Journal of Universal Computer Science, vol. 20, no. 3 (2014), 302-328
submitted: 24/3/13, accepted: 4/2/14, appeared: 1/3/14 © J.UCS

Service Composition Management: A Risk-Driven Approach

Shang-Pin Ma
(National Taiwan Ocean University, Keelung, Taiwan
albert@ntou.edu.tw)

Ching-Lung Yeh
(National Taiwan Ocean University, Keelung, Taiwan
19957009@ntou.edu.tw)

Ping-Chang Chen
(National Taiwan Ocean University, Keelung, Taiwan
10157026@ntou.edu.tw)

Abstract: How to effectively and efficiently monitor, manage, and adapt web services in a composite service or a service-oriented application is becoming a significant issue. In this paper, we argue that it is insufficient to only solve emerging service faults at the deployment time or runtime; instead, we propose that the prediction of service faults is equally important. We devised a risk-driven service composition management (RDSCM) approach including four main phases: (1) preparation, (2) planning, (3) monitoring and reaction, and (4) analysis. By applying the proposed approach, risky component services can be removed earlier, and the fault source can be tracked and identified more easily when a fault occurs. We developed a prototype to realize the proposed approach, and conducted experiments to verify the approach. The implementation and experiments demonstrate that the proposed risk-driven approach can effectively and efficiently ensure the robustness of a service-oriented system.
Key Words: service management, risk management, service composition
Category: D.2, D.2.11

1 Introduction

Service-Oriented Architecture (SOA) has become an important trend in software engineering for developing loosely coupled applications and integrating legacy and modern systems. Accordingly, how to effectively and efficiently monitor, manage, and adapt web services in a composite service or a service-oriented application is also becoming a significant issue. Numerous approaches have provided solutions to perform service monitoring and provide management for improving various types of quality of service (QoS), such as availability, reliability, and response time. In these approaches, [Zeng et al. 2004] and [Alrifai et al. 2012] focused on selecting web services to satisfy user requests according to the QoS degrees, and [Calinescu et al. 2011] proposed a multiphase lifecycle for service-based systems (SBS), which can fulfill user demands by using dynamic adaptation. [Baresi et al. 2007] presented multiple methods, including retry, rebind, reorganize, and changing monitoring rules, to recover service faults during execution, and [Friedrich et al. 2010] applied the error chain paradigm to recover multiple affected services when a fault occurs.

In this paper, we argue that only solving emerging service faults at the deployment time or runtime is insufficient; instead, we believe that predicting and tracking service faults is equally important. Risk analysis is an approach that is commonly used in the project management domain [Kwan and Leung 2011]. The potential problems that may hinder the development of a project are called risks. If problems occur, a project may be bogged in difficulties. Therefore, the risk should be reduced before project execution. Risk management techniques are currently used in various domains, such as electricity [Kettunen et al. 2010], wireless network security [Tsai and Huang 2011], medical services [Schmuland 2005], and software design [Verdon and McGraw 2004]. The risk concept is also applied to enhance the service-oriented design process for selecting appropriate business partners [Kokash 2007]. These efforts demonstrate that risk management can effectively avoid the risk of damage.

Thus, in this study, we apply the risk notion to the web service management mechanism to efficiently foresee possible problems or weaknesses in a service-oriented system. Our proposed risk-driven service composition management (RDSCM) approach includes four main phases: (1) preparation, (2) planning, (3) monitoring and reaction, and (4) analysis. In the proposed approach, we develop a method to calculate the service risk exposure for estimating the possible faults that reside in a service composition, and devise a process to construct a service dependency graph (SDG) and fault tracking paths (FTP) to track service faults and perform appropriate recovery activities. By applying the proposed approach, risky component services can be removed earlier, and the fault source can be tracked and identified more easily when any fault occurs. We believe that the proposed risk-driven approach can effectively and efficiently ensure the robustness of a service-oriented system. The goals of this study are twofold: (a) to provide a systematic approach to efficiently predict and monitor the risks of component services and composite services and supply appropriate risk mitigation actions; and (b) to furnish a systematic method for effectively tracking failed services and recovering composite services by performing multiple recovery actions.

The structure of the paper is organized as follows: Section 2 presents a review of extant research related to web service management. Section 3 introduces the details of the proposed service composition management approach. Section 4 shows the prototype system, the RDSCM engine (RDSCME), and the experimental setup and results. The final section offers a conclusion.

2 Related Work

Service management covers a range of activities, such as configuring and collecting metrics, tuning performance, and ensuring QoS (e.g., reliability, availability, and response time). Here, we introduce various studies that have addressed different issues in service management.

Solving the optimization problem for the service composition is one of major research tracks in the field of web service management [Liu and Deters 2008]. [Zeng et al. 2004] presented a middleware platform, AgFlow, to enable the quality driven composition of web services. AgFlow provided two QoS-driven service selection approaches (local optimization and global planning) to maximize user satisfaction and an adaptive execution engine that reacts to changes occurring during the execution of a composite service. AgFlow applied the Simple Additive Weighting (SAW) technique to select optimal web services and considered five generic QoS values: price, duration, reputation, success rate, and availability. [Alrifa and Risse 2009, Alrifai et al. 2012] presented a hybrid solution that combines global optimization with local selection techniques to address the performance issue of normal global optimization for service composition. They used mixed integer programming to find the optimal decomposition of global constraints into local constraints and to find the best services that satisfy the local constraints. Although these efforts can effectively find service compositions with optimal QoS, they focus on preparation/planning phase in the whole service management lifecycle and do not supply comprehensive monitoring and reaction methods. Besides, these approaches need spend considerable computation cost to perform local selection or global optimization. In our approach, efficient fault prevention and recovery is our principal goal. Accordingly, we provided an efficient QoS-driven risk analysis method to avoid utilizing highly risky services, and furnished a systematic approach to recover service faults.

[Calinescu et al. 2011] introduced a tool-supported framework, QoS management and optimization of service-based systems (QoSMOS), for the development of SBSs that achieve QoS requirements by dynamically adapting to changes. QoSMOS is realized in four stages: (1) monitor, (2) analyze, (3) plan, and (4) execute. In the monitoring stage, QoSMOS can discover requirement violations and trigger adaptation strategies for an SBS. Different from QoSMOS, our proposed approach includes a preparation phase before execution to prepare the necessary artifacts to perform risk analysis and ease further monitoring and recovering actions. Besides, comparing with QoSMOS, our approach can efficiently remove highly risky services in the early stages.

[Baresi et al. 2007] presented an approach to address faulty behaviors in service-oriented systems. They proposed several reaction strategies to make compositions self-healing when service flow faults occur, including retrying, rebinding, reorganizing, changing the monitoring rules, changing the monitoring parameters, calling handlers, notifying, warning, and stopping. In [Baresi et al. 2010], authors proposed the concept of self-supervising BPEL processes that can assess the proper execution behavior and react according to user-defined rules. This approach leverages the techniques of the separation of concerns to maintain the actual service composition process and supervision directives separate at the design time, and intertwines the two elements at runtime. Our approach also covers monitoring and recovering activities, and applies reaction strategies, which can be arranged in the planning phase according to the risk analysis results and can be performed in the monitoring and reaction phase.

[Friedrich et al. 2010] proposed a self-healing approach to handle exceptions in service-based processes, and repair faulty activities by using a model-based approach. This approach improves a service process based on a set of design-time repair actions and rules for applying such repair actions. This approach follows the error chain paradigm, in which a failure in a running process may be caused by different faults in the system components. Our approach also emphasizes the error chain concept, and devises a fault tracking method. Notably, our approach leverages QoS information, which is missing in Friedrich's work, to perform risk-driven fault prevention efficiently.

