International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:11, 2009

Building the Reliability Prediction Model of Component-Based Software Architectures

Pham Thanh Trung, Huynh Quyet Thang

***Abstract*—**Reliability is one of the most important quality attributes of software. Based on the approach of Reussner and the approach of Cheung, we proposed the reliability prediction model of component-based software architectures. Also, the value of the model is shown through the experimental evaluation on a web server system.

***Keywords*—**component-based architecture, reliability prediction model, software reliability engineering.

I. Introduction

RELIABILITY is one of the most important quality attributes of software. The approach of Reussner [1] provides a way to predict the reliability on the basic of parametric contracts and Markov models. A clear understanding of the reliability from users' point of view has been introduced by Cheung [6]. He states that the fact that users rarely request a faulty service or users often request a faulty service must be considered in calculating the reliability.

In this paper, we present the research result about predicting the reliability of component-based software architectures that depends on reliabilities of components and operational profiles of use case packages. Developed mainly from [1, 6, 11, 12, 13], we proposed to build the reliability prediction model of component-based software architectures and showed the value of the model. The content of this paper is presented as following: Section 2 presents the component-based software reliability based on the reliabilities of resources and components. Section 3 presents the component-based software reliability based on operational profiles. Section 4 presents the reliability prediction model based on the combination of the two models above. Section 5 presents the experimental evaluation.

Manuscript received January 20, 2009. This work was supported in part by the Vietnam's Ministry of Science and Technology under Grant KHCB2.034.06.

Pham Thanh Trung is system software engineerer in Dicom Vietnam Corpration. (E-mail: pthanhtrung82@yahoo.com).

Huynh Quyet Thang is the Associate Professor at Software Engineering Department, Faculty of Information Technology, Hanoi University of Technology, Hanoi, Vietnam (E-mail: thanghq@it-hut.edu.vn).

II. Resource-Dependent Reliability

A. The differences in reliability behaviors between resources and components

A software component can only fail during its execution because a fault procedure of a component needs to be executed to cause a failure. From that, the reliability of a software component is a function of usage profiles. On the other hand, a resource can fail at any time independent of its usages, that is, the reliability of a resource is a function of time.

Thus, we must consider this difference when calculating the reliability. A resource must be available until the last usage is finished. Therefore, we need to determine the time a resource must be available, that is, the time until the last usage is finished.

The approach of Reussner [1] provides a way to predict the reliability on the basic of parametric contracts and Markov models. There, the entry R(i, j) of the potential matrix of a Markov model is the expected number of visits to the state s_j from the state s_i. The fact that the potential matrix contains the expected numbers brings us an idea to use these values to calculate the execution time of a service.

A clear understanding of the reliability from users' point of view has been introduced by Cheung [6]. He states that the fact that users rarely request a faulty service or users often request a faulty service must be considered in calculating the reliability. Therefore, we can extend his view of reliability: if a faulty resource is not used, it will not influence the reliability from users' point of view.

B. Calculating the time consumption of a service

The only entities that consume time in a Markov model are states (if not, it can be easily transformed into this form). Therefore, if we have number of visits to each state on an arbitrary path through the model, we can calculate the expected time consumption of a service.

Assume that we have a Markov model $MM_d = (\sum, S, F, s_0, \delta, u)$ for a service d and that we have the expected number of visits to each state in MM_d and its expected time consumption. From that, we can calculate the total time consumption of the service d.

We have:

$$E[X_d] = \sum_{s \in S} v_s * E[X_s]$$

where v_S is the expected number of visits to state s, the

$E[X_S]$ is the expected time consumption of state s and $E[X_d]$ is the total time consumption of the service d.

C. Calculating the usage period of a resource

As above, in order to have a more accurate reliability prediction, we need to determine the last time the service uses the resource.

1) The simplest case (one resource – one state)

In this case, each state is associated with a single and unique resource that is always used during the visit to the state.

Assume that we have the service effect specification of service d as Markov model MM_d. Each state s_j of MM_d uses a single and unique resource r_j and use this resource during the whole execution.

The time that resource r_j must be available is equal to the time consumption of the service until the last visit to the state s_j. Assume that we have an arbitrary path from the start state to the final state:

$$s_0 s_{i1} \dots s_j \dots s_{ik} s_{ik+1} \dots s_j \dots s_f$$

Let V(i, j) be the expected number of visits to each state s_i before the last visit to the state s_j and let $E[X_j]$ be the expected execution time of state s_j. After that, the total time consumption until the last visit to s_j is:

$$E\left|X_{r_j}\right| = \sum_{s_i \in S} V(i,j) * E\left|X_i\right|$$

In this case, the last usage of the associated resource r_j is equivalent to the last visit to s_j From that, $E\left|X_{r_j}\right|$ is the time that the resource r_j must be available.

Now, we need to determine V(i, j). Let a_j be the beginning part of the path until the last visit to s_j. From that, we have:

$$V(j, j) = R(1, j)$$

where R is the potential matrix of the Markov model MM_d because all the visits to s_j must be in a_j.

