A Reliability Model Based on Heterogeneous Software Architecture

Chang Liu, Bin Liu, Lian Ruan
Department of System Engineering of Engineering Technology
Beihang University
Beijing, China
liuchang@dse.buaa.edu.cn

Abstract—Application of existing architecture-based software reliability models is strictly limited to software in which the component transitions satisfy the Markov properties. When software has both deterministic and probabilistic behaviors, these models intentionally model it as a Markov process, of which estimation of reliability with poor accuracy come as a result. The objective of this paper is to present a new reliability model that can resolve the heterogeneous software architectures with or without Markov properties. We present an algorithm generating Markov model from heterogeneous software architecture modeled in UML sequence diagram. A comparison with existing models is given with a case study.

Keywords—software reliability; software architecture; heterogeneous architecture; software component; Markov chain; UML sequence

I. INTRODUCTION

With the rapid progress of software architecture, software developers begin to treat software as a composition of some collaborating components instead of a whole entity. In traditional “black-box” software reliability models, software is considered as a whole and only its interactions with the outside world are modeled. These approaches are not applicable to the component-based, large-scale software systems any more. The architecture-based reliability models taking consideration of software internal structures have become focus of researches recently.

According to the summaries of the architecture-based reliability models [1,2,3], the state-based and path-based models are most frequently studied and used. State-based models [4,5,6,7] define a software state as an execution of a component and model the software architecture as a Markov chain. Path-based models [8,9,10] take a sequence of components in a typical run as an execution path. And the software reliability is computed based on reliabilities and frequencies of the paths.

Most of existing models above assume that software architectures have Markov properties. When this hypothesis is not satisfied, the application of these models will encounter problems. At present, the reliability models tackling on this kind of software architectures are rare. The model by Wen-Li Wang and Mei-Hwa Chen [11] has focused on this aspect but it is too complex to apply in software design process. A Heterogeneous Architecture-Based Reliability (HABR) model is presented in this paper. It can model the heterogeneous software architecture which means having both deterministic and probabilistic component transitions and transfer software architecture from heterogeneous model into Markov model. A more accurate estimation of reliability can be made using HABR model compared with the existing models.

II. LIMITATIONS OF EXISTING MODELS

Software system reliability can be represented as function $R_s=f(R_i,p_{ij})$, in which $R_s$ is system reliability; $R_i$ is reliability of component $i$; $p_{ij}$ is transition probability from component $i$ to $j$.

Component transition probabilities represent the frequencies of the interactions between components. State-based models use the transition probabilities as the elements in Markov matrix and traversal the matrix to compute the system reliability. Path-based model [10] also takes advantage of transition probabilities to get the execution paths. Both of them assume that component transitions are stochastic processes with the probability $p_{ij}=n_{ij}/n_i$, in which $n_{ij}$ is number of times component $i$ transfer to component $j$; $n_i=\sum n_{ij}$ is number of times component $i$ transfers to all components it can reach; $\sum p_{ij}=1$.

These models can yield reliability estimation with good accuracy when component transitions satisfy Markov properties. But when deterministic transitions exist between components, the estimation will lose accuracy. Fig. 1 is an example for explicit illustration.

![Figure 1. Deterministic component transitions.](image)

Component $A$ calls $B$ and $C$ in sequence in each execution. Application control flows to component $D$ after $C$ returns to $A$.
and terminates after \( D \) finishes execution. Transition probabilities are \( p_{i,x} = p_{i,x} = 1/3 \) and \( p_{i,x} = p_{i,x} = 1 \). If each component reliability is assumed to \( r \), the application reliability is \( r^r/(3-r^r) \) according to state-model [4,5] and \( 2r^3/3 + r^3/3 \) to path-model [10]. But there is only one execution path actually, that is \( A_1; A_2; C_1; D \), in which \( A_1; A_2; A_3 \) forms an entire execution of \( A \). The reliability is \( r^r \) according to system reliability model [12]. The significant difference exists among three models above. The causes of this situation are summarized below:

1) Existing models did not take the deterministic behaviors into account and intentionally model the deterministic behaviors as stochastic processes with transition probabilities. The purpose of using transition probabilities is to find different execution paths. Deterministic behaviors have no effects on execution paths at all. So they cannot be treated as stochastic processes and counted transition probabilities by frequencies when a component calls some other components or uncounted transition probabilities by probabilities. The purpose of using transition probabilities is to conceal component interactions details are concealed.

2) To callee, the transition probability equal to 1 when it returns to caller, which results in loops that do not really exist in actual run between caller and callee.

