A Hierarchical Reliability Model of Service-Based Software System

Lijun Wang, Xiaoying Bai, Lizhu Zhou
Department of Computer Science and Technology
Tsinghua University
Beijing, China
lj-wang04@mails.tsinghua.edu.cn, {baixy, dcszlz}@tsinghua.edu.cn

Yinong Chen
Department of Computer Science and Engineering
Arizona State University
Tempe, USA
yinong@asu.edu

Abstract—Service-Oriented Architecture (SOA) introduces a new paradigm where data, service, and the service composition logic are decoupled in an open environment. Software systems are built and evolved online by dynamic discovering and binding to the open services accessible through standard protocols. It requires new reliability modeling techniques to facilitate the analysis of dynamic collaborations and to be adaptable to the changes of service composition and system configuration at runtime.

The paper proposes a hierarchical reliability model, in which the system reliability is calculated according to the layered superimposition of the reliabilities of data, services, fault-tolerant mechanism and service composition logic. At the basic-service layer, the service reliability is considered in the context of data reliability and service pools with backup alternatives. At the composition layer, a DTMC (Discrete Time Markov Chain) model is created for analyzing system reliability based on the reliability of the constituent services and their execution rate which are decided by the composite control structure and the operation scenarios. The DTMC state transition diagram can be automatically generated by analyzing and transforming the layered superimposition of the application following the transformation rules. The hierarchical modeling framework enables change adaptation at various levels. In addition, the models can be continuously learned and dynamically adjusted by profiling on the runtime monitoring log files.

Keywords—software reliability; service-oriented architecture; reliability model; service pool; markov chain

I. INTRODUCTION

Service-oriented architecture (SOA) is a promising computing paradigm for software development in a heterogeneous open environment [18][19]. Systems are created by reusing services available and building new composite services. Services interoperate through standard protocols and in a loosely coupled fashion. The architecture enables dynamic interoperation and collaboration among service providers, service brokers, and service consumers, and thus can be flexibly adapted to continuous changing business requirements by dynamic service re-composition and system re-configuration.

SOA presents unique requirements to reliability analysis. Traditional software is defined by code, document and data. Most of the artifacts are owned, controlled, and maintained by the application builder. In contrast, services in SOA can be distributed, located and owned by independent providers. For example, an application “Real estate map” can be built by mashing up the “Google Map” services with “property price listing” services [15]. To cope with this new paradigm, new mechanisms are necessary to ensure the quality of the services outside the scope that the application builder can control. Unfortunately, the dynamic and collaborative nature of SOA system imposes new challenges to traditional reliability research and practice.

Particularly, our research is motivated by the following challenges:

• How to measure service reliability? In a service-based system, services, service integrations and service input/output data are decoupled from each other. They may be built, provided and maintained by different providers, and dynamic mashed up by different application builders to create systems in different context of business processes. The reliability of the integrated service-based system depends on the reliability of the composition, the constituent services, and the data imported from external resources. However, different providers may present different levels of service qualities. The reliability of the service should be measured w.r.t. all the constituent elements.

• How to model the reliability of service fault-tolerance mechanism in an open environment? SOA has the potential to provide software and system fault-tolerant capabilities in an open environment with multiple online alternative services available through the standard interface and interoperation protocols. Ideally, it can greatly enhance the system sustainability, which enables dynamic binding to a new service when the currently invoked service failed and the Internet provides a large number of backups. However, it takes extra cost to find the replacement and the rebinding to the new service may fail due to unstable Internet environment.

• How to evaluate the reliability of dynamic architecture? Traditional architecture-based reliability models are always created and evaluated offline. It assumes that the software architecture is designed offline and its constituents are permanent
during the software lifecycle. However, the service-based system (SBS) is characterized by dynamic service publication, registration, discovery, binding, and composition. It allows the application architecture to be changed and evolved at runtime by re-binding to a new service, and re-configuring the service workflow. To reflect the changes, the reliability model in the level of SBS or service composition needs to be adapted to the architecture changes to allow for online adaptive reliability evaluation.

To address the above unique problems of service-based system, new methods and modeling techniques are necessary. Up to now, only a few attempts have been made in this new area [10][11][21]. This paper proposed a hierarchical approach for reliability modeling and analysis as shown in Fig. 1.

![Figure 1. The modeling framework](Image)

A service-based system in general is modeled as a workflow made up of activities (processes) and control constructs. The activities can be bound to various service implementations that satisfy the required interface and functionality. The layered reliability modeling framework is constructed to decompose the system reliability analyses into two perspectives: 1) How reliable is each activity? 2) How does the reliability of each activity contribute to the overall system reliability?