For each other state s_i, we need to determine the fraction of the expected number of visits R(1, i) that is in the beginning part a_j. Let take a look at the definition of the matrix F of the Markov model: each entry F(i, j) is the probability to visit the state s_j from the state s_i on an arbitrary path. The entry F(i, j) is calculated as following:

$$F(i,j) = \begin{cases} \dfrac{R(i,j)}{R(j,j)} & if\ s_i \neq s_j \\ 1 - \dfrac{1}{R(i,i)} & otherwise \end{cases}$$

Therefore, the expected number of visits to s_i until the last visit to s_j is

$$V(i, j) = R(1, i) * F(i, j).$$

for all states s_i with $i \neq j$. Then, the product R(1, i) * F(i, j) is the fraction of the expected number of visits to the state s_i before the state s_j is visited.

2) The extended case (one resource – several states)

In this case, several states are associated with the same resource r. Assume that we have the service effect specification of service d as Markov model MM_d and that we have S_r as the set of states associated with resource r.

We want the Markov model $MM_{r,d}$ to stop its execution whenever the resource r is used. To do this, a new Markov model $MM_{r,d}$ in which the states s_j in S_r are the final states is formed from MM_d by removing all the outgoing transitions from each state in S_r, that is, the transition matrix P_r of $MM_{r,d}$ contains a row of zeros for every state in S_r. Also, all the final states of $MM_{r,d}$ form a probability distribution. Because the execution must reach exactly one final state, the sum of the number of visits to all final states must be one. The value $R_r(i, j)$ for the state s_j in S_r is the probability to visit s_j from the state s_i. Moreover, the values $R_r(i, n)$ are the probabilities of going from the state s_i to the final state s_n without using the resource r. The probability $\omega_r(i)$ of using the resource r after or during the visit to the state s_i is given as following:

$$\omega_r(i) = \sum_{s_j \in S_r} R_r(i,j)$$

Because the final states form a probability distribution, we have:

$$\omega_r(i) = 1 - R_r(i,n)$$

After that, the expected number of visits to the state s_i before the last usage of the resource r is:

$$\upsilon_r(i) = \begin{cases} R(1,i) & if\ s_i\ in\ S_r \\ R(1,i) * \omega_r(i) & otherwise \end{cases}$$

Moreover, the expected time the resource r must be available is:

$$E[X_r] = \sum_{s_i \in S} \upsilon_r(i) * E[X_i]$$

3) The general case

In this case, a state requires the associated resources only for a part of its execution. Let $u_r(i)$ be the probability of using resource r during the execution of state s_i, let $X_{r,i}$ be the expected time of using resource r in state s_i and let $q_r(i)$ be the probability of using resource r after leaving state s_i. Figure 1 shows the relationship between the probabilities $u_r(i)$ and $q_r(i)$.

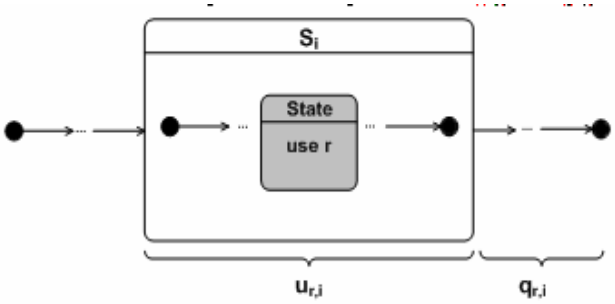

Fig. 1: The probabilities $u_r(i)$ and $q_r(i)$

Now, we can calculate $\omega_r(i)$ - the probability of using r on the path from the start state s_i to the final state of the superior Markov model or the probability of using r in s_i or after leaving s_i.

$$\omega_r(i) = 1 - (1 - u_r(i))(1 - q_r(i)) \quad (1)$$

Next, we transform the equation (1) into:

$$\bar{\omega}_r(i) = \bar{u}_r(i) * \bar{q}_r(i)$$

Where

$$\bar{u}_r(i) = 1 - u_r(i), \quad \bar{q}_r(i) = 1 - q_r(i)$$

and

$$\bar{\omega}_r(i) = 1 - \omega_r(i)$$

As above, we want the execution to stop when r is used, that is, the final state of MM_d is reached if and only if the resource is not used. Because there is a certain probability that the resource is not required, we cannot remove all outgoing transitions of states using r. That is , the execution must continue with probability $\bar{u}_r(i)$ and terminate with probability $u_r(i)$.

Next, we simplify the internal service effect specification of a state s_i to a single transition between the start and the final state with the probability $\bar{u}_r(i)$, that is, the execution stops in the internal start state of state s_i if the resource r is used and continues otherwise. Figure 2 shows the simplified Markov model.

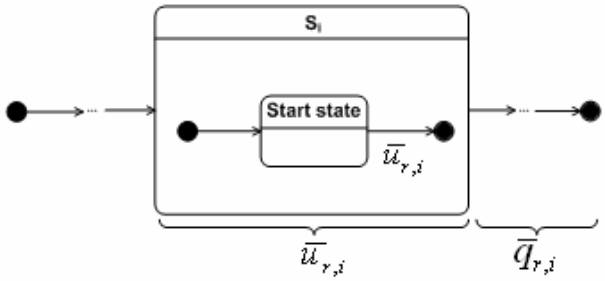

Fig. 2: The simplified service effect specification.