[El Haddad et al. 2008] proposed a selecting algorithm for composing web services considering transactional properties and QoS characteristics. They stated that a web service has three behavioral properties, including a non-transactional property [the retriable (r) property], and two transactional properties [the pivot (p) and compensable (c) properties]. Accordingly, the services are divided into four types: p, c, pr, and cr. They also adopted the risk concept, and defined two risk levels for a transactional system: The service results can or cannot be compensated for. We propose a different QoS-based risk model, and use the types of behavioral properties to perform appropriate recovery actions, which are lacking in El Haddad's work.

2.1 Analysis and Comparison

To elaborate the contribution of the proposed approach, comparison with other service management methods along with four dimensions: stages involved in the service management lifecycle, service selection method, anomaly detection mechanism, and service recovery mechanism, are provided. The analysis and comparison is shown in Table 1.

Comparing with these representative works for web service management, RDSCM covers all necessary aspects, including preparation, planning, monitoring, reaction, and analysis, in the management lifecycle, whereas other solutions only focus on specific stages. Another major difference and advantage of RDSCM is that all required activities for service management, such as selecting services, detecting anomalies, and recovering service faults, are accomplished through

Table 1: Comparison with other service management methods

	Stages involved in the service management lifecycle	**Service selection method**	**Anomaly detection mechanism**	**Service recovery mechanism**
AgFlow	Planning and re-planning	Local optimization and global planning	Detecting the situation when a service is unable to attain its expected QoS	Re-planning using the same global optimization method
Alrifai's approach	None	Global planning with local optimization	None	None
QoSMOS	Four stages: monitor, analysis, plan, and execute	None	None	None
Baresi's approach	Monitoring and reaction	None	Detecting violation of pre-defined rules	(1) Reacting based on pre-defined rules; (2) Supporting a variety of recovery actions, such as re-try, re-bind, re-organize, change monitoring rule, etc.
Friedrich's approach	Process design, diagnosis, and repair	None	None	Performing repair actions, such as retry, compensate, and substitute based on the error chain paradigm
El Haddad's approach	None	Service selection based on transactional properties and QoS characteristics	None	None
RDSCM (Our approach)	Four phases: preparation, planning, monitoring and reaction, and analysis	Service substitution based on risk analysis	Detecting the situation when the risk of a service becomes high	Performing recovery actions, such as retry, compensation, or substitution, based on the generated fault tracking path

QoS-based risk analysis and tracking. The proposed risk analysis and tracking method can efficiently prevent possible faults with few computation resources.

3 RDSCM: Risk Driven Service Composition Management

This section describes the proposed approach in detail, including the core concepts and the process of RDSCM.

3.1 Core Concepts for the Proposed Approach

To establish the proposed service composition management mechanism, we introduce the significant concepts first by using the UML (Unified Modeling Language) class diagram, and represent each concept as a class (Figure 1). A composite service is the aggregation of multiple component services that are described by service profiles. An SDG (service dependency graph) can be produced based on the flow of a composite service to ease service fault tracking. The component service risk consists of risk impact and risk probability, which are calculated according to the composite service flow structure and the historical QoS data. The component service risk can be mitigated by performing replication or substitution actions. The composite service risk is estimated by accumulating the values of all the component service risks.

A service flow instance is initiated when the service requester uses a composite service. When any service failure occurs during the execution of the service flow instance, we can track the FTP (fault tracking path), which consists of a service fault source, intermediate nodes, and a service failure occurrence point to locate the cause of faults and to perform appropriate recovery actions. The FTP is automatically generated based on the built SDG. Three reaction strategies are included in this study: “Substitution,” “Compensation,” and “Re-invoke.” In the following subsections, we elaborate on important concepts, including the qualities of web service, the service profile, the service risk, and the reaction strategy.

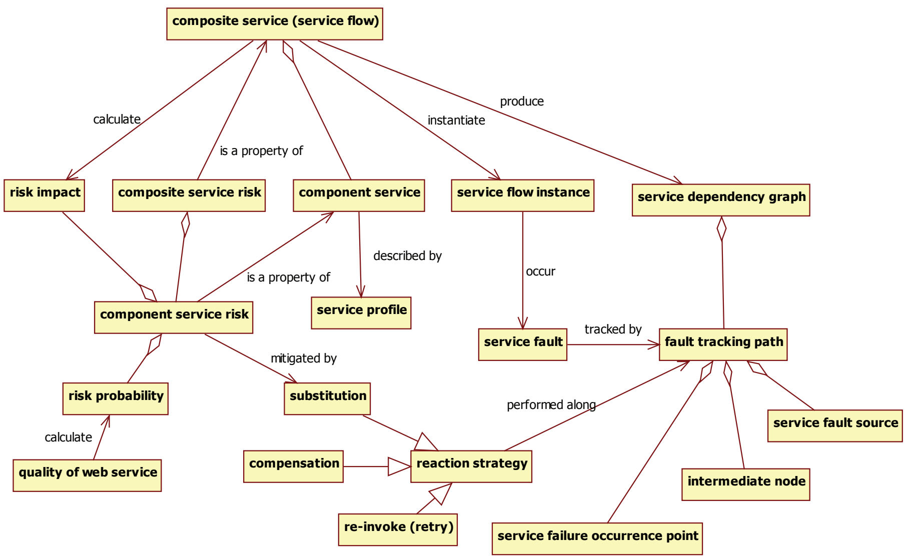


Figure 1: Conceptual model

3.1.1 Motivating Example: Travel Service

To illustrate the proposed approach, we have provided a motivating example of an application designed to assist travellers (see Figure 2). This example presents a composite service comprising 10 component services, including make a flight reservation, make a hotel reservation, and so forth. Each component service can be replaced by other interface-compliant services.

#	Service interface name	Service ID
S_1	Search flights	SF01
S_2	Make a flight reservation	MAFR01
S_3	Generate flight reservation report	GFRR01
S_4	Generate E-tickets	GET01
S_5	Cancel the flight reservation	CTFR01
S_6	Search hotels	SH01
S_7	Make a hotel reservation	MAHR01
S_8	Generate hotel reservation report	GHRR01
S_9	Print the hotel information	PTHI01
S_{10}	Cancel the hotel reservation	CTHR01

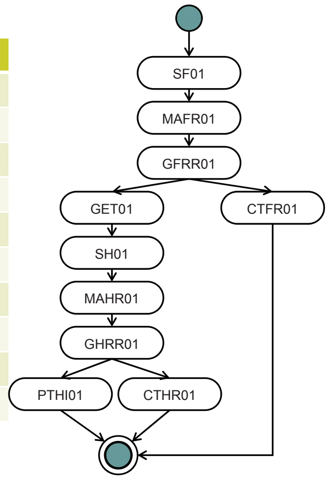


Figure 2: Example services associated with travel (travel service)

3.1.2 Quality of Web Service

We consider three generic quality attributes for evaluating component services:

– Response Time. Given a component service s, the response time $q_t(s)$ measures the delay time in seconds between the moment the service s obtains the input data and the moment the component service s sends the output data.

– Availability. The availability $q_{av}(s)$ for a component service s is the probability that the service is accessible. The value of the availability of a service s is calculated using Equation (1).

$$q_{av}(s) = \frac{T_a(s)}{\theta} \tag{1}$$

where T_a is the total duration in which the service s is available during the last θ seconds.

– Reliability. The reliability $q_{re}(s)$ of a component service s is the probability that a request is responded to correctly. The value of the availability of a service s is calculated using Equation (2).

$$q_{re}(s) = \frac{N_c(s)}{K} \quad (2)$$

where N_c is the number of times the service s has been successfully completed within the maximum expected timeframe, and K is the total number of invocations.

It is noted that the term "component service" indicates an operation of a web service component in this study, not the whole service component including multiple operations.

