Finding Faulty Components in a Dynamic Distributed System at Runtime

Joël van den Berg
Finding Faulty Components in a Dynamic Distributed System at Runtime

THESIS

submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Joël van den Berg
born in Monster, the Netherlands

TU Delft
Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl
Finding Faulty Components in a Dynamic Distributed System at Runtime

Author: Joël van den Berg
Student id: 1322982
Email: j.j.j.vandenberg@student.tudelft.nl

Abstract

This document describes the research performed on fault isolation in dynamic distributed systems at runtime. An existing Spectrum-based Multiple Fault Localization approach is used as the basis for fault isolation, but is adapted and optimized so it can be used for online diagnosis. The result is an algorithm, coined AIMBACH, which finds the combination of components that can explain the observed failures and orders these combinations by the likelihood that each combination explains the observed failures. The AIMBACH algorithm is implemented in a Service Oriented Architecture. This architectural methodology is implemented a lot in businesses because of its properties. A transaction, which is used by AIMBACH as a spectrum, is defined by the operations of services that were invoked due to a request coming into the system. The information, which is required to define the transaction, is obtained from the system at runtime. The implementation adds little data, but significant time overhead. Based on the accuracy, the implementation outperforms all the Single Fault Localization techniques and approaches the performance of the Spectrum-based Multiple Fault Localization approach that the AIMBACH algorithm was based on.

Thesis Committee:

Chair: Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor: H.G. Gross, Faculty EEMCS, TU Delft
Company supervisor: Maikel Lobbezoo, Adyen
Committee Member: Dr.ir. A.J.H. Hidders, Faculty EEMCS, TU Delft
Preface

This thesis is a result of research performed on Service Oriented Architectures and the problems of isolating faults in such architectures. It is written in partial fulfillment of the requirements for the degree of Master of Science in Computer Science at Delft University of Technology. The research was conducted from September 2011 until March 2012 at Delft University of Technology and Adyen.

I would like to thank Hans-Gerhard (Gerd) Gross for helping me through the process of writing this thesis and for coming up with ideas on how to make progress. I also want to thank Adyen for giving me insight in their business to give me a better idea of the practical requirements of a solution. Furthermore, I would like to thank Tiago Espinha for introducing me to Turmeric SOA and providing the most recent papers of his research. I further want to thank Cuiting Chen and Tiago Espinha for providing a use case platform to test the implementation of AIMBACH. Another thanks goes out to Rui Maranhao, who helped me to understand his original Spectrum-based Multiple Fault Localization algorithm and motivated me to develop the AIM and BACH algorithms.

A special thanks goes out to family and friends for listening to the problems that I faced during the research and for having me explain the subject of my research several times, which helped me to communicate the problem and to keep the story comprehensible.

Finally I would like to thank the other students that came and went in the lab room on the 8th floor for their tips and company during coffee breaks.

Joël van den Berg
Delft, the Netherlands
May 30, 2012
Contents

Preface iii

Contents v

1 Introduction 1

2 Devising an Incremental Spectrum-based Multiple Fault Localization Algorithm 5
   2.1 Spectrum-based Fault Localization Based on C Code ...................... 6
   2.2 Applying Spectrum-based Fault Localization in the online case .......... 8
   2.3 Spectrum-based Multiple Fault Localization ............................ 10
   2.4 An incremental implementation of the Spectrum-based Multiple Fault Localization technique ........................................... 13
   2.5 Summary ........................................................................ 17

3 Obtaining the Required Information at Runtime for AIMBACH from Systems Implementing Service Oriented Architectures 19
   3.1 Service Oriented Architecture ............................................. 19
   3.2 Diagnosis in a Service Oriented Architecture ............................ 23
   3.3 Defining the required data for AIMBACH ................................. 24
   3.4 Obtaining the process id .................................................... 25
   3.5 Defining a transaction ...................................................... 27
   3.6 Summary ........................................................................ 28

4 Description of the Controlled Experiment 29
   4.1 Obtaining the service involvement information from a Service Oriented Architecture ...................................................... 30
   4.2 Implementing the Spectrum-based Multiple Fault Localization Algorithm ................................................................. 32
   4.3 Combining the obtained information with online Spectrum-based Multiple Fault Localization ..................................................... 33
   4.4 Implementing a Service Oriented Architecture with AIMBACH .......... 33
## CONTENTS

4.5 Simulation of a Service Oriented System .................................................. 35
4.6 Implementing AIMBACH in an existing Service Oriented Architecture ........ 37
4.7 Mutation testing ....................................................................................... 37
4.8 Summary ................................................................................................. 38

5 Evaluation .................................................................................................. 41
  5.1 Barinel versus AIM and BACH to order the MHSs ................................. 41
  5.2 Evaluating Time Overhead in the Spicy Stonehenge Application ............ 43
  5.3 Evaluating Data Overhead in the Spicy Stonehenge Application .......... 45
  5.4 Determining the accuracy of the method with Spicy Stonehenge .......... 46
  5.5 Determining the accuracy of the method with the simulator ................. 49
  5.6 Comparing different Fault Localization methods ................................... 53
  5.7 Discussion ............................................................................................. 55
  5.8 Summary ............................................................................................... 56

6 Related Work .............................................................................................. 57
  6.1 Monitoring of systems ........................................................................... 57
  6.2 Failure and fault detection ...................................................................... 59

7 Conclusions and Future Work ..................................................................... 61
  7.1 Contributions ........................................................................................ 61
  7.2 Conclusions .......................................................................................... 62
  7.3 Future work .......................................................................................... 63

Bibliography ..................................................................................................... 65
Chapter 1

Introduction

In order to understand what fault diagnosis is all about, it is important to determine the definitions of fault, error and failures in a system. Faults are human made mistakes that can lead to an error if the fault becomes part of the execution in a system. This error can later in the process lead to a failure, which can be identified as a deviation from the normal execution of the program.

Faults occur in many processes on a daily basis. While most of these faults are tolerable, these tolerable faults can develop into more serious faults [37]. These serious faults are failures and can be life threatening or dangerous to the environment or ecology. In order to create a reliable system, it needs to diagnose the tolerable fault before it can become a failure.

The process of fault diagnosis is separated in two steps: fault detection and fault isolation. The fault detection phase determines whether something has gone wrong. The next phase, fault isolation, locates the source of this wrong behavior. If the fault isolation process is not robust, a fault may be located incorrectly, which makes the fault detection stage worthless. On the other hand, if both the detection and isolation processes for a fault are robust, then these processes can be used for self-healing measures [37].

This thesis focuses on the process of fault isolation in dynamic distributed systems. In order to identify the difficulties of locating faults in dynamic distributed systems, it is required to know the properties that make a system distributed and dynamic.

Distributed systems are composed of multiple independent subsystems that appear to its user as a single coherent system [44]. These kinds of systems have emerged to increase the computational power of the whole system by distributing resources and necessary computations amongst its subsystems. The subsystems are autonomous and the resources on the subsystems may not be accessible by other subsystems.

Dynamic systems are systems that can be changed at runtime. This change can be the addition, removal or replacement of components; or altering connectors to achieve new behavior [35]. Because of the possible changes, the exact communication paths can only be known at runtime. And since it cannot be assumed that an autonomous subsystem knows about the runtime environment of the other subsystem, fault isolation may become the tedious task of looking at the behavior of each subsystem just before the occurrence of the failure.
1. **Introduction**

As dynamic distributed systems grow in size, the time required for fault isolation may increase to a point at which it is infeasible to perform fault isolation manually. However, businesses are still eager to adopt dynamic distributed architectures for creating mission-critical systems, because of the costs and risk of shutting down such systems for maintenance [34]. Being able to adapt a system at runtime means that the system can stay online while the components receive the necessary updates. Examples of mission-critical systems are police emergency systems, nuclear safety systems and security systems. These systems need to be up and running 24/7 and need to be fault tolerant. Another reason for businesses to adopt dynamic distributed architectures is that the needs of users imminently change [32].

A dynamic distributed architectural pattern that is being advocated in the industry as the next evolutionary step, is the Service Oriented Architecture (SOA) pattern [9]. Architectures derived from this pattern are characterized by loosely coupled (distributed) components that dynamically bind to each other. It is, therefore, easy to insert, remove and update components in the system.

Because the systems can be easily changed, errors can also be easily introduced into a running distributed system. These errors become apparent either because a component is faulty or because of compatibility issues between components. Take for example a security component that is responsible for preventing water from boiling, thus keeping it below 100 degrees Celsius or 212 degrees Fahrenheit. When this component requests the thermometer what temperature it is measuring, it returns the value of 120. The thermometer returns the temperature in Celsius, but due to incompatibilities, the security component expects Fahrenheit and keeps the heating source on, because it thinks the water is not yet boiling.

However, because of the costs and risk involved, it is not possible to take mission-critical systems offline to perform fault diagnosis either. Duplicating mission-critical systems is also not possible because of their size and complexity and because it is impossible to simulate the traffic that would go through a mission-critical system. This means that diagnosis needs to be performed in parallel with the running system.

Performing fault diagnosis parallel to the running system is also referred to as online diagnosis. While offline diagnosis requires the system to be offline in order to detect and isolate faults, online diagnosis can detect and isolate faults while the system performs its regular operations. This means, however, that the information that is required to perform the fault diagnosis also needs to be obtained from the system at runtime. This can lead to data and time overhead on the regular operations of the system.

Therefore, the goal of the project of this thesis project is to develop and evaluate a fault isolation technique, which can be efficiently applied in a dynamic distributed system at runtime. The data overhead should be kept low, as this may cause bottlenecks in some parts of the architecture. Also, the time overhead should be kept low. This is especially important in mission-critical systems, where delay may prevent the collection of important data that needs to be acted upon. Finally, because robust fault isolation is desired, the accuracy should be very high.

One fault isolation technique is the Spectrum-based Fault Localization (SFL). The SFL approach suggested by Abreu et al. [3] analyzes the execution of a program by dividing it in components that may or may not be involved in the execution of the program on some input. This information is combined with information about whether the execution failed or not.
The combination of this information for an execution is called a spectrum and is therefore a representation of an execution of the program on some input. After several executions, several spectra have been collected and are put in a matrix, which then consists of several component vectors and an error vector. A component vector describes in which execution the component was and was not involved. Each component vector is compared to the error vector, which indicates which execution failed, by using the Ochiai similarity coefficient. The component with the highest Ochiai coefficient is the most likely faulty component.

The Ochiai value is calculated by counting three possible observations in an execution of the program for a component and a failure: (1) the component is part of the execution and no failure occurred, (2) the component is part of the execution and a failure occurred, (3) the component is not part of the execution and a failure occurred. This information can be updated every time the program is executed by increasing the counter for either observation.

Since the Ochiai value in the SFL approach by Abreu et al. [3] can be updated incrementally it has been found to be a potential candidate for fault isolation at runtime by Piel et al. [38]. However, there are two issues that still need to be tackled.

The first issue is determining the start and end point of an execution at runtime, which is important for defining a spectrum. In the SFL approach by Abreu et al. [3] the start of an execution is determined by the moment at which the program starts running. The end of the execution is determined by the moment the program returns a response and stops running. All the code blocks that were affected between the start and the end of an execution are involved in the execution and are, therefore, part of the spectrum. Because the information is collected after the program has finished, this approach corresponds to offline diagnosis. However, in the situation described by Piel et al. [38] the business components are always in a running state, either waiting for messages from other business components or performing calculations. Fault diagnosis therefore needs to be performed online. The business components interact with each other by exchanging input messages. Once a business component receives a message, it is said to be involved in the current spectrum. Monitors are added and are executed along with the business logic. Monitors also observe the business logic and can, therefore, detect the occurrence of a failure. These monitors are activated by receiving a message from the components that are being monitored. Upon activation of a monitor a spectrum is created by determining whether a failure occurred and combining this information with the business components that were involved in the current spectrum.

However, since interaction can happen simultaneously, all the components might become involved in the current spectrum before a monitor is activated. This would result in a fault isolation accuracy that is as bad as just guessing which component is faulty [38]. Therefore, a better solution needs to be found for defining the start and end of an execution when performing online diagnosis.

The second issue is that the SFL method only finds single faulty components. Updates to a Service Oriented System may come in batches of components and therefore might introduce multiple faults. Abreu et al. [2] also developed a Spectrum-based Multiple Fault Localization (SMFL) technique and proved that this method has a higher accuracy in the situation where multiple faults can occur. The SMFL technique, however, does not use the Ochiai coefficient, but a Bayesian approach. As mentioned before, the Ochiai coefficients
can be calculated incrementally and can, therefore, be used in an online approach [24]. The Bayesian approach, on the other hand, performs the analysis on the whole matrix. This means that every time a new execution occurs, the recompilation of an ever increasing matrix is required.

In summary, there are several problems that will have to be tackled during this thesis. The first problem is finding an algorithm that can be used at runtime for the isolation of multiple faults. The second problem is determining the information that is required from the system at runtime and determining the way that this information is obtained. The last problem is determining the start and end of an execution in a dynamic distributed system at runtime. In order to tackle these problems, this thesis will answer the following research questions.

<table>
<thead>
<tr>
<th>Research Question 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can an incremental Spectrum-based Multiple Fault Localization algorithm be devised?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Research Question 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>What information is required at runtime to be able to perform the adapted method?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Research Question 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>How is this information retrieved from a dynamic distributed system at runtime?</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Research Question 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>How can the beginning and end of an execution be determined in dynamic distributed systems at runtime?</td>
</tr>
</tbody>
</table>

One of the contribution of this thesis is a method for the isolation of faults in a dynamic distributed system at runtime. The method consists of two algorithms, the first of which incrementally finds Minimum-size Hitting sets, coined AIM. The second algorithm, coined BACH, uses the observations of executions and failures to adapt the so-called BACH values to order the found Minimum-size Hitting Sets upon. Another contribution of the thesis is the definition of a transaction in a dynamic distributed system, which can be compared to an execution in a program. The final contribution is the definition of the required data for performing the AIM and BACH algorithms. Refer to chapter 7 for more details on the contributions.

The remainder of this document is outlined as follows. In chapter 2 the Spectrum-based Fault Localization methods will be investigated and adapted. After this chapter, a more in depth analysis of the domain and the required data will be performed in chapter 3. The research questions are by then answered and the findings will be used in a controlled experiment which is described in chapter 4. Then in chapter 5 an evaluation will be performed with the controlled experiment. After this, a summary will be made of some of the related work in this subject field in chapter 6. Finally, in chapter 7 the results and process of the thesis will be discussed and a conclusion as well as suggestions for future work will be given.
Chapter 2

Devising an Incremental Spectrum-based Multiple Fault Localization Algorithm

In this chapter fault isolation will be discussed. Fault isolation is an essential part in the diagnosis of any system and is usually preceded the fault detection phase. There are many methods for fault diagnosis suggested in literature, which can be broadly divided into model-based techniques, knowledge-based methodologies and empirical techniques [37]. Abreu et al. [3] add another type of method, which is Spectrum-based Fault Localization (SFL).

The SFL technique makes use of a collection of program spectra. A program spectrum is a finite, easily obtainable characterization of a program execution on a dataset [39]. In the case of the SFL technique by Abrue et al. [3], a spectrum consists of the parts of the program that were involved in a program execution.

The result of each program execution is compared to the expected outcome for the execution. If the expected outcome deviates from the result of the execution, the program execution is said to have failed. This information is added to the program spectrum for each execution. When the diagnosis starts, this information is translated into a matrix with component vectors and an error vector. The component vectors indicate whether a component was involved in a spectrum and the error vector contains values for whether the program execution for a spectrum has failed.

Each component vector is then compared to the error vector using a similarity coefficient. A similarity coefficient tells how similar the component vector is to the error vector, which means that the higher the similarity coefficient becomes, the more likely it is that the component is the source of the failed executions. Abreu et al. [3] use a similarity coefficient called Ochiai, because they found that this coefficient results in the highest accuracy.