Let $MM_{r,d}$ be the composed Markov model using the simplified service effect specifications. Then the entry $R_r(i, n)$ of the potential matrix of $MM_{r,d}$ is the probability of going to the final state without using resource r. Because the final state can only be reached if the internal transition of s_i is taken and the resource is not used after leaving s_i, the entry associated with the internal start state of s_i corresponds to the probability $\bar{\omega}_r(i)$. Moreover, because the superior final state can only be reached if the resource is not used after leaving s_i, the entry associated with the internal final state corresponds to the probability $\bar{q}_r(i)$.

The insertion of the simplified version of s_i into the superior Markov model is equivalent to a multiplication of every path through s_i by the factor $\bar{u}_r(i)$. Therefore, instead of inserting an additional state and transition, we can integrate $\bar{u}_r(i)$ into the superior service effect specification by multiplying either every ingoing or outgoing transition of s_i by $\bar{u}_r(i)$.

- If we scale the outgoing transitions, $\bar{u}_r(i)$ is considered after visiting s_i. Then the entry $R_r(i, n)$ of the potential matrix contains the probability $\bar{\omega}_r(i)$ of not using r on a path from the internal start state of s_i to the final state of the superior Markov model.
- Otherwise, if we use $\bar{u}_r(i)$ to scale the ingoing transitions of s_i, the multiplication by $\bar{u}_r(i)$ is applied before visiting s_i. Then the entry $R_r(i,n)$ corresponds to the probability $\bar{q}_r(i)$.

Given the probabilities $\bar{u}_r(i)$ and $\bar{q}_r(i)$, the time consumption X_i and the usage period $X_{r,i}$ of resource r for each state s_i of the Markov model MM_d, we can calculate $X_{r,d}$, the expected time the resource r is used during the execution of service d:

$$E\left|X_{r,d}\right| = \sum_{s_i \in S} R_d(1,i)(q_r(i)E\left|X_i\right| + \bar{q}_r(i)E\left|X_{r,i}\right|)$$

D. Calculating the resource-dependent reliability

Until now, we can determine the usage period of a resource. The system reliability can be calculated if we have the usage period for each resource and their reliabilities. The reliability of a resource is a function of time which is assumed to be an exponential distribution. So, given the expected usage period of a resource $E[Y_r]$, the reliability for this period is $R(E[Y_r])$.

III. Operational Profile-Dependent Reliability

Using the Markov model as basic to give prediction of the reliability accepts an assumption: the service executed next depends only on the current state. So, the importance of the path through the control flow of the program is neglected, that is, the input data as well as the user behavior are neglected. Motivated by this assumption is quite popular in Markov-based reliability prediction, we have developed a way to build operational profile which accounts for the influence of the input data and the user behavior.

A use case with many bugs can seem reliable if the user spends so little time running it that none of the many bugs are found. Conversely, a use case that has few bugs can seem unreliable if the user spends so much time running it that all those few bugs are found.

Thus, we must consider this when calculating the reliability, that is, we must build operational profiles for the scenarios that make up a single use case and then operational profiles for a package of use cases.

A. Building the operational profile of use case scenarios

The problem is to calculate probabilities of use case scenarios. Figure 3 shows a use case with numbered blocks representing steps in the use case. This use case is made of six scenarios that represent different paths through the use case, they are: 1, 2, 3, 4; 1, 2, 3, 1; 1, 2, 3a, 4; 1, 2, 3a, 4a; 1, 2, 3a, 4b; 1, 2, 3a, 4c. In this, each path outgoing each step in the use case is assigned a probability. So for example, after a user has executed step 2, we expect that 80% of the time the user will go to step 3 next and 20% of the time they will go to step 3a. After probabilities are assigned to each path, the probability of

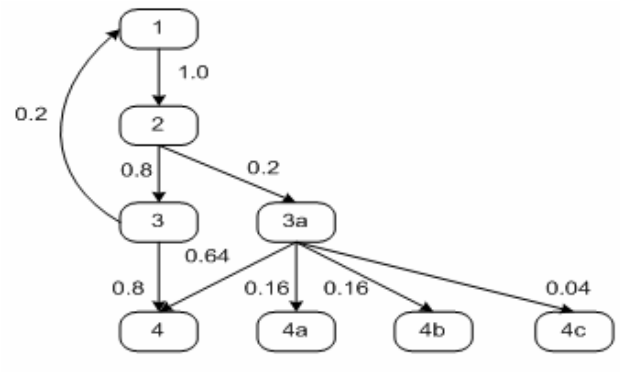

Fig. 3: Scenarios of a use case

each scenario is calculated by taking the product of the probabilities of paths in that scenario.