3.1.3 Service Profile

To efficiently manage component services, we must obtain and store the important attributes of a component service, including the service name, service interface, service type, service input, service output, and service provider. The service name is a unique identifier for a component service. The service interface plays the role of the specification with which the service is realized. Candidate failover services should follow the same service interface. Service inputs and outputs are used to build the SDG, which aims to ease fault tracking (described in Section 3.2). The service type can be identified from two viewpoints: *Retriable* and *Compensable*. If a component service guarantees a successful termination after a finite number of invocations, it is *Retriable*; otherwise, it is not retriable. If the component service supports compensable transactions, it is *Compensable*; conversely, if the component service execution does not affect the state of the service, it is *Non-compensatory*. Therefore, services can be classified into four types: N (non-compensatory), C (compensable), NR (non-compensatory and retriable), and CR (compensable and retriable).

3.1.4 Service Risk

In the proposed approach, we use the notion of risk to predict possibility faults or QoS declines. According to the definitions in [Verdon and McGraw 2004], two important concepts are relevant: (a) the object of the protection efforts, which can be a system component, data, or a complete system; and (b) the risk, which is the probability that an object suffers an event for a given negative impact. For analyzing the risk of a service composition, two levels of protection targets may be considered: the service execution engine, which is responsible for controlling and managing service flow, and the component services, which are hosted in distributed web servers. Because the service execution engine is always embedded in the central service bus, it is most likely constructed with better hardware resources than individual web servers, and is more likely built with more mature fault tolerance mechanisms at the operating system or server levels to achieve higher stability. Thus, in this study, we focus on the risk analysis and tracking of component services. We follow the risk definition specified in CMMI [CMMI Product Team 2010] to calculate the risk exposure of a component service for a service composition by using Equation (3):

$$Risk = Probability \times Impact \tag{3}$$

Two elements of risk, risk probability and risk impact, are described as follows.

Risk Probability. Risk probability is a value that estimates the stability of a component service. We adopt the QoS values of the component service by using historical data to calculate Equation (4):

$$P = (1 - W_p) \times T + W_p \times AR \tag{4}$$

where W_p is a weight value to represent the importance of an AR value, AR is the rating of availability and reliability, as shown in Table 2, and T is a value calculated using Equation (5) to evaluate the stability of the service response time.

Table 2: Rating rules for availability and reliability

Rule	AR Rating
Availability or Reliability is between 0% and 90%	1.0
Availability or Reliability is between 90% and 99%	0.8
Availability or Reliability is between 99% and 99.9%	0.6
Availability or Reliability is between 99.9% and 99.999%	0.4
Both Availability and Reliability are more than 99.999%	0.2

$$T = \frac{STD(RT)}{Mean(RT)} \tag{5}$$

where $STD(RT)$ is the standard deviation of all records of response time for a service, and $Mean(RT)$ is the arithmetic mean of all records of response time for a service.

The reason why applying only one rating for availability and reliability is because either low availability or low reliability should be treated as high risk probability, thus, a service is regarded as with low risk probability only when it is with both high availability and high reliability. Accordingly, we devised a set of rating rules (shown in Table 2) to address the above concerns by considering availability and reliability simultaneously. Currently, the value ranges of availability/reliability in these rules are pre-defined and the rating is derived by normalizing the popular 1-to-5 rating scale to the scale between 0 and 1. These rules can be adapted by the system administrator for fitting different contexts in different service domains.

To illustrate how to compute T, we take the QoS data shown in Table 3 as an example. After calculation, the value of AR is 0.8, and the value of T is 0.2. If $w_p = 0.7$, the value of risk probability P is 0.62. Risk probability would be calculated when the service flow starts, and could be continuously updated in the following phases.

Table 3: Examples for the rating of response time

Attribute	Value
Arithmetic mean of response time	8.20
Standard deviation of response time	1.64
Availability	99.60%
Reliability	98.25%
Probability	0.62

Risk Impact. Risk impact is a value that indicates possible damage when a component service in a composite service malfunctions. In this study, we calculate risk impact by extending the method proposed in [Ma et al. 2010] because, if a component service is more important, the damage is greater when this service cannot operate correctly. In this study, the risk impact value is aggregated by the Importance Point (IP) assigned by the user and the analysis results of execution path analysis. The user can assign an IP to each node, ranging from 1 to 10, with a higher score indicating that the corresponding component service is more important than others. We then convert all scores to reduce their sum to 1, to derive I_u. On the back side, we leverage the basis path-testing method [Hutcheson 2003] to retrieve all possible representative execution paths for the composite service, and assign an Execution Point (EP) score to each service node based on their probability of emergence by applying Equation (6). We then convert all scores to reduce their sum to 1, to derive I_e. Finally, we can calculate the risk impact through Equation (7).

$$EP(S_n) = \frac{BP(S_n)}{BP} \tag{6}$$

where $BP(S_n)$ is the number of basis paths that include service node n, and BP is the number of all basis paths.

$$I = \frac{(1 - w_i) \times I_e + w_i \times I_u}{I_{max}} \tag{7}$$

where I_u is the converted score assigned by users, I_e is the converted score analyzed from the execution path, w_i is the weight for representing the importance of I_u in this equation, and I_{max} is the maximum of all I values for all component services.

For the motivating example, we can calculate the impacts of all component services as follows. We define $S = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8, S_9, S_{10}\}$ to represent the composite service. Since booking airline tickets and reserving hotel rooms is the ultimate goal of this composite service, the importance point assigned by the user $UP = \{1, 2, 1, 1, 1, 1, 2, 1, 1, 1\}$, and the weight $w_i = 0.7$ to lay more stress on the user's viewpoint. Three execution paths $\{S_1 - S_2 - S_3 - S_5, S_1 - S_2 - S_3 - S_4 - S_6 - S_7 - S_8 - S_9, S_1 - S_2 - S_3 - S_4 - S_6 - S_7 - S_8 - S_{10}\}$ are in the flow; therefore, the execution points are calculated through execution path analysis $EP = \{1, 1, 1, \frac{2}{3}, \frac{1}{3}, \frac{2}{3}, \frac{2}{3}, \frac{2}{3}, \frac{1}{3}, \frac{1}{3}\}$. After this computation, the value of set $I_u = \{0.083, 0.167, 0.083, 0.083, 0.083, 0.083, 0.167, 0.083, 0.083, 0.083\ \}$, $I_e = \{0.15, 0.15, 0.15, 0.10, 0.05, 0.10, 0.10, 0.10, 0.05, 0.05\ \}$, and $I = \{0.7368, 1, 0.7368, 0.5789, 0.4211, 0.5789, 0.8421, 0.5789, 0.4211. 0.4211\ \}$.

3.1.5 Reaction Strategy

As mentioned, three reaction strategies are included in this study: Substitution, Compensation, and Re-invoke. "Substitution" is a strategy that attempts to select another failover service with the same interface as the faulty one. The interface of a service is recorded in its service profile. "Compensation" is used to restore the correct object states that were correct, in case these were affected by a fault. Only compensable services should be compensated for, whereas non-compensatory services are not required to perform compensation. "Re-invoke" re-executes the same service invocation with the same parameters and contracts.

3.2 Risk-Driven Service Composition Management Process

Based on these concepts, we devised a management process, RDSCM, which includes four sub-processes: preparation, planning, monitoring and reaction, and analysis.