According to Piel et al. [38], SFL is the most light-weight method in the field of fault isolation. This is because using similarity coefficients, like Ochiai, have an ultra-low computational complexity and can be incrementally computed. Because similarity coefficients can be incrementally computed, they can be used in an online approach [24].

However, SFL does not consider the possibility of having multiple faulty components
2. Devising an Incremental Spectrum-based Multiple Fault Localization Algorithm

in the system. The Spectrum-based Multiple Fault Localization [2] (SMFL) method does take the possibility of multiple faulty components in account and using this method instead of SFL results in a higher fault isolation accuracy in situations where multiple faulty components may be present.

The SMFL technique, however, does not benefit from the same properties as the SFL technique. In contrast with SFL, SMFL by Abreu et al. [2] does not use similarity coefficients. Instead, it uses SFL for modeling the execution of a program in combination with model-based techniques to generate and order the possible faulty components. The SMFL approach created by Abreu et al. is called Barinel and has been the starting point for the creation of an incremental SMFL.

This new incremental SMFL approach is the main contribution of this thesis and like Barinel, it uses spectra for modeling the execution of a program. But, unlike Barinel, it uses an incremental approach for generating combinations of possible faulty components, which are called diagnosis sets. In order to be able to order these diagnosis sets, the formula for calculating the Ochiai value for a component is extended to calculate a value for a diagnosis set. This formula is used in the extended Bayes’ rule to calculate the final value on which to order the diagnosis sets. With the adaptation of the Ochiai calculation and the extended Bayes’ rule, the values can be calculated incrementally.

The incremental approach to generate the diagnosis sets is called Adaptive Incremental Minimum-size hitting sets (AIM). A hitting set is a set containing a number of components. For each spectrum that describes a failed execution, a hitting set contains at least one component that was involved in that execution. The Minimum-size Hitting Set (MHS) is the smallest possible hitting set. However, an MHS does not necessarily isolate the faulty components correctly. So instead of taking the MHS, the collection of all the possible hitting sets are taken, with the condition that no hitting set in the collection contains all the elements of a smaller hitting set in the collection. This results in a collection of hitting sets with the smallest possible size.

The remainder of this chapter will describe the implementation of the discussed techniques to show why the incremental SMFL approach is required for fault isolation at runtime. This chapter is concluded with a description of an implementation of the new incremental SMFL technique.

2.1 Spectrum-based Fault Localization Based on C Code

The SFL approach by Abreu et al. [3] is based on C code. The components in their SFL technique are blocks of a function in C. An example of such a function can be seen in program 2.1.

For each input for program 2.1, the execution of the algorithm will involve certain blocks of code. The combination of the code blocks that are involved vary depending on the input for the program. Keeping record of which block of code is involved in an execution is called a spectrum. To this spectrum a simple flag, which represents whether or not an error occurred during the execution, is added. To find out whether an error occurred, the result of the program is compared to the expected outcome. If the result and the expected outcome
Program 2.1 An example of a faulty C program divided in blocks to use in the SFL method.

```c
void RationalSort(int n, int *num, int *den){
    /* block 1 */
    int i,j,temp;
    for(i=n-1; i>=0; i--){
        /* block 2 */
        for(j=0; j<i; j++){
            /* block 3 */
            if(RationalGT(num[j], den[j], num[j+1], den[j+1])) {
                /* block 4 */
                /* Bug: forgot to swap denominators */
                temp = num[j];
                num[j] = num[j+1];
                num[j+1] = temp;
            }
        }
    }
}
```

Figure 2.1: An illustration of how spectra and errors are combined as the input for SFL.

are equal, the execution has passed. If, on the other hand, the result and the expected outcome differ, then the execution has failed.

Multiple spectra make up a matrix, which is called the activity matrix and can be seen in figure 2.1. So, the activity matrix describes several executions of the program. Each column of the activity matrix is called a component’s coverage vector or, in short, component vector. A component vector contains the information that tells in which execution of the program the code block was involved.

The pass and fail information of the executions make up the error vector. The error vector, therefore, contains the information about which execution failed and which execution succeeded. The activity matrix in combination with the error vector are the input for SFL [3].

Once several spectra are known with corresponding errors, the Ochiai coefficients can be calculated for each of the N components. The Ochiai coefficient of a component tells
how similar its coverage vector is to the error vector. The more similar the two vectors are, the more likely it is that the component belonging to the coverage vector is the actual faulty component. Besides Ochiai, there are other similarity coefficients. However, Abreu et al. [3] have proven that the Ochiai coefficient returns the best diagnostic accuracy and will, therefore, be used throughout this thesis.

An example of how the Ochiai coefficient correctly identifies the faulty component can be seen in table 2.1 where \( S_O \) is the Ochiai coefficient calculated by formula 2.1. In this formula, \( a_{pq}(j) \) is the number of times where component \( j \) has value \( p \) in a spectrum and the error for this spectrum is \( q \). The definition of \( a_{pq}(j) \) can be written as the formula:

\[
    a_{pq}(j) = |\{i | x_{ij} = p \land e_i = q\}|
\]

These values for a component for each possible combination of \( p \) and \( q \) will be called the A-values.

\[
    S_O(j) = \frac{a_{11}(j)}{\sqrt{(a_{11}(j) + a_{01}(j)) \ast (a_{11}(j) + a_{10}(j))}}
\]

Formula 2.1: The formula for calculating the Ochiai coefficient for component \( j \).

In table 2.1, code block 4 is correctly identified as the faulty component, because it has the highest value for \( S_O \).

### 2.2 Applying Spectrum-based Fault Localization in the online case

The problem with the approach by Abreu et al. [3] is that it performs fault isolation offline. An input is given to the program and when it returns a response, the code blocks that were involved in the execution can be added to the spectrum and the result can be compared to the expected outcome. This information about the execution is used to make a spectrum in the activity matrix. After several of such executions are collected, the analysis is performed and the program is no longer running.

However, to be able to locate faults at runtime an online approach is needed. A description of online problems is given by Irani et al. [24]. An online problem uses an approach
Applying Spectrum-based Fault Localization in the online case

which supplies data to an algorithm incrementally, to which the algorithm also returns the
output incrementally. The downside of such an online approach is that the algorithm returns
output based on incomplete input. The algorithm can only hope to approximate the optimal
solution for which all the input is known beforehand.

A research on the applicability of SFL in an online case is performed by Piel et al. [38].
Here, they want to use SFL in combination with automatic health monitoring to be able
to identify faulty components in a running system. SFL is the most suitable technique for
runtime diagnosis because of its low computational complexity [38]. The computational
complexity is low due to the fact that only the \( A\)-values have to be updated for each compo-
nent after an execution. This means that the history is not important for SFL to work and the
algorithm does not suffer from having to compile increasingly larger matrices of spectrum
data.

But the problem of online SFL is defining the spectra. Instead of using SFL to diagnose
functions, the situation described by Piel et al. [38] uses SFL to diagnose business compo-
nents. While the C program under investigation by Abreu et al. [3] is considered finished
after it has returned a result, a system over which fault diagnosis has to be performed at
runtime is never in a finished state. Piel et al. [38] therefore investigated several solutions
for the problem of defining spectra at runtime.

Piel et al. [38] attempt to solve this problem by adding several monitors to the archi-
tecture. These monitors can be activated by the business components that they monitor.
After activation the business components that were involved in the execution are added to
a spectrum. A business component may become involved in an execution due to some
functionality in the system. This functionality causes input messages to be exchanged be-
tween the different components. As a business component receives such an input message,
it becomes involved in the execution of that functionality in the system. A monitor also
determines whether an execution has failed or succeeded. The combination of this infor-
mation can be added to the activity matrix after which the involvement information for the
components is reset.

The problem with the business components is that they may send messages at any time
and thus messages may simultaneously arrive at components. This means that in a spectrum,
more components may be activated than should be part of a certain functionality of the sys-
tem. This also means that all the business components can become involved in a spectrum if
the spectrum is not determined soon enough. This causes a weak correlation between fault
and failure in the activity matrix. Therefore, a technique is required to determine the cut-off
of the spectra for SFL.

Piel et al. [38] conclude that the best result is returned by doing a pre-analysis of how
the system operates. The approach is also referred to as the transactional policy. The
pre-analysis consists of determining which components correspond to a certain interactive
functionality. This can be done with automatic analysis or by the programmer. When
a monitor can determine whether the execution has passed or failed, it will consider the
components for the monitored functionality and add these to the spectrum. Afterwards, the
involvement for the same components that were considered is reset.

Although the transactional policy is the most accurate, it suffers from having to pre-
analyze the possible interactions in the system. In dynamic systems, the interaction between
components change and for each change a pre-analysis needs to be performed. Furthermore, the way the components interact may change due to the faulty component. The difference between the pre-analyzed model and the actual interaction can lead to this policy omitting the faulty component [38].

2.3 Spectrum-based Multiple Fault Localization

Although there is proof that SFL can be used for online diagnosis, there is an issue that needs to be tackled first. The SFL considers the cases where single faults occur. In a dynamic distributed system multiple faults might be introduced either by changing several components simultaneously or by revealing additional faults on other components when changing a component.

Abreu et al. [2] also proposed an approach for the situation where multiple components might be faulty by combining the activity matrix with a model-based diagnosis approach using Bayesian reasoning. The algorithm that implements this method is called Barinel and is an implementation of the Spectrum-based Multiple Fault Localization (SMFL) approach. However, Barinel cannot be applied in the online case and therefore another implementation is required for the SMFL approach. This new incremental version for SMFL will be described in section 2.4.

Although it is easy to explain how the new incremental SMFL approach works, understanding how Barinel works proves why it cannot be used in the online case. Therefore, Barinel will be described first.

Barinel is split into two parts. The first part determines the Minimum-size Hitting Sets of the activity matrix. The second part attaches values to these sets to order the sets upon. The same division can be seen in the proposed incremental SMFL technique.

2.3.1 Minimum-size Hitting Set generation

The first step of Barinel is to determine the Minimum-size Hitting Sets (MHSs) from the activity matrix. A MHS is defined as the smallest possible set of components for which the following holds in terms of SFL: for each spectrum for which the error vector contain a 1, there is a component in the set, which coverage vector also contains a 1 for that spectrum.

These MHSs are also called candidates or diagnosis sets. The MHSs are calculated with the Staccato algorithm [1]. Staccato is a light-weight algorithm for determining the MHSs for a matrix. Although it is light-weight, it still cannot be applied for the online case, as the algorithm requires the compilation of the activity matrix, which might become very large. To illustrate how the analysis uses the activity matrix, the way Staccato operates will be described.

To explain how Staccato works, assume an activity matrix and error vector as can be seen in table 2.2. This algorithm sorts the components in the activity matrix by Ochiai coefficient, which results in a ranking of \( \{3, 2, 1\} \). Since \( a_{11}(3) = |\{i | e_i = 1\}| = 2 \), component 3 alone makes an MHS. This set is added to the solutions and component 3 is stripped from the matrix, resulting in the matrix which can be seen in table 2.3.
Spectrum-based Multiple Fault Localization

Table 2.2: Input for staccato

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>$e_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>$S_O$</td>
<td>0.5</td>
<td>0.7</td>
<td>0.8</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: After removing component 3

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>$e_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Next, component 2 is considered and stripped from the matrix. Also the rows where $x_2 = 1$ and $e_i = 1$ are removed, resulting in the matrix which can be seen in table 2.4.

<table>
<thead>
<tr>
<th></th>
<th>$e_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.4: Input for recursive step of staccato

This matrix is also the new input for the recursive call to Staccato. Since $a_{11}(1) = |\{i | e_i = 1\}| = 1$, component 1 will be stripped from the matrix and returned as the only MHS for this matrix. To each of the returned sets from the recursive call, which in this case is only $\{1\}$, component 2 is appended. If the resulting set is not subsumed by sets already in the solutions, it is added to the solutions. A set $A$ is said to subsume set $B$ if each element in set $A$ is also in set $B$. In this case, any of the solutions found ($\{\{3\}\}$) does not subsume the found solution ($\{2,1\}$) and it is added to the solutions, so that the solutions become $\{\{3\}, \{2,1\}\}$.

Then the first Staccato invocation would consider component 1, the last one in the ranking. The recursive step will return $\{2\}$ in this case. Since the resulting set $\{1,2\}$ is subsumed by set $\{2,1\}$, this set will not be added and the final result by the Staccato algorithm is (still) $\{\{3\}, \{2,1\}\}$.

As can be seen from the example, the whole activity matrix is analyzed in order to come up with a solution. Since every can change the Ochiai values for the components, the order in which they should be considered changes. Because of this change in order, the whole matrix has to be analyzed again to obtain the new MHSs. As more executions fail with different sets of components in the spectrum, the MHSs can become more complex. As the MHSs grow in size, there will be more recursive calls in Staccato, which increase the time required to analyze the whole matrix.

In order to solve the problem of having to analyze the whole matrix with the addition
of a new spectrum, it is required to find an incremental solution for finding the MHSs of an activity matrix in combination with the error vector. The incremental solution should be usable for online diagnosis.

However, finding MHSs is just the first part of the Barinel algorithm that cannot be used in the online case. The other part calculates the likelihood of each MHS that it only consists of all the faulty components. This likelihood is used to order the MHSs upon.

2.3.2 Diagnosis set ranking

Since the list of MHSs can be very long, having to find the fault in the components in the MHSs can become the same as finding the fault in every component. If the MHSs were ordered so that the MHS which is the most likely to contain just the faulty components is put up front, then this would mean that the fault would only be searched for (and found) in the faulty components.

Ordering the MHSs that were found by Staccato is the second part of the Barinel algorithm. This second part takes the MHSs that are the result of Staccato as input and performs a maximum likelihood estimation over the components of each set. Continuing from the example, assume $d_1 = \{3\}$ and $d_2 = \{2, 1\}$ as the diagnosis sets and the same input matrix as before.

The epsilon policy in formula 2.2 is used to calculate the probability that an execution fails given that the diagnosis set is correct. A diagnosis set is correct when the diagnosis set (MHS) contains just the faulty components. For a description on why the epsilon policy is used for the calculation and for a more detailed explanation of the calculation, please refer to the paper of Abreu et al. [1].

Because the executions are independent, the product can be taken over the probabilities, which results in the probability that an error occurs given that the diagnosis set is correct, or, in other words, the likelihood estimation. Taking the product over the result of using the epsilon policy for each spectrum when continuing from the example results in the following likelihood estimation formulas.

$$
Pr(e|d_1) = (1 - h3) \cdot (1 - h3) \cdot h3
$$

$$
Pr(e|d_2) = (1 - h1) \cdot (1 - h2) \cdot h1
$$

Maximizing over these likelihood formulas results in $Pr(e|d_1) \approx 0.15$ and $Pr(e|d_1) \approx 0.25$. Take $p = 0.1$ and $Pr(e) = \frac{2}{5}$, then $Pr(d_1|e) = 0.15 \cdot p / Pr(e)$ and $Pr(d_2|e) = 0.25 \cdot p^2 / Pr(e)$, which after normalization yield $Pr(d_1|e) \approx 0.857$ and $Pr(d_2|e) \approx 0.143$. The ranked diagnosis is therefore $\{\{3\},\{1,2\}\}$.

One could think that an incremental approach could be acquired by adding a new epsilon term to the end of the likelihood estimation formula each time a spectrum is added to the
activity matrix. However, once an MHS changes, a new likelihood estimation formula needs to be generated from the activity matrix, which, again, requires the compilation of the whole matrix.

2.4 An incremental implementation of the Spectrum-based Multiple Fault Localization technique

It has been shown that neither the first nor the second part of the Barinel algorithm can be used in online diagnosis. This is the reason why during the project of this thesis two new algorithms have been created. The first algorithm finds the MHSs in an incremental way by adapting the currently known MHSs into the new MHSs by only using the information from the spectrum and the associated error information. The new algorithm has been coined the Adaptive Incremental Minimum-size hitting set (AIM) algorithm.

