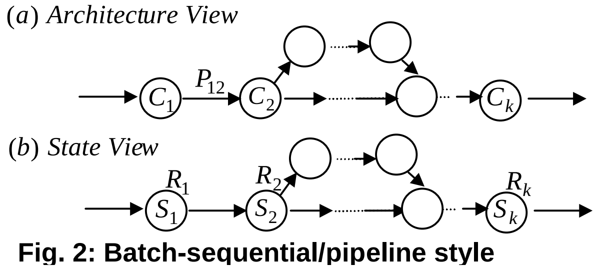

Fig. 2: Batch-sequential/pipeline style

Assuming that the architecture is composed of k components, in these styles there will be k states in the Markov chain. The transition matrix M can be obtained as follows:

$$\begin{cases} M(i,j) = R_i P_{ij},\ S_i \text{ can reach } S_j \\ M(i,j) = 0,\ S_i \text{ can not reach } S_j \end{cases},\ \text{for } 1 \le i, j \le k \ \ldots\ldots (2)$$

where $M(i,j)$ is the probability of successfully reaching state S_j from S_i.

3.2 Parallel/Pipe-filter style

Under a sequential execution environment, only a single component is executed at a time. However, in a concurrent execution environment, components are commonly running simultaneously. The behavior of a software system with respect to the execution process can be modeled by using a Markov chain [16, 20] in which a state is defined by the execution of multiple components.

In a parallel or a pipe-filter architectural style, multiple components can be executed concurrently, as shown in the dotted area of Figure 3(a). The difference between these two styles is that parallel computation is generally in multi-processors environment, whereas pipe-filter style occurs commonly in a single processor, multi-processes environment. Figure 3(b) is the state diagram of the parallel/pipe-filter architectural style, where the executions of the components C_2 to C_{k-1} are congregated into the state S_{p1} which is an element of the parallel state set S_p.

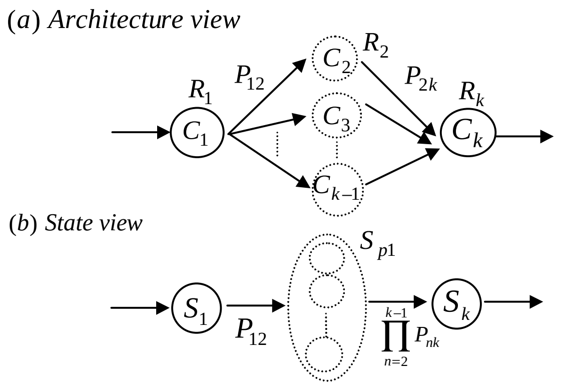

Fig. 3: Parallel or pipe-filter style

In Figure 3(a), there are k components in which l=k-2 components are running concurrently into the same state; therefore, the total number of states is k-l+1. Because of the characteristics of parallel style, the transition probabilities from component C_1 to components C_2, C_3, …., and C_{k-1}, are all equal to P_{12}, which is now the transition probability from state S_1 to S_{p1}. For convenience, we introduce $\{S_i\}$, which returns the row number or the column number of state variable S_i in a matrix. Entry *M({ S_{p1} },{ S_k })* requires that all the components from C_2 to C_{k-1} in state S_{p1} perform successfully and finally reach state S_k. Because component reliabilities and transition probabilities are all independent of each other, therefore the value of *M({ S_{p1} },{ S_k })* is equal to $\prod_{n=2}^{k-1} R_n P_{nk}$, which is the product of all the component reliabilities in this state and the transition probabilities from components C_2, C_3, …, and C_{k-1} to component C_k , respectively.

For *k* components, the transition matrix can be obtained:

$$\begin{cases} M(i,j) = R_i P_{ij},\ S_i \notin S_p \\ M(i,j) = \prod\limits_{C_n \text{ in } S_i} R_n P_{nj},\ S_i \in S_p & ,\begin{array}{l}\text{for } 1 \le i, j \le |S| \\ \text{and } 1 \le n \le k\end{array} \\ M(i,j) = 0,\ S_i \text{ can not reach } S_j \end{cases} \quad \text{......(3)}$$

3.3 Fault Tolerance

A fault-tolerance architectural style consists of a primary component and a set of backup components, which may be implemented in different algorithms or data structures, from the primary component. These components, including the primary and the backups, are placed in parallel so that when one component fails, the others can still provide services. Given the architecture displayed in Figure 4(a), those dotted components from C_3 to C_{k-3} are modeled as backup components to primary component C_2. Inside the dotted rectangle shown in Figure 4(b), these components are congregated into the state S_{b1} which is an element of the fault tolerance state set S_b .