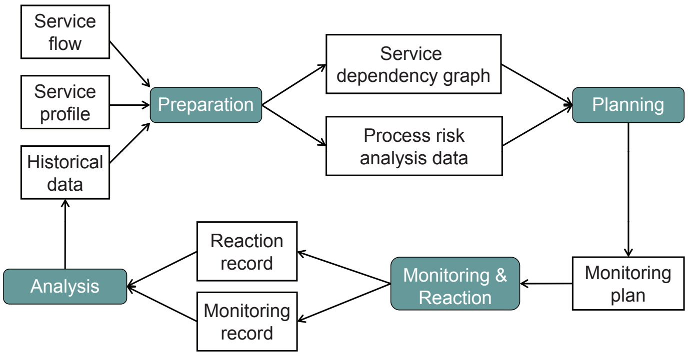


Figure 3: Risk-Driven Service Composition Management Process

As shown in Figure 3, in the preparation phase, RDSCM calculates the risk impact of each component service based on the service flow, and computes the risk probability of each component service according to historical QoS data. The risk exposure values can be determined by aggregating the impact and probability data. In addition, RDSCM also analyzes the service flow and service profiles of all component services in the flow to generate the SDG. In the planning phase, RDSCM produces a monitoring plan based on the SDG, and processes risk analysis data. In the monitoring and reaction phase, RDSCM monitors the composite service according to the monitoring plan. If any fault occurs, RDSCM selects the appropriate recovery actions specified in the monitoring plan. Finally, in the analysis phase, RDSCM analyzes all execution records and updates the historical QoS data, which influence the risk analysis results of other instances of the same composite service or other composite services.

3.2.1 Preparation Process

As shown in Figure 4, in the preparation phase, the risk of the component services that are in the composite service is analyzed. If the risk of a component service is higher than a given threshold, RDSCM attempts to select another service with the same service interface, to replace the risky one. The substitute strategy, such as selecting the service with the best QoS or selecting the service

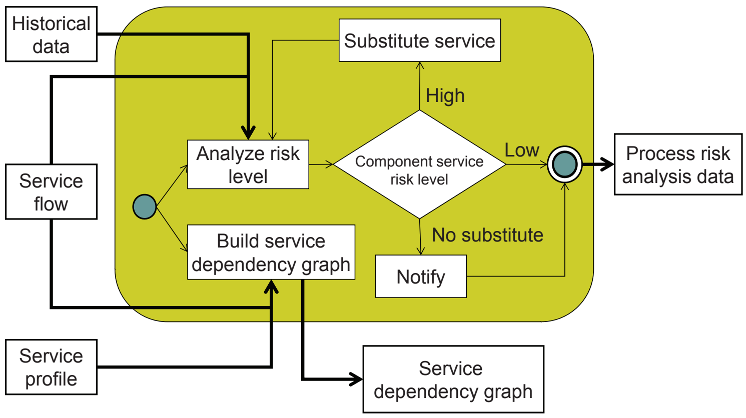


Figure 4: Preparation Process

used most frequently, can be chosen in advance. If the risk of the substitute service is still too high, RDSCM stops the process and shows an alert message to the user.

Building the Service Dependency Graph. In the preparation phase, the SDG is built according to Algorithm 1. Opposite to the service flow showing the execution sequence, the SDG represents the dependency relationship among the component services for a composite service. For constructing the dependency graph, we extract the data flow among the component services (i.e., the source and the target of the service data), and invert the service flow.

In the motivating example, the input and output data among component services are analyzed first (shown in Figure 5). The dependency graph of the travel service is generated based on Algorithm 1 (shown in Figure 6). In this case, because service MAFR01 accepts input data O_1 from service SF01, we can assert that MAFR01 depends on SF01, and the edge representing this dependency relationship in the SDG is connected from source MAFR01 to target SF01. By following these graph construction rules, a complete SDG can be generated automatically.

3.2.2 Planning Process

In this phase, RDSCM automatically produces a monitoring plan document, which can be modified by the manager. The plan includes four parts, as follows:

1. *Monitoring Attribute*. This part specifies the QoS attributes the manager plans to monitor. In default mode, the SDG monitors the execution time, availability, and reliability.

Algorithm 1 Building the Service Dependency Graph

```
retrieve the list of component services which are in the composite service cs
and generate an array css
declare a graph g
for each service s_i in css do
    for each service s_j in css do
        if output of s_j is a part of input of s_i then
            add s_j to the s_i's pre-service list s_i.slt
        end if
    end for
end for
for each service s_i in css do
    for each service s_k in s_i.slt do
        add an edge from s_i to s_k in graph g
    end for
end for
return g
```

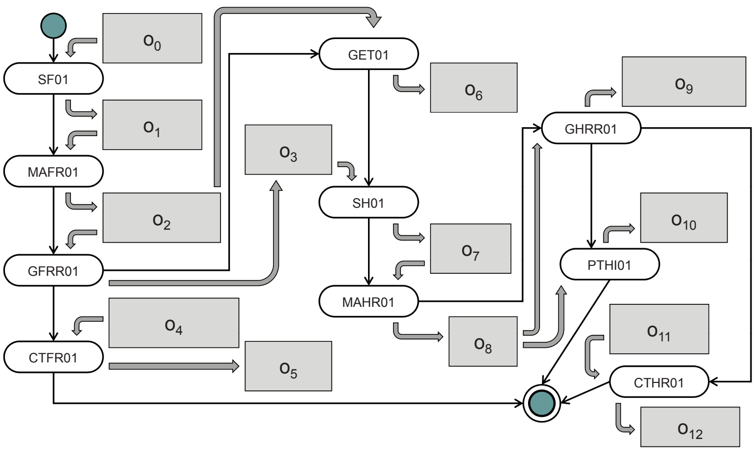


Figure 5: The Dataflow for the Travel Composite Service

2. *Monitoring Threshold.* This part sets the threshold for high risk or medium risk.

3. *Monitoring Frequency.* In the preparation phase, the risk degree is computed and classified into three types: high, medium, and low. The highly risky service is substituted immediately, but the services with medium or low risk require careful monitoring. Thus, this part arranges the monitoring frequency (i.e., the cycle time of the QoS probe). The frequency of services with medium risk is generally more intensive than services with low risk.

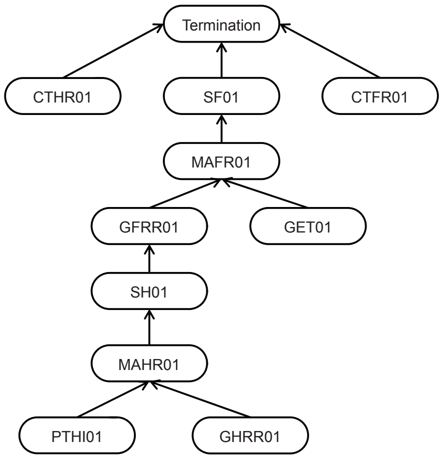


Figure 6: The Service Dependency Graph for the Travel Composite Service

4. *Substitution Policy.* This part determines the procedure to swap the faulty or risky service. The available substitution policies include choosing the most used services, choosing the service with the best QoS, and choosing the service with the lowest risk.

3.2.3 Monitoring and Reaction Process

In this phase, RDSCM monitors the execution of the composite service based on the monitoring plan. This phase has two main tasks: the first is collecting the execution data with all service instances or additional service-probing records according to the monitoring frequency setting, and the second is detecting and recovering the service faults. We can define an emerging service fault when a component service suffers the following situations: (1) The component service is malfunctioning or missing; (2) The component service cannot satisfy the requirements or returns data with the wrong format; and (3) The component service becomes highly risky because of low availability/reliability or instability of performance.