From the first perspective, each activity is associated with an active service and a pool of backup spare services. In this research, the DTMC (Discrete Time Markov Chain) model is used to analyze the pool’s reliability. For each service in the pool, various factors are considered including the reliabilities of service functions, network connection, service binding, and input/output data.

From the second perspective, the focus is on the execution rate of each activity. The control flow is analyzed using the single-entry-single-exit DTMC state transition model. The transition probability matrix is defined for each type of control constructs. The runtime monitoring mechanism is introduced for operation profile analysis so that it can dynamically adapt the model to the changes in the workflow structure and the execution profile. A prototype system is implemented and examples are exercised to illustrate the proposed approach.

The rest of the paper is organized as follows. Section II discusses related work. Section III introduces the reliability model for basic services. Section IV studies the reliability analysis methods and techniques for composite services. Section V presents the prototype and the experiments. Section VI concludes the paper.

II. RELATED WORK

Software reliability has been defined as the probability that no failure occurs in a specified environment during a specified (continued) exposure period. Many models have been proposed for software reliability evaluation. Ramamoorthy and Bastani classified software reliability models according to the development phases of software life-cycle [16]. Goel [6] categorized software reliability models according to the nature of failure process.

In recent years, the architecture-based reliability model proposed a white-box approach for software reliability analysis which has been widely used in object-oriented and component-based software [7][8][9][17][22]. The models evaluate software reliability based on the component reliability and the control and data flow among the components. Various models have been proposed such as state-based, path-based, and additive models. In general, the architecture model assumes that 1) the reliabilities of the components are known; 2) the application architecture is relative stable; and 3) the connections among the components are reliable. These assumptions are fine for product-oriented software development where the components are developed internally and packed into products. But for the service-oriented software, these assumptions may be not applicable as discussed in Section I.

As pioneer research attempts, Grassi et al. [10][11] and Tsai et al. [21] exercised the path-based architecture model to service-based software reliability analysis. Grassi et al. [10][11] extended the components to more generic services which require the collaboration from other services based on three factors: the failure probability of a service, the AND/OR completion model of the services, and the dependency model among the services. Tsai et al. [21] considered the dynamic nature of service architecture where the structure and the inter-connections between the services can be changed. They incorporated the probability of each path into the reliability modeling and adjusted the reliability of each service with a factor of its execution probability. Tsai et al. [20] further identified the three factors for reliability analysis of SOA applications including: data reliability, services reliability, and workflow reliability.

Developed on the existing research achievements, the paper contributes in the following aspects: 1) it defines a unified reliability modeling framework that hierarchically superimpose the reliability of data, service, service pool and workflow; 2) it tries the state-based (DTMC) architecture model for the service pool reliability analysis and composite service control flow analysis; 3) it investigates both the static
and dynamic approaches for reliability modeling. By static approach, the service composition specification can be automatically transformed to the DTMC state transition diagram. By dynamic approach, it online monitors the changes in the system composition, configuration and operational profiles to build or adjust continuously the reliability model.

III. Basic Service Reliability Model

Software reliability (R) can be defined as the probability that software runs without failure during a predefined period of time. Software fails when the outside inputs trigger the defects in a specific operational environment. The stochastic process of fault triggering can be characterized by a Poisson process and the reliability of the software can be estimated using the exponential distribution formula as defined by Musa’s reliability model [14].

\[ R = e^{-\lambda t} \]  

where, \( \lambda \) is the failure rate, which is a probability density function of operational time, namely \( \lambda(t) \). For simplicity, \( \lambda \) is commonly represented as a constant (\( \lambda \in [0, 1] \)) using the expectation of \( \lambda(t) \). \( t \) in the formula (1) refers to the software execution time, which is set to 1 when \( R \) is defined as the probability of successful invocation in a unit of time or a timed step. Hence, the formula (1) becomes:

\[ R = e^{-\lambda} \]  

In practice, it also can be measured approximately as:

\[ R = 1 - \lambda \]  

Traditional software reliability models only consider the reliability in a relative closed computation environment. However, services in SOA are bound, invoked, and interoperate in the open environment where it may suffer from the instable and unreliable network connections and platform services like middleware failures. The service reliability should reflect not only how it behaves, but also how it is perceived by the remote clients. This research introduces the concepts of binding reliability and connecting reliability for modeling the reliability in this aspect. Hence, the service (such as a web service) reliability is defined as a function of three factors: the functional reliability, the binding reliability, and the quality of the data also plays an important role in the overall system reliability. As shown in Fig. 2, the service may present different reliability on different data. Data reliability can include data correctness, accuracy, and consistency, especially for long-life data that are transmitted across multiple services and systems. Data are even harder than services to monitor, evaluate, and quality control. Techniques like data provenance have been developed for SOA systems to trace data from creation to routing, processing, storage, and to their final destination.