For example, the probability of scenario 1, 2, 3, 4 is 1 x 0.8 x 0.8 = 0.64

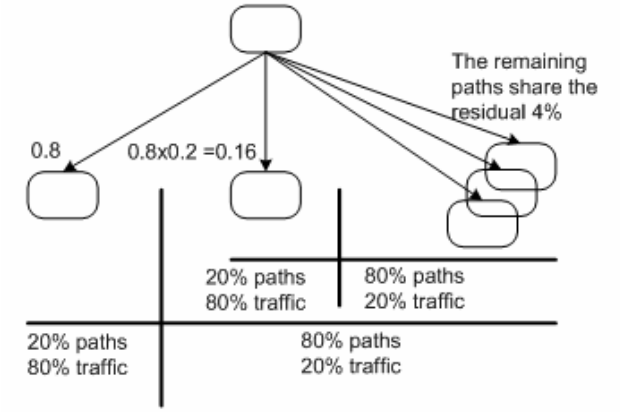

Fig. 4: 80% is allocated to a single use case outgoing path.

Ideally, we can conduct a usability study. Otherwise, we can apply the Pareto principle to operational profiles: 20% of the paths outgoing a use case step will carry 80% of the user traffic. We take the number of paths exiting a use case step and multiply by 20%; for most use cases this will yield 1 or 2 exit paths (e.g., 20% of 10 exit paths = 2). Distribute 80% of the traffic across these 1 or 2 steps. For the remaining steps, distribute 20% of the traffic across them.

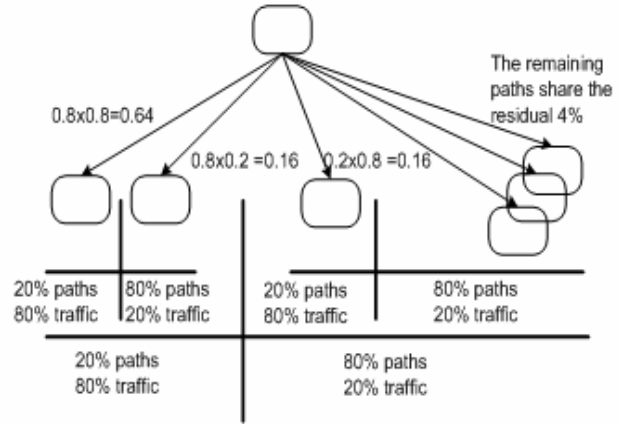

Fig. 5: 80% is allocated to two use case outgoing paths

Figure 4 and Figure 5 show the two examples of successive application of Pareto principle where the initial 80% of traffic is allocated to 1 and 2 exit paths, respectively.

B. Building the operational profile of a use case package

Until now, we can build the operational profile of use case scenarios. However, few projects deal with just a single use case, so next we must build the operational profile for a package of use cases with generalization relationship, extend relationship and include relationship as defined in UML 2.0. [4]. We'll proceed as the following:

- First, we estimate the usage frequency of base use cases.
- Next, we estimate the usage frequency of include and extend use cases used stand-alone.
- Finally, we estimate the usage frequency of include and extend use cases used by the base use cases.

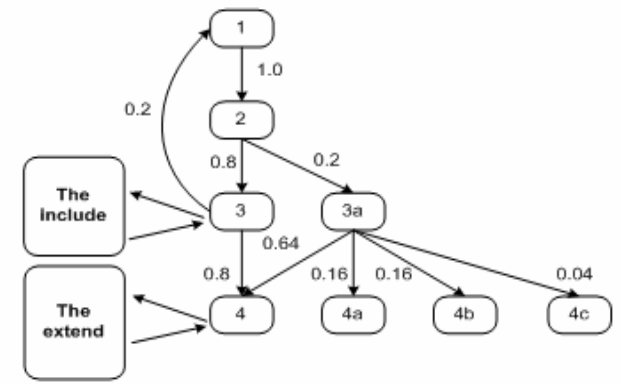

Fig. 6: The graph of the base use case.

In these three steps, step 1 and step are quite simple, only the step 3 need the probabilities that include use cases and extend use cases are actually used.

Inclusions and extensions in UML both have the property that flow of control returns to the base use case at the same point where the inclusion/extension took place (i.e., the inclusion point or extension point, respectively). [4]

Let's return to the graph of use case in Figure 3 and assume that it is the graph of base use case. Then let's say that steps 3 and 4 of the graph are the inclusion and extension points for an include use case and extend use case, respectively, as

TABLE I
PROBABILITY OF EACH SCENARIO IN FIGURE 3

Scenario	Probability
1, 2, 3, 4	0.64
1, 2, 3, 1	0.16
1, 2, 3a, 4	0.13
1, 2, 3a, 4a	0.03
1, 2, 3a, 4b	0.03
1, 2, 3a, 4c	0.01
Total	1.00