The second algorithm orders the result of AIM, just as the second part of Barinel did with the results of Staccato. This algorithm uses a small adaptation of the original Ochiai formula to calculate similarity coefficients for the diagnosis sets. It has already been shown that the Ochiai coefficients can be calculated incrementally because the used A-values can be updated incrementally. The adaptation of the Ochiai formula also uses the A-values and can, therefore, also be calculated incrementally. Because this second algorithm is based on the same division as Barinel, but uses Ochiai coefficients instead, it is coined BArinel using oCHiai coefficients (BACH).

The AIM and BACH algorithms can be executed consecutively. Therefore, the successive execution of these two algorithms will be referred to as AIMBACH. In order to show why AIMBACH is an incremental solution for isolating multiple faults with the use of a Spectrum-based Multiple Fault Localization approach, the AIM and BACH algorithms will be described in detail.

2.4.1 Adaptive Incremental Minimum-size hitting set

The assumption behind the AIM algorithm is that when no errors occur, there will be no set (or actually all possible combination of the components in the system) that covers the errors in the error vector. Once an error occurs, each component in the spectrum is an MHS. When another error occurs, there are two possibilities for each set in the current solution:

1. The set contains one of the elements that was involved in the new spectrum
2. The set does not contain any of the elements that was involved in the new spectrum and therefore cannot cover the new error

In the first case, the set is kept as a solution. In the second case the set would cover the error if it was unified with each component that was involved in the new spectrum. However, if such a unified set is subsumed by another set in the solution, then it is no use adding this unified set to the solution. The algorithm that results from this intuition is shown in algorithm 1, where the definition of isNotSubsumedByAnyOf(e,X) is given by the following formula.
isNotSubsumedByAnyOf(e,X) : \{i \mid k = |X_i|, i = \{1, 2, \cdots, k\}, |X_i \cap e| = |X_i|\} = \emptyset

Algorithm 1 AIM algorithm
Inputs: current MHSs M, involved components in spectrum S, error e
Output: New MHSs that cover all previous occurred errors and e
1 \quad M' \leftarrow \emptyset
2 \quad \text{if } e = 0 \text{ do}
3 \quad \quad \text{return } M
4 \quad \text{else do}
5 \quad \quad \text{if } M = \emptyset \text{ do}
6 \quad \quad \quad \text{for all } s_k \in S \text{ do}
7 \quad \quad \quad \quad s' \leftarrow \{s_k\}
8 \quad \quad \quad M' \leftarrow M' \cup s'
9 \quad \quad \text{endfor}
10 \quad \quad \text{endif}
11 \quad \quad \text{while } M \neq \emptyset \text{ do}
12 \quad \quad \quad m_k \leftarrow \text{pop}(M)
13 \quad \quad \quad \text{if } m_k \cap S \neq \emptyset \text{ do}
14 \quad \quad \quad \quad M' \leftarrow M' \cup m_k
15 \quad \quad \quad \text{else do}
16 \quad \quad \quad \quad \text{for all } s_k \in S \text{ do}
17 \quad \quad \quad \quad \quad m' \leftarrow m_k \cup s_k
18 \quad \quad \quad \quad \quad \text{if } \text{isNotSubsumedByAnyOf}(m',M' \cup M) \text{ do}
19 \quad \quad \quad \quad \quad \quad M' \leftarrow M' \cup m'
20 \quad \quad \quad \quad \quad \text{endif}
21 \quad \quad \quad \quad \text{endfor}
22 \quad \quad \quad \text{endif}
23 \quad \quad \text{endwhile}
24 \quad \text{endif}
25 \quad \text{return } M'

When taking the same input as before (table 2.2), the first spectrum \( s = \{1, 3\} \) and the error \( e = 1 \). Since at this point M is empty, the algorithm will make a new set with 1 in it and a new set with 3 in it and adds the sets to the new MHS collection. The algorithm, therefore, will return \{\{1\},\{3\}\}.

In the next round, the spectrum \( s = \{2, 3\} \) and the error \( e = 1 \), but this time each component is considered in \( M = \{\{1\},\{3\}\} \). The first set \{1\} is removed from \( M \) and since \{1\} does not contain component 2 or 3, first 2 is added to it. This results in the set \{1,2\}, which is not yet subsumed by the sets in \( M = \{\{3\}\} \) or \( M' = \emptyset \). So set \{1,2\} is added to \( M' \). Then component 3 is added to \{1\}. However, since \{1,3\} is subsumed by set \{3\} in \( M \), \{1,3\} will not be added to \( M' \).
An incremental implementation of the Spectrum-based Multiple Fault Localization technique

Then set \{3\} in \(M\) is considered. The first (and only) element of this set is 3 and since 3 \(\in\) \(s\) \(\Rightarrow\) \{3\} \(\cap\) \(s\) \(\neq\) \(\emptyset\), set \{3\} is added to \(M'\). The returned MHSs \(M'\) is therefore \{\{1,2\},\{3\}\}. Finally, for the last spectrum the error \(e = 0\) and therefore \(M\) is returned, which is then still \{\{1,2\},\{3\}\}.

Clearly, this algorithm updates the currently known MHSs by the information about the new spectrum and the error for this spectrum. This means that it can be used in an online diagnosis approach. The results of AIM are the same as the results from the Staccato algorithm, which means that an algorithm to order the MHSs upon is still required.

2.4.2 BArinel using oCHiai coefficients

The algorithm that calculates the values upon which to order the MHSs is the BACH algorithm. The BACH algorithm calculates so called BACH coefficients by using an adaptation of the formula for calculating the Ochiai coefficient and the extended Bayes’ rule. The sets are then sorted in ascending order based on the BACH value of each diagnosis set. Again, it is important to see why BACH is an incremental solution and, therefore, the description of the BACH algorithm is given.

The Barinel algorithm is based on Bayes’ theorem, which can be seen in its basic form in formula 2.3.

\[
\Pr(d_k|obs_i) = \frac{\Pr(obs_i|d_k) \cdot \Pr(d_k)}{\Pr(obs_i)}
\]

Formula 2.3: Bayes’ rule

In formula 2.3, \(obs_i\) refers to the new spectrum and the error for this spectrum [2]. In contrast with Abreu. et al [2], the probability of \(obs_i\) is not assumed to be equal for each observation. Instead it is assumed that this probability cannot be known at a certain point in time during the execution. This assumption comes from the assumption that no information about the system is known beforehand. This means that it is likely that the activity matrix at one point may not contain all the components of the system.

Because of the assumption that \(obs_i\) cannot be known, formula 2.3 is rewritten to formula 2.4, which is the extended form of Bayes’ theorem and eliminates \(\Pr(obs_i)\).

\[
\Pr(d_k|obs_i) = \frac{\Pr(obs_i|d_k) \cdot \Pr(d_k)}{\sum_j \Pr(obs_i|d_j) \cdot \Pr(d_j)}
\]

Formula 2.4: Extended Bayes’ rule

Since the Extended Bayes’ rule is used for SMFL, the different parts of the theorem are interpreted as follows.

- \(\Pr(d_k|obs_i)\) - The probability of the diagnosis being correct when a spectrum with an error occurs.
- \(\Pr(obs_i|d_k)\) - The probability of the error to occur when the diagnosis is correct.
2. **DEVISING AN INCREMENTAL SPECTRUM-BASED MULTIPLE FAULT LOCALIZATION ALGORITHM**

- \( \Pr(d_k) \) - The probability that the diagnosis is correct.

The hardest part is to define how \( p_1 = \Pr(d_k) \) and \( p_2 = \Pr(\text{obs}_i|d_k) \) are calculated. The resulting formula for \( p_1 \) that is suggested in this thesis was created by the fact that the probability that the diagnosis is correct is the probability of every component in the set generating an error when involved. This probability is calculated by first calculating the probability that each component generates an error when it is involved, after which the product over these probabilities is taken. The resulting formula which calculates \( p_1 \) can be seen in formula 2.5.

\[
\Pr(d) = \prod_{n \in d} \frac{a_{11}(n)}{a_{11}(n) + a_{10}(n)}
\]

**Formula 2.5:** Calculating the probability of every component in a diagnosis set \( d \) being faulty and thus generating errors

In order to find a formula to calculate \( p_2 \) a simulation was developed and used in this thesis. For this simulation a number of components can be defined. For each component a separate probability that it is involved in an execution can be defined. Faulty components are defined as a subset of all the components in the simulation. Finally, the general probability that a faulty component generates an error can be defined. From these sets and probabilities an execution is generated. With this execution the A-values are calculated for each component. With the A-values, the values necessary for ordering the sets were calculated.

Several methods for calculating \( \Pr(d_k|\text{obs}_i) \) have been attempted. The methods have in common that they use A-values to calculate an Ochiai like value for a whole set instead of for just one component. The first method was to take the sum over the Ochiai values for each component. However, this resulted in higher values for bigger sets. Although it might be possible that the biggest set is the correct diagnosis set, it is certainly not always the case.

The next method was to take the product over the Ochiai values of each component. This resulted, however, in the opposite. Since probabilities are generally smaller than 1, sets with more components would generally be less likely to be the correct diagnosis set.

The final method that was used by BACH is to take the sum over the A-values rather than the Ochiai coefficients for each components. These accumulated A-values are then used in the already known Ochiai formula. The resulting formula which calculates the Ochiai coefficient for a set, which can be seen in formula 2.6, is used to calculate \( p_2 \).

\[
S_O(d) = \frac{\sum_{n \in d} a_{11}(n)}{\sqrt{(\sum_{n \in d} a_{11}(n) + \sum_{n \in d} a_{01}(n)) \cdot (\sum_{n \in d} a_{11}(n) + \sum_{n \in d} a_{10}(n))}}
\]

**Formula 2.6:** Calculating the Ochiai coefficient for a whole diagnosis set \( d \).

Now \( \Pr(d_k|\text{obs}_i) \) can be used to calculate **BACH coefficients** using formula 2.4 for each of the found diagnosis sets. Finally, any sorting algorithm can be used to sort the diagnosis sets on this BACH coefficient. Initially \( \Pr(d_k|\text{obs}_i) \) is set to 1 for each set.

Continuing from the running example, the collection of diagnosis sets after AIM becomes \( \{\{1,2\},\{3\}\} \). Calculating \( p_1 \) for each set results in \( \Pr(\{1,2\}) = \frac{1}{1+1} \cdot \frac{1}{1+0} = 0.5 \) and...
\[ \Pr(\{3\}) = \frac{2}{2} = 1. \]  And calculating \( p_2 \) for both sets results in \( SO(1,2) = \frac{2}{\sqrt{(2+2)(2+1)}} \approx 0.58 \) and \( SO(3) = \frac{2}{\sqrt{(2+1)^2}} = \frac{2}{\sqrt{3}} \approx 0.82. \)

When these values are used as the input for formula 2.4, the BACH coefficients become \( BACH(\{1,2\}) = 0.5 \cdot 0.58 + 1 \cdot 0.82 \approx 0.74 \) and \( BACH(\{3\}) = 1 \cdot 0.82 \approx 0.74. \) The final result of AIMBACH is, therefore, \( \{\{3\},\{1,2\}\} \), which is the same ordering as the Barinel algorithm concluded.

The BACH algorithm is now also incremental, for the same reason that SFL was incremental: it uses the A-values for ordering the diagnosis sets, which can be incrementally determined. This makes AIM and BACH usable in the online case. Moreover, since AIM and BACH can be performed in succession, the combination can also be used in the online case.

2.5 Summary

This chapter has been about finding an algorithm for fault isolation in a dynamic distributed system at runtime. Although Piel et al. [38] proved that Spectrum-based Fault Localization (SFL) can be used online, this approach can only isolate a single fault. Since an update of a dynamic distributed system can introduce multiple faults, it is required to find an fault isolation algorithm that can locate multiple faults. The Spectrum-based Fault Localization approach by Abreu et al. [2], however, could not be used because it does not profit from the same incremental properties of SFL.

Therefore, two new algorithms have been devised that follow the Barinel algorithm created by Abreu et al. [2], but perform the two steps of Barinel incrementally. The resulting algorithms are Adaptive Incremental Minimum-size hitting set (AIM) and BArinel using oCHiai values (BACH). These algorithms can be used consecutively to find and order the diagnosis sets based on the new spectrum and accompanying error value that are added to the activity matrix. The consecutive use of AIM and BACH is referred to as the AIMBACH algorithm.

The description for the AIM and BACH algorithms have been given and show that the algorithms can work in the online case.

The simulation that was defined to find \( p_2 \) was a simulation of a particular case. This case consists of a component which is always involved in the execution and several other components that are involved in the execution at random. There are three faulty components that contain faults. These three components are involved in the execution at random.

The goal of AIMBACH when devising a good formula for \( p_2 \), was to correctly identify these three components. AIMBACH did this successfully with the use of formula 2.6. Because a well defined simulation was used, it might be the case that AIMBACH has a lower accuracy in other specific setups of the simulation. This means that formula 2.6 needs to be used in other setups as well. This has been done during the controlled experiment, which is described in chapter 4.
Chapter 3

Obtaining the Required Information at Runtime for AIMBACH from Systems Implementing Service Oriented Architectures

Before AIMBACH can be used in a dynamic distributed system, information needs to be obtained from the system. To find out what information is required and how to obtain this information from the system at runtime to perform online fault isolation, a specific kind of distributed dynamic system will be investigated. This system is the result of using the Service Oriented Architecture pattern.

Businesses are eagerly adopting Service Oriented Architectures because systems that follow the Service Oriented Architecture pattern can be changed dynamically. This means that the systems do not have to be taken offline when a component requires an update, which is important for mission critical systems. This also means that businesses can more easily meet the needs of their users by adding or updating components to support more or better functionality.

As systems grow because of the increasing requirements and evolution of the system, it becomes harder to manually find the source of a fault when it occurs. Automatic diagnosis is therefore desirable, but the properties of Service Oriented Architectures can make this process more complex. To illustrate why it becomes more complex, this chapter will list those properties that may affect the diagnosis process and in particular, the fault isolation process.

This chapter also defines the required data, which is required to perform AIMBACH. This chapter concludes with a definition of a transaction, which is used by AIMBACH as a spectrum.

3.1 Service Oriented Architecture

A Service Oriented Architecture (SOA) is composed of two different components: the consumer and the service, which are named differently throughout the literature [36, 23]. The
3. Obtaining the Required Information at Runtime for AIMBACH from Systems Implementing Service Oriented Architectures

![Diagram showing consumer-service interaction in a Service Oriented Architecture]

Figure 3.1: Overview of the consumer-service interaction in a Service Oriented Architecture.

A service publishes its address to a registry, so that a consumer can perform a lookup on the registry and obtain the address to which to send its requests. The consumer can then send its requests to the service, which in turn might return a response. To the consumer the service is a black box. An overview of this relation can be seen in figure 3.1. A Service Oriented Environment commonly contains many consumer and service components.

According to McGovern et al. [31], there are several properties that should be supported by a SOA. In order to give a complete overview of the properties and because the properties are sometimes related to each other, all of the properties mentioned by McGovern et al. are listed below.

- **Discoverable and Dynamically Bound** Services can be located through a registry. This registry only returns the services that a consumer requires and is allowed to have access to. Once a service is located by a consumer the consumer formats its request in a format the service expects and binds it to the transport type that the service expects.

- **Self-Contained and Modular** Services are self-contained and modular. Services being self-contained means that they do not depend on the state and resources of other services. The level of modularity of services can be determined by applying five criteria. These five criteria are used to determine whether a component is sufficiently
modular, but can also be applied to services according to McGovern et al. [31]. The five criteria are the following.