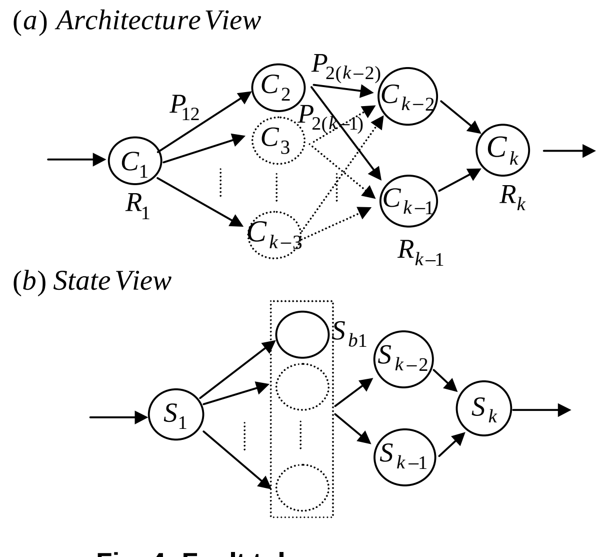

Fig. 4: Fault tolerance

We assume that all the backup components have the same transition probabilities as the primary component to each subsequent component. From Figure 4(a), assume that there are *k* components in which *l*=*k*-4 components are running as fault tolerance in the same state; therefore, the total number of states is *k*-*l*+1. The concurrent characteristics of fault tolerance style is similar to parallel style, where the transition probabilities from component C_1 to components C_2, C_3, …., and C_{k-3}, are all equal to P_{12}, which is now the transition probability from state S_1 to S_{b1}.

However, state S_3 improves the reliability only when state S_2 fails. Similarly, state S_4 enhances the reliability when both states S_2 and S_3 fail. Thus, the reliability of reaching states S_{k-1} and S_{k-2} from S_1, we have to consider when state S_2 is always active, when state S_2 fails but S_3 is active, and so on. By induction, entry *M(1,{ S_{b1} })* is equal to $R_2 + \sum_{n=3}^{k-3}\left(\left(\prod_{m=2}^{n-1}(1-R_m)\right)R_n\right)$. Since we assume that all the backup components have the same transition probabilities as the primary component to each subsequent component, the values of *M({ S_{b1} },{ S_{k-1} })* and *M({ S_{b1} },{ S_{k-2} })* are equal to $R_{b1}P_{2(k-1)}$ and $R_{b1}P_{2(k-2)}$, respectively.

Thus in a fault tolerance architecture with *k* components, the transition matrix can be constructed as follows:

$$\begin{cases} M(i,j) = R_i P_{ij},\ S_i \notin S_b \\ M(i,j) = R_{a1} + \sum\limits_{q=a2}^{ar}\left(\left(\prod\limits_{m=a1}^{q-1}(1-R_m)\right)R_n\right) \\ \qquad ,S_i \in S_b \text{ and } S_i \text{ includes } C_{a1} \text{ to } C_{ar} \\ M(i,j) = 0,\ S_i \text{ can not reach } S_j \end{cases}$$

, for $1 \le i, j \le |S|$ and $1 \le a_r \le k$ ………………………..(4)

3.4 Call-and-Return

In the call-and-return style, the execution of one component may request some services provided by the other components before transferring its complete control authority to others. Thus, after such a request is fulfilled by the called components, the control still returns to the calling component and executes the next statement from where the component left.

Therefore, the called components may execute multiple times with only one time execution of the calling component. In Figure 5(b), the state view is obtained by one-to-one mapping to the architecture view shown in Figure 5(a) where state S_1 is the calling state while S_2 is the called state that provides services. During the execution, state S_2 may be visited multiple times before state S_1 gives control over to the final state S_3.