A. Data Reliability

Different from traditional software development, the data sent to a service may be maintained and provided by an independent data provider. Hence, the data may be unreliable and the quality of the data also plays an important role in the overall system reliability. As shown in Fig. 2, the service may present different reliability on different data. Data reliability can include data correctness, accuracy, and consistency, especially for long-life data that are transmitted across multiple services and systems. Data are even harder than services to monitor, evaluate, and quality control. Techniques like data provenance have been developed for SOA systems to trace data from creation to routing, processing, storage, and to their final destination [20].

Taking the data reliability into consideration, the original service functional reliability \( R_f \) is adjusted to the service reliability in the data context \( R_{f-context} \). Here, we introduce the concept of data reliability \( R_d \). For an input data \( in \) to a service, \( R_d(in) \) means the probability that the data is correctly transferred from the data source to the service interface. Apparently, \( R_d \) and \( R_f \) are two independent factors that affect the success of service execution. Assume that \( \{ in_i \} \) is the set of input data of the invoked service, the adjusted \( R_{f-context} \) in (formula 4) is defined as follows:

\[ R_{f-context} = \prod_i R_f(in_i) \cdot R_f \]  

- \( R_f \) is the functional reliability with failure rate \( \lambda_f \). It is an inherent QoS (Quality of Service) attribute on the service side. In other words, \( R_f \) stands for the traditional software reliability. It can be either provided by the service providers together with service registration, or evaluated independently by the service broker and consumers;
- \( R_b \) is the binding reliability with failure rate \( \lambda_b \). It is defined as the probability of successful matching of service requirements in terms of interface, functionality, performance and other properties. In this paper, we do not consider service match failures and take \( R_b = 1 \), or \( \lambda_b = 0 \);
- \( R_c \) is the connecting reliability with failure rate \( \lambda_c \). \( R_c \) represents the probability of successful service invocation, or the service function availability.

In this paper, the service functional reliability \( R_f \) is further refined w.r.t. its input data reliability and the service pool mechanism for online fault-tolerance.
This paper analyzes the reliability of the service pool with a DTMC model. An invocation to the service associating with the pool is defined as a timed step. The state of the pool is defined by the invocation to a specific service in the pool. A failure in current invoked service will trigger a state transition of the service pool. Fig. 3 shows the state transition diagram of a service pool. It assumes that:

- A service pool consists of $N$ alternative services, namely, $PL(ws') = \{ws^1, ws^2, ..., ws^N\}$;
- A failed service will not be recovered; and
- The service pool does not change in a timed step.

**Definition 1:** The DTMC-based reliability model of a service pool is a pair $R_{pl} = (S, P)$.

- $S$: is the state space that includes $N+2$ states where $N$ is the length of the service pool $PL(ws')$, namely, $S = \{s_1, s_2, ..., s_N, s_C, s_F\}$. Among these, $s_i$ (i.e. [1, N]) is the active state stands for an invocation to $ws^i$, and $s_1$ is the sole initial state stands for the first invoked service in $PL(ws')$; $s_C$ and $s_F$ are two terminal states (or absorbing state) that stand respectively for success and failure of the whole service pool;
- $P: S \times S \rightarrow [0, 1]$ is a matrix of the transition probability, and its elements $p_{ij}$ (i.e. the probability transferring from $s_i$ to $s_j$) are as follows:

\[
\begin{align*}
  p_{i,j} = \begin{cases} 
  \lambda_{ws^i}, & i \in [1, N-1] \\
  \lambda_{s_1} - \lambda_{ws^i}, & i \in [1, N] \\
  \lambda_{s_C} - \lambda_{ws^i}, & i \in [1, N-1] \\
  \lambda_{s_F}, & i \in [1, N] \\
  0, & \text{otherwise}.
\end{cases}
\end{align*}
\]

where, $\lambda_i$ refers to the basic service failure rate $\lambda_{ws^i}$, and $\lambda_{ws^i}$ means the partial $\lambda_i$ that can be monitored or observed online.