- **Modular Decomposability** - Breaking the application into smaller modules. Service design should identify the smallest possible software unit that can be reused in a different context. The main focus is to make the modules reusable.

- **Modular Composability** - Services should be independent so that they can be used in different application domains from the domain they were designed for. This makes the services reusable as well.

- **Modular Understandability** - The functionality of a service should be understandable for a person without the person requiring information about the other services.

- **Modular Continuity** - Services should hide their implementation details from other services. If a service reveals too much details about its implementation, a change in the service could lead to required changes in other services.

- **Modular Protection** - Faults occurring in a service should not cascade to other services or consumers.

- **Interoperability** Each service provides a way to invoke operations through a connector type. An interoperable service has a connector which consists of a protocol and a data format which each potential consumer can understand.

- **Loose Coupling** Coupling refers to the amount of dependencies between services and consumers. Coupling is increased by consumers requiring a lot of information about the service they invoke. Furthermore, coupling is increased by the shared use of frameworks, database or by services having information about each other’s implementation. Loose coupling is accomplished by SOA by separating the interface from the implementation and the use of dynamic binding.

- **Network-Addressable Interface** It should be possible for a consumer to invoke a service through network. This allows services to be location-independent so that services are allowed to change their physical location without impairing the functionality of the SOA. Services may be invoked through a local interface if consumer and service are on the same machine, but the service should be able to simultaneously receive requests from across the network.

- **Coarse-Grained Interfaces** Whether a service is fine or coarse-grained depends on the amount of functionality a service provides and the amount of data an operation returns. The more functionality a service provides and the more data an operation returns, the more coarse-grained the service or operation is. In a SOA where services are more coarse grained, there are less network hops, which increases the performance. However, if services are more coarse-grained, they are less modular due to the modular decomposability criterion.

When an operation is more coarse-grained, it will return more data to the consumer. However, the consumer might not require all the information, which means useless
data has been sent, which decreases the performance. On the other hand if an operation is fine-grained, it will return less data and other operations might be required to obtain all the information a consumer requires. This leads to more network invocations of the service which also decreases performance.

A good balance for how coarse grained a service and its operations are needs to be found by the designer of the SOA. A balance between a fine and coarse-grained interface results in multi-grained services and methods.

- **Location Transparency** The combination of dynamic binding and network addressable interfaces renders the location of the implementation of the interface unimportant. Services can change from their physical location. Location transparency is also increased by SOA because the consumer has no direct dependency on the interface of a service.

- **Composability** Composability is closely related to the modular property of SOA. Services can be assembled into new applications. These applications can be very different from the application that the developer of the service had in mind. Services may be composed as an application composition, as a service federation or in a service orchestration. The application composition uses the services and bind them together for a specific purpose. Service federation assembles services to create a service with more functionality. Finally, service orchestration executes transactions that span over several services in an organization. These transactions are sometimes referred to as business process.

- **Self-Healing** Self-healing systems have the ability to recover from a failure during execution without human intervention. SOA are more self-healing because services are bound to and executed dynamically at runtime. If a service fails, the consumer can bind to another service provided that it supports the same functionality. This is because if a service implementation fails, another service can complete the transaction since the consumer interacts with the interface and not with the implementation of the service.

These properties can be an advantage to businesses if implemented correctly, but can increase the difficulty of fault diagnosis. Because services are dynamically bound, the composition of services that are used in the processing of functionality of the system may vary per request. This means that the most accurate monitoring approach by Piel et al. [38], where monitors are used to define the transaction, cannot work, because this approach requires the pre-analysis of a system.

Because services are self-contained and modular, information about a certain functionality should not be shared amongst services, which means that each service is a black box to each other. Furthermore, requests received by a service are handled independently. These two properties makes it hard to connect a component to a spectrum for a Spectrum-based Fault Analysis. This problem also surfaces in the system used by Piel et al. [38].
Mission critical system, which have to be updated while they are running, take advantage of dynamic binding and modular properties of SOA. To make such systems more self-healing, however, requires fault diagnosis to be performed.

3.2 Diagnosis in a Service Oriented Architecture

Diagnosis of a system is usually performed in three steps: fault detection, fault isolation and testing [21]. A failure occurs after a faulty component produces an error because of some interaction with this component. This error might be propagated through some other components until it reaches a points where the system can detect a deviation from its normal behavior with the use of failure detection techniques [5]. The behavior of a fault producing an error and resulting in a failure is also referred to as fault, error, failure propagation. Because the component at which the failure was detected is not necessarily the faulty component producing the error, fault isolation requires knowledge about the behavior of the system just before the failure. This behavior includes the transactions from one component to the other. The process of fault detection and fault isolation is also called fault diagnosis.

In fault diagnosis, there is a distinction between online and offline fault diagnosis. Online fault diagnosis is performed while the system is running while offline diagnosis can only be performed while the system is not running. In mission critical systems, which cannot be taken offline due to costs and possible risks involved, only online fault diagnosis is possible [34].

Because an SOA is dynamic, it is possible to take a component offline, without harming the functionality of the system. If the runtime information is collected somewhere, it is possible to find out the actions of the component that lead to the failure. However, it is possible that the error cascaded through several components. This means that the runtime information needs to be analyzed for each component that was involved in the execution of the functionality of the system. This means that the components may have to be taken offline one by one. The situation in which the fault isolation starts from the component where the failure occurred can be seen in figure 3.2.

The solution that is created in this thesis adds a bit of information to the requests that are exchanged. This information together with some details of the services, which can be locally determined, can be used as input for the newly created AIMBACH algorithm. The result is that instead of having to traverse every component and find the runtime information just before the failure occurred, just one or a few components need to be inspected. If the fault localization procedure is robust, i.e. can identify the faulty component correctly every time, the results can be used in self-healing procedures [37], which would mean that no manual inspection is needed.

The situation where the correct faulty component that lead to the failure is identified can be seen in figure 3.3. Here, the runtime information is collected to perform the Spectrum-based Fault Localization analysis on. One could argue that service 0 was already the most reasonable to contain the error, since it was the only updated component in the system. However, take for example a service that implements an operation which is faulty, but not
used in the system. Once a correct service is added that does invoke this operation, the fault becomes apparent, although the service that was added is correct.

3.3 Defining the required data for AIMBACH

To perform AIMBACH to isolate faults, information is required from the system. Most of this information will be directly available from the services in the Service Oriented Architecture.

As a starting point for defining the required data, a paper by Chen et al. [12] is used. In this paper an attempt is made to obtain execution paths of the system, which can then be used to form the runtime topology of services. An execution path is an ordering of service operations from the whole architecture that were involved during the execution of
Obtaining the process id

The different parts of information that is required to construct the runtime topology are the following.

- **Service id**: A unique identifier for each service based on its name, namespace and the url of its description file.

- **Interface id**: A unique identifier for each service operation, which is defined by the operation name and its argument types.

- **Process id**: A unique identifier for an execution of some functionality in the system, which is used to tie all requests together.

- **Sequence id**: A number which is used to order the operation invocations of the services that are called in an execution.

The service id and the interface id can be obtained from each service separately. The framework used by Chen et al. [12] already supports functionality for obtaining this information.

An exact method for determining the sequence id is not given, but the characteristics of this value should be that it is incremented every time a new operation is invoked during the process. However, AIMBACH does not require an ordering of the invocations of the operations on a service. So this piece of information is not required and will not be collected from the system.

Chen et al. [12] already gives the description of the process id and describes the problem of obtaining this information. In this thesis, the process id will be a unique identifier for the execution of a functionality, defined by using the service id of the first service that was involved in the execution. A counter can be maintained in a service to append a number to each service id to guarantee the uniqueness of the process id. This counter can also be maintained globally, which would make the use of the process id superfluous and obtaining this information simple. Instead it is assumed that this information is not maintained globally and therefore that it needs to be obtained from the service at runtime as well.

However, with just this information, the Spectrum-based Fault Localization approach can still not be used, because there is no indication of whether a failure has occurred for a certain process id. However, the theory of detecting failures is outside the scope of this thesis and therefore it is assumed that this information can be obtained. As will be shown in chapter 6, there are already approaches to find these failures as they occur.

### 3.4 Obtaining the process id

The method of obtaining the process id is a result of approaches suggested in literature. Since the literature that is related to this thesis is already discussed in chapter 6, there will be no in depth discussion on this literature performed here, but rather a conclusion is drawn. The conclusion from literature is that there is no specific method for obtaining information
3. **Obtaining the Required Information at Runtime for AIMBACH from Systems Implementing Service Oriented Architectures**

![Diagram]

**Figure 3.4:** The different types of and the source of the information that is required.

From a Service Oriented Architecture at runtime which is better than the other. Specific methods are suggested to perform specific tasks.

In the case of AIMBACH, only the process id cannot be easily obtained. In order to be able to keep the process id consistent and obtainable from each service, the following requirements need to apply.

1. The component which monitors the service that is involved first in the execution needs to determine the *process id*.

2. This *process id* needs to be transferred from the incoming request to the outgoing request of the service.

3. The *process id* can be obtained when an operation is invoked at the service.

The second rule can be a problem as services are a black box for each consumer and the request going out of the service is independent of the request that was sent to a service. This is due to the modularity property of services.

During the project of this thesis, a scheme has been devised to obtain all the necessary data. This scheme can been seen in figure 3.4. The scheme shows how the information is constructed and transferred between the components. The process id is first determined after which it can be collected.

In this thesis it was decided to retrieve the information from each component, rather than at specific points in the architecture. This decision was made because the services cannot share information. If the information would only be collected at specific points in the architecture, all the information that needed to be collected should be transferred between the services, which would increase the data overhead of each request.
Defining a transaction

3.5 Defining a transaction

In order to have the most accurate fault isolation method, we require a good definition for a transaction, as suggested by Piel et al. [38]. In SFL, a spectrum is defined by the components that were involved in an execution. However, in the online case for a SOA, it is hard to determine which operation on a service was invoked for a certain functionality, because several operations can be invoked at the same time for different functionalities. The definition of a transaction should be easily translatable in a spectrum for Spectrum-based Fault Localization, so that the transaction in SOA can be used as a spectrum in AIMBACH.

De Pauw et al. [15] define a *web service transaction* as being "a sequence of messages and invocations initiated by an action such as a consumer call." However, for SFL the messages are not important. Furthermore, for SFL it is not required to know the order in which the components were involved in the spectrum. Since the term action is also very broad in this definition, this has been narrowed down to the event of receiving a request from a consumer. So instead of this definition, the *transaction* defines a *set of invocations initiated by a request received from the consumer*.

As discussed before, the *process id* is used to tie the requests together. In fact, the process id is used to tie all the requests in a transaction together. What also has been discussed is that the component that monitors the first service that is involved in the execution determines the *process id*. The first service that is involved in the execution is the same service which receives the initial request from the consumer. The component that monitors this service determines the *process id*. The operation that this service executes is the start of the transaction.

Now that the start of a transaction has been defined, there also needs to be a definition for the end of the transaction, which is harder to determine due to the potential types of communication between services in SOAs. There are several possible types of communication between consumer and service, which are described by Shatz [43]. The most high level distinction can be made between synchronous and asynchronous communication [43, 30].

In synchronous communication (see figure 3.5(a)), the synchronous sender sends a message and waits until it receives an acknowledgment from the synchronous receiver or until the receiving invocation returns from its operation. The advantage is that when a message is lost or corrupted before it is received, there are no recovery measures necessary. The disadvantage is that there is less concurrency.

For asynchronous communication (see figure 3.5(b)), the sender does not wait for the receiver to receive the message and the receiver does not issue an acknowledgment. As the sender and receiver are not synchronized, a buffer is required for the messages that have been sent but not yet received. Because the sender does not have to wait, it can continue its calculations and therefore supports concurrency. However, recovery measures are necessary when messages are lost or damaged.

In general, a transaction has ended when all the operations involved in the transaction have finished their calculation for that transaction. In other words, a *termination detection algorithm* is required to find out when a transaction has finished.

There are *termination detection algorithms* known in literature for systems using synchronous communication [45] as well as asynchronous communication [14]. However, im-
3. Obtaining the Required Information at Runtime for AIMBACH from Systems Implementing Service Oriented Architectures

Figure 3.5: The two possible types of communication within a Service Oriented Architecture.

Implementing and proving these algorithms is outside the scope of this thesis. Therefore, the implementation in chapter 4 will describe a trivial method of detecting termination for asynchronous communication and the result of using such an approach will be discussed in chapter 5.

3.6 Summary

It has been shown that the use of SOA give business many advantages when implemented correctly. However, the same advantages also impose problems on fault diagnosis: because services are modular, easily changeable and self-contained, it is hard to define a spectrum at runtime, which is required for online Spectrum-based Fault Localization.

By obtaining the process id, interface id, service id and information about whether a failure occurred, transactions can be defined in SOA, which can be translated to spectra.

If the online Spectrum-based Fault Localization can identify the faulty component correctly every time, this approach can be used to make dynamic distributed systems self-healing.
Chapter 4

Description of the Controlled Experiment

Since by now the requirements for the implementation of a Spectrum-based Fault Localization method are known, it is possible to start with implementing the method so it can be used in a Service Oriented System. Research performed at Delft University of Technology led to the use of the Turmeric SOA framework\(^1\). The Scale.It.Up project uses this framework to implement the Apache Stonehenge project\(^2\).

Apache Stonehenge describes a number of services that operate with each other to make a stock market system where users can buy and sell shares. The version of Apache Stonehenge that has been implemented by Scale.It.Up is called Spicy Stonehenge.

In order to prove that the proposed method can work in a real system, a controlled experiment needs to be performed. Key issues are the time overhead, data overhead and accuracy of the implementation of the method.

Unfortunately, no real system could be found that could provide information that could proof the full workings of the proposed methods. Instead of using a real system as a case study, two controlled experiments were performed.

In the first experiment, the time and data overhead are investigated with the Spicy Stonehenge system as it is supplied\(^3\).

The second experiment, where the accuracy of the method is being investigated, also uses the Spicy Stonehenge system, in which several mutants of the services are deployed. The accuracy is also tested on test sets that were generated by the simulation that was created in order to tune the BACH algorithm.

---

\(^1\)For more information, visit https://www.ebayopensource.org/
\(^2\)http://incubator.apache.org/stonehenge/
\(^3\)Cloned at March 5th 2012 from GIT repository: https://github.com/SERG-Delft/spicy-stonehenge
4. Description of the Controlled Experiment

Figure 4.1: The Service Provider Framework (SPF) architecture.

4.1 Obtaining the service involvement information from a Service Oriented Architecture

In Turmeric SOA the consumer and the service can be defined and implemented. The service and consumer are entities with similar architectures. Since only the service will be used in order to extract the required information from the system, only the architecture of the service is discussed. The service in Turmeric is implemented in the Service Provider Framework (SPF), of which the architecture can be seen in figure 4.1.\(^4\)

A request that enters a service created with Turmeric SOA passes certain parts of the SPF architecture. After a request has been received by the service, the Server Message Processor sends the contents of the request through the in pipeline. This basically means that the handlers that are attached to this pipeline can execute operations on the contents of the request. After the content returns to the Server Message Processor, the content is sent to the service implementation (ServiceImpl in figure 4.1). After the service has performed its operation, it returns a response message, which is send through the out pipeline. This pipeline may also have handlers attached to it that execute operations on the contents of the response message. After this, the transport protocol is determined and the response is send back to the consumer.

Because the operation on the service might involve sending requests to other services, the required data need to be determined and obtained before the service performs its operations on the request. Therefore, a custom handler is added to the in pipeline. This custom handler is called the RequestLogHandler. The RequestLogHandler class has an initialize method that is called when the class is initialized. This is the point where the service id is defined as a combination of the uri, namespace and the name of the service.

\(^4\)Taken from the Turmeric SOA Documentation
Figure 4.2: Database scheme.