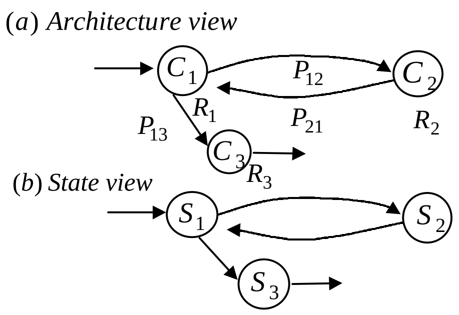

Fig. 5: Call-and-return style

The entry *M(1,3)* is equal to R_1P_{13}, which is the reliability of state S_1 multiplied by the transition probability from S_1 to S_3. Likewise, the entry *M(2,1)* can be computed as the reliability of state S_2 multiplied by the transition probability from S_2 to S_1. The most important entry is *M(1,2)* which only considers the transition probability from S_1 to S_2 without considering the reliability of state S_1. This is because state S_1 will be visited only once before entering state S_3 regardless of how many times state S_2 is visited. Therefore, the reliability of state S_1 only needs to be considered when it transfers the control to state S_3.

Assuming there are *k* components, the total number of states is therefore *k*. The transition matrix *M* can be constructed as follows:

$$\begin{cases} M(i,j)=R_iP_{ij},\ S_i \text{ can reach } S_j \\ M(i,j)=P_{ij},\ S_i \text{ can reach } S_j \\ \quad \text{, and } S_j \text{ is a called component.} \\ M(i,j)=0,\ S_i \text{ can not reach } S_j \end{cases}, \text{for } 1\le i,j\le k \text{ …… (5)}$$

4 Architecture-based Reliability Modeling

We have shown how to compute the reliability of a system based on a single architectural style in the previous section. For an architecture composed of heterogeneous styles, the reliability of a system can be modeled as following.

Step 1: Identify the architectural styles in a system, based on the design specification of a system.

Step 2: Transform the architecture view(s) into state view(s) of each style, and compute the reliability and transition probability of each state in the state views.

Step 3: Integrate these state views into a global state view of the system, based on the architecture view of a complete system.

Step 4: Construct the transition Matrix based on the global state view of the system and compute the reliability of the system.

Similarly, we can consolidate formulas 2 to 5 to construct the transition matrix *M* with totally *k* states of a heterogeneous software architecture including batch-sequential/pipeline, parallel/pipe-filter, fault tolerance, and call-and-return styles:

$$\begin{cases} M(i,j)=0,\ S_i \text{ can not reach } S_j \\ M(i,j)=R_iP_{ij},\ S_i \notin S_p \text{ and } S_i \notin S_b \\ M(i,j)=\prod_{C_n \text{ in } S_i} R_nP_{nj},\ S_i \in S_p \\ M(i,j)=R_{a1}+\sum_{q=a2}^{ar}\left(\left(\prod_{m=a1}^{q-1}(1-R_m)\right)R_n\right),\ S_i \in S_b \\ \quad \text{and } S_i \text{ includes } C_{a1} \text{ to } C_{ar}\}. \\ M(i,j)=P_{ij},\ S_j \text{ is a called component} \end{cases}$$

, for $1\le i,j,ar,n\le k$

The following example is used to demonstrate how to estimate the reliability of a system consisting of heterogeneous styles. Let Figure 6 be the directed graph representing the architecture view of a software system with fifteen components, where C_1 represents the input component and C_{15} represents the output component. In this architecture, C_3 and C_4 are running in parallel. Component C_{10} is a backup component of C_9. Components C_{11}, C_8, and C_5 are running as call-and-return style, where C_{11} is the calling component of C_8 while C_8 is the calling component of C_5. The other components are running in sequential manner with branching factors and may proceed in cyclic loops.