As shown in Figure 7, in the monitoring and reaction phase, RDSCM enters the waiting state to wait for an instance to be instantiated. If an external fault (a fault outside the current service flow instance) occurs during this time, it affects all instances of this composite service. Therefore, RDSCM increases the risk probability of the component service if faulty instances occur, and inspects its risk degree to determine if it is higher than the threshold. When an instance is instantiated, RDSCM runs the instance. If the instance is faulty, RDSCM

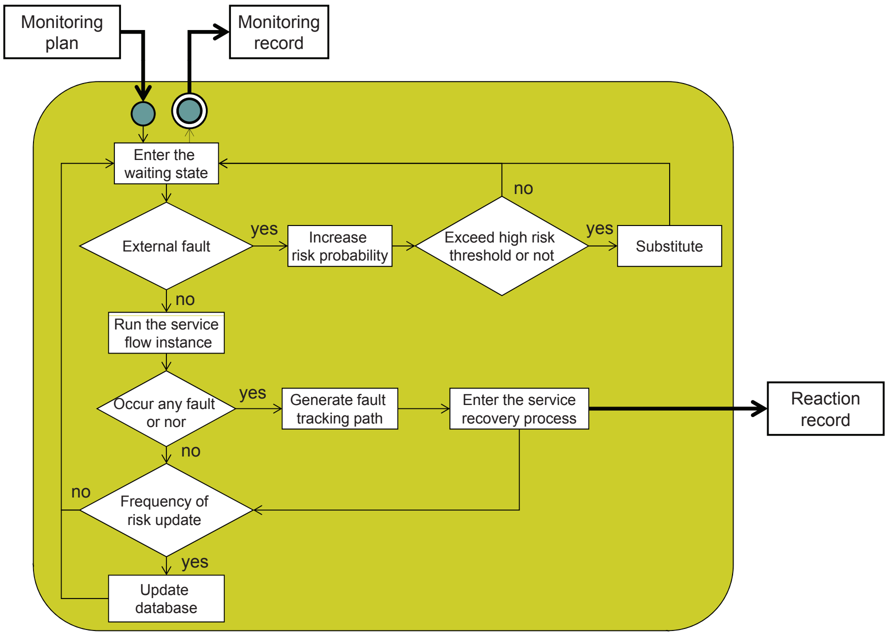


Figure 7: Monitoring and Reaction Process

generates an FTP. Based on the FTP, RDSCM enters the service recovery process to recover the instance. Finally, RDSCM bases the configuration property of the risk update frequency to update the QoS database. In the following subsections, we discuss generating the FTP and the service recovery process.

Generating the Fault Tracking Path. If an instance reveals faults, RDSCM uses the SDG to produce an FTP, and selects reaction strategies and performs recovery actions for the component services in the FTP. Following the error chain paradigm, an FTP includes the failure occurrence point, the fault source, and zero or more intermediate services. In addition, zero or more non-failure services precede the fault source in the service flow. For example, in the motivating example (shown in Figure 8), if service SH01 is the failure occurrence point that reveals the fault, the possible longest FTP SH01-GFRR01-MAFR01-SF01 is established first because any of these four services may be the fault source. If service SF01 is confirmed as the fault source, the actual FTP is also SH01-GFRR01-MAFR01-SF01, and service GFRR01 and MAFR01 are assigned as intermediate services. If service GFRR01 is determined as the fault source, the FTP is SH01-GFRR01 without any intermediate service.

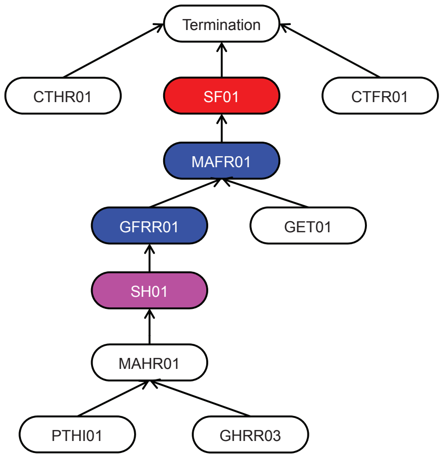


Figure 8: The Fault Tracking Path for the Travel Composite Service

Service Recovery Process. When the fault source is determined, the RDSME conducts the service recovery process to fix the flow instance. First, the RDSME increases the risk of the fault source (and updates the risk value into the database), and derives the service type (described in Section 3.1) of the fault source for further processing. Depending on the type, RDSCM performs the recovery actions to recover the source. For example, if the service type of the fault source is CR, RDSCM compensates for this service first, and then re-invokes it and tests if the service is workable. If the service is still malfunctioning, RDSCM selects another interface-compatible service according to the substitution policy to substitute for the faulty service. RDSCM may then compensate for intermediate services or the failure occurrence point first (if the service is compensable) because the incorrect data may affect the business state of these services, and then RDSCM re-invokes the service (if the service is retriable). The reaction strategy of service substitution is unnecessary for non-fault-source services because these services are functioning. By following these steps, all services in the FTP are recovered in the inverse sequence of the FTP, and the faulty composite service is then recovered.

4 Implementation and Experimentation

This section presents a prototype system, RDSCME (RDSCM Engine), and the experimental setup and results.

4.1 System Prototype: Risk-Driven Service Composition Management Engine

In this subsection, we demonstrate the prototype system RDSCME[1] that implements the proposed RDSCM approach. At the beginning of use, the RDSCME retrieves the profiles of all component services to generate a service flow representing the composite service. In this case, the composite service flow is a travel service that includes 10 component services, such as booking flight tickets and reserving hotels. Second, if any component service is judged too risky from the analysis of its risk probability and risk impact, the system automatically substitutes another interface-compatible service for the risky service. Third, the system generates the SDG. The system provides a user interface to display the service flow by choosing Label 1, service attributes by choosing Label 2, the SDG by choosing Label 3, and the service recovery flow by choosing Label 4.

The fundamental functionality of the RDSCME is producing a service flow graph (Figure 9) by retrieving the description of the composite service and the profiles of component services. Vertices represent the component services, and the edges represent the execution sequence. By pressing the “Analysis” button, the user can browse the attributes of all the component services in the table shown in Figure 10. In this table, if the risk level of any component service is higher than the threshold, the corresponding row is displayed in red to emphasize the risky situation. The user can then press the “Substitute” button to replace the highly risky service by another service according to the predefined substitution policy. In this case, Service GHRR03 is the substitute of Service GHRR01, based on the substitution policy. When all the component services in the composite service are not highly risky, the RDSCME can generate the SDG.

For simulating the process of service fault tracking and recovery, the user can first press the “Service Dependency Graph” button to browse the SDG. In the graph, the vertices are the component services, and the edges are the dependency relationships. The RDSCME provides a simulation function to allow the user to simulate the fault tracking. The user can select a failure occurrence point, and then RDSCME generates a chain to express the path from the failure occurrence point to the root in the SDG. One of the services in this path is the fault source. The user can select a fault source in this chain, and RDSCME splits the chain into three sections. The failure occurrence point is in purple, the intermediate nodes are in blue, and the fault source is in red (shown in Figure 11).