Thus, the reliability $R_{pl}$ or failure rate $\lambda_{pl}$ of the service pool can be defined, respectively, as the probability that the service pool sojourns at the absorbing state $s_C$ or $s_F$. From analyzing this model, formula (7) is derived to solve $R_{pl}$ or $\lambda_{pl}$:

\[
R_{pl} = \sum_{i=1}^{N} (1-\lambda_i) \prod_{j=1}^{N} \lambda_{ws^j}, \text{ or}
\]

\[
\lambda_{pl} = \sum_{i=1}^{N} (\lambda_i - \lambda_{ws^i}) \prod_{j=1}^{N} \lambda_{ws^j} + \prod_{j=1}^{N} \lambda_{ws^j}.
\]

When all failures of services are detectable, namely, $\lambda_{ws^i} = \lambda_i$, $\lambda_{pl}$ from the formula (7) will be reduced to:

\[
\lambda_{pl} = \prod_{i=1}^{N} \lambda_i
\]
These results can be used to quantitatively control the expected reliability of the service pool, and guide the design of the service pool. A unique benefit of SBS is the reliability adaptation. The application builders can choose and change the capacity of the pool to achieve a balance between cost and reliability. They can also dynamically adjust the constituent services in the pool according to the runtime monitored service reliability. During a period of time, they can drop the services with decreased quality from the pool, and add the services newly published with better performance.

IV. Composite Service Reliability Analysis

As shown in Fig. 4, service composition reliability is modeled and evaluated from two perspectives:

- The static perspective: The service composition specification can be used for architecture-based reliability analysis, such as the OWL-S (Web Ontology Language for Service) [24] specification which defines the composition workflow. The specification is interpreted to identify the component services, service interactions, data flow and control flow. Reliability model is constructed to reflect the characteristics of the composition.

- The dynamic perspective: By profiling on the log file of runtime monitoring, the reliability model is constructed dynamically to reflect the real service operations and interactions.

Static reliability model can be used before service deployment for architecture evaluation and early-stage quality prediction. Dynamic reliability model is used to capture the runtime system behavior so that it can be adapted to the dynamic changes in service composition and system configuration. Static model and dynamic model can cross verify each other to detect the incompleteness and inconsistencies between the expected and actual system behavior.

Web Services standard OWL-S models the composite service as a workflow of processes which are organized by the control constructs. The control flow between services can be modeled by a Markov process that the future behavior of the system, at any given time, is conditionally independent of the past behavior. In this research, the DTMC state model is used for both of the static and dynamic reliability modeling. In the DTMC, each state represents the execution of a service and the state transition represents the invocation probabilities of next services under current state. The reliability of the system depends on both of the execution reliability of each individual service (i.e. state reliability) and its execution rate.

For static modeling, the DTMC state model is automatically generated by transforming the service process model. For dynamic modeling, the model is dynamically constructed by runtime monitoring of the service execution paths. For both of the models, transition probabilities, which are defined by next services invocation probabilities, can be obtained from the operating profiles logged by the monitoring facilities.

A. The Service Process Model

In this research, the OWL-S [24] standard specification is used as the composition architecture model of the service-based software. OWL-S is a W3C standard to provide a computer-interpretable description of the services, service access and service composition using the OWL ontologies. Building upon SOAP and WSDL, the OWL-S model describes the service capabilities to enable service invocation, composition, monitoring, etc. The composition of services is modeled as a workflow of processes including atomic, simple, and composite processes. Each composite process holds a Control Construct which is one of the following: Sequence, Split, Split-Join, Any-Order, Choice, If-Then-Else, and Iterate (Repeat-While and Repeat-Until). The constructs, which defined the control flow of the integrated system, can contain each other recursively, so the composite process can model all kinds of possible workflows.

B. Failure Behavior Analysis

In the OWL-S specified process model, the composite process is recursively defined by composite and atomic processes, which are connected by the control constructs. The control constructs define the control flow of the composite services, and the IOPE (input, output, precondition, effects) defines the data flow and dependencies between the services. A failure of the system can be traced to either one of the constituent elements including the control constructs, the atomic process, the execution conditions, and the input/output data. As shown in Fig. 1, the failures are considered at two levels: the composite level and the basic service level. Based on the OWL-S process model, the glue logic such as control and condition are analyzed at the composite-level; and the binding, connection, data and functions of an atomic process are at the service-level.

An atomic process is an abstract description of the required functionality which can be dynamically bound to external services. The input/output data of the process definition are mapped to the service interface messages and decide the context for the service execution. In this research, the atomic process is considered as a service placeholder and associated with a service pool of candidate services that satisfying the demanded functionalities. Hence, the failure rate of the atomic process \( R_{\text{imp}} \) is defined as that of the service pool as discussed in Section III. That is, \( \lambda_{\text{imp}} = \lambda_{p} \).