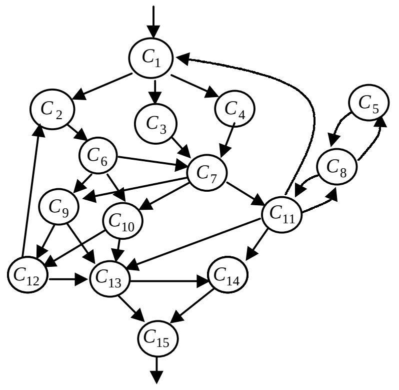

Fig. 6: Architecture view

The reliability of the components is shown in the following:

$R_1=0.998$	$R_2=0.990$	$R_3=0.980$
$R_4=0.995$	$R_5=0.999$	$R_6=0.985$
$R_7=0.996$	$R_8=0.975$	$R_9=0.990$

$R_{10} = 0.998 \quad R_{11} = 0.950 \quad R_{12} = 0.965$

$R_{13} = 0.970 \quad R_{14} = 0.980 \quad R_{15} = 0.992$

To simplify the model, we assume the transition probabilities P_{ij} between the components C_i and C_j have already considered the connector reliabilities.

$P_{1,2} = 0.40 \qquad P_{1,3} = P_{1,4} = 0.60$

$P_{2,6} = P_{3,7} = P_{4,7} = P_{5,8} = 1.00$

$P_{6,7} = 0.10 \qquad P_{6,9} = P_{6,10} = 0.90$

$P_{7,11} = 0.25 \qquad P_{7,9} = P_{7,10} = 0.75$

$P_{8,5} = 0.20 \qquad P_{8,11} = 0.80$

$P_{9,12} = P_{10,12} = 0.70 \quad P_{9,13} = P_{10,13} = 0.30$

$P_{11,1} = 0.15 \qquad P_{11,8} = 0.20$

$P_{11,13} = 0.50 \qquad P_{11,14} = 0.15$

$P_{12,2} = P_{12,13} = 0.50$

$P_{13,14} = 0.40 \qquad P_{13,15} = 0.60$

$P_{14,15} = 1.00$

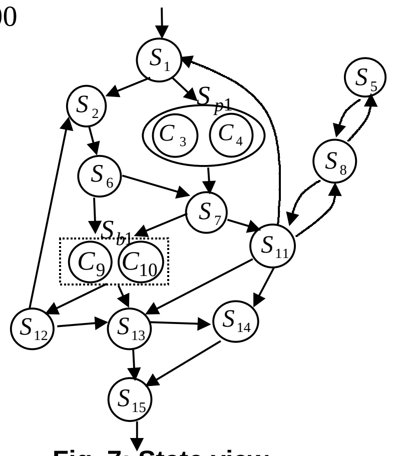

Fig. 7: State view

The state view of the system, which is depicted in Figure 7, is transformed from the architecture view shown in Figure 6. We obtain transition matrix M:

$$\begin{array}{c|ccccccccccccc}
 & S_1 & S_2 & S_{p1} & S_5 & S_6 & S_7 & S_8 & S_{b1} & S_{11} & S_{12} & S_{13} & S_{14} & S_{15} \\
\hline
S_1 & 0 & .3992 & .5988 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_2 & 0 & 0 & 0 & 0 & .99 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_{p1} & 0 & 0 & 0 & 0 & 0 & .9751 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_5 & 0 & 0 & 0 & 0 & 0 & 0 & .999 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_6 & 0 & 0 & 0 & 0 & 0 & .0985 & 0 & .8865 & 0 & 0 & 0 & 0 & 0 \\
S_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .747 & .249 & 0 & 0 & 0 & 0 \\
S_8 & 0 & 0 & 0 & .2 & 0 & 0 & 0 & 0 & .78 & 0 & 0 & 0 & 0 \\
S_{b1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .699986 & 0 & 0 & 0 \\
S_{11} & .1425 & 0 & 0 & 0 & 0 & 0 & .2 & 0 & 0 & .299994 & .475 & .1425 & 0 \\
S_{12} & 0 & .4825 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .4825 & 0 & 0 \\
S_{13} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .388 & .582 \\
S_{14} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & .98 \\
S_{15} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

