Improving Service Diagnosis Through Invocation Monitoring

Cuiting Chen, Hans-Gerhard Gross and Andy Zaidman
Delft University of Technology, the Netherlands
Email: {cuiting.chen;h.g.gross;a.e.zaidman}@tudelft.nl

***Abstract*—Service oriented architectures support software runtime evolution through reconfiguration of misbehaving services. Reconfiguration requires that the faulty services can be identified correctly. Spectrum-based fault localization is an automated diagnosis technique that can be applied to faulty service detection. It is based on monitoring service involvement in passed and failed system transactions.**

Monitoring only the involvement of services sometimes leads to inconclusive diagnoses. In this paper, we propose to extend monitoring to include also the invocation links between the services. We show through simulations and a case study with a real system under which circumstances service monitoring alone inhibits the correct detection of a faulty service, and how and to which extent the inclusion of invocation monitoring can lead to improved service diagnosis.

***Keywords*-fault localization, spectrum, similarity coefficient**

I. Introduction

Service-oriented architecture (SOA) is an architectural pattern that supports the construction of dynamic, adaptable, and evolvable systems well [1]. Evolution takes place simply through runtime reconfiguration and versioning of the services involved in a SOA, e.g. through exchanging a faulty service for a healthy one [2]. Due to their highly dynamic nature, and the ultra-late binding of the service instances, which is one of the inherent characteristics of SOA [3], traditional development-time quality assurance approaches must be superseded by techniques targeting operation time.

Spectrum-based fault localization (SFL) is a statistics-based, automatic diagnosis technique that has been demonstrated to perform well in pinpointing critical services during runtime [4]. It works by automatically inferring a diagnosis from observed symptoms [5]. A diagnosis is a ranking of the potentially misbehaving services, and the symptoms are observations about service involvement in system transactions, plus pass/fail information for each transaction [4][6].

Even though, SFL was found to perform well in SOA, there are specific circumstances under which suboptimal diagnoses are achieved. A particular issue arises when service instances are tightly coupled. This means that several services are (almost) always invoked in combination, thereby inhibiting discriminative information to be produced in observations used by SFL, leading to inconclusive or ambiguous diagnoses. In other words, tightly coupled services are correctly pinpointed, but assigned the same rank in the diagnosis, as if they were one single service in its own right. Another issue arises when services exhibit fault intermittency. This means a service only fails sometimes when invoked.

Experiments performed for earlier work [7] suggest that ambiguity and intermittency can be resolved through incorporating more detailed information to be used by SFL. One approach is the instrumentation of the services in order to retrieve finer grained observations on the code block level of a service implementation. This has been demonstrated to be successful in [7], but with the limitation that each service must support the instrumentation. Another approach would be the inclusion of information expressing the invocations between the services. We refer to these observations as *invocation link activation observations*. That way, the observations used by SFL do not only incorporate which services have been involved in a particular transaction, but, additionally, which routes through which invocation links between services were taken in a transaction. Our hypothesis is that this additional information can help to improve SFL-based diagnosis.

From these considerations, we can formulate the following research questions to be addressed in this paper:

- **RQ1** (To which extent) can the usage of information expressing activation of links between services improve diagnosis?
- **RQ2** How does topology, i.e. the organization of the invocation links between services, affect diagnosis, and are there general characteristics of topology that improve diagnosis?

The main contributions of this work are an approach and algorithm that can be used to incorporate link invocation information in SFL, and a case study to demonstrate the extent to which invocation link information can improve SFL-based diagnoses.

The remainder of the paper is organized as follows. Section II briefly introduces spectrum-based fault localization and explains how it can be applied in service-based systems. Section III explains the problem in detail, and how it can be addressed. Section IV shows how system simulations can be used to outline the approach and make an initial assessment. Section V evaluates our approach with a real service-based system, and Section VI discusses the results of the experiments performed. Finally, Sections VII and VIII present the related work and conclude the paper.

II. Background

A. Spectrum-based Fault Localization

SFL calculates a diagnosis, i.e. a ranking of potentially faulty components (source code lines, blocks, etc.), from symptoms, i.e. observations about component involvement in system executions, plus pass/fail information about the executions [6]. Component involvement is expressed in the form of so-called block-hit-spectra, producing for each execution a binary coverage value per component [8][9] with covered=1 and not covered=0. This can be derived through coverage tools or monitors. Each system execution leads to a spectrum, represented by a column in a so-called activity matrix. Each spectrum is associated with a binary verdict (pass=0, fail=1) from an oracle. Execution of several transactions adds spectra to the activity matrix. Each line in the activity matrix represents activation of one component over time. The verdicts lead to a binary output vector with pass/fail information. The diagnosis is calculated through applying a similarity coefficient (SC) to each component activation vector and the output vector. The similarity denotes the likelihood of a component being the faulty one, and, therefore, determines its position in the ranking. Any SC may be used; however, the Ochiai SC has been found to work best [10]. Intuitively, SFL works by comparing the different combinations of component involvements in the individual system operations. Components that have not taken part in an activity or are used more in passing activities are less likely faulty in case a failure is detected. This basic SFL approach is illustrated in Table I by means of a simple Ruby program. The program is comprised of components $C_0 - C_{10}$, and it is exercised with 6 system transactions, leading to the corresponding component activations for each transaction $t_1 - t_6$ noted down in the activity matrix. Four transactions fail (output=1); two pass (output=0). The Ochiai SC is calculated for the output vector and each component's activation vector. Finally, the similarity values are brought in a descending order. This results in C_3 being ranked top with 100% likelihood, which represents the location of the fault in this example system (fault marked in bold).