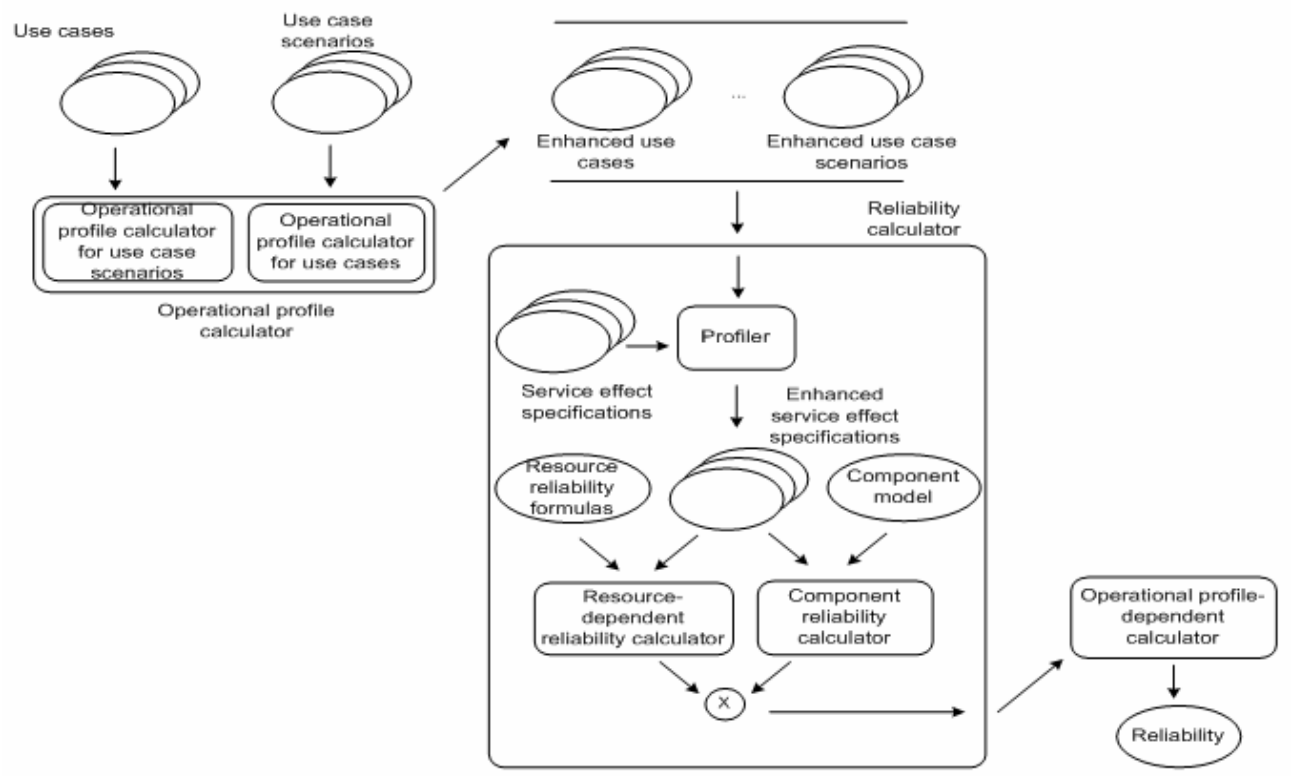

Fig. 7: The reliability prediction model of component-based software architectures

showed in Figure 6.

As above, we need the probability that the base use case will actually invoke the inclusion or the extension in Figure 6. Table I provides the probability of each scenario of Figure 3.

Because the inclusion is invoked in the step 3, that is, it's invoked in the scenarios 1, 2, 3, 4 and 1, 2, 3, 1, the probability that the base use case invokes the inclusion is:

0.64 (probability of 1, 2, 3, 4) + 0.16 (probability of 1, 2, 3, 1) = 0.80.

Similarly, the probability of invoking the extension is:

0.64 (probability of 1, 2, 3, 4) + 0.13 (probability of 1, 2, 3a, 4) = 0.77

However, remember the UML definition of an extend use case: the extension is "subject to specific conditions specified in the extension." [4]. If the condition, specified as part of the extend use case, is met, the extension is executed; otherwise, the base use case flow resumes as is. Of course, the Pareto principle says that 20% of the outgoing paths from a use case step will account for 80% of the traffic. Because we've only got two possible choices: extension or not, we take the probability of no extension at 0.8, that is, the probability for extension is 0.2.

Therefore, the probability that the base use case invokes the extension is the probability of the use case flow taking one of the scenarios on which the extension lies (scenarios 1, 2, 3, 4 and 1, 2, 3a, 4 in Figure 3) times the probability that the users actually wants an extension:

(0.64 + 0.13) * 0.2 = 0.15

C. Calculating the operational profile-dependent reliability

Assume that a use case j includes n scenarios numbered from 1 to n, and then the reliability of the system with regard to the use case j is given by:

$$r_j = \sum_{i=1}^{n} p_i r_i$$

Where p_i is the probability of scenario i, r_i is the reliability of the system with regard to scenario i.

Given the reliability of the system with regard to each use case, the reliability of the system with regard to use cases can be calculated. Assume that the system includes m use cases numbered from 1 to m, and then the reliability of the system with regard to the use cases is given by:

$$R = \sum_{j=1}^{m} p_j r_j$$

Where p_j is the probability of use case j, r_j is the reliability of the system with regard to the use case j.