According to the proposed service recovery process, the RDSCME can generate the recovery plan: a new service flow (Figure 12). In this case, the fault source is identified as the first component service: Search Flight Service SF01; five new activities are inserted after the failure occurrence point: Search Hotel

[1] The system prototype can be accessed via http://resrv-92.se.ntou.edu.tw:8080/RDSCM/prototype/RDSCME-prototype.zip

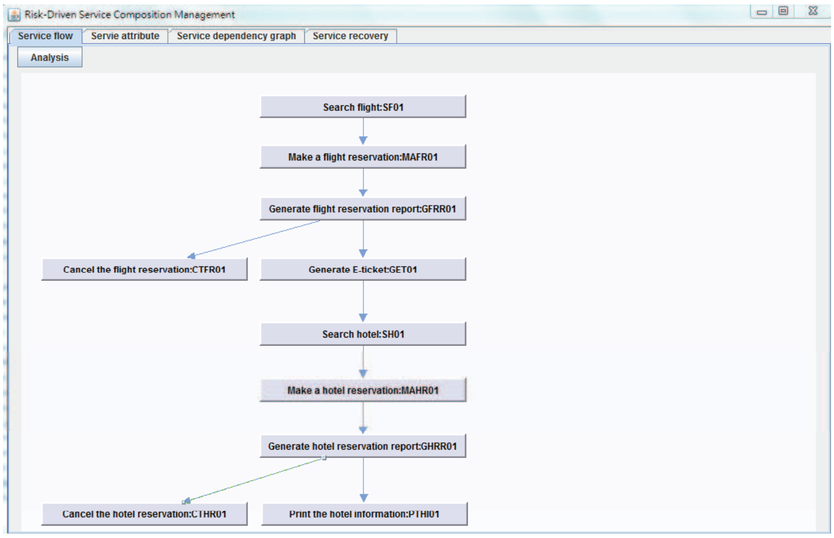
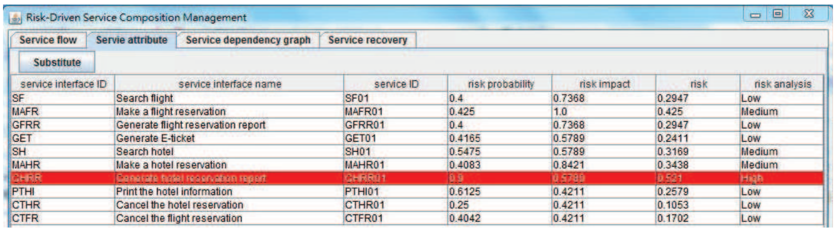


Figure 9: Service flow GUI



service interface ID	service interface name	service ID	risk probability	risk impact	risk	risk analysis
SF	Search flight	SF01	0.4	0.7368	0.2947	Low
MAFR	Make a flight reservation	MAFR01	0.425	1.0	0.425	Medium
GFRR	Generate flight reservation report	GFRR01	0.4	0.7368	0.2947	Low
GET	Generate E-ticket	GET01	0.4165	0.5789	0.2411	Low
SH	Search hotel	SH01	0.5475	0.5789	0.3169	Medium
MAHR	Make a hotel reservation	MAHR01	0.4083	0.8421	0.3438	Medium
GHRR	Generate hotel reservation report	GHRR01	0.9	0.5789	0.521	High
PTHI	Print the hotel information	PTHI01	0.6125	0.4211	0.2579	Low
CTHR	Cancel the hotel reservation	CTHR01	0.25	0.4211	0.1053	Low
CTFR	Cancel the flight reservation	CTFR01	0.4042	0.4211	0.1702	Low

Figure 10: Service attribute table

Service SH01. If SF01 does not operate well temporarily, we can compensate and re-invoke SF01 (the first activity) to perform a recovery; otherwise, the substitute service SH02 is invoked (the second activity) to assume the SH01 task. Services MAR01, GFRR01, and SH01 should be compensated for and re-invoked sequentially (the third, fourth, and fifth activities) to complete the recovery flow.

4.2 Experimentation

This section describes the verification of the proposed approach. The goal of the experiments[2] is demonstrating the ability of the proposed process to efficiently and effectively enhance the robustness of the composite services. We devised two

[2] Experimental setup and results can be accessed via http://resrv-92.se.ntou.edu.tw:8080/RDSCM/experiments/RDSCM-experiments.zip

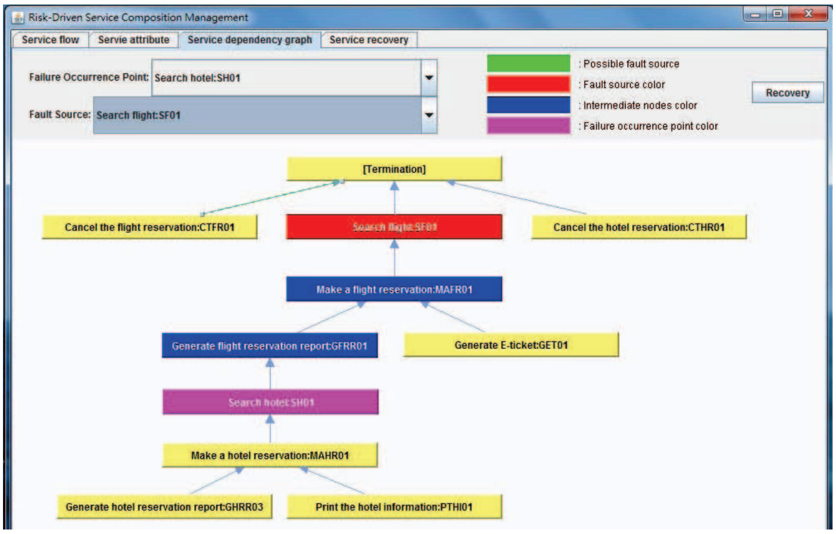


Figure 11: Displaying the Fault Tracking Path (FTP) in the Service Dependency Graph (SDG)

experimental scenarios: (1) Simplified travel composite service using real web services; and (2) Complex travel composite service using large-scaled QoS data of virtual web services. For both experiments, we compared our approach with the traditional service execution method without applying any service management methods, and the widely-used service optimization method, SAW (Simple Additive Weighting techniques) [Ye and Mounla 2008], which can effectively find service combinations with good QoS. Notably, the SAW method is utilized by AgFlow [Zeng et al. 2004], a widely-cited approach. In the SAW-based service selection method, the QoS values are normalized by scaling them to a value between 0 and 1. The total QoS score of a service is the summation of values of all QoS factors. Component services can be selected according to the calculated QoS score before the execution of the composite service. Comparing with SAW-based service selection, RDSCM only swaps component services for highly risky ones without actively selecting component services.

For both two experiments, we compared three alternatives: traditional service execution, RDSCM, and SAW-based service selection, from three viewpoints: fault occurrence, preparation time and the rate of service utilization. The preparation time is the time spent (before service execution) to select services in the SAW-based service selection method or the time spent to perform risk analysis and swap services in RDSCM. Obviously, there is no preparation time for the traditional service execution method. The rate of service utilization is the pro-

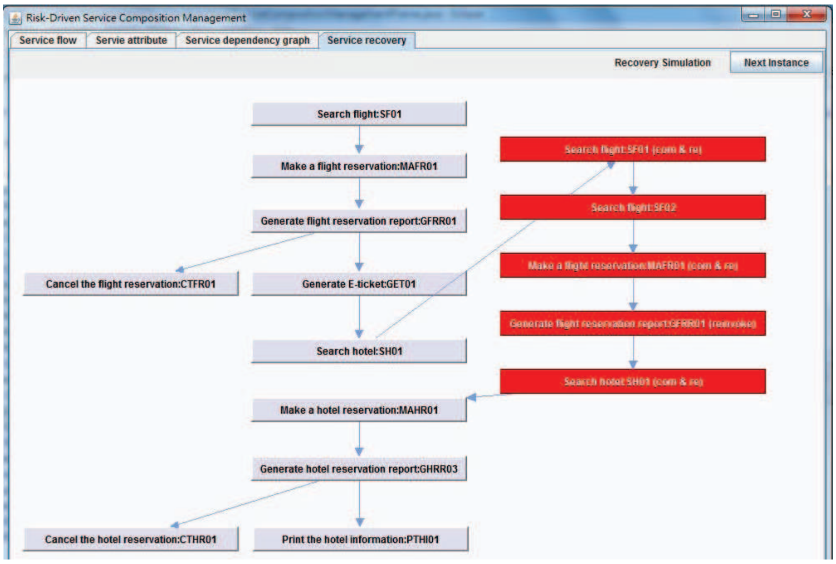


Figure 12: The flow of recovery actions

portion of selected and utilized component services to all candidate component services.