Table I
Illustration of basic SFL

C	Character counter	t_1	t_2	t_3	t_4	t_5	t_6	SC
	def count(string)	[Activity Matrix]						
C_0	let = dig = other = 0	1	1	1	1	1	1	0.87
C_1	string.each_char { |c|	1	1	1	1	1	1	0.87
C_2	if c===/[A-Z]/	1	1	1	1	0	1	0.93
C_3	**let += 2**	1	1	1	1	0	0	1.00
C_4	elsif c===/[a-z]/	1	1	1	1	0	1	0.93
C_5	let += 1	1	1	0	0	0	0	0.71
C_6	elsif c===/[0-9]/	1	1	1	1	0	1	0.93
C_7	dig += 1	0	1	0	1	0	0	0.71
C_8	elsif not c===/[a-zA-Z0-9]/	1	0	1	0	0	1	0.47
C_9	other += 1 }	1	0	1	0	0	1	0.47
C_{10}	return let, dig, other	1	1	1	1	1	1	0.87
	end							
	Output vector (verdicts)	1	1	1	1	0	0	

B. SFL for Service-based Systems

Applying SFL in service-based systems requires the SFL concepts to be adapted to the service context. This has implications in terms of the component granularity, system activation, component coverage and the verdicts.

The service represents the natural component granularity. It is the basic unit that can be restarted, exchanged, or otherwise treated, in case an error is detected. Alternatively, a service operation, which represents a business functionality of a service, may denote a finer level of granularity that is easy to observe.

Activation in service-based systems is not so obvious, and it cannot be done through merely exercising test cases. Because a service instance can serve many application contexts, it will not be exclusively activated from within one application, but from a potentially arbitrary number of other applications operating in other contexts. The application of SFL in a service-based system requires a system execution to be made explicit through a unique transaction ID. This separates the service executions of different application contexts.

Component involvement in transactions is typically measured through coverage tools. However, since there is no single controlling authority that can produce service coverage information, involvement of a service in a transaction must be produced differently. As such, applying SFL in service-based systems requires dedicated monitors, which observe the service communication and associate the services/operations with their corresponding transaction IDs. This can either be done by the services themselves or through modern service frameworks. For example, Apache's Axis2[1], Redhat's JBoss[2], or Ebay's Turmeric[3] come well-equipped with extensive monitoring facilities that can be adopted to producing service involvement information.

A transaction's pass/fail information comes from an oracle. Runtime errors, exceptions, warnings and logs are natural choices for realizing oracles in service-based systems. They are readily available through the platforms managing the communication between services, or they are initiated through the business logic, i.e., the services themselves.

C. Implementation of SFL for Service-based Systems

We base our implementation on Ebay's open source service framework Turmeric. It offers many inbuilt features that support the (online) collection of system data required for applying SFL in service-based systems. This allows the implementation of online monitoring to record the block-hit spectra for SFL with minimum amendments, yielding a slender design. Our prototypical implementation is summarized

[1]http://axis.apache.org

[2]http://www.redhat.com/products/jbossenterprisemiddleware/

[3]https://www.ebayopensource.org/index.php/Turmeric

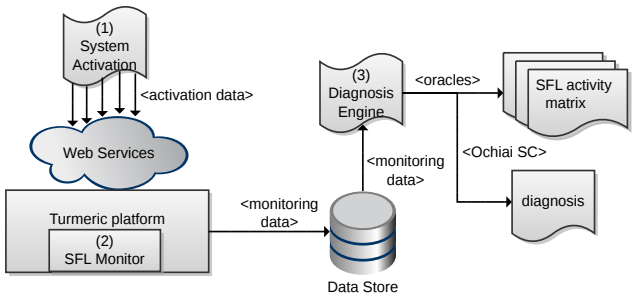

Figure 1. Monitoring and diagnosis architecture based on Turmeric

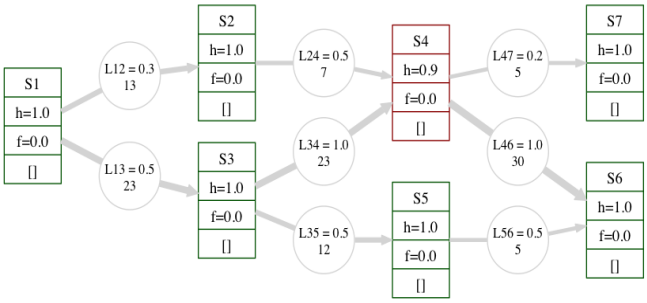

Figure 2. Example topology: illustration of the problem

briefly in the following. A more detailed description is also available [4].

Typically, services would be activated at the application interface through user interaction. However, in our case, system activation is automated through SoapUI[4] combined with JMeter[5] for evaluation purposes. These tools are used to create SOAP messages and execute them automatically, thereby mimicking real user interaction.

Involvement of a service in a transaction is derived from Turmeric's online monitoring facilities [11]. It provides a specific pipeline message-handling mechanism, which can be extended by additional custom-built handlers dedicated to monitoring incoming and outgoing traffic of a service. The monitors associate transactions with the respective service handlers that process the transaction, resulting in service activation information per unique transaction. Because this mechanism is able to distinguish incoming and outgoing messages, it can also be used to produce invocation link activation information, simply by associating an outgoing transaction of one service with an incoming transaction of another service.

Verdicts are also generated based on Turmeric's message handling facility. Dedicated monitors are used to log upcoming exceptions or other noteworthy events and outcomes into a data store for further analysis. Thus, any one of these noteworthy occurrences can be associated with a unique transaction ID and be used as verdict for calculating a diagnosis.

The actual diagnosis is conducted offline in a diagnosis engine. It is designed as a separately operating application that collects the service involvement information and verdicts produced by the oracles. Activities and verdicts are transformed into an activity matrix and an output vector for further calculation of a diagnosis. This implementation is summarized in Fig. 1.