IV. THE RELIABILITY PREDICTION MODEL OF COMPONENT-BASED SOFTWARE ARCHITECTURES

Fig. 7 shows the reliability prediction model of component-based software architectures:

1. Starting with use cases and use case scenarios, the results of analyzing the system, the operational profile calculator with two components which are a operational profile calculator for use case scenarios and a operational profile calculator for use cases gives the enhanced use cases with

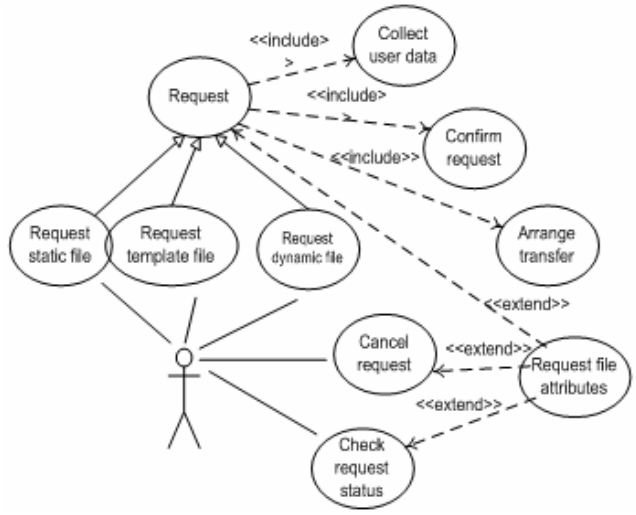

Fig. 8. Use case diagram of the web server system

the probabilities of use cases and the enhanced use case scenarios with the probabilities of use case scenarios.

2. Next, each of the enhanced use case scenarios is entered the reliability calculator. The principle of operation of the reliability calculator is described as following:

- The set of service effect specifications and the entered use case scenario form the inputs of the profiler; the output is a set of the enhanced service effect specifications with the information about transition probabilities and the execution times of each state. Basically, a profiler is a performance analysis tool that allows measuring the behavior of a program during execution, namely the frequency and the period among function calls. Nowadays, profilers are quite popular, for example JProbe for Java of Quest Software, dotTrace Profiler of JetBrains for .NET.
- With the enhanced service effect specifications and the component model [2], the component reliability calculator gives the reliability of the system (not include the resource-dependent reliability of the system). Basically, the component reliability calculator starts with calculating the reliability of the top level composition component and then recourses until a basic component is reached [1].
- Concurrently, the resource-dependent reliability calculator accepts the resource reliability formulas and the enhanced service effect specifications as inputs and gives the resource-dependent reliability of the system. Basically, the resource-dependent reliability calculator repeats the following for each resource:
  - Calculate the usage period of each resource and then use the corresponding resource reliability formula to calculate the resource-dependent reliability of the system with regard to each resource.
  - And the resource-dependent reliability of the system is the product of all resource-dependent reliabilities of all resources.
- Next, the reliability of the system with regard to the entered use case scenario is the product of the resource-dependent reliability of the system and the reliability of the system (not include the resource-dependent reliability).

3. Finally, the operational profile-dependent calculator is used to calculate the total reliability of the system.

Not mentioned in this model is the way to build up the service effect specifications, the component model and the resource reliability:

- The service effect specifications can be provided by component suppliers or can be generated from the source code of components.
- The component model [2] can be taken out from designing the system.

Base Use Cases	Times per Day	Probability
Request	0	0.00
Request static file	50	0.03
Request template file	350	0.20
Request dynamic file	25	0.01
Cancel request	25	0.01
Check request status	100	0.06

Probability of Use by Base Use Case			
Collect user data	Confirm request	Arrange transfer	Request file list
Include	Include	Include	Extend
0.80	0.80	0.80	0.15
0.80	0.80	0.80	0.15
0.80	0.80	0.80	0.15
0.80	0.80	0.80	0.15
			0.20
			0.20

Include & Extend Use Cases	Times per Day Standalone	Times per Day with Base Use Case	Times per Day	Probability
Collect user data	0	340	340	0.20
Confirm request	0	340	340	0.20
Arrange transfer	15	340	355	0.21
Request file attributes	40	89	129	0.08
TOTAL			1714	1.00

= 0 x 0.15 + 50 x 0.15 + 350 x 0.15 + 25 x 0.10 + 25 x 0.20 + 100 x 0.20

Fig. 9: The operational profile of use cases of the web server system.

- The resource reliability is assumed to have an exponential distribution where $1/\lambda$ is the rate of failure occurrences.

V. EXPERIMENTAL EVALUATION

The evaluated system is a web server. Fig. 8 shows the use case diagram of the web server system with relationships between use cases.

Fig. 9 shows the operational profiles of use cases of the web server system. Table 2 shows the logged information during the execution of the web server by using a profiler.

The logged information from table 2 can be used in conjunction with the service effect specifications to get the enhanced service effect specifications with transition probabilities and the execution times of each state. We use the logged information to retrace each logged execution path of a service in its specification, that is, the number of visits to each state and transition is counted. The number of visits to a state must be greater or equal than the number of visits to its outgoing transitions, since the state must be visited before one of these transitions is taken. Furthermore, if the state is a final state, it might happen that the execution terminates in the state and no further transitions are taken. So, the transition probability from a state s_i to a state s_j is: $P(_{i, j}) = T_{i, j}/S_i$

Figure 10 shows the enhanced service effect specification of the service HandleRequest of StaticFileProvider.

VI. FUTURE WORK & FURTHER DEVELOPMENT

The modeling of different return values and errors of a service enables the modeling of algorithms like fault tolerance and load balancing in distributed systems. Both are common methods to increase the reliability and performance of a software system. However, it is important to notice that the reliabilities of copies of the same software component are not independent. This must be considered when determining the overall system reliability.