3) Existing models are lack of means of differentiating component transition patterns. Nowadays software architecture modeling methods have been able to describe the component interactions in explicit ways. But it seems that reliability models did not take fully advantage of them. The component interactions details are concealed.

III. SOFTWARE ARCHITECTURE MODEL IN UML SEQUENCE DIAGRAM

HABR model uses UML sequence diagram [13] to model software architecture. UML sequence diagram can define time orders and patterns of component interactions in detail by which the third cause in section 2 could be resolved. For the purpose of estimating reliability, some formalization extension need to be made based on previous works [14,15].

**Definition 1:** A sequence diagram is defined by the tuple

\[ sd=(I, L, M, msg, source, target) \]

**Definition 2:** The interacting components set in \( sd \) is

\[ I = \{l_1, l_2, \ldots, l_n\} \]

**Definition 3:** \( K_i \) is set of locations sending or receiving messages in life line of component \( i \) and \( L \) is whole set in \( sd \)

\[ K_i = \{0,1, \ldots, l_n\} \]

\[ L = \{<i,l,x>|i \in I, l \in K_i, if \ l \ is \ sending \ location \ x=1 \ else \ x=2\} \]

**Definition 4:** A message set \( M \) contains message type, guard conditions and ID

\[ M \subseteq \text{MessageType} \times \text{GuardConditions} \times \text{ID} \]

**Definition 5:** \( msg \) is a function mapping locations to a message

\[ msg: L \rightarrow P(M) \]

**Definition 6:** source and target are functions mapping a message to a component, \( \forall <i,l,x> \in L, mid \in M: \)

\[ \text{source: mid} \rightarrow i, \text{if}(mid \in \text{msg}(<i,l,x>)) \]

\[ \text{target: mid} \rightarrow i, \text{if}(mid \in \text{msg}(<i,l,x>)) \]

**Definition 7:** Locations in \( sd \) have partial ordering \( \leq_L \times L \) which satisfies:

1) \( \forall i,l \in I, l \in K_i, x \in \{!, ?\}; <i,l,x> \leq <i,l+1,x> \)

2) \( \forall i,l,l' \in I, l,l' \in K_i, mid \in M: \)

\[ mid \in \text{msg}(<i,l,x>) \wedge mid \in \text{msg}(<i',l',x>) \Rightarrow <i,l,x> \leq <i',l',x> \]

3) Transfer closure property formed by 1) and 2)

**Theorem 1:** A message is unique in an \( sd \)

\[ \forall <i,l,x>, <i',l',x> \in L: \]

\[ mid \in \text{msg}(<i,l,x>) \wedge mid \in \text{msg}(<i',l',x>) \Rightarrow <i,l,x> = <i',l',x> \]

**Theorem 2:** A location can have more than one sending messages but only one receiving message. Sending and receiving cannot occur at the same time

\[ \forall <i,l,x> \in L, mid \in \text{msg}(<i,l,x>); (#(msg(<i,l,x>))>1 \Rightarrow x = ! \}

**Definition 10:** \( \Leftarrow \) is a partial ordering of messages in \( sd \)

\[ \forall mid, mid' \in M, \exists <i,l,x> \in L, s.t. \]

\[ msg(<i,l,x>) = mid \wedge \exists <i',l',x>' \in L, s.t. msg(<i',l',x>') = mid'; \]

\[ <i,l,x> \leq <i',l',x>' \Rightarrow mid \Rightarrow mid' \]

IV. RELIABILITY ANALYSIS BASED ON COMPONENT TRANSITION PATTERNS

Component transition patterns can be modeled base on Definition 1–6. The transition orders can be obtained based on locations and messages partial orderings in Definition 7–8 and Theorem 1–2. In this section we show how to generate Markov model from the heterogeneous model using the architecture information in UML sequence diagram. In this course the first and second cause in section 2 are treated appropriately.

A. Cheung’s Markov Reliability Model

It is assumed that software is a composition of individual components and a software state is defined as a component executing. And component failures are independent with each other and each causes system failure. The transitions between components are independent from its execution history. Then the software satisfies the Discrete Time Markov Chain [4]. If there are \( n \) components \( \{C_1, C_2, \ldots, C_n\} \) in the software among which \( C_i \) is the starting and \( C_n \) is the ending component in execution path. Each state \( i \) in the Markov chain is represented by execution of component \( C_i \) with reliability \( R_i \). The transition probability between component \( C_i \) and \( C_j \) is \( p_{i,j} \). Then the transition matrix of Markov chain is:

\[
\begin{pmatrix}
0 & R_{p_{11}} & 0 & \cdots & 0 \\
R_{p_{21}} & 0 & R_{p_{22}} & \cdots & 0 \\
0 & R_{p_{32}} & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & 0
\end{pmatrix}
\]
\(M^k(1,j)\) is the probability of component \(i\) transfer to \(j\) after \(k\) steps transitions. Thus \(M^k(1,n)\) is the probability of component \(i\) transfer to \(n\) after \(k\) steps transitions.