III. Problem Statement and Approach

The topology shown in Fig. 2 and its corresponding activity matrix and diagnosis in Table II illustrate the problem addressed in this paper. This topology is comprised of six healthy services with $h = 1.0$ and one faulty service S_4 with $h = 0.9$, representing low intermittent fault behavior. Intermittency of 0.9 means that if S_4 is invoked, it will fail in 10% of the cases. Failure probability is set to $f = 0.0$ in all services, meaning that once a fault is activated, it will not be detected immediately (i.e., turn into failure), making diagnosis more realistic and difficult. Services S_3, S_4 and S_6 are tightly coupled, indicated through the highest possible invocation probabilities of their respective links between them ($L_{13} = L_{34} = L_{46} = 1.0$). It means when S_3 is invoked, its subsequent tightly linked services S_4 and S_6 will also always be invoked.

Table II
Activity Matrix and Diagnosis: Illustration of the Problem

Nodes	Activity Matix	SC_o
L24	0000001001010000001000000010000000010000010000000	0.436
S6	1010011111010011111101110010101000011000111001000	0.340
L46	1010011111010011111101110010101000011000111001000	0.340
S4	1010011111010011111101110010101000011000111001000	0.340
L12	0000001001010001001100000110100000010000010001100	0.320
S2	0000001001010001001100000110100000010000010001100	0.320
L56	0000000001000000100000010010000000000000010000000	0.258
S1	1111111111111111111111111111111111111111111111111	0.245
L13	1010011111010011110101110010101000001000101001000	0.241
S3	1010011111010011110101110010101000001000101001000	0.241
L34	1010011111010011110101110010101000001000101001000	0.241
L35	0000000011000011100001110010001000000001000010000	0.167
S5	0000000011000011100001110010001000000001000010000	0.167
S7	0000001010000000100001000000000000000000000010000	0.000
L47	0000001010000000100001000000000000000000000010000	0.000
Output	0000000000000000001000000010000000001000000000000	

A diagnosis in which only the activity of the services was considered would lead to S_4 and S_6 being ranked top with $SC_o = 0.34$, leading to an ambiguous result. However, after introducing invocation link information into the calculation of the diagnosis, as demonstrated in Table II, the service S_4 becomes more suspicious, since it is associated with the top ranked invocation link L_{24}. This is reasonable, because all incoming and outgoing invocation links associated with a service represent more precise information about the activity of a service over time. Each invocation link represents a specific path leading into a service or leaving the service. If there are more paths to be observed, this leads to more

[4]http://www.soapui.org
[5]http://jmeter.apache.org

varied activation observations for the service through the different paths, and, therefore, to more accurate information about which path lead to the activation of a fault. In the case of tightly-coupled services, but also when services exhibit intermittent fault behavior, the additional information is beneficial for the diagnosis.

In other words, any of the links associated with a service represent a potentially different path through the service, thereby increasing the observation granularity. This is similar to adding monitors inside the services without touching their implementations, and we expect it can lead to similar results as reported in [7], without having to instrument the service implementation. In fact, this exploits topological information of the system, and it may be regarded as a first step towards combining SFL with model-based diagnosis [12].

The specification shown in Alg. 1 defines the algorithm used to exploit the additional information introduced by the invocation link observations. It takes as input the ranking R of services and invocation links produced by SFL and the topological information A of the system, i.e. which service is associated with a link, and returns a set of potentially faulty services as diagnosis D. If all SC in R are 0.0, there was no observed failure. All services are considered to be healthy. If there was a failure and each item in R with the highest SC is a service, it returns these services as the diagnosis D. Otherwise, it means that some links ranked higher than or equal to services, and we can exploit the invocation link information. In this case, it extracts all links L that are ranking higher than or equal to the top-ranked service, and then it checks which services have the highest number of associations with those links. These services are stored in S. If S only contains one service, i.e. $|S| == 1$, then the algorithm returns D with the service as potentially faulty service. Otherwise, if there are more services with the same highest number of associations, it selects the ones with the highest SC and returns them as diagnosis D.

The algorithm determines the services with the highest number of associations with higher- or equally highly-ranked links. An invocation link ranking higher indicates that it is more likely to activate the fault and cause the failure. Therefore, a service which is associated more with these higher-ranked links is more likely to contain the fault than other services. In other words, services that are more associated with higher-ranked links are more related to the paths traversing those links which were covered when a fault was activated. Since a link cannot be faulty, the service is convicted that participates more in these paths that lead to fault activation.

In a nutshell, components that are more activated in failing transactions are more likely faulty. Invocation links are components that cannot be faulty. Services that are more associated with those assumed faulty links, are more likely faulty.

For example, service S_4 in Fig. 2 has two associations with the higher-ranked links L_{24} and L_{46}, S_6 has one association with the higher-ranked links, i.e. zero associations with L_{24}, and one association with L_{46}. That way, we can say that service S_4 is more suspicious to be faulty than service S_6, because service S_4 is participating more in transactions involving the assumed more likely faulty links L_{24} and L_{46}. Because links cannot be faulty, service S_4 becomes the most likely convict in this example case, which represents a correct diagnosis.

Algorithm 1 $Diagnose(R, A)$

Require: R : Ranking of the services and links produced by SFL;
A : Associations between services and links (Topological information)
Ensure: D : Set of potentially faulty services as diagnosis

1: $T, L, S, D \leftarrow \emptyset$
2: sc_{top} = $getHighestSC(R)$
3: **if** (sc_{top} != 0.0) **then**
4: $T \leftarrow \{i | i.sc == sc_{top}$ and $i \in R\}$
5: **if** ($\forall i \in T$ and i is $service$) **then**
6: $D \leftarrow T$
7: **else**
8: $service_{tr}$ = $getTopRankedService(R)$
9: $L \leftarrow \{l | l.sc \geq service_{tr}.sc$ and l is $link$ and $l \in R\}$
10: $S \leftarrow getServicesWithHighestNumOfAssoc(L, A)$
11: **if** ($|S|$ == 1) **then**
12: $D \leftarrow S$
13: **else**
14: sc_{max} = $getHighestSC(S)$
15: $D \leftarrow \{i | i.sc == sc_{max}$ and $i \in S\}$
16: **end if**
17: **end if**
18: **end if**
19: **return** D

IV. System Simulations

A. SFL Simulator

In order to validate our approach quickly and easily, we performed the initial assessment with our SFL simulator.[6] It provides functions for setting up component topologies, executing the topologies thereby gathering coverage information, and calculating diagnoses. As an example refer to the topology shown in Fig. 2 and its corresponding activity matrix and diagnosis in Table II. These are generated by our simulator.