The interdependency of components is limited to service calls which are made from one component to the other, that is, if one component fails, this has no influence on the other components in the system as long as these not require any services of this component. We also considered the dependency of components that are deployed on the same resource, that is, if two components are deployed on the same resource, both depend on it and their reliability has a certain dependency. However, there is still dependence between successive software executions. This must be considered as well.

Since a considerable uncertainty exists in the estimates of the operational profile and components reliabilities then a significant uncertainty exists in calculated software reliability. Moreover, the way of estimating software reliability by plugging point estimates of unknown parameters into the model may not be appropriate since it discards any variance due to uncertainty of the parameters. Therefore, a methodology for uncertainty analysis of software reliability suitable for large complex component based applications and applicable throughout the software life cycle is needed.

The fact the components that are rarely executed usually handle critical functionalities such as exception handling or recovery. That is, components with small execution rates might be the most sensitive components to the changes in the operational profile. Therefore, a method for studying the sensitivity of software usage to changes in the operational profile is needed.

One important fact should not be overlooked - the quality of the reliability predictions depends not only on the methods used, but also on the quality of the failure data. One reason for low data quality is due to the fact that in most cases problem and change tracking repositories used today were not designed with failure analysis. Another major reason of low data quality is the lack of consistency and discipline in the process of recording the data. Therefore, a method for improving the process of collecting and recording the failure data, making real failure data from variety of sources publicly available is needed.

TABLE II
THE LOGGED INFORMATION OF THE WEB SERVER SYSTEM

Call	Caller	Class	Interface.Method	Time
1	0	XMLConfigReader	IConfigReader.ReadConfiguration	3004320
2	0	WebserverConfig	IWebserverConfig.getConfigFilesPath	0
3	0	WebserverConfig	IWebserverConfig.getConfigFilesPath	0
4	0	WebserverMonitor	IWebserverMonitor.InitializeWriteAccess	500720
5	4	WebserverConfig	IWebserverConfig.getDebugFile	400576
6	4	WebserverConfig	IWebserverConfig.getLogFile	0
7	0	DefaultDispatcher	IDispatcher.Start	801152
…	…	…	…	…

Conducting a large software application requires to analyze the adequacy, applicability, and the software reliability model. For that purpose, an innovative approach to efficiently extract and more accurately analyze a large amount of empirical data is needed. Applying the theoretical results on a large scale field study allows us to test how and when the model works, to understand its limitations, and outline the issues that need attention in the future research studies.

VII. CONCLUSION

The approach of Reussner [1] provides a way to predict the reliability on the basic of parametric contracts and Markov models. There, the entry R(i, j) of the potential matrix of a Markov model is the expected number of visits to the state s_j from the state s_i. The fact that the potential matrix contains the expected numbers brought us an idea to use these values to calculate the execution time of a service.

A clear understanding of the reliability from users' point of view has been introduced by Cheung [6]. He states that the fact that users rarely request a faulty service or users often request a faulty service must be considered in calculating the

reliability. Therefore, we extended his view of reliability: if a faulty resource is not used, it will not influence the reliability from users' point of view.

Using the Markov model as basic to give prediction of the reliability accepts an assumption: the service executed next depends only on the current state. So, the importance of the path through the control flow of the program is neglected, that is, the input data as well as the user behavior are neglected. Motivated by this assumption is quite popular in Markov-based reliability prediction, we have developed a way to build operational profile which accounts for the influence of the input data and the user behavior.

REFERENCES

[1] R. H. Reussner, I. H. Poernomo, and H. W. Schmidt, "Reasoning on software architectures with contractually specified components," in Component-Based Software Quality: Methods and Techniques, ser. LNCS, A. Cechich, M. Piattini, and A. Vallecillo, Eddition. Springer-Verlag, Berlin, Germany, 2003, no. 2693, pp. 287-325.

[2] S. Becker, "The palladio component model," University of Oldenburg, Tech. Rep., 2004, http://se.informatik.uni-oldenburg.de/pubdbfiles/pdf/TechReport%20Component%20Model.pdf

[3] B. Meyer, "Applying "Design by Contract"," IEEE Computer, vol. 25, no. 10, pp. 40-51, Oct. 1992.

[4] OMG, "UML 2 superstructure, final adopted specification," http://www.omg.org/docs/ptc/03-08-02.pdf, 27.12.2004.

[5] J. D. Musa, A. Iannino, and K. Okumoto, Software Reliability - Measurement, prediction, application. New York: McGraw-Hill, 1987.

[6] R. C. Cheung, "A user-oriented software reliability model," IEEE Transactions on Software Engineering, vol. 6, no. 2, pp. 118-125, Mar. 1980, special collection from COMPSAC '78.

[7] S. M. Ross, Introduction to Probability Models, 4th ed. Academic Press, 1989.

[8] A. Burns and A. Wellings, Real-Time Systems and Programming Languages, 2nd Edition. Addison-Wesley, 1996.

[9] C. Cinlar, Introduction to Stochastic Processes. Englewood Cliffs, NJ: Prentice-Hall, 1975.