Regarding the experimental environment, all experiments were conducted in a desktop computer with the following configuration: Intel i7-2600 3.4GHz with 4G RAM, 500G hard disk, and Windows 7 (32 bit). In next two sub-sections, we discuss these two scenarios in detail.

4.2.1 Experimentation for Real Web Services

In the first experimental scenario, we devised another relatively small process (shown in Figure 13) which offers travel service by comprising three abstract component services: HBS (Hotel Booking Service), HN2LLS (HotelName to Latitude/Longitude Service), and LL2AS (Latitude/Longitude to Address Service). Each abstract component service can bind five real web services provided by Google, Bing, EzTravel, LionTravel, etc.. Historical QoS data for these 15 web services were retrieved through probing all services twice per minute by using Apache JMeter[3], a Java-based performance measurement tool.

We set the high-risk threshold as 0.6 and conducted this experiment 125 times for visiting all service combinations. For the risk probability, we set the weight of A/R (i.e., w_p in Equation 4) as 0.7 to lay emphasis on the importance

[3] http://jmeter.apache.org/

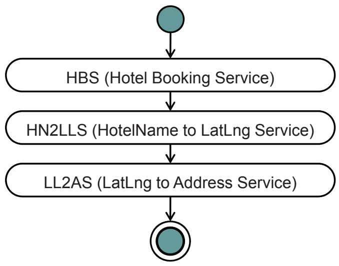


Figure 13: Simplified travel composite service

of availability/reliability. For the risk impact, we also set the weight of the user assignment (i.e., w_i in Equation 7) as 0.7 to stress on the importance of the user viewpoint, and assigned the IP of the "hotel booking service" as 2, and the other two services as 1.

Figure 14, 15, and 16 show the experimental results. Regarding the effect of fault prevention, approximately 95% of the faults (comparing with the traditional service execution method) are avoided through the proposed risk-driven management approach. Although the SAW-based service selection method shows the same performance of fault prevention with RDSCM, it needs more preparation time (122.86% of RDSCM) and relies on a small number of component services with high QoS score (it utilized 20% of all candidate component services); Contrarily, RDSCM can efficiently prevent possible service faults by spending less preparation time and decentralizing loads to higher proportion of component services (RDSCM utilized 55.33% of all candidate component services). Distributing loads can help avoiding degradation of QoS for the services with high QoS score.

4.2.2 Experimentation for Large-Scaled Virtual Web Services

The second experimental process is based on the motivating example: the travel composite service, comprising 10 abstract component services to perform a large-scaled verification. For each component services, we prepared 500 corresponding virtual services with QoS values that are normally distributed. We set the high-risk threshold as 0.3. For the risk probability, we set the weight of A/R as 0.7 to emphasize the importance of availability/reliability. For the risk impact, we set the weight of the user assignment as 0.5 to balance the importance of the user assignment and basis path analysis, and assigned the IP of both the "make a flight reservation" service and the "make a hotel reservation" service as 2, and the others as 1.

For simulating service faults during the execution of a composite service,

Figure 14: The experiment results for real services: faults occurred

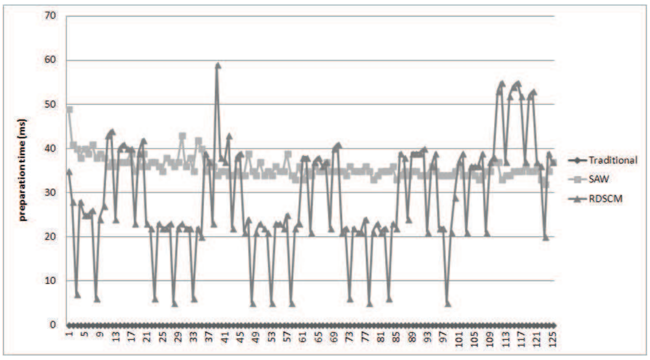


Figure 15: The experiment results for real services: preparation time

we generate faults according to the risk probability of each component service. Initially, we generate a random number γ ranging from 0.0 to 1.0. In general cases, if the risk probability of a component service is larger than γ, a fault is generated by our simulator. However, to simulate the situation of "the higher the risk probability is, the more easily a fault occurs," we separate the services into five levels according to risk probability. For the services with higher risk probability, we lower the chance to generate faults. Through this method, we can simulate service fault occurrences similar to actual cases.

We simulate the execution of travel composite services 1,000 times for the three methods, and then repeat this experiment five times. In other words, we have five sets of experimental results including 1,000 data items of service execution. Figure 17, 18, and 19 show the experimental results. The experiment results are similar to the experiment for real services. Regarding the effect of

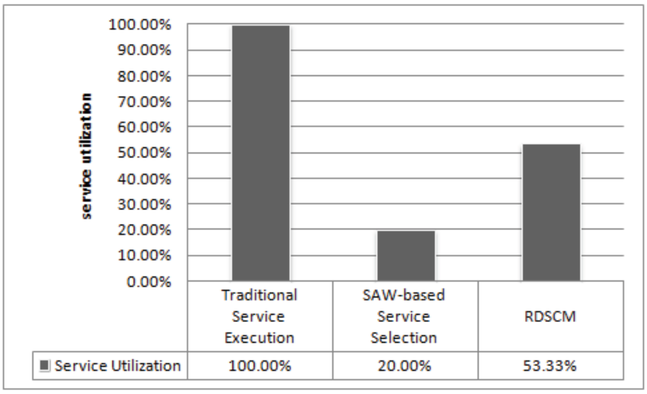


Figure 16: The experiment results for real services: rate of service utilization

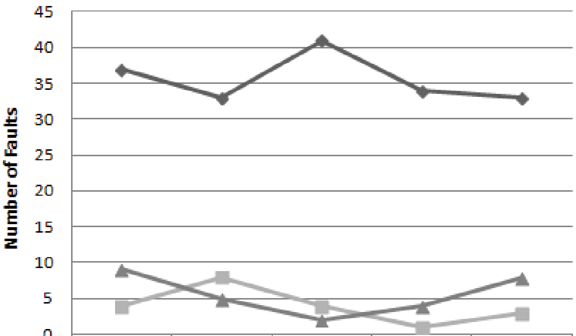
fault prevention, approximately 84% of the faults (comparing with the traditional service execution method) are avoided by using the proposed risk-driven management approach. In this experiment, although the SAW-based service selection method also shows almost the same performance of fault prevention with RDSCM, it still needs more preparation time (average 127.6%) and relies on a few component services with high QoS score (it only utilized 0.2% of all candidate component services). In a nutshell, RDSCM can efficiently prevent likely service faults by spending less preparation time and distribute loads to higher proportion of candidate component services (RDSCM utilized about 72% of all candidate component services). Besides, the faults occurred can be recovered since RDSCM can perform appropriate recovery actions to heal faulty composite services.

5 Conclusion

This paper presents a risk-driven approach to manage composite web services. The objective of this study is to reduce the probability and damage from service faults. The implementation and experiments demonstrate that the proposed risk-driven approach can effectively and efficiently ensure the robustness of a service-oriented system. The proposed approach offers the following key contributions:

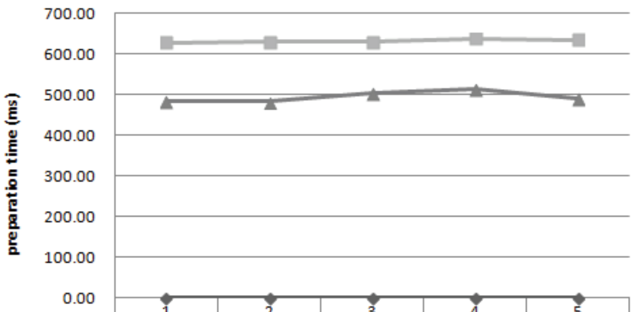
1. Integrates the risk management mechanism with the web service management techniques.