$R_c = R_3 \times R_4 = 0.9751 \rightarrow$ parallel

$P_{1,c} = P_{1,3} = P_{1,4} = 0.6$

$P_{c,7} = P_{3,7} \times P_{4,7} = 1.00 \rightarrow$ parallel

$R_{b1} = R_9 + (1 - R_9) R_{10} = .99998 \rightarrow$ fault tolerance

$M(\{S_1\},\{S_c\}) = R_1 P_{1,c} = .998 \times .6 = .5988$

$M(\{S_c\},\{S_7\}) = R_c P_{c,7} = .9751 \times 1 = .9751$

$M(\{S_{11}\},\{S_8\}) = P_{11,8} = .2 \rightarrow$ call-and-return

$M(\{S_{b1}\},\{S_{12}\}) = R_{b1} P_{9,12} = .699986 \rightarrow$ fault tolerance

$M(\{S_{b1}\},\{S_{13}\}) = R_{b1} P_{9,13} = .299994 \rightarrow$ fault tolerance

$n = 13$

$|I - M| = 0.424631$

$|E| = 0.375892$

$$T(1,\{S_{15}\}) = (-1)^{n+1} \frac{|E|}{|I - M|} = 0.8852$$

The overall system reliability R is obtained as:

$$R = T(1,\{S_{15}\}) \times R_{15} = 0.8781$$

5 An Experiment

A simulator of an ATM bank system is utilized to validate the correctness of our architecture-based reliability model. This software consists of 11 components and contains six natural faults. We conducted an experiment on this software to obtain the reliability of each component and the overall system reliability. The operational behaviors in this experiment were collected to calculate the transition probability between components.

According to those six faults, seven versions of the system were constructed, in which version 7 contains all the faults but the others contain one fault each. For the first six versions, we randomly generated inputs to estimate the reliability of each individual faulty component until it is converged. Similarly, to compute the reliability of the entire system, version 7 was used.

The reliability of each individual component, the transition probabilities between components, and the overall system reliability through the experiment are listed as follows:

$R_1 = 1.0 \qquad R_2 = 0.982 \quad R_3 = 0.97$

$R_4 = 0.96 \qquad R_5 = 1.0 \qquad R_6 = 0.996$

$R_7 = 0.99 \qquad R_8 = 1.0 \qquad R_9 = 1.0$

$R_{10} = 0.8999 \quad R_{11} = 1.0$

$P_{1,2} = 1.0$

$P_{2,3} = P_{2,4} = 0.999 \qquad P_{2,11} = 0.001$

$P_{3,5} = 0.227 \qquad P_{3,6} = 0.669 \qquad P_{3,8} = 0.104$

$P_{4,5} = 0.227 \qquad P_{4,6} = 0.669 \qquad P_{4,8} = 0.104$

$P_{5,2} = 0.048 \qquad P_{5,6} = 0.951 \qquad P_{5,11} = 0.001$

$P_{6,3} = 0.4239 \qquad P_{6,4} = 0.4239 \qquad P_{6,7} = 0.1$

$P_{6,9} = 0.4149 \qquad P_{6,11} = 0.0612$

$P_{7,6} = P_{8,6} = 1.0$

$P_{9,6} = 0.01 \qquad P_{9,10} = 0.99$

$P_{10,6} = 1.0$

System reliability R : 0.526

Architecture-Based Reliability Model:

In this system, three architectural styles, which include batch-sequential, call-and-return, and fault tolerance styles, are identified.

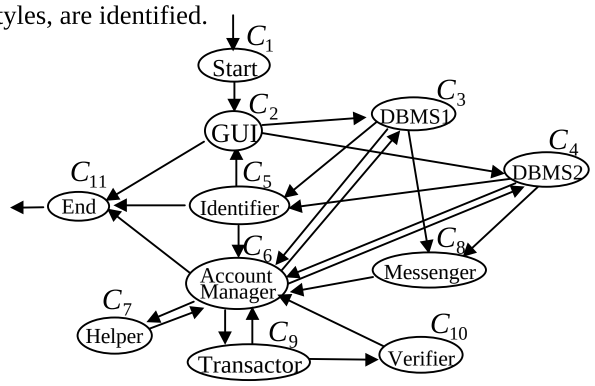

Fig. 8: Architecture view

Figure 8 shows the architecture view of this system. The *Start* component is the initial component while the *End* component is the final component. Basically, the system is running in batch-sequential manner in addition to the following. Components *DBMS1* and *DBMS2* are categorized into fault tolerance style where *DBMS2* is a backup component of *DBMS1* to increase the fault tolerance capability. Components *Account Manager* and *Helper* form a call-and-return style in which *Helper* component can be called multiple times before *Account Manager* transfers its control authority to others.