Let \(T^k=I+M^k+M^{2k}+\ldots+\sum_{k=0}^{\infty}M^k=(1-I)^{-1}[E/|I-M|]\) in which \(I\) is the identity matrix with same dimension of \(M\), and \(E\) is remaining matrix excluding the \(n\)th row and first column of the matrix \((I-M)\). The overall software system reliability can be computed as \(R_n=T(1,n)R_0\).

### B. Deterministic Transition Pattern

If there exist deterministic transitions between \(C_i\) and \(\{C_1,C_2,\ldots,C_m\}\), which means \(C_i\) calls these \(m\) components in a predefined sequence and terminates finally. In this pattern, we define the “MessageType” in UML sequence diagram as “call” and “GuardConditions” as NULL. The return messages from \(\{C_1,C_2,\ldots,C_m\}\) to \(C_i\) is defined as “return”.

According to existing reliability models, there are \(p_j=1/(m+1)\) and \(M_j=Rp_j\), which has been proved incorrect in section 2. Reconsiderations of transition relationships are given below.

1) Component \(C_i\) executes and calls \(\{C_1,C_2,\ldots,C_m\}\) in sequence. In this process \(\{C_1,C_2,\ldots,C_m\}\) execute in a sequential pattern [5] in which each component executes exactly once in sequence when \(C_i\) is considered as a container for \(\{C_1,C_2,\ldots,C_m\}\). So from the reliability view, the return messages from \(\{C_1,C_2,\ldots,C_m\}\) to \(C_i\) do not affect the reliability estimation in this process. We define transitions \(p_{12}=p_{23}=\ldots=p_{m-1m}=1\) to represent this sequential pattern and do not consider their return messages to \(C_i\) any more.

2) When the process terminates from \(C_n\), it also executes just once. It is a serial system that \(C_i\) and \(\{C_1,C_2,\ldots,C_m\}\) form. According to the partial ordering defined in UML sequence model, we make a transition from \(C_i\) to \(C_1\) its first target component \(C_i\) and let the transition probability \(p_{1i}=1\). After this procedure the reliability of \(C_i\) is computed repeatedly any more.

3) The component \(C_m\) is the last component in the path now. This is because the reliability of \(C_i\) needs to be considered only once before \(C_i\) transfers control to ending state “r”, which has been done above. So we modify the return from \(C_m\) to \(C_i\) to a new transition from \(C_m\) to ending state “r” with the probability \(p_{mi}=1\).

Using the methods above, we can generate a Markov chain from a sequence of components having deterministic transition patterns.

### C. Probabilistic Transition Pattern

Probabilistic transition is a stochastic process. Almost all existing reliability models are based on probabilistic transition pattern. In this pattern the messages branch from the locations where the probabilistic transitions happen. If there are probabilistic transitions between \(C_i\) and \(\{C_1,C_2,\ldots,C_j\}\), then “MessageType” is defined as “choose” and “GuardConditions” is filled with the value of branching message’s guard conditions. There are different values for each target component in \(\{C_1,C_2,\ldots,C_j\}\) and \(C_i\) chooses only one of them to run each time according to a specific value.

Let \(P(GuardConditions=C_j)\) represent the probability of satisfying the guard conditions of transition from \(C_i\) to \(C_j\). The transition probability from \(C_i\) to \(C_j\) is now defined as \(p_{ij}=P(GuardConditions=C_j)\). What makes this new definition of transition probability more accurate and different from existing reliability models is that it limits transition probabilities to a set of target components with the same left side of guard conditions expression and excludes deterministic transitions and probabilistic transitions with other guard conditions.

### D. Heterogeneous transition patterns

Heterogeneous pattern is that there have both deterministic and probabilistic transitions in software architecture. Our methods can still be applied to heterogeneous pattern by using the same procedures iteratively. We treat locations having either deterministic or probabilistic sending messages as starting points of some individual paths. For each location, we traverse along with the outgoing transitions until there is no location sending message in a component. The present component is then defined as a termination of this path. The termination component transition is assigned to the target component of the next sending location after the starting location of this path. The Markov model can be generated in this way by traversing the whole UML sequence diagram with modifying components transition probabilities. The specific Markov model generating algorithm from UML sequence diagram is given in next section.