Once the RequestLogHandler is initialized the invoke method is called every time a request is received by the service. This invoke method can access the contents of the request and the transport headers of the request. The transport headers contain meta information about the contents of the request. The advantage of adding information as a transport header, is that it means that the contents of the requests are untouched, which makes the approach less intrusive. Also, the transport headers can be easily extended, because there are no strict rules about what should and should not be in the headers.

Therefore, the process id is transferred by adding it as a transport header. If the process id is not encountered amongst the transport headers, it means that the operation on the service receiving the request is the first operation in the transaction. This means that the process id needs to be defined here. Therefore, the value of a counter, which increases every time the service receives a request, is appended to the service id. This results in a unique identifier that can be used as the process id. Once the process id is determined, it is added as a transport header to the request.

The invoke method also determines the operation and the argument types that is invoked on the service, which are used to define the interface id.

After deciding the values of the service id, interface id and process id, the RequestLogHandler sends these values to the service_hit table in the database, which consists out of two simple tables, which can be seen in figure 4.2. The value of the failure column is initially set to 0 and can be changed when a failure occurs.

The finished_executions table has one column (id) which contains the process id of the transactions that have finished. The process id is added after one second after the start of the transaction. This is because no operation in the systems that is evaluated should take longer than a few milliseconds. This solution for determining the end of a transaction is simple, but as said before, the precise determination of the end of a transaction is out of the scope of this thesis.

Since the database is used by all components in the system, internal and external locking has been used to guarantee the correct insertion of data.
4. Description of the Controlled Experiment

Figure 4.3: The adaption of the existing architecture provided by Turmeric SOA to obtain the required data.

The components that are involved in obtaining the required data of the Service Oriented Architecture can be seen in figure 4.3. The green components in this figure show the modified components. All the components that are involved in obtaining the required data are added to the original architecture of the system and no modifications are made to any component that is in the original architecture.

The only thing left is to determine whether a failure occurred. Determining whether a failure occurred is done by adding a ResponseHandler to the out pipeline. Without going too much into the details, this ResponseHandler determines whether there will be errors returned in the response that the service will send. If so, the process id is taken from the headers of the request that caused the response and for every involved operation in the transaction the service\_hit\_failure column is set to 1.

4.2 Implementing the Spectrum-based Multiple Fault Localization Algorithm

The implementation of the online Spectrum-based Multiple Fault Localization, using AIM and BACH, consists of three of packages. Each package relates to a step of the Barinel and AIM and BACH algorithms and each contain several classes. The class diagram can be seen in figure 4.4.

The different components of the online SMFL algorithm act as follows:

- **dev.log.barinel.activity**
  - **Vector<K,V>** - A generic type class that maintains a header, and values as an ArrayList.
  - **ErrorVector** - Binds K to String and V to Integer and represents the error vector, the last vector of an activity matrix.
Combining the obtained information with online Spectrum-based Multiple Fault Localization

- **ComponentVector** - Also binds K to String and V to Integer, but represents the components and whether they are involved or not. This class also maintains the \( A\)-values for each component. The Ochiai coefficient can be calculated on the fly when it is needed.

- **Matrix** - Maintains an ErrorVector and multiple ComponentVector and an AIM object.

  - **dev.log.barinel.mhs**
    - **HittingSet** - The class that represents an MHS (or diagnosis set) with an ArrayList of components and a value that is used for ordering.
    - **AIM** - Maintains the list of hitting sets. Checks whether a MHS is subsumed by the currently known MHSs to get an accurate MHS list and is able to sort the MHSs currently known.

  - **dev.log.barinel.bach**
    - **Ochiai** - A static class that calculates the Ochiai coefficients for each hitting set.
    - **Bach** - A static class which uses the Ochiai class to calculate the BACH coefficient.

4.3 Combining the obtained information with online Spectrum-based Multiple Fault Localization

The last step is to use the gathered information to define the transactions, which are used as spectra in AIMBACH. The process id of all the finished transactions in the systems are added to the finished_executions table (see figure 4.2). These process id relate to the process id that is used to group all the operation invocations in the services. A finished transaction can, therefore, be defined by grouping all the entries of the service_hit table by the process_id that can be found in the finished_executions table.

Whether a transaction failed is determined by looking at the operations which are involved in the transaction. If one operation in the transaction has a true value in the failure column, then the whole transaction has failed.

The information about the involved operations and failure is the input for the AIM algorithm (see algorithm 1 with the involved operations as the spectrum \( S \) and the failure as the error \( e \)). For each of the sets that result from the AIM algorithm, the BACH algorithm calculates the BACH coefficient on which the sets are ordered.

4.4 Implementing a Service Oriented Architecture with AIMBACH

The first SOA that was created with the Turmeric SOA framework represents a calculator, of which the architecture can be seen in figure 4.5. The calculator was used to implement
4. DESCRIPTION OF THE CONTROLLED EXPERIMENT

Figure 4.4: The class diagram of the online Spectrum-based Multiple Fault Localization algorithm.

AIMBACH in the architecture of Turmeric SOA. The same implementation is used in Spicy Stonehenge, which will be described later in this chapter. The main operational purpose of the calculator is to solve simple math problems.

The Parse service receives all requests, which contain a string representing a calculation. By determining the used operator, the variables are sent to any of the other services. Take for example the string "3^4". The Parse service will take this string, extracts the 3 and 4 from it and sends it to the Power service. This service takes the arguments and sends 3 and 3 to the Multiply service, which returns 9. Now the Power service sends a 9 and a 3, to which the Multiply service returns a 27. This is repeated a final time, so that Multiply service returns a 81, which is returned by the Power service and then the Parse service. A sequence diagram can be seen in figure 4.6 to illustrate the calculation of "3^4" by the calculator.

The implementation of the calculator has several advantages. The first advantage is that it is known which services will be involved when executing a functionality of a service. This can be used to determine whether the correct information is obtained from the SOA.

Another advantage is the synchronous communication between the different services and the fact that only the parse service will be called from outside the system. This knowl-
edge has been used to define the end of the transaction. Without going too deep into the details, the event of returning a response by the Parse service can be marked as the end of the transaction.

The final advantage is that since the calculator should solve simple math problems, the expected result of such a computation can be compared to the response of the calculator. A deviation between the expected result and the response is regarded as a failure.

4.5 Simulation of a Service Oriented System

Based on how the calculator works, a simulator was developed for a Service Oriented Architecture. The goal of the simulator is to make it more easy to see how the accuracy of AIMBACH is influenced by different Service Oriented Architectures. Besides testing the accuracy of AIMBACH, this simulation has also been used to test the accuracy of other methods of fault isolation suggested by literature.

A system defined in the simulator consists of a number of operations that have a probability of being involved. A subset of these operations can be defined as being faulty. The simulation supports the definition of a general probability that a faulty component results in a failure.
Figure 4.6: The call flow for calculating $3^4$ in the calculator application.

The result of the simulation is a vector of involved components and the occurrence of a failure. The vector is written to the database (figure 4.2), which then can be used in the AIMBACH algorithms to generate and order the MHSs. Because the simulation can operate with AIMBACH, it was also possible to test different calculations of the BACH coefficients.

The advantage of this simulation is that it is very easy to simulate the execution of random transactions. However, operations that are executed on services can have a strong correlation. An example could be a login operation, which always invokes an operation that checks the credentials in another service. Therefore, the simulation cannot be used for evaluating how correlation between operations influence the performance of AIMBACH.

Other examples of situations that can be simulated are the following.

- Faulty operation hardly being invoked
- Faulty operations hardly resulting in failures
• Many or few operations involved in transactions

4.6 Implementing AIMBACH in an existing Service Oriented Architecture

Because the simulation does not involve sending and receiving requests or performing operations, the simulation cannot be used to perform measures based on the size of the requests and the time required to perform operations. Because it is not possible to perform such measures, another system is required, which performs the sending of requests and executes implementations of operations on a service.

The Spicy Stonehenge\textsuperscript{5} application attempts to implement the Apache Stonehenge project\textsuperscript{6}. The Apache Stonehenge project consist of several example applications that should provide a good basis of an implementation of the Service Oriented Architecture. However, the Apache Stonehenge project is no longer actively developed.

Spicy Stonehenge implements one of the SOA examples by Apache Stonehenge, which simulates a stock trader application where users can buy and sell stocks and track current values of stocks and currency. Spicy Stonehenge consists of three services that are able to communicate with each other, and a web-based application from which the service operations are called. Besides the small amount of components, there is hardly any communication between the different services.

In order to cope with the small amount of components, a load balancing approach is used that routes the requests from the PHP application to different instantiations of servers that have the three services deployed. The load balancer that is used is Inlab’s Balance\textsuperscript{7}. Inlab’s Balance captures input on a port it is listening to and forwards the input to a range of domain and port combinations in a round-robin fashion.

Increasing the communication between the components in Spicy Stonehenge requires the further development of Spicy Stonehenge.

4.7 Mutation testing

Spicy Stonehenge can also be used for evaluating the accuracy of AIMBACH. The advantage is that Spicy Stonehenge contains operations that are correlated. This correlation cannot be simulated with the simulator that has been developed. By evaluating the accuracy using Spicy Stonehenge it is, therefore, possible to see the effects of correlation between operations on the accuracy of AIMBACH.

In order to be able to isolate faulty operations in Spicy Stonehenge, they need to be introduced. This does not suggested that Spicy Stonehenge contains no faulty operations and that, therefore, it will not be possible to isolate faulty operations with AIMBACH. Instead, the faulty operations are introduced so that the input of the experiment is controlled.

\textsuperscript{5}For more information, visit https://www.ebayopensource.org/
\textsuperscript{6}For more information, visit http://incubator.apache.org/stonehenge/
\textsuperscript{7}http://balance.sourceforge.net
4. DESCRIPTION OF THE CONTROLLED EXPERIMENT

An approach in literature that is used for proving that diagnosis in the form of unit testing works, is mutation testing [22, 4]. Jefferson et al. [33] describes mutation testing as a helpful way to create test data by introducing many versions of the software, each containing one fault. When a mutant fails to execute, it is said to be killed and can be removed from the test set.

Mutation testing is used in the field of unit testing in order to test the unit tests that a programmer writes. If a test fails to find a mutation, it probably needs to be adapted to capture such a mutant.

Since Spicy Stonehenge was written in Java, a Java mutator needed to be found. A well known Java mutator is MuJava [28], but it is no longer being developed, and it does not support generic types which are used throughout the Spicy Stonehenge application. Alternatives like Jumble\(^8\), Jester\(^9\) and Simple Jester\(^10\) are also no longer being developed. A mutation engine that is still being developed is PIT\(^11\). Since it is still in development it means that it may not be perfect, but people are actively involved in the project.

MuJava was not chosen for mutation testing, because it would require to rewrite the Java code, so that it would not use generic types. Jumble, Jester and Simple Jester were not chosen either because it was hard to get the mutators to work or because it would be hard to automate the process of generating mutants and deploying them in Spicy Stonehenge. PIT, on the other hand, was easy to use, had proper documentation and the process was, in the end, easy to automate.

The problem with PIT, however, is that the software as it is supplied keeps the mutants in memory and runs the unit tests on the mutant in memory. In this way, unit testing can be done faster, but it means that the mutants cannot be accessed. However, unit testing cannot be used, because the services of Spicy Stonehenge need to be deployed and the mutants need to be deployed together with the services. The solution is to use the mutation engine class directly from the library file of PIT to write the mutants to disk\(^12\).

In this controlled experiment, mutation testing is used to verify whether AIMBACH can correctly isolate the faulty operation. Because AIMBACH is an approach which can isolate multiple faults, several mutants are introduced in the experiment.

4.8 Summary

The AIMBACH method has been implemented in the Turmeric SOA framework. At first by using the calculator SOA to verify the results of AIMBACH and using the simulator to improve the calculation of the BACH value calculated by AIMBACH. The simulator is also used to generate spectra of different sizes and setups of architectures. With these spectra it is possible to say more about how different setups of SOA can influence the accuracy of AIMBACH.

---

\(^8\)http://jumble.sourceforge.net/
\(^9\)http://jester.sourceforge.net/
\(^10\)http://ivan.truemesh.com/archives/000725.html
\(^11\)http://pitest.org/
\(^12\)For more information about this process, please contact the developer of PIT
AIMBACH has also been implemented into Spicy Stonehenge to be able to measure how much data and time overhead the current implementation of AIMBACH has. By adding mutated services to the architecture of Spicy Stonehenge, the architecture can be used to evaluate the accuracy of AIMBACH as well. The advantage of using Spicy Stonehenge is that it contains correlated operations, which cannot be simulated with the simulator that has been developed.
Chapter 5

Evaluation

Since now the Service Oriented Architectures and simulator that will be used in the controlled experiment have been implemented, the time and data overhead and accuracy can be evaluated for AIMBACH.

First the performance of both the offline and the online methods for isolating faults are compared to each other. They will be compared in terms of the time that is required to perform the analysis when a new transaction occurs in the system.

Next, the data and time overhead are determined by using the original Spicy Stonehenge application with and without the modifications required for the analysis and compare the time and data required in these systems.

The accuracy of the method will be tested by using Spicy Stonehenge and by using the simulation.

This chapter is concluded with a discussion about the results of the evaluation.

5.1 Barinel versus AIM and BACH to order the MHSs

In order to show why AIM and BACH were necessary, a few test sets were generated with the use of the simulation. In order to be able to add some time measurements to the Barinel algorithm, it was implemented in Java following the description by Abreu et al. [2].

In the simulation the parameters that can be seen in table 5.1 are set. First a simulation is performed with just five of the ten components. The second simulation uses all ten components. This is done to show how the number of components influences the time taken for AIM and Staccato.

In order to be able to properly compare both methods for the online case, each algorithm is performed each time an execution is added, which is how the algorithms in the online case would behave. The results can be seen in table 5.2.

Since the second part of Barinel also requires the compilation of the whole matrix, likewise behavior is expected. The result of the analysis can be seen in table 5.3. The values are the average of 3 runs with the test set.

These experiments show that Barinel will not perform well in systems as the amount of components increases. Spicy Stonehenge has about 33 operations and therefore 33 po-
Table 5.1: Settings for the simulation that will be used for comparing AIM with Staccato

<table>
<thead>
<tr>
<th>component</th>
<th>hit probability</th>
<th>faulty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.85</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0.25</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0.3</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0.6</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0.7</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0.7</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>failure probability</td>
<td>0.75</td>
</tr>
</tbody>
</table>

Table 5.2: The run times of Staccato and AIM for different amount of components after a number of runs.

<table>
<thead>
<tr>
<th>Time taken (ms)</th>
<th>algorithm</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>staccato</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>components</td>
<td>aim</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>8.980</td>
<td>0.7978</td>
<td>1.058</td>
</tr>
<tr>
<td>10</td>
<td>34.05</td>
<td>1.064</td>
<td>1.286</td>
</tr>
<tr>
<td>15</td>
<td>56.11</td>
<td>1.297</td>
<td>1.432</td>
</tr>
<tr>
<td>20</td>
<td>93.94</td>
<td>1.481</td>
<td>1.598</td>
</tr>
<tr>
<td>25</td>
<td>135.4</td>
<td>1.647</td>
<td>1.726</td>
</tr>
<tr>
<td>30</td>
<td>206.4</td>
<td>1.875</td>
<td>1.809</td>
</tr>
</tbody>
</table>

Table 5.3: The run times of the second part of Barinel and BACH for different amount of components after a number of runs.

