ABSTRACT
The reliability analysis of web services is often focused on the web service components, ignoring the impact of the middleware located beneath the web services. A service-based software system is a multilayered system that includes the web service (WS), shared resources, and the hosting application server (AS). It is conjectured that the reliability prediction of the web services is improved if the reliability model accounts for such underlying layers. The initial experiment illustrates that the AS and shared resources can impact the overall reliability of web services greatly. This observation is demonstrated by simulating the interaction between a web service and the AS.

Categories and Subject Descriptors
D.2.4 [Software Engineering]: Software/Program Verification-Reliability

General Terms
Design, Reliability

Keywords

1. INTRODUCTION
Architecture-based reliability analysis of software systems has gained increased attention in recent years as it can provide better understanding of the relationship between reliability measure and underlying components, guiding the improvement in architecture design. Prevalent architecture-based analysis techniques are broadly classified into two categories, namely, path-based and state-based. A thorough discussion of architecture-based reliability approaches and their limitations is presented in [1]. Often architecture-based reliability approaches are based on individual reliability of components because it is easier to comprehend the failures and their causes. But in a distributed system, such as a web service (WS) environment, determining sources of failures is challenging. This is due to the hierarchical complexity of layers involved in satisfying web services. For instance, one of the major causes of failure in web services is resource exhaustion, which can happen by receiving a large number of simultaneous requests to the shared infrastructure.

This study follows the reliability definition of web services by Hwang et al. [2], which defines the WS reliability as the probability the web service successfully responds within a reasonable period of time. This means that any time-out or resource exhaustion situation originating from underlying shared infrastructure is considered a failure as well. These failures, which frequently happen in web services, are missing in many of architecture-based WS reliability approaches [3] [4]. The main causes of failure in web applications have been analyzed in a report from Carnegie Mellon University [5]. This report has used information from some major companies such as Google, IBM, Intel, and Microsoft. Based on this report, a significant cause of failure is not logical or computational error, but rather system overload, configuration errors, and resource exhaustion. Many of the failures appear because of insufficient shared resources in run-time environment, and human/operator errors. These failures can profoundly degrade the reliability of web services. This study includes the shared resources as a viable source for failures. By including the shared resources, we expect more accurate reliability evaluation to be perceived.

This study translates the WS environment into a multilayered model, where each called layer provides services for the calling layer. Therefore, instead of dividing a service-based system to various components, this system is divided into layers consisting of web server, application server, web service, and database. The layers can be considered as “gray boxes” or “gray layers”, where a layer constitutes an autonomous logical behavior, and yet some details of code behavior and configuration settings are taken into account. The layered approach has some benefits in comparison to previous component-based approaches. Defining layers instead of components can lead to less complicated modeling, as layers are mostly autonomous packages. Additionally, as each layer often has a separate failure repository and CVS, finding the originating layer of each failure is more straightforward than detecting the component responsible for the failure.

To present this approach, a state-based modeling, namely Stochastic Petri Net (SPN) [6], will be used. SPN offers a graphical notation and a high-level specification mechanism to
represent the architecture of distributed and asynchronous systems. Section 2 proposes the approach, section 3 presents an experimental example, and section 4 concludes the study.

2. THE MODELING APPROACH
As this study concentrates on web service environment and its related middleware, the failure probability of the operating system and hardware on which the AS runs are not considered. The essence of the approach is combining the architecture-based reliability modeling along with some of the AS performance parameters discussed in [7] and [8]. To convert the layered approach to a Petri net model that can be evaluated, six phases are needed. These phases and their relationships are shown in Figure 1. This section explains the purpose of each phase.

2.1 Dynamic Analysis
The purpose of the dynamic analysis phase is to augment the layers with some extra code for gathering information about the runtime behavior of the system.

2.1.1 Instrument AS with a Profiler
Since the architecture-based software reliability needs insights into the dynamic behavior of the system, the first step is to instrument the software with a profiler [9]. A profiler like JRAT [10] can provide the frequency of control transfer between classes and the call graph diagram during a corresponding execution. Also it can provide the time spent in each class. Instrumentation not only gathers all information from AS run-time behavior, but also collects that information from the deployed web service as well.

2.1.2 Execute WS Test Cases
Various service requests are created in order to collect three different sets of information. The first set of information represents the frequency or the patterns of interaction among the layers; the second set contains information about the execution times of each layer, while the third is used to collect failure data about the each layer.

2.2 Static Analysis
The purpose of this phase is to obtain information about the static structure of the system.

2.2.1 Extract Main Layers
In order to find the architecture, it is necessary to investigate the logical and/or physical layers of the system. A suitable and manageable level of extraction needs to be determined. Major layers in service provider will be determined using some architectural tool such as Structure 101[11], which produces several views helpful for understanding the architecture.