A topology is created by defining a number of components. Each component is defined by the component name, component health, and failure probability. Health denotes the probability that a component will not produce an error. Failure probability determines the likelihood of a component to issue a failure immediately when a fault is propagated to it or its own fault is activated. Once a fault is activated it is checked in every subsequent component invocation whether it leads to failure. If a failure is detected, the execution is stopped. If all component failure probabilities

[6]https://github.com/SERG-Delft/sfl-simulator

are set to 0.0, the error is detected at the end of the execution. Components in a topology can be connected through defining an invocation link between them with an associated invocation probability. This defines the likelihood that a linked component will be invoked during execution.

Based on the topology with components and invocation links, the simulator can be controlled to perform executions. This requires that one or several entry points (components or links) are activated. Every activation of the topology leads to a particular control flow according to the initially defined probabilities, thereby generating coverage observations for the activity matrix and pass/fail information. These observations are collected and used to calculate a diagnosis ranking.

B. Pilot Simulation

In order to investigate how invocation link activation information influences the diagnosis for a service-based system, we used the SFL simulator to build a trial topology (Fig. 3). It is comprised of 12 components with different incoming and outgoing numbers of invocation links between them. Components C_4, C_5, and C_6 are set to be faulty, and they represent the study subjects on which we focus our interest. From initial experiments, performed for [7], we figured that the number of incoming and outgoing links might be significant for improving diagnosis through adding invocation link information (compare with Fig. 2). This comes from how additional invocation link monitors can separate the specific invocation paths leading into and coming out of components.

The three faulty components shown in Fig. 3 represent three extreme cases, i.e. a component with one incoming link and several outgoing links (C_4), a component with several incoming links and one outgoing link (C_6), and a component with several incoming and outgoing links (C_5). In order to study the effects of invocation link activation information on diagnosis, the topology is executed according to different criteria.

In each experiment, the failure probabilities of the components are varied, i.e. $P_f = 0.0$ or $P_f = 1.0$, representing the probability that a failure can be detected when a fault was triggered. In addition, the invocation probabilities P_i between the concerned (faulty) components and their peers are varied, i.e. high interaction probability $P_i = 0.9$, low interaction probability $P_i = 0.1$. This represents the probability that a component associated with this link is activated. In each experiment, one of the components is set to be faulty with intermittency, i.e. low health $h = 0.1$ and high health $h = 0.9$, and it represents the probability that a faulty component will fail when invoked.

Table III summarizes the results of the experiments performed with these diverse topology setups. Every line in the table represents three experiments comprised of 500 diagnoses each. Every experiment was carried out with a specific topology setup, indicated in the first three columns,

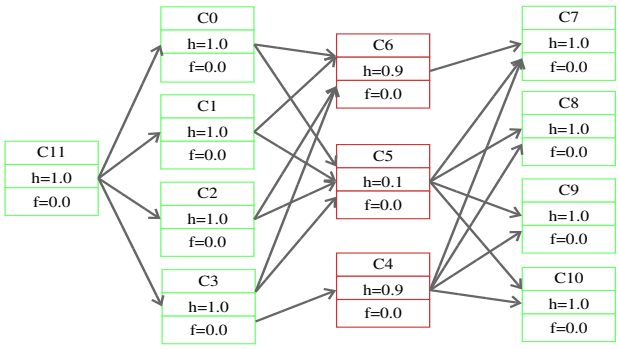

Figure 3. Topology for the pilot simulation

Table III
PILOT SIMULATION (500 ACTIVATIONS)

Topology Setup			C_4		C_5		C_6	
P_f	P_i	h	better	worse	better	worse	better	worse
0.0	0.9	0.9	8.1%	3%	15%	0%	0.4%	0%
0.0	0.1	0.9	24.1%	3.4%	28.6%	2.9%	5.4%	0%
0.0	0.9	0.1	0%	0%	89%	0%	0.2%	0%
0.0	0.1	0.1	17.6%	2.7%	10.7%	0.2%	4.9%	2.2%
1.0	0.9	0.9	0%	47.9%	9.8%	0%	2.6%	0%
1.0	0.1	0.9	0%	3.6%	1.2%	1.2%	1.3%	1.3%
1.0	0.9	0.1	0%	0%	0%	0.2%	0%	0%
1.0	0.1	0.1	0.5%	8.7%	0%	1.7%	0%	0.7%

and with every of the three concerned components, C_4, C_5, or C_6 set to be faulty. For each of the 500 activations, the simulator was set to calculate one diagnosis based on only component activation observations, and another diagnosis based on both component and invocation link activation observations. The result of a diagnosis can be classified as *correct*, *ambiguous* or *incorrect*. A *correct diagnosis* pinpoints the faulty component correctly and uniquely (no duplicate top rankings). An *ambiguous diagnosis* pinpoints the faulty component but includes other healthy components on the same rank (duplicate top rankings). An *incorrect diagnosis* ranks any arbitrary healthy component higher than the faulty component.

Table III shows for each of the concerned components the percentage of how much better or worse the diagnoses become through incorporating invocation link activation information compared with merely using component activation information. The percentage is calculated based on the total number of failed transactions. *Better* means that an initially incorrect or ambiguous diagnosis can be performed correctly, through including invocation link information. *Worse* means that an initially correct diagnosis would become ambiguous or incorrect through including invocation link information.

From Table III, we can see that if the failure probability is low, i.e. $P_f = 0.0$ (top part of the table), using invocation link activation information is more beneficial, in general. All

concerned components show more *better* than *worse* results. Component C_5 scores the highest improvements, which, we believe, is attributable to its high number of incoming *and* outgoing invocation links.