<table>
<thead>
<tr>
<th>Time taken (ms)</th>
<th>algorithm</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Barinel</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>BACH</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>components</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>7.98</td>
<td>1.21</td>
<td>1.34</td>
</tr>
<tr>
<td>10</td>
<td>8.40</td>
<td>1.26</td>
<td>1.51</td>
</tr>
<tr>
<td>15</td>
<td>9.00</td>
<td>1.32</td>
<td>1.77</td>
</tr>
<tr>
<td>20</td>
<td>9.74</td>
<td>1.38</td>
<td>1.96</td>
</tr>
<tr>
<td>25</td>
<td>10.64</td>
<td>1.42</td>
<td>2.42</td>
</tr>
<tr>
<td>30</td>
<td>11.62</td>
<td>1.46</td>
<td>2.94</td>
</tr>
</tbody>
</table>
tential components in a spectrum. This illustrates that performing fault isolation over 30 components is not uncommon.

Also the algorithm takes incrementally more time, which makes sense because the whole matrix needs to be recompiled for each new transaction

5.2 Evaluating Time Overhead in the Spicy Stonehenge Application

Time and data overhead are big issues when proposing a new method, because if the overhead is too high, the method cannot be used in practice. Time overhead is the increase of time taken for an operation to execute due to some functionality of the system. In this case, the functionality is the fault diagnosis performed by AIMBACH.

Time overhead can cause bottlenecks at specific points in the system when the time it takes to execute an operation is higher than the time interval between the executions of operations. The bottleneck can be overcome if the processing of information for the method can be done concurrently, which is the case for the proposed method. However, AIMBACH is currently not implemented in a fully concurrent fashion, because a central database is used that may cause the gathering of information to take additional time due to synchronization mechanics.

The time overhead is calculated by the average extra time required for performing a transaction on the system. The time is measured from the perspective of the consumer as being the time between sending a request and receiving a response. These times can easily be determined by using SoapUI\(^1\), which is a tool in which (SOAP) requests can be defined and sent to a destination. Also the response of the request is made visible.

Measuring the time overhead of AIMBACH in Spicy Stonehenge requires operations to be invoked. There are several ways to invoke operations at the three services of Spicy Stonehenge. One possibility is to send SOAP messages directly to the services. Another possibility is to use the web-based application to create requests that are sent to the services.

Sending SOAP messages is easy to automate if the contents of the requests are known beforehand. The problem with Spicy Stonehenge, however, is that some operations are not fully supported and can make the measurements inaccurate, e.g. an operation could return without doing actual calculations.

In order to find out what the content of the requests should be and to find out which operations are supported, the web-based application is used. The web-based application uses just the operations that are supported and creates requests for these operations. These requests can be captured by Apache’s TCPMon and so, the contents of these requests can be used in the automated process. The following operations are supported by Spicy Stonehenge.

- **login** - Logging into the applications
- **buy** - Buy a certain amount of stocks

\(^1\)http://www.soapui.org/
• **getOrders** - Get the last few of orders performed with this accounts

• **getAccountData** - Application information about the account

• **getAccountProfileData** - Personal information of the account holder

• **getWalletData** - Get the different amounts of currency for the account

• **getAllQuotes** - Get all the stock quotes known in the system

• **getQuote** - Get a specific quote by an identifier

• **logout** - Logging out of the application

For sending the requests and determining the response time for each request, SoapUI is being used. The HTTP connection is kept alive between the requests, which means that time for building the connection is neglected. Therefore, the response time represents the time required for sending the request to the service, performing the operation on the service and returning a response.

First, the requests are sent in order to Spicy Stonehenge while AIMBACH is not implemented in the architecture. This is repeated ten times. Then, the requests are sent in order to Spicy Stonehenge which does have AIMBACH implemented, which is also repeated ten times.

The times (in milliseconds) for Spicy Stonehenge without using AIMBACH to return a response can be seen in table 5.4. For Spicy Stonehenge using AIMBACH, the response times can be seen in table 5.5.

It is easy to see that the implementation of the method has a negative impact on the response times of the requests. The average added overhead over each operation is 90% and the average of the total is 80%.

<table>
<thead>
<tr>
<th>components</th>
<th>runs</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 2 3 4 5 6 7 8 9 10</td>
<td></td>
</tr>
<tr>
<td>login</td>
<td>15 13 12 12 22 16 14 20 12 12</td>
<td>14.8</td>
</tr>
<tr>
<td>buy</td>
<td>36 26 29 26 38 28 26 30 25 36</td>
<td>30</td>
</tr>
<tr>
<td>getOrders</td>
<td>24 15 21 13 17 14 16 14 14 14</td>
<td>16.2</td>
</tr>
<tr>
<td>getAccountData</td>
<td>15 9 12 9 10 8 8 9 8 12</td>
<td>10</td>
</tr>
<tr>
<td>getAccountProfileData</td>
<td>12 8 12 8 12 7 7 11 8 8</td>
<td>9.3</td>
</tr>
<tr>
<td>getWalletData</td>
<td>10 8 11 8 11 7 8 8 7 8</td>
<td>8.6</td>
</tr>
<tr>
<td>getAllQuotes</td>
<td>9 10 10 8 11 8 8 8 8 9</td>
<td>8.9</td>
</tr>
<tr>
<td>getQuote</td>
<td>10 9 9 9 9 8 7 8 7 8</td>
<td>8.4</td>
</tr>
<tr>
<td>logout</td>
<td>8 8 10 9 9 7 6 7 7 8</td>
<td>7.9</td>
</tr>
<tr>
<td>total</td>
<td>139 106 126 102 139 103 101 115 96 115</td>
<td>114.2</td>
</tr>
</tbody>
</table>

Table 5.4: The response times for several requests in the system without the implementation of the method
5.3 Evaluating Data Overhead in the Spicy Stonehenge Application

Data overhead increases the amount of bandwidth that is required to be able to process all the requests that flow through the system. There might be points in the architecture where the amount of data that can flow from a component is restricted. This means data can get lost or cannot be transferred fast enough, which causes a bottleneck at this point in the architecture.

For determining the data overhead it is required to measure the size of the requests and responses that are sent in a Service Oriented Architecture. AIMBACH only adds data to requests sent by consumers in the form of additional headers. Therefore, the size of the requests with and without the required data needs to be measured.

The data overhead is expected to be low because there is just little information sent with the original request, but since in a Service Oriented Architecture the requests can be very small, it will still be worth while to investigate the overhead. The data overhead is determined by looking at the size of the requests that come into the three services.

In order to calculate the data overhead, the same operations are used as in the timing overhead analysis. Since the operations may invoke multiple services, there are more re-
quests than operations.

The size of a request is determined when it is received by the service. This means that the first messages received by the services have an overhead of 0, because there is no additional information added. In the Spicy Stonehenge system almost none of the operations require the invocation of other services in the system, so there are almost no requests sent that contain the additional information.

This can be seen in table 5.6 as there are many requests with 0 overhead. This means that the average overhead would be really small. So, instead of taking the average over the data overhead for every request, only the requests that were part of internal communication will be used in the calculation of the average. The requests that were part of internal communication can be easily identified, because they are the only requests that have an overhead value larger than 0 in table 5.6.

The conclusion is that the average overhead is 11 percent of the size of a request sent between two services in Spicy Stonehenge. But, of course, the test set is very small. For this implementation, the average size of the added header and header value combined is 95 byte. Compared to the other headers that are already added by the Turmeric SOA framework, 95 bytes is twice the average size of a header and its header value. And even though twice the size of the average header is not that large, the size can be further decreased by using a less verbose process id.

5.4 Determining the accuracy of the method with Spicy Stonehenge

Before the accuracy of AIMBACH can be calculated, a formula is required that can be used for Spectrum-base Multiple Fault Localization techniques. The formula suggested by Piel et al. [38], which in their paper has been used to calculate the accuracy for Spectrum-based Fault Localization techniques with different definitions of transactions, is also used in this thesis.
5.4.1 Specifying the accuracy

The accuracy of a fault isolation technique says something about how useful the technique is. In order to compare different methods to each other, the formula suggested by Piel et al. [38] has been used. The formula they propose for comparison between different fault diagnosis techniques is Relative Wasted Effort. This formula can be seen in formula 5.1. This formula calculates the percentage of the number of healthy components that need to be investigated before the faulty components are considered.

$$\text{Relative Wasted Effort} = \frac{C_d}{M - M_f}$$

Formula 5.1: Added relative effort for inspecting a set of potentially faulty components

In order to illustrate how this value is calculated, an example will be used. Take 1 and 2 as faulty components; 3, 4 and 5 as non faulty components and \{4,1\},\{4,3\},\{2,5\},\{1,2\} as the set of diagnosis sets. First component 4 will be considered. Since 4 is not a faulty component, inspecting this component is wasted effort and the total Wasted Effort (WE) is increased by 1. Next, component 1 is considered, which indeed is a faulty component, which means that this step does not add WE. Since component 4 is already considered, the next component in line is component 3, which adds 1 to the WE. Finally, component 2 is considered, no WE is added. There are 3 correct components and, therefore, the RWE is $\frac{2}{3}$.

Although RWE is not specifically mentioned to illustrate the purpose of determining the accuracy, the definition is basically the inverse of the the accuracy measurement specified by Abreu et al. [3]. The formula for accuracy given by Abreu et al. measures the percentage of blocks that need NOT be considered when searching for the fault by traversing by the ranking. However, their formula needs to be adapted for Spectrum-based Multiple Fault Localization and was, therefore, not used.

5.4.2 Calculating the accuracy

The accuracy is first investigated with the Spicy Stonehenge application. In order to increase the size of the system, there are five servers started with the same three services. However in several servers, mutations of classes used by the services are introduced, so that failures will occur.

Because services might invoke operation asynchronously, a simple approach is used for determining the end of the transaction. This simple approach adds the process id to the finished executions after a certain time after the start of the transaction. The value for this ‘timeout’ is set to a second.

With PIT, the TraderServiceManager class which is part of the BusinessService service, is mutated into several mutants and a mutant is chosen that will let the buy operation end in failure. This mutant is inserted on the first server.

For the evaluation, it is necessary to have the different operations be executed by services on different servers, so that the buy operation is involved in both a failed and a succeeded execution. Ideally, the execution of the operations would be randomly distributed
over the services on different servers, so that every operation on every server might get involved. However, Inlab’s Balance\textsuperscript{2} can only distribute the requests in a round robin fashion. This has to be kept in mind when determining how many operations should be in the test set.

Take, for example, four operations on two servers, of which the second operation on the first server is the faulty one. When executing the four operations, the faulty operation will be executed on the second server and succeeds. In the second test run, because the four operations can be evenly distributed over the two servers, the second operation will again execute on the second server and succeed. This means that the faulty operation will never be executed.

Furthermore, communication between services may not happen because of a faulty operation. Take, for example, three servers with three operations each. The first operation is faulty on the third server. If the operations execute correctly, no other operations are invoked. Except for the second operation, which will invoke two other operations. The faulty operation will not invoke any other operations. During the first test run, a total of five operations will be executed, which means that the test run ends with the third operation on the second server. The second test run, the faulty operation will be executed and therefore three operations will be executed. These three operations can be evenly distributed over the three servers. This means that the following rounds, the operations will be divided in the same way over the three servers.

To accommodate the problem, the Spicy Stonehenge test set is extended with two operations. This means that if the test set does not involve the execution of the faulty buy operation, 13 operations will be executed. On the other hand, if the test set does involve the execution of the faulty buy operation, 11 operations will be executed. These two numbers together add up to a total of 24 operations, which cannot be equally divided over the servers either.

This test set is run 5 times, while the analyzer takes the finished executions from the database and performs the AIM and BACH algorithms over them. The second run ends up in a failure. The results can be seen in table 5.7.

<table>
<thead>
<tr>
<th>BACH coefficient</th>
<th>Server</th>
<th>Service</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>BusinessService</td>
<td>buy</td>
</tr>
</tbody>
</table>

Table 5.7: The results of the analysis with one faulty operation

This result was expected, because when the buy operation fails, it is the first and only operation in the transaction, so only this operation will be involved in a failure. In order to better illustrate the possibilities of Spectrum-based Fault Localization, there should be more possible MHSs for the activity matrix. This is realized by mutating the ConfigurationService, which is only invoked by the buy operation at the end of the transaction, which involves multiple operation invocations. The TypeFactory class is, therefore, mutated and the mutant is added to the ConfigurationService on the first server.

\textsuperscript{2}http://balance.sourceforge.net
More possible diagnosis sets are generated by using this mutant, which can be seen in table 5.8. Here, also the value for BACH can be seen for each diagnosis set after 5 runs and after 10 runs of the test set. An operation in the diagnosis set in the table is presented as $<\text{server#}>::<\text{service}>::<\text{operation}>$.

<table>
<thead>
<tr>
<th>BACH</th>
<th>Diagnosis sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.261</td>
<td>1:ConfigurationService::getQSLocations</td>
</tr>
<tr>
<td>0.185</td>
<td>4:OrderProcessorService::submitOrder, 4:BusinessService::buy</td>
</tr>
<tr>
<td>0.185</td>
<td>4:OrderProcessorService::submitOrder, 5:OrderProcessorService::submitOrder</td>
</tr>
<tr>
<td>0.185</td>
<td>3:BusinessService::buy, 4:BusinessService::buy</td>
</tr>
<tr>
<td>0.185</td>
<td>3:BusinessService::buy, 5:OrderProcessorService::submitOrder</td>
</tr>
</tbody>
</table>

Table 5.8: The results of the analysis with one faulty operation

As can be seen in table 5.8, the BACH value for the faulty component is the highest, which means that AIMBACH identified the faulty operation correctly. Furthermore, as more runs are performed, the BACH value increases for the faulty component from 0.261 to 0.449. This means that as more transactions are added, the suspicion rises that the faulty component is indeed the faulty one.

The next step is to introduce multiple faults, which is done by combining the two mutants that were used before. The mutant of BusinessService is, therefore, added to server 3, while the mutant of ConfigurationService is added to server 1.

After five runs, the output of the analyzer is equal to what can be seen in table 5.7, except that this time the fault is detected on server 3, just as expected. The faulty getQSLocations operation is not executed in the first five runs and, therefore, there is no evidence that this operation might be faulty. Within ten runs, the getQSLocations operation is executed and produces an error. The result of AIMBACH after ten runs can be seen in table 5.9.

Of all the test cases, the accuracy measured in terms of RWE is 0. This is due to the fact that a faulty operation always produces an error and that the most transactions involve just one operation. The simulator is, therefore, used to simulate architectures where transactions involve more operations and where the probability of faulty operations producing an error is lowered.

### 5.5 Determining the accuracy of the method with the simulator

Before the simulator is used, a certain property of the simulator needs to be discussed. The simulator randomly produces operations involved in a transaction. This means that there is no correlation between operations, which makes the transactions for the simulator more complex, but leads to better accuracy. Take, for example, a transaction starting in operation 1, followed by 2, followed by 3, which will only occur if operation 1 is invoked. If any of these operations would be faulty, all three of them would be equally likely to be faulty after the analysis.
With that said, the first simulation simulates Spicy Stonehenge with the same operations as seen before. This means that the set of operations consists of the nine operations which have been used in determining the data and time overhead. To this test set, the two operations OrderProcessorService::submitOrder and ConfigurationService::getQSLocations are added. These are the two operations that are invoked by the buy operation. There are 3 servers in the simulation, which results in a total of 33 operations. A graphical overview of the operations in the simulation can be seen in figure 5.1.

Since the results for the situation with one and two faulty components which always produces errors, have already been seen, the probability that a service is hit is set to 0.1; three faulty operations are added: BusinessService::buy on each server; and the probability that any of these operations produce an error is set to 0.5.

An average of the RWE is calculated by running the simulator ten times with the same number of transactions. The results for varying amounts of transactions can be seen in table 5.10.

As the numbers of transactions increases, the value for the RWE seems to go down. This is because more evidence is added against or for a certain diagnosis. However, there also seems a big increase in the time taken for the analysis. This is because of the random behavior of the simulator, which randomly generates a lot of possible diagnoses. There is, however, some to be gained when the maximum number of faulty components is known.