### E. Algorithm of UML sequence diagram to Markov model

Based on the analysis above, we can generate Markov chain from software architecture modeled in UML sequence diagram. Fig. 2 is the specific algorithm.
Fig. 3 and Fig. 4 are parts of pseudo-code of the algorithm. The algorithm of traversal of the components in UML sequence diagram is given in Fig. 3. The components transitions formed with probabilities according to transition patterns in depicted in Fig. 4.

```java
int current = HLML //current state
(void) current != ending point
{
    Form transition(current, 1);
    if (current != ending point) // present component has no sending messages
    {
        while (stack is not empty)
        {
            POP(S, location) //pop component and message by time order
            if (current != first tagged location) //Form transition label of next message of the path starting component
                return;
            else
                push(S, location); //Form transition structure
                if (current != first tagged location)
                {
                    next(S, 0, pt, bp, new transition structure);
                    push(S, location);
                }
                else
                    break;
            while (current != ending point)
            {
                Form transition(1, location); //repeat the traversal iteratively
                current = ending point;
            }
        }
    }
}
```

Figure 3. Algorithm of traversal of UML sequence diagram

### V. CASE STUDY

In this section we apply HABR model to a case and compare the estimation result with existing reliability models [4,5]. The case is illustrated in Fig. 5. The application begins from component $C_1$ which calls $C_2$, $C_3$ and $C_4$ in deterministic sequence and terminates when $C_1$ finishing execution. Component $C_2$ calls either $C_5$ or $C_6$ according to branching guard conditions. Component $C_3$ calls $C_7$ in deterministic pattern. The reliabilities of components are listed in Table 1. The probabilistic distribution of branching guard conditions is: $P(condition=5) = 0.75$; $P(condition=6) = 0.25$.

<table>
<thead>
<tr>
<th>Component</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_1$</td>
<td>0.99</td>
</tr>
<tr>
<td>$C_2$</td>
<td>0.98</td>
</tr>
<tr>
<td>$C_3$</td>
<td>0.95</td>
</tr>
<tr>
<td>$C_4$</td>
<td>0.99</td>
</tr>
<tr>
<td>$C_5$</td>
<td>0.90</td>
</tr>
<tr>
<td>$C_6$</td>
<td>0.89</td>
</tr>
<tr>
<td>$C_7$</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Figure 6(a) and (b) are Markov models generated by state-based models [4,5] and HABR model respectively. The transition probabilities are marked besides the arrow headed lines.
The components transition probabilities and software system reliability $R_s$ are showed below:

- **State-based models** [4,5]:
  \[
P_{12}=p_{13}=p_{41}=0.25; \ p_{25}=0.375; \ p_{26}=0.125; \]
  \[
P_{21}=p_{31}=p_{32}=0.5; \ p_{52}=p_{62}=p_{14}=1; \]
  \[
R_s=0.8386
\]

- **HABR model**:  
  \[
P_{12}=p_{13}=p_{61}=p_{31}=p_{34}=1; \ p_{25}=0.75; \ p_{26}=0.25; \]
  \[
R_s=0.8108
\]

Using classical reliability system model [12], the “real” reliability of this case can also be computed as:

\[
R_s=R_1\times R_2\times(0.75R_3+0.25R_4)\times R_5\times R_6=0.8108
\]

The comparison of reliability estimation results can show that using HABR model will get closer estimation to the software system “real” reliability.

VI. CONCLUSIONS

In this paper, we have proposed a HABR model which offers a new method to reliability estimation based on software architecture. HABR model uses UML sequence diagram to model software architecture, which can be tightly integrated with software design process. The components transition patterns is explicitly defined and transition probabilities is properly computed according to the architecture model constrains in UML sequence diagram. HABR model can generate Markov chain from heterogeneous architecture not satisfying the Markov properties. And we have proved through a case that HABR model offers more accurate reliability estimation than existing models.

Compared with the only heterogeneous reliability model by Wen-Li Wang and Mei-Hwa Chen [11], HABR model is much more applicable by using UML sequence diagram as the input of reliability estimation. And it is a precondition of automation that the Markov chain generating algorithm is generally understandable and relatively simple. The implementation of the estimation tool based on HABR model is our next work.

REFERENCES