2. Provides a method to calculate the service risk exposure for estimating the possible faults that reside in a service composition.



	1	2	3	4	5
Traditional Service Execution	37	33	41	34	33
SAW-based Service Selection	4	8	4	1	3
RDSCM	9	5	2	4	8

Figure 17: The experiment results for virtual services: faults occurred



	1	2	3	4	5
Traditional Service Execution	0.00	0.00	0.00	0.00	0.00
SAW-based Service Selection	628.66	631.77	630.09	638.66	635.47
RDSCM	485.15	483.87	504.83	513.95	492.34

Figure 18: The experiment results for virtual services: preparation time

3. Furnishes a systematic approach to construct an SDG (service dependency) and an FTP (fault tracking path) to efficiently track service faults and recover the faulty composite service.

Our future research plan is to further analyze all execution records, to determine the common causes of service faults and establish service fault patterns for further preventing fault occurrences.

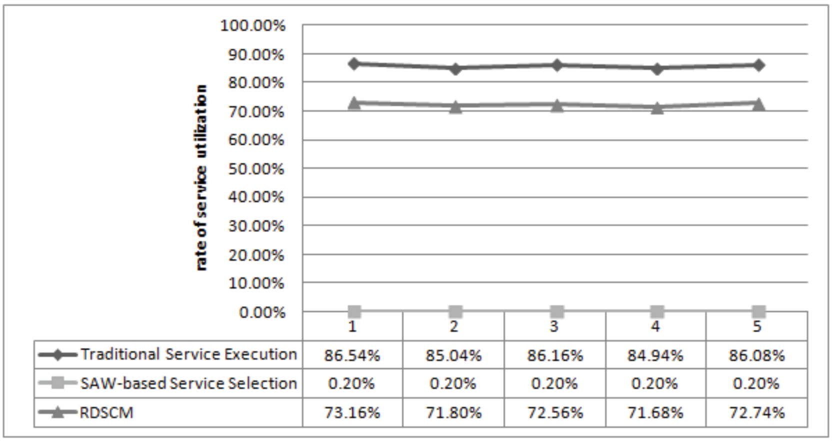


Figure 19: The experiment results for virtual services: rate of service utilization

Acknowledgement

This research was sponsored by National Science Council in Taiwan under the grant NSC 102-2221-E-019-024.

References

[Alrifa and Risse 2009] Alrifai, M., Risse, T.: "Combining global optimization with local selection for efficient QoS-aware service composition"; Proc. 18*th* Int. Conf. on World Wide Web (WWW '09), Madrid, Spain (2009), 881-890.

[Alrifai et al. 2012] Alrifai, M., Risse, T., Nejdl, W.: "A hybrid approach for efficient web service composition with end-to-end QoS constraints"; ACM Transactions on the Web, 6, 2 (Jun 2012) 7:1-7:31.

[Baresi et al. 2007] Baresi, L., Ghezzi, C., Guinea, S.: "Towards self-healing composition of services"; Contributions to Ubiquitous Computing, volume 42 of Studies in Computational Intelligence, Springer Berlin Heidelberg (2007) 27-46.

[Baresi et al. 2010] Baresi, L., Guinea, S., Nano, O., Spanoudakis, G.: "Comprehensive monitoring of BPEL processes"; IEEE Internet Computing, 14, 3 (May 2010) 50-57.

[Calinescu et al. 2011] Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: "Dynamic QoS management and optimization in service-based systems"; IEEE Transactions on Software Engineering, 37, 3 (May 2011) 387-409.

[El Haddad et al. 2008] El Haddad, J., Manouvrier, M., Ramirez, G., Rukoz, M.: "QoS-driven selection of web services for transactional composition"; Proc. Int. Conf. on Web Services (ICWS '08), Beijing, China (2008), 653-660.

[Erradi et al. 2007] Erradi, A., Maheshwari, P., Tosic, V.: "WS-Policy based monitoring of composite web services"; Proc. 5*th* Euro. Conf. on Web Services (ECOWS '07), Halle, Germany (2007), 99-108.

[Friedrich et al. 2010] Friedrich, G., Fugini, M., Mussi, E., Pernici, B., Tagni, G.: "Exception handling for repair in service-based processes"; IEEE Transactions on Software Engineering, 36, 2 (Mar 2010) 198-215.

[Hutcheson 2003] Hutcheson, M.: "Software Testing Fundamentals: Methods and Metrics"; Wiley, New York, USA, 2003.

[Kettunen et al. 2010] Kettunen, J., Salo, A., Bunn, D.W.: "Optimization of electricity retailer's contract portfolio subject to risk preferences"; IEEE Transactions on Power Systems, 25, 1 (Feb 2010) 117-128.

[Kokash 2007] Kokash, N.: "Risk management for service-oriented systems"; Proc. 7*th* Int. Conf. on Web Engineering (ICWE'07), Como, Italy (2007), 563-568.

[Kwan and Leung 2011] Kwan, T.W., Leung, H.K.N.: "A risk management methodology for project risk dependencies"; IEEE Transactions on Software Engineering, 37, 5 (Sep 2011) 635-648.

[Liu and Deters 2008] Liu, D., Deters, R.: "Management of service-oriented systems"; Service Oriented Computing and Applications, 2, 2-3 (Jul 2008) 51-64.

[Ma et al. 2010] Ma, S.P., Kuo, J.Y., FanJiang, Y.Y., Tung, C.P., Huang, C.Y.: "Optimal service selection for composition based on weighted service flow and genetic algorithm"; Proc. Int. Conf. on Machine Learning and Cybernetics (ICMLC 2010), Qingdao, China (2010), 3252-3256.

[Maamar et al. 2005] Maamar, Z., Mostefaoui, S.K., Yahyaoui, H.: "Toward an agent-based and context-oriented approach for web services composition"; IEEE Transactions on Knowedge and Data Engineering, 17, 5 (May 2005) 686-697.

[Moser et al. 2008] Moser, O., Rosenberg, F., Dustdar, S.: "Non-intrusive monitoring and service adaptation for WS-BPEL"; Proc. 17*th* Int. Conf. on World Wide Web (WWW '08), Beijing, China (2008), 815-824.

[Papazoglou and van den Heuvel 2005] Papazoglou, M.P., van den Heuvel, W.-J: "Web services management: A survey"; IEEE Internet Computing, 9, 6 (Nov 2005) 58-64.

[Schmuland 2005] Schmuland, C.: "Value-added medical-device risk management"; IEEE Transactions on Device and Materials Reliability, 5, 3 (Sep 2005) 488-493.

[CMMI Product Team 2010] CMMI Product Team: "CMMI for development, version 1.3"; Technical report, Software Engineering Institute (SEI), Carnegie Mellon University, USA (2010).

[Tsai and Huang 2011] Tsai, H.Y., Huang, Y.L.: "An analytic hierarchy process-based risk assessment method for wireless networks"; IEEE Transactions on Reliability, 60, 4 (Dec 2011) 801-816.

[Verdon and McGraw 2004] Verdon, D., McGraw, G.: "Risk analysis in software design"; IEEE Security and Privacy, 2, 4 (Jul 2004) 79-84.

[Ye and Mounla 2008] Ye, X., Mounla, R.: "A hybrid approach to QoS-aware service composition"; Proc. Int. Conf. on Web Services (ICWS '08), Beijing, China (2008), 62-69.

[Zeng et al. 2004] Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam J., Chang, H.: "QoS-aware middleware for web services composition"; IEEE Transations on Software Engineering, 30, 5 (May 2004) 311-327.