With the architecture view and the information of style integration, a state view diagram is mapped shown in Figure 9. From the state diagram, matrix M is shown as follows:

$$\begin{array}{c|cccccccccc}
 & S_1 & S_2 & S_{b1} & S_5 & S_6 & S_7 & S_8 & S_9 & S_{10} & S_{11} \\
\hline
S_1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
S_2 & 0 & 0 & .981018 & 0 & 0 & 0 & 0 & 0 & 0 & .000982 \\
S_{b1} & 0 & 0 & 0 & .2267 & .6682 & 0 & .1039 & 0 & 0 & 0 \\
S_5 & 0 & .048 & 0 & 0 & .951 & 0 & .104 & 0 & 0 & .001 \\
S_6 & 0 & 0 & .4222 & 0 & 0 & .1 & 0 & .4132 & .1 & .061 \\
S_7 & 0 & 0 & 0 & 0 & .99 & 0 & 0 & 0 & 0 & 0 \\
S_8 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
S_9 & 0 & 0 & 0 & 0 & .01 & 0 & 0 & 0 & .99 & 0 \\
S_{10} & 0 & 0 & 0 & 0 & .8999 & 0 & 0 & 0 & 0 & 0 \\
S_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

$n = 10$

$|I - M| = 0.106059$

$|E| = \text{-}0.059289$

$$T(1, \{S_{11}\}) = (-1)^{n+1} \frac{|E|}{|I - M|} = 0.559$$

The overall system reliability R is obtained as:

$R = T(1, \{S_{11}\}) \times R_{11} = 0.56$

In comparison with the reliability, 0.526, obtained from the experiment, we observed the difference between these two values is 3.4%, which gives us a promising result of this study.

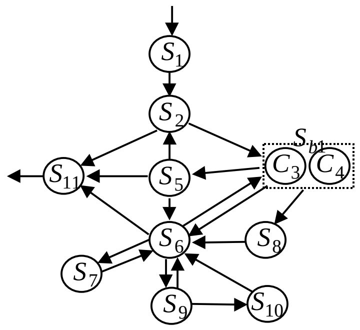

Fig. 9: State View

6 Conclusions and Future Work

We present a heterogeneous reliability model based on the characteristics of architectural styles. The reliability of the system depends not only on the reliabilities of the components and connectors, but also on the transition probabilities.

Similar models have been proposed in the past for modeling software reliability [6, 12, 16, 17], and much work can grow out of this research topic. Our ongoing research projects in this area are focused in the following two directions: (1) Enhancing this study horizontally and vertically by evaluating other architectural styles across various application domains, and incorporating design metrics into the software architecture evaluation. (2) Extending the style-based discrete reliability model to a continuous reliability model which takes the MTTF of each component and connector as the input parameters and predicts the reliability of the application, where the MTTF takes the component/connector and process failures into account.

The success of this research will promote more effective architecture design processes and thereby contribute to the improvement of software development.

References

[1] G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northrop, and A. Zaremski. Recommended best industrial practice for software architecture evaluation. Technical Report CMU/SEI-96-TR- 025 and ESC-TR-96-025, Jan uary 1997.

[2] R. Allen and D. Garlan. Formal connectors. Technical Report CMU-CS-94-115, Carnegie Mellon University, 1994.

[3] R. Allen and D. Garlan. Formalizing architectural connection. In Proceedings of the Sixteenth International Conference on Software Engineering, Sorrento, Italy, May 1994.

[4] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice.* Addison Wesley Longman, Inc., 1998.

[5] M. H. Chen, M. H. Tang, and W. L. Wang. Effect of software architecture configuration on the reliability and performance estimation. In *Proceedings of the 1998 IEEE Workshop on Application-Specific Software Engineering and Techno l-ogy*, March 1998.