2.2.2 Extract Shared Resources
Using performance modeling literature review, the main resources that cause delay in the system will be identified as well. These components also influence the reliability of web services. As an example, in JBoss AS, there are at least three main components in the form of shared resources that have direct impact on performance and reliability: HTTP/AJP thread pool in the web container of web server, EJB instance pool in EJB container of AS, and database connection pool of the data access layer [8]. These components can be configured by deployers using XML files before each AS startup.

2.3 Form Architecture Model
Considering the major layers and resources extracted from the previous step, an architecture model will be built. Although the main layers and their relationships are developed during the static analysis, running the test cases during the dynamic analysis can also assist in the discovery of layer relationships.

2.4 Parameter Estimation
The purpose of this phase is to attain the information needed to build the stochastic behavior among and within the layers.

2.4.1 Estimate Transition Probabilities between Layers
Transition probabilities among layers are estimated from the information collected during the dynamic analysis phase. The transition probability from a layer A to a layer B is calculated by
enumerating the number of times A transfers control to B, divided by total number of times that A transfers control to other layers.

2.4.2 Estimate Time Spent in Each Layer
By instrumentation of AS during the dynamic analysis phase, information on time spent in each layer can be provided. JRAT assists in preparing a profile for each test case, which consists of the time spent in the methods, classes, packages, and consequently the layers called.

2.4.3 Gather Failure Data for Each Layer
To find the reliability of the system, the failure probability of each layer needs to be determined. This is done by analyzing the failure data collected from the test cases during the dynamic analysis phase. It is also possible to extract some failure data from different resources such as source code change logs [9].

2.5 Form Petri Net Model
At this phase, Mobius [12] is used to form the Petri net model. Mobius is a powerful tool that allows for hierarchical modeling of the layers by developing the submodels for each layer and then arranging them according to the needs. The stochastic relationship among the layers is formed by the transition rates obtained in the dynamic analysis phase. Each layer is represented by an exponential timed transition.

2.6 Gather and Analyze Results
Using the model formed in the previous phase, the final phase of this approach is to run the Petri net model, collect, and analyze the results. Some parameters of interest are the overall reliability of the web service system, and determining which layer or component contributes the most to the overall reliability.

3. AN EXPERIMENTAL EXAMPLE
As illustrated in Figure 2, an overall Petri net model with appropriate submodels can be constructed. As an example, Figure 3 provides a model for the service requests that arrive at the middleware. Duke’s Bank WS [13] running on JBoss AS is a web service that is used in this experiment. This web service uses a stateless EJB to manage its data and its program logic. In JBoss AS, Apache Tomcat is the default web server. The major shared resources in this experiment are the HTTP threads and EJB instances, which can be configured by the administrator using server.xml and standardjboss.xml files. In Figure 3, the timed and immediate transitions are shown by thick and thin bars, respectively. Each flat dot at the output end of a transition indicates the probability of taking a different path to the rest of the model once the transition fires. Places display pre-state and post-state of each transition. The thread and EJB instances pools are represented by places whose multiplicities are shown as tokens in Pthreadpool and Pejbinstancepool, respectively. An input gate, shown as a triangle that leads to a transition, represents the enabling or disabling conditions of the transition. For instance, if Pthreadpool is not depleted, transition T03 will not fire. Otherwise, if all tokens in Pthreadpool is consumed and there are new requests, T03 will fire and a token is created into Down, causing the system to fail. Therefore, as long as there is a token in Pthreadpool, the model is active or if the model fails, the failure is not due to the lack of threads in the thread pool.

Tarrival gives the rate of arrivals per unit of time. Transition Twebserver represents the rate at which the web server executes. To ensure the results obtained are accurate, the timed activities must be disabled once the model shows a failure, i.e., when a token is placed in Down. Input gate Ig2 is used for this purpose, i.e., it disables Tarrival, Twebserver, Tapplicationserver, and Tws as soon as a failure occurs, which prevents the model from generating further reward values.

Tws and Tapplicationserver represent the rate at which web service and AS run, respectively. ToDB models the database portion of Figure 2. T07 presents the success or failure of the database. The shared resources for the database are not shown in the Petri net model for the sake of simplicity, but it can be easily added similar to the shared resources for the web service and the application server. Table 1 lists the hypothetical values for the parameters set needed to run the Petri net model. In the model once a transition with probability cases is fired, only one of the cases will be followed. For example, once Twebserver fires, the cases 1, 2, and 3 of Twebserver show the probability that the web server fails, successfully transfers execution to the rest of the AS, and successfully finishes by returning the shared resource, respectively. Running the first experiment for five units of time, the probability of having a token placed in Down is 0.125, which implies the reliability is 0.875.