However, it was said that no assumption can be made on the maximum number of possible faulty components. This assumption cannot be made because updated components might result in new errors produced by non updated faulty components, which were believed to be correct. Therefore, another approach is suggested to decrease the number of diagnosis sets that are considered. This approach is based on the intuition that once the BACH coefficient for a diagnosis set drops below a certain threshold, the diagnosis set is

<table>
<thead>
<tr>
<th>Bach 10 runs</th>
<th>Diagnosis sets</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.406</td>
<td>3:BusinessService::buy 1:ConfigurationService::getQSLocations</td>
</tr>
<tr>
<td>0.148</td>
<td>3:BusinessService::buy 1:BusinessService::buy 2:BusinessService::buy</td>
</tr>
<tr>
<td>0.148</td>
<td>3:BusinessService::buy 1:BusinessService::buy 3:OrderProcessorService::submitOrder</td>
</tr>
<tr>
<td>0.148</td>
<td>3:BusinessService::buy 2:OrderProcessorService::submitOrder 2:BusinessService::buy</td>
</tr>
<tr>
<td>0.148</td>
<td>3:BusinessService::buy 2:OrderProcessorService::submitOrder 3:OrderProcessorService::submitOrder</td>
</tr>
</tbody>
</table>

Table 5.9: The results of the analysis with two faulty operations
Figure 5.1: The 33 operations that are defined in the Spicy Stonehenge simulation.

<table>
<thead>
<tr>
<th>Number of transactions</th>
<th>25</th>
<th>50</th>
<th>75</th>
</tr>
</thead>
<tbody>
<tr>
<td>RWE</td>
<td>0.233</td>
<td>0.148</td>
<td>0.036</td>
</tr>
</tbody>
</table>

Table 5.10: Relative Wasted Effort for varying amounts of transactions with \( Pr(\text{faulty operation producing error}) = 0.5 \)

taken out of consideration. This approach is better than to just remove diagnosis sets of a certain size, because it requires no information about the number of faulty components.

The reason why it works can be explained by the way the BACH coefficients are calculated by the BACH algorithm. The BACH coefficient is calculated by using the \( A\)-values, which are incremented every time a transaction takes place. The larger these values get, the lower the impact of a new transaction on the BACH coefficients. So the intuition is that once the BACH coefficient for a certain set is below a threshold, the diagnosis set with this BACH coefficient will not be able to become the most likely true diagnosis set for the seen failures.

Although, in general, the method of using a threshold is better because the number of faulty operations is not known beforehand, in this case it is known that there are 3 faulty operations in the Service Oriented Architecture. Therefore, there is no reason to look for diagnosis sets that are larger than that. A simple test run shows that discarding diagnosis sets which contain more than 3 operations, takes the time of performing the analysis down to less than a second, which took 71 seconds before. So for the following simulations, the same technique was applied.

In a real system, failures are supposed to never occur and, therefore, the chance of...
5. Evaluation

A failure occurring is expected to be very low. This is why the probability that a faulty component results in a failure is lowered to 0.1. The expectation is that less evidence is generated for a component to be faulty and therefore it should be harder to pinpoint the true faulty component, increasing the RWE. The results of varying number of transactions for the simulation with the lower probability of generating an error can be seen in table 5.11.

<table>
<thead>
<tr>
<th>Number of transactions</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>RWE</td>
<td>0.4</td>
<td>0.33</td>
<td>0.27</td>
<td>0.20</td>
</tr>
</tbody>
</table>

Table 5.11: Relative Wasted Effort for varying amounts of transactions with Pr(faulty operation producing error)=0.1

The numbers in table 5.11 are a lot larger than in the simulation with the higher probability for failures. There seems to be a strong relation between the number of failures and the RWE. This was expected, because of the way the Ochiai coefficient is calculated for a diagnosis set.

When an operation is involved in a transaction which fails, the Ochiai coefficient will increase. If, on the other hand, an operation is involved in a transaction which succeeds, the Ochiai coefficient decreases. However, the Ochiai coefficient will increase with a larger amount than the amount with which it will decrease. Since the faulty operations in the simulation have a lower probability of producing an error, the Ochiai coefficients will decrease more frequent than they will increase. This means that the Ochiai coefficients will not deviate as quickly as in the simulation where the probability of a faulty operation producing an error is higher.

Therefore, the simulator is run until there is a certain number of failing transactions, in order to see the relation between the number of failures and the RWE. The results of this simulation can be seen in table 5.12.

<table>
<thead>
<tr>
<th>Number of failures</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
</tr>
</thead>
<tbody>
<tr>
<td>RWE</td>
<td>0.34</td>
<td>0.32</td>
<td>0.31</td>
<td>0.13</td>
</tr>
</tbody>
</table>

Table 5.12: Relative Wasted Effort for varying amounts of failures with Pr(faulty operation producing error)=0.1

Although there seems to be a relation, because the RWE decreases when the failures are increased, it still seems too random, as the RWE suddenly drops with 40 failures. Due to the fact that the number of transactions is not bound, the number of transactions might be any number larger or equal to the number of failures. This means that what should be looked at is the ratio between the number of failures and the total number of transactions. Therefore, the RWE is compared to \( \frac{\text{# failures}}{\text{# transactions}} \) as well. The results of the comparison can be seen in table 5.13.

Table 5.13 shows that the RWE is also random for \( \frac{\text{# failures}}{\text{# transactions}} \). Because the RWE is not depended on either the number of transactions or the number of failures, the next possibility is that the RWE is dependent on the probability of a faulty operation producing an error.
### 5.6 Comparing different Fault Localization methods

Since there is now a good comprehension of when the algorithm works best, we will compare the implementation with other methods for locating faults. For this, ten different test sets are created. These test sets are evaluated using the Ochiai value for single faults [3], the Tarantula value [25], the Minimum formula [46], Barinel [2] and AIMBACH (AIM and BACH applied sequentially), the method developed in this thesis. It needs to be mentioned that Xu et al. [46] go into more depth and use Minimum in combination with other methods to come up with the solution. Since there are no Key Block Chains in this controlled experiment, the M(inimum)KBC with(out) tie break cannot be applied.

During the comparison of the different algorithms, the threshold that was discussed before is used to decrease the number of diagnosis sets after AIMBACH performed the analysis. The threshold is set to a value of 0.01, meaning that whenever a diagnosis set

<table>
<thead>
<tr>
<th># failures</th>
<th># transactions</th>
<th># failures/transactions</th>
<th>RWE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>103</td>
<td>0.097</td>
<td>0.21</td>
</tr>
<tr>
<td>10</td>
<td>73</td>
<td>0.137</td>
<td>0.30</td>
</tr>
<tr>
<td>20</td>
<td>106</td>
<td>0.189</td>
<td>0.30</td>
</tr>
<tr>
<td>20</td>
<td>103</td>
<td>0.194</td>
<td>0.15</td>
</tr>
<tr>
<td>30</td>
<td>219</td>
<td>0.137</td>
<td>0.12</td>
</tr>
<tr>
<td>30</td>
<td>198</td>
<td>0.152</td>
<td>0.03</td>
</tr>
<tr>
<td>40</td>
<td>233</td>
<td>0.172</td>
<td>0</td>
</tr>
<tr>
<td>40</td>
<td>216</td>
<td>0.185</td>
<td>0.15</td>
</tr>
<tr>
<td>50</td>
<td>249</td>
<td>0.201</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>217</td>
<td>0.230</td>
<td>0.03</td>
</tr>
</tbody>
</table>

Table 5.13: Relative Wasted Effort for varying amount of failures and corresponding amount of runs

In order to confirm whether the RWE is dependent on the probability of a faulty operation producing an error, this probability is increased from 0.05 to 0.25 with steps of 0.05. The results can be seen in table 5.14.

<table>
<thead>
<tr>
<th>Pr</th>
<th>RWE</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>0.515</td>
</tr>
<tr>
<td>0.1</td>
<td>0.697</td>
</tr>
<tr>
<td>0.15</td>
<td>0.606</td>
</tr>
<tr>
<td>0.2</td>
<td>0.030</td>
</tr>
<tr>
<td>0.25</td>
<td>0</td>
</tr>
<tr>
<td>0.35</td>
<td>0</td>
</tr>
<tr>
<td>0.4</td>
<td>0</td>
</tr>
<tr>
<td>0.45</td>
<td>0</td>
</tr>
<tr>
<td>0.5</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.14: Relative Wasted Effort for varying probabilities of faulty components producing errors for 200 transactions

The results in table 5.14 suggest that the accuracy of AIMBACH is perfect when the probability of a faulty operation producing an error is equal to or higher than 0.25. In other words, when in a Service Oriented Architecture the faulty operations produce an error with probability equal or higher than 0.25, AIMBACH is a robust algorithm for isolating these faulty operations.
5. Evaluation

has less than 1 percent probability of being the correct diagnosis, it will not be considered as a possible correct diagnosis any more. Increasing the threshold can decrease the time required for the analysis even further, but it is likely that this will make the algorithm less accurate. Since the methods are compared to each other based on the accuracy, the accuracy of AIMBACH must not be affected.

The Wasted Effort for using AIMBACH with the threshold compared to the other methods suggested in literature can be seen in table 5.15. This time, Wasted Effort was used because it is more clear than the RWE and the test sets each contain 33 operations. $Pr(hit)$ is the probability that each operation is involved. $Pr(fail)$ is the probability that a faulty operation produces an error.

In the last three test sets, not all faulty operations produced an error in a failed transaction. This would mean that all the correct operations would have to be considered, resulting in a RWE of 1. In practice this would mean that some of the faulty components would get fixed and that the remaining faulty components will be isolated by another run of the fault diagnosis.

<table>
<thead>
<tr>
<th>test set #</th>
<th>Tarantula</th>
<th>Ochiai</th>
<th>Minimum</th>
<th>Barinel</th>
<th>AIMBACH</th>
<th>Parameters</th>
<th>Wasted Effort</th>
<th># runs</th>
<th># fails</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0.5</td>
<td>200</td>
<td>111</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.5</td>
<td>0.25</td>
<td>200</td>
<td>55</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.25</td>
<td>0.25</td>
<td>200</td>
<td>33</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
<td>0.25</td>
<td>200</td>
<td>46</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
<td>0.15</td>
<td>193</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>0.1</td>
<td>0.25</td>
<td>97</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.1</td>
<td>0.1</td>
<td>96</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>98</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>46</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>25</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 5.15: Wasted Effort for different methods of Fault Localization and the parameters for the simulator generating the test sets. The numbers below the algorithms represent the RWE after analyzing a test set.

What can be concluded from the results in table 5.15 is that it is worth to use a multiple fault isolation technique in the case where the presence of multiple faulty operations is possible. Since there is always a possibility that multiple faulty operations are introduced in a Service Oriented Architecture, it is good practice to implement a multiple fault isolation technique.

Also, AIMBACH seems to perform almost as good as the Barinel approach. If (Relative) Wasted Effort values are used for the case where multiple faults occur, then the way it
was calculated in this thesis requires that each faulty component also produces an error in at least one test case.

5.7 Discussion

The information that is obtained from the architecture enables the approach to find faults at service operation level. If more detailed information can be obtained, for example on method level, a more fine grained fault isolation approach can be applied, but this automatically increases the amount of components analyzed, which may increase the time necessary to come to a reliable diagnosis.

During the evaluation, the end of a transaction was determined with a timeout. The operations that were invoked due to a request send to the system within one second after receiving the request are part of the transaction. The operations that are invoked after that will not be part of the transaction, which means that important information might get lost. A better solution would be to use an oracle which can see the calculations and messages which are still active in the system due to the request. If there are no messages and no calculations active in the system, then the transaction has ended.

The adaptation of the original SMFL algorithm includes the use of a simulation to tweak the formula which calculates the BACH coefficients. Other formulas might be better in ordering the diagnosis sets when faulty components do not produce an error with a high enough probability. The other formulas might, therefore, perform better in real cases, because in Service Oriented Architectures, faults are not supposed to happen and everything will be done to prevent faults leading to a failure, especially in mission critical systems. Also, the occurrence of a failure depends on many factors. Granularity of the system, type of error produced by a fault and system load are just a few of these. In other words, the test sets generated by the simulator used in the analysis of the accuracy may not be representing the real scenario.

Since there was no accuracy measure for Spectrum-based Multiple Fault Localization methods found, the (Relative) Wasted Effort was used and calculated in a simple way. This method is affected by the order of the components in a diagnosis set and it is arguable whether this should be the case. Furthermore, there can be other elimination approaches for incorrect diagnosis sets. Take, for example, the same example used for the calculation of (R)WE. After component 4 was found to be correct, the diagnosis sets containing component 4 could be removed, because it would never be an element in a correct diagnosis set. This would mean that component 3 would not be considered and therefore the (R)WE would be lower. On the other hand, such elimination methods could increase the RWE if for example the diagnosis sets would be \{\{4,1\},\{4,2\},\{3,2\}\}.

Another issue with the (R)WE was that it fails to produce reasonable results in cases where not all faulty components were hit. That is why it would have been better to use a metric that did take in account this possibility, which could be found in literature.
5.8 Summary

AIMBACH has been evaluated with several indicators of performance: time overhead, data overhead and accuracy. The time and data overhead are indicators of whether the implementation can be integrated into an existing system. In the controlled experiment that was performed with Spicy Stonehenge, there was much time overhead.

This is due to the fact that the operations in Spicy Stonehenge do not require much time to execute compared to the time the AIMBACH implementation needs to collect the required information and write the information to the database. Another cause of the high amount of time overhead, may be due to the fact that the AIMBACH implementation acts as a request handler in the architecture of Turmeric SOA. It has not been verified whether adding a request handler to the Turmeric SOA adds time overhead.

The data overhead, on the other hand, was little. Even though the current format of the added information is very verbose, the AIMBACH implementation only adds 11 percent overhead to the requests. This number is measured by only looking at the requests that have the added information required by the AIMBACH algorithm. If all the requests and responses that make up the total data exchange in a Service Oriented Architecture would be taken in account, the value of the data overhead would be even less.

The last indicator is the accuracy, which is the most important measure for fault isolation methods. If a failure occurs, Spectrum-based Fault Localization (SFL) techniques will always produce a set of components. If the SMFL technique is accurate, it will place the diagnosis set with the faulty components at the start of the list. Therefore, an engineer will first inspect the faulty components, find the faults and no effort is wasted on inspecting any correct component.

AIMBACH has been proven to be a highly accurate SMFL technique for Service Oriented Architectures if measured by the Relative Wasted Effort (RWE). However, SMFL will have a maximum RWE if any faulty operations has not been involved in a transaction.

It has also been shown that the accuracy depends on the probability of a faulty component generating an error. A faulty component needs to generate an error that leads to a failure with a probability of 0.25 in order for AIMBACH to be accurate.
Chapter 6

Related Work

There has been much research on the topics that were discussed during this thesis. The most prominent topics of discussion being (1) the collection of data from Service Oriented Systems and using this data for runtime monitoring in order to get a better understanding of the system and (2) fault diagnosis. This chapter will not be an exhaustive research on everything that has been said about these topics in literature. However, the descriptions of the literature will be quite detailed in order to make clear where AIMBACH stands in relation to the other solutions.

6.1 Monitoring of systems

Ghezzi et al. [20] acknowledge the large amount of interpretations of the term monitoring. Therefore, a distinction is made between monitoring approaches by the type of properties that can be modified, the way these properties can be coupled, the methods of obtaining data, the degree of invasiveness and the time it takes to discover anomalies. AIMBACH monitors non-functional properties. This thesis was based on the idea of collaboration through choreography where every service knows what to do and for which it is important to monitor the non-functional properties. Ghezzi et al. propose four very prominent sources of data collection, but it is in no way complete: AIMBACH uses data collected from the headers of communication channels, which is not mentioned. AIMBACH is minimally invasive, both on the specification (there is no specification needed) as well as on the execution (business and monitoring logic can execute independently). Finally, AIMBACH can be seen as a proactive monitoring approach. This is because AIMBACH determines the A-values based on the A-values for previous transactions and the new transaction. With this information, AIMBACH produces a set of diagnosis sets. For each of these sets AIMBACH makes it progressively more likely or unlikely that the set contains all the faulty components.