[6] R. C. Cheung. A User-Oriented Software Reliability Model. *IEEE Transactions On Software Engineering*, 6(2):118, March 1980.

[7] P. C. Clements. Coming attractions in software architecture. Technical Report CMU/SEI-96-TR-008 and ESC-TR-96-008, Carnegie Mellon University, Software Engineering Institute, January 1996.

[8] G. Dutton and D. Sims. Patterns in oo design and code could improve reuse. *IEEE Software*, 11(3):101, May 1994.

[9] D. Garlan. What is style? In Proceedings of Dagshtul Workshop on Software Architecture, February 1995.

[10] D. Garlan, R. Allen, and J. Ockerbloom. Exploiting style in architectural design environments. In *Proceedings of SIGSOFT'94: Foundations of Software Engineering*. ACM Press, December 1994.

[11] D. Garlan and M. Shaw. An introduction to software architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering,* volume 1. World Scientific Publishing Company, 1993.

[12] A. L. Goel and K. Okumoto. A Markovian model for reliability and other Performance measures of software systems. *National Computer Conference,* 1979.

[13] S. S. Gokhale, W. E. Wong, K. S. Trivedi, and J. R. Horgan. An analytical approach to architecture-based software reliability prediction. In *Proceedings of IEEE International Computer Performance and Dependability Symposium (IPDS),* September 1998.

[14] S. S. Gokhale, M. R. Lyu, K. S. Trivedi. Reliability Simulation of Component-Based Software Systems. In *Proceedings of ISSRE*'98, pp 192-201.

[15] R. Kazman, G. Abowd, L. Bass and P.Clements. Scenario-Based Analysis of Software Architecture. *IEEE Software*, November 1996.

[16] Laprie and K. Kanoun, Software reliability and system reliability, Handbook of Software Reliability Engineering, pp 27-70, McGraw-Hill, New York, 1996.

[17] B. Littlewood. A Reliability Model for Systems with Markov Structure. *Applied Statistics,* 24(2):172, 1975.

[18] N. Medvidovic, P. Oreizy, J. E. Robbins, and R. N. Taylor. Using object-oriented typing to support a r-chitectural design in the c2 style. In *Proceedings of SIGSOFT'96: The Fourth Symposium on the Foundations of Software Engineering (FSE4), San Francisco, CA,* October 1996.

[19] N. Medvidovic, P. Oreizy, and R. N. Taylor. Reuse of off-the-shelf components in c2- style archtiectures. In *Proceedings of 19th International Conference on Software Engineering,* May 1997.

[20] J. Musa, A. Iannino, K. Okumoto, Software Reliability Measurement, Prediction, Application. McGraw-Hill, 1987.

[21] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Eng. Notes,* 17(4):40--52, October 1992.

[22] M. Shaw. Procedure calls are the assembly language of system interconnection: Connectors deserve first-class status. In *Proceedings of the Workshop on Studies of Software Design,* May 1993.

[23] M. Shaw. Software architectures for shared information systems. Technical Report CMU-CS-93-126, Carnegie Mellon University, School of Computer Science, 1993.

[24] M. Shaw and D. Garlan. Characteristics of higher-level languages for software architecture. Technical Report CMU-CS-94-210, Carnegie Mellon University, December 1994.

[25] M. Shaw and D. Garlan. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, 1996.

[26] B. Spitznagel and D. Garlan. Architecture based performance analysis. In Proceedings of the 1998 Conference on Software Engineering and Knowledge Engineering, 1998.

[27] R. N. Taylor, N. Medvidovic, K. M. Anderson, E. J. Whitehead Jr., J. E. Robbins, K. A. Nies, P. Oreizy, and D. L. Dubrow. A component and message-based architectural style for gui software. *IEEE Transactions on Software Engineering,* June 1996.

[28] W. Tracz. Dssa frequently asked questions (faq). *ACM Software Engineering Notes,* 19(2):52--56, June 1995.

[29] W. L. Wang, M. H. Tang, and M. H. Chen. Software Architecture Analysis – A Case Study, submitted to CompSac 99.