An interesting result that we did not anticipate initially is the poor performance when the failure probability is high, i.e. $P_f = 1.0$. This is shown in the bottom part of Table III. In this case, invocation observation carries not merely useless, but even misleading information. This comes from how the simulator treats failure probability. It stops a transaction if a failure is detected in a component, thereby dismissing all information about its outgoing invocation links. This leads to component C_4 issuing the worst results, because of its low overall number of considered invocation links, i.e. only one incoming link. Because C_5 and C_6 have more incoming links, that can be considered in the diagnosis, their results are not so bad. This suggests that for the sake of diagnosability real systems should have more invocation links between their components/services, and they should be built to recover from failure and continue operation.

Other interesting observations are the effects of health on the calculation of diagnoses. When failure probability is high, i.e. 1.0, and health is low, i.e. 0.1, it means an activation always causes a failure immediately. In this case, C_5 and C_6 are only becoming worse, i.e. invocation link activation information has no improvement at all. In addition, the overall worst case can be observed for component C_4 when failure probability, invocation probability and health probability are all high, i.e. $P_f = 1.0$, $P_i = 0.9$ and $h = 0.9$.

From these simulations, we can conclude that using invocation link observations in SFL is beneficial if the topology is highly interconnected (many invocation links between the services), and if a failure is detected, the system should recover and continue its operation, if possible.

C. Simulation with a Real System

After having established a strong empirical relation between the number of incoming and outgoing invocation link activation observations and the quality of an SFL-based diagnosis, the next step is to assess our approach through simulation with a real system, which represents a more realistic setup compared to the pilot simulation. We use a simulation of our case study system presented in Section V.

The simulated system consists of a number of components, i.e. service interfaces, and invocation links between them. Two of the components that exhibit poor diagnosability in the real system are set to be faulty with low intermittency of $h = 0.8$, all other components are set to be healthy. The two poorly diagnosable components are *ExchangeCurrencyService* and *OrderProcessorService*. In the following, we refer to them as *ECS* and *OPS*, respectively. The invocation probabilities between the components used for simulation are determined experimentally, based on the implementation logic plus the test data used in order to execute the real system. Failure probability in the simulation is set to 0.0, reflecting the behavior of the real system, i.e. faults are not detected immediately.

Table IV
DETERMINING THE NUMBER OF SIMULATION ACTIVATIONS

System Activations	Minimal Number of Failed Activations	Maximal Number of Failed Activations	Deviation
100	62	82	10.00%
500	393	416	4.60%
1000	776	818	4.20%
2000	1585	1614	1.45%
5000	3985	4046	1.22%

The number of simulations is set to a high value, i.e. 2000, in order to create a statistically significant data set. One problem with simulating real systems is that the simulation of service and invocation link activation is completely random, solely based on the predetermined probabilities, whereas, in the real system, invocations follow distinct patterns coming from the system's usage profile. In order to retrieve a meaningful dataset in the simulation, it is therefore essential to generate many activations. Table IV shows how the number of activations leads to a realistic number of failed transactions in the simulation. A low number of activations in the simulation results in high deviation of the number of failed transactions compared to the real system. Only at 2000 activations in the simulator, the deviation in failed transactions compared to the real system becomes acceptably small. Any more activations in the simulation do not improve the deviation from the real system significantly. Hence our choice of 2000 activations for the simulation.

In the simulation, both components, ECS and OPS, are activated with component activation observation enabled, and then with both component and invocation link activation observation enabled. Table V presents the total number of activation failures, how many of the failed activations lead to incorrect (inc), ambiguous (amb) and correct (cor) diagnoses based on two activation criteria, respectively. In both cases, diagnosis improves considerable when invocation activation information is included, i.e. an improvement from 49.5% to 63.4% correct diagnoses for component ECS, and from 24.1% up to 52.6% correct diagnoses for component OPS.

Table VI shows more details about how the inclusion of invocation link activation information makes diagnoses better or worse in the simulations. For service ECS, 235 diagnoses (out of 1616) are better, of which 132 ambiguous diagnoses can be turned into correct diagnoses (Amb→Cor), and also 103 incorrect diagnoses can be turned into correct ones (Inc→Cor). However, 11 correct diagnoses are turned into incorrect diagnoses (Cor→Inc). For service OPS, the improvement is much better. Inclusion of invocation link actviation information improves the diagnoses in 485 cases (out of 1703), 115 ambiguous diagnoses can be resolved, and 370 diagnoses can be corrected. We did not find any worse

Table V
SIMULATION RESULTS FOR 2000 ACTIVATIONS

Comp.	# of Fail.	Component Activation Inc	Amb	Cor	Cor-%	Comp. + Invocation Activation Inc	Amb	Cor	Cor-%
ECS	1616	624	192	800	49.5%	577	15	1024	63.4%
OPS	1703	1165	127	411	24.1%	785	22	896	52.6%

Table VI
DETAILED DISTRIBUTION OF BETTER AND WORSE DIAGNOSES THROUGH INVOCATION COVERAGE

Services	Better Diagnoses Total	Amb→Cor	Inc→Cor	Worse Diagnoses Total	Cor→Amb	Cor→Inc
ECS	235	132	103	11	0	11
OPS	485	115	370	0	0	0

diagnoses for service OPS. In future work, we will analyze these results carefully and try to determine why some cases issue worse results. This may indicate a limitation of our approach in terms of which kind of topology might be misleading diagnosis.

V. CASE STUDY

In order to evaluate our approach more thoroughly, we conducted an experiment on our original case study SFL Stonehenge[7] introduced in [4], and adapted it to the requirements implied by our problem statement. The system was extended to deal with invocation link activation information.