[10] S. Mullender, Distributed Systems, 2nd Edition. Longman Publishing Group, 1993

[11] S. Ozekici and R. Soyer, "Reliability of software with an operational profile," European Journal of Operational Research, vol. 149 (2003), pp. 459-474, May 2002. [Online]. Available: www.sciencedirect.com

[12] D. Hamlet, D. Mason, and D. Woit, "Theory of software reliability based on components," in Proceedings of the 23rd International Conference on Software Engeneering (ICSE-01). Los Alamitos, California: IEEE Computer Society, May12-19 2001, pp. 361-370.

[13] K. Goseva-Popstojanova and S. Kamavaram, Assessing Uncertainty in Reliability of Component-Based Software Systems, 14th IEEE International Symposium on Software Reliability Engineering (ISSRE 2003), Denver, CO, Nov. 2003, pp. 307-320.

[14] S. Kamavaram and K. Goseva-Popstojanova , Sensitivity of Software Usage to Changes in the Operational Profile, 28th NASA/IEEE Software Engineering Workshop, Greenbelt, MD, Dec. 2003, pp. 157-164.

[15] K. Goseva-Popstojanova, Quality of Failure Data - The Good, the Bad, and the Ugly , Reliability Analysis of System Failure Data Workshop, Cambridge, UK, March 2007.

[16] K. Goseva-Popstojanova and M. Hamill Architecture-Based Software Reliability: Why only a Few Parameters Matter? , 31st Annual IEEE International Computer Software and Applications Conference (COMPSAC 2007), Beijing, July 2007.

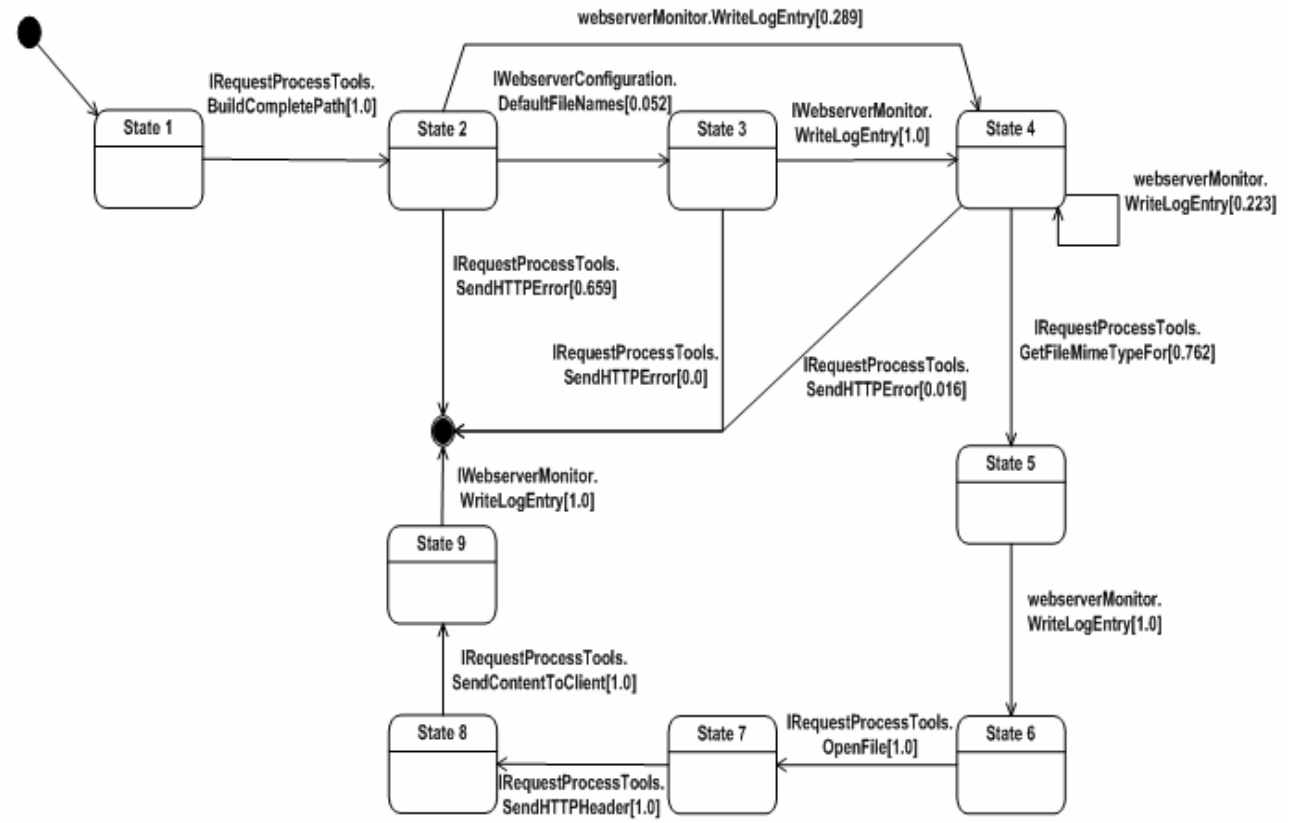

Fig. 10: The enhanced service effect specification of the service HandleRequest of StaticFileProvider.