With the distinction suggested by Ghezzi et al. [20], it is possible to explain the characteristics of several other approaches suggested by literature. One field of research of runtime monitoring is on compositions described using the Business Process Execution Language. This language can be used to create orchestration based collaborations between services. One approach that uses BPEL is proposed by Mahbub et al. [29]. The proposed framework
can monitor behavioral properties defined in BPEL as well as assumptions using event calculus. The specifications of monitoring are therefore intrusive, but the execution of the monitoring approach can be performed in parallel to the original system.

The approach suggested by Barbon et al. [6] is similar and also uses BPEL. However, the definition of monitoring properties is performed differently. Here, the Run-Time Monitor Specification Language is used to create events and check boolean and numerical formulas on class and instance level. The engine that was used for executing BPEL processes is the Active BPEL engine.

Common in both the approaches of Barbon et al. and Mahbub et al. is the fact that they monitor deviations of the normal execution. In this thesis such deviations are defined as failures, which means that these approaches perform the first step of fault diagnosis, which is fault detection.

An approach that is less intrusive on the specification is proposed by Sahai et al. [41]. The system uses a common component in SOAP toolkits, which are called routers. Proxies are added to these routers, which can capture incoming messages, record data about the exchange and then forward the message to the ultimate recipient. Since SOAP usually is encrypted by SOAP toolkits, port sniffing and server-side filters cannot be used. The goal is to check Service Level Agreements by generating monitors from these agreements and check them for fulfillment.

The Cayuga event monitoring system created by Demers et al. [17] only uses events, which are defined as relational tuples of data. This makes the collection of events like a database. But, rather than static tables, Cayuga monitors streams of events. The Cayuga Event Language is a query language, which has a SQL-like syntax, that can be performed over the events. The information that is obtained in this thesis can also be seen as relational data and is also saved as such in a database. This already raised the question of the performance bottleneck in reading and writing from the database, which may possibly be solved by using an approach like Cayuga.

An approach that has not had much attention in literature yet is the use of Aspect Oriented Programming languages to obtain the information. One such language is AspectJ, an Aspect Oriented Programming language for Java, and has been used by Espinha et al. [18]. They used this language to create a runtime topology, a trace of the requests as they traverse through the system.

Another interesting approach is the use of existing specifications and to extent their capabilities. One such approach is the Web Service Constraint Language (WS-CoL) as suggested by Baresi et al. [7, 8], which is an extension to the WS-Policy language where the programmer can add constraints on both functional as well as non-functional requirements. A preprocessor can parse the WS-Policy file, extract the policies and replace the necessary calls to services with calls to the monitoring manager. The monitoring manager verifies the policies and then invokes the service. This approach provides a uniform framework that can be applied by every engine that can interpret the WS-CoL language.

The last monitoring approach that will be mentioned is the Debugging Deployed Distributed Systems (D3S) developed by Liu et al. [27]. Their approach uses code injection to add predicates to global states in the system, which can be added to the running system. The result is a transient approach with little overhead.
Of course there are several other approaches that have been proposed [26, 40, 42], but these are more or less the same as the approaches discussed already.

### 6.2 Failure and fault detection

A taxonomy has been made by Delgado et al. [16] for a number of runtime software-fault monitoring tools. These tools have been compared on a number of characteristics, which can mainly be divided in specification language, monitor, event handler and operational issues. However, the definition of fault that is used by Gates et al. is actually the definition of an error used in this thesis and the systems that are analyzed here are not web service specific. However they may prove useful in detecting errors, which was a field of research left open in this thesis.

An interesting view on how to spot deviations from normal behavior has been suggested by De Pauw et al. [15] and is based on dynamic distributed systems. Paths are extracted from a set of traced executions and are grouped together. For these executions the execution times are also known. An exceptional long execution time can be an indication of a deviation from normal behavior. Also a pattern that only represents one or a few transactions and the possible inclusion of exception handling or incomplete messages may be a sign of a failure. The advantage here is achieved by the fact that exceptions are easier to spot by eye than thousands of transactions.

Another approach which spots deviations, is proposed by Fu et al. [19] and uses log analysis to extract log keys and teach a Finite State Automaton to distinct normal work flow from abnormal work flow. However, the systems that are the subject of this thesis are dynamic and for every change, the automaton would have to be retaught.

In the field of fault isolation there has also been a lot of research. The first approach that will be discussed is suggested by Chen et al. [13] and uses Probing Services to run specific test transactions. Events are abnormal conditions which may be detected by Probing Service and indicate that the services along the path of the test transaction may not be in good health. However, events may be transient, which means that it cannot be reproduced and no diagnosis should start. Such transient events are not ignored in Spectrum-based Fault Localization and it is debatable whether transient events (errors) should be diagnosed.

Steinder et al. [21] created a survey on the different types of fault isolation techniques and categorizes them by type. The Spectrum-based Multiple Fault Localization is a combination of a model-based approach and a Bayesian approach. The approach suggested in this thesis can also be qualified as such.

An approach which uses SMFL and whose goal is to isolate faults at runtime in a certain architecture, the same goal of this thesis, is created by Casanova et al. [11]. Their solution is divided in three steps. The first step is to acquire transaction types from the style of system under analysis. The next step is to use probes and event monitoring mechanisms to determine a finished transaction, the elements involved and whether the transaction succeeded. The results of the monitoring phase are then used in the SMFL algorithm. The determination of transactions is done by defining transaction families. These transaction families are defined by a parameterized pattern of behaviors expressed in terms of architectural ele-
6. RELATED WORK

The solution uses a time frame to define a set of transactions over which the SMFL algorithm is performed. The settings of this time frame is very important and only after 30 seconds (which is the largest time frame setting used) the correct diagnosis is given.

Cardoso et al. [10] indicate several issues on using SFL for fault diagnosis at runtime. The first is that errors are fuzzy, while SFL depends on a black and white classification for errors. The second is the precondition that the same test suite should always deliver the same results. In this thesis, a test suite could be seen as a set of requests fed to a certain service. It is not clear how this issue influences the results of SFL, however. The next issue is that the granularity of the inspected components of the system should be well chosen to keep the resource utilization in control. The last three issues have already been discussed during this thesis: determining a transaction, determining whether it fails or succeeds and how to obtain the data that is required for SFL.
Chapter 7

Conclusions and Future Work

This chapter sums up what was encountered during this thesis and is split into three sections. The first section describes the contributions that were made in this thesis. This section is followed by a section in which the research questions are answered by summing up the conclusions of this thesis. Because the subjects discussed in this thesis can still be researched further, some propositions will be made for future work, which can be read in the final section.

7.1 Contributions

During this thesis, several contributions have been made. The first contribution is the online Spectrum-based Multiple Fault Localization approach, coined AIMBACH. AIMBACH consists of two parts, the first of which is an algorithm that incrementally adapts the Minimum-size Hitting Sets (MHSs) for each transaction, which is called the Adaptive Incremental Minimum-size hitting set (AIM) algorithm. To decrease the wasted effort, the MHSs are fed to an algorithm which calculates the likelihood value of the set being the correct diagnosis and orders all the diagnosis sets on this value. This algorithm, coined BArinel using oCHiai (BACH), is the second part of AIMBACH.

Because there can be a large number of Minimum-size Hitting Sets, the calculation of the likelihood value can take a lot of time. This is why the BACH coefficient is used to prove that a diagnosis set is so unlikely to be the correct diagnosis set for the seen failure, that it can be removed from the analyzed sets. The reduction of the number of diagnosis sets is the second contribution of thesis. The value of this threshold during the case study was set to 0.01 and drastically decreases the time required to perform AIMBACH, while not influencing the accuracy negatively.

For AIMBACH to work, information is required from the Service Oriented Architecture. The third contribution is the definition of the required information, which consists of the service id, operation id, process id and failure information. The service id and operation id can be obtained from the service executing the operation. The process id needs to be determined at the start of a transaction and transferred to the other services which have
operations that are involved in the transaction. The failure information is a flag which is set once a failure is detected.

The last contribution is made by defining a transaction for the Spectrum-based Fault Localization approaches. The execution of operations in a system are the result of requests send to the system. This means it is possible to define a transaction by each operation invoked as a result of a request. Since communication in Service Oriented Architectures or any other dynamic distributed system can be asynchronous, it is hard to define the end of a transaction. That is why a simple solution was used: only operations invoked within one second after receiving the request are part of the transaction.

7.2 Conclusions

In order to draw the conclusions of this thesis the research questions that were stated in chapter 1 will be answered in order, after which additional conclusions are given.

Research Question 1
Can an incremental Spectrum-based Multiple Fault Localization algorithm be devised?

The Ochiai coefficient has proven to be a good value to order components upon in a single fault case at runtime. This coefficient can be calculated incrementally and was, therefore, used as a base for a new incremental Spectrum-based Multiple Fault Localization algorithm, coined AIMBACH. AIMBACH constructs the Minimum-size Hitting sets for the activity matrix incrementally and uses the A-values, which are determined incrementally, to calculate a value to order the Minimum-size Hitting sets upon.

Research Question 2
What information is required at runtime to be able to perform the adapted method?

The information required for SMFL depends on the granularity of the analysis. The implementation of AIMBACH in this thesis used the SMFL approach to find faults on service operation level. For this analysis, it is required to know the operation that was invoked, on which service it was invoked, whether the operation ended in a failure and which transaction it was part of. On which service which operation was invoked can only be determined at runtime, because of the dynamic properties of the system. The transaction is determined by looking at the process id of the request that was received by the service. If the process id cannot be found in the request, it needs to be determined by the service. Whether an operation ended in a failure needs to be determined by using failure detection techniques. Such techniques can spot deviations of normal behavior by monitoring certain properties of running systems.

Research Question 3
How is this information retrieved from a Service Oriented System at runtime?

There is no specific method of obtaining information from a Service Oriented System, basically because a Service Oriented Architecture is an architectural pattern rather than a
fixed architecture with certain points where data can be obtained. The implementation of AIMBACH in this thesis uses components in the pipeline of Turmeric SOA. A request enters this pipeline before it is delivered to the service and components added here can access the contents of the request.

**Research Question 4**
How can the beginning and end of an execution be determined in dynamic distributed systems at runtime?

In Service Oriented Architectures an execution is defined as a transaction. The transaction is defined as the set of invocations initiated by a request received from the consumer. The start of the transaction is, therefore, defined as the first operation that is a result of the received request. A transaction has ended when all the operations involved in the transaction have finished their calculation for that transaction.

Now that the research questions are answered, a few, more general, conclusions will be discussed.

This thesis started off as a research on how faults could be isolated in Service Oriented Architectures in order to make the debugging process more easy. Service Oriented Architectures were chosen, because businesses find them a good approach to ensure that components can be updated, added and removed without taking down the system. The approach that has been proposed in this thesis, however, does not limit itself to Service Oriented Systems, because no particular properties of Service Oriented Architectures are used. However, the terminology used for specifying the required data was aimed at Service Oriented Architectures, but it should be possible to make an easy translation to any other distributed system.

During the evaluation it was shown that the proposed method performs well in relation to other approaches, but the time overhead problem must be investigated before it can really be used in a practical situation. This could require a distributed way of storing data that is required for the analysis, or a more direct way of using the obtained data in the analysis.

Although the algorithm performed well, it is required to have oracles in place that can determine whether a transaction has finished and whether it has finished in a failing or successful state. Literature shows that there are approaches for doing this, but because of time restrictions, such approaches were not implemented in the case study performed in this thesis.

### 7.3 Future work

The results of this thesis are promising and probably a good step in finding an automated diagnosis approach. However, the approach could not be tested in an empirical study, because there was no good existing system to proof the results with. Simulation is a good and easy way to create large artificial systems but lacks some key features that may be present in distributed systems and can influence the accuracy of fault isolation techniques. Examples of such features are load balancers, and components that share a causal relation.
Literature shows several approaches to detect failures in distributed systems. Since each approach may conclude different failing states, it is interesting to find out how such failure detection approaches influence the accuracy of fault localization techniques in general, and the approach developed during this thesis specifically.

Another issue that requires attention are failures that are induced by computations outside the system. For Service Oriented System specifically it may be true that other systems attempt to interact with the system by using a web accessible interface. This means that there might be faulty components that are introducing failures in the system that cannot be automatically diagnosed. On the other hand it may show that the system does not work comply its specifications. The question of how such information can be used in self diagnosing systems should be researched.

In this thesis a not too thorough mention has been made about mutation testing. This is because the found approaches are only useful in unit tests. The self diagnosis methods, like failure detection and fault isolation are also developed to fix faults that were introduced by a programmer, which is much like unit tests are supposed to do. Seeing that there is no general approach for testing fault isolation approaches in general and the potential of mutation testing in unit tests, it is interesting to find out whether an approach similar to mutation testing can be developed for diagnosis methods in distributed systems.

It has been proven that the approach works, but in a running system there will always be transactions and there is no need to have the algorithm run for eternity, as only little will change after a certain amount of time. There might also be cases where the algorithm concludes that two sets are equally likely to contain faults and perhaps a second run of the algorithm can be a tiebreaker for such cases. More investigation needs to be performed about the conclusions that the approach reaches in a real-life system.

As seen in chapter 6 there are several other methods of isolating faults. In this thesis the SMFL approach was chosen, because the single fault case was already proven useful in practice. However, this thesis did not compared different multiple fault localization approaches at runtime. So, if there might be other methods that are more useful at runtime, they need to be compared to AIMBACH.
Bibliography


BIBLIOGRAPHY


66


67
[35] Peyman Oreizy, Nenad Medvidovic, and Richard N. Taylor. Architecture-based run-
time software evolution. In Proceedings of the 20th international conference on Soft-
Computer Society.

[36] Mike P. Papazoglou. Service-oriented computing: Concepts, characteristics and di-
rections. In Proceedings of the Fourth International Conference on Web Information
Systems Engineering, WISE ’03, pages 3–, Washington, DC, USA, 2003. IEEE Com-
puter Society.


[38] E. Piel, A. Gonzalez-Sanchez, H. Gross, and A.J.C. van Gemund. Spectrum-based
health monitoring for self-adaptive systems. In Self-Adaptive and Self-Organizing
2011.

[39] Thomas Reps, Thomas Ball, Manuvir Das, and James Larus. The use of program
profiling for software maintenance with applications to the year 2000 problem. In
Proceedings of the 6th European SOFTWARE ENGINEERING conference held jointly
with the 5th ACM SIGSOFT international symposium on Foundations of software en-
gineering, ESEC ’97/FSE-5, pages 432–449, New York, NY, USA, 1997. Springer-
Verlag New York, Inc.

11th IEEE International Conference on Requirements Engineering, RE ’03, pages 65–

[41] Akhil Sahai, Vijay Machiraju, Mehmet Sayal, Aad van Moorsel, and Fabio Casati. Au-
tomated sla monitoring for web services. In Metin Feridun, Peter Kropf, and Gilbert
Babin, editors, Management Technologies for E-Commerce and E-Business Applica-
tions, volume 2506 of Lecture Notes in Computer Science, pages 28–41. Springer


[43] S.M. Shatz. Communication mechanisms for programming distributed systems. Com-

[44] Andrew S. Tanenbaum and Maarten Van Steen. Distributed Systems: Principles and

[45] Chengzhong Xu and F.C.M. Lau. Efficient termination detection for loosely syn-
chronous applications in multicomputers. Parallel and Distributed Systems, IEEE
Transactions on, 7(5):537 –544, may 1996.