SFL Stonehenge is a service-based system simulating the stock market. It supports users in buying and selling of stocks, checking orders, and performing currency conversion operations for foreign stock acquisition. Figure 4 illustrates the basic service architecture of the system. It is comprised of 10 web services including one *external currency exchange service*, plus a *web application* for user interaction. In addition, it accesses two data stores. The services provide the following operations. *BusinessBasicService* and *BusinessAccountService* provide the functions for user authentication, login, and the user account. *BusinessOPService* and *BusinessStockService* are used for buying and selling stock, checking orders, and compiling market summaries. *QuoteService* and *OrderProcessorService* are used to process the stock orders placed by a user. *ExchangeCurrencyService* and *ExchangeCheckService* are responsible for the currency operations, and the *ConfigurationService* binds all the other services together, and acts like a registry.

A. Conducting the Case Study

The case study system is the same system that we used in the simulations with two faulty services exhibiting poor diagnosability, i.e. *ExchangeCurrencyService* (ECS) and *OrderProcessorService* (OPS). Both services exhibit tight coupling with their peers plus intermittent fault behavior. The goal of the case study is to assess to which extent the inclusion of invocation link activation information can improve their diagnosability.

We applied the PIT mutation tool[8] in order to create 65 faulty service versions, 24 faulty versions of *ECS* and 41 faulty versions of *OPS*. For each of the 65 faulty system versions, we use JMeter[9] to execute 48 web service requests as test scenarios in order to cover all service operations. Upon completion of all transactions for one faulty system version, the diagnosis engine is invoked to parse the monitoring data, identify the failures in the system, and create an activity matrix with an output vector. The monitoring is provided through the Turmeric framework, mentioned in Sect. II-C and detailed in [4]. Turmeric already logs all required transaction information, e.g., the traces of a service invoking other services. In other words, the invocation link activation information is readily available in the existing monitors.

In order to assess to which extent the additional invocation link activation information makes diagnoses better or worse for the two faulty services, we invoked the diagnosis engine twice per execution. First, it creates activity matrices that are only comprised of service interface activation data. Second, it creates activity matrices that include both service interface activation data plus invocation link activation data. The two data sets can then be compared. The whole experiment is designed for the single fault case. We ensure that each of the 65 versions of the system contains only one fault, either in *ECS* or in *OPS*.

B. Case Study Results

Table VII summarizes the case study results. It shows for each of the two faulty services, ECS and OPS, the total number of passed and failed transactions (pass/fail) in the experiment. Some transactions pass, because the faults introduced by the mutations are not triggered. Then, it shows for the failed transactions, the incorrect, ambiguous and correct diagnosis results based on two activation criteria, i.e., for service interface activation information on the left hand side, and for both service interface plus invocation link activation information on the right hand side. The results indicate considerable improvements in diagnoses that are based on service activation information plus invocation link activation. ECS improves from 13.6% up to 86.4% correct diagnoses, and OPS improves from 70.3% up to 91.9% correct diagnoses.

Table VIII shows details on how the diagnoses in the case study become better or worse after including the invocation link activation information. For ECS, 17 diagnoses are improved from incorrect to correct. For OPS, 2 diagnoses are

[7]https://github.com/SERG-Delft/sfl-stonehenge

[8]http://pitest.org/

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

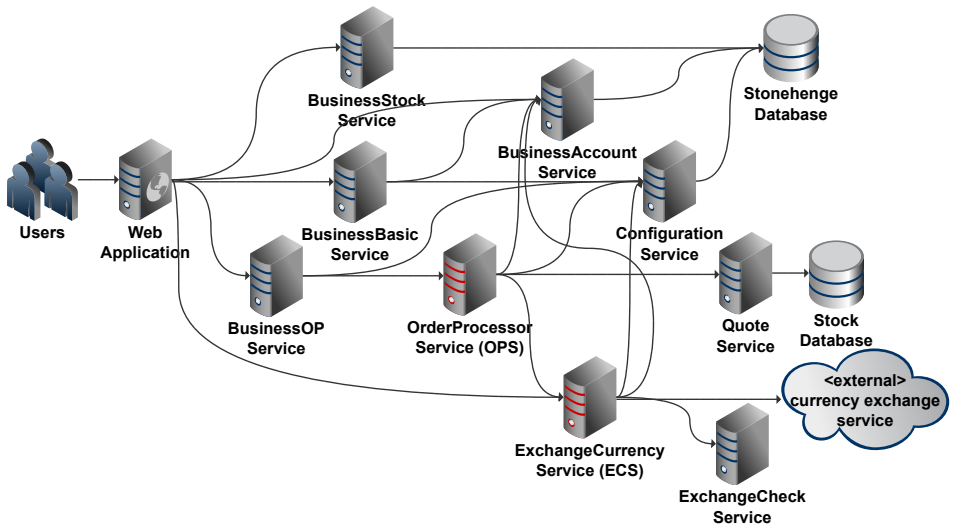

Figure 4. Case study system: SFL stonehenge

Table VII
DIAGNOSIS RESULTS FOR SFL STONEHENGE

Service	Pass	Fail	Serv. Interface Activation Inc	Amb	Cor	Cor-%	Serv. Iface + Link Activation Inc	Amb	Cor	Cor-%
ECS	2	22	19	0	3	13.6%	3	0	19	86.4%
OPS	4	37	9	2	26	70.3%	3	0	34	91.9%

Table VIII
DETAILED RESULTS FOR BETTER AND WORSE DIAGNOSES

Service	Better Diagnoses Total	Amb→Cor	Inc→Cor	Worse Diagnoses Total	Cor→Amb	Cor→Inc
ECS	17	0	17	1	0	1
OPS	8	2	6	0	0	0

improved from ambiguous to correct, 6 are improved from incorrect to correct. OPS does not receive any worse result, while for service ECS, one diagnosis deteriorates, i.e. from correct to incorrect. Careful analysis of this single worse diagnosis leads us to an explanation. The faulty service ECS is not only invoked by other services, but also directly from the user. Since we did not take the invocation link activations between users and services into account, this missing invocation link, which actually always activates the fault, cannot help to improve the diagnsis. This indicates the importance of including all the invocation links of the topology in the calculation of diagnoses. Once this link is added, the incorrect diagnosis can be corrected.