The failure of the composite process can be traced to the atomic process following the execution path governed by the control construct. Hence, the reliability of the composite
service is decided by the reliability of its constituent services as well as their execution rate. For example, for the Choice construct with $n$ choices, suppose an even distribution that each branch has a probability of $1/n$ to be executed. Then, the failure of a service at one branch only has $1/n$ probability to cause the failure of the construct. In other words, the failure rate of the service contributes $1/n$ to the failure rate of the overall construct.

Based on the failure behavior analysis, the constructs can be classified into two categories: serial and parallel.

In a serial construct, only one service can be invoked and executed in a time step; while in a parallel construct, more than one service can be invoked and executed. The serial constructs can be further classified into two subcategories: unconditioned and conditioned. For unconditioned constructs, all the services will be definitely invoked and the failure of any service will cause a system failure; for a conditioned construct, the services will be selectively executed and the failure of a service has a weight $w$ to cause the system failure, where $w$ is the number of service invocations. Loop is a special case of conditioned construct, which can be executed 0 or many times. That is, the execution times of services in a loop can be transformed into the probability of re-executing oneself.

The parallel constructs are classified into synchronous and asynchronous. Even though the system waits in different ways of the paralleled process, both synchronous and asynchronous require the completeness of all the branches, hence, failure in any branch will also cause the system failure. If only the reliability is taken into account, then a parallel construct can be flattened into a serial construct which is executed in the same form of Any-Order. In most cases, services lie in the different split of a parallel construct need not to interact with each other. In this paper, we only consider the scenarios that services executed in parallel are independent to other. Further research can introduce the Interactive Markov Chains (IMC) [23] can be used for interactions analysis.

Table I lists the categories of control constructs. The corresponding transformation rules are defined in next section.

### Table I. Categories of Control Construct

<table>
<thead>
<tr>
<th>Category</th>
<th>OWL-S Construct</th>
<th>Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unconditioned</td>
<td>Sequence</td>
<td>Sequence</td>
</tr>
<tr>
<td></td>
<td>Any-Order</td>
<td>Any-Order</td>
</tr>
<tr>
<td>Conditioned</td>
<td>If-Then-Else, Choice</td>
<td>Choice</td>
</tr>
<tr>
<td></td>
<td>Iterate: Repeat-White, Repeat-Until</td>
<td>Loop</td>
</tr>
<tr>
<td>Parallel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronous</td>
<td>Split-Join</td>
<td>Any-Order</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>Split</td>
<td>Any-Order</td>
</tr>
</tbody>
</table>

C. The DTMC-based Execution Profile Model

This research uses state transition model to analyze the probability of execution paths [10][11][21][24]. A single-entry-single-exit DTMC is defined for modeling the state-based behavior of the composite services. Assume that

- A composite service consists of $K$ services: $CS(K) = \{w_1, w_2, \ldots, w_K\}$;
- The services are connected through control constructs as defined in the OWL-S standard; and
- The configuration and composition of the composite service does not change in a time step.

We have the following definition of the DTMC-based execution profile model of composite service.

**Definition 2**: The single-entry-single-exit DTMC of composite services is defined as a pair $(S, P)$

- $S$: is the state space of the composite service in a run cycle, $S = \{s_0, s_1, s_2 \ldots s_K, s_T\}$. Among these, $s_0$ and $s_T$ are added to ensure a single-entry-single-exit model, $s_0$ is the sole initial state and $s_T$ is the sole terminal state (or absorbing state); $s_i, (1 < i < K)$ is defined as the state that service $w_i$ is just invoked.
- $P$: $S \times S \rightarrow [0, 1]$ is a matrix of the transition probability. $p_{ij}$ denotes transition probability from $s_i$ to $s_j$ where $p_{ij} = 0$ means $s_i$ cannot reach $s_j$; otherwise, $p_{ij}$ is the probability of invocation of next service that result in state $s_j$ from state $s_i$. Initially, it can be either estimated at the design time or obtained from a training set of usage scenarios.

Cheung [5] first proposed the DTMC model to architecture-based reliability of component-based software system. Wang et al. [22] refined the model to various architecture styles including pipeline, pipe-filer, call-and-return, and fault tolerant styles. The paper further adjusts the definition of the DTMC and the transition probability matrix $P$ for the control constructs of the composite process model. Fig. 5 shows the transformed state model for the four categories of constructs including sequence (order), any order, choice and loop. Table II lists the definition of the transition matrix $P = [p_{ij}]$.