In the second experiment, Figure 4.a shows the reliability changes by varying the maximum size of the HTTP thread and EJB instance pools. The rest of the parameters stay fixed as listed in Table 1. Figure 4.a demonstrates that if the number of HTTP threads and EJB instances are less than the arrival rate, reliability is reduced drastically. For instance, since the total number of arrivals in 5 units of time is 25 and the size of HTTP thread pool is 2, the system reliability becomes extremely low. Note that for the experiments in this study, the system model is set up such that a token is deposited in Down as soon as the first input request cannot be serviced, regardless of whether the future requests can be serviced. Furthermore, reliability of the system improves as the maximum value of EJB instance pool and HTTP thread pool increase until the reliability reaches a plateau. For example, when the size of the thread pool is 10, 14, or 18, not much variation in reliability is observed. It should be intuitive that the continuous increase in the thread pool, due to the increase in the arrival rate, will ultimately overload the system, leading to degraded performance and reliability. This aspect of reliability is not shown in the model presented here. Figure 4.b presents how reliability changes through the time. In this experiment different sizes for HTTP thread pool are examined. It is observed that by increasing
the size, system reliability improves. But, after reaching a threshold, any increase on the pool will not have much effect on the reliability.

In the next experiment, Figure 5 shows the sensitivity of the web service reliability due to changes of failure probabilities in the AS and the web service. The rest of the parameter values are fixed as shown in Table 1. As expected, higher failure probabilities for the AS and web service will negatively affect the overall system reliability. The experiments have shown that certain parameters can generate significant effects on reliability. Although these experiments can be improved by providing realistic parameter values, they have demonstrated valuable insight in the overall reliability aspect of a multilayer web service system.

Table 1. Parameters set for Petri net model

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Parameter value</th>
<th>Parameter name</th>
<th>Parameter value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarrival rate</td>
<td>5</td>
<td>Tws rate</td>
<td>20</td>
</tr>
<tr>
<td>Twebserver prob.</td>
<td>0.0001</td>
<td>T07 prob. case1</td>
<td>0.0001</td>
</tr>
<tr>
<td>case1</td>
<td></td>
<td>T07 prob. case2</td>
<td>0.9999</td>
</tr>
<tr>
<td>Twebserver prob.</td>
<td>0.4999</td>
<td>Pthreadpool</td>
<td>10</td>
</tr>
<tr>
<td>case2</td>
<td></td>
<td>Twebserver rate</td>
<td>20</td>
</tr>
<tr>
<td>Twebserver</td>
<td>0.3999</td>
<td>Pthreadpool1</td>
<td>10</td>
</tr>
<tr>
<td>Tapplicationserv</td>
<td>0.4</td>
<td>Pejbinstancepool</td>
<td>0.9999</td>
</tr>
<tr>
<td>er.prob.case1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tapplicationserv</td>
<td>0.2</td>
<td>Ig1</td>
<td>(Mark(Pthreadpool)==0) &amp;&amp; (Mark(Start)&gt;0)</td>
</tr>
<tr>
<td>er.prob.case2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tapplicationserv</td>
<td>0.9999</td>
<td>Ig2</td>
<td>Mark(Down)==0</td>
</tr>
<tr>
<td>er.prob.case3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tapplicationserv</td>
<td>0.0001</td>
<td>Ig3</td>
<td>(Mark(Pejbinstancepool) l&gt;=0) &amp;&amp; (Mark(TosAS)==0)</td>
</tr>
<tr>
<td>er.prob.rate</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tws prob.case1</td>
<td>2005.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.9999</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0.001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. CONCLUSION AND CONTRIBUTION

The proposed approach has laid out the mechanism in the reliability evaluation of web services using SPNs. The main contribution of this research study is the reliability modeling of the entire WS software system that embodies the WS and the underlying layers. Therefore, a more accurate reliability prediction of WS software is achievable. These experiments have shown that the reliability of web services is highly sensitive to the configuration parameters that control the shared resources.

5. ACKNOWLEDGMENTS

This research is funded in part by Department of Defense (DoD)/Air Force Office of Scientific Research (AFOSR), NSF Award Number FA9550-07-1-0499, under the title “High Assurance Software”. In addition, the authors thank Prof. Roshanak Roshandel from Seattle University for her helpful comments on an earlier draft of the paper.

6. REFERENCES