VI. DISCUSSION AND LESSONS LEARNED

In the simulations and the case study we could identify considerable improvements by incorporating invocation link activation information into the calculation of SFL-based diagnoses. Our approach works, because it applies the same rules of the basic SFL that work for component activation information, also to the invocation link activation information. That is, services that participate more in failing transactions are more likely faulty, plus services that are more associated with *links* participating more in failing transactions, are more likely faulty.

A. Revisiting the Research Questions

RQ1 – *The extent to which the usage of invocation link information can improve diagnosis:* Both simulation and case study demonstrate that incorporating invocation link activation information in addition to service interface activation information can significantly improve diagnoses performed by spectrum-based fault localization. In the simulations of the case system, correct diagnoses for service ECS could be improved by around 14%-points through the additional observations, and for service OPS by around 29%-points. Interestingly, the overall improvement in correct diagnoses in the real system is higher than for the simulation, i.e. improvement for ECS by around 73%-points, and for OPS by around 22%-points, when including the additional observations. We believe this much better result in the real service-based system compared to its simulation comes from the fact that the simulator generates completely random invocation combinations beteen services, whereas, in the real system, service invocations follow less dynamic combinations, according the system's typical usage patterns. In other words, in the real system, much less different paths are exercised leading to a few prominent invocation patterns, whereas the simulation produces many more different service invocation combinations. This is an

interesting observation which will be further researched in the future, i.e. can the combination of usage profile plus its associated invocation patterns be used as information in order to improve diagnosis?

When we compare the current results with the results of our earlier work presented in [7], in which we instrument the services themselves with additional observation points, it becomes apparent that including information about the invocation links between services is inferior (about 10%–15% worse). For the same case study system, this other approach could achieve 100% correct and unambiguous service diagnoses [7]. However, the huge advantage of invocation link observations is that they can be retrieved through the service platform, whereas, for our earlier approach, the service implementations had to be amended, which is not always possible.

RQ2 – *The effects of topology on the quality of the diagnosis:* We refer to topology as the organization of the activation observation points in the service-based system, i.e. the service and link coverage monitors. Through the simulations and the conduction of the case study, we can demonstrate that the topology of the observation points has, indeed, an effect on the quality of the diagnoses calculated by SFL. In the case study, we compared the number of correctly performed diagnoses for merely monitoring service interface activation vs. interface plus invocation link activation, and observed considerable improvements. The improvements come from how the additional invocation link activation information helps split the topology into finer grained and separable units thereby helping to discriminate better the various service invocation paths. The simulations suggest that a high number of incoming and outgoing invocation links is beneficial, however, through the case study we found that any more than one incoming and outgoing link is improving the results. We believe, it is not so much the total number of incoming and outgoing links which makes a difference, but how those links lead to more diverse activation of the execution paths within a service, thereby exploiting information similar to that generated by service-internal monitors, as demonstrated in [7]. These effects will be studied in future work.

As general guideline for determining the monitoring topology of a service-based system, we propose to

- split the monitoring of services into finer grained units representing better the service's different functions.
- exploit additional information better that is suitable to separate the execution paths of the transactions flowing through the services.

VII. Related work

Chen et al. present *Pinpoint* [13], a similar diagnosis approach plus a tool using similarity coefficients in order to infer a diagnosis from system activation and component involvement. However, even though their title suggests otherwise, they do not address the specific issues of diagnosing services, i.e. the problems of inter-service diagnosis, and the fact that services are used in different contexts. Yan, et al. [14], [15], propose a model-based approach to diagnose orchestrated Web service processes. Modeling is done through discrete event systems, which imposes a heavy burden on the user of the technique. Zhang et al. [16], [17] describe approaches for diagnosing quality-of-service problems in service-oriented architectures. Mayer and colleagues [18], [19], describe a similar diagnosis approach that is based on analyzing execution traces of failed transactions.

Wong et al. [20] discuss a number of code coverage-based heuristics to be used in fault localization. Grosclaude describes a model-based monitoring approach for diagnosing component-based systems, and suggests to use transactions IDs in order to associate messages sent between components [21]. This is also proposed by [13], and we see it as a standard approach to determine which service takes part in which system transaction. Chatzigiannakis and Papavassiliou [22] use principle component analysis in order to identify faulty nodes in sensor networks.

Heward et al. in [23] describe an algorithm for optimization of monitoring configurations for web services. They use their optimization algorithm in order to reduce the monitoring overhead in a service-based system, something that would also benefit our proposed techniques.

Li et al. [24] describe an approach for control flow analysis and coverage for web services. Thye use their approach for testing purposes. Bartolini et al. [25] propose service coverage criteria that are based on service invocation monitoring. Their approch is also used for testing. Baresi et al. [26] introduce smart monitors for composed services, and Moser et al. [27] and Spanoudakis et al. [28] describe non-intrusive monitors for service-based systems.

VIII. Conclusion and Future work

In this paper, we demonstrated how to incorporate invocation link activation information in Spectrum-based Fault Localization techniques to diagnose a service system. We devised an algorithm to deduce the faulty service based on the association number of services and higher-ranked invocation links. The pilot simulation revealed that the invocation links together with our algorithm can improve the diagnosis for component with diverse interactions when the fault does not cause failure immediately. Experiments on both simulation and real case study system further confirmed that the invocation link information can significantly improve the diagnosis under more realistic setting-up.

In the future, we are going to explore for which type of a topology can the invocation link information used for better diagnosis, and which other context information, such as systems usage profile, can be also used for diagnosis.

ACKNOWLEDGMENT

We would like to acknowledge NWO for sponsoring this research through the Jacquard ScaleItUp project. Also many thanks to our industrial partners Adyen and Exact.

REFERENCES

[1] G. Canfora and M. Di Penta, "Service-oriented architectures testing: A survey," in *Software Engineering*, ser. LNCS. Springer, 2009, vol. 5413, pp. 78–105.