![Figure 5. The transformation from control construct to DTMC model](image-url)
TABLE II. THE DEFINITION OF TRANSITION MATRIX

<table>
<thead>
<tr>
<th>Order</th>
<th>Transition Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_{ij} = 1, \quad 0 \leq i \leq K, \quad j = i + 1, \quad p_{K,K+1} = 1, \quad$ means $p_{K,T}$;</td>
<td></td>
</tr>
<tr>
<td>$p_{ij} = 0, \quad$ otherwise</td>
<td></td>
</tr>
<tr>
<td>Any-Order</td>
<td>$p_{ij} = 1/K, \quad 1 \leq j \leq K$</td>
</tr>
<tr>
<td>$p_{i,1} = 1/K, \quad 1 \leq i \leq K$</td>
<td></td>
</tr>
<tr>
<td>$p_{ij} = 1/K, \quad 1 \leq j \leq K, \quad i \neq j$</td>
<td></td>
</tr>
<tr>
<td>$p_{ij} = 0, \quad$ otherwise</td>
<td></td>
</tr>
<tr>
<td>Choice</td>
<td>$p_{ij} =$ the branch execution probability, $1 \leq j \leq K$,</td>
</tr>
<tr>
<td>and $\sum p_{ij} = 1$</td>
<td></td>
</tr>
<tr>
<td>$p_{i,1} = 1, \quad 1 \leq i \leq K$</td>
<td></td>
</tr>
<tr>
<td>$p_{ij} = 0, \quad$ otherwise</td>
<td></td>
</tr>
<tr>
<td>Loop</td>
<td>$p_{0,i} =$ the probability of skipping the loop</td>
</tr>
<tr>
<td>$p_{0,1} = 1 - p_{0,i}, \quad$ the probability of executing the loop</td>
<td></td>
</tr>
<tr>
<td>$p_{i,1} =$ the probability of breaking the loop</td>
<td></td>
</tr>
<tr>
<td>$p_{1,1} = 1 - p_{1,1}, \quad$ the probability of continuing the loop</td>
<td></td>
</tr>
</tbody>
</table>

We can use the steady-state analysis method of Markov chain [7][22][23] to solve the model above as follows:

1) The transition probability matrix $P$ is modified to $P'$ with $P_{T,0} = 1$. That is, the composite service is modeled as continuous running system which break the absorbing state $s_T$ and transfer it to the initial state $s_0$ with probability 1;

2) Define the steady-state distribution vector $\eta = [\eta_1, \eta_T]$ ($1 \leq i \leq K$), which can be evaluated by the equation $\eta = \eta P'$, where $\eta_T$ is the steady-state probability of sojourning at the absorbing state $s_T$;

3) Add the universal constraint condition (namely, all the probabilities of the steady-state distribution add up to 1) to get the linear equation group without correlation (for simplicity, use $s_{K+1}$ to stand for $s_T$);

$$\left\{ \begin{array}{l} \eta_j = \sum_{i=0}^{K+1} \eta_j p_{ij}, \quad j \in [0, K + 1] \\ \sum_{j=0}^{K+1} \eta_j = 1 \end{array} \right. \quad (9)$$

4) Solve the formula (9) to get $\eta$;

5) Based on $\eta$, calculate the weight $w_i$ (namely, the execution rate in a run of the composite service) of all states associated with services;

$$w_i = \frac{\eta_i}{\eta_T}, \quad i \in [1, K] \quad (10)$$

6) At the last step, the failure rate $\lambda_{CS}$ of the composite service is the weighted sum of $\lambda_i$ ($i \in [1, K]$). Note that $\lambda_0 = \lambda_T = 0$, since $s_0$ and $s_T$ are two auxiliary states without services associated. Accordingly, the reliability $R_{CS}$ of the composite service can be calculated by formula (2).

$$\lambda_{CS} = \sum_{i=1}^{K} w_i \lambda_i \quad (11)$$

V. PROTOTYPE AND EXPERIMENTS

We can use the steady-state analysis method of Markov chain [7][22][23] to solve the model above as follows:

1) The transition probability matrix $P$ is modified to $P'$ with $P_{T,0} = 1$. That is, the composite service is modeled as continuous running system which break the absorbing state $s_T$ and transfer it to the initial state $s_0$ with probability 1;

2) Define the steady-state distribution vector $\eta = [\eta_1, \eta_T]$ ($1 \leq i \leq K$), which can be evaluated by the equation $\eta = \eta P'$, where $\eta_T$ is the steady-state probability of sojourning at the absorbing state $s_T$;

3) Add the universal constraint condition (namely, all the probabilities of the steady-state distribution add up to 1) to get the linear equation group without correlation (for simplicity, use $s_{K+1}$ to stand for $s_T$);

$$\left\{ \begin{array}{l} \eta_j = \sum_{i=0}^{K+1} \eta_j p_{ij}, \quad j \in [0, K + 1] \\ \sum_{j=0}^{K+1} \eta_j = 1 \end{array} \right. \quad (9)$$

4) Solve the formula (9) to get $\eta$;

5) Based on $\eta$, calculate the weight $w_i$ (namely, the execution rate in a run of the composite service) of all states associated with services;

$$w_i = \frac{\eta_i}{\eta_T}, \quad i \in [1, K] \quad (10)$$

6) At the last step, the failure rate $\lambda_{CS}$ of the composite service is the weighted sum of $\lambda_i$ ($i \in [1, K]$). Note that $\lambda_0 = \lambda_T = 0$, since $s_0$ and $s_T$ are two auxiliary states without services associated. Accordingly, the reliability $R_{CS}$ of the composite service can be calculated by formula (2).

$$\lambda_{CS} = \sum_{i=1}^{K} w_i \lambda_i \quad (11)$$

V. PROTOTYPE AND EXPERIMENTS

We can use the steady-state analysis method of Markov chain [7][22][23] to solve the model above as follows:

1) The transition probability matrix $P$ is modified to $P'$ with $P_{T,0} = 1$. That is, the composite service is modeled as continuous running system which break the absorbing state $s_T$ and transfer it to the initial state $s_0$ with probability 1;

2) Define the steady-state distribution vector $\eta = [\eta_1, \eta_T]$ ($1 \leq i \leq K$), which can be evaluated by the equation $\eta = \eta P'$, where $\eta_T$ is the steady-state probability of sojourning at the absorbing state $s_T$;

3) Add the universal constraint condition (namely, all the probabilities of the steady-state distribution add up to 1) to get the linear equation group without correlation (for simplicity, use $s_{K+1}$ to stand for $s_T$);

$$\left\{ \begin{array}{l} \eta_j = \sum_{i=0}^{K+1} \eta_j p_{ij}, \quad j \in [0, K + 1] \\ \sum_{j=0}^{K+1} \eta_j = 1 \end{array} \right. \quad (9)$$

4) Solve the formula (9) to get $\eta$;

5) Based on $\eta$, calculate the weight $w_i$ (namely, the execution rate in a run of the composite service) of all states associated with services;

$$w_i = \frac{\eta_i}{\eta_T}, \quad i \in [1, K] \quad (10)$$

6) At the last step, the failure rate $\lambda_{CS}$ of the composite service is the weighted sum of $\lambda_i$ ($i \in [1, K]$). Note that $\lambda_0 = \lambda_T = 0$, since $s_0$ and $s_T$ are two auxiliary states without services associated. Accordingly, the reliability $R_{CS}$ of the composite service can be calculated by formula (2).

$$\lambda_{CS} = \sum_{i=1}^{K} w_i \lambda_i \quad (11)$$

A prototype of the system is constructed to facilitate runtime monitoring, automatic model transformation, dynamic model adjustment, and reliability calculation. Fig. 6 shows the overall architecture. It is implemented on the Eclipse platform with open source project support such as OWL-S modeler Protégé, OWL-S API and execution engines. Monitors are instrumented to the engines including the SOAP engine from the service provider’s side and the OWL-S engine from the service composition side [2][3].

Figure 6. Prototype system architecture

Figure 7. The example composite service
Experiments are conducted to illustrate the proposed approach. Fig. 7 shows an example OWL-S structure of a composite service, which covers four types of control constructs (Sequence, Split-Join, If-Then-Else and Repeat-Until) over five services ($\{ws_{1}, ws_{2}, ws_{3}, ws_{4}, ws_{5}\}$ or their relative service pools $PL(ws_i)$) bound to the five atomic processes ($\{proc_{1}, proc_{2}, proc_{3}, proc_{4}, proc_{5}\}$) in the OWL-S specification. Each service takes a set of input/output data. We simulate the functional services and the different types of system failures such as data, function, and binding failures.

![Figure 8. The transformed DTMC-model](image)

![Figure 9. The simulated transition probability](image)

Fig. 8 shows the DTMC model that is automatically generated based on the OWL-S workflow analysis. Fig. 9 shows the transition probability between the states that are calculated based on the monitored execution profile with a set of simulated inputs, where $p_{01}=p_{12}=p_{23}=1$, $p_{02}=p_{21}=p_{13}=0$, $p_{33}=0$, $p_{34}=0.372$, $p_{35}=0.628$, $p_{43}=0.964$, $p_{45}=0.036$.

By means of the steady-state analysis of the DTMC model, all the execution ratios (or weights) of services in the workflow can be determined by formula (9) and (10), as shown in Table III. Observation shows that state "s3", which denotes the execution of $proc_{3}$, accounts for the highest weight in the composite service.

<table>
<thead>
<tr>
<th>ID</th>
<th>Atomic Published</th>
<th>Offline</th>
<th>Online</th>
<th>Composite Published</th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc1</td>
<td>0.98</td>
<td>0.958</td>
<td>0.952</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proc2</td>
<td>0.95</td>
<td>0.899</td>
<td>0.898</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proc3</td>
<td>0.92</td>
<td>0.841</td>
<td>0.781</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proc4</td>
<td>0.89</td>
<td>0.786</td>
<td>0.640</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Proc5</td>
<td>0.86</td>
<td>0.732</td>
<td>0.695</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Experiment 1**: Service reliability analysis without pool mechanism.

In this experiment, we do not take the pool mechanism into consideration and only examine the reliability of atomic and composite services. Table IV shows the published, offline, and online reliability of each service and that of the whole composite service. For atomic services, 1) the published reliability is what the service provider claims and submits together with the service registration; 2) the offline reliability is tested for each service invoked individually; and 3) the online reliability is tested for each service invoked in the composition workflow. For composite service, the reliability is calculated based on the transformed DTMC model, the simulated transition probability, and the reliability of each constituent service for offline / online analysis. Each result is the average of 10,000 simulated executions. The results show that: 1) the services’ reliability decrease when they are affected by online factors such as connectivity reliability and data reliability; 2) the reliability of the composite service could be very low.

**Experiment 2**: Service reliability analysis with pool mechanism.

In this experiment, service pools are established for each atomic service in the composite service. The size of the pool, that is the number of alternative services, is increased gradually to analyze the impact of pool's size on the service reliability. Fig. 10 shows the average results of 10,000 simulated executions. In Fig. 10-(a), the size of each pool changes individually. In Fig. 10-(b), the size of all pools changes together. The results show that: 1) the increase of the pool size can increase the service reliability; 2) reliability goes stable with a size around 3. As shown in Fig. 10-(b), incorporation of pool mechanism can greatly enhance the
system reliability. For example, a pool with size 3 alternatives can increase the reliability by 107% compared with a pool with only 1 service available.

**Experiment 3: Sensitivity analysis**

In this experiment, we simulate the services with the same reliability 0.8 and change the reliability of each service individually to see how they contribute to the composite reliability in the specific workflow. As steady analysis shows (see Table III), the execution of proc3 weights the highest among the five services while that of proc4 weights the lowest. The remaining three services have equal weights in the system. The experiments results shown in Fig. 11 further illustrate the relationships between services and system reliability growth. The system reliability grows fastest with proc3 and slowest with proc4 among the five services. However, by changing each service individually, the system reliability can only enhance at most 37%.

![Figure 11. Sensitivity analysis](image)

The sensitivity analysis can help reliability design in multiple ways. For examples, it can help to identify the services that contribute most to the system reliability or unreliability. The key services can be optimized from either architecture design perspective or runtime fault-tolerance perspective to be assigned with a service pool with more alternatives. The application builder can decide on the trade-offs between cost and reliability. It can also introduce the adaptation mechanism so that the system can automatically adjust the distribution and the length of service pools in case that the reliability obtained at runtime is below the predefined thresholds.

**VI. CONCLUSION**

Service-based systems can be constructed dynamically by integrating loosely coupled data and functions into workflows. These data and functions are designed to have standard interfaces and the interfaces are exposed as Web-based remote methods. To address the challenges of dynamic collaboration, the paper proposes a hierarchical modeling framework that models the system reliability as the superimposition of data, service, fault-tolerant mechanism, and service compositions. With tool support, the reliability models can be automatically established and continuously adjusted online so that it can be flexible adaptive to the changes in the service composition and system configuration. Experiments show that the dynamic modeling approach and the decomposed models reflect the characteristics of SOA systems and are helpful for reliability analysis.

**ACKNOWLEDGMENT**

This research is supported by National Science Foundation China (No. 60603035), Beijing Natural Science Foundation (No. 4072014) and National High Technology Program 863 (No. 2006AA01Z157).

**REFERENCES**