[2] K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay, and M. Munro, "Service-based software: the future for flexible software," in *Proc. Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2000, pp. 214–221.

[3] G. Lewis and D. Smith, "Service-oriented architecture and its implications for software maintenance and evolution," in *Frontiers of Software Maintenance (FOSM)*. IEEE, 2008, pp. 1–10.

[4] C. Chen, H.-G. Gross, and A. Zaidman, "Spectrum-based fault diagnosis for service-oriented software systems," in *Proc. of the Int'l Conf. on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2012.

[5] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. van Gemund, "A practical evaluation of spectrum-based fault localization," *Journal of Systems and Software*, vol. 82, no. 11, pp. 1780–1792, 2009.

[6] A. Gonzalez-Sanchez, R. Abreu, H.-G. Gross, and A. J. van Gemund, "Spectrum-based sequential diagnosis," in *Proc. Int'l Conf. on Artificial Intelligence (AAAI)*. AAAI Press, 2011, pp. 189–196.

[7] C. Chen, H.-G. Gross, and A. Zaidman, "Improving service diagnosis through increased monitoring granularity," in *7th Intl Conf. on Software Security and Reliability*, Washington, DC, June, 18–20 2013, p. to appear.

[8] T. Reps, T. Ball, M. Das, and J. Larus, "The use of program profiling for software maintenance with applications to the year 2000 problem," in *European Softw. Engineering Conf. & Symp. on Foundations of Softw. Engineering (ESEC/FSE)*, ser. LNCS. Springer, 1997, vol. 1301, pp. 432–449.

[9] P. Zoeteweij, R. Abreu, R. Golsteijn, and A. J. van Gemund, "Diagnosis of embedded software using program spectra," in *Proc. Int'l Conf. and Workshops on Engineering of Computer-Based Systems (ECBS)*. IEEE, 2007, pp. 213–220.

[10] R. Abreu, P. Zoeteweij, and A. J. van Gemund, "An evaluation of similarity coefficients for software fault localization," in *Proc. Int'l Symp. on Dependable Computing (PRDC)*. IEEE, 2006, pp. 39–46.

[11] C. Chen, A. Zaidman, and H.-G. Gross, "A framework-based runtime monitoring approach for service-oriented software systems," in *Int'l Workshop on Quality Assurance for Service-Based Applications (QASBA)*. ACM, 2011, pp. 17–20.

[12] J. de Kleer and J. Kurien, "Fundamentals of model-based diagnosis," in *Fault Detection, Supervision and Safety of Technical Processes*. IFAC, 2003, pp. 25–36.

[13] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: problem determination in large, dynamic internet services," in *Prod. Int'l Conf on Dependable Systems and Networks (DSN)*. IEEE, 2002, pp. 595–604.

[14] Y. Yan and P. Dague, "Modeling and diagnosing orchestratedweb service processes," in *Proc. Int'l Conf on Web Services (ICWS)*. IEEE, 2007, pp. 51–59.

[15] Y. Yan, P. Dague, Y. Pencole, and M.-O. Cordier, "A model-based approach for diagnosing fault in web service processes," *International Journal of Web Services Research (IJWSR)*, vol. 6, no. 1, 2009.

[16] J. Zhang, Y. Chang, and K.-J. Lin, "A dependency matrix based framework for QoS diagnosis in SOA," in *Proc. Int'l Conf on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2009, pp. 1–8.

[17] J. Zhang, Z. Huang, and K. Lin, "A hybrid diagnosis approach for QoS management in service-oriented architecture," in *Int'l Conf. on Web Services (ICWS)*. IEEE, 2012, pp. 82–89.

[18] W. Mayer, G. Friedrich, and M. Stumptner, "Diagnosis of service failures by trace analysis with partial knowledge," in *Service-Oriented Computing*, ser. LNCS. Springer, 2010, vol. 6470, pp. 334–349.

[19] ——, "On computing correct processes and repairs using partial behavioral models," in *20th European Conference on Artificial Intelligence (ECAI)*, 2012, pp. 582–587.

[20] W. E. Wong, V. Debroy, and B. Choi, "A family of code coverage-based heuristics for effective fault localization," *Journal of Systems and Software*, vol. 83, no. 2, pp. 188–208, 2010.

[21] I. Grosclaude, "Model-based monitoring of component-based software systems," in *Int'l Workshop on Principles of Diagnosis*, 2004, pp. 155–160.

[22] V. Chatzigiannakis and S. Papavassiliou, "Diagnosing anomalies and identifying faulty nodes in sensor networks," *Sensors Journal, IEEE*, vol. 7, no. 5, pp. 637 –645, May 2007.

[23] G. Heward, J. Han, J.-G. Schneider, and S. Versteeg, "Runtime management and optimization of web service monitoring systems," in *Proc. Int'l Conf on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2011, pp. 1–6.

[24] L. Li, W. Chou, and W. Guo, "Control flow analysis and coverage driven testing for web services," in *Int'l Conf. on Web Services (ICWS)*. IEEE, 2008, pp. 473–480.

[25] C. Bartolini, A. Bertolino, and E. Marchetti, "Introducing service-oriented coverage testing," in *Automated Software Engineering - Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on*, 2008, pp. 57–64.

[26] L. Baresi, C. Ghezzi, and S. Guinea, "Smart monitors for composed services," in *Proc. Int'l Conf. on Service-Oriented Computing (ICSOC)*. ACM, 2004, pp. 193–202.

[27] O. Moser, F. Rosenberg, and S. Dustdar, "Non-intrusive monitoring and service adaptation for ws-bpel," in *Proc. Int'l Conf. on World Wide Web (WWW)*. ACM, 2008, pp. 815–824.

[28] G. Spanoudakis and K. Mahbub, "Non-intrusive monitoring of service-based systems," *International Journal of Cooperative Information Systems*, vol. 15, no. 03, pp. 325–358